1

MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS
STANDARD REFERENCE MATERIAL 1010a
(ANSI and ISO TEST CHART No 2)

National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses Service

Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

**THREE DEGREES OF FREEDOM MODEL FOR PREDICTING
THE RIDE QUALITY OF OFF-ROAD VEHICLES**

by

MICHAEL CAMPBELL GALWAY, B.Eng., P.Eng.

**A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfillment of
the requirements for the degree of**

Master of Engineering

**Department of Mechanical and Aerospace Engineering
Carleton University
Ottawa, Ontario
1991**

© copyright

1991, Michael Campbell Galway

National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses Service Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ISBN 0-315-68882-3

Canada

**The undersigned recommend to the Faculty of Graduate Studies
and Research acceptance of the thesis**

**"THREE DEGREES OF FREEDOM MODEL FOR PREDICTING
THE RIDE QUALITY OF OFF-ROAD VEHICLES"**

**submitted by Michael Campbell Galway
in partial fulfilment of the requirements for
the degree of Master of Engineering.**

Thesis Supervisor

**Chair, Department of Mechanical
and Aerospace Engineering**

Carleton University

ABSTRACT

A computer model for predicting the ride quality of tracked or wheeled vehicles over deterministic terrain profiles is developed and discussed. This model, the Ride Prediction Model (RPM), computes the heave, pitch, and roll motion of the vehicle hull. The wheels are constrained to follow the terrain. The upper track is modelled using circular arcs (pseudo-catenaries), while the lower track has a non-uniform, variable tension. Trailing arm (with transverse torsion bars) or pillar suspensions may be modelled. Using the heave, pitch, and roll time histories of the hull, ride quality may be predicted using the absorbed power, ISO 2631, or Griffin's criteria.

The model is evaluated using experimental data, and compared with the VehDyn II ride prediction program used by NATO.

ACKNOWLEDGEMENTS

I wish to express my appreciation to Dr. Wong for the guidance he has given me as my thesis advisor over the past four and a half years, and for the diligence and enthusiasm with which he has reviewed my work.

I must also declare the debt of appreciation I owe to the many professional colleagues at the Department of National Defence with whom I have worked, at both PMO AALAV and LETE. Their advice, technical guidance, and moral support during the past three years was invaluable, and is manifested everywhere in this thesis. Major Al Howard bears special mention for the sustained interest with which he has followed my progress on this thesis, and for the strong encouragement (pressure?) with which he has urged completion of it. I also wish to thank Mr. Dwight Christensen for the full commitment with which he supported my efforts in the final phase of this work, and without which I would not have completed it.

Finally, I'd like to thank my family and friends, who feigned (credibly) an interest in ride dynamics over the course of this thesis. Their words of encouragement helped me overcome all obstacles which I met.

Table of Contents

	Page
Acceptance Sheet	ii
Abstract	iii
Acknowledgements	iv
Table of Contents	v
List of Tables	vii
List of Figures	viii
List of Symbols	x
 CHAPTER I: Overview of Vehicle Simulation	 1
1.1 Introduction	1
1.2 Computer Simulation of Vehicle Behaviour	2
1.3 Tracked Vehicles	8
1.4 Terrain-Vehicle Interaction	15
1.5 Ride Comfort Criteria	28
 CHAPTER II: Description of the Model	 36
2.1 Introduction	36
2.2 Solution of the Equations of Motion	39
2.3 Determining Roadwheel Location	41
2.4 Forces Acting on the Roadwheels	53
2.5 Solution of Wheel Forces	58
2.6 Tracks	64
2.7 Evaluation of Static Equilibrium Position (STATIC.EXE)	75
2.8 Ride Quality Assessment (RIDE-Q.EXE)	77

	Page
CHAPTER III: Experimental Evaluation of the Model	84
3.1 Introduction.....	84
3.2 The M113A1 Armoured Personnel Carrier	85
3.3 Test Terrain Profiles	87
3.4 Methodology to Obtain Experimental Data	90
3.5 Comparison of Experimental and Predicted Results	92
3.6 Discussion of Experimental and Predicted Results	122
CHAPTER IV: Conclusions and Recommendations	135
4.1 Conclusions.....	135
4.2 Recommendations.....	137
REFERENCES.....	139
APPENDIX 1: User's Manual for RPM.....	145
APPENDIX 2: Vehicle Data Files.....	160
APPENDIX 3: Terrain Data Files	167
APPENDIX 4: Source Code for TERPRED.EXE	174
APPENDIX 5: Source Code for STATIC.EXE.....	189
APPENDIX 6: Source Code for RPM.EXE.....	223
APPENDIX 7: Source Code for RIDE-Q.EXE.....	311
APPENDIX 8: Sample Ride Quality Reports.....	351

LIST OF TABLES

TABLE	DESCRIPTION	PAGE
I	Comparison of Features of Four Time-Domain Models.....	4
II	Measured and Predicted Ride Quality of APC Over Sine Wave	102
III	Measured and Predicted Ride Quality of APC over 150mm Bump (16 km/h).....	113
IV	Measured and Predicted Ride Quality of APC over 150mm Bump (24 km/h).....	114
V	Measured and Predicted Ride Quality of APC over 150mm Bump (32 km/h).....	115
VI	Measured and Predicted Ride Quality of APC over LETE Course 48.....	123
VII	Measured and Predicted (2-D RPM) Ride Quality of APC over LETE Course 48 (2-D RPM).....	124

LIST OF FIGURES

FIGURE	DESCRIPTION	PAGE
1	Degrees of freedom as allocated by RPM.....	7
2	Effect of sag on total tension at catenary support.....	12
3	Experimentally measured track profile - M113 APC.....	13
4	Effect of terrain wavelength on hubpath displacement	17
5	The track filtering effect.....	19
6	Schematic of pillar suspension	26
7	Schematic of trailing arm suspension.....	27
8	Block diagram for RPM	38
9	Hansen's method of determining roadwheel location.....	42
10	Multiple osculation points for a given suspension locus	44
11	Hubpath geometry with respect to terrain profile	45
12	Location of roadwheel by intersection of hubpath and suspension loci	47
13	Terrain bridging by wheel.....	49
14	Retrograde motion from application of hubpath equations.....	50
15	Hubpath generation by TERPRED.EXE.....	51
16	Free-body diagram for pillar suspension.....	60
17	Free-body diagram for trailing-arm suspension	61
18	Constituent segments of lower track.....	66
19	Constraint on track slope	69
20	Evaluation of intermediate segment length.....	70
21	Evaluation of track-roadwheel tangency point.....	71
22	Geometry of pseudo-catenary (circular arc).....	74
23	Error associated with pseudo-catenary approximation	76
24	M113 Armoured Personnel Carrier	88
25	Profile of LETE Course 48.....	91
26	Sine Wave - Front Hull - Low Damping.....	93
27	Sine Wave - Front Hull - Nominal Damping.....	94

28	Sine Wave - Front Hull - High Damping.....	95
29	Sine Wave - Front Hull - VehDyn II.....	96
30	Sine Wave - Rear Hull - Low Damping.....	97
31	Sine Wave - Rear Hull - Nominal Damping.....	98
32	Sine Wave - Rear Hull - High Damping.....	99
33	Sine Wave - Rear Hull - VehDyn II.....	100
34	Half-Round - 16 km/h - Front Hull	104
35	Half-Round - 16 km/h - Rear Hull	105
36	Half-Round - 24 km/h - Front Hull	106
37	Half-Round - 24 km/h - Rear Hull	107
38	Half-Round - 32 km/h - Front Hull	108
39	Half-Round - 32 km/h - Rear Hull	109
40	Half-Round - 24 km/h - Front Hull - VehDyn II.....	110
41	Half-Round - 24 km/h - Rear Hull - VehDyn II.....	111
42	LETE Course 48 - Front Hull - 3-D RPM	116
43	LETE Course 48 - Front Hull - 2-D RPM	117
44	LETE Course 48 - Front Hull - VehDyn II.....	118
45	LETE Course 48 - Rear Hull - 3-D RPM	119
46	LETE Course 48 - Rear Hull - 2-D RPM	120
47	LETE Course 48 - Rear Hull - VehDyn II.....	121
48	Experimentally measured track noise.....	128
49	Vehicle Speed - LETE Course 48 - Experimental.....	131
50	Time shift of field data with respect to simulation data.....	133
51	Contents of typical META.CNT file.....	148
52	Contents of typical run-time data file.	151

LIST OF SYMBOLS

SYMBOL	DEFINITION
a	Horizontal coordinate of roadwheel centre (trailing-arm suspension).
$a(t)$	Acceleration as function of time.
a_c	Semi-distance between catenary supports.
a_{RMS}	RMS acceleration estimated from VDV.
a_t	Tolerable acceleration for exposure of duration "t".
a_x	Longitudinal acceleration at driver's seat.
a_y	Heave acceleration at driver's seat.
a_z	Lateral acceleration at driver's seat.
a_1	Tolerable acceleration for one minute exposure.
$\hat{a}$	Frequency-weighted acceleration.
$\hat{a}_{\text{cq}}$	Axis-weighted acceleration.
$\hat{a}_{\text{rw}}$	Frequency-weighted acceleration in longitudinal axis.
$\hat{a}_{\text{yw}}$	Frequency-weighted acceleration in vertical axis.
$\hat{a}_{\text{zw}}$	Frequency-weighted acceleration in lateral axis.
$A(f)$	Absorbed-power-converted Fourier transform of acceleration.
A_f	Frontal area of vehicle.
b	Vertical coordinate of roadwheel centre (trailing-arm suspension).
c	Horizontal coordinate of torsion bar (trailing-arm suspension).
C_d	Aerodynamic drag coefficient.
d	Vertical coordinate of torsion bar (trailing-arm suspension).
D	Chord length of pseudo-catenary.
D_a	Density of air.
D_x	Horizontal component of damper force (trailing arm suspension).
D_y	Vertical component of damper force (trailing arm suspension).
f	Frequency.

$f(u)$	Vertical displacement of terrain profile.
f_c	Centre frequency of a third-octave band.
f_r	Specific rolling resistance.
F	Force input to body.
F_{ax}	Combined horizontal spring and damper force (pillar suspension).
F_{ay}	Combined vertical spring and damper force (pillar suspension).
F_p	Force perpendicular to pillar (pillar suspension).
F_R	Motion resistance due to dampers.
F_s	Spring force.
F_x	Longitudinal force.
F_y	Vertical force.
g	Function of time.
G	Function of frequency; discrete Fourier transform of $g(t)$.
H	Sag of catenary.
I_{xx}	Roll mass moment of inertia.
I_{zz}	Pitch mass moment of inertia.
k	Time parameter (discrete Fourier transform).
L	Chord length of pseudo-catenary half arc.
L_c	Horizontal distance between catenary supports.
L_s	Compressed spring length (pillar suspension).
L_x	Longitudinal moment arm.
L_y	Vertical moment arm.
L_z	Lateral moment arm.
m	Vehicle mass.
M	Suspension reaction moment at hull.
M_0	Net moment.
n	Number of roadwheels on ground.
N	Ground reaction force at roadwheel.
P	Average absorbed power.
P_D	Power dissipated in dampers.
q	Frequency parameter (discrete Fourier transform).

Q	Number of data points (discrete Fourier transform).
R	Roadwheel radius.
R_a	Aerodynamic drag.
R_i	Radius of i^{th} roadwheel.
s	Length of track in catenary.
S_a	Frequency-weighted acceleration spectral density.
t	Time.
T	Duration (time).
T_T	Track tension.
T₀	Reference duration (10 minutes).
T_{15VDV}	Predicted time to reach 15 VDV.
u	Longitudinal displacement of terrain profile.
v	Velocity input to body.
V	Vehicle speed.
VDV	Vibration Dose Value.
w	Weight per unit length of track.
W	Weight.
x	Longitudinal displacement.
x_{ci}	Horizontal coordinate of i^{th} roadwheel centre.
x_r	x coordinate of driver's seat in local coordinates.
x_i	Horizontal coordinate of i^{th} track/wheel tangency point.
y	Vehicle heave.
y_{ci}	Vertical coordinate of i^{th} roadwheel centre.
y_r	y coordinate of driver's seat in local coordinates.
y_i	Vertical coordinate of i^{th} track/wheel tangency point.
z_r	z coordinate of driver's seat in local coordinates.

α	Angle of pillar with vertical (pillar suspension).
$\alpha_{1,2}$	Angle of track with horizontal at pseudo-catenary support.
β	Angle of track with horizontal at roadwheel.
β_c	Inclination of pseudo-catenary chord with horizontal.
δ	Angle of track with horizontal at roadwheel.
δR	Difference in radius of adjacent roadwheels.
δx_c	Longitudinal distance between roadwheel centres.
δy_c	Vertical distance between roadwheel centres.
Θ	Vehicle pitch.
λ	Spatial wavelength.
τ	Shear force.
ϕ	Vehicle roll.

CHAPTER I

OVERVIEW OF VEHICLE SIMULATION

1.1 Introduction

The quantification of vehicle performance is of great interest to engineers involved in either the design or procurement of vehicles. Vehicle designers, for one, wish to know the effect of design changes on vehicle performance in order that they may optimize the design. Procurement engineers, on the other hand, wish to compare different competing vehicles in an objective manner. Vehicles are thus described quantitatively in terms of their acceleration, handling, ride, and so forth.

Traditionally, full-scale testing on actual vehicles was required to investigate vehicle performance. Such testing was expensive and time consuming. Today, the cost of designing a new or modified vehicle by test alone would be prohibitive. Instead, extensive use of computer models is made in the initial design or assessment stage, with reduced test programs used later for validation of the final design.

One area in which computer modelling has been particularly useful is suspension design, which affects vehicle ride and handling. Vehicles are rarely designed around their suspensions, and so the suspension must be tailored for the vehicle in order to meet the conflicting goals of good handling and good ride.

Suspension design is especially critical when considering the ride of off-road vehicles. The speed of off-road vehicles is often limited not by engine power or torque, but by the tolerance of human operators and passengers to vibration induced by travel over rough terrain [1]*. The ride behaviour of tracked vehicles is particularly well suited to analysis by computer, since the non-linear dynamics of the track are not amenable to manual analysis.

This thesis presents a computer model, the Ride Prediction Model (RPM), which can predict the ride behaviour of wheeled and tracked vehicles of various suspension types traversing terrain of known profile.

1.2 Computer Simulation of Vehicle Behaviour

Computer vehicle performance models may generally be categorized according to the method of analysis: analysis using finite element methods (FEM); analysis in the frequency domain; or analysis in the time domain.

*Numbers in square brackets are references.

Some computer models have used FEM techniques to predict ride performance [2]. These may be especially worthwhile when applied to the study of tracked vehicles, since tracks behave in a very non-linear manner that is difficult to accurately model by other methods. The individual track shoes that make up most types of tracks lend themselves readily to translation into finite elements. However, the very great number of elements required make it costly in terms of computer time and post-analysis.

Frequency domain analysis employs Laplace transform techniques under the assumption that linear relationships govern the equations of motion [3]. Although analysis in the frequency domain gives elegant solutions, the linear assumptions required are rarely realized in life. Wheel lift-off from the ground, suspension bump-stops, and damper blow-off are just three real-world phenomena which invalidate linear assumptions.

Among the most useful models have been time domain models. The ability to use non-linear relationships results in more realistic predictions. Simulation in the time domain was selected for RPM.

For comparison, three other time domain models may be quickly described. The chief features of each (as well as those of RPM) are summarized in Table I.

The NATO Reference Mobility Model (NRMM) is the standardized military vehicle assessment model of North Atlantic Treaty Organization (NATO)

Table 1: Comparison of features of four time domain models.

	NRMM VEHDYN	HANSEN'S MODEL	BENNETT &PENNY MODEL	RIDE PREDICTION MODEL
degrees of freedom	27 ¹	3	10	3
roll included	NO	YES	NO	YES
wheel filtering	YES	YES	YES	YES
rolling resistance	NO	NO	YES	YES
aerodynamic resistance	NO	NO	NO	YES
torsion bar suspension	NO	NO	YES	YES
rotary damper	NO	NO	YES	NO
variable lower track tension	NO	NO	YES ²	YES
integration routine	R-K	HPCM	R-K	R-K
ISO 2631 prediction	NO	NO	NO	YES
Griffin VDV prediction	NO	NO	NO	YES
absorbed power prediction	YES	NO	NO	YES

R-K = Runge-Kutta

HPCM = Hamming Predictor-Corrector-Modifier

Notes: 1 Maximum number of DOFs.

2 Constant tension drop between roadwheels.

nations [4]. It consists of several independent modules which analyse different aspects of vehicle performance. One of these, VehDyn II, evaluates vehicle ride performance [5]. This is a two-dimensional (heave and pitch, no roll) model, using a variable step, fourth-order Runge-Kutta integration scheme. Both wheeled and tracked vehicles may be modelled, and a number of suspension types are permitted, including bogey and pillar suspensions. Unfortunately, these suspensions seem to reflect an older generation of American military vehicles (the NRMM is a descendant of a series of models produced by the United States Department of the Army in the mid 1970's). Trailing arm, transverse torsion bar suspensions are singularly absent from the model, and yet they are overwhelmingly the suspension of choice in modern heavy fighting vehicles, both wheeled and tracked. The user must therefore try to model this type of suspension using the independent pillar suspension, which is not very satisfactory.

Hansen's ride model for tracked vehicles is a three-dimensional (3-D) model, including roll as well as heave and pitch [6]. It is also a three degrees-of-freedom (DOF) model (the degrees of freedom being heave, pitch, and roll) with the vehicle's roadwheels constrained to follow the terrain profile. By denying the roadwheels any degrees of freedom, the complexity and stiffness of the differential equations of motion are reduced. (Equation "stiffness" is related to the highest natural frequency present in a system. Since the roadwheels have a much smaller mass than the vehicle hull, they have much higher natural frequencies than the hull. The inclusion of the roadwheels in the equations of motion would therefore make the equations stiffer or more sensitive to changes in integration stepsize.) Although

Hansen's model permits use of only pillar-type independent suspensions, this is really no less adequate than the NRMM, as many of the NRMM's allowable suspension types are rarely encountered in existing vehicles anyway (the bogey-type suspension is a good example). The differential equations are solved by a Hamming Predictor-Modifier-Corrector algorithm. This is a very stable, very accurate, variable step method.

The Bennett-Penny model was developed at the Royal Military College of Science (RMCS) in Shrivenham, England [7]. It is a two-dimensional model like the NRMM, with up to ten degrees of freedom (since each roadwheel is permitted a degree of freedom). Solution of the equations of motion is obtained with a variable step Runge-Kutta routine. Unlike either the NRMM or Hansen's model, transverse torsion bar suspensions may be modelled. In keeping with British interest in rotary dampers [8, 9, 10], either rotary or telescopic dampers may be simulated (in contrast to the previous two models which allow only telescopic dampers).

The Ride Prediction Model is, like Hansen's model, a 3-D, 3 DOF model, with hull heave, pitch, and roll as degrees of freedom (Figure 1). A constant step, fourth order Runge-Kutta method was chosen for solution of the equations of motion, both because of its simplicity and reliability, and because the equations are not stiff enough to necessitate going to a more advanced method. (One researcher has recommended against using variable step solution methods for vehicle dynamics models [11].) Tracked, wheeled, and hybrid (for example, half-tracked) vehicles

Figure 1: Degrees of freedom as allocated by RPM.

may be modelled, with any combination of pillar or trailing arm (transverse torsion bar) suspensions. The model allows only telescopic dampers.

1.3 Tracked Vehicles

Tracked vehicles are a large and important class of ground vehicles. The use of tracks on a ground vehicle has several significant advantages over wheels alone:

- a. tracks reduce vehicle sinkage in soft terrain by spreading the vehicle weight over a large area;
- b. tracks distribute tractive effort more evenly over the supporting surface than wheels; and
- c. tracks bridge depressions in the ground, reducing wheel displacement (the track filtering effect).

Tracked vehicles may be classified according to their track type: large pitch, small pitch, or continuous belt. The pitch of the track is the individual length of the track shoes that are joined together to form the track. If the pitch is close to (or greater than) the distance between adjacent road wheels, the pitch is considered large. Large-pitch vehicles are used where ride quality is of less importance than vehicle sinkage and low-speed traction (the conditions under which heavy construction equipment is used, for example). Since the model described here is concerned with ride quality, large-pitch vehicles are not of interest. In contrast,

vehicles which are required to traverse soft, uneven ground at high speed must trade off some ground sinkage for better ride by using smaller pitch tracks. This class of vehicles includes commercial all-terrain vehicles as well as military tactical and support vehicles.

The ride quality for a vehicle with a track is very different from the ride of even the same vehicle with its track removed. In addition to the low frequency vibration induced by travel over rough, uneven terrain, which is experienced by all wheeled and tracked vehicles, a tracked vehicle will experience a high-frequency vibration generated as the individual shoes pass over the drive sprocket and idler, and as they strike the ground [12]. The ride quality of short-pitch vehicles (such as tracked military vehicles) is often better on uneven, soft ground than on a flat, hard surface, because the energy of the shoe as it impacts the ground is absorbed by the softer terrain.

1.3.1 Track Models

Tracks are by far the most difficult vehicle element to model successfully, having dynamic behaviour that is much more complex than the relatively simple motion of springs or dampers. The tracks of a vehicle act as a mechanism to transfer mass between the vehicle's sprung and unsprung masses, as a variable suspension stiffness, as a terrain filtering mechanism, as a major source of damping, and as an important contributor to high frequency vibration. Clearly the track must have a significant effect on the dynamics of the vehicle.

The track may be thought of as a continuous flexible band running around the vehicle's roadwheels and drive sprocket (and idler wheel, if present), and resting on any track rollers the vehicle may have. The portion of track hanging between the drive sprocket and idler wheel is part of the vehicle's sprung mass - any upward force on the vehicle hull must accelerate that part of the track upwards with the hull. The portion of track lying on the ground between the first and last roadwheels is part of the unsprung mass. As the vehicle bounces up and down on its suspension, the peripheral distance around the running gear changes, and the length of track hanging from the sprocket and idler must increase or decrease to accommodate this change in periphery (assuming inextensible track). As track is borrowed from this upper strand, the sprung mass decreases, and as the strand increases (due to suspension depression) the sprung mass increases.

In addition to this mass transfer mechanism, the track also acts as a non-linear suspension. As track is borrowed from the upper strand, the centre of gravity of the strand must rise, and the potential energy of the strand must increase. A force must be generated to effect this change in potential energy. The upward motion of the hull with respect to the roadwheels is thus resisted.

Any track model must exhibit these two characteristics: mass transfer, and variable stiffness. NRMM models the track by linear springs which act vertically between adjacent roadwheels. However, although this models variable suspension stiffness, it does not consider mass transfer. A better approach is the catenary model used by many other programs [5, 6, 13]. The track is modelled as if it is in

static equilibrium at every instant, with the upper track strand lying in a catenary (a hyperbolic curve characteristic of the deflection shape of a cable suspended in a gravity field between two support points). To conform to the strict mathematical definition of a catenary, the track must have no bending stiffness, and must have its mass evenly and continuously distributed along its length. This isn't strictly true for small pitch tracks, but may be a fair approximation.

The tension within the catenary is attributable to the weight of the cable. Under equilibrium conditions, the sums of vertical and horizontal forces must be zero. The vertical tension obviously must increase with vertical displacement along the catenary curve, from zero vertical tension at the curve minimum, to maximum tension at the highest support. The horizontal tension, however, must be constant at every point on the catenary. As the length of cable within the catenary is shortened, the angle of the catenary at its supports becomes shallower. In order to maintain sufficient vertical tension to support the weight of the cable, the horizontal tension must increase, and hence the total tension at the supports must increase (Figure 2).

The catenary allows track mass to be stored and removed, and thus meets the first required characteristic of a good track model. The tension also varies as the catenary sag is increased or reduced, and thus the variable stiffness criterion is also met. The catenary model is thus superior to the simple spring model of the track.

Figure 3 shows the static track deflection shape as physically measured on an M113 armoured personnel carrier (APC). The track is supported by the third of

Figure 2: Effect of sag on total tension at catenary support.

Figure 3: Experimentally measured track profile - M113 APC.

five roadwheels, approximately half way between the drive sprocket and idler, and also lies on the fourth and fifth roadwheels. Consequently, only the first half of the track is lying in what could be called a catenary. Note that the track slope initially decreases from the drive sprocket support. This is due both to the fact that the track is not pin-jointed at the top of the sprocket, but is wrapped around the sprocket, and to the fact that the rubber bushings between track shoes have considerable torsional stiffness. Because of the rubber bushings, the track exhibits both linear and torsional spring and damper characteristics [14]. This torsional stiffness is required to prevent the track from being thrown off the sprocket and idler. The bushings are also responsible for the high damping forces generated by the track which manifest themselves as rolling resistance [15]. The torsional stiffness of the bushings is at variance with the assumption of zero bending resistance required by the catenary equations.

Examination of high-speed video of track behaviour shows that the track does not hang in a quasi-catenary during travel over rough terrain [16]. Rather, waves are generated within the top track segment. These waves are reflected back and forth between the drive sprocket and idler. Under this wave action the drive sprocket oscillates as alternating high and low tension is applied to it. Since this behaviour is not easily modelled, the catenary model remains the most satisfactory track model.

The bottom span of track which lies on the ground bridges depressions in the surface and transmits tractive effort from the drive sprocket to the ground. Since

the application of torque at the drive sprocket produces a higher tension on the track being "reeled" in than on the track rolling off the sprocket, there must be a drop in track tension on the ground between the rear roadwheel and the front roadwheel. Shear forces between the track and ground are generated, and are distributed along the track-ground interface in some manner dependent upon the ground reaction stresses. For tracked vehicle ride simulation it is usual to either assume constant track tension [6] or to assume equal tension drops between each roadwheel [7]. RPM, however, calculates the tension drop individually for each roadwheel based on equilibrium considerations.

1.4 Terrain-Vehicle Interaction

Vibration in vehicles is chiefly attributable to the terrain surface, to the vehicle running gear (tracks or tires), and to the vehicle powertrain [37]. From a ride point of view, the terrain-vehicle interaction is the most important interaction. This interaction may be altered by the running gear through the phenomenon of wheel filtering or track filtering.

1.4.1 Vehicle Vibration

When a vehicle traverses uneven terrain, it experiences vibration over a wide range of frequencies. The low frequency (less than 30 Hz) vibration, which contributes most to perceived ride discomfort, is directly attributable to the terrain

profile. Higher frequencies are generated by shocks (due to hull grounding or suspension bottoming, for example) [12].

As the vehicle moves forward over the terrain, changes in terrain profile force the wheels to be displaced. The wheels are coupled to the vehicle hull through suspension assemblies composed of springs and dampers. The force transmitted to the hull is a function of the spring and damper characteristics, suspension geometry, roadwheel displacement function in time, and roadwheel inertia. In most cases, the roadwheel mass is small in comparison to the mass of the hull, and so the wheel inertial effects may be ignored (this is the basis used by Hansen for constraining the roadwheels to follow the terrain, thus reducing the number of degrees of freedom in his model).

1.4.2 Wheel Filtering

It has often been noted that the roadwheels of a vehicle act as non-linear low-pass filters between the ground and the vehicle [17, 3]. This is illustrated by Figure 4, which shows the hubpath of a wheel traversing four sinusoidal terrain profiles of different spatial frequencies, as follows:

- a. zero frequency (i.e. flat level terrain);
- b. low frequency (i.e. profile wavelength much greater than wheel radius);

Figure 4: Effect of terrain wavelength on hubpath displacement.

- c. intermediate frequency (i.e. profile wavelength of similar magnitude to wheel radius); and
- d. high frequency (i.e. profile wavelength much less than wheel radius).

As can readily be seen, the hubpath is substantially the same as the surface profile for zero or low spatial frequencies. The wheel passes these profiles virtually unchanged to the wheel hub, and thence to the vehicle. In contrast, at high spatial frequencies, the wheel rides on the crests of the waves, so that the hub is unperturbed by the profile. The wheel rejects or attenuates such high frequency profiles. At intermediate frequencies, the profile is modified in a complex manner before reaching the hub. Clearly, the wheel acts as a low-pass filter, rejecting high spatial frequencies, and passing low spatial frequencies.

Some simple models will assume that the surface profile, unmodified by wheels, is the forcing function exciting the vehicle. However, more comprehensive models must fully account for wheel filtering.

1.4.3 Track Filtering

In addition to the wheel filtering effect, tracked vehicles experience a track filtering effect [7]. One can imagine two roadwheels positioned on adjacent crests of a terrain profile, with an intermediate roadwheel sitting in the trough between them (Figure 5). On a wheeled vehicle, the intermediate wheel would sink into the trough until the suspension forces balanced the wheel's weight. This occurs virtually

Figure 5: The track filtering effect.

independent of adjacent wheels for independently suspended wheels. For a tracked vehicle, however, the track couples the roadwheels together, so that they cannot act truly independently of one another, even if they are independently suspended.

Because the track has a tension acting through its entire length, and because the track is wrapped around the bottom of the intermediate roadwheel, there must therefore be an upward force on the wheel equal to the vertical component of the vector sum of the two tensile forces. Depending on the magnitude of this force, and on wheel weight and suspension forces, this track force might lift the roadwheel off the ground. In effect, the track acts as a bridge across the depression in the ground. The downward displacement of the roadwheel is thus attenuated. Hansen's model and RPM both relax constraints on the roadwheel to permit wheel lift-off under these circumstances.

1.4.4 Vehicle Suspension Elements

The suspension of a vehicle has three components: a spring element or elements, a damper, and a structural element which connects the wheel to the vehicle, and to which the spring and damper are attached. These components must be combined in a manner which will give the vehicle good ride characteristics and good handling properties. These two goals are in conflict since good ride requires a soft suspension, while good handling requires a harsh suspension. There exists a continuum of ride and handling characteristics between these two extremes.

The suspension spring is a compliant mechanism designed to allow the vehicle wheels to ride over surface unevenness without perturbing the motion of the hull, and at the same time to keep the roadwheels on the ground so that traction and directional control can be maintained. Soft springs produce a smooth ride, but may result in the suspension bottoming out on rough terrain since they are poor at absorbing impacts (they have insufficient energy storage capacity). Since suspension bottoming is a primary cause of high shock forces on the hull, a soft suspension may actually worsen ride for off-road vehicles [1]. Control may also be lost if the wheels leave the ground.

The suspension bumpstop is a secondary spring that is sometimes present. The purpose of the bumpstop is to restrict the upward (jounce) or downward (rebound) motion of the wheel on its suspension, so as to prevent damage to both the suspension and the wheel. The bumpstop is typically a solid steel or rubber boss with which some part of the suspension comes into contact at the suspension limits. Owing to the material of the bumpstop, its stiffness is normally very much higher than the stiffness of the main suspension spring. Because the two stiffnesses act in parallel, the resulting stiffness is very near the stiffness of the bumpstop. Even a small compression of the bumpstop will generate a large spring force which is transmitted to the hull. These bumpstop impacts are undesirable from a ride quality point of view.

The ride behaviour of a vehicle is also dependent on the characteristics of its suspension's damping elements. If too much damping is present, high impact forces

will be transmitted to the hull. If too little damping is provided, the hull will oscillate in an uncomfortable manner. In selecting the correct damping parameters for a vehicle, the main functions of the suspension dampers must be taken into account:

- a. to damp vehicle pitching under acceleration (i.e. dive (forward pitch) and squat (rearward pitch)); and
- b. to reduce wheel vibration on rough terrain in order to ensure that the wheels stay firmly on the ground and that the vehicle remains controllable.

Research has shown that damping elements are a major source of noise and vibration in the vehicle [18]. Over smooth road, 90% of road vibration is transmitted to the chassis through the dampers (as opposed to through the springs or other suspension linkages), and on rough terrain 75% of the vibration is attributable to damping forces. Dampers can actually worsen ride on smooth roads and over discrete bumps.

Damper characteristics are usually not symmetrical in jounce and rebound. The jounce damping rate is generally much higher than the rebound rate. This ensures that the wheel is not easily thrown off the ground in jounce by bumps, but that it is quickly forced back down onto the ground in rebound. The different damping rates are achieved through internal orifice and valve design.

Whatever the manner of damping, energy dissipated at the damper must be recovered from some source. For ground vehicles, the ultimate energy source is the engine. It has been shown that the damping force actually appears as a motion resistance acting on the vehicle [19, 20]. The actual resisting force can be determined from the rate of energy dissipation in the dampers and from the forward speed of the vehicle:

$$F_R = P_D / V \dots\dots\dots (1)$$

The energy lost in this way appears as heat in the dampers.

Dampers can be separated into three groups: linearly viscous, exponentially viscous, and Coulomb. Linearly viscous dampers generate a force proportional to the relative velocity of the damper ends, while exponentially viscous dampers (or shock absorbers) produce a force proportional to the square of the velocity. Coulomb dampers produce a damping force independent of velocity (eg wet friction discs). Viscous dampers have a blow-off velocity or pressure, at which bypass valves are opened to the damping fluid. This prevents both internal damage to the damper, and the generation of high damping forces at high vehicle speed.

Continuous use of dampers over rough terrain will lead to overheating, which reduces the energy absorption capacity of the damper, and reduces its life [21]. It has been shown that hydraulic shock absorbers on an off-road vehicle can lose all efficacy after as little as 30 minutes continuous operation over rough terrain [22], due to reduction in the damping fluid's viscosity.

On tracked vehicles, it is usual to place dampers or shock absorbers on the first and last roadwheels only, where wheel travel is greatest, since damping the intermediate wheels has less effect on ride quality. There is of course a nominal amount of structural and Coulomb damping present at every wheel station. The Bennett-Penny model includes damping friction at all wheel stations.

1.4.5 Suspension Configurations

Modern off-road vehicles overwhelmingly use independent suspensions for each wheel station. Independent suspensions give better ride since each wheel may respond separately to discrete obstacles such as rocks or depressions. In addition, independent suspensions allow greater wheel travel (i.e. greater displacement to the suspension bumpstop). Wheel travel has been identified as one of the major contributors to ride quality, since bumpstop impact creates very high accelerations on the hull. Increased wheel travel also reduces the amount of vehicle pitching required to surmount obstacles [23]. This has been reflected in military vehicle design, where wheel travel has been increasing in recent years (the Leopard II tank, for example, has 175 mm more wheel travel than its predecessor, the Leopard I [1]).

Pillar suspensions and trailing arm suspensions are often seen on off-road vehicles. Both allow good wheel travel. Castor, camber, and lateral wheel spacing are maintained constant for pillar-type suspensions throughout the limits of travel, and so this suspension type is used mostly for steered wheel assemblies (steering can be adversely affected by variable castor and camber). Trailing arm suspensions do

not maintain constant castor and camber while deflecting, and hence are inappropriate for steered assemblies [24].

The pillar suspension consists of a helical spring connected at one end to the vehicle hull, and at the other to the roadwheel (Figure 6). If damping is required, a damper may be placed inside the coils of the spring. Kinematically, as the wheel is displaced by the ground, the wheel centre describes a straight line defined by vehicle inclination and the castor angle of the suspension. It is usual for the suspension to be tilted back (i.e. for there to be a positive castor angle) in order that a castor trail be created (the castor angle is typically 2° - 5° for automobiles [25], and up to 27° for motorcycle front forks [17]). The trail ensures that the wheel centres itself if it deviates from rolling straight ahead. Very large wheel travels are difficult to attain with the pillar suspension because of the problem of achieving suitable bending stiffness.

The trailing-arm suspension consists of a swinging arm (or roadarm) which is pivoted so that it swings in the pitch plane of the vehicle (Figure 7). Because the wheel is located behind the pivot point it is called trailing-arm. A spring may be fitted in several ways, most commonly either as a strut between the arm and the chassis (such as in the rear suspension of a motorcycle), or as a transverse torsion bar which doubles as the arm's pivot point (such as in the rear suspensions of the Canadian Forces' Armoured Vehicle General Purpose (AVGP)). Trailing-arm suspensions with transverse torsion bars are widely used in off-road vehicles because of their simplicity and durability, and because high wheel travels are possible. This

Figure 6: Schematic of pillar suspension.

Figure 7: Schematic of trailing arm suspension.

design also allows the vehicle's attitude to be trimmed by rotating the torsion bars in their mounts.

1.5 Ride Comfort Criteria

Much effort has been expended during the past fifty years into investigating the response of human beings to whole-body vibration (vibration to which the entire body is subjected as opposed to local vibration). This research has shown that the body is most sensitive to vibration in the frequency range of 2 to 8 Hz [26], although upper limits of 80, 50, and 30 Hz have been suggested for the assessment of ride quality [27, 28, 29]. The way in which the body responds to vibration is very complex and not fully understood, as ride comfort perception is dependent not only on physiological, and environmental factors, but also upon psychological factors such as motivation and expectation. For example, vibration that would be unacceptable in an automobile or passenger railcar might be well within expected norms for off-road vehicles, and may actually be desired for some recreational applications [26].

Ride criteria attempt to assess the severity of ride vibration in a repeatable and scientifically measurable manner, with a view to providing information on likely impact on human comfort and ability to carry out tasks efficiently. The three ride criteria used by RPM are ISO 2631, absorbed power, and Griffin's VDV. Since these criteria are based on human response to vibration as a function of frequency, the time domain accelerations must be transformed to the frequency domain for ride analysis.

1.5.1 Absorbed Power

Absorbed power was developed by the United States Army's Tank Automotive Command (TACOM) as a physically meaningful description of ride quality [26]. When a human body is subjected to vibration, the various tissues and organs which constitute the body are deformed. Some of the vibration energy is lost through this deformation because the tissues exhibit hysteresis. The rate at which this energy is lost is called the absorbed power. For a solid, inelastic object, the absorbed power will be zero since no physical distortion of the object will occur.

In subjective tests, it has been found that the measured absorbed power of random vibration correlates well with perceived ride quality, whereas acceleration levels do not match perception. This suggests that absorbed power is the physical mechanism responsible for the physiological aspect of human vibration response.

The calculation of absorbed power assumes that a transfer function can be found that relates the vibration input to the absorbed power. This in turn assumes that the human body acts as a linear system. Over the frequency range from one hertz to 30 Hz, the range of principal interest in ride assessment, this assumption is valid [30]. If, however, the vibration is so severe as to cause the individual to lose contact with the vibrating surface, the assumption is invalid and the body behaviour becomes non-linear.

Average absorbed power is calculated from the input force and velocity at the body [29]:

$$P = \lim_{T \rightarrow \infty} \frac{1}{T} \int_0^T \mathbf{F} \cdot \mathbf{v} \, dt \dots\dots\dots (2)$$

The input force $\mathbf{F}$ can be found by means of a transfer function relating force to acceleration, so that in practice an accelerometer can be used to measure the force. This transfer function (which represents the effective body mass at a given frequency) was developed through empirical tests [29].

Alternatively, the absorbed power may be found from acceleration alone. Since force and velocity can both be expressed in terms of acceleration, equation 2 can be manipulated to give absorbed power as a function of acceleration only. The RMS acceleration of vibration is modified by a "conversion constant" that varies with frequency, and which converts RMS acceleration to average absorbed power. The conversion constants are available as polynomial functions of frequency, with units of Ws^4/ft^2 [29]. For broadband vibration, the converted acceleration spectral density (i.e. the absorbed power spectral density) is used:

$$P = \int_0^\infty S_a \, df \dots\dots\dots (3)$$

Because absorbed power is a scalar quantity, it can be summed for orthogonal directions, and even for multiple input points. Thus, the absorbed power technique allows concurrent vibration inputs through the seat and feet to be assessed.

Empirical studies have shown that under normal conditions a driver will tolerate a continuous vibrational input corresponding to an absorbed power level of 6 watts [7]. Under extraordinary conditions, higher levels may be tolerated for

shorter periods of time. The 6-watt limit is used by NRMM for computing the ride-limited vehicle speed. Absorbed power is used extensively in the Canadian and American defence establishments, but is not commonly used elsewhere.

1.5.2 ISO 2631

ISO 2631 was adopted by the International Organization for Standardization (ISO) from a United States military standard, MIL STD 1472A [27]. This standard is based on empirical studies of vibration tolerance in three axes: vertical, lateral, and fore-aft. Essentially it relates the root-mean-square (RMS) acceleration of a vibration of known frequency to three limiting boundaries: reduced comfort, fatigue/decreased proficiency, and exposure limit. The reduced comfort boundary is suggested for passenger environments - for example trains, ships, or aircraft that transport passengers. RMS accelerations in these environments should not exceed the reduced comfort boundary. The fatigue/decreased proficiency boundary is intended for guidance in situations where workers are subjected to continuous vibration (agricultural equipment operators, for instance). The exposure limit boundary is proposed as a limit beyond which there may be risk of injury.

Unlike the absorbed power method, the limits described do not represent any physical phenomena, but merely delineate the acceptable accelerations found during jury-type testing. ISO 2631 thus makes no attempt to explain the mechanism responsible for human vibration response.

One important assumption of ISO 2631 is that no significant interactions occur between vibrations of different frequencies. Thus, for a given vibration environment, only the worst frequency component of the vibration spectrum need be considered to assess ride quality. In the case of broadband vibration, as might occur when traversing a random roughness surface, the RMS accelerations at the centre frequencies of a standardized series of one-third octave bands (from 1.0 to 80 Hz) are calculated. The least tolerable acceleration is selected for ride assessment.

The one-third octave band analysis may make it difficult to compare different ride qualities. To facilitate such a comparison, ISO 2631 permits the calculation of a frequency-weighted RMS acceleration. These frequency weightings are actually the inverse of the fatigue/decreased proficiency boundary. For situations in which the acceleration in more than one axis is significant, the axis-weighted vector sum of the frequency weighted accelerations may be found, as follows:

$$\hat{a}_{eq} = \sqrt{(1.4 \cdot \hat{a}_{xw}^2) + (1.4 \cdot \hat{a}_{zw}^2) + \hat{a}_{yw}^2} \dots\dots\dots (4)$$

This equivalent acceleration may be compared with similarly obtained values for a relative indication of ride quality, or can be compared directly with vertical acceleration limits in the 4 to 8 Hz band to obtain the recommended exposure duration for a given comfort regime (reduced comfort, fatigue/decreased proficiency, or exposure limit). ISO 2631 also permits the calculation of acceptable acceleration given the duration of the vibration and the comfort regime of interest:

$$a_t = a_1 \cdot (T_0 / T)^{1/4} \dots\dots\dots (5)$$

1.5.3 Griffin's Criterion

Griffin's Vibration Dose Value (VDV) is a fundamentally different approach to ride quality assessment from that of either absorbed power or ISO 2631. It was proposed as an alternative to other analysis techniques in order to address concerns with the applicability of existing methods to non-ergodic vibration environments [31]. It has been used as a ride assessment criterion in some computer models [8, 13].

A non-ergodic environment is one in which the statistical measures of vibration are not stationary, but vary over time. Environments in which mechanical shocks occur irregularly are good examples of non-ergodicity. One measure of ergodicity is crest factor. Crest factor is defined as the ratio of the maximum variation from the mean of a signal, to the RMS value of the signal. Although ISO 2631 allows the use of its ride boundaries and weighting factors for vibration time histories with crest factors up to six, caution is advised when applying the standard to vibration environments with crest factors higher than three.

The VDV is unique because it is a cumulative value rather than a time average. Based on the premise that the effects of vibration are additive over time, it has the advantage that high shock levels are directly reflected in the assessment, and not flattened out by averaging. Griffin's general methodology borrows heavily from ISO 2631, but with some important differences.

The VDV is formally defined as:

$$\text{VDV} = \left[\int_{t=0}^{t=T} \dot{a}^4 dt \right]^{\frac{1}{4}} \dots\dots\dots (6)$$

The fourth power is used to reflect the acceleration-time relationship that Griffin has identified in subjective tests: a doubling of vibration acceleration amplitude will result in a sixteen-fold drop in tolerable duration. Note that the VDV has units of metres per second raised to the power of 1.75, and that it is numerically equivalent to the root mean quad acceleration with duration one second having the same effect.

The frequency weightings are similar to those of ISO 2631. Griffin has chosen to de-emphasize the lower frequency range of vertical accelerations in favour of the higher frequencies, and has expanded the scope of ISO 2631 by including nine new acceleration inputs, along with appropriate weightings:

- a. three linear orthogonal inputs for the feet;
- b. three linear orthogonal inputs for the back; and
- c. three rotational inputs about the orthogonal seat axes.

Naturally, these extra inputs should only be used when appropriate. The VDV's for separate concurrent inputs may be summed by weighting each VDV according to the axis of input, and taking the quad root of the sum of quad weighted VDV's. This ensures that the total VDV reflects the severity of the most serious individual vibrations.

As a general guide, Griffin advises that a VDV of 15 m/s^{1.75} is approximately equivalent to the ISO 2631 4½ hour exposure limit (i.e. both represent the same vibration acceleration). Griffin suggests that vibration exceeding this limit should be approached cautiously in terms of health and safety hazard to exposed persons.

In addition to the Vibration Dose and Vibration Dose Value, Griffin has also defined several other useful quantities:

- a. the RMS acceleration may be calculated using Griffin's frequency weightings for direct comparison to ISO 2631 results where the crest factor is low;
- b. a comfort scale matched to these RMS values may be used to aid in qualitative assessment of ride; and
- c. the time to reach a VDV of 15 (T_{15VDV}) can be calculated to allow comparison between the VDV and ISO 2631. T_{15VDV} is defined as:

$$T_{15VDV} = \frac{15^4}{VDV^4} T \dots\dots\dots (7)$$

Like ISO 2631, Griffin's criterion seeks to describe human vibration response, rather than to explain the underlying mechanism.

CHAPTER II

DESCRIPTION OF THE MODEL

2.1 Introduction

RPM was originally conceived as a modification to Hansen's model, incorporating some of his recommendations (such as including drive torque and rolling resistance), as well as implementing variable track tension within the lower track. However, as the scope of the changes widened to include adding trailing-arm suspensions, more realistic suspension kinematics, a simplified catenary model, improved hubpath calculation, and ride prediction using ride quality criteria, it became clear that the band-aid modification of the original model was becoming unwieldy. It was thus decided to generate entirely new code with new methodology, borrowing only basic concepts from Hansen's model. At the same time, to obtain the best effect from implementation of the new model on an IBM compatible PC (including access to screen graphics functions in order to build a user interface), Pascal was chosen in favour of FORTRAN as the programming language. The result is a more accessible program that models vehicles in a much more comprehensive and realistic manner.

RPM is organized into three processing stages comprising four different modules, each stage self-contained, but relying upon the results generated by the

preceding stage. These stages may be referred to as the pre-processor, the main processor, and the post-processor (Figure 8).

The pre-processor, comprising two independent modules, computes basic information required by the main processor. The first module, TERPRED.EXE, is the hubpath prediction model. It takes the wheel radii used on the vehicle and the terrain profile over which the vehicle is to traverse, and calculates the displacement of the wheel centres as a function of horizontal distance from the profile origin. The second module, STATIC.EXE, uses geometric, inertial, and mechanical properties of the vehicle and its suspension to determine the static equilibrium position of the vehicle on hard, level ground.

The main processor, RPM.EXE, uses vehicle and terrain data, as well as the static equilibrium and hubpath data generated by the pre-processor, to create an output table giving elapsed time, elapsed distance, and hull heave, pitch, and roll displacements, velocities, and accelerations. This table represents the solution of the non-linear differential equations describing the vehicle's dynamic behaviour.

The post-processor consists of the module RIDE-Q.EXE. This processing stage takes the output table from the main processor, and analyzes the time-history data in the frequency domain using three separate ride criteria. These ride criteria (absorbed power, ISO 2631, and Griffin's VDV) represent a wide range of methodologies commonly used in ride quality assessment. They give a consistent

Figure 8: Block diagram for RPM.

means of comparing the rides of different vehicles, or of similar vehicles on different terrains.

2.2 Solution of the Equations of Motion

A model having separate degrees of freedom for each wheel, in addition to those for the vehicle body, requires the solution of a stiff set of differential equations [6, 7] (i.e. the modelled system is typified by high natural frequencies associated with the vehicle road wheels). Since the solution of the differential equations of motion requires the use of a stepsize smaller than the period of the highest natural frequency, a model of this sort requires a great deal of computer time. By reducing the model to only three degrees of freedom (bounce, pitch, and roll of the hull), the number of equations to be solved are reduced, and larger stepsizes can be used since the hull natural frequencies are much lower than those of the wheels. The roadwheels, instead of interacting freely with the ground, are thus constrained to follow the surface profile, which effectively acts as a forcing function for the vehicle system, transformed by the filtering effects of the tracks and wheels.

The dynamic motion of the vehicle hull can be described in terms of the generalized coordinates (y, θ, ϕ) which give the heave, pitch, and roll displacements about the centre of gravity (C.G.) of the vehicle. Since the wheels are constrained to follow the profile, the forces acting on the hull at any instant can be determined given the vehicle's position with respect to the surface profile, and the recent

dynamic history of the vehicle (specifically, the relative velocity between the hull and the roadwheel centres, which is required in order to calculate damper forces). Once the forces are known, the heave, pitch, and roll accelerations of the C.G. can be found using Newton's Second Law of Motion:

$$y'' = \Sigma F_y / m \dots\dots\dots (8)$$

$$\theta'' = \Sigma (F_x \cdot L_y + F_y \cdot L_x) / I_z \dots\dots\dots (9)$$

$$\phi'' = \Sigma (F_x \cdot \sin(-\theta) \cdot L_z + F_y \cdot \cos\theta \cdot L_z) / I_{xx} \dots\dots\dots (10)$$

These accelerations can then be double integrated to find the vehicle displacement given by the generalized coordinates.

RPM uses a fixed stepsize, fourth order Runge-Kutta routine to perform the double integration. The vehicle system is reduced to six independent, linear differential equations:

$$\begin{aligned} DY_1 &= Y_1' = y' \\ DY_2 &= Y_2' = \theta' \\ DY_3 &= Y_3' = \phi' \dots\dots\dots (11) \\ DY_4 &= Y_4' = DY_1' = y'' \\ DY_5 &= Y_5' = DY_2' = \theta'' \\ DY_6 &= Y_6' = DY_3' = \phi'' \end{aligned}$$

The vehicle is initially in the static equilibrium position, with the first and second derivatives of the general coordinates set to zero. To ensure that there are no instantaneous forces applied to the hull, the vehicle begins its traverse of the terrain

on flat, level ground, which is appended to the terrain profile. The vertical displacement of this lead-on terrain is set equal to the vertical displacement of the first terrain data point.

Calculation of the second derivatives of the generalized coordinates requires the evaluation of the three main types of forces acting on the vehicle hull: spring forces, damping forces, and track forces. These forces can only be determined after the position of the wheels with respect to the vehicle hull is known.

2.3 Determining Roadwheel Location

Hansen's approach to determining the instantaneous location of the roadwheels was to use an iterative technique to ascertain the profile point which would first contact the wheel if the wheel were allowed to slide toward the ground along its suspension locus (Figure 9). Because he assumed that the suspension locus always remains perpendicular to the horizontal regardless of vehicle attitude, the horizontal displacement of the wheel hub was fixed by vehicle position, and it was only necessary to solve for the hub's vertical displacement. This was accomplished by sampling an arbitrary and finite number of terrain points beneath the wheel; the terrain point which, when in contact with the wheel, resulted in the highest wheel hub location was deemed to be the ground-wheel contact point.

This methodology had several disadvantages. First, the assumption that the suspension locus remains normal to the horizontal is justified only for pillar-type

Figure 9: Hansen's method of determining roadwheel location.

suspensions and small hull pitch angles, and is thus invalid for trailing-arm suspensions and large pitch angles (which are not uncommon during off-road travel). Adapting the methodology to accommodate these factors would greatly increase its complexity, since both horizontal and vertical displacement would have to be solved for, instead of just the vertical displacement. Second, the amount of computer time required is fairly high since the hub height must be calculated for every sampled profile point. Even on flat, level terrain where calculation of the hub height is trivial, the terrain must be sampled at a set number of profile points. And third, good accuracy requires that a large number of terrain profile points be sampled, which increases the time penalty further. It was therefore necessary to develop a new methodology that would accommodate more complex suspension kinematics and more realistic hull pitch angles, and that would avoid turning the subroutine into a computational bottleneck on relatively slow personal computers.

The chief complication of this task was due to the fact that it is not just any ground-wheel intersection point that must be found, but the ground-wheel osculation point (i.e. the intersection point for which ground and wheel are tangent). There may in fact be multiple osculation points, in which case the point which results in the highest wheel hub displacement must be determined (Figure 10).

The hubpath (i.e. the locus of the wheel centre as it follows the terrain profile) can be generated by sliding a line segment of length R (the radius of the wheel) along the profile, while keeping it perpendicular to the profile at each point (Figure 11). The hub will always be above the point of ground-wheel contact (which

Figure 10: Multiple osculation points for a given suspension locus. Four osculation points are shown here.

Figure 11: Hubpath geometry with respect to terrain profile.

is to say that the perpendicular will always be at an angle to the horizontal of between 0° and 180°). The slope of the hubpath is equal to the slope at the corresponding point of ground-wheel contact. This can be expressed mathematically by the following parametric equations:

$$x = u - R \cdot \sin \{ \arctan [f'(u)] \} \dots\dots\dots (12)$$

$$y = f(u) + R \cdot \cos \{ \arctan [f'(u)] \} \dots\dots\dots (13)$$

Therefore, coordinates $(u, f(u))$ on the surface profile can be mapped onto coordinates (x, y) on the hubpath. Unfortunately, the parameters cannot be separated to give $(u, f(u))$ as a function of x and y . That is, given the horizontal coordinate of a terrain point, the corresponding wheel hub location can be found; however, given the horizontal coordinate of the wheel hub, the corresponding terrain profile point cannot be determined. This is awkward, since during the simulation process the suspension locus is known, and the terrain profile point is unknown.

This difficulty is overcome in RPM by defining the hubpath prior to the dynamic simulation, and solving for the intersection between the hubpath and the suspension locus during the actual simulation. This method provides both good efficiency and accuracy, and reduces the burden of computation required in the main processor. Instead of solving for the osculation of a circle and a curve, the problem is reduced to solving for the intersection point of two curves (i.e. the intersection of the hubpath and the suspension locus), which is almost trivial (Figure 12). In practice, there will almost always be only one intersection point for

Figure 12: Location of roadwheel by intersection of suspension and hubpath loci.

the two curves, as opposed to multiple osculation points. In addition, since the hubpath is generated by a pre-processor, it is possible to use the same hubpath in any number of other simulations without incurring additional overhead.

During generation of the hubpath, a complication may be introduced by the existence of hubpath retrograde loops. If the radius of curvature of the ground profile at a point is less than the radius of the wheel, and if the centre of curvature lies above the ground, then the wheel can never be tangent to the ground at that point, but will bridge the depression (Figure 13). However, rigorous application of equations 12 and 13 under these circumstances will create a loop in the hubpath that suggests the hub undergoes a retrograde motion. This is physically impossible, and thus these retrograde loops must be identified and removed when the hubpath is generated (Figure 14).

The algorithm used by TERPRED.EXE, the hub-path generation module, is as follows (see Figure 15):

- a. a cubic spline is fitted to the series of profile points provided by the user;
- b. at equidistant points on this spline, equations 12 and 13 are applied to determine the corresponding wheel centre location;
- c. the wheel profile points are checked to ensure that no retrograde occurs. This will happen if the profile has crevices that are too narrow

Figure 13: Terrain bridging by wheel. Radius of curvature of concave terrain feature (r) is less than radius of roadwheel (R).

Figure 14: Retrograde motion from application of hubpath equations.

Figure 15: Hubpath generation by TERPRED.EXE.

for the wheel to enter. In this case, the node at the intersection point of the retrograde loop must be found, and data points that are part of the loop must be stripped from the set. A retrograde loop is detected by checking that consecutive wheel centre horizontal displacements are always increasing positively - a decrease indicates retrograde;

- d. a cubic spline is fitted to the wheel centre points;
- e. equidistant points on the wheel centre profile are interpolated; and
- f. the procedure is repeated for other wheel radii that may be present on the vehicle.

This procedure may produce an erroneous wheel profile if discontinuities occur in the original terrain profile. For this reason, hubpaths for discrete obstacles such as half-rounds, blocks, or ramps should be generated manually. Paths should be inspected visually to ensure they have no obvious aberrations.

Within the main processor, the generated hubpath is used as the suspension forcing function. Instead of solving for the osculation of the wheel and terrain, the program finds the discrete intersection point between the wheel centre locus (as defined by suspension geometry) and the wheel path, using analytical methods. Linear interpolation of hubpath points is used.

2.4 Forces Acting on the Roadwheels

Once the locations of the roadwheels with respect to the vehicle hull are known, the forces acting on the roadwheels can be assessed. Because the dynamic effects of the roadwheels are ignored, the wheels can be considered to be in static equilibrium at any instant [6]. It is apparent that if the suspension forces acting on the roadwheels are known, the suspension forces acting on the vehicle hull are likewise known (Newton's Third Law of Motion). Therefore, the determination of these forces is necessary for the solution of equations 8 through 10.

2.4.1 Suspension Forces Acting on the Roadwheel

The pillar suspension consists of a helical spring fixed as a cantilever to the hull at one end, and to the wheel hub at the other end. A telescopic damper may be placed in parallel to the spring. If the net force at the wheel hub is aligned exactly axial to the spring, a pure spring force will be created. If however, the force is not exactly axial, a reaction force perpendicular to the spring axis will have to be supported. Since the damper is assumed to be pin-jointed at both ends, it cannot support lateral loads, and so the spring is assumed to carry both the compressive spring load and the lateral perpendicular reaction. Because the lateral force at the hull attachment point is equal and opposite to the lateral force at the hub, a moment will also be generated that must be resisted at the hull attachment point.

The trailing arm suspension consists of a transverse torsion bar fixed to the hull and linked to the roadwheel by a rigid arm. A damper may be connected between the hull and the wheel hub. If the net force at the hub does not act exactly perpendicularly to the roadarm, the arm will have to support an axial reaction force in addition to the perpendicular spring force. Because the damper is again assumed to be pin-jointed at both ends, it cannot support any lateral loads.

The helical spring force is calculated by finding the difference in spring length between the free uncompressed spring and the instantaneous length, multiplied by the linear spring constant. If the bumpstop has been reached, the spring constant is the parallel stiffness of both springs. The spring force always acts along the axis of the spring.

The spring torque for a trailing arm suspension is calculated by finding the instantaneous wind-up from the roadarm's free angle and multiplying by the torsional stiffness of the torsion bar. If the bumpstop has been reached, the spring constant is the parallel stiffness of both springs. Associated with the spring torque is a spring force which acts perpendicular to the roadarm, and is simply the ratio of the spring torque to the roadarm length.

The damper force is the product of the relative speed of the damper ends and the constant damping coefficient. Damper velocity is found by determining the difference in length of the damper at two sequential instants in time, and dividing the difference by the time interval. If the calculated velocity is greater than the

blow-off velocity, maximum damping force is assumed. The damping force always acts along the axis of the damper.

2.4.2 Ground Forces Acting on the Roadwheel

Two orthogonal forces exist at the osculation point of the wheel and the ground, one of which acts radially towards the wheel centre, and one of which acts tangentially to the wheel. The first of these is the normal or ground reaction force, while the second force is the ground shear force. The ground is assumed to support any normal or shear force without deformation. The shear force can be generated by either tractive or braking effort applied to the running gear.

2.4.3 Track Forces Acting on the Roadwheel

For tracked vehicles, tractive effort is applied against the ground through the track. Since the idler is assumed to free spin on frictionless bearings, application of torque to the drive sprocket will result in a higher track tension at the back of the vehicle than at the front. Therefore, the tension in the track lying on the ground must drop between the rear and front roadwheels. This drop is due to shear forces at the track-ground interface. Since the shear forces are idealized to exist only at the ground-wheel osculation points, each roadwheel will experience a high track tension to the rear of the osculation point, and a low track tension forward of the osculation point. The net track force will generate a reaction force at the wheel

hub. Under certain conditions, the track tension may be sufficient to lift the roadwheel off the ground.

2.4.4 External Resistances Acting on the Roadwheel

A real vehicle must continuously perform work to overcome external forces acting against it if it is to maintain a constant forward speed. RPM allows constant aerodynamic and rolling resistance to be applied to the vehicle to simulate these real-world forces.

Aerodynamic resistance is a function of relative airspeed, aerodynamic drag coefficient, and vehicle frontal area. Although it may be less significant for heavy tracked vehicles moving slowly over soft soil than the resistances associated with ground sinkage, aerodynamic resistance will represent the most significant loss for wheeled road vehicles moving at high speed on hard pavement. The general formula for aerodynamic drag is as follows:

$$R_a = \frac{D_a \cdot C_d \cdot A_f \cdot V^2}{2} \dots\dots\dots (14)$$

The value of the drag coefficient C_d is quite variable, but a value of 1.17 can be taken to be representative of many off-road vehicles [15]. The aerodynamic drag acts through the aerodynamic centre of the vehicle, the location of which depends on the direction of air flow with respect to the vehicle. If the aerodynamic centre is not coincident with the vehicle C.G., a moment will be generated about the C.G.

RPM assigns a constant aerodynamic drag arbitrarily, based on vehicle type and vehicle speed. Allowable vehicle types are trucks, automobiles, motorcycles, heavy tracked vehicles, and light tracked vehicles. Each type has a typical frontal area and drag coefficient which may not be changed by the user. The aerodynamic centre is assumed by RPM to always be coincident with the vehicle C.G. The dynamic effect of aerodynamic resistance is thus restricted to the pitching moment generated by the tractive effort required to overcome the resistance.

Rolling resistance is a major source of motion resistance for both wheeled and tracked vehicles. It is especially significant for tracked vehicles, the specific rolling resistance being three to four times higher for tracked vehicles than wheeled vehicles [32]. The main mechanism contributing to rolling resistance for tracked vehicles is internal track friction, specifically flexing of the rubber bushings. It has been estimated that 70% to 85% of rolling resistance losses for heavy military tracked vehicles is due to internal track friction.

There are several empirical expressions available for tracked vehicle rolling resistance, of which two are presented here [32]. The first is applicable to heavy military tracked vehicles (velocity in m/s):

$$f_r = 3.8 + 0.17 V \dots\dots\dots 15)$$

The second is appropriate for modern, light-armoured vehicles (for example, tracked APCs):

$$f_r = 3.0 + 0.17 V \dots\dots\dots 16)$$

The specific rolling resistance is strongly influenced by vehicle speed.

Pneumatically-tired vehicles, in contrast, are assigned a speed-independent rolling resistance based on weight, since the rolling resistance is relatively insensitive to speed for these vehicles.

RPM calculates rolling resistance on the basis of vehicle type, weight, and speed. The rolling resistance is assumed to act at the vehicle C.G. Therefore, the dynamic effects of the rolling resistance are restricted to the pitching moment generated by the tractive effort required to overcome the rolling resistance.

2.5 Solution of Wheel Forces

Six distinct cases can be identified for which static equilibrium equations for the roadwheel must be developed:

- a. pillar suspension without tracks;
- b. pillar suspension with tracks and front drive sprocket;
- c. pillar suspension with tracks and rear drive sprocket;
- d. trailing arm suspension without tracks;
- e. trailing arm suspension with tracks and front drive sprocket; and
- f. trailing arm suspension with tracks and rear drive sprocket.

The distinction between front and rear drive-sprockets reflects the fact that the drive torque is unknown at this point while the idler torque is known to be always zero (since the idler is assumed to be mounted on frictionless bearings). Therefore, for tracked vehicles the equations must always be solved from the roadwheel nearest the idler to the roadwheel nearest the drive sprocket. Figures 16 and 17 depict the generic free-body diagrams for the pillar and trailing arm suspensions respectively.

Examination of Figure 16 reveals ten forces, five of which are known (F_a , T_1 , W , R_a , and R_r), and five of which are unknown (F_p , M , T_2 , τ , and N). From Figure 17, there are twelve forces, seven of which are known (M , D_x , D_y , T_1 , W , R_a , and R_r), and five of which are unknown (F_x , F_y , T_2 , τ , and N). This suggests a need for five equations. The first of these is trivial, since the drop in tension must equal the shear force on the ground for tracked vehicles:

$$\tau = (T_2 - T_1) \dots\dots\dots (17)$$

τ can thus be replaced to leave four unknowns, requiring four independent equations. Three equations are immediately obvious:

$$\Sigma F_x = 0 \dots\dots\dots (18)$$

$$\Sigma F_y = 0 \dots\dots\dots (19)$$

$$\Sigma M_0 = 0 \dots\dots\dots (20)$$

A fourth equation is required for tracked wheels. As noted earlier, the external resistances R_a and R_r are assumed to act at the vehicle C.G. They do not

Figure 16: Free-body diagram for pillar suspension.

Figure 17: Free-body diagram for trailing-arm suspension.

therefore directly affect the wheel equations. However, to maintain constant forward vehicle velocity, these external forces must be opposed by the combined tractive efforts available at each wheel. An artificial constraint is therefore imposed: the tractive effort required to oppose the rolling and aerodynamic resistance is distributed equally among all wheels on the ground:

$$N \cdot \sin\theta + (T_2 - T_1) \cdot \cos\theta = (R_a + R_r)/n \quad \dots\dots\dots (21)$$

This constraint is necessary because there is no other simple basis for determining the distribution of shear forces on the ground.

Note that while the trailing arm suspension type is solved using global coordinates, the pillar suspension type is rotated so that the suspension axis is oriented to the vertical. This rotation greatly reduces the complexity of the equations, since the spring and damper forces both act along the axis of the suspension. It is therefore natural to calculate axial and perpendicular forces, without converting first to global coordinates.

For untracked wheels on a pillar suspension, the equations of static equilibrium in matrix form are as follows (equation 22):

$$\begin{bmatrix} -\cos\alpha & 0 & \sin\theta \\ -\sin\alpha & 0 & \cos\theta \\ L_s & 1 & 0 \end{bmatrix} \begin{bmatrix} F_p \\ M \\ N \end{bmatrix} = \begin{bmatrix} F_{ax} \\ F_{ay} + W \\ 0 \end{bmatrix}$$

For tracked wheels on a pillar suspension, the equations of static equilibrium in matrix form are as follows (equation 23):

$$\begin{bmatrix} -\cos\alpha & 0 & \pm(\cos\delta - \cos\theta) & \sin\theta \\ -\sin\alpha & 0 & (\sin\delta - \pm\sin\theta) & \cos\theta \\ L_s & 1 & 0 & 0 \\ 0 & 0 & -\pm\cos\theta & \sin\theta \end{bmatrix} \begin{bmatrix} F_p \\ M \\ T_2 \\ N \end{bmatrix} = \begin{bmatrix} F_{ax} + \pm T_1(\cos\beta - \cos\theta) \\ F_{ay} - T_1(\sin\beta + \pm\sin\theta) + W \\ 0 \\ -\pm T_1\cos\theta + (R_r + R_a)/n \end{bmatrix}$$

The sign functions ("±") should be replaced by "+1" for front sprocket vehicles, and "-1" for rear sprocket vehicles.

For untracked wheels on a trailing-arm suspension, the equations of static equilibrium in matrix form are as follows (equation 24):

$$\begin{bmatrix} -1 & 0 & -\sin\theta \\ 0 & -1 & \cos\theta \\ (d-b) & (a-c) & 0 \end{bmatrix} \begin{bmatrix} F_x \\ F_y \\ N \end{bmatrix} = \begin{bmatrix} D_x \\ D_y + W \\ M \end{bmatrix}$$

For tracked wheels on a trailing-arm suspension, the equations of static equilibrium in matrix form are as follows (equation 25):

$$\begin{bmatrix} -1 & 0 & \pm(\cos\delta - \cos\theta) & -\sin\theta \\ 0 & -1 & (\sin\delta - \pm\sin\theta) & \cos\theta \\ (d-b) & (a-c) & 0 & 0 \\ 0 & 0 & -\pm\cos\theta & -\sin\theta \end{bmatrix} \begin{bmatrix} F_x \\ F_y \\ T_2 \\ N \end{bmatrix} = \begin{bmatrix} D_x + \pm T_1(\cos\beta - \cos\theta) \\ D_y - T_1(\sin\beta + \pm\sin\theta) + W \\ M \\ -\pm T_1\cos\theta + (R_r + R_a)/n \end{bmatrix}$$

The equations of static equilibrium are solved using Gaussian elimination with pivotal condensation. As can be seen, the fourth constraint equation is

unnneeded and thus discarded if no track is present. Tracked wheels must be solved in sequence starting at the wheel station nearest the idler, and working towards the drive sprocket, since the track tension at the idler is known and the tension at the drive sprocket is unknown. Tracked wheels are solved in cascade (i.e. the unknown track tension of one wheel becomes the known track tension for the adjacent wheel after solution), while untracked wheels can be solved independently of one another.

If the ground reaction force is negative, a tensile force acting between the ground and wheel is implied. Since there is no mechanism to generate tensile forces, this must be interpreted as meaning the forces on the wheel are sufficient to pull it off the ground. The wheel is therefore placed in its zero-force configuration, and the weight of the wheel assembly is added to the sprung mass. The number of wheels on the ground is decremented by one.

2.6 Tracks

Like Hansen's model, RPM considers the track to consist of two distinct segments: an upper segment which hangs freely between the drive sprocket and idler, and a lower segment which lies on the ground between the sprocket and idler. The upper track is assumed to act as a catenary, defined in terms of the inclination and distance between the sprocket and idler (which are simple functions of the vehicle attitude), and by the amount of track hanging between the supports. From track continuity, the amount of track in the upper segment is known to be the difference of the total track and lower segment lengths. Computation of the

catenary characteristics therefore requires the calculation of the track length on the ground.

2.6.1 Track Length on the Ground

The lower track segment can be subdivided into six sub-segment types (see Figure 18):

- a. the front spridler* arc. The arc length from the top of the spridler to the tangency point with the front track segment;
- b. the front track segment. The linear segment from the front spridler tangency point to the first roadwheel tangency point;
- c. the roadwheel arc. The arc between the tangency points of the anterior and posterior linear track segments on either side of a roadwheel;
- d. the intermediate track segment. The track segment lying between adjacent roadwheels;

* spridler is an NRMM term combining "drive sprocket" and "idler wheel", and meaning either.

Figure 18: Constituent segments of lower track.

- e. the rear track segment. The linear segment from the last roadwheel tangency point to the rear spridler tangency point; and
- f. the rear spridler arc. The arc length between the tangency point with the rear track segment and the top of the rear spridler.

The remainder of the track hangs between the tops of the spridlers, and forms the upper track segment. Summation of the individual lengths of the six types of track sub-segments noted above yields the total lower track length.

Evaluation of the lengths of the front and rear spridler arcs, and the front and rear track segments is trivial, requiring only the application of equations defining the tangent between two circles [33]:

$$x_i = x_{ci} + R_i \cdot \left[\frac{-\delta R \cdot \delta x_c + \delta y_c \cdot \sqrt{\delta x_c^2 + \delta y_c^2 - \delta R^2}}{\delta x_c^2 + \delta y_c^2} \right] \dots\dots\dots (26)$$

$$y_i = y_{ci} + R_i \cdot \left[\frac{-\delta R \cdot \delta y_c - \delta x_c \cdot \sqrt{\delta x_c^2 + \delta y_c^2 - \delta R^2}}{\delta x_c^2 + \delta y_c^2} \right] \dots\dots\dots (27)$$

Calculation of the intermediate track segment lengths, and of the roadwheel arcs is more complicated. Prior to calculating ground track length, the locations of the wheels as defined by hull attitude, suspension geometry, and ground profile are determined. It is known that the track must be tangent to the wheel and ground at the wheel-ground contact point. Given the contact points of adjacent roadwheels, the initial and terminal points of the intermediate track segment are known. Since the track on the ground is assumed to be incapable of supporting bending moments,

it must conform to convex objects (i.e. roadwheels and terrain surface bumps) when placed under tension, but not concave objects (i.e. terrain surface depressions). This implies that the track slope is constrained to always decrease between the rearward roadwheel and the adjacent front roadwheel. Any increase in slope would indicate compliance to a cavity, which is not allowed (Figure 19).

To ensure the track slope meets this requirement, the terrain interval between adjacent wheel-ground contact points is subdivided into an arbitrary number of points at which the terrain displacement is found. The slope between each point and its preceding point is tested. If the slope increases between consecutive segments, the adjacent points may define part of the track length. If the slope decreases, the preceding point cannot define part of the track segment since it lies in a cavity (Figure 20). That point is discarded, and testing continues, starting with the last good point and the present point. It is possible that all the intermediate points will be discarded, in which case the track segment is simply the tangent length between the two roadwheels.

The slope between the front wheel-ground contact point and the first good intermediate point must be tested to ensure that the slope is not less than the wheel tangent at the contact point. If it is, the track would lie inside the wheel, which is physically impossible. In this case, the tangent between the roadwheel and the first good intermediate point must be determined. A similar procedure is used for the rear roadwheel, except that the slope must not be greater than the tangent, and the last good intermediate point is used (Figure 21).

Figure 19: Constraint on track slope.

Figure 20: Evaluation of intermediate segment length.

if $a > b$, track will interfere with wheel
therefore, must find track/wheel tangency point

Figure 21: Evaluation of track-roadwheel tangency point.

The length of the intermediate track segment is the sum of the lengths of the sub-segments between adjacent terrain sample points. The length of the roadwheel arc can be found once the track tangency points for the intermediate track segments are known.

2.6.2 Track Tension Due to Sag

Once the lower track length is known, the shape and tension of the upper track segment can be investigated. If the track sag between the drive sprocket and the idler or supporting roadwheels is modelled as a catenary, the sag and tension are governed by the following catenary equations [6]:

$$H = \frac{T_T}{w} \left[\cosh \left[\frac{w(L_c - a_c)}{T_T} \right] - \cosh \left[\frac{-wa_c}{T_T} \right] \right] \dots\dots\dots (28)$$

$$s = \frac{T_T}{w} \left[\sinh \left[\frac{w(L_c - a_c)}{T_T} \right] - \sinh \left[\frac{-wa_c}{T_T} \right] \right] \dots\dots\dots (29)$$

The unknowns in these equations are the horizontal component of track tension, T , and the distance from the catenary minimum to either support, a . These equations unfortunately cannot be solved explicitly for the unknowns, but must be solved iteratively.

To avoid computationally slow iterative methods, it was decided to approximate the catenary curve with a circular arc. The length of the arc is set

equal to the length of track suspended in the catenary, with the arc definition being completed by the chord between the two suspension points. If the exact equations describing arc length are used, the same problem as with the catenary equations is encountered: the equations cannot be solved explicitly for tension in terms of the known parameters. However, Huyghen's approximation formula gives a very accurate expression for arc length in a form that can be algebraically manipulated [34]:

$$s = \frac{8 \cdot L - D}{3} \dots\dots\dots (30)$$

Using geometric properties, the centre of the arc can be found, and thus the angles at the supports can be calculated (Figure 22):

$$\alpha_1 = \beta_c + 180^\circ - 2 \cdot \arcsin \left[\frac{4 \cdot D}{3 \cdot s + D} \right] \dots\dots\dots (31)$$

$$\alpha_2 = \beta_c - 180^\circ + 2 \cdot \arcsin \left[\frac{4 \cdot D}{3 \cdot s + D} \right] \dots\dots\dots (32)$$

The vertical sum of tensions must equal the weight within the catenary, while the horizontal sum of tensions must be zero. From these constraints, the total tension at each support can be determined:

$$T_1 = \frac{w}{\sin \alpha_1 - \cos \alpha_1 \cdot \tan \alpha_2} \dots\dots\dots (33)$$

$$T_2 = \frac{w}{\cos \alpha_2 \cdot \tan \alpha_1 - \sin \alpha_1} \dots\dots\dots (34)$$

Figure 22: Geometry of pseudo-catenary (circular arc).

Figure 23 shows the error introduced by using circular arcs (pseudo-catenaries) to approximate catenaries for span-to-track-length ratios between 0.9 and 0.999, and for support inclinations (the horizontal angle between the arc endpoints) of 0° to 30°. The jaggedness of the curves reflects the small number of data points used. For comparison, the M113 track displacement curve given in Figure 3 has a span-to-track-length ratio of 0.968 between the drive sprocket and the third roadwheel. The error given in Figure 23 is expressed in terms of the total tension calculated for the upper track support. Between ratios of 0.9 and about 0.997 the error is within $\pm 10\%$ and quite acceptable. At higher ratios, the catenary equations are no longer applicable, since the longitudinal stiffness of the rubber bushings becomes the governing factor on track tension, and thus the escalating errors at these ratios is not of concern. Track support angles of greater than 30° are unlikely to be encountered, and thus errors for these angles are not shown.

2.7 Evaluation of Static Equilibrium Position (STATIC.EXE)

The static equilibrium position of the vehicle is the attitude of the stationary vehicle on flat, level ground. Because the vehicle suspension is defined in terms of the uncompressed lengths or angles of the springs, the static equilibrium (characterized by the heave, roll, and pitch of the hull) position must be calculated.

To solve for the static equilibrium position, the vehicle is initially positioned with hull pitch and roll set to zero. The position is iterated using a binary search method, with the forces and moments acting on the hull computed after each iteration, until the heave, pitch, and roll accelerations are reduced to some

Figure 23: Error associated with pseudo-catenary approximation.
Circular arcs using Huyghen's approximation.

acceptable limit. The attitude data (which comprises the initial conditions for the solution of the dynamic equations of motion) is then appended to the vehicle data file. Track tension, if applicable, is also appended to this file, for information purposes.

2.8 Ride Quality Assessment (RIDE-Q.EXE)

The ride quality assessment performed by RIDE-Q.EXE uses the heave acceleration acting at the vehicle C.G., and the pitch and roll accelerations, in conjunction with three different ride criteria (absorbed power, ISO 2631, and Griffin) to predict ride severity. The three ride criteria require the transformation of the acceleration time histories into the frequency domain, in order that the accelerations may be appropriately frequency-weighted.

Although the acceleration of the vehicle is expressed in terms of accelerations at the vehicle C.G., the driver or passenger will rarely be seated directly at this point. As the seat location moves away from the C.G., the effects of hull, pitch, and roll become more significant (i.e. the ride quality changes from point to point). To correctly assess the ride quality at the driver/passenger's seat, RPM allows the user to designate any point within the vehicle, or even outside it, at which the ride is to be predicted. The accelerations at the vehicle origin are then transferred to that point.

The driver/passenger is assumed to always be aligned with the local vehicle axes. That is, as the vehicle rotates about the pitch and roll axes, the individual stays fixed with respect to the vehicle. The coordinate axes defining the vertical, fore-aft, and lateral axes through the subject's body must therefore rotate. If the seat location is not at the C.G., rotation of the hull about the pitch or roll axes will generate centripetal effects. RPM considers all rotational effects when transferring the C.G. accelerations to the seat location. The appropriate transformations are as follows:

$$a_x = y'' \sin \theta - x_r (\theta')^2 - \theta'' \cdot (y_r \cos \beta - z_r \sin \beta) \dots \dots \dots (35)$$

$$a_y = [y'' \cos \theta - (y_r \cos \beta - z_r \sin \beta) (\theta')^2 - \theta'' x_r] \cdot \cos \beta - y_r (\beta')^2 + z_r \beta'' \dots (36)$$

$$a_z = [y'' \cos \theta - (y_r \cos \beta - z_r \sin \beta) (\theta')^2 - \theta'' x_r] \cdot \sin \beta - z_r (\beta')^2 + y_r \beta'' \dots (37)$$

In order to differentiate between a_x (fore-aft body acceleration) and a_z (lateral body acceleration) it is necessary to indicate the orientation of the body - either facing the direction of motion, or sitting sideways. This is necessary as the three ride criteria assign different frequency weightings to accelerations in the fore-aft and the lateral directions. Although a driver or passenger will usually sit with his fore-aft axis parallel with that of the vehicle, in some vehicles passengers are seated perpendicular to the direction of motion (eg in military transport vehicles).

Transformation of the acceleration time-histories to the frequency domain is accomplished using a Fast Fourier Transform (FFT) routine. The mathematical definition of discrete Fourier transform used by the FFT is as follows [35]:

$$G(\omega/Q) = \sum_{k=0}^{Q-1} g(kT) \cdot e^{-j\omega k/Q} \dots\dots\dots (38)$$

The FFT makes use of trigonometric manipulation to significantly reduce the number of computations required to calculate the discrete Fourier transform. The FFT algorithm adapted for RPM was very basic, with no consideration given to scaling, axis shifting, or inverse transformation. These capabilities were all implemented through extensive modification of the original algorithm.

Whenever discrete data is used to represent a continuous waveform, there is some loss of information, which may lead to aliasing. Aliasing makes it impossible to identify higher frequency components in a sampled waveform. The Nyquist sampling theorem states that the sampling rate of a general waveform must be at least twice the highest frequency present (which can be reduced by filtering) in the waveform to avoid aliasing. In RPM, the solution to the differential equations is determined at some arbitrary time step chosen by the user (i.e. the continuous solution is discretely sampled), but the output is written to file at a different sampling rate. Effectively, the sampled computed waveform is further sampled for file output. If the frequency band of interest for ride analysis extends to 30 Hz, it follows from the Nyquist sampling theorem that the output step should be no greater than 0.01667 seconds, for a sampling rate of 60 Hz. However, if the

integration time step chosen was 0.001 seconds, the frequency content of the output sampled waveform would extend to 500 Hz. Thus, all frequencies from 60 Hz to 500 Hz in the output sample would be aliased. To avoid this, the Runge-Kutta solution samples are filtered by a four-pole low-pass Butterworth filter (as used by VehDyn II) with cut-off frequency equal to the output sampling frequency (in this case, 60 Hz). This ensures that when the FFT of the output samples is determined, there is no aliasing.

2.8.1 Ride Assessment Using Absorbed Power

Equation 3 gave a formula for calculation of the absorbed power from the power spectral density. Power spectral density is related to the Fourier transform of a function as follows:

$$S_a = \lim_{T \rightarrow \infty} \left[\frac{2}{T} \left| A(f) \right|^2 \right] \dots \dots \dots (39)$$

Substitution of this equation into equation 3 gives Parseval's theorem, which states that energy content in the time domain is identical to energy content in the frequency domain [35]:

$$\int_{-\infty}^{\infty} g^2(t) dt = \int_{-\infty}^{\infty} |G(f)|^2 df \dots \dots \dots (40)$$

Thus, absorbed power as a time domain quantity can be calculated within the frequency domain using the frequency weighted FFT of the acceleration time history, without transforming back to the time domain.

The absorbed power is calculated for the three axes of primary interest, and (if requested by the user) for the vertical foot axis. The sum of these absorbed powers is provided in the output.

2.8.2 Ride Assessment using ISO 2631

ISO 2631 stipulates a series of 20 centre frequencies of third octave bands between 1.0 Hz and 80.0 Hz at which RMS accelerations are to be calculated. For computational reasons, formulae for the centre frequencies and third-octave band limits were derived. The formula for centre frequency is:

$$f_c = 2^{n/3}, \quad n=0, 1, 2, \dots, 19 \quad \dots \dots \dots (41)$$

The formulae for the third octave band limits are:

$$\text{upper limit} = 2^{1/6} \cdot f_c \quad \dots \dots \dots (42)$$

$$\text{lower limit} = 2^{-1/6} \cdot f_c \quad \dots \dots \dots (43)$$

From Parseval's Theorem (equation 40), it is apparent that if the time acceleration $a(t)$ is of finite duration T , and zero elsewhere, the RMS value of $a(t)$ is found by dividing the left-hand side by T and taking the square root. This implies that the RMS value of a function can be identically found by dividing the right-hand side by T and taking the square root (i.e. the RMS value of a time signal can also be calculated in the frequency domain). Simpson's rule is used to integrate the acceleration FFT between the upper and lower limits for each centre frequency, and

then the integrated area is divided by the time history duration to give RMS acceleration.

To calculate the overall weighted acceleration, the transform is frequency weighted, and then transformed back to the time domain. Simpson's method is then used to integrate the squared acceleration time history, which is then divided by the time history duration to give the frequency weighted mean square acceleration. The square root of this gives the desired overall RMS acceleration.

In addition, the crest factor and the equivalent allowable exposure times for the reduced comfort, fatigue/decreased proficiency, and exposure limit boundaries are found. The equivalent allowable exposure time is found by rearranging equation 5 to give:

$$T = (a_1^2 / a_2^2) \cdot T_0 \dots\dots\dots (44)$$

Finally, the axis-weighted RMS acceleration is computed.

2.8.3 Ride Assessment using Griffin's VDV

The assessment using Griffin's VDV is accomplished in a similar way to that using ISO 2631. The vibration dose value is computed using the frequency-weighted acceleration FFT. The weighted transform is transformed back to the time domain.

2

MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS
STANDARD REFERENCE MATERIAL 1010a
(ANSI and ISO TEST CHART No 2)

Simpson's method is then used to integrate the quad acceleration time history. The quad root of this gives the VDV.

The crest factor and the time to 15 VDV are then calculated. For crest factors of less than six, the estimated RMS frequency-weighted acceleration can be useful. The estimated RMS acceleration is calculated as follows:

$$a_{ERMS} = VDV / (1.4 \cdot T^{0.25}) \dots\dots\dots (45)$$

Griffin's comfort scale is used in a slightly modified form to rank ride severity from zero ("not uncomfortable") to five ("extremely uncomfortable"), based on the estimated RMS acceleration.

CHAPTER III

VALIDATION OF MODEL

3.1 Introduction

Hansen's model, the precursor to RPM, was validated using experimental data gathered at the Department of National Defence's Land Engineering Test Establishment (LETE), Orleans, Ontario. This data consisted of vertical acceleration time histories measured at three corners of an M113A2 APC hull as the vehicle traversed a concrete sine wave course. Although the amplitudes predicted by Hansen's model were close to those experienced by the vehicle, the predicted acceleration time history was compressed in time when compared with the experimental data.

Originally, Hansen's model was going to be used as a benchmark against which the results from RPM could be compared. Unfortunately, there were several problems with this. First, although he reported the experimental vehicle to be an M113A2, his data file shows that he simulated an M113A1, which has radically different suspension characteristics. Second (and more significantly) his pitch and roll moments of inertia were in error by more than an order of magnitude. These discrepancies made it impossible to use his published results for comparison

purposes. Since a running version of his model is not available, it was not possible to generate new results to compare with RPM. Fortunately, VeyDyn II (which shares certain similarities with Hansen's model) was available, and was thus used as the benchmark program.

For validation of the model, experimental data was gathered at LETE giving the hull accelerations for an M113A1 APC traversing various courses at different speeds. RPM and VehDyn II were then used to simulate the experimental runs. The wider scope of the collected experimental and predicted data permitted a much more comprehensive analysis of the model's performance than Hansen was able to do.

3.2 The M113A1 Armoured Personnel Carrier

The M113 family of tracked vehicles has been in production since 1960. It is the most widely used APC among NATO nations, and is a standard test bed for tracked vehicle studies.

The M113 has five roadwheels in each track, with the track suspended between a drive sprocket at the front of the vehicle, and an idler wheel at the back. Rear sprockets are more efficient, but APCs must have the engine (and consequently the drive train) at the front of the vehicle to permit a rear ramp for disembarking troops. Since the M113 has no intermediate track support rollers

attached to the hull, the track lies on the last three roadwheels at rest, although it lifts off once tractive torque is applied at the sprocket.

Track tension is adjusted on the M113 by means of a grease cylinder attached to the idler. The idler wheel is not attached rigidly to the hull, but is mounted eccentrically on a second wheel, and is held in position by the grease cylinder on one side, and track tension on the other. By pumping grease into the cylinder the idler is forced back along an arc. This increases the peripheral track length required, which increases the track tension. The idler need only be moved a few centimetres back or forward to significantly affect track tension. If the track tension is too loose, the track may be thrown off the sprocket. If the tension is too tight, track pins may be bent, or the track may break.

The M113 has torsion bars at each wheel station which extend from one side of the vehicle to the other. This means that the roadwheels on the right side of the vehicle lie slightly behind the corresponding roadwheels on the left side, since the corresponding torsion bars cannot physically lie on the same axis. The right track has an extra track shoe to account for the resulting difference in peripheral lengths. Because terrain inputs at corresponding roadwheels on the left and right tracks are not in phase, vehicle roll can be induced by the offset of the torsion bars.

The A1 variant of the M113 has dampers mounted on the first and last roadwheels. In contrast, the M113A2 has an additional damper mounted on the

second roadwheel. The damping coefficients and mounting geometries of dampers on the two vehicles are significantly different.

Two different M113 data files have been created for the validation of RPM. The first is a pitch plane model (i.e. the vehicle is represented by a two-dimensional model; the mass and pitch mass moment of inertia of the vehicle have been halved to maintain similitude). The second is a more complex three dimensional model, used to illustrate the simulation of roll behaviour. In general the pitch plane model is to be preferred since it reduces computation time by a factor of two.

The M113A1 is shown schematically in Figure 24.

3.3 Test Terrain Profiles

Test runs were performed over three different terrain profiles: a concrete sinewave course (the same one used to generate Hansen's field data); a 6 inch (150 mm) half-round discrete obstacle; and LETE Course No. 48, a natural random roughness course. These three courses were used to subject the vehicle to the widest possible variety of terrain excitation regimes, as follows:

- a. steady-state excitation in the frequency range near the vehicle's natural frequencies (sinewave course);

Figure 24: M113 Armoured Personnel Carrier.

- b. transient excitation (half-round obstacle); and
- c. broad band, random excitation (LETE Course No. 48).

The sine wave course is a concrete track, 243 metres long, with a sinusoidal profile. The first half of the course has been constructed with the wave crests perpendicular to the direction of travel, in order to exercise the bounce and pitch response of traversing vehicles. The second half of the course has the wave crests skewed at an angle of -12° to the direction of travel, so as to excite vehicle roll (this part of the course was not used during the field tests). The width of the course is 3.66 metres (versus the vehicle width of 2.69 metres). The peak-to-peak amplitude of the waveform is approximately 0.14 metres, while the wavelength is 0.76 metres. The wave crests are level with the asphalt lead-ons to the course at both ends.

The 150 mm half-round obstacle is a semi-cylindrical obstacle constructed from a section of steel pipe cut longitudinally, and filled with concrete to resist collapsing. It lies anchored to the ground by four metre-long steel pins on a flat asphalt road which extends several hundred metres in either direction.

LETE Course 48 is a road of natural bedrock, 70 metres in length, created by stripping brush and soil from the surface to expose the underlying rock. The course profile is known from a laser theodolite survey that was conducted some time ago along two tracks separated by approximately 2.5 metres, at one foot intervals. The RMS values of the profiles, determined from the detrended survey data, are

56mm for the south track, and 59mm for the north track. Figure 25 shows the course profile for both tracks.

3.4 Methodology to Obtain Experimental Data

An M113A1 APC had been instrumented for a project being performed by LETE on behalf of the Defence Research Establishment Suffield (DRES). The instrumentation consisted, in part, of three Endevco piezoresistive accelerometers (type 2262C-25) mounted on the hull, three signal conditioners (Endevco type 4423) for the accelerometers, a rotary optical encoder mounted on the driver's-side track sprocket, and a MegaDAC digital acquisition system.

The hull accelerometers were mounted vertically on the outside of the vehicle roof, epoxied to bare aluminum. The mounting points were as follows (coordinates given with respect to the vehicle CG, in metres):

- a. in front of and to the left of the driver's hatch (1.39, 1.25, -1.21);
- b. underneath the left rear lifting point (-2.18, 1.39, -1.16); and
- c. underneath the right rear lifting point (-2.18, 1.39, 1.17).

The accelerometers used were rated from -25g to +25g, with frequency response of 0-650 Hz, $\pm 5\%$.

Figure 25: Profile of LETE Course 48.

The sprocket-mounted encoder, designed and constructed by DRES, was required to measure vehicle speed, since the roughness of the courses made the use of a fifth wheel impossible. The true vehicle speed will in general be less than the speed indicated by the encoder because of track slippage and uneven ground. The rotary optical encoder at the heart of the device generates 300 pulses per revolution, which means that even low vehicle speeds can (theoretically) be measured accurately. It was discovered during the course of the project that sprocket oscillations occur when the vehicle traverses uneven ground, due to fluctuations in track tension. These oscillations can falsely suggest that the vehicle's forward speed is undergoing rapid variation. Fortunately, by filtering out the higher frequencies the underlying trend in vehicle speed can be established.

The MegaDAC system installed in the crew compartment was configured to filter the accelerometer signals above 30 Hz, and to poll each signal channel 150 times per second (a signal channel is the output from any transducer fed as input to the MegaDAC). Digitization was accomplished with 16 bit coding, which corresponds to a maximum quantization error of 0.0015% (i.e. there was negligible A/D error). Filtering was performed on the analog signal before digitization. The digitized signal was stored on a magnetic tape cartridge for later analysis.

3.5 Comparison of Experimental and Predicted Results

3.5.1 Concrete Sine Wave Traverse

Figures 26 through 33 show various results that were obtained experimentally and through simulation for the APC traversing the concrete sine wave at a nominal

Figure 26: Experimental vs. predicted acceleration.
Sine wave, 8 km/h, hull front, low damping RPM.

Figure 27: Experimental vs. predicted acceleration.
Sine wave, 8 km/h, hull front, nominal damping RPM.

Figure 28: Experimental vs. predicted acceleration.
Sine wave, 8 km/h, hull front, high damping RPM.

Figure 29: Experimental vs. predicted acceleration.
Sine wave, 8 km/h, hull front, low damping VehDyn II.

Figure 30: Experimental vs. predicted acceleration.
Sine wave, 8 km/h, hull rear, low damping RPM.

Figure 31: Experimental vs. predicted acceleration.
Sine wave, 8 km/h, hull rear, nominal damping RPM.

Figure 32: Experimental vs. predicted acceleration.
Sine wave, 8 km/h, hull rear, high damping RPM.

Figure 33: Experimental vs. predicted acceleration.
Sine wave, 8 km/h, hull rear, low damping VehDyn II.

forward speed of 8 km/h. Figures 26 through 29 depict the vertical acceleration at the front of the hull, while Figures 30 through 33 show the corresponding acceleration-time histories for the rear of the hull. A two-second period in the middle of the traverse was arbitrarily chosen as being representative of the entire test period, while showing enough detail to allow comparison of the experimental and predicted data. The measured and computed time histories were aligned at $t = 0$ (i.e. the point at which the vehicle entered the course) to give each set of data the same time axis.

Figures 26 through 28 compare the time history predicted by RPM with the measured time history for low, nominal, and high damping rates. The low damping rate (38 180 Ns/m) and the high damping rate (60 950 Ns/m) represent the damping coefficient of the dampers at working and ambient temperatures respectively [36], while the nominal rate is simply the arithmetic average of the first two rates. Figure 29 combines the time history of the measured data with that predicted by VehDyn II for a low damping rate.

Figures 30 through 33 give the same information as Figures 26 through 29, except for the rear of the hull, as previously mentioned.

Table II compares the ride quality associated with the actual and simulated traverses. The average absorbed powers calculated by RPM at the driver's seat for the experimental data, for the data generated by RPM at all three damping rates, and for the data predicted by VehDyn II for the low damping rate are given, as well

Table II: Measured and Predicted Ride Quality of APC Over Sine Wave

(forward speed = 8 km/h)

	FIELD TEST	RPM LOW DAMPING	RPM NOMINAL DAMPING	RPM HIGH DAMPING	VEHDYN II RPM GENERATED	VEHDYN II SELF GENERATED
AVG ABSORBED POWER (W)	8.8	9.2 (4.5%)	20.2 (130%)	50.3 (472%)	17.~ (93%)	17.0 (93%)
ISO 2631 FREQUENCY WEIGHTED RMS ACCELERATION (m/s ²)	1.5	1.5 (0.0%)	2.2 (47%)	3.6 (140%)	2.0 (33%)	N/A
GRIFFIN'S VIBRATION DOSE VALUE (m/s ^{1.75})	6.5	6.0 (-7.7%)	9.4 (45%)	14.3 (120%)	7.5 (15%)	N/A

Note: Numbers in brackets are percentage errors with respect to the measured results.

as the absorbed power computed independently by VehDyn II. VehDyn II computes average absorbed power using a time-domain algorithm based upon equation 2, and shown as the self-generated results in the ride quality tables presented here. In contrast, RPM performs the calculations in the frequency domain, using equation 3.

In addition, the ISO 2631 frequency-weighted, vertical RMS accelerations, and predicted VDV's as calculated by RPM for the same data sets, are given in Table II.

3.5.2 Half-Round Obstacle Crossing

Figures 34 through 41 compare the experimentally measured time histories of the APC crossing a 150mm half- round obstacle with simulated time histories from RPM and VehDyn II. The time axes have been shifted so that the initial bump impacts are aligned. Figures 34 and 35 compare the field data with results from RPM for an obstacle crossing speed of 16 km/h (Figure 34 shows the data at the front of the hull, Figure 35 for the rear). For clarity, only the low and high damping rate RPM scenarios are shown. There is a monotonic relationship between damping rate and dynamic response, so the nominal damping rate response can be approximated by imagining a curve lying between the high and low damping rate curves. Note that the left ordinal axis corresponds to the field data, while the right ordinal axis is associated with the simulation results. Similarly, Figures 36 and 37 show the results for a crossing speed of 24 km/h, and Figures 38 and 39 graph the

Figure 34: Experimental vs. predicted acceleration.
150 mm bump, hull front, 16 km/h, RPM.

Figure 35: Experimental vs. predicted acceleration.
150 mm bump, hull rear, 16 km/h, RPM.

Figure 36: Experimental vs. predicted acceleration.
150 mm bump, hull front, 24 km/h, RPM.

Figure 37: Experimental vs. predicted acceleration.
150 mm bump, hull rear, 24 km/h, RPM.

Figure 38: Experimental vs. predicted acceleration.
150 mm bump, hull front, 32 km/h, RPM.

Figure 39: Experimental vs. predicted acceleration.
150 mm bump, hull rear, 32 km/h, RPM.

Figure 40: Experimental vs. predicted acceleration.
150 mm bump, hull front, 24 km/h, VehDyn II.

Figure 41: Experimental vs. predicted acceleration.
150 mm bump, hull rear, 24 km/h, VehDyn II.

data for a vehicle speed of 32 km/h. For the sake of brevity, VehDyn data is only plotted for a crossing speed of 24 km/h (Figures 40 and 41).

Tables III, IV, and V tabulate calculated ride quality for both experimental and computed data sets, at vehicle crossing speeds of 16 km/h, 24 km/h, and 32 km/h respectively. Results for RPM simulations using nominal damping are included.

3.5.3 LETE Course 48 Traverse

Figures 42 through 47 show measured and computed acceleration data for the M113A1 APC traversing LETE Course 48 at a nominal speed of 8 km/h. The data has been aligned at $t=0$ such that both the experimental and simulated data share the same time axis. The first three figures (42 to 44) are concerned with vertical acceleration at the front of the hull, while the next three (45 to 47) give results for the rear of the hull. The order of presentation is:

- a. field data vs. low damping, full-vehicle (i.e. 3 DOF) RPM scenario;
- b. field data vs. low damping, half-vehicle (i.e. 2 DOF) RPM scenario; and
- c. field data vs. low damping, half-vehicle VehDyn II scenario.

Table III: Measured and Predicted Ride Quality of APC Over 150mm Bump

(forward speed = 16 km/h)

	FIELD TEST	RPM LOW DAMPING	RPM NOMINAL DAMPING	RPM HIGH DAMPING	VEHDYN II RPM GENERATED	VEHDYN II SELF GENERATED
AVG ABSORBED POWER (W)	3.4	0.3 (-91%)	0.2 (-94%)	0.3 (-91%)	6.6 (94%)	6.9 (103%)
ISO 2631 FREQUENCY WEIGHTED RMS ACCELERATION (m/s ²)	4.0	2.3 (-43%)	1.5 (-63%)	1.4 (-65%)	3.8 (-5.0%)	N/A
GRIFFIN'S VIBRATION DOSE VALUE (m/s ^{1.75})	3.4	1.1 (-68%)	1.2 (-65%)	1.8 (-47%)	4.1 (21%)	N/A

Note: Numbers in brackets are percentage errors with respect to the measured results.

Table IV: Measured and Predicted Ride Quality of APC Over 150mm Bump

(forward speed = 24 km/h)

	FIELD TEST	RPM LOW DAMPING	RPM NOMINAL DAMPING	RPM HIGH DAMPING	VEHDYN II RPM GENERATED	VEHDYN II SELF GENERATED
AVG ABSORBED POWER (W)	4.0	0.8 (-80%)	0.6 (-85%)	0.6 (-85%)	5.4 (35%)	5.2 (31%)
ISO 2631 FREQUENCY WEIGHTED RMS ACCELERATION (m/s ²)	5.1	5.5 (7.8%)	4.4 (-14%)	3.8 (-25%)	7.4 (45%)	N/A
GRIFFIN'S VIBRATION DOSE VALUE (m/s ^{1.75})	3.9	1.8 (-54%)	1.6 (-59%)	1.7 (-56%)	4.2 (7.7%)	N/A

Note: Numbers in brackets are percentage errors with respect to the measured results.

Table V: Measured and Predicted Ride Quality of APC Over 150mm Bump

(forward speed = 32 km/h)

	FIELD TEST	RPM LOW DAMPING	RPM NOMINAL DAMPING	RPM HIGH DAMPING	VEHDYN II RPM GENERATED	VEHDYN II SELF GENERATED
AVG ABSORBED POWER (W)	7.0	2.2 (-69%)	2.2 (-69%)	2.3 (-67%)	4.7 (-33%)	4.9 (-31%)
ISO 2631 FREQUENCY WEIGHTED RMS ACCELERATION (m/s ²)	12.5	7.8 (-38%)	6.2 (-50%)	5.5 (-56%)	7.4 (-41%)	N/A
GRIFFIN'S VIBRATION DOSE VALUE (m/s ^{1.75})	4.3	2.7 (-37%)	2.5 (-42%)	2.5 (-42%)	4.3 (0.0%)	N/A

Note: Numbers in brackets are percentage errors with respect to the measured results.

Figure 42: Experimental vs. predicted acceleration.
 LETE 48, 8 km/h, hull front, low damping 3-D RPM.

Figure 43: Experimental vs. predicted acceleration.
 LETE 48, 8 km/h, hull front, low damping 2-D RPM.

Figure 44: Experimental vs. predicted acceleration.
 LETE 48, 8 km/h, hull front, low damping VehDyn II.

Figure 45: Experimental vs. predicted acceleration.
 LETE 48, 8 km/h, hull rear, low damping 3-D RPM.

Figure 46: Experimental vs. predicted acceleration.
 LETE 48, 8 km/h, hull rear, low damping 2-D RPM.

Figure 47: Experimental vs. predicted acceleration.
 LETE 48, 8 km/h, hull rear, low damping VehDyn II.

RPM data for the nominal and high damping rate scenarios has not been plotted. Again, the left ordinal axis refers to the field data, and the right ordinal axis to the simulated data.

Table VI gives ride quality assessments for the field data, for the three (low, nominal, and high damping) RPM simulations using a full-vehicle model, and for the VehDyn II simulation using a half-vehicle model. Table VII gives the ride quality of three RPM time histories for a half-vehicle model.

3.6 Discussion of Experimental and Predicted Results

3.6.1 APC on the Sine Wave

The RPM prediction of accelerations at the front and rear of the hull during the sine wave traverse agree well with the experimental values. At the front of the hull (Figures 26, 27, and 28) the acceleration waveform is characterized by a sharp positive peak occurring with a period of about 0.35 seconds. At a speed of 8 km/h, this represents a spatial displacement of about 0.77 metres - approximately the wavelength of the sine wave course. It is apparent, therefore, that these acceleration spikes are generated as the front roadwheel hits each new crest. Smaller peaks are created as the other roadwheels cross each crest. The numbers

Table VI: Measured and Predicted (3-D RPM) Ride Quality of APC Over LETE Course 48

(forward speed = 8 km/h)

	FIELD TEST	RPM LOW DAMPING	RPM NOMINAL DAMPING	RPM HIGH DAMPING	VEHDYN II RPM GENERATED	VEHDYN II SELF GENERATED
AVG ABSORBED POWER (W)	1.9	8.2 (332%)	8.8 (363%)	15.1 (695%)	1.8 (-5.3%)	1.5 (-19%)
ISO 2631 FREQUENCY WEIGHTED RMS ACCELERATION (m/s ²)	0.8	2.5 (213%)	~ 7 (238%)	3.7 (363%)	0.7 (-13%)	N/A
GRIFFIN'S VIBRATION DOSE VALUE (m/s ^{1.75})	4.4	17.6 (300%)	18.9 (330%)	10.4 (136%)	14.2 (-222%)	N/A

Note: Numbers in brackets are percentage errors with respect to the measured results.

Table VII: Measured and Predicted (2-D RPM) Ride Quality of APC Over LETE Course 48

(forward speed = 8 km/h)

	FIELD TEST	RPM LOW DAMPING	RPM NOMINAL DAMPING	RPM HIGH DAMPING	VEHDYN II RPM GENERATED	VEHDYN II SELF GENERATED
AVG ABSORBED POWER (W)	1.9	1.6 (-16%)	2.6 (37%)	4.3 (126%)	1.8 (-5.3%)	1.5 (-19%)
ISO 2631 FREQUENCY WEIGHTED RMS ACCELERATION (m/s^2)	0.8	1.0 (25%)	1.1 (38%)	1.4 (75%)	0.7 (-13%)	N/A
GRIFFIN'S VIBRATION DOSE VALUE ($m/s^{1.75}$)	4.4	11.2 (155%)	16.4 (273%)	18.9 (330%)	14.2 (-222%)	N/A

Note: Numbers in brackets are percentage errors with respect to the measured results.

annotating Figure 26 indicate which roadwheel is creating which peak. These smaller peaks may be seen in the RPM-generated waveform, although less distinctly.

The predicted magnitudes and phases are most accurate for the low-damping scenario (Figure 26). As the damping rate increases, the magnitude of the waveform increases (Figures 27 and 28). These trends can also be seen at the rear of the hull (Figures 30, 31, and 32). The good fit between experimental and predicted data associated with the low damping rate is appropriate, since it would seem probable that after several runs over the course the dampers would be well exercised, and warmed to operating temperature.

If the vehicle is idealized as a one degree of freedom system excited by a moving base, the phenomenon of increasing acceleration with increasing damping rate does not seem unreasonable. The natural frequency of the vehicle in heave is approximately 1.5 Hz, while the temporal frequency of the sine wave course at 8 km/h is almost 3 Hz. The frequency ratio is thus about 2. At frequency ratios greater than 1.414, the magnitude of the force transmitted to the system mass is known to increase with damping rate, for this type of idealized system.

Figure 29 shows that VehDyn II failed to properly predict the acceleration experienced at the front of the hull. Specifically, the predicted waveform is symmetrical about the x axis, whereas the experimental waveform is skewed to the positive. This probably reflects the inadequacies of the track spring model used by VehDyn II, which does not properly account for the downward force exerted on the

hull at the sprocket and idler by the track at high tensions. Further, the predicted time history does not seem to reflect the motion of the intermediate roadwheels over the crests (i.e. there are no minor peaks in the waveform). From Figure 33, it can be seen that VehDyn II was more successful at predicting the rear hull acceleration, although it consistently overestimates the peak values.

Table II shows excellent agreement between the ride quality parameters computed for the experimental data and the RPM data (low damping rate scenario), the largest error being -7.7%. At higher damping rates, the agreement rapidly drops off - not surprisingly given the poor agreement between the waveforms. VehDyn II compares unfavourably with RPM with regard to the ride quality assessment, giving much poorer predictions of ride quality. Note that the two absorbed power computations for the VehDyn II data, independently generated using different methods (in the time domain by VehDyn II, and in the frequency domain by RPM), are in agreement, validating the absorbed power subroutine of RIDE-Q.EXE.

3.6.2 APC over the Half-Round Obstacle

With reference to Figures 34 through 39, there is considerable noise on the experimental signal compared with the simulated data. This high-frequency noise is largely attributable to track-induced vibration. As was noted in Chapter I, a high frequency vibration component is generated by the impact of individual track shoes on the ground as the vehicle moves forward. This is particularly noticeable on hard terrain, such as the asphalt course that was traversed for this field test. The peak

acceleration of this track noise is about 2 m/s^2 , so it is not insignificant (Figure 48). Because RPM models the track as being continuous, this noise cannot be simulated.

For these test runs, RPM consistently underestimates the acceleration level. With regards to accelerations at the front of the vehicle, the high damping rate scenario is closest to approximating the initial impact peak as the vehicle hits the bump, but is not so successful at predicting subsequent acceleration levels. On the other hand, the low damping rate scenario severely underestimates the initial peak, but is better at simulating ensuing acceleration levels. At the rear of the vehicle, the accelerations are better predicted using the high damping vehicle model. The better fit afforded by the high damping rate scenario probably reflects the fact that, unlike the sine wave test run, the vehicle was either stationary or travelling across smooth asphalt (except for the crossing of the obstacle itself), and consequently the dampers did not have an opportunity to warm to operating temperature. The underestimation of the acceleration levels could be caused by underestimating the true suspension stiffness [8].

The second peak in the experimental data at the rear of the hull (Figures 35 and 37) has been completely truncated by the RPM simulation. It is significant that this peak exists in the 16 and 24 km/h time histories, but is absent from the 32 km/h data (Figure 39). In fact, this peak is generated by the rear track idler striking the bump as the vehicle passes over, as has been described in other studies [38]. At 32 km/h, the vehicle apparently clears the bump before interference between the bump and idler can occur. RPM does not model impact forces between the ground

Figure 48: Experimentally measured track noise.
Level asphalt road, 8 km/h, hull front.

and the vehicle sprung mass, and consequently cannot predict these impacts. In contrast, VehDyn II requires spridler stiffnesses as input, in order to compute these impact forces. It can be seen in Figure 41 that the second peak generated by idler impact is predicted by VehDyn II. However, VehDyn II clearly overpredicts trough amplitudes, probably a reflection of the limitations of the track-spring model.

It has been noted in experimental trials that when a tracked vehicle hits a bump, it undergoes deceleration [39]. However, the driveline is compliantly coupled to the ground through the flexible track. Therefore, the sprocket continues to rotate at approximately the same rate, with the effect that more track is reeled in than is paid out. A rear-sprocket vehicle will squat as the sprocket pulls itself and the back of the hull down. conversely, a front-sprocket vehicle will dive as the sprocket tries to flip the vehicle forward about itself. Because a constant velocity model, ipso facto, will not simulate this sort of deceleration, the motion cannot be replicated. The simulated vehicle will not, therefore, be subjected to the same forcing function as the actual vehicle, and cannot give accurate predictions. For example, it is evident that if the vehicle dives under the combined effect of deceleration and driveline interaction, a greater front-hull vertical acceleration might be expected than that predicted by a constant velocity model. This is what occurs in Figures 36 and 37.

The driveline interaction peculiar to a vehicle decelerated by hitting an obstacle is also responsible for pitch damping. Since the drive sprocket, coupled to the driveline, is not free to turn, vehicle pitching causes the sprocket to turn with

respect to the track. This action is resisted by the torsional stiffness of the track bushings [13, 39]. Since neither track damping properties nor driveline mechanics are modelled by RPM, there is some error in the predicted pitch response of the model (i.e. it must be underdamped), which will again have an impact on the forcing function seen by the vehicle.

With regard to Tables III, IV, and V, the absorbed power and ISO acceleration values are not valid as the crest factors at the driver's seat for the three sets of field data were 5.7, 7.0, and 5.1, respectively. Although ISO 2631 can be used with crest factors as high as 6.0, it is advisable to use it only for crest factors lower than 3.0. The VDV's for the RPM data are relatively poor (errors of between -42% to -46% for the high damping rate scenario), which is to be expected given the underestimation of the acceleration levels, although the VDV's for the experimental data are probably elevated somewhat due to the inclusion of track noise. VehDyn II appears to give better predictions, but this is clearly coincidental given the poor agreement between the waveforms it predicted and the field data.

3.6.3 APC on LETE Course 48

The computer generated predictions for LETE Course 48 were the least satisfactory of those completed. This reflects less on the adequacies of the computer models than on the difficulty of obtaining usable field data for model validation. Figure 49 shows the actual vehicle speed obtained from the sprocket-mounted encoder as the vehicle traversed the course. Although nominally 8 km/h,

Figure 49: Vehicle speed-LETE Course 48
Experimental

the speed varied between 5 km/h and 11 km/h. This variation in speed was due to the difficulty in maintaining correct accelerator pedal pressure (due to foot bounce), and to the natural inclination of the driver to reduce speed over the roughest parts of the course. The effect of this can be seen in Figure 50, which compares the position of the actual vehicle with the position of a vehicle moving at a constant forward speed of 8 km/h. For example, at 10 seconds into the traverse, the real vehicle was 0.8 seconds behind where it ideally would have been. This chart could be used in conjunction with Figures 43 through 48 to compare features on the field and simulated histories. Therefore, a feature occurring at 10 seconds in the field data should occur at 10.8 seconds in the model data.

Unfortunately, comparison between the real and simulated data is further confounded by the fact that if the vehicles do not interact with terrain features at the same time, the attitude (heave, pitch, roll) of the vehicles at that point on the terrain may differ, and consequently the accelerations on the hulls will differ. This is to say that comparison between the vehicle accelerations is invalid unless both vehicles share the same forcing function, which is clearly not the case if the vehicles are travelling at different speeds. This also suggests that simulation using a half-vehicle model on a terrain having separate profiles on each side of the vehicle is invalid.

A further difficulty in the case of the data for LETE Course 48 was that the vehicle underwent considerable yawing at certain points on the course. Aside from displacing the vehicle from the known track profiles, yawing can generate hull

Figure 50: Time shift of field data with respect to simulation data.

accelerations through centripetal effects that cannot be simulated with a 3 DOF model.

The poor agreement of the data in Tables VI and VII reflects these problems. The crest factor at the driver's seat for the experimental accelerations was 8.8, which invalidates the absorbed power and ISO 2631 ride parameters. The VehDyn II data and the 2-D RPM data are invalid because they cannot represent the 3-D nature of the profile.

CHAPTER IV

CONCLUSIONS AND RECOMMENDATIONS

4.1 Conclusions

RPM generated accurate results for the traverse of the sine wave course, which validated its response under steady state conditions. The ride quality parameters computed for the simulation results were in close agreement to those found for the experimental data. The RPM results were superior to those generated by VehDyn II, reflecting a superior track model (catenary vs. spring), and more accurate suspension kinematics.

The RPM data computed for the traverse of the half-round obstacle was less satisfactory. Although the general shape of the transient waveform was captured, the accelerations were underestimated. This may reflect an underestimation of the suspension stiffness. Because RPM does not consider spridler impact forces, an important acceleration peak at the rear of the vehicle was missed at the two lower vehicle speeds. Although VehDyn II simulated these impact forces, it was less successful than RPM at producing an accurately shaped waveform. The inclusion of VDV as a ride quality parameter was useful, given the fact that the high crest factors

encountered reduced the usefulness of the absorbed power and ISO 2631 parameters.

The RPM results for the random roughness course were inconclusive due mostly to high variability in traverse speed of the actual test vehicle. VehDyn II is unsuitable for predicting vehicle response on this kind of terrain as it is a half-vehicle model. Again, due to high crest factors, the VDV was the most useful ride quality parameter.

From the runs over three different terrain profiles, it can be concluded that a constant velocity model will give best results under steady-state conditions, for which the assumption of constant velocity is most valid.

There are two other factors that effect the accuracy of all simulations, which should be noted in passing. First, it is often difficult to accurately measure vehicle properties, especially inertial properties, and this may have a very real effect on the accuracy of the simulation [7, 40]. Second, the generation of acceleration data based upon vehicle displacement (i.e. a second order derivative) is very susceptible to error [7, 38]. Displacement predictions from simulation models are often very good, but ride quality assessment is based on acceleration, not displacement.

4.2 Recommendations

RPM should prove to be a useful starting point for future research. Of particular value are the subroutines which model suspension kinematics in a realistic fashion, which replace the classic catenary equations with explicit pseudo-catenary formulae, which evaluate wheel forces with horizontal forces and track tension considered, and which assess ride quality using three different and useful criteria, with frequency domain methods. The two most common suspension configurations (pillar and trailing arm) seen on heavy off-road military vehicles are modelled by RPM.

Worthwhile future enhancements of RPM would include the following:

- a. the capacity to model rotary dampers;
- b. the creation of a library of formulae to automatically generate hubpath data for discontinuous obstacles (such formulae are, for example, available for trapezoidal berms and trenches [41]);
- c. the introduction of aerodynamic drag coefficient and vehicle frontal area that can be changed by the user, instead of being fixed by the program;
- d. the ability to run a full-vehicle data file in either half or full-vehicle mode, without requiring the user to create separate half and full-vehicle data files;

- e. the consideration of spridler impact forces; and
- f. the ability to use a variable speed profile in conjunction with the terrain profile (this would greatly increase the utility of test data gathered in the real world).

One area which needs to be studied with greater detail is that of track modelling. As has been stated, the catenary model is currently the best choice for simulating both mass-transfer effects and variable suspension stiffness. However, it has also been noted that the track does not hang in a quasi-static catenary much of the time. In addition, real tracks exhibit torsional stiffness and hysteretic damping. How well does the catenary (and the pseudo-catenary) model these effects? It would be extremely worthwhile to run validation tests, perhaps with an instrumented track shoe that would permit the measurement of track tension while crossing obstacles, etc. Some data of this sort exists in the literature [14], and could be used if one could obtain the associated vehicle parameters to make it useful.

The biggest obstacle to a model becoming a useful tool is the establishment of the user's faith in the results. With these suggested enhancements, and with greater investigation and validation of track effects, the Ride Prediction Model should become an efficient and credible aid to the vehicle designer or procurement engineer.

REFERENCES

1. Bryson, R.A. 1988. Quantifying battle tank mobility - a manufacturer's view. *J. of Terramechanics*. 25,1: 56-67.
2. Doyle, G.R. Jr., and G.H. Workman. 1979. Prediction of track tension when traversing an obstacle. SAE paper 790416.
3. Hoppe, C.H. 1968. Design for the rough terrain environment. SAE paper 680098.
4. Haley, P.W., et al. NATO Reference Mobility Model, Edition I: User's Guide, Vol. I (Technical Report No. 12503). U.S. Army Tank-Automotive Research and Development Command, Warren, Michigan. October 1979.
5. Creighton, D.C. Revised Vehicle Dynamics Module: User's Guide for Computer Program VehDyn II (Technical Report SL-86-). U.S. Army Waterways Experiment Station, Vicksburg Mississippi. April 1986.
6. Hansen, G.H. "Three degrees of freedom dynamic computer model for tracked vehicles." M.Eng. Thesis, Carleton University, 1986.

7. Bennett, M.D., and P.H.G. Penny. Dynamic simulation of track laying vehicles. ISTVS 8th Int. Conf. on the Performance of Off-Road Vehicles and Machines. Cambridge: ISTVS, 1984.
8. Craighead, I.A., et al. 1986. Validation tests on computer simulation programs modelling off-road vehicles. J. of Terramechanics. 23,1: 13-22.
9. Briggs, P.G. Technical proposal for a tandem hydropneumatic suspension system for M109 Howitzer and variants. England : Airlog, 1988.
10. Trackpower Transmissions Ltd. Trackpower M113 retrofit technology demonstrator - preliminary technical information. Coventry, England : Trackpower Transmissions Ltd., 1988.
11. Ellis, J.R. Modelling and Simulation of Handling Characteristics, in Lecture Notes for "Braking and Handling of Heavy Commercial Vehicles", Carleton University, Feb 21-25, 1977.
12. United States of America. U.S. Army. TOP 1-1-050: Vibration Testing. 4 March 1974.
13. MacLaurin, E.B. et al. Using a computer model to determine preferred characteristics for the suspensions of tracked vehicles. ISTVS 9th Int. Conf., Barcelona, Spain: ISTVS, 1987.

14. Galaitsis, A.G. 1984. TRAXION: A model for predicting dynamic track loads in military vehicles. *J. of Vibration, Acoustics, Stress, and Reliability in Design.* 106: 286-291.
15. Wong, J.Y. Theory of Ground Vehicles. John Wiley & Sons, New York. 1978.
16. Department of National Defence. APC Track Trials (High speed video), Tape E-69. Produced by Land Engineering Test Establishment, Orleans, Ont.
17. Black, S.H., and D.L. Taylor. 1979. Simulation of off-road motorcycle ride dynamics. SAE paper 790261.
18. Fukushima, N., et al. 1983. Optimum characteristics of automotive shock absorbers under various driving conditions and road surfaces. *Int. J. of Vehicle Design.* 4,5: 463-472.
19. Segel, L., and X-P Lu. 1982. Vehicular resistance to motion as influenced by road roughness and highway alignment. *Australian Road Res.* 12,4: 211-221.
20. Dzielski, J.E., and J.K. Hedrick. 1984. Energy dissipation due to vehicle/track interaction. *Vehicle Systems Dynamics.* 13 (1984): 315-337.

21. Browne, A.L., and J.A. Hamburg. GMR-5103: On-Road Measurement of the Energy Dissipated in Automotive Shock Absorbers. Detroit, MI: General Motors Research, 1985.
22. Jokhadze, G.D. 1988. The efficient use of vehicles with pneumatic suspensions in bad road conditions. *J. of Terramechanics*. 25,3: 223-229.
23. Siorek, R.W. 1973. Experimental investigation of effect of wheel travel on tracked vehicle mobility. SAE paper 730036.
24. Newton, K., et al. The Motor Vehicle (9th edition). Iliffe Books, London. 1972.
25. Hillier, V.A., and F.W. Pittuck. Fundamentals of Motor Vehicle Technology (2nd edition). Hutchinson Educational Ltd., London. 1972.
26. Stikeleather, L.F., et al. 1972. A study of vehicle vibration spectra as related to seating dynamics. SAE paper 720001.
27. International Organization for Standardization. ISO 2631/1: Evaluation of human exposure to whole-body vibration - Part 1: General requirements. 15 May 1985.

28. Janeway, R.N. 1966. A practical approach to truck ride instrumentation and evaluation - a summary. SAE paper 660140.
29. Lee, R.A., and F. Pradko. 1968. Analytical analysis of human vibration. SAE paper 680091.
30. Pradko, F., et al. 1965. Human vibration analysis. SAE paper 650426.
31. Griffin, M.J. Vibration dose values for whole body vibration: some examples. Conference on Human Response to Vibration. Edinburgh: Herriot-Watt University, 1984.
32. United Kingdom. Royal Military College of Science. General Design / Vehicle Performance (M.Sc. Course, Military Vehicle Technology). Shrivenham, England, 1982.
33. McCullough, M.K., and E.J. Haug. Terra-Dynamics of High Mobility Track Vehicles (Technical Report No. 85-22). Center for Computer Aided Design, University of Iowa, Iowa City, Iowa. December 1985.
34. Salisbury, J.K., ed. Kent's Mechanical Engineers' Handbook: Power Volume (12th edition). John Wiley & Sons, New York. 1957.

35. Brigham, E.O. The Fast Fourier Transform. Prentice Hall, Inc., Englewood Cliffs, New Jersey. 1974.
36. Norrish, M.E. (FMC Corp.). Telex to DCMEM 2-2 (Department of National Defence). March 13, 1981.
37. Butkunas, A.A. 1969. Random vibration analysis and vehicle development. SAE paper 690109.
38. Hunt, J.A. "A scale model tracked vehicle for computer simulation validation." M.Eng. Project, Royal Military College of Science, Shrivenham, England, 1985.
39. Wheeler, P. 1977. Tracked vehicle ride dynamics computer program. SAE paper 770048.
40. Galway, M.C. Definition of Requirement for LETE Vehicle Inertial Properties Test Facility (Internal Report No. 3/89). Orleans, Ont.: Department of National Defence, 1989.
41. Morgan, T.W. "A dynamic vehicle simulation model to predict obstacle clearance." M.Eng. Thesis, Royal Military College of Canada, Kingston, Ontario, 1984.

APPENDIX 1

USER'S GUIDE TO RPM

USER'S GUIDE TO RPM

Introduction

With reference to Figure 8, the Ride Prediction Model comprises four executable files (TERPRED, STATIC, RPM, and RIDE-Q), which must be run in the indicated order (with the exception of TERPRED and STATIC which are mutually independent, and can thus be run in either order). Four data files must be supplied by the user: a control file, a run-time data file, a terrain data file, and a vehicle data file. These are described more fully in the following sections. The program generates seven output files, six of which are of interest to the user, and four of which may be suppressed if not required.

The User-Provided Control File (META.CNT)

The purpose of this file is to communicate to the program the names and resident directories of user provided input files, and the desired names and directories of program generated output files. This gives the user maximum flexibility in managing the flow of data, and can be exploited when writing batch processes to run the program.

In order that the program can find the control file, it must be given the name META.CNT. Furthermore, it must exist in the current directory at run-time.

Figure 51 shows a typical META.CNT file. The format of the file is as follows:

Line 1: User's descriptive title of the program run (optional - leave blank if not required);

Line 2: name and path of user-provided vehicle data file;

Line 3: name and path of user-provided run-time data file;

Line 4: name and path of user-provided terrain data file;

Line 5: name and path of TERPRED-generated hubpath data file;

Line 6: name and path of RPM-generated time-history data file;

Line 7: name and path of RIDE-Q-generated FFT of acceleration time-history (enter DEFAULT to suppress creation);

Line 8: name and path of RIDE-Q-generated FFT of acceleration time-history, weighted according to the absorbed power criterion (enter DEFAULT to suppress creation);

Line 9: name and path of RIDE-Q-generated acceleration time-history, weighted according to the ISO 2631 criterion (enter DEFAULT to suppress creation);

```
M113A1 @ 8 km/h OVER SINE WAVE, LOW DAMPING
C:\TPASCAL\VEH\M113-A1.DLO
C:\TPASCAL\DATA\APCDATA8.DAT
C:\TPASCAL\TERR\SINE.DAT
C:\TPASCAL\TERR\M113SINE.DAT
D:\RUN\SINE-DLO.DAT
DEFAULT
DEFAULT
DEFAULT
DEFAULT
D:\RUN\SINE-DLO.REP
```

Figure 51: Contents of typical META.CNT file.

Line 10: name and path of RIDE-Q-generated acceleration time-history, weighted according to the Griffin's criterion (enter DEFAULT to suppress creation); and

Line 11: name and path of RIDE-Q-generated ride quality report.

If an error code of 2 or 3 is generated by the program at run-time, it indicates a problem with the META.CNT file. Error code 2 is generated if a file does not exist, while error code 3 is produced if an attempt is made to access a non-existent path. Check the following:

- a. META.CNT must exist in the current directory;**
- b. when running any of the programs, the user-provided data files must exist in the paths and with the names specified in META.CNT;**
- c. if running RPM.EXE, the TERPRED-generated hubpath data file must exist as specified in META.CNT; and**
- d. if running RIDE-Q.EXE, the RPM-generated time-history data file must exist as specified in META.CNT.**

The User-Provided Run-Time Data File

The purpose of this file is to provide various parameters required by the program, while allowing the user to modify the parameters. A typical run-time data file is shown in Figure 52. The format of the file is as follows:

Line 1: accuracy, maximum number of iterations (integer), integration stepsize, output interval (integer), vehicle speed (m/s), auto quit (integer);

Line 2: driver's seat coordinates (x, y, z), in metres from C.G.;

Line 3: driver's feet coordinates (x, y, z), in metres from C.G.;

Line 4: driver's back coordinates (x, y, z), in metres from C.G.; and

Line 5: seating orientation (integer).

The accuracy and maximum number of iterations are parameters required by iterative search procedures used in STATIC and RPM. The former specifies how close a converging search must come to the unknown root (true value zero) before terminating the search successfully. The latter dictates how many trials should be attempted before the search terminates unsuccessfully.

The integration stepsize and print interval determine the time increment between solutions of the equations of motion, and the time interval between writing

```
0.00001 100 0.0016667 4 4.4704 0
1.000 0.210 -0.991
1.000 0.210 -0.991
1.000 0.210 -0.991
0
```

Figure 52: Contents of typical run-time data file.

discrete solutions to the time-history data file. A print interval of two indicates that the every second solution will be captured.

The auto quit is a feature intended to allow the user to run the program either as a batch or as a single process. If the former, auto quit should be entered as zero. This will cause the program to terminate at the end of the computations, and permit the next line of a batch file to be executed. In contrast, if the value is set to some other value than zero, the program will wait for the user to press a key before terminating. This keeps the terminal values of the program on the screen if the user wishes to inspect them.

The coordinates do not necessarily have to represent the coordinates of the driver's seat. They may, for example, indicate a point in a cargo or passenger area to generate ride quality for these areas. If ride quality data associated with either the feet or back is not required, the user can set the coordinates at these points equal to those of the seat, and thereby suppress calculations for these inputs. The seating orientation should be set equal to zero for passengers facing forward, and to one for passengers facing sideways.

The User-Provided Terrain Data File

This file gives TERPRED information about the terrain profile. In fact, up to four terrain profiles may be entered. This is useful if, for example, a full-vehicle model is being used, with the wheel spacing at the front axle different than that at

the rear (i.e. the vehicle produces four wheel ruts). Typical terrain data files can be examined in Appendix 3. The general format is as follows:

Line 1: descriptive title (optional - leave blank if not required);

Line 2: number of track profiles (four maximum);

Line 3: horizontal displacement (m), vertical displacements (m) of each track profile; and

Line n: horizontal displacement (m), vertical displacements (m) of each track profile.

There may be a maximum of 1701 data points (i.e. the file may have 1703 lines maximum).

The User-Provided Vehicle Data File

This file encapsulates all the geometric, mechanical, and inertial properties of the vehicle required for dynamic simulation. Because of the length of this data file, the lines are coded to allow quick interpretation of the data by the user.

Therefore, the first seven characters of each line are reserved for the line code, and the user should not input data in these columns. Several examples of vehicle data files may be found in Appendix 2. The format is as follows:

Line 1: descriptive title (optional - leave blank if not required);

Line A: vehicle type. Options are:

- a. light-tracked (25 tonnes or less),**
- b. heavy-tracked (more than 25 tonnes), or**
- c. truck (use for all wheeled vehicles);**

Line B: sprung mass (kg), gross mass (kg);

Line C: pitch mass moment of inertia (kgm^2);

Line D: roll mass moment of inertia (kgm^2);

Line E: number of track assemblies (1 to 4), number of terrain profiles (1 to 4);

Line F-1 (repeat for each track profile): number of different roadwheel radii travelling the profile, list of radii (m);

Line G-0 (repeat for each track assembly; track "0" is for untracked wheels): number of wheels in the assembly;

(Repeat lines H through N for each track assembly. Omit for track "0")

Line H-1: track weight per unit length (N/m), track linear elasticity (N/m);

Line I-1: total track length (m);

Line J-1: location of drive sprocket:

a. front, or

b. rear;

Line K-1: front spridler radius (m), rear spridler radius (m);

Line L-1: horizontal coordinates of front and rear spridlers with respect to hull C.G. (m);

Line M-1: vertical coordinates of front and rear spridlers with respect to hull C.G. (m);

Line N-1: lateral coordinates of front and rear spridlers with respect to hull C.G. (m);

(Repeat lines O through W for each wheel station in the track assembly; then repeat for each track assembly. The first index of the line code refers to the track assembly (0 through 4). The second index refers to the wheel station (1 through 10))

Line O-0-1: suspension type:

a. helical (i.e. pillar), or

b. torsion (i.e. trailing arm),

and profile with which wheel station is associated (1 through 4);

Line P-0-1: for helical suspensions, castor angle (radians). For torsion suspensions, roadarm length (m);

Line Q-0-1: roadwheel radius (m);

Line R-0-1: roadwheel assembly mass (kg);

Line S-0-1: (x, y, z) coordinates of spring/hull attachment point (m) with respect to hull C.G.;

Line T-0-1: for helical suspensions, uncompressed spring length (m), spring constant (N/m). For torsion suspensions, free angle above horizontal (radians), spring constant (Nm/rad);

Line U-0-1: for helical suspensions, compression to bumpstop (m), bumpstop spring constant (N/m). For torsion suspensions, angle of bumpstop above horizontal (radians), bumpstop spring constant (Nm/rad);

Line V-0-1: (x, y, z) coordinates of damper/hull attachment point (m) with respect to hull C.G.; and

Line W-0-1: damping constant (Ns/m), blow-off velocity (m/s).

Running TERPRED.EXE

Once the data files have been prepared, run TERPRED. If the hubpath output file named in META.CNT is non-existent, it will be created. If it already exists, the new data will be written over top of the old. The hubpath data need only

be created for each unique combination of roadwheel radii and terrain profiles. That is, as long as the user does not change the terrain profile or the roadwheel radii in the vehicle data file, the hubpath data may be used. Thus, if the vehicle suspension parameters are changed, or the vehicle is run at a different speed, it is unnecessary to recreate the hubpath data, since the existing data remains valid. This is the advantage of offloading part of the computational effort from the main processor to a pre-processor.

Once the hubpath data is generated, it is advisable to visually inspect it using Lotus 1-2-3, Quattro Pro, or some other software package that allows the user to graph data. This is to ensure that there are no discontinuities or anomalies in the created data. For example, if a half-round obstacle is used as the input terrain, the discontinuity at the leading and trailing edges of the bump will result in anomalous displacements in the hubpath data. These problems can be easily perceived graphically, and can then be corrected.

Running STATIC.EXE

Once the user files have been produced, STATIC can be run. STATIC appends static attitude data to the user-generated vehicle data file. This data appears as lines X through Z in the vehicle data file. Line X gives the location of the C.G. above the ground (in metres), and the pitch and roll of the vehicle (in radians). Line Y gives the horizontal displacement of the roadwheels with respect to the vehicle C.G. (in metres). Line Z gives the track tension in newtons, which is not used by the program, but is provided for user information.

If STATIC has previously been run for the vehicle data file, the program will terminate without doing any calculation. Therefore, if the data file is modified, it is necessary to remove lines X through Z before attempting to run STATIC again.

Running RPM.EXE

Once STATIC and TERPRED have successfully been run, the user may invoke RPM. Program runs can be quite lengthy depending on the terrain length, integration stepsize, and vehicle configuration. However, RPM displays the approximate percentage of the run completed (in terms of the total run distance over the terrain required) in order to give the user some indication of the run-time remaining.

When the run is complete, the time-history data file will be complete. The format of this file is as follows:

Column 1: elapsed time;

Column 2: elapsed distance;

Column 3: hull heave;

Column 4: hull pitch;

Column 5: hull roll;

Column 6: hull heave velocity;

Column 7: hull pitch velocity;

Column 8: hull roll velocity;

Column 9: hull heave acceleration;

Column 10: hull pitch acceleration; and

Column 11: hull roll acceleration.

Running RIDE-Q.EXE

Once the time-history data file has been generated, RIDE-Q can be executed. The primary output of interest is the ride quality report. If the ride quality report already exists when RIDE-Q is run, the new data will be appended to the report. This is useful if, for example, the ride quality at different locations in the vehicle is being evaluated from the same time-history data file.

APPENDIX 2

VEHICLE DATA FILES

Notes on Appendix 2:

The two files included here are the RPM vehicle data files (before running STATIC) representing the M113A1 using half and full vehicle models. Although these two files are for the low damping rate vehicle (i.e. a damping rate of 15 411 Ns/m), the files for the nominal damping rate and high damping rate scenarios are identical except for the damping rates (38 180 Ns/m, and 60 950 Ns/m respectively).

M113-A1, nominal track tension (idler position), half model.

A	light-tracked		
B	5058.9	4500.5	
C	9000.0		
D	4531.0		
E	1	1	
F-1	1	0.3048	
G-0	0		
G-1	5		
H-1	600.4	3000.0	
I-1	9.45		
J-1	Front		
K-1	0.2143	0.2191	
L-1	1.9812	-2.0384	
M-1	-0.3921	-0.4858	
N-1	0.0	0.0	
O-1-1	torsion		
P-1-1	0.3175	1	
Q-1-1	0.3048		
R-1-1	50.8		
S-1-1	1.4400	-0.5300	0.0
T-1-1	-2.3910	9558.0	
U-1-1	-3.2220	7300000.0	
V-1-1	0.9080	-0.4300	0.0
W-1-1	15411	0.30480	
O-1-2	torsion		
P-1-2	0.3175	1	
Q-1-2	0.3048		
R-1-2	45.8		
S-1-2	0.7800	-0.5300	0.0
T-1-2	-2.391	9558.0	
U-1-2	-3.342	7300000.0	
V-1-2	0.2480	-0.4300	0.0
W-1-2	0.0	0.0	
O-1-3	torsion		
P-1-3	0.3175	1	
Q-1-3	0.3048		
R-1-3	45.8		
S-1-3	0.1200	-0.5300	0.0
T-1-3	-2.391	9558.0	
U-1-3	-3.342	7300000.0	
V-1-3	0.2362	-0.5970	0.0
W-1-3	0	0.0	
O-1-4	torsion		
P-1-4	0.3175	1	
Q-1-4	0.3048		
R-1-4	45.8		
S-1-4	-0.5300	-0.5300	0.0
T-1-4	-2.391	9558.0	

U-1-4 -3.342 7300000.0
V-1-4 -0.4307 -0.5790 0.0
W-1-4 0 0.0
O-1-5 torsion
P-1-5 0.3175 1
Q-1-5 0.3048
R-1-5 50.8
S-1-5 -1.1900 -0.5300 0.0
T-1-5 -2.391 9558.0
U-1-5 -3.342 7300000.0
V-1-5 -1.7590 -0.4450 0.0
W-1-5 15411 0.30480

M113-A1, nominal track tension (idler position), full model.

A	light-tracked		
B	7366.0	9001.0	
C	18000.0		
D	4531.0		
E	2 2		
F-1	1	0.3048	
F-2	1	0.3048	
G-0	0		
G-1	5		
G-2	5		
H-1	600.4	3000.0	
I-1	9.45		
J-1	Front		
K-1	0.2143	0.2191	
L-1	1.9812	-2.0384	
M-1	-0.3921	-0.4858	
N-1	1.1670	1.1670	
H-2	600.4	3000.0	
I-2	9.45		
J-2	Front		
K-2	0.2143	0.2191	
L-2	1.9812	-2.0384	
M-2	-0.3921	-0.4858	
N-2	-1.1670	-1.1670	
O-1-1	torsion		
P-1-1	0.3175	2	
Q-1-1	0.3048		
R-1-1	50.8		
S-1-1	1.4400	-0.5300	1.1670
T-1-1	-2.3910	9558.0	
U-1-1	-3.2220	7300000.0	
V-1-1	0.9080	-0.4300	1.1670
W-1-1	15411	0.3048	
O-1-2	torsion		
P-1-2	0.3175	2	
Q-1-2	0.3048		
R-1-2	45.8		
S-1-2	0.7800	-0.5300	1.1670
T-1-2	-2.391	9558.0	
U-1-2	-3.342	7300000.0	
V-1-2	0.2480	-0.4300	1.1670
W-1-2	0.0	0.0	
O-1-3	torsion		
P-1-3	0.3175	2	
Q-1-3	0.3048		
R-1-3	45.8		
S-1-3	0.1200	-0.5300	1.1670
T-1-3	-2.391	9558.0	

U-1-3 -3.342 7300000.0
 V-1-3 0.2362 -0.5970 1.1670
 W-1-3 0 0.0
 O-1-4 torsion
 P-1-4 0.3175 2
 Q-1-4 0.3048
 R-1-4 45.8
 S-1-4 -0.5300 -0.5300 1.1670
 T-1-4 -2.391 9558.0
 U-1-4 -3.342 7300000.0
 V-1-4 -0.4307 -0.5790 1.1670
 W-1-4 0 0.0
 O-1-5 torsion
 P-1-5 0.3175 2
 Q-1-5 0.3048
 R-1-5 50.8
 S-1-5 -1.1900 -0.5300 1.1670
 T-1-5 -2.391 9558.0
 U-1-5 -3.342 7300000.0
 V-1-5 -1.7590 -0.4450 1.1670
 W-1-5 15411 0.3048
 O-2-1 torsion
 P-2-1 0.3175 1
 Q-2-1 0.3048
 R-2-1 50.8
 S-2-1 1.4400 -0.5300 -1.1670
 T-2-1 -2.3910 9558.0
 U-2-1 -3.2220 7300000.0
 V-2-1 0.9080 -0.4300 -1.1670
 W-2-1 15411 0.3048
 O-2-2 torsion
 P-2-2 0.3175 1
 Q-2-2 0.3048
 R-2-2 45.8
 S-2-2 0.7800 -0.5300 -1.1670
 T-2-2 -2.391 9558.0
 U-2-2 -3.342 7300000.0
 V-2-2 0.2480 -0.4300 -1.1670
 W-2-2 0.0 0.0
 O-2-3 torsion
 P-2-3 0.3175 1
 Q-2-3 0.3048
 R-2-3 45.8
 S-2-3 0.1200 -0.5300 -1.1670
 T-2-3 -2.391 9558.0
 U-2-3 -3.342 7300000.0
 V-2-3 0.2362 -0.5970 -1.1670
 W-2-3 0 0.0
 O-2-4 torsion

```

P-2-4  0.3175 1
Q-2-4  0.3048
R-2-4  45.8
S-2-4  -0.5300 -0.5300 -1.1670
T-2-4  -2.391  9558.0
U-2-4  -3.342  7300000.0
V-2-4  -0.4307 -0.5790 -1.1670
W-2-4  0 0.0
O-2-5  torsion
P-2-5  0.3175 1
Q-2-5  0.3048
R-2-5  50.8
S-2-5  -1.1900 -0.5300 -1.1670
T-2-5  -2.391  9558.0
U-2-5  -3.342  7300000.0
V-2-5  -1.7590 -0.4450 -1.1670
W-2-5  15411 0.3048

```

APPENDIX 3

TERRAIN DATA FILES

Notes on Appendix 3:

The terrain profile data file for LETE Course No. 48 is given here (full-vehicle profiles). The sine wave and half-round obstacle are not included as they can be reconstituted from their geometric properties. The half-vehicle profile for LETE Course No. 48 is not shown as the profile vertical displacements are simply the arithmetical average of the left and right vertical displacements for the full-vehicle profile.

LETE COURSE 48 (26APR90)

2

0 0.109728 -0.11887
 0.3048 0.134112 -0.10668
 0.6096 0.140208 -0.09753
 0.9144 0.155448 -0.06553
 1.2192 0.115824 -0.05638
 1.524 0.074676 -0.07162
 1.8288 0.050292 -0.08534
 2.1336 0.054864 -0.09906
 2.4384 0.124968 -0.13563
 2.7432 0.131064 -0.10820
 3.048 0.1143 -0.04724
 3.3528 0.111252 -0.01066
 3.6576 0.102108 -0.02438
 3.9624 0.13868 -0.02895
 4.2672 0.169164 0
 4.572 0.179832 0.039624
 4.8768 0.178308 0.056388
 5.1816 0.192024 -0.01371
 5.4864 0.187452 -0.03352
 5.7912 0.18288 -0.06705
 6.096 0.204216 -0.07162
 6.4008 0.20574 -0.12801
 6.7056 0.219456 -0.14173
 7.0104 0.176784 -0.02438
 7.3152 0.23622 0.001524
 7.62 0.242316 0.051816
 7.9248 0.242316 0.085344
 8.2296 0.196596 0.085344
 8.5344 0.16002 -0.01981
 8.8392 0.111252 -0.02743
 9.144 0.0762 -0.10668
 9.4488 0.067056 -0.06248
 9.7536 0.097536 0.050292
 10.0584 0.096012 0.12192
 10.3632 0.094488 0.131064
 10.668 0.134112 0.141732
 10.9728 0.132588 0.155448
 11.2776 0.156972 0.155448
 11.5824 0.248412 0.146304
 11.8872 0.313944 0.1524
 12.192 0.323088 0.147828
 12.4968 0.31242 0.112776
 12.8016 0.332232 0.094488
 13.1064 0.31242 0.059436
 13.4112 0.28956 0.047244
 13.716 0.263652 0.032004
 14.0208 0.234696 0.036576

14.3256 0.219456 0.059436
14.6304 0.23622 0.077724
14.9352 0.2667 0.057912
15.24 0.28194 0.050292
15.5448 0.28956 0.067056
15.8496 0.332232 0.115824
16.1544 0.345948 0.124968
16.4592 0.359664 0.12954
16.764 0.318516 0.1143
17.0688 0.265176 0.102108
17.3736 0.234696 0.09144
17.6784 0.225552 0.082296
17.9832 0.227076 0.0762
18.288 0.227076 0.070104
18.5928 0.2667 0.070104
18.8976 0.268224 0.085344
19.2024 0.263652 0.057912
19.5072 0.242316 0.024384
19.812 0.231648 0.04572
20.1168 0.211836 -0.03200
20.4216 0.170688 -0.04267
20.7264 0.16764 -0.04572
21.0312 0.187452 -0.0381
21.336 0.150876 -0.04876
21.6408 0.128016 -0.04419
21.9456 0.117348 -0.10668
22.2504 0.085344 -0.12954
22.5552 0.05334 -0.1143
22.86 0.010668 -0.10515
23.1648 0.042672 -0.10363
23.4696 0.123444 -0.02895
23.7744 0.149352 0.05334
24.0792 0.146304 0.100584
24.384 0.12954 0.09144
24.6888 0.0762 0.051816
24.9936 0.112776 0.024384
25.2984 0.028956 0.013716
25.6032 0.00762 -0.09753
25.908 0.073152 -0.10210
26.2128 0.155448 -0.06705
26.5176 0.204216 -0.07924
26.8224 0.240792 0.006096
27.1272 0.291084 0.155448
27.432 0.17526 0.132588
27.7368 0.126492 0.120396
28.0416 0.1524 0.184404
28.3464 0.35814 0.19812
28.6512 0.384048 0.217932
28.956 0.429768 0.124968

29.2608 0.455676 0.1524
29.5656 0.44196 0.217932
29.8704 0.364236 0.249936
30.1752 0.417576 0.248412
30.48 0.43434 0.242316
30.7848 0.43434 0.242316
31.0896 0.428244 0.246888
31.3944 0.425196 0.2667
31.6992 0.44958 0.207264
32.004 0.499872 0.207264
32.3088 0.466344 0.307848
32.6136 0.455676 0.341376
32.9184 0.461772 0.323088
33.2232 0.445008 0.257556
33.528 0.50292 0.225552
33.8328 0.504444 0.231648
34.1376 0.522732 0.249936
34.4424 0.530352 0.227076
34.7472 0.505968 0.231648
35.052 0.501396 0.185928
35.3568 0.519684 0.188976
35.6616 0.527304 0.23622
35.9664 0.475488 0.284988
36.2712 0.531876 0.306324
36.576 0.591312 0.326136
36.8808 0.57912 0.313944
37.1856 0.58674 0.248412
37.4904 0.554736 0.217932
37.7952 0.52578 0.222504
38.1 0.528828 0.28956
38.4048 0.544068 0.268224
38.7096 0.5334 0.2667
39.0144 0.551688 0.283464
39.3192 0.509016 0.278892
39.624 0.513588 0.263652
39.9288 0.528828 0.283464
40.2336 0.5334 0.237744
40.5384 0.499872 0.237744
40.8432 0.536448 0.237744
41.148 0.522732 0.20574
41.4528 0.481584 0.17526
41.7576 0.475488 0.172212
42.0624 0.52578 0.192024
42.3672 0.519684 0.181356
42.672 0.499872 0.184404
42.9768 0.501396 0.1905
43.2816 0.509016 0.224028
43.5864 0.512064 0.249936
43.8912 0.5334 0.272796

44.196 0.515112 0.303276
 44.5008 0.486156 0.234696
 44.8056 0.440436 0.217932
 45.1104 0.422148 0.202692
 45.4152 0.445008 0.195072
 45.72 0.496824 0.219456
 46.0248 0.507492 0.254508
 46.3296 0.52578 0.240792
 46.6344 0.516636 0.234696
 46.9392 0.512064 0.217932
 47.244 0.498348 0.19812
 47.5488 0.519684 0.195072
 47.8536 0.505968 0.192024
 48.1584 0.483108 0.185928
 48.4632 0.463296 0.173736
 48.768 0.446532 0.155448
 49.0728 0.409956 0.13716
 49.3776 0.399288 0.1143
 49.6824 0.405384 0.138684
 49.9872 0.397764 0.13716
 50.292 0.37338 0.108204
 50.5968 0.364236 0.09906
 50.9016 0.367284 0.112776
 51.2064 0.381 0.100584
 51.5112 0.384048 0.09906
 51.816 0.368808 0.103632
 52.1208 0.362712 0.094488
 52.4256 0.37338 0.094488
 52.7304 0.36576 0.085344
 53.0352 0.355092 0.085344
 53.34 0.35052 0.102108
 53.6448 0.35814 0.092964
 53.9496 0.364236 0.09906
 54.2544 0.381 0.064008
 54.5592 0.381 0.056388
 54.864 0.384048 0.074676
 55.1688 0.43434 0.092964
 55.4736 0.44196 0.169164
 55.7784 0.438912 0.181356
 56.0832 0.431292 0.166116
 56.388 0.493776 0.184404
 56.6928 0.554736 0.16764
 56.9976 0.611124 0.103632
 57.3024 0.623316 0.080772
 57.6072 0.641604 0.111252
 57.912 0.635508 0.211836
 58.2168 0.595884 0.208788
 58.5216 0.537972 0.155448
 58.8264 0.466344 0.096012

59.1312 0.469392 0.094488
59.436 0.470916 0.141732
59.7408 0.486156 0.204216
60.0456 0.524256 0.24384
60.3504 0.537972 0.294132
60.6552 0.54864 0.35052
60.96 0.583692 0.409956
61.2648 0.612648 0.42672
61.5696 0.64008 0.423672
61.8744 0.638556 0.422148
62.1792 0.659892 0.440436
62.484 0.669036 0.408432
62.7888 0.691896 0.42672
63.0936 0.714756 0.451104
63.3984 0.789432 0.486156
63.7032 0.9144 0.608076
64.008 1.00584 0.626364
64.3128 1.0287 0.652272
64.6176 1.014984 0.64008
64.9224 1.025652 0.673608
65.2272 0.981456 0.659892
65.532 1.056132 0.6858
65.8368 1.062228 0.801624
66.1416 1.069848 0.880872
66.4464 1.071372 0.874776
66.7512 1.0668 0.85344
67.056 1.175004 0.964692
67.3608 1.205484 1.01346
67.6656 1.258824 0.928116
67.9704 1.267968 0.949452
68.2752 1.228344 0.93726
68.58 1.220724 0.992124
68.8848 1.239012 1.034796
69.1896 1.239012 1.071372
69.4944 1.27254 1.0287

APPENDIX 4**SOURCE CODE FOR TERPRED.EXE**

Notes on Appendix 4:

The source code in Appendices 4 through 7 was written in Borland's Turbo Pascal 5.5, on an IBM PC XT compatible, and run on both this platform and on a 33 MHz 386 machine. The user interface routines use autodetection routines that can determine whether MGA, CGA, EGA, or VGA graphics are available. If there is some conflict with the video adapter on the user's machine, the user should invoke the executable files followed by m (e.g. TERPRED m) to force the display to monochrome mode. Certain Turbo Pascal units (e.g. CRT) are required for compilation.

(*****)

program TERPRED

(c) 1990 Michael Campbell Galway

The purpose of this program is to generate hub paths for use by the main Ride Prediction Model. Pre-process generation of this data removes a heavy computation burden from the main program. Since this data is general for a given set of wheel radii and a given terrain profile, the pre-processing of this data need only be done once, while variables such as vehicle speed and spring rates may be freely changed in the main processor.

The general procedure is to fit a set of cubic splines to the raw terrain data (as many as 2701 data points may be entered), and to use the resultant slope data to calculate the centre of a wheel of given radius sitting tangent to the terrain at various points. A check is made to ensure that a wheel sitting at a given point does not intersect the surrounding ground. A set of cubic splines is then fitted to the wheel centre data in order that equidistant wheel centre coordinates may be interpolated from the data. This data is the hub path data required by the RPM.

B.....cubic spline coefficient for terrain data.
 BHub.....cubic spline coefficient for hub path data.
 C.....cubic spline coefficient for terrain data.
 CHub.....cubic spline coefficient for hub path data.
 D.....cubic spline coefficient for terrain data.
 Delta.....constant horizontal difference for input data.
 DHub.....cubic spline coefficient for hub path data.
 F1.....meta control file.
 F2.....vehicle data file.
 F3.....raw terrain data file.
 F4.....generated hub path data file.
 FName2.....name of vehicle data file.
 FName3.....name of raw terrain data file.
 FName4.....name of hub path data file.
 I.....counter.
 Index.....index to spline.
 Interference...flags interference between ground and wheel.
 J.....counter.
 K.....counter.
 L.....counter.
 Line.....line code used in vehicle data file.
 N.....number of input data points.
 Radius.....radius of wheel.
 TanTheta.....slope of terrain.

```

| TestIndex.....index to spline for interference check.
| Theta.....angle of terrain wrt horizontal.
| U.....horizontal coordinate wrt spline.
| Wheels.....number of wheel radii.
| X.....raw horizontal terrain data.
| Xc.....horizontal centre of wheel.
| XHub.....horizontal coordinate of hub path.
| XT.....arbitrary terrain point.
| XTest.....test value for interference check.
| Xx.....arbitrary terrain point wrt to spline.
| Y.....raw vertical terrain data.
| Yc.....vertical centre of wheel.
| YHub.....vertical coordinate on hub path.
| YOut.....equidistant vertical coordinate of hub path.
| YT.....arbitrary terrain data point.
| YTerrain.....terrain test value for interference check.
| YTest.....wheel coordinate for interference check.
|
| *****
)

```

```

uses crt, Environ, graph;

```

```

type

```

```

    Matrix = array [1..1701] of real;
    BigMem = ^Matrix;

```

```

function Splint (N : integer; X, Y, B, C, D : BigMem; XT : real)
    : real;

```

```

( This function uses a simple binary search method to identify
| which spline should be used for interpolation, and then returns
| the interpolated value for a given value of the independent
| variable.
|

```

```

| The function is based on a FORTRAN routine by J.E.Nielson
| included in the 1983 course notes for 94.265, as published
| by the Department of Systems Engineering, Carleton University.
|

```

```

| B.....coefficient of cubic spline.
| C.....coefficient of cubic spline.
| D.....coefficient of cubic spline.
| Dx.....independent variable wrt interpolating spline.
| Left....left limit of binary search.
| Mid.....bisected search range.

```

```

| N.....number of data points.
| Right...right limit of binary search.
| X.....array of horizontal data points.
| XT.....independent variable for interpolation.
| Y.....array of vertical data points.
|
)

```

```

var

```

```

    Left, Mid, Right : integer;
    Dx                : real;

```

```

begin

```

```

    Left := 1;

```

```

    Right := N;

```

```

    repeat

```

```

        Mid := (Left + Right) div 2;

```

```

        if (Mid = Left) then

```

```

            begin

```

```

                Dx := XT - X^[Left];

```

```

                Splint := Y^[Left] + Dx * (B^[Left]
                    + Dx * (C^[Left] + Dx * D^[Left]));

```

```

            exit;

```

```

        end else if (XT < X^[Mid]) then

```

```

            Right := Mid

```

```

        else Left := Mid;

```

```

    until (true = false);

```

```

end;

```

```

procedure Spline (N : integer; X, Y, B, C, D : BigMem);

```

```

( This procedure fits a set of cubic splines to an arbitrary
| number of input data points. At least 4 and no more than
| 2701 data points must be provided. The ends are constrained
| by fitting cubics to the final points.
|

```

```

| The procedure is based on a FORTRAN routine by J.E.Nielson
| included in the 1983 course notes for 94.265, as published
| by the Department of Systems Engineering, Carleton University.
|

```

```

| B.....coefficient of cubic spline.

```

```

| C.....coefficient of cubic spline.

```

```

| D.....coefficient of cubic spline.

```

```

| Delta...intermediate variable.

```

```

| I.....counter.

```

```

| IStop...counser.
| J.....counter.
| N.....number of input data points.
| Temp....intermediate variable.
| X.....horizontal data.
| Y.....vertical data.
|
)

var
  I, IStop, J : integer;
  Delta       : BigMem;
  Temp        : real;

begin
  new (Delta);

  if ((N < 4) or (N > 2701)) then halt;
  IStop := N - 1;

  for I:=1 to IStop do
    begin
      D^[I] := X^[I+1] - X^[I];
      Delta^[I] := (Y^[I+1] - Y^[I]) / D^[I];
    end;

    B^[1] := -D^[1];

    for I:=2 to IStop do
      B^[I] := 2*(D^[I-1] + D^[I]) -sqr (D^[I-1]) /B^[I-1];

      B^[N] := -D^[N-1] -sqr (D^[N-1]) /B^[N-1];
      Temp := (Delta^[3] -Delta^[2]) / (X^[4] -X^[2])
              -(Delta^[2] -Delta^[1]) / (X^[3] -X^[1]);
      C^[1] := sqr (D^[1]) *Temp / (X^[4] -X^[1]);

      for I:=2 to IStop do
        C^[I] := (Delta^[I] -Delta^[I-1]) -D^[I-1] *C^[I-1] /B^[I-1];

        Temp := (Delta^[N-1] -Delta^[N-2]) / (X^[N] -X^[N-2])
                -(Delta^[N-2] -Delta^[N-3]) / (X^[N-1] -X^[N-3]);
        C^[N] := -sqr (D^[N-1]) *Temp
                / (X^[N] -X^[N-3]) -D^[N-1] *C^[N-1] /B^[N-1];
        C^[N] := C^[N] /B^[N];

      for J:=1 to IStop do
        begin
          I := N - J;
          C^[I] := (C^[I] -D^[I] *C^[I+1]) /B^[I];
        end;

```

3

MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS
STANDARD REFERENCE MATERIAL 1010a
(ANSI and ISO TEST CHART No 2)

```

for I:=1 to IStop do
  begin
    B^[I] := Delta^[I] -D^[I] *(C^[I+1] +2 *C^[I]);
    D^[I] := (C^[I+1] -C^[I]) /D^[I];
    C^[I] := 3 * C^[I];
  end;

  dispose (Delta);
end;

var
  Delta, Percent, TanTheta, Theta, U, Xc, XT      : real;
  XTest, Xx, Yc, YT, YTerrain, YTest             : real;

  BColour1, BColour2, BColour3                   : byte;
  FColour1, FColour2, FColour3                   : byte;
  Colour, GraphDriver, GraphMode                 : integer;
  I, II, Index, J, K, KK, L, N, NumTracks        : integer;
  TestIndex                                       : integer;

  F1, F2, F3, F4                                 : text;

  FName2, FName3, FName4                         : string;
  Line                                             : string [7];
  Interference                                    : boolean;
  Radius                                           : array [1..4,1..4]
                                     of real;
  Wheels                                           : array [1..4]
                                     of integer;

  B, BHub, C, CHub, D, DHub, X, XHub, Y, YHub     : BigMem;
  YOut                                             : array [1..4,1..1001]
                                     of real;

begin
  (  Assign space in heap for "big" variables:
)
  new (X);
  new (Y);
  new (B);
  new (C);
  new (D);
  new (BHub);
  new (CHub);
  new (DHub);

```

```

new (XHub);
new (YHub);

( Initialize window information (for display). Determine whether
| monochrome graphics adapter is present or if user requested
| monochrome override (by using command line parameter 'C'). Set
| the colour flag Colour appropriately to 1 or zero.
|
| Remove the cursor so it won't appear on the process status screen.
)

InitWindowInfo;
detectgraph (GraphDriver, GraphMode);
Colour := 0;
if (paramcount > 0) then Colour := 1;
if ((BooleanMono = true) or (graphdriver = 7)) then Colour := 2;

CursorSize (CursorInvisible, BooleanMono);

( Set background colours and text colours for the process status
| screen. Note that colours are automatically set to black or white
| for monochrome displays. Colours are:
|
| 0....black
| 1....blue
| 3....cyan
| 7....gray
| 15...white
)

case Colour of
  0 : begin
    BColour1 := 1;
    BColour2 := 3;
    BColour3 := 7;

    FColour1 := 15;
    FColour2 := 15;
    FColour3 := 0;
  end;
  1 : begin
    BColour1 := 0;
    BColour2 := 0;
    BColour3 := 15;

    FColour1 := 15;
    FColour2 := 15;
    FColour3 := 0;
  end;
  2 : begin

```

```

        BColour1 := 0;
        BColour2 := 0;
        BColour3 := 15;

        FColour1 := 15;
        FColour2 := 7;
        FColour3 := 0;
        end;
end;

( Set up the background for the process status screen (title
| line, exit message box).
)
textcolor (FColour1);
textbackground (BColour1);
clrscr;

gotoxy (14, 2);
write ('Ride Prediction Model - Wheelpath Generation Processor');
gotoxy (14, 3);
write ('          (c) 1990   Michael Campbell Galway');
OpenWindow (8, 19, 72, 25, BColour1, FColour2, single);
gotoxy (2, 2);
write ('Process Message:');
textbackground (BColour3);
OpenWindow (10, 22, 69, 23, BColour3, FColour3, none);

textcolor (FColour3 + 128);
gotoxy (2, 1);
write ('Working:');
gotoxy (2, 2);
textcolor (FColour3);
write ('Initializing...           ');

assign (F1, 'Meta.CNT');
reset (F1);

( File assignment: F1....meta-control file;
|                  F2....vehicle data file;
|                  F3....raw terrain data;
|                  F4....terrain and hub-path data.
|
| Read user selected file names from meta-control file.
)

readln (F1);
readln (F1, FName2);
readln (F1);

```



```

readln (F1, FName3);
readln (F1, FName4);

assign (F2, FName2);
assign (F3, FName3);
assign (F4, FName4);

reset (F2);
reset (F3);
rewrite (F4);

( Read number of track profiles to be considered.
)
readln (F3);
readln (F3, NumTracks);

( Get number of different radii roadwheels defined in vehicle data
| file.
)
for I:=1 to 6 do
    readln (F2);

for KK:=1 to NumTracks do
    begin
        read (F2, Line, Wheels[KK]);

        for I:=1 to Wheels[KK] do
            read (F2, Radius[KK,I]);
        readln (F2);
    end;

( Read terrain data.
)

for KK:=1 to NumTracks do
    begin
        reset (F3);
        readln (F3);
        readln (F3);
        I := 1;

        repeat
            read (F3, X^[I]);
            for II:=2 to KK do read (F3, Y^[II]);
            readln (F3, Y^[I]);
            I := I + 1;
        until eof (F3);

        N := I - 1;
    end;

```

```

( Fit set of cubic splines to terrain data.
)
  Spline (N, X, Y, B, C, D);

  Delta := (X^N) - X^1) / (N - 1);

  OpenWindow (10, 22, 69, 23, BColour3, FColour3, none);

  textcolor (FColour3 + 128);
  gotoxy (2, 1);
  write ('Working:');
  gotoxy (2, 2);
  textcolor (FColour3);
  write ('Calculating hubpaths...');

( Set up process status box. )

  OpenWindow (13, 4, 68, 16, BColour2, FColour2, double);
  gotoxy (24, 2);
  write ('STATUS:');
  gotoxy (6, 4);
  write ('Wheel #1: ', (KK-1) * 100 div NumTracks,
        '% completed.', ' (Radius = ', Radius[KK,1]:6:4, ').');
  if (Wheels[KK] > 1) then
  begin
    gotoxy (6, 6);
    write ('Wheel #2: ', (KK-1) * 100 div NumTracks,
          '% completed. (Radius = ', Radius[KK,2]:6:4, ').');
  end;
  if (Wheels[KK] > 2) then
  begin
    gotoxy (6, 8);
    write ('Wheel #3: ', (KK-1) * 100 div NumTracks,
          '% completed. (Radius = ', Radius[KK,3]:5:4, ').');
  end;
  if (Wheels[KK] > 3) then
  begin
    gotoxy (6, 10);
    write ('Wheel #4: ', (KK-1) * 100 div NumTracks,
          '% completed. (Radius = ', Radius[KK,4]:5:4, ').');
  end;

  for J:=1 to Wheels[KK] do
  begin
    if (J > 1) then
    begin

```

```

    OpenWindow (28, 3+2*J, 32, 3+2*J, BColour2,
                FColour2, none);
    gotoxy (2,1);
    write (KK *100 div NumTracks, '%');
end;

OpenWindow (28, 5+2*J, 32, 5+2*J, BColour3, FColour3, none);

L := 0;

( For each roadwheel radius, calculate the locus of the hub as the
| wheel rolls over the terrain surface.
)

    for I:= 1 to N do
        begin

            gotoxy (2,1);

            Percent := round (I /N *(KK *100 div NumTracks));
            write (Percent :3:0, '%');

            XT := X^[1] +(I -1) *Delta;
            Index := round ((XT -X^[1]) /Delta) +1;
            if (Index > N-1) then Index := N - 1;
            Xx := XT - X^[Index];
            Yt := Y^[Index] +B^[Index] *Xx +C^[Index] *sqr (Xx)
                +D^[Index] *Xx *sqr (Xx);
            TanTheta := B^[Index] +2 *C^[Index] *Xx *XT
                +3 *D^[Index] *sqr (Xx) *XT;
            Theta := arctan (TanTheta);
            Xc := XT -Radius[KK,J] *sin (Theta);
            Yc := Yt +Radius[KK,J] *cos (Theta);

            Interference := false;

( Check that the roadwheel does not interfere with the ground (ie,
| that it only touches the ground at the point of tangency. If
| not, set the interference flag to 'true'.
)

            for K:=1 to 5 do
                begin
                    XTest := Xc +(K -3) /2 *Radius[KK,J];
                    TestIndex := round ((XTest -X^[1]) /Delta) +1;
                    if (TestIndex < 1) then TestIndex := 1;
                    if (TestIndex > N-1) then TestIndex := N - 1;
                    YTest := -sqrt (abs (sqr (Radius[KK,J])
                        -sqr (Xtest -Xc))) +Yc;
                    U := XTest -X^[TestIndex];
                    YTerrain := Y^[TestIndex] +B^[TestIndex] *U

```

```

        +C^[TestIndex] *sqr (U) +D^[TestIndex]
        *U *sqr (U);
        if ((YTerrain - YTest) > 1E-10)
            then Interference := true;
        end;

( If there was no interference, then save the wheel centre
| coordinates as valid hub path data points.
)

        if (not Interference) then
            begin
                L := L + 1;
                XHub^[L] := Xc;
                YHub^[L] := Yc;
            end;
        end;

( Fit a set of cubic splines to the hub path data.
)

        Spline (L, XHub, YHub, BHub, CHub, DHub);

( Interpolate equidistant hub path coordinates from the spline.
)

        for I:=1 to N do
            begin
                XT := X^[1] +(I -1) *Delta;
                YOut[J,I] := Splint (L, XHub, YHub, BHub, CHub, DHub, XT);
            end;
        end;

OpenWindow (10, 22, 69, 23, BColour3, FColour3, none);

    extcolor (FColour3 + 128);
    gotoxy (2, 1);
    write ('Working:');
    gotoxy (2, 2);
    textcolor (FColour3);
    write ('Outputting hub path data to '+FName4+'.');

( Save the hub path information column-wise as follows:
|
| column 1- horizontal coordinate.
| column 2- vertical terrain coordinate.
| column 3- vertical hub path coordinate, first radius.
| column 4- vertical hub path coordinate, second wheel radius.
| column 5- vertical hub path coordinate, third wheel radius.
| column 6- vertical hub path coordinate, fourth wheel radius.
|

```

```

| Naturally, columns 4 through 6 appear only where necessary.
)

    writeln (F4);
    writeln (F4, N);

    for J:= 1 to N do
        begin
            write (F4, X^[J], ' ', Y^[J]);
            for I:=1 to Wheels[KK] do
                write (F4, ' ', YOut[I,J]);
            writeln (F4);
        end;
    writeln (F4);
end;

( Free memory from heap.
)

dispose (X);
dispose (Y);
dispose (B);
dispose (C);
dispose (D);
dispose (BHub);
dispose (CHub);
dispose (DHub);
dispose (XHub);
dispose (YHub);

close (F4);

( Keep the process status screen displayed until the user hits any
| keyboard key. Then reset screen and cursor, and append the
| equilibrium data to the vehicle data file.
)

OpenWindow (10, 22, 69, 23, BColour3, FColour3, none);

sound (523);
delay (60);
nosound;
delay (60);
sound (587);
delay (60);
nosound;

textcolor (FColour3 + 128);
gotoxy (2, 1);
write ('Success:');
gotoxy (2, 2);

```

```
textcolor (FColour3);  
write ('Wheelpaths generated. Press any key to continue... ');  
  
repeat until keypressed;  
  
window (1,1,80,25);  
textbackground (BColour1);  
textcolor (FColour1);  
  
CursorSize (CursorSmall, BooleanMono);  
  
clrscr;  
  
end.
```

APPENDIX 5

SOURCE CODE FOR STATIC.EXE

Notes on Appendix 5:

Units required for compilation of STATIC.EXE that are not included in Appendix 5 are common to RPM.EXE, and are therefore included in Appendix 6 with the source code for RPM.EXE.


```

unit EqDiff;

( (c) 1990 Michael Campbell Galway
)

interface

uses Data, SuspProc, Sup_Proc, TerrProc, TrakProc;

procedure Derive (SprungMass, IPitch, IRoll : real;
                  NumberOfTracks : integer;
                  NumberOfTrackedWheels : ITrackVector;
                  UnitWeight, Elasticity, TrackLength : TrackArray;
                  Sprocket : AlphaArray;
                  SpridlerRadius, XSpridler, YSpridler,
                    ZSpridler : SpridlerMatrix;
                  RoadArm, WheelRadius, WheelWeight, XSpring,
                    YSpring, ZSpring, SpringLength, SpringConst,
                    BumpStop, BumpConst : WheelMatrix;
                  TypeSusp : AlphaMatrix;
                  var Y, Dy : EqnVector;
                  var XWheel : WheelMatrix;
                  var HorizT : real);

```

implementation

```

procedure Derive (SprungMass, IPitch, IRoll : real;
                  NumberOfTracks : integer;
                  NumberOfTrackedWheels : ITrackVector;
                  UnitWeight, Elasticity, TrackLength : TrackArray;
                  Sprocket : AlphaArray;
                  SpridlerRadius, XSpridler, YSpridler,
                    ZSpridler : SpridlerMatrix;
                  RoadArm, WheelRadius, WheelWeight, XSpring,
                    YSpring, ZSpring, SpringLength, SpringConst,
                    BumpStop, BumpConst : WheelMatrix;
                  TypeSusp : AlphaMatrix;
                  var Y, Dy : EqnVector;
                  var XWheel : WheelMatrix;
                  var HorizT : real);

```

```

( This procedure calculates the forces and moments acting on
| the vehicle at a given vehicle position.
|
| Angle.....angle of upper track at supports.
| ASpringX.....absolute spring-hull coordinate.
| ASpringY.....absolute spring-hull coordinate.
| Bounceforce.....heave force.

```

BumpConst.....bumpstop spring constant.
 BumpStop.....compression to bumpstop.
 DeltaSprung.....weight of off-ground wheels.
 DriveTension.....track tension due to drive torque.
 Dummy.....dummy variable.
 Dy.....differential variables.
 Elasticity.....track linear elasticity.
 Flag.....flag for off-ground wheels.
 FrontTrackAngle.....track angle at front of wheel.
 GroundWheelContact...point of ground-wheel contact.
 GroundWheelSlope.....slope at ground-wheel contact point.
 Gx.....returned by equation solver.
 Gy.....returned by equation solver.
 GZ.....returned by equation solver.
 Heave.....vertical displacement of cg.
 HorizT.....horizontal component of upper track tension.
 I.....counter.
 IPitch.....pitch mass moment of inertia about cg.
 IRoll.....roll mass moment of inertia about cg.
 J.....counter.
 LowerSpridlerAngle...track angle from ground to spridler.
 LowerTrackLength.....track length on ground.
 N.....number of track catenaries.
 NumberOfTrackedWheels
 number of wheels in a given track.
 NumberOfTracks.....number of tracks on a vehicle.
 NumGroundWheels.....number of wheels on the ground.
 Pitch.....pitch displacement about cg.
 PitchMoment.....pitch moment about cg.
 R.....ground reaction on wheel.
 RearTrackAngle.....track angle at rear of wheel.
 RoadArm.....roadarm length/castor angle.
 Roll.....roll displacement about cg.
 RollMoment.....roll moment about cg.
 SpridlerRadius.....idler/sprocket radius.
 SpringConst.....stiffness of spring.
 SpringForceX.....horizontal spring force.
 SpringForceY.....vertical spring force.
 SpringLength.....compressed spring length/angle.
 Sprocket.....flag for location of drive sprocket.
 SprungMass.....sprung mass.
 Support.....flag to indicate roadwheels supporting track.
 SuspReactX.....total horizontal suspension reaction.
 SuspReactY.....total vertical suspension reaction.
 SuspTorque.....suspension reaction torque.
 TotTension.....total tension in upper track.
 TrackLength.....total track length.
 TypeSusp.....torsion bar or helical spring suspension.
 T1.....known track tension.

```

| T2.....unknown track tension.
| UnitWeight.....weight per unit length of track.
| WheelRadius.....roadwheel radius.
| WheelWeight.....roadwheel weight.
| Xa.....absolute (global) coordinate.
| Xr.....relative (local) coordinate.
| XSpridler.....spridler centre coordinate.
| XSpring.....spring-hull coordinate.
| XWheel.....roadwheel centre coordinate.
| Y.....independent variable.
| Ya.....absolute (global) coordinate.
| Yr.....relative (local) coordinate.
| YSpridler.....spridler centre coordinate.
| YSpring.....spring-hull coordinate.
| YWheel.....roadwheel centre coordinate.
| Za.....absolute (global) coordinate.
| Zr.....relative (local) coordinate.
| ZSpridler.....spridler centre coordinate.
| ZSpring.....spring-hull coordinate.
| ZWheel.....roadwheel centre coordinate.
|
|
)
var
  I, J, NumGroundWheels                : integer;

  ASpringX, ASpringY, BounceForce, DeltaSprung, Dummy : real;
  Gx, Gy, G2, Heave, Pitch, PitchMoment, R, Roll    : real;
  RollMoment, T1, T2, Xa, Xr, Ya, Yr, Za, Zr        : real;

  FrontTrackAngle, GroundWheelContact            : WheelMatrix;
  GroundWheelSlope, RearTrackAngle, SpringForceX   : WheelMatrix;
  SpringForceY, SuspReactX, SuspReactY, SuspTorque : WheelMatrix;
  YWheel, ZWheel                                  : WheelMatrix;

  N                                          : ITrackVector;
  DriveTension, LowerTrackLength            : TrackArray;
  LowerSpridlerAngle                       : SpridlerMatrix;
  Angle, TotTension                        : CatMatrix;
  Support, Flag                             : IWheelMatrix;
  CosRoll, CosPitch, SinRoll, SinPitch      : real;

label 10, 20;

begin

( Set coordinates to values from Runge Kutta routine. )

  Heave := Y[1];
  Pitch := Y[2];

```

```

Roll := Y[3];

CosRoll := cos (Roll);
CosPitch := cos (Pitch);
SinRoll := sin (Roll);
SinPitch := sin (Pitch);

( Find wheel positions. )

for I:=0 to NumberOfTracks do
  begin

    for J:=1 to NumberOfTrackedWheels[I] do

      ( Find absolute coordinates of spring-hull attachment points. )

        begin
          Coord ('R->A', Heave, CosRoll, SinRoll, CosPitch, SinPitch,
            0, 0, Xa, Ya, Za,
            XSpring[I,J], YSpring[I,J], ZSpring[I,J]);

          ( Lateral position of the spring-hull attachment point must shift
            | as the vehicle rolls, but the wheel is constrained to remain
            | directly below it in order to avoid camber thrust which would
            | produce a yaw moment.
          )

            ZWheel[I,J] := Za;

            Terrain (Heave, Roll, Pitch, 0, 0, WheelRadius[I,J],
              Xa, Ya, Za,
              RoadArm[I,J], TypeSusp[I,J],
              GroundWheelContact[I,J], GroundWheelSlope[I,J],
              XWheel[I,J], YWheel[I,J], Flag[I,J],
              SpringLength[I,J], BumpStop[I,J], ProfileCode[I,J]);

            end;

          ( For roadwheels which are determined to be off the ground, the
            | horizontal ground-wheel contact point and the slope at the
            | ground-wheel contact point are still required in order to
            | calculate the lower track length. These are set respectively
            | to the lower tangency point with the next rearward wheel
            | (except for the rearmost wheel, which uses the next foremost wheel
            | for these calculations) and to the slope of the lower tangent
            | between the two wheels. Note that this is not done for
            | untracked wheels.
          )

            if (I > 0) then
              for J:=1 to NumberOfTrackedWheels[I] do
                if (Flag[I,J] = 1) then

```

```

        if (J <> NumberOfTrackedWheels[I]) then
            CircleCircleTangent (WheelRadius[I,J],
                                WheelRadius[I,J+1], XWheel[I,J],
                                XWheel[I,J+1], YWheel[I,J],
                                YWheel[I,J+1],
                                GroundWheelContact[I,J], Dummy,
                                Dummy, Dummy,
                                GroundWheelSlope[I,J])
        else
            CircleCircleTangent (WheelRadius[I,J-1],
                                WheelRadius[I,J], XWheel[I,J-1],
                                XWheel[I,J], YWheel[I,J-1],
                                YWheel[I,J],
                                Dummy, GroundWheelContact[I,J],
                                Dummy, Dummy,
                                GroundWheelSlope[I,J]);
    end;

(   Get lower track shape and length as it lies between wheels and
    |   the ground.
)

if NumberOfTracks > 0 then
    LowerTrack (GroundWheelContact, GroundWheelSlope, WheelRadius,
                XWheel, YWheel, Heave, CosRoll, SinRoll, CosPitch,
                SinPitch, 0, 0,
                SpridlerRadius, XSpridler, YSpridler, ZSpridler,
                NumberOfTracks,
                NumberOfTrackedWheels, LowerTrackLength,
                FrontTrackAngle, RearTrackAngle, LowerSpridlerAngle);

(   Calculate spring forces on each suspension assembly.   )

for I:=0 to NumberOfTracks do
    begin

        for J:=1 to NumberOfTrackedWheels[I] do
            begin

(   Helical and torsional spring forces are calculated differently.
    |   Note for example that torsional forces are calculated in the
    |   local vehicle coordinate system, whereas axial helical spring
    |   forces are calculated in the global coordinate system. This is
    |   done for simplicity. The moment calculated for the torsion bar
    |   is unaffected by coordinate transformation, and may therefore
    |   be calculated in the local system.
    )

                if TypeSuspn[I,J] = 'torsion' then
                    begin
                        Coord ('A->R', Heave, CosRoll, SinRoll, CosPitch, SinPitch,

```

```

        0, 0, XWheel[I,J],
        YWheel[I,J], ZWheel[I,J], Xr, Yr, Zr);
    Spring (XSpring[I,J], YSpring[I,J], Xr, Yr,
        SpringConst[I,J], BumpConst[I,J], SpringLength[I,J],
        BumpStop[I,J], 'torsion', SpringForceX[I,J], Dummy);
end;

    if TypeLusp[I,J] = 'helical' then
    begin
        Coord ('R->A', Heave, CosRoll, SinRoll, CosPitch, SinPitch,
            0, 0, Xa, Ya, Za,
            XSpring[I,J], YSpring[I,J], ZSpring[I,J]);
        Spring (Xa, Ya, XWheel[I,J], YWheel[I,J], SpringConst[I,J],
            BumpConst[I,J], SpringLength[I,J], BumpStop[I,J],
            'helical', SpringForceX[I,J], SpringForceY[I,J]);
    end;

end;
end;

( Calculate the forces due to upper track sag. )

if NumberOfTracks > 0 then
begin
    UpperTrack (Heave, CosRoll, SinRoll, CosPitch, SinPitch,
        0, 0, NumberOfTracks,
        LowerTrackLength, TrackLength, UnitWeight, Elasticity,
        WheelRadius, XWheel, YWheel, NumberOfTrackedWheels,
        SpridlerRadius, XSpridler, YSpridler,
        ZSpridler, Angle,
        TotTension, Support, N);
    HorizT := TotTension[1,1,1] * cos (Angle[1,1,1]);
end;

( The coefficients and constants for the set of linear simultaneous
| equations that defines roadwheel static equilibrium are now
| known. A Gauss-Seidel method with pivoting is used to invert the
| equations and solve them. Note that the solution routine is called
| separately for non-tracked wheels since their equations are
| set up differently from those for tracked wheels.
)

I := 0;

( Set the weight added to the sprung mass by suspended roadwheels
| initially to zero.
)

NumGroundWheels := NumberOfTrackedWheels[I];

( Decrement the number of roadwheels on the ground by the number of

```

```

|   roadwheels known to be off the ground. Increment the sprung mass
|   by the mass of these roadwheels.
)

DeltaSprung := 0;
for J:=1 to NumberOfTrackedWheels[I] do
  if (Flag[I,J] = 1) then
    begin
      NumGroundWheels := NumGroundWheels - 1;
      DeltaSprung := DeltaSprung + WheelWeight[I,J];
    end;

for J:=1 to NumberOfTrackedWheels[I] do
  begin
    if (Flag[I,J] = 0) then
      begin
        T2 := 0;
        Coord ('R->A', Heave, CosRoll, SinRoll, CosPitch, SinPitch,
              0, 0,
              ASpringX, ASpringY, Dummy, XSpring[I,J],
              YSpring[I,J], ZSpring[I,J]);
        Pivot (I, XWheel[I,J], YWheel[I,J], ASpringX,
              ASpringY, 0, 0, 0, 0, SpringForceX[I,J], 0, 0,
              SpringForceX[I,J], SpringForceY[I,J],
              GroundWheelSlope[I,J], T2, WheelWeight[I,J],
              NumGroundWheels,
              TypeSusp[I,J], 'rear', Gx, Gy, G2, T1, R);

(  If the ground reaction is less than zero, then there must be a
|  tensile force being supported by the wheel. This is not physically
|  possible, and thus the wheel must be assumed to be off the ground.
|  Reaction forces are set to zero, and the sprung mass is increased
|  by the weight of the wheel. Otherwise, suspension forces are
|  interpreted according to the suspension type.
)

    if R < 0 then
      begin
        DeltaSprung := DeltaSprung + WheelWeight[I,J];
        SuspReactX[I,J] := 0;
        SuspReactY[I,J] := 0;
        SuspTorque[I,J] := 0;
      end else
      begin
        if TypeSusp[I,J] = 'torsion' then
          begin
            SuspTorque[I,J] := 0;
            SuspReactX[I,J] := Gx;
            SuspReactY[I,J] := Gy;
          end;
        end;

```

```

        if TypeSusp[n][j] = 'helical' then
            begin
                SuspTorque[n][j] := G2;
                SuspReactX[n][j] := Gx + SpringForceX[n][j];
                SuspReactY[n][j] := Gy + SpringForceY[n][j];
            end;
        end;
    end else
        begin
            DeltaSprung := DeltaSprung + WheelWeight[n][j];
            SuspReactX[n][j] := 0;
            SuspReactY[n][j] := 0;
            SuspTorque[n][j] := 0;
        end;
    end;
end;

( The equation solving code for tracked wheels follows. Note that it
| is repeated for front and rear sprocket drive vehicles. Front
| sprocket vehicles must be solved from the rearmost roadwheel
| forward, while rear sprocket vehicles must be solved from the
| foremost wheel backwards. This is because the drive tension is
| initially unknown, whereas the tension at the idler is known from
| track sag considerations.
)

for i:=1 to NumberOfTracks do
    begin
10:   NumGroundWheels := NumberOfTrackedWheels[i];

    ( Decrement the number of roadwheels on the ground by the number of
    | roadwheels known to be off the ground. Increment the sprung mass
    | by the mass of these roadwheels.
    )

        DeltaSprung := 0;
        for j:=1 to NumberOfTrackedWheels[i] do
            if (Flag[i][j] = 1) then
                begin
                    NumGroundWheels := NumGroundWheels - 1;
                    DeltaSprung := DeltaSprung + WheelWeight[i][j];
                end;

            if Sprocket[i] = 'Front' then
                begin

    ( The first known tension is the tension supported by the rear idler.
    )

                    T2 := TotTension[i,N[i],2];

    ( Solve the equations from the back of the vehicle to the front.
    )

```



```

    for J:=NumberOfTrackedWheels[I] downto 1 do
      begin
        ( Don't solve for wheels that are known to be off the ground.
        )

        if (Flag[I,J] = 0) then
          begin
            Coord ('R->A', Heave, CosRoll, SinRoll, CosPitch, SinPitch,
              0, 0,
              ASpringX, ASpringY, Dummy, XSpring[I,J],
              YSpring[I,J], ZSpring[I,J]);
            Pivot (I, XWheel[I,J], YWheel[I,J], ASpringX,
              ASpringY, RearTrackAngle[I,J], 0, 0,
              FrontTrackAngle[I,J], SpringForceX[I,J], 0, 0,
              SpringForceX[I,J], SpringForceY[I,J],
              GroundWheelSlope[I,J], T2, WheelWeight[I,J],
              NumGroundWheels, TypeSusp[I,J], Sprocket[I],
              Gx, Gy, G2, T1, R);

            if R < 0 then
              begin
                Flag[I,J] := 1;
                DeltaSprung := DeltaSprung + WheelWeight[I,J];
                SuspReactX[I,J] := 0;
                SuspReactY[I,J] := 0;
                SuspTorque[I,J] := 0;
                NumGroundWheels := NumGroundWheels - 1;

                ( Restart the analysis from the back wheel with the number of
                | roadwheels on the ground decremented by one.
                )

                goto 10;
              end else
                begin
                  SuspReactX[I,J] := Gx;
                  SuspReactY[I,J] := Gy;
                  T2 := T1;
                  if TypeSusp[I,J] = 'torsion' then
                    SuspTorque[I,J] := 0;
                  if TypeSusp[I,J] = 'helical' then
                    begin
                      SuspTorque[I,J] := G2;
                      SuspReactX[I,J] := SuspReactX[I,J]
                        + SpringForceX[I,J];
                      SuspReactY[I,J] := SuspReactY[I,J]
                        + SpringForceY[I,J];
                    end;
                end;
              end else
                end else

```

```

begin

( If the wheel has been determined to be off the ground by terrain-
| wheel geometry, treat it the same as a wheel pulled off the ground
| by suspension forces.
)

    SuspReactX[I,J] := 0;
    SuspReactY[I,J] := 0;
    SuspTorque[I,J] := 0;
end;
end;

( The drive tension has been determined.)

    DriveTension[I] := T1
end;

    if Sprocket[I] = 'Rear' then
begin

( The first known tension is the tension supported by the front idler.
)

    T2 := TotTension[I,1,1];

( Solve the equations from the front of the vehicle to the back.
)

    for J:=1 to NumberOfTrackedWheels[I] do
begin

( Don't solve for wheels that are known to be off the ground.
)

        if (Flag[I,J] = 0) then
begin
            Coord ('R->A', Heave, CosRoll, SinRoll, CosPitch, SinPitch,
                0, 0,
                ASpringX, ASpringY, Dummy, XSpring[I,J],
                YSpring[I,J], ZSpring[I,J]);
            Pivot (I, XWheel[I,J], YWheel[I,J], ASpringX,
                ASpringY, RearTrackAngle[I,J], 0, 0,
                FrontTrackAngle[I,J], SpringForceX[I,J], 0, 0,
                SpringForceX[I,J], SpringForceY[I,J],
                GroundWheelSlope[I,J], T2, WheelWeight[I,J],
                NumGroundWheels, TypeSusp[I,J], Sprocket[I], Gx,
                Gy, G2, T1, R);

( Check whether the reaction force is tensile (physically
| impossible) or compressive.
)

            if R < 0 then

```

```

begin
  Flag[I,J] := 1;
  DeltaSprung := DeltaSprung + WheelWeight[I,J];
  SuspReactX[I,J] := 0;
  SuspReactY[I,J] := 0;
  SuspTorque[I,J] := 0;
  NumGroundWheels := NumGroundWheels - 1;

( Restart the analysis from the front wheel with the number of
| roadwheels on the ground decremented by one.
)

  goto 10;
end else
begin
  SuspReactX[I,J] := Gx;
  SuspReactY[I,J] := Gy;
  T2 := T1;
  if TypeSusp[I,J] = 'torsion' then
    SuspTorque[I,J] := 0;
  if TypeSusp[I,J] = 'helical' then
    begin
      SuspTorque[I,J] := G2;
      SuspReactX[I,J] := SuspReactX[I,J]
        + SpringForceX[I,J];
      SuspReactY[I,J] := SuspReactY[I,J]
        + SpringForceY[I,J];
    end;
  end;
end else
begin

( If the wheel has been determined to be off the ground by terrain-
| wheel geometry, treat it the same as a wheel pulled off the ground
| by suspension forces.
)

  SuspReactX[I,J] := 0;
  SuspReactY[I,J] := 0;
  SuspTorque[I,J] := 0;
end;
end;

( The drive tension has been determined. )

  DriveTension[I] := T1
end;
end;

```

```

( Assess hull forces, beginning with the total force in vertical
| (heave) direction on the hull.
)

BounceForce := 0;
for I:=0 to NumberOfTracks do
  begin
    for J:=1 to NumberOfTrackedWheels[I] do
      begin

( Sum the vertical suspension reactions.  )

        BounceForce := BounceForce + SuspReactY[I,J];
      end;

      if (I >= 1) then
        begin

( If the vehicle has tracks, add the vertical component of the track
| sag tension to the vertical force.
)

          BounceForce := BounceForce
            -TotTension[I,1,1] *sin (Angle[I,1,1])
            +TotTension[I,N[I],2] *sin (Angle[I,N[I],2]);

( Add the vertical component of track tension attributable to drive
| sprocket tension.
)

          if Sprocket[I]='Rear' then
            BounceForce := BounceForce
              +DriveTension[I]
              *sin (LowerSpridlerAngle[I,2])
              -TotTension[I,1,1]
              *sin (LowerSpridlerAngle[I,1]);

            if Sprocket[I]='Front' then
              BounceForce := BounceForce
                +TotTension[I,1,1]
                *sin (LowerSpridlerAngle[I,2])
                -DriveTension[I]
                *sin (LowerSpridlerAngle[I,1]);

          end;
        end;

( Add the downward force due to the weight of the hull.  )

        BounceForce := BounceForce -(SprungMass +DeltaSprung) *9.81;

```

```

( Find the total pitch moment acting on the hull. )

PitchMoment := 0;
for I:=0 to NumberOfTracks do
  begin
    for J:=1 to NumberOfTrackedWheels[I] do
      begin
        Coord ('R->A', Heave, CosRoll, SinRoll, CosPitch, SinPitch,
              0, 0, Xa, Ya, Za,
              XSpring[I,J], YSpring[I,J], ZSpring[I,J]);

( Vertical suspension reactions and horizontal suspension reactions
| should be weighted according to their moment arms and summed about
| the vehicle c.g. Suspension reaction torque may be added directly.
)

        PitchMoment := PitchMoment + SuspReactY[I,J] *(Xa);
        PitchMoment := PitchMoment + SuspReactX[I,J] *(Ya -Heave);
        PitchMoment := PitchMoment + SuspTorque[I,J];

        if TypeSusp[I,J] ='torsion' then
          begin

( Torsion bar type suspensions must be treated separately to add
| their wind-up torque.
)

            PitchMoment := PitchMoment + SpringForceX[I,J];
            end;
          end;

        if (I >= 1) then
          begin

( If the vehicle has tracks, add the contributions of the track
| tensions and off the drive tensions.
)

            Coord ('R->A', Heave, CosRoll, SinRoll, CosPitch, SinPitch,
                  0, 0, Xa, Ya, Za,
                  XSpridler[I,1], YSpridler[I,1], ZSpridler[I,1]);

            PitchMoment := PitchMoment
              -TotTension[I,1,1] *sin (Angle[I,1,1])
              *(Xa -SpridlerRadius[I,1]
              *sin (Angle[I,1,1]))
              -TotTension[I,1,1] *cos (Angle[I,1,1])
              *(Ya -Heave +SpridlerRadius[I,1]
              *cos (Angle[I,1,1]));

```

```

if Sprocket[I]='Rear' then
  PitchMoment := PitchMoment
    -TotTension[I,1,1]
    *sin (LowerSpridlerAngle[I,1])
    *(Xa +SpridlerRadius[I,1]
    *sin (LowerSpridlerAngle[I,1]))
    -TotTension[I,1,1]
    *cos (LowerSpridlerAngle[I,1])
    *(Ya -Heave -SpridlerRadius[I,1]
    *cos(LowerSpridlerAngle[I,1]));

if Sprocket[I]='Front' then
  PitchMoment := PitchMoment
    -DriveTension[I]
    *sin (LowerSpridlerAngle[I,1])
    *(Xa +SpridlerRadius[I,1]
    *sin (LowerSpridlerAngle[I,1]))
    -DriveTension[I]
    *cos (LowerSpridlerAngle[I,1])
    *(Ya -Heave -SpridlerRadius[I,1]
    *cos (LowerSpridlerAngle[I,1]));

Coord ('R->A', Heave, CosRoll, SinRoll, CosPitch, SinPitch,
0, 0, Xa, Ya, Za,
XSpridler[I,2], YSpridler[I,2], ZSpridler[I,2]);

PitchMoment := PitchMoment
+TotTension[I,N[I],2] *sin (Angle[I,N[I],2])
*(Xa -SpridlerRadius[I,2]
*sin (Angle[I,N[I],2]))
+TotTension[I,N[I],2] *cos (Angle[I,N[I],2])
*(Ya -Heave +SpridlerRadius[I,2]
*cos (Angle[I,N[I],2]));

if Sprocket[I]='Rear' then
  PitchMoment := PitchMoment
    +DriveTension[I]
    *sin (LowerSpridlerAngle[I,2])
    *(Xa -SpridlerRadius[I,2]
    *sin (LowerSpridlerAngle[I,2]))
    +DriveTension[I]
    *cos (LowerSpridlerAngle[I,2])
    *(Ya -Heave -SpridlerRadius[I,2]
    *cos (LowerSpridlerAngle[I,2]));

if Sprocket[I]='Front' then
  PitchMoment := PitchMoment
    +TotTension[I,N[I],2]
    *sin (LowerSpridlerAngle[I,2])

```

```

        *(Xa -SpridlerRadius[I,2]
        *sin (LowerSpridlerAngle[I,2]))
        +TotTension[I,N[I],2]
        *cos (LowerSpridlerAngle[I,2])
        *(Ya -Heave -SpridlerRadius[I,2]
        *cos(LowerSpridlerAngle[I,2]));

    end;

end;

( Find the total roll moment acting on hull. )

RollMoment := 0;
for I:=0 to NumberOfTracks do
    begin
        for J:=1 to NumberOfTrackedWheels[I] do
            begin

( Vertical suspension reactions and horizontal suspension reactions
| should be weighted according to their moment arms and summed about
| the vehicle c.g.
)

                Coord ('R->A', Heave, CosRoll, SinRoll, CosPitch, SinPitch,
                0, 0, Xa, Ya, Za,
                XSpring[I,J], YSpring[I,J], ZSpring[I,J]);
                RollMoment := RollMoment - SuspReactY[I,J] *cos (Pitch) *Za;
                RollMoment := RollMoment - SuspReactX[I,J] *sin (-Pitch) *Za;
            end;

            if (I >= 1) then
                begin

( If the vehicle has tracks, add the contributions of the track
| tensions and off the drive tensions.
)

                    Coord ('R->A', Heave, CosRoll, SinRoll, CosPitch, SinPitch,
                    0, 0, Xa, Ya, Za,
                    XSpridler[I,1], YSpridler[I,1], ZSpridler[I,1]);

                    RollMoment := RollMoment
                    +TotTension[I,1,1] *sin (Angle[I,1,1]) *Za
                    *cos (Pitch)
                    +TotTension[I,1,1] *cos (Angle[I,1,1]) *Za
                    *sin (-Pitch);
                end;
            end;
        end;
    end;
end;

```

```

if Sprocket[I]='Rear' then
  RollMoment := RollMoment
    +TotTension[I,1,1]
    *sin (LowerSpridlerAngle[I,1]) *Za
    *cos (Pitch)
    +TotTension[I,1,1]
    *cos (LowerSpridlerAngle[I,1]) *Za
    *sin (-Pitch);

if Sprocket[I]='Front' then
  RollMoment := RollMoment
    +DriveTension[I]
    *sin (LowerSpridlerAngle[I,1]) *Za
    *cos (Pitch)
    +DriveTension[I]
    *cos (LowerSpridlerAngle[I,1]) *Za
    *sin (-Pitch);

Coord ('R->A', Heave, CosRoll, SinRoll, CosPitch, SinPitch,
  0, 0, Xa, Ya, Za,
  XSpridler[I,2], YSpridler[I,2], ZSpridler[I,2]);

RollMoment := RollMoment
  -TotTension[I,N[I],2] *sin (Angle[I,N[I],2]) *Za
  *cos (Pitch)
  -TotTension[I,N[I],2] *cos (Angle[I,N[I],2]) *Za
  *sin (-Pitch);

if Sprocket[I]='Rear' then
  RollMoment := RollMoment
    -DriveTension[I]
    *sin (LowerSpridlerAngle[I,2]) *Za
    *cos (Pitch)
    -DriveTension[I]
    *cos (LowerSpridlerAngle[I,2]) *Za
    *sin (-Pitch);

if Sprocket[I]='Front' then
  RollMoment := RollMoment
    -TotTension[I,N[I],2]
    *sin (LowerSpridlerAngle[I,2]) *Za
    *cos (Pitch)
    -TotTension[I,N[I],2]
    *cos (LowerSpridlerAngle[I,2]) *Za
    *sin (-Pitch);

end;

end;

```



```

(   Calculate heave, pitch, and roll accelerations from the force
|   and moments, and assign these to the differential variables.
)
  Dy[4] := BounceForce / (SprungMass + DeltaSprung);
  Dy[5] := PitchMoment / IPitch;
  Dy[6] := RollMoment / IRoll;

(   Make final assignment of variables before return to Runge-Kutta
|   routine.
)
  Dy[1] := Y[4];
  Dy[2] := Y[5];
  Dy[3] := Y[6];

end;

begin
end.

```

(*****)

program STATIC

(c) 1990 Michael Campbell Galway

The purpose of this program is to evaluate the static equilibrium position of a tracked or wheeled vehicle. This data is required for the Ride Prediction Model.

Equilibrium data consists of centre of gravity height above the ground (heave), pitch attitude about the cg, roll attitude above the cg, and wheel centre longitudinal position with respect to the cg. Data is computed by evaluating heave, pitch, and roll accelerations at different vehicle attitudes. The equilibrium position is the attitude for which these accelerations are below some tolerance (much smaller than zero). The binary search method is used to solve for the appropriate values of heave, pitch, and roll.

Accuracy.....criterion for computation of equilibrium data.

BColour1.....background colour of process status screen.

BColour2.....background colour of process status box.

BColour3.....background colour of solution boxes.

BumpConst.....spring constant of bump stop.

BumpStop.....roadarm susp'ns: angle above horizontal of
bumpstop.

helical susp'ns: spring compression to bumpstop.

Colour.....0 if monochrome, 1 if colour adapter.

EqnNum.....number of differential equations (6).

FColour1.....text colour.

FColour2.....text colour.

FColour3.....text colour.

FName1.....vehicle data file name.

FName2.....control file name.

F1.....meta-control file.

F2.....vehicle data file.

F3.....process control file.

GraphDriver.....type of graphics driver detected.

GraphMde.....graphics mode detected.

I.....a counter.

IPitch.....pitch mass moment of inertia of vehicle hull.

IRoll.....roll mass moment of inertia of vehicle hull.

Iterate.....number of iterations permitted for convergence.

J.....a counter

NumberOfTrackedWheels

.....number of wheels in a track.

NumberOfTracks...number of tracks on vehicle.

```

| RoadArm.....roadarm susp'ns: length of roadarm.
|               helical susp'ns: castor angle from vertical.
| SpridlerRadius...radii of sprocket and idler.
| SpringConst.....spring constant.
| SpringLength.....roadarm susp'ns: free angle of roadarm above
|                   x axis.
|               helical susp'ns: free length of spring.
| Sprocket.....'front' if front drive, else 'rear'.
| SprungMass.....mass of sprung body.
| TrackLength.....total tracklength per track.
| TypeSusp'n.....'torsion' if roadarm suspension, else 'helical'.
| TypeVehicle.....'truck', 'light armoured vehicle', 'heavy armoured
|                 vehicle', 'automobile', or 'motorcycle'.
| UnitWeight.....weight of track per unit length.
| WheelRadius.....radii of roadwheels.
| WheelWeight.....weight of roadwheel assemblies.
| X.....independent variables in differential equations.
| XWheel.....equilibrium position of wheel centres.
| XSpridler.....longitudinal position of spridlers from cg.
| XSpring.....longitudinal positions of spring/hull
|               attachment pts.
| YSpridler.....vertical position of spridlers from cg.
| YSpring.....vertical positions of spring/hull attachment pts.
| ZSpridler.....lateral positions of spridlers from cg.
| ZSpring.....lateral positions of spring/hull attachment pts.
|
| *****}

```

```

uses Crt, Data, Environ, EqDiff, Graph, Sup_Proc, SuspProc,
    TrakProc;

```

```

procedure Equilibrium (Accuracy, SprungMass, IPitch, IRoll : real;
    EqNum, Iterate, NumberOfTracks : integer;
    TypeVehicle : Alphabet;
    NumberOfTrackedWheels : ITrackVector;
    UnitWeight, Elasticity, TrackLength : TrackArray;
    Sprocket : AlphaArray;
    SpridlerRadius, XSpridler, YSpridler,
    ZSpridler : SpridlerMatrix;
    RoadArm, WheelRadius, WheelWeight, XSpring,
    YSpring, ZSpring, SpringLength, SpringConst,
    BumpStop, BumpConst: WheelMatrix;
    TypeSusp'n : AlphaMatrix;

```

```

var X : EqnVector;
var XWheel : WheelMatrix;
BColour1, BColour2, BColour3,
    FColour1, FColour2, FColour3 : byte;
var HorizT : real);

( This procedure calculates the equilibrium position of the
| vehicle, using the binary search method of solving equations.
|
| Accuracy.....criterion for computation of equilibrium data.
| BColour2.....background colour of process status box.
| BColour3.....background colour of solution boxes.
| BumpConst.....spring constant of bump stop.
| BumpStop.....roadarm susp'ns: angle above horizontal of
|               bumpstop.
|               helical susp'ns: spring compression to bumpstop.
| Dummy.....holder for unused (damping) variables.
| EqnNum.....number of differential equations (6).
| F.....acceleration array.
| FColour2.....text colour.
| FColour3.....text colour.
| H.....search parameter.
| I.....a counter.
| IPitch.....pitch mass moment of inertia of vehicle hull.
| IRoll.....roll mass moment of inertia of vehicle hull.
| Iterate.....number of iterations permitted for convergence.
| J.....a counter
| Length.....heave with hull pushed to first bumpstop.
| Length2.....test values of heave.
| NumberOfTrackedWheels
|       .....number of wheels in a track.
| NumberOfTracks...number of tracks on vehicle.
| P.....search parameter.
| R.....search parameter.
| RoadArm.....roadarm susp'ns: length of roadarm.
|               helical susp'ns: castor angle from vertical.
| Sign.....direction of search to solution.
| SpridlerRadius...radii of sprocket and idler.
| SpringConst.....spring constant.
| SpringLength.....roadarm susp'ns: free angle of roadarm above
|               x axis.
|               helical susp'ns: free length of spring.
| Sprocket.....'front' if front drive, else 'rear'.
| SprungMass.....mass of sprung body.
| Sum.....sum of heave, roll, and pitch accelerations.
| S1.....search parameter.
| S2.....search parameter.
| Test.....test value to determine Sign.
| TrackLength.....total tracklength per track.

```

```

| TypeSuspns.....'torsion' if roadarm suspension, else 'helical'.
| TypeVehicle.....'truck', 'light armoured vehicle', 'heavy armoured
|                   vehicle', 'automobile', or 'motorcycle'.
| UnitWeight.....weight of track per unit length.
| WheelRadius.....radii of roadwheels.
| WheelWeight.....weight of roadwheel assemblies.
| X.....independent variables in differential equations.
| XSpridler.....longitudinal position of spridlers from cg.
| XSpring.....longitudinal positions of spring/hull
|               attachment pts.
| XWheel.....longitudinal wheel centre positions.
| X1.....new value of X in iterative solution.
| YSpridler.....vertical position of spridlers from cg.
| YSpring.....vertical positions of spring/hull attachment pts.
| ZSpridler.....lateral positions of spridlers from cg.
| ZSpring.....lateral positions of spring/hull attachment pts.
)

```

```

var
  I, J                      : integer;
  Factor, Length, Length2, Sum, Test, X1  : real;
  FMin, FMax, XMin, XMax, H, P, R, S1, S2, Sign      : array [1..3] of real;
  F                      : EqnVector;
  Dummy                  : WheelMatrix;

```

```

begin

```

```

( Set up process status box. )

```

```

  OpenWindow (13, 4, 68, 16, BColour2, FColour2, double);
  gotoxy (17, 2);
  write ('Iterations:');
  gotoxy (3, 5);
  write ('Heave:');
  gotoxy (3, 7);
  write ('Pitch:');
  gotoxy (3, 9);
  write ('Roll :');
  gotoxy (21, 5);
  write (' Acc:');
  gotoxy (21, 7);
  write (' Acc:');
  gotoxy (21, 9);
  write (' Acc:');
  if (NumberOfTracks > 0) then
    begin
      gotoxy (3, 11);
      write ('Horizontal track tension:');
      OpenWindow (42, 14, 51, 14, BColour3, FColour3, none);

```

```

end;

OpenWindow (42, 5, 46, 5, BColour3, FColour3, none);
OpenWindow (22, 8, 31, 8, BColour3, FColour3, none);
OpenWindow (22, 10, 31, 10, BColour3, FColour3, none);
OpenWindow (22, 12, 31, 12, BColour3, FColour3, none);
OpenWindow (40, 8, 64, 8, BColour3, FColour3, none);
OpenWindow (40, 10, 64, 10, BColour3, FColour3, none);
OpenWindow (40, 12, 64, 12, BColour3, FColour3, none);
window (1, 1, 80, 25);

( Find the height of the cg when the vehicle hull is depressed
| until the first bumpstop is met. This is the initial value of
| heave. The search is begun with the vehicle compressed because
| the track will not fit on the vehicle if the uncompressed
| configuration is used.
|
| The cg height is found for each suspension element compressed
| to its bumpstop, and the greatest of these heights is used.
)

Length := -99999;

for I:=0 to NumberOfTracks do
begin
  for J:=1 to NumberOfTrackedWheels[I] do
begin
  Length2 := -YSpring[I,J] + WheelRadius[I,J];

( If the suspension element is a swinging arm, the cg height is
| the sum of the wheel radius, the vertical distance between the
| cg and the spring-hull attachment point, and the vertical component
| of the roadarm when it is swung to meet the bumpstop.
)

  if TypeSusp[I,J] = 'torsion' then
    Length2 := Length2 - RoadArm[I,J] *sin (BumpStop[I,J]);

( If the suspension element is a helical spring, the vertical
| component of the roadarm is replaced by the vertical component of
| the compressed spring. Remember that helical suspensions are
| allowed to be castored.
)

  if TypeSusp[I,J] = 'helical' then
    Length2 := Length2 +(SpringLength[I,J] -BumpStop[I,J])
      *cos (RoadArm[I,J]);
  if Length2 > Length then Length := Length2;
end;

```

```

end;

( Set up the search parameters.
| Set heave, pitch, and roll displacements and accelerations to zero.
)

for I:=1 to EqnNum do
  begin
    S1[I] := 1;
    P[I] := 0;
    R[I] := 3;
    X[I] := 0;
    X[I+3] := 0;
  end;

( Set the initial heave to the compressed bumpstop configuration. )

X[1] := Length;

( Run tests to determine whether heave, pitch, and roll should be
| incremented or decremented in order to converge on the
| equilibrium position. This is done by finding accelerations
| at two close positions for each variable.
)

for I:=1 to EqnNum do
  begin

    textcolor (FColour3);
    gotoxy (11, 23);
    write ('First initialization pass- DOF = ', I, ' ');

    Derive (SprungMass, IPitch, IRoll,
            NumberOfTracks, NumberOfTrackedWheels,
            UnitWeight, Elasticity, TrackLength, Sprocket,
            SpridlerRadius, XSpridler, YSpridler, ZSpridler, RoadArm,
            WheelRadius, WheelWeight, XSpring, YSpring, ZSpring,
            SpringLength, SpringConst, BumpStop, BumpConst,
            TypeSusp, X, F, XWheel, HorizT);

    Test := F[I+3];
    X[I] := X[I] - 0.01;

    textcolor (FColour3);
    gotoxy (11, 23);
    write ('Second initialization pass- DOF = ', I, ' ');

    Derive (SprungMass, IPitch, IRoll,
            NumberOfTracks, NumberOfTrackedWheels,

```

```

UnitWeight, Elasticity, TrackLength, Sprocket,
SpridlerRadius, XSpridler, YSpridler, ZSpridler, RoadArm,
WheelRadius, WheelWeight, XSpring, YSpring, ZSpring,
SpringLength, SpringConst, BumpStop, BumpConst,
TypeSusp, X, F, XWheel, HorizT);

if (Test < F[I+3]) then
begin
  Sign[I] := +1;
end else
begin
  Sign[I] := -1;
end;

X[I] := 0;
X[1] := Length;
S2[I] := Sign[I];
H[I] := 0.1;
end;

( Call procedure Derive iteratively to converge on the equilibrium
| position. Note that all variables are adjusted after each
| calculation of the accelerations.
)

textcolor (FColour3);
gotoxy (11, 23);
write ('Initialization complete. Solving... ');

for I:=1 to Iterate do
begin
  Derive (SprungMass, IPitch, IRoll,
    NumberOfTracks, NumberOfTrackedWheels,
    UnitWeight, Elasticity, TrackLength, Sprocket,
    SpridlerRadius, XSpridler, YSpridler, ZSpridler, RoadArm,
    WheelRadius, WheelWeight, XSpring, YSpring, ZSpring,
    SpringLength, SpringConst, BumpStop, BumpConst,
    TypeSusp, X, F, XWheel, HorizT);

  Sum := 0;

  for J:=1 to EqNum do
    begin

( If the acceleration for a degree of freedom is above a certain
| tolerance, calculate a new value for that degree of freedom.
)

```



```

if (abs (F[J+3]) > Accuracy /100) then

    begin

        Sign[J] := Sgn (F[J+3]);
        if (Sign[J] <> S2[J]) then H[J] := 0.5 *H[J];

        X1    := X[J] +Sign[J] *H[J];

    ( Write updated information to the process status box on screen.  )

        gotoxy (42, 5);
        write (1 :3, ' ');
        gotoxy (22, 8);
        write (X[1] :8 :5, ' ');
        gotoxy (22, 10);
        write (X[2] :8 :5, ' ');
        gotoxy (22, 12);
        write (X[3] :8 :5, ' ');
        gotoxy (40, 8);
        write (F[4], ' ');
        gotoxy (40, 10);
        write (F[5], ' ');
        gotoxy (40, 12);
        write (F[6], ' ');
        if (NumberOfTracks > 0) then
            begin
                gotoxy (40, 14);
                write (' ', HorizT :8 :5);
            end;

    ( Generate a sum of the accelerations for all degrees of freedom.  )

        Sum := Sum +abs (X1 -X[J]) +abs (F[J+3]);
        X[J] := X1;
        S2[J] := Sign[J];
    end;
end;

( If the sum of the accelerations is within a given tolerance,
| then the equilibrium position has been found. Write this
| information to the exit message box.
)

if (Sum < Accuracy) then
    begin

        sound (523);
        delay (60);

```

```

        nosound;
        delay (60);
        sound (587);
        delay (60);
        nosound;

        textcolor (FColour3 + 128);
        gotoxy (11, 22);
        writeln ('Success:');
        textcolor (FColour3);
        gotoxy (11, 23);
        write ('Equilibrium position found. Press any key... ');
        exit;
    end;
end;

( Convergence has not been reached within the allowed number of
| iterations. Write a failure report to the exit message box.
)

sound (523);
delay (120);
nosound;

textcolor (FColour3 + 128);
gotoxy (11, 22);
write ('Failure:');
textcolor (FColour3);
gotoxy (11, 23);
write ('Equilibrium position not found. ');
repeat until keypressed;
window (1, 1, 80, 25);
textbackground (BColour1);
textcolor (FColour1);
CursorSize (CursorSmall, true);
clrscr;
halt;
end;

var
    Accuracy, IPitch, IRoll, SprungMass, HorizonT : real;
    X : EqnVector;
    UnitWeight, TrackLength, Elasticity : TrackArray;
    SpridlerRadius, XSpridler, YSpridler, ZSpridler : SpridlerMatrix;
    RoadArm, WheelRadius, WheelWeight, XSpring : WheelMatrix;
    YSpring, ZSpring, SpringLength, SpringConst : WheelMatrix;

```

```

    BumpStop, BumpConst, XWheel           : WheelMatrix;

    Colour, EqNum, GraphDriver, GraphMode  : integer;
    I, Iterate, J, NumberOfTracks          : integer;
    NumberOfProfiles                       : integer;
    BColour1, BColour2, BColour3           : byte;
    FColour1, FColour2, FColour3           : byte;
    DiffRRRadii, NumberOfTrackedWheels    : ITrackVector;

    F1, F2, F3                             : text;
    FName1, FName2                         : string;
    Line                                   : string(7);
    TypeVehicle                           : Alphabet;
    Sprocket                              : AlphaArray;
    TypeSuspension                         : AlphaMatrix;

begin

(   Initialize window information (for display). Determine whether
|   monochrome graphics adapter is present or if user requested
|   monochrome override (by using command line parameter 'C'). Set
|   the colour flag Colour appropriately to 1 or zero.
|
|   Remove the cursor so it won't appear on the process status screen.
)

    InitWindowInfo;
    detectgraph(GraphDriver, GraphMode);
    Colour := 0;
    if (paramcount > 0) then Colour := 1;
    if ((BooleanMono = true) or (graphdriver = 7)) then Colour := 2;

    CursorSize (CursorInvisible, BooleanMono);

(   Set background colours and text colours for the process status
|   screen. Note that colours are automatically set to black or white
|   for monochrome displays. Colours are:
|
|       0....black
|       1....blue
|       3....cyan
|       7....gray
|       15...white
)

    case Colour of
        0 : begin
            BColour1 := 1;
            BColour2 := 3;

```

```

        BColour3 := 7;

        FColour1 := 15;
        FColour2 := 0;
        FColour3 := 0;
        end;
1 : begin
    BColour1 := 0;
    BColour2 := 0;
    BColour3 := 15;

    FColour1 := 15;
    FColour2 := 15;
    FColour3 := 0;
    end;
2 : begin
    BColour1 := 0;
    BColour2 := 0;
    BColour3 := 15;

    FColour1 := 15;
    FColour2 := 7;
    FColour3 := 0;
    end;
end;

( Set up the background for the process status screen (title
| line, exit message box).
)

textcolor (FColour1);
textbackground (BColour1);
clrscr;

gotoxy (14, 2);
write ('Ride Prediction Model - Vehicle Equilibrium Processor');
gotoxy (14,3);
write ('          (c) 1990   Michael Campbell Galwey');
OpenWindow (8, 19, 72, 25, BColour1, FColour1, single);
textbackground (BColour3);
OpenWindow (10, 22, 69, 23, BColour3, FColour3, none);

textcolor (FColour3 + 128);
gotoxy (2, 1);
write ('Working:');
gotoxy (2, 2);
textcolor (FColour3);
write ('Reading vehicle data.

for I:=1 to 4 do

```

');

```

    begin
    new (Terr[I]);
    for J:=0 to 10 do new (Terr[I]^Y[J]);
end;

( Open standard meta-control file Meta.CNT, and read names of
| vehicle data file and STATIC control file.
)

assign (F1, 'Meta.CNT');
reset (F1);

readln (F1);
readln (F1, FName1);
readln (F1, FName2);

assign (F2, FName1);
assign (F3, FName2);

reset (F2);
reset (F3);

EqnNum := 3;

( Read in vehicle data. Note that damping data is not
| required, and thus is skipped over.
)

readln (F2);
readln (F2, Line, TypeVehicle);
readln (F2, Line, SprungMass);
readln (F2, Line, IPitch);
readln (F2, Line, IRoll);
readln (F2, Line, NumberOfTracks, NumberOfProfiles);

for J:=1 to NumberOfProfiles do
begin
  read (F2, Line, DiffRURadii[J]);
  if (DiffRURadii[J] > 5) then halt;
  for I:=1 to DiffRURadii[J] - 1 do
    read (F2, Terr[J]^Wheel[I]);
  readln (F2, Terr[J]^Wheel[DiffRURadii[J]]);

  for I:=1 to DiffRURadii[J] do
  begin
    Terr[J]^Y[I]^0 := Terr[J]^Wheel[I];
    Terr[J]^Y[I]^1 := Terr[J]^Wheel[I];
  end;
end;

```

```

end;

for I:=0 to NumberOfTracks do
begin
  readln (F2, Line, NumberOfTrackedWheels[I]);
end;

for I:=1 to NumberOfTracks do
begin
  readln (F2, Line, UnitWeight[I], Elasticity[I]);
  readln (F2, Line, TrackLength[I]);
  readln (F2, Line, Sprocket[I]);
  readln (F2, Line, SpridlerRadius[I,1], SpridlerRadius[I,2]);
  readln (F2, Line, XSpridler[I,1], XSpridler[I,2]);
  readln (F2, Line, YSpridler[I,1], YSpridler[I,2]);
  readln (F2, Line, ZSpridler[I,1], ZSpridler[I,2]);
end;

for I:=0 to NumberOfTracks do
begin
  for J:=1 to NumberOfTrackedWheels[I] do
begin
  readln (F2, Line, TypeSusp[n[I,J]);
  readln (F2, Line, RoadArm[I,J], ProfileCode[I,J]);
  readln (F2, Line, WheelRadius[I,J]);
  readln (F2, Line, WheelWeight[I,J]);
  readln (F2, Line, XSpring[I,J], YSpring[I,J], ZSpring[I,J]);
  readln (F2, Line, SpringLength[I,J], SpringConst[I,J]);
  readln (F2, Line, BumpStop[I,J], BumpConst[I,J]);
  readln (F2);
  readln (F2);
end;
end;

(  If there is an extra line of data in the vehicle data file
|  beyond what the program expected, the program assumes this is
|  equilibrium data that has previously been computed. The program
|  therefore aborts. An exit message is displayed. After the user
|  hits any keyboard key, the cursor is reset, and the screen is
|  cleared.
)

if not eof (F2) then
begin
  sound (523);
  delay (120);
  nosound;

```

```

    textcolor (FColour3 + 128);
    gotoxy (2, 1);
    write ('Process aborted:');
    gotoxy (2, 2);
    textcolor (FColour3);
    write ('Equilibrium data exists in ' + FName1 + '.');
    repeat until keypressed;
    window (1, 1, 80, 25);
    textbackground (BColour1);
    textcolor (FColour1);

    CursorSize (CursorSmall, true);

    clrscr;

    exit;
end;

( If no previously computed equilibrium data is detected, then
| control information is read from the control data file, and
| procedure Equilibrium is called to begin computation.
)

readln (F3, Accuracy, Iterate);

for I:=1 to 4 do
begin
    Terr[I]^Interval := 100;
    Terr[I]^Limit := 1;
    Terr[I]^Y[0]^0 := 0;
    Terr[I]^Y[0]^1 := 0;
end;

Equilibrium (Accuracy, SprungMass, IPitch, IRoll,
    EqnNum, Iterate, NumberOfTracks,
    TypeVehicle,
    NumberOfTrackedWheels,
    UnitWeight, Elasticity, TrackLength,
    Sprocket,
    SpridlerRadius, XSpridler, YSpridler,
    ZSpridler, RoadArm, WheelRadius, WheelWeight, XSpring,
    YSpring, ZSpring, SpringLength, SpringConst,
    BumpStop, BumpConst, TypeSusp, X, XWheel,
    BColour1, BColour2, BColour3,
    FColour1, FColour2, FColour3, HorizonT);

close (F2);

```

```

close (F3);

( Keep the process status screen displayed until the user hits any
| keyboard key. Then reset screen and cursor, and append the
| equilibrium data to the vehicle data file.
)

repeat until keypressed;

textbackground (BColour1);
textcolor (FColour1);

CursorSize (CursorSmall, true);

clrscr;

append (F2);
writeln (F2, 'X      ', X[1], ' ', X[2], ' ', X[3]);

for I:=0 to NumberOfTracks do
begin
  if (NumberOfTrackedWheels[I] > 0) then
    write (F2, 'Y-', I, ' ');
  for J:=1 to NumberOfTrackedWheels[I] do
    write (F2, XWheel[I,J], ' ');
  if (NumberOfTrackedWheels[I] > 0) then
    writeln (F2);
end;

if (NumberOfTracks > 0) then
begin
  writeln (F2, 'Z      ', HorizonT);
end;

close (F2);

end.

```


APPENDIX 6**SOURCE CODE FOR RPM.EXE**

```
unit Data;
```

```
( (c) 1990 Michael Campbell Galway )
```

```
interface
```

```
( This unit initializes the different variable types used by the other
| units and subroutines. These types are made globally accessible.
)
```

```
type
```

```

Alphabet      = string [13];
AlphaArray    = array [1..4] of Alphabet;
AlphaMatrix   = array [0..4, 1..10] of Alphabet;
TrackArray    = array [1..4] of real;
SpridlerMatrix = array [1..4, 1..2] of real;
WheelMatrix   = array [0..4, 0..11] of real;
IWheelMatrix  = array [0..4, 0..11] of integer;
IWheelVector  = array [0..11] of integer;
WheelVector   = array [0..11] of real;
TrackVector   = array [0..4] of real;
Cheat         = array [0..100] of real;
TerrainData   = record
    Wheel      : array [1..12] of real;
    Interval   : real;
    Limit      : integer;
    Y          : array [0..12] of ^Cheat;
end;
ITrackVector  = array [0..4] of integer;
LookUp        = array [0..3, 0..14] of real;
CatMatrix     = array [1..4, 1..11, 1..2] of real;
EqnVector     = array [1..6] of real;
```

```

var NumberOfTracks, NumEquations, NumSteps : integer;
    SprungMass, IPitch, IRoll, StepSize : real;
    TypeVehicle : Alphabet;
    NumberOfTrackedWheels : ITrackVector;
    UnitWeight, TrackLength : TrackArray;
    Sprocket : AlphaArray;
    SpridlerRadius, XSpridler, YSpridler, ZSpridler : SpridlerMatrix;
    RoadArm, WheelRadius, WheelWeight, XSpring, YSpring, ZSpring,
        SpringLength, SpringConst, BumpStop, BumpConst, XDamp, YDamp,
        ZDamp, DampConst, VMax : WheelMatrix;
    TypeSuspns : AlphaMatrix;
    Vel : real;
    Terr : array [1..4] of ^TerrainData;
    ProfileCode : IWheelMatrix;

    F20 : text;
```

```
implementation
```

```
begin
  assign (F20, 'g:wheel.dat');
  rewrite (F20);
end.
```

```

unit Environ;

( This is a library of procedures and functions used with the interface
| of the RPM program and its processors. These routines are specific
| to the IBM PC environment. Some of the routines are from the
| following source:
|
|   Edwards, C.C. Advanced Techniques in Turbo Pascal. Sybex,
|   San Francisco. 1987.
|
| The other routines are inspired by similar routines in Edwards.
)

interface

    uses CRT, DOS;

    type
        BorderType = (single, double, none);
        CursorType = (CursorSmall, CursorLarge, CursorInvisible);

    var
        BooleanMono : boolean;

        procedure Borders (X1, Y1, X2, Y2 : byte; Index : integer);
        procedure InitWindowInfo;
        procedure CursorSize (Size : CursorType; BooleanMono : boolean);

        procedure OpenWindow (X1, Y1, X2, Y2 : byte; BColour, FColour : byte;
                               BorderStyle : BorderType);

implementation

    procedure Borders (X1, Y1, X2, Y2 : byte; Index : integer);

    ( This procedure draws either single or double borders around windows
    | on the screen. The character arrays and some of the variable names
    | have been borrowed from Edwards, but the algorithm is different.
    |
    | H.....chosen horizontal line character.
    | Horizontal...choice of horizontal line characters.
    | I.....counter.
    | Index.....no border, or single or double borders.
    | LowerLeft....choice of lower left corner characters.
    | LowerRight...choice of lower right corner characters.
    | UpperLeft....choice of upper left corner characters.
    | UpperRight...choice of upper right corner characters.
    | V.....chosen vertical line character.

```

```

|   Vertical.....choice of vertical line characters.
|   X1.....starting column.
|   X2.....ending column.
|   Y1.....starting row.
|   Y2.....ending row.
|
|
)

```

```
const
```

```

  UpperLeft : array [1..2] of char = (#218, #201);
  UpperRight : array [1..2] of char = (#191, #187);
  LowerLeft : array [1..2] of char = (#192, #200);
  LowerRight : array [1..2] of char = (#217, #188);
  Vertical : array [1..2] of char = (#179, #186);
  Horizontal : array [1..2] of char = (#196, #205);

```

```
var
```

```

  I : byte;
  H, V : char;

```

```
begin
```

```
  if (Index = 0) then exit;
```

```
  H := Horizontal[Index];
```

```
  V := Vertical [Index];
```

```
  gotoxy (1, 1);
```

```
  write (UpperLeft[Index]);
```

```
  gotoxy (X2 - X1 + 1, 1);
```

```
  write (UpperRight[Index]);
```

```
  gotoxy (1, Y2 - Y1 + 1);
```

```
  write (LowerLeft[Index]);
```

```
  gotoxy (X2 - X1 + 1, Y2 - Y1 + 1);
```

```
  write (LowerRight[Index]);
```

```
  for I:=X1+1 to X2-1 do
```

```
    begin
```

```
      gotoxy (I - X1 + 1, 1);
```

```
      write (H);
```

```
      gotoxy (I - X1 + 1, Y2 - Y1 + 1);
```

```
      write (H);
```

```
  end;
```

```
  for I:=Y1+1 to Y2-1 do
```

```
    begin
```

```
      gotoxy (1, I - Y1 + 1);
```

```
      write (V);
```

```
      gotoxy (X2 - X1 + 1, I - Y1 + 1);
```

```
      write (V);
```

```

    end;
end;

procedure OpenWindow (X1, Y1, X2, Y2 : byte; BColour, FColour : byte;
                     BorderStyle : BorderType);

var
    Index : integer;

begin
    textcolor (FColour);
    textbackground (BColour);

    if (BorderStyle = none) then Index := 0;
    if (BorderStyle = single) then Index := 1;
    if (BorderStyle = double) then Index := 2;

    Window (X1, Y1, X2 + 1, Y2);
    clrscr;

    Borders (X1, Y1, X2, Y2, Index);
end;

procedure InitWindowInfo;

var
    Regs : Registers;

begin
    intr ($11, Regs);
    BooleanMono := (lo (Regs.AX) and $30) = $30;
end;

procedure CursorSize (Size : CursorType; BooleanMono : boolean);

const
    CursorValues : array [0..3] of integer = ($0607, $0007, $0C00, $0000);

var
    Regs : Registers;

begin
    Regs.AX := $0100;
    if Size = CursorInvisible then
        Regs.CX := $2607

```

```
    else  
      Regs.Cx := CursorValues[ord (BooleanMono) *2 +ord (Size)];  
      intr ($10, Regs);  
    end;
```

```
begin  
end.
```

```

unit Filter;

( (c) 1990 Michael Campbell Galway
)

interface

type
    QuintQuadX = array [1..5, 1..4] of real;

procedure FilterSetUp (StepSize : real;
                       CutOffFreq : real;
                       var Cof : QuintQuadX);
procedure LowPassFilter (Xin : real;
                        var YOut : real;
                        var Cof : QuintQuadX);

implementation

procedure FilterSetUp (StepSize : real;
                       CutOffFreq : real;
                       var Cof : QuintQuadX);

( This procedure sets up the parameters required for the four-pole
  | Butterworth filter. It is called once before the first call to
  | the low-pass filter. The code is adapted from an amalgam of
  | two sources: an absorbed power program written by the Vehicle
  | Mobility Section (VMS) of the Defence Research Establishment
  | Suffield (DRES), an agency of the Canadian Department of National
  | Defence; and a digital filtering routine from the VehDyn II module
  | of the NATO Reference Mobility Model (NRMH) developed by the
  | Waterways Experiment Station (WES), an agency of the United States
  | Department of the Army.
  |
  | ABut.....filter parameter.
  | Ang.....filter parameter.
  | BB0.....filter parameter.
  | BB1.....filter parameter.
  | BB2.....filter parameter.
  | BB3.....filter parameter.
  | BB4.....filter parameter.
  | CBut.....filter parameter.
  | Cof.....filter parameter.
  | CSQ.....filter parameter.
  | CutOffFreq...cut off frequency of low pass filter (Hz).
  | I.....counter.
  | J.....counter.

```



```

| SampleRate....number of samples per second.
| StepSize.....constant time interval between samples.
| VR.....filter parameter.
}

var
  Ang, ABut, BB0, BB1, BB2, BB3, BB4, CBut, CSQ, SampleRate, VR : real;
  I, J : integer;

begin
  SampleRate := 1 / StepSize;
  VR := CutOffFreq / (0.5 * SampleRate);
  A..g := 0.5 * VR * pi;
  CBut := cos (Ang) / sin (Ang);

  BB0 := 1.0;
  BB1 := 2.6131259;
  BB2 := 3.4142136;

  CSQ := sqr (CBut);

  BB3 := 2.6131259;
  BB4 := 1.0;
  ABut := BB0 + BB1 * CBut + BB2 * CSQ + BB3 * CBut * CSQ + BB4 * sqr (CSQ);
  Cof[1,3] := 1 / ABut;
  Cof[2,3] := 4 / ABut;
  Cof[3,3] := 6 / ABut;
  Cof[4,3] := Cof[2,3];
  Cof[5,3] := Cof[1,3];
  Cof[5,4] := 1;
  Cof[1,4] := (4.0 + 2 * BB1 * CBut - 2 * BB3 * CBut * CSQ
    - 4 * BB4 * sqr (CSQ)) / ABut;
  Cof[2,4] := (6.0 - 2 * BB2 * CSQ + 6 * BB4 * sqr (CSQ)) / ABut;
  Cof[3,4] := (4.0 - 2 * BB1 * CBut + 2 * BB3 * CBut * CSQ
    - 4 * BB4 * sqr (CSQ)) / ABut;
  Cof[4,4] := (1.0 - BB1 * CBut + BB2 * CSQ - BB3 * CBut * CSQ
    + BB4 * sqr (CSQ)) / ABut;

  for I:=1 to 2 do
    for J:=1 to 5 do
      Cof[J,I] := 0;

end;

procedure LowPassFilter (XIn : real;
                          var YOut : real;

```

```

var Cof : QuintQuad();

( This procedure implements a four pole low pass filter. It is used
| by RPM to prevent aliasing caused by outputting data at a lower
| frequency than it is generated. The code was adapted from a FORTRAN
| routine developed by the Waterways Experiment Station (WES), an
| agency of the United States Department of the Army, for the
| VehDyn II module of the NATO Reference Mobility Model (NRMN).
|
| Cof.....filter parameters.
| I.....counter.
| J.....counter.
| T.....intermediate value.
| Xin.....unfiltered input value.
| YOut....filtered output value.
)

var
  I, J : integer;
  T : real;

begin
  J := round (Cof[5,4]);
  T := Xin * Co:[5,3];
  for I:=1 to 4 do
    begin
      T := T +Cof[J,1] *Cof[I,3] -Cof[J,2] *Cof[I,4];
      J := J + 1;
      if (J > 5) then J := 1;
    end;
    YOut := T;
    Cof[J,2] := T;
    Cof[J,1] := Xin;
    Cof[5,4] := J;
  end;

end.

```

```

unit DiffEqns;

( (c) 1990 Michael Campbell Galwey
)

interface

uses crt, Data, Environ, Filter, Sup_Proc, SuspProc, TerrProc, TrakProc;

procedure Derive (DeltaT, SprungMass, IPitch, IRoll, Rr, Ra, Time,
    Vel : real;
    NumberOfTracks : integer;
    NumberOfTrackedWheels : ITrackVector;
    UnitWeight, Elasticity, TrackLength : TrackArray;
    Sprocket : AlphaArray;
    SpridlerRadius, XSpridler, YSpridler,
        ZSpridler : SpridlerMatrix;
    RoadArm, WheelRadius, WheelWeight, XSpring,
        YSpring, ZSpring, SpringLength, SpringConst,
        BumpStop, BumpConst, XDamp, YDamp, ZDamp,
        DampConst, VMax : WheelMatrix;
    TypeSusp : AlphaMatrix;
    var Y, Dy : EqnVector;
    var DampLength : WheelMatrix);

procedure RungeKutta (StepSize, SprungMass, IPitch, IRoll, Rr, Ra,
    Vel : real;
    NumberOfTracks, NumEquations, PrintOut : integer;
    NumberOfTrackedWheels : ITrackVector;
    UnitWeight, Elasticity, TrackLength : TrackArray;
    Sprocket : AlphaArray;
    SpridlerRadius, XSpridler, YSpridler,
        ZSpridler : SpridlerMatrix;
    RoadArm, WheelRadius, WheelWeight, XSpring,
        YSpring, ZSpring, SpringLength, SpringConst,
        BumpStop, BumpConst, XDamp, YDamp, ZDamp,
        DampConst, VMax : WheelMatrix;
    TypeSusp : AlphaMatrix;
    var Y : EqnVector;
    var DampLength : WheelMatrix;
    var F5 : text;
    BColour2, FColour2, BColour3, FColour3 : byte);

implementation

procedure Derive (DeltaT, SprungMass, IPitch, IRoll, Rr, Ra, Time,
    Vel : real;
    NumberOfTracks : integer;
    NumberOfTrackedWheels : ITrackVector;

```

```

UnitWeight, Elasticity, TrackLength : TrackArray;
Sprocket : AlphaArray;
SpridlerRadius, XSpridler, YSpridler,
  ZSpridler : SpridlerMatrix;
RoadArm, WheelRadius, WheelWeight, XSpring,
  YSpring, ZSpring, SpringLength, SpringConst,
  BumpStop, BumpConst, XDamp, YDamp, ZDamp,
  DampConst, VMax : WheelMatrix;
TypeSusp : AlphaMatrix;
var Y, Dy : EqnVector;
var DampLength : WheelMatrix);

```

```

( This procedure calculates the forces and moments acting on
| the vehicle at a given time and vehicle position.
|

```

```

| Angle.....angle of upper track at supports.
| ASpringX.....absolute spring-hull coordinate.
| ASpringY.....absolute spring-hull coordinate.
| BounceForce.....heave force.
| BumpConst.....bumpstop spring constant.
| BumpStop.....compression to bumpstop.
| DampConst.....damper constant.
| DampForceX.....horizontal damping force.
| DampForceY.....vertical damping force.
| DampLength.....previous damper length.
| DeltaSprung.....weight of off-ground wheels.
| DeltaT.....time increment.
| DriveTension.....track tension due to drive torque.
| Dummy.....dummy variable.
| Dummy1.....dummy variable.
| Dummy2.....dummy variable.
| Dummy3.....dummy variable.
| Dy.....differential variables.
| Elasticity.....track linear elasticity.
| Flag......flag for off-ground wheels.
| FrontTrackAngle.....track angle at front of wheel.
| GroundWheelContact...point of ground-wheel contact.
| GroundWheelSlope.....slope at ground-wheel contact point.
| Gx.....returned by equation solver.
| Gy.....returned by equation solver.
| G2.....returned by equation solver.
| Heave.....vertical displacement of cg.
| I.....counter.
| IPitch.....pitch mass moment of inertia about cg.
| IRoll.....roll mass moment of inertia about cg.
| J.....counter.
| LowerSpridlerAngle...track angle from ground to spridler.
| LowerTrackLength.....track length on ground.
| N.....number of track catenaries.

```

```

| NumberOfTrackedWheels
|         .....number of wheels in a given track.
| NumberOfTracks.....number of tracks on a vehicle.
| NumGroundWheels.....number of wheels on the ground.
| Pitch.....pitch displacement about cg.
| PitchMoment.....pitch moment about cg.
| R.....ground reaction on wheel.
| Ra.....aerodynamic resistance.
| RearTrackAngle.....track angle at rear of wheel.
| RoadArm.....roadarm length/caster angle.
| Roll.....roll displacement about cg.
| RollMoment.....roll moment about cg.
| Rr.....rolling resistance.
| SpridlerRadius.....idler/sprocket radius.
| SpringConst.....stiffness of spring.
| SpringForceX.....horizontal spring force.
| SpringForceY.....vertical spring force.
| SpringLength.....compressed spring length/angle.
| Sprocket.....flag for location of drive sprocket.
| SprungMass.....sprung mass.
| Support.....flag to indicate roadwheels supporting track.
| SuspReactX.....total horizontal suspension reaction.
| SuspReactY.....total vertical suspension reaction.
| SuspTorque.....suspension reaction torque.
| Time.....elapsed time.
| TotTension.....total tension in upper track.
| TrackLength.....total track length.
| TypeSuspns.....torsion bar or helical spring suspension.
| T1.....known track tension.
| T2.....unknown track tension.
| UnitWeight.....weight per unit length of track.
| Vel.....vehicle velocity.
| VMax.....blow-off velocity of damper.
| WheelRadius.....roadwheel radius.
| WheelWeight.....roadwheel weight.
| Xa.....absolute (global) coordinate.
| XDamp.....hull-damper coordinate.
| Xr.....relative (local) coordinate.
| XSpridler.....spridler centre coordinate.
| XSpring.....spring-hull coordinate.
| XWheel.....roadwheel centre coordinate.
| Y.....independent variable.
| Ya.....absolute (global) coordinate.
| YDamp.....hull-damper coordinate.
| Yr.....relative (local) coordinate.
| YSpridler.....spridler centre coordinate.
| YSpring.....spring-hull coordinate.
| YWheel.....roadwheel centre coordinate.
| Za.....absolute (global) coordinate.

```

```

| ZDamp.....hull-damper coordinate.
| Zr.....relative (local) coordinate.
| ZSpridler.....spridler centre coordinate.
| ZSpring.....spring-hull coordinate.
| ZWheel.....roadwheel centre coordinate.
|
}

var
  I, J, NumGroundWheels           : integer;

  ASpringX, ASpringY, BounceForce, DeltaSprung : real;
  CosRoll, CosPitch, SinRoll, SinPitch         : real;
  Dummy, Dummy1, Dummy2, Dummy3             : real;
  Gx, Gy, Gz, Heave, Pitch, PitchMoment, R, Roll : real;
  RollMoment, T1, T2, Xa, Ya, Za, Xr, Yr, Zr   : real;

  DampForceX, DampForceY, FrontTrackAngle      : WheelMatrix;
  GroundWheelContact, GroundWheelSlope         : WheelMatrix;
  RearTrackAngle, SpringForceX, SpringForceY    : WheelMatrix;
  SusReactX, SusReactY, SusTorque, XWheel       : WheelMatrix;
  YWheel, ZWheel                               : WheelMatrix;
  XASpring, YASpring, ZASpring                 : WheelMatrix;

  N                                           : ITrackVector;
  DriveTension, LowerTrackLength             : TrackArray;
  LowerSpridlerAngle                         : SpridlerMatrix;
  Angle, TotTension                         : CatMatrix;
  Support, Flag                             : IWheelMatrix;

label 10, 20;

begin

( Set coordinates to values from Runge Kutta routine. )

  Heave := Y[1];
  Pitch := Y[2];
  Roll  := Y[3];
  CosRoll := cos (Roll);
  SinRoll := sin (Roll);
  CosPitch := cos (Pitch);
  SinPitch := sin (Pitch);

( Find wheel positions. )

  for I:=0 to NumberOfTracks do
    begin

```

```

for J:=1 to NumberOfTrackedWheels[I] do
  begin
(  Find absolute coordinates of spring-hull attachment points.  )

    Coord ('R->A', Heave, CosRoll, SinRoll, CosPitch, SinPitch,
      Time, Vel, XASpring[I,J], YASpring[I,J], ZASpring[I,J],
      XSpring[I,J], YSpring[I,J], ZSpring[I,J]);

(  Lateral position of the spring-hull attachment point must shift
|  as the vehicle rolls, but the wheel is constrained to remain
|  directly below it in order to avoid camber thrust which would
|  produce a yaw moment.
)

    ZWheel[I,J] := ZASpring[I,J];

    Terrain (Heave, Roll, Pitch, Time, Vel, WheelRadius[I,J],
      XASpring[I,J], YASpring[I,J], ZASpring[I,J],
      RoadArm[I,J], TypeSusp[I,J],
      GroundWheelContact[I,J], GroundWheelSlope[I,J],
      XWheel[I,J], YWheel[I,J], Flag[I,J],
      SpringLength[I,J], BumpStop[I,J], ProfileCode[I,J]);
(  writeln (F20, Time, XWheel[I,J], YWheel[I,J]);
end;

(  For roadwheels which are determined to be off the ground, the
|  horizontal ground-wheel contact point and the slope at the
|  ground-wheel contact point are still required in order to
|  calculate the lower track length. These are set respectively
|  to the lower tangency point with the next rearward wheel
|  (except for the rearmost wheel, which uses the next foremost wheel
|  for these calculations) and to the slope of the lower tangent
|  between the two wheels. Note that this is not done for
|  untracked wheels.
)

  if (I > 0) then
    for J:=1 to NumberOfTrackedWheels[I] do
      if (Flag[I,J] = 1) then
        if (J <> NumberOfTrackedWheels[I]) then
          CircleCircleTangent (WheelRadius[I,J],
            WheelRadius[I,J+1], XWheel[I,J],
            XWheel[I,J+1], YWheel[I,J],
            YWheel[I,J+1],
            GroundWheelContact[I,J], Dummy1,
            Dummy2, Dummy3,
            GroundWheelSlope[I,J])
        else
          CircleCircleTangent (WheelRadius[I,J-1],
            WheelRadius[I,J], XWheel[I,J-1],
            XWheel[I,J], YWheel[I,J-1],
            YWheel[I,J],
            GroundWheelContact[I,J], Dummy1,
            Dummy2, Dummy3,
            GroundWheelSlope[I,J])
        end
      end
    end
  end
end

```

[illegible]


```

        Damper (Xa, Ya, XWheel[I,J], YWheel[I,J], DeltaT, VMax[I,J],
                DampConst[I,J], DampLength[I,J], DampForceX[I,J],
                DampForceY[I,J]);
    end;
end;

if TypeSusp[I,J] = 'helical' then
    begin
        Spring (XASpring[I,J], YASpring[I,J], XWheel[I,J],
                YWheel[I,J], SpringConst[I,J],
                BumpConst[I,J], SpringLength[I,J], BumpStop[I,J],
                'helical', SpringForceX[I,J], SpringForceY[I,J]);
        if DampConst[I,J] <> 0 then
            Damper (XASpring[I,J], YASpring[I,J], XWheel[I,J], YWheel[I,J], DeltaT, VMax[I,J],
                    DampConst[I,J], DampLength[I,J], DampForceX[I,J],
                    DampForceY[I,J]);
        end;
    end;
( If the damping constant for a given suspension assembly is non-zero,
| then that assembly is assumed to have a damper, and damping forces
| are calculated. Otherwise, damping forces are set to zero.
)

    if (Flag[I,J] = 1) then
        begin
            DampForceX[I,J] := 0;
            DampForceY[I,J] := 0;
        end;

    if (DampConst[I,J] = 0) then
        begin
            DampForceX[I,J] := 0;
            DampForceY[I,J] := 0;
        end;
    end;
end;

( Calculate the forces due to upper track sag. )

if NumberOfTracks > 0 then
    UpperTrack (Heave, CosRoll, SinRoll, CosPitch, SinPitch,
                Time, Vel, NumberOfTracks,
                LowerTrackLength, TrackLength, UnitWeight, Elasticity,
                WheelRadius, XWheel, YWheel, NumberOfTrackedWheels,
                SpridlerRadius, XSpridler, YSpridler,
                ZSpridler, Angle,
                TotTension, Support, N);

( The coefficients and constants for the set of linear simultaneous

```

```

| equations that defines roadwheel static equilibrium are now
| known. A Gauss-Seidel method with pivoting is used to invert the
| equations and solve them. Note that the solution routine is called
| separately for non-tracked wheels since their equations are
| set up differently from those for tracked wheels.
|
}

I := 0;

{ Set the weight added to the sprung mass by suspended roadwheels
| initially to zero.
|
}

NumGroundWheels := NumberOfTrackedWheels[I];

{ Decrement the number of roadwheels on the ground by the number of
| roadwheels known to be off the ground. Increment the sprung mass
| by the mass of these roadwheels.
|
}

if (NumGroundWheels <> 0) then
begin
  DeltaSprung := 0;
  for J:=1 to NumberOfTrackedWheels[I] do
    if (Flag[I,J] = 1) then
      begin
        NumGroundWheels := NumGroundWheels - 1;
        DeltaSprung := DeltaSprung + WheelWeight[I,J];
      end;

  if (NumGroundWheels = 0) then
    begin
      Dy[1] := Y[4];
      Dy[2] := Y[5];
      Dy[3] := Y[6];
      Dy[4] := -9.81;
      Dy[5] := 0;
      Dy[6] := 0;
      exit;
    end;
end;

for J:=1 to NumberOfTrackedWheels[I] do
begin
  if (Flag[I,J] = 0) then
    begin
      T2 := 0;
      Pivot (I, XWheel[I,J], YWheel[I,J], XASpring[I,J],
        YASpring[I,J], 0, DampForceX[I,J], DampForceY[I,J],
        0, SpringForceX[I,J], Ra, Rr, SpringForceX[I,J],
        SpringForceY[I,J], GroundWheelSlope[I,J], T2,

```

```

WheelWeight[I,J], NumGroundWheels, TypeSusp[I,J],
'rear', Gx, Gy, G2, T1, R);

( If the ground reaction is less than zero, then there must be a
| tensile force being supported by the wheel. This is not physically
| possible, and thus the wheel must be assumed to be off the ground.
| Reaction forces are set to zero, and the sprung mass is increased
| by the weight of the wheel. Otherwise, suspension forces are
| interpreted according to the suspension type.
)

    if R < 0 then
    begin
        DeltaSprung := DeltaSprung + WheelWeight[I,J];
        SuspReactX[I,J] := 0;
        SuspReactY[I,J] := 0;
        SuspTorque[I,J] := 0;
    end else
    begin
        if TypeSusp[I,J] = 'torsion' then
        begin
            SuspTorque[I,J] := 0;
            SuspReactX[I,J] := Gx;
            SuspReactY[I,J] := Gy;
        end;
        if TypeSusp[I,J] = 'helical' then
        begin
            SuspTorque[I,J] := G2;
            SuspReactX[I,J] := Gx + DampForceX[I,J]
                               + SpringForceX[I,J];
            SuspReactY[I,J] := Gy + DampForceY[I,J]
                               + SpringForceY[I,J];
        end;
    end;
end else
begin

( If the wheel has been determined to be off the ground by terrain-
| wheel geometry, treat it the same as a wheel pulled off the ground
| by suspension forces.
)

        SuspReactX[I,J] := 0;
        SuspReactY[I,J] := 0;
        SuspTorque[I,J] := 0;
    end;
end;

( The equation solving code for tracked wheels follows. Note that it
| is repeated for front and rear sprocket drive vehicles. Front

```

```

| sprocket vehicles must be solved from the rearmost roadwheel
| forward, while rear sprocket vehicles must be solved from the
| foremost wheel backwards. This is because the drive tension is
| initially unknown, whereas the tension at the idler is known from
| track sag considerations.
|
|
}
  for I:=1 to NumberOfTracks do
    begin
10:   NumGroundWheels := NumberOfTrackedWheels[I];

  ( Decrement the number of roadwheels on the ground by the number of
  | roadwheels known to be off the ground. Increment the sprung mass
  | by the mass of these roadwheels.
  |
  )

    DeltaSprung := 0;
    for J:=1 to NumberOfTrackedWheels[I] do
      if (Flag[I,J] = 1) then
        begin
          NumGroundWheels := NumGroundWheels - 1;
          DeltaSprung := DeltaSprung + WheelWeight[I,J];
        end;

    if (NumGroundWheels = 0) then
      begin
        Dy[1] := Y[4];
        Dy[2] := Y[5];
        Dy[3] := Y[6];
        Dy[4] := -9.81;
        Dy[5] := 0;
        Dy[6] := 0;
        exit;
      end;

    if Sprocket[I] = 'Front' then
      begin

  ( The first known tension is the tension supported by the rear idler.
  |
  )
        T2 := TotTension[I,N[I],2];

  ( Solve the equations from the back of the vehicle to the front.
  |
  )
        for J:=NumberOfTrackedWheels[I] downto 1 do
          begin

  ( Don't solve for wheels that are known to be off the ground.
  |
  )

```

```

if (Flag[I,J] = 0) then
  begin
    Pivot (I, XWheel[I,J], YWheel[I,J], XASpring[I,J],
           YASpring[I,J], RearTrackAngle[I,J],
           DampForceX[I,J], DampForceY[I,J],
           FrontTrackAngle[I,J], SpringForceX[I,J], Ra, Rr,
           SpringForceX[I,J], SpringForceY[I,J],
           GroundWheelSlope[I,J], T2, WheelWeight[I,J],
           NumGroundWheels, TypeSusp[I,J], Sprocket[I],
           Gx, Gy, G2, T1, R);
    if R < 0 then
      begin
        Flag[I,J] := 1;
        DeltaSprung := DeltaSprung + WheelWeight[I,J];
        SuspReactX[I,J] := 0;
        SuspReactY[I,J] := 0;
        SuspTorque[I,J] := 0;
        NumGroundWheels := NumGroundWheels - 1;

( Restart the analysis from the back wheel with the number of
| roadwheels on the ground decremented by one.
)

        goto 10;
      end else
        begin
          SuspReactX[I,J] := Gx;
          SuspReactY[I,J] := Gy;
          T2 := T1;
          if TypeSusp[I,J] = 'torsion' then
            SuspTorque[I,J] := 0;
          if TypeSusp[I,J] = 'helical' then
            begin
              SuspTorque[I,J] := G2;
              SuspReactX[I,J] := SuspReactX[I,J] + DampForceX[I,J]
                                + SpringForceX[I,J];
              SuspReactY[I,J] := SuspReactY[I,J] + DampForceY[I,J]
                                + SpringForceY[I,J];
            end;
          end;
        end else
          begin
( If the wheel has been determined to be off the ground by terrain-
| wheel geometry, treat it the same as a wheel pulled off the ground
| by suspension forces.
)

            SuspReactX[I,J] := 0;
            SuspReactY[I,J] := 0;
            SuspTorque[I,J] := 0;

```

```

        end;
    end;

( The drive tension has been determined.)

    DriveTension[I] := T1
end;

    if Sprocket[I] = 'Rear' then
        begin
            DeltaSprung := 0;
( The first known tension is the tension supported by the front idler.
)
            T2 := TotTension[I,1,1];

( Solve the equations from the front of the vehicle to the back.
)
            for J:=1 to NumberOfTrackedWheels[I] do
                begin

( Don't solve for wheels that are known to be off the ground.
)
                    if (Flag[I,J] = 0) then
                        begin
                            Pivot (I, XWheel[I,J], YWheel[I,J], XASpring[I,J],
                                YASpring[I,J], RearTrackAngle[I,J],
                                DampForceX[I,J], DampForceY[I,J],
                                FrontTrackAngle[I,J], SpringForceX[I,J], Ra, Rr,
                                SpringForceX[I,J], SpringForceY[I,J],
                                GroundWheelSlope[I,J], T2, WheelWeight[I,J],
                                NumGroundWheels, TypeSusp[I,J], Sprocket[I], Gx,
                                Gy, G2, T1, R);

( Check whether the reaction force is tensile (physically
| impossible) or compressive.
)
                            if R < 0 then
                                begin
                                    Flag[I,J] := 1;
                                    DeltaSprung := DeltaSprung + WheelWeight[I,J];
                                    SuspReactX[I,J] := 0;
                                    SuspReactY[I,J] := 0;
                                    SuspTorque[I,J] := 0;
                                    NumGroundWheels := NumGroundWheels - 1;

( Restart the analysis from the front wheel with the number of
| roadwheels on the ground decremented by one.
)
                                goto 10;

```

```

end else
  begin
    SuspReactX[I,J] := Gx;
    SuspReactY[I,J] := Gy;
    T2 := T1;
    if TypeSusp[I,J] = 'torsion' then
      SuspTorque[I,J] := 0;
    if TypeSusp[I,J] = 'helical' then
      begin
        SuspTorque[I,J] := G2;
        SuspReactX[I,J] := SuspReactX[I,J] +DampForceX[I,J]
          +SpringForceX[I,J];
        SuspReactY[I,J] := SuspReactY[I,J] +DampForceY[I,J]
          +SpringForceY[I,J];
      end;
    end;
  end else
    begin
      ( If the wheel has been determined to be off the ground by terrain-
      | wheel geometry, treat it the same as a wheel pulled off the ground
      | by suspension forces.
      )
      SuspReactX[I,J] := 0;
      SuspReactY[I,J] := 0;
      SuspTorque[I,J] := 0;
    end;
  end;

  ( The drive tension has been determined. )

  DriveTension[I] := T1
  end;
end;

( Assess hull forces, beginning with the total force in vertical
| (heave) direction on the hull.
)

BounceForce := 0;
for I:=0 to NumberOfTracks do
  begin
    for J:=1 to NumberOfTrackedWheels[I] do
      begin
        ( If the suspension is of the helical type, the vertical suspension
        | reaction is the sum of damping and spring forces. Suspensions of

```

```

| the torsion bar type do not have damping forces included in the
| vertical suspension reaction forces. These vertical forces must
| be summed.
)
    BounceForce := BounceForce + SuspReactY[I,J];
    if TypeSusp[I,J] = 'torsion' then
        begin
            BounceForce := BounceForce + DampForceY[I,J];
        end;
    end;

    if (I >= 1) then
        begin
( Add the vertical component of track tension attributable to drive
| sprocket tension.
)
            if Sprocket[I]='Rear' then
                BounceForce := BounceForce
                    + DriveTension[I]
                    * sin (LowerSpridlerAngle[I,2])
                    - TotTension[I,1,1]
                    * sin (LowerSpridlerAngle[I,1]);

            if Sprocket[I]='Front' then
                BounceForce := BounceForce
                    + TotTension[I,1,1]
                    * sin (LowerSpridlerAngle[I,2])
                    - DriveTension[I]
                    * sin (LowerSpridlerAngle[I,1]);

        end;
    end;

( Add the downward force due to the weight of the hull. )

    BounceForce := BounceForce - (SprungMass + DeltaSprung) * 9.81;

( Find the total pitch moment acting on the hull. )

    PitchMoment := 0;
    for I:=0 to NumberOfTracks do
        begin
            for J:=1 to NumberOfTrackedWheels[I] do
                begin
( Vertical suspension reactions and horizontal suspension reactions

```



```

| should be weighted according to their moment arms and summed about
| the vehicle c.g. Suspension reaction torque may be added directly.
)

PitchMoment := PitchMoment + SuspReactY[I,J]
               *(XASpring[I,J] -Vel *Time);
PitchMoment := PitchMoment + SuspReactX[I,J]
               *(YASpring[I,J] -Heave);
PitchMoment := PitchMoment + SuspTorque[I,J];

if TypeSusp[I,J] ='torsion' then
begin

( Torsion bar type suspensions must be treated separately to add
| their damping forces and the torsion bar wind-up torque.
)

Coord ('R->A', Heave, CosRoll, SinRoll, CosPitch, SinPitch,
Time, Vel, Xa, Ya, Za,
XDamp[I,J], YDamp[I,J], ZDamp[I,J]);
PitchMoment := PitchMoment +DampForceY[I,J]
               *(Xa -Vel *Time);
PitchMoment := PitchMoment +DampForceX[I,J] *(Ya -Heave);
PitchMoment := PitchMoment + SpringForceX[I,J];
end;
end;

if (I >= 1) then
begin

( If the vehicle has tracks, add the contributions of the track
| tensions and off the drive tensions.
)

Coord ('R->A', Heave, CosRoll, SinRoll, CosPitch, SinPitch,
Time, Vel, Xa, Ya, Za,
XSpridler[I,1], YSpridler[I,1], ZSpridler[I,1]);

if Sprocket[I]='Rear' then
PitchMoment := PitchMoment
               -TotTension[I,1,1]
               *sin (LowerSpridlerAngle[I,1])
               *(Xa -Vel *Time +SpridlerRadius[I,1]
               *sin (LowerSpridlerAngle[I,1]))
               -TotTension[I,1,1]
               *cos (LowerSpridlerAngle[I,1])
               *(Ya -Heave -SpridlerRadius[I,1]
               *cos(LowerSpridlerAngle[I,1]));

if Sprocket[I]='Front' then
PitchMoment := PitchMoment
               -DriveTension[I]

```

```

* sin (LowerSpridlerAngle[I,1])
*(Xe -Vel *Time +SpridlerRadius[I,1]
* sin (LowerSpridlerAngle[I,1]))
-DriveTension[I]
*cos (LowerSpridlerAngle[I,1])
*(Ye -Heave -SpridlerRadius[I,1]
*cos (LowerSpridlerAngle[I,1]));

Coord ('R->A', Heave, CosRoll, SinRoll, CosPitch, SinPitch,
Time, Vel, Xe, Ye, Za,
XSpridler[I,2], YSpridler[I,2], ZSpridler[I,2]);

if Sprocket[I]='Rear' then
PitchMoment := PitchMoment
+DriveTension[I]
* sin (LowerSpridlerAngle[I,2])
*(Xe -Vel *Time -SpridlerRadius[I,2]
* sin (LowerSpridlerAngle[I,2]))
+DriveTension[I]
*cos (LowerSpridlerAngle[I,2])
*(Ye -Heave -SpridlerRadius[I,2]
*cos (LowerSpridlerAngle[I,2]));

if Sprocket[I]='Front' then
PitchMoment := PitchMoment
+TotTension[I,N[I],2]
* sin (LowerSpridlerAngle[I,2])
*(Xe -Vel *Time -SpridlerRadius[I,2]
* sin (LowerSpridlerAngle[I,2]))
+TotTension[I,N[I],2]
*cos (LowerSpridlerAngle[I,2])
*(Ye -Heave -SpridlerRadius[I,2]
*cos (LowerSpridlerAngle[I,2]));

end;

end;

( Find the total roll moment acting on hull. )

RollMoment := 0;
for I:=0 to NumberOfTracks do
begin
for J:=1 to NumberOfTrackedWheels[I] do
begin

( Vertical suspension reactions and horizontal suspension reactions

```

```

| should be weighted according to their moment arms and summed about
| the vehicle c.g.
)

```

```

RollMoment := RollMoment - SuspReactY[I,J] *cos (Pitch)
              *ZASpring[I,J];
RollMoment := RollMoment - SuspReactX[I,J] *sin (-Pitch)
              *ZASpring[I,J];

```

```

if TypeSusp[I,J] = 'torsion' then
begin

```

```

( Torsion bar type suspensions must be treated separately to add
| their damping forces and the torsion bar wind-up torque.
)

```

```

Coord ('R->A', Heave, CosRoll, SinRoll, CosPitch, SinPitch,
Time, Vel, Xa, Ya, Za,
XDamp[I,J], YDamp[I,J], ZDamp[I,J]);
RollMoment := RollMoment -DampForceY[I,J] *cos (Pitch) *Za;
RollMoment := RollMoment -DampForceX[I,J] *sin (-Pitch) *Za;

```

```

end;
end;

```

```

if (I >= 1) then
begin

```

```

( If the vehicle has tracks, add the contributions of the track
| tensions and off the drive tensions.
)

```

```

Coord ('R->A', Heave, CosRoll, SinRoll, CosPitch, SinPitch,
Time, Vel, Xa, Ya, Za,
XSpridler[I,1], YSpridler[I,1], ZSpridler[I,1]);

```

```

if Sprocket[I]='Rear' then
RollMoment := RollMoment
              +TotTension[I,1,1]
              *sin (LowerSpridlerAngle[I,1]) *Za
              *cos (Pitch)
              +TotTension[I,1,1]
              *cos (LowerSpridlerAngle[I,1]) *Za
              *sin (-Pitch);

```

```

if Sprocket[I]='Front' then
RollMoment := RollMoment
              +DriveTension[I]
              *sin (LowerSpridlerAngle[I,1]) *Za
              *cos (Pitch)
              +DriveTension[I]

```

```

        *cos (LowerSpridlerAngle[I,1]) *Za
        *sin (-Pitch);

Coord ('R->A', Heave, CosRoll, SinRoll, CosPitch, SinPitch,
      Time, Vel, Xa, Ya, Za,
      XSpridler[I,2], YSpridler[I,2], ZSpridler[I,2]);

if Sprocket[I]='Rear' then
  RollMoment := RollMoment
    -DriveTension[I]
    *sin (LowerSpridlerAngle[I,2]) *Za
    *cos (Pitch)
    -DriveTension[I]
    *cos (LowerSpridlerAngle[I,2]) *Za
    *sin (-Pitch);

if Sprocket[I]='Front' then
  RollMoment := RollMoment
    -TotTension[I,N[I],2]
    *sin (LowerSpridlerAngle[I,2]) *Za
    *cos (Pitch)
    -TotTension[I,N[I],2]
    *cos (LowerSpridlerAngle[I,2]) *Za
    *sin (-Pitch);

end;

end;

( Calculate heave, pitch, and roll accelerations from the force
| and moments, and assign these to the differential variables.
)

Dy[4] := BounceForce / (SprungMass + DeltaSprung);
if (Dy[4] < -9.81) then
  begin
    Dy[4] := -9.81;
    PitchMoment := 0;
    RollMoment := 0;
  end;
Dy[5] := PitchMoment / IPitch;
Dy[6] := RollMoment / IRoll;

( In order to prevent cumulative error causing a slow roll etc,
| set displacements and accelerations below a certain threshold
| to zero.
)

for I:=4 to 6 do

```

```

begin
  if (abs (Y[I]) <= 0.0001) then Y[I] := 0;
  if (abs (Dy[I]) <= 0.01) then Dy[I] := 0;
end;

( Make final assignment of variables before return to Runge-Kutta
| routine.
)
Dy[1] := Y[4];
Dy[2] := Y[5];
Dy[3] := Y[6];

( writeln (Time, 'Bounce, Pitch, Roll:', Dy[4], Dy[5], Dy[6]);
  writeln (y[1], y[2], y[3]);
end;

procedure RungeKutta (StepSize, SprungMass, IPitch, IRoll, Rr, Ra,
  Vel : real;
  NumberOfTracks, NumEquations, PrintOut : integer;
  NumberOfTrackedWheels : ITrackVector;
  UnitWeight, Elasticity, TrackLength : TrackArray;
  Sprocket : AlphaArray;
  SpridlerRadius, XSpridler, YSpridler,
    ZSpridler : SpridlerMatrix;
  RoadArm, WheelRadius, WheelWeight, XSpring,
    YSpring, ZSpring, SpringLength, SpringConst,
    BumpStop, BumpConst, XDamp, YDamp, ZDamp,
    DampConst, VMax : WheelMatrix;
  TypeSusp : AlphaMatrix;
  var Y : EqnVector;
  var DampLength : WheelMatrix;
  var F5 : text;
  BColour2, FColour2, BColour3, FColour3 : byte);

( This subroutine solves a system of six differential equations
| using the Classical Runge Kutta method. The algorithm was
| translated into Pascal from a FORTRAN program authored by
| J.E. Neilson found in the following publication:
|
| 94.265 Course Notes : Computer Methods in Engineering. Dept. of
| Systems Engineering, Faculty of Engineering, Carleton
| University, Ottawa, Ont. 1983.
|
| Dy.....derivative of dependent variable.
| I.....index variable.
| J.....index variable.
| NumEquations...number of differential equations to solve.
| NumSteps.....total time of solution divided by stepsize.

```

```

| StepSize.....stepsize of solution.
| Work1.....working vector.
| Work2.....working vector.
| X.....independent variable (time).
| Y.....dependent variable.
)

```

```

var
  Count, I, J, NumSteps           : longint;
  AHeave, APitch, ARoll, CutOffFreq, Front, Heave : real;
  Pitch, Rear, Roll, VHeave, VPitch, VRoll, X    : real;
  Dy, Work1, Work2                : EqnVector;
  ACof, BCof, CCof, DCof, ECof, FCof, GCof, HCof, ICof : QuintQuadX;

```

```

begin
  Y[1] := Y[1];
  Y[4] := 0;
  Y[5] := 0;
  Y[6] := 0;
  Dy[1] := 0;
  Dy[2] := 0;
  Dy[3] := 0;
  Dy[4] := 0;
  Dy[5] := 0;
  Dy[6] := 0;

```

```

( Set up process status box. )

```

```

  OpenWindow (13, 4, 68, 16, BColour2, FColour2, double);
  gotoxy (25, 2);
  write ('STATUS:');
  gotoxy (7, 5);
  write ('Elapsed time      :');
  gotoxy (7, 7);
  write ('Elapsed distance  :');
  gotoxy (7, 9);
  write ('Percent completed :');

```

```

  OpenWindow (40, 8, 48, 8, BColour3, FColour3, none);
  OpenWindow (40, 10, 48, 10, BColour3, FColour3, none);
  OpenWindow (40, 12, 48, 12, BColour3, FColour3, none);
  window (1, 1, 80, 25);

```

```

( Find the foremost and rearmost spring attachment points. )

```

```

  Front := -999999.;
  Rear  := 999999.;

```

```

for I:=0 to NumberOfTracks do
  begin
    if (NumberOfTrackedWheels[I] <> 0) then
      begin
        if (XSpring[I,1] >= Front) then
          Front := XSpring[I,1];
        if (XSpring[I,NumberOfTrackedWheels[I]] <= Rear) then
          Rear := XSpring[I,NumberOfTrackedWheels[I]];
        end;
      end;
  end;

( Set the number of integration steps equal to the number of
| time intervals of StepSize required to traverse the terrain
| profile PLUS 1.5 times the c.g. to foremost spring distance
| PLUS 1.5 times the c.g. to rearmost spring distance, at the
| speed Vel. The extra distance added to the terrain profile
| is used to ensure that the vehicle doesn't start or stop on
| the terrain (starting on the terrain could produce very high
| initial hull forces that would be totally erroneous.
)
NumSteps := trunc ((Terr[1]^Limit +1.5 *Front -Rear)
                  *Terr[1]^Interval /Vel /StepSize) +1;

( Start the vehicle a distance equal to 1.5 times the c.g. to
| foremost spring distance to avoid starting on the profile.
)
X := -Front *1.5 /Vel;

(X := 1.767;
y[1] := 1.0177;
y[2] := -0.0729;
y[4] := -0.3326;
y[5] := -0.6484;
dy[1] := -0.3326;
dy[2] := -0.6484;
dy[4] := -4.8369;
dy[5] := -0.9377;)

( Initialize output interval count. )

Count := PrintOut;

( Initialize low-pass filters. )

CutOffFreq := 1 /(2 *PrintOut *StepSize);

FilterSetUp (StepSize, CutOffFreq, ACof);
FilterSetUp (StepSize, CutOffFreq, BCof);

```

```

FilterSetUp (StepSize, CutOffFreq, CCof);
FilterSetUp (StepSize, CutOffFreq, DCof);
FilterSetUp (StepSize, CutOffFreq, ECof);
FilterSetUp (StepSize, CutOffFreq, FCof);
FilterSetUp (StepSize, CutOffFreq, GCof);
FilterSetUp (StepSize, CutOffFreq, HCof);
FilterSetUp (StepSize, CutOffFreq, ICof);

for I := 1 to NumSteps do
begin
( Write updated information to the process status box on screen. )

    gotoxy (40, 8);
    write (' ', X :5 :2, ' ');
    gotoxy (40, 10);
    write (' ', (X*Vel) :5 :2, ' ');
    gotoxy (40, 12);
    write (' ', (I /NumSteps *100) :5 :2, ' % ');

    Count := Count + 1;

( Filter all data in preparation for print sampling. )

    LowPassFilter (Y[1], Heave, ACof);
    LowPassFilter (Y[2], Pitch, BCof);
    LowPassFilter (Y[3], Roll, CCof);
    LowPassFilter (Y[4], VHeave, DCof);
    LowPassFilter (Y[5], VPitch, ECof);
    LowPassFilter (Y[6], VRoll, FCof);
    LowPassFilter (Dy[4], AHeave, GCof);
    LowPassFilter (Dy[5], APitch, HCof);
    LowPassFilter (Dy[6], ARoll, ICof);

( Check whether it is time to output information. )

    if (Count > PrintOut) then
    begin
(      writeln (F5, X      :8:4, ' ',
                    X*Vel  :8:4, ' ',
                    Y[1]   :8:4, ' ',
                    Y[2]   :8:4, ' ',
                    Y[3]   :8:4, ' ',
                    Y[4]   :8:4, ' ',
                    Y[5]   :8:4, ' ',
                    Y[6]   :8:4, ' ',
                    Dy[4]  :8:4, ' ',
                    Dy[5]  :8:4, ' ',
                    Dy[6]  :8:4);)

```



```

        writeln (F5, X      :8:4, ' ',
                X*Vel :8:4, ' ',
                Heave :8:4, ' ',
                Pitch :8:4, ' ',
                Roll  :8:4, ' ',
                VHeave :8:4, ' ',
                VPitch :8:4, ' ',
                VRoll  :8:4, ' ',
                AHeave :8:4, ' ',
                APitch :8:4, ' ',
                ARoll  :8:4);

        Count := 1;
    end;
(
    writeln (X, Y[1], Y[2], Y[3], Dy[4], Dy[5], Dy[6]);
    writeln (X, Y[1], Dy[1], Dy[4]);
    writeln;)

    for J := 1 to NumEquations do
        begin
            Work1[J] := Y[J];
        end;

    Derive (StepSize, SprungMass, IPitch, IRoll, Rr, Ra, X,
            Vel, NumberOfTracks,
            NumberOfTrackedWheels, UnitWeight, Elasticity,
            TrackLength, Sprocket, SpridlerRadius, XSpridler,
            YSpridler, ZSpridler, RoadArm, WheelRadius,
            WheelWeight, XSpring, YSpring, ZSpring, SpringLength,
            SpringConst, BumpStop, BumpConst, XDamp, YDamp, ZDamp,
            DampConst, VMax, TypeSusp, Y, Dy, DampLength);

    for J := 1 to NumEquations do
        begin
            Y[J] := Work1[J] + StepSize * Dy[J] / 2.0;
            Work2[J] := Dy[J];
        end;

    X := X + StepSize / 2.0;
    Derive (StepSize, SprungMass, IPitch, IRoll, Rr, Ra, X,
            Vel, NumberOfTracks,
            NumberOfTrackedWheels, UnitWeight, Elasticity,
            TrackLength, Sprocket, SpridlerRadius, XSpridler,
            YSpridler, ZSpridler, RoadArm, WheelRadius,
            WheelWeight, XSpring, YSpring, ZSpring, SpringLength,
            SpringConst, BumpStop, BumpConst, XDamp, YDamp, ZDamp,
            DampConst, VMax, TypeSusp, Y, Dy, DampLength);

    for J := 1 to NumEquations do

```

```

begin
  Y[J] := Work1[J] +StepSize *Dy[J] /2.0;
  Work2[J] := Work2[J] +2.0 *Dy[J];
end;

Derive (StepSize, SprungMass, IPitch, IRoll, Rr, Ra, X,
        Vel, NumberOfTracks,
        NumberOfTrackedWheels, UnitWeight, Elasticity,
        TrackLength, Sprocket, SpridlerRadius, XSpridler,
        YSpridler, ZSpridler, RoadArm, WheelRadius,
        WheelWeight, XSpring, YSpring, ZSpring, SpringLength,
        SpringConst, BumpStop, BumpConst, XDamp, YDamp, ZDamp,
        DampConst, VMax, TypeSuspn, Y, Dy, DampLength);

for J := 1 to NumEquations do
  begin
    Y[J] := Work1[J] +StepSize *Dy[J];
    Work2[J] := Work2[J] +2.0 *Dy[J];
  end;

  X := X +StepSize /2.0;
  Derive (StepSize, SprungMass, IPitch, IRoll, Rr, Ra, X,
          Vel, NumberOfTracks,
          NumberOfTrackedWheels, UnitWeight, Elasticity,
          TrackLength, Sprocket, SpridlerRadius, XSpridler,
          YSpridler, ZSpridler, RoadArm, WheelRadius,
          WheelWeight, XSpring, YSpring, ZSpring, SpringLength,
          SpringConst, BumpStop, BumpConst, XDamp, YDamp, ZDamp,
          DampConst, VMax, TypeSuspn, Y, Dy, DampLength);

  for J := 1 to NumEquations do
    begin
      Y[J] := Work1[J] +StepSize *(Work2[J] +Dy[J]) /6.0;
    end;

  end;
end;

begin

  end.

```

(c) 1990 Michael Campbell Galwey

The vehicle is modelled with three degrees of freedom: vertical heave; pitch about the cg with a fixed attitude axis as reference; and roll about the cg with the vehicle longitudinal axis as reference. The wheels are constrained to follow the terrain, thus providing a displacement dependent exciting force upon the hull. The differential equations of motion are solved using a fourth order Runge Kutta method.

```

Accuracy.....criterion for computation of equilibrium data.
BColour1.....background colour of process status screen.
BColour2.....background colour of process status box.
BColour3.....background colour of solution boxes.
BumpConst.....spring constant of bump stop.
BumpStop.....roadarm susp'ns: angle above horizontal of
                bumpstop.
                helical susp'ns: spring compression to bumpstop.
Colour.....0 if monochrome, 1 if colour adapter.
EqnNum.....number of differential equations (6).
FColour1.....text colour.
FColour2.....text colour.
FColour3.....text colour.
FName1.....vehicle data file name.
FName2.....control file name.
F1.....meta-control file.
F2.....vehicle data file.
F3.....process control file.
GraphDriver.....type of graphics driver detected.
GraphMode.....graphics mode detected.
I.....a counter.
IPitch.....pitch mass moment of inertia of vehicle hull.
IRoll.....roll mass moment of inertia of vehicle hull.
Iterate.....number of iterations permitted for convergence.
J.....a counter
NumberOfTrackedWheels
                .....number of wheels in a track.

```

```

| NumberOfTracks...number of tracks on vehicle.
| RoadArm.....roadarm susp'ns: length of roadarm.
|               helical susp'ns: castor angle from vertical.
| SpridlerRadius...radii of sprocket and idler.
| SpringConst.....spring constant.
| SpringLength.....roadarm susp'ns: free angle of roadarm above
|                   x axis.
|                   helical susp'ns: free length of spring.
| Sprocket.....'front' if front drive, else 'rear'.
| SprungMass.....mass of sprung body.
| TrackLength.....total tracklength per track.
| TypeSuspns.....'torsion' if roadarm suspension, else 'helical'.
| TypeVehicle.....'truck', 'light armoured vehicle', 'heavy armoured
|                   vehicle', 'automobile', or 'motorcycle'.
| UnitWeight.....weight of track per unit length.
| WheelRadius.....radii of roadwheels.
| WheelWeight.....weight of roadwheel assemblies.
| X.....independent variables in differential equations.
| XWheel.....equilibrium position of wheel centres.
| XSpridler.....longitudinal position of spridlers from cg.
| XSpring.....longitudinal positions of spring/hull
|               attachment pts.
| YSpridler.....vertical position of spridlers from cg.
| YSpring.....vertical positions of spring/hull attachment pts.
| ZSpridler.....lateral positions of spridlers from cg.
| ZSpring.....lateral positions of spring/hull attachment pts.
|
| *****)

```

```

uses Crt, Data, Environ, DiffEqns, Graph, Sup_Proc, SuspProc,
    TrakProc;

```

```

var
    Accuracy, GVM, IPitch, IRoll, Ra, Rr      : real;
    SprungMass, X1, X2                        : real;
    Y                                           : EqnVector;
    Elasticity, UnitWeight, TrackLength        : TrackArray;
    SpridlerRadius, XSpridler, YSpridler, ZSpridler : SpridlerMatrix;
    RoadArm, WheelRadius, WheelWeight, XSpring : WheelMatrix;
    YSpring, ZSpring, SpringLength, SpringConst : WheelMatrix;
    BumpStop, BumpConst, XWheel                : WheelMatrix;
    XDamp, YDamp, ZDamp, DampLength            : WheelMatrix;

```

```

Colour, EqnNum, GraphDriver, GraphMode, Iterate : integer;
I, J, K, NumberOfTracks, PrintOut           : integer;
Auto, NumberOfProfiles                       : integer;
BColour1, BColour2, BColour3                 : byte;
FColour1, FColour2, FColour3                 : byte;
DiffRURadii, NumberOfTrackedWheels          : ITrackVector;

F1, F2, F3, F4, F5                           : text;
FName1, FName2, FName3, FName4               : string;
Line                                           : string(7);
TypeVehicle                                  : Alphabet;
Sprocket                                      : AlphaArray;
TypeSuspension                               : AlphaMatrix;

label 10;
begin

( Initialize window information (for display). Determine whether
| monochrome graphics adapter is present or if user requested
| monochrome override (by using command line parameter 'C'). Set
| the colour flag Colour appropriately to 1 or zero.
|
| Remove the cursor so it won't appear on the process status screen.
)

InitWindowInfo;
detectgraph (GraphDriver, GraphMode);
Colour := 0;
if (paramcount > 0) then Colour := 1;
if ((BooleanMono = true) or (graphdriver = 7)) then Colour := 2;

CursorSize (CursorInvisible, BooleanMono);

( Set background colours and text colours for the process status
| screen. Note that colours are automatically set to black or white
| for monochrome displays. Colours are:
|
| 0....black
| 1....blue
| 3....cyan
| 7....gray
| 15...white
)

case Colour of
  0 : begin
    BColour1 := 1;
    BColour2 := 3;
    BColour3 := 7;

```

```

        FColour1 := 15;
        FColour2 := 0;
        FColour3 := 0;
    end;
1 : begin
    BColour1 := 0;
    BColour2 := 0;
    BColour3 := 15;

    FColour1 := 15;
    FColour2 := 15;
    FColour3 := 0;
    end;
2 : begin
    BColour1 := 0;
    BColour2 := 0;
    BColour3 := 15;

    FColour1 := 15;
    FColour2 := 7;
    FColour3 := 0;
    end;
end;

( Set up the background for the process status screen (title
| line, exit message box).
)
textcolor (FColour1);
textbackground (BColour1);
clrscr;

gotoxy (20, 2);
write ('Ride Prediction Model - Main Processor');
gotoxy (20,3);
write (' (c) 1990 Michael Campbell Galway');
OpenWindow (8, 19, 72, 25, BColour1, FColour1, single);
gotoxy (2, 2);
write ('Process Message:');
textbackground (BColour3);
OpenWindow (10, 22, 69, 23, BColour3, FColour3, none);

textcolor (FColour3 + 128);
gotoxy (2, 1);
write ('Working:');
gotoxy (2, 2);
textcolor (FColour3);
write ('Reading vehicle data. ');

(goto 10;)
for i:=1 to 4 do

```

```

    begin
    new (Terr[I]);
    for J:=0 to 12 do
        new (Terr[I]^Y[J]);
    end;

(   Open standard meta-control file Meta.CNT, and read names of
|   vehicle data file and STATIC control file.
)

    assign (F1, 'Meta.CNT');
    reset (F1);

    readln (F1);
    readln (F1, FName1);
    readln (F1, FName2);
    readln (F1);
    readln (F1, FName3);
    readln (F1, FName4);

    assign (F2, FName1);
    assign (F3, FName2);
    assign (F4, FName3);
    assign (F5, FName4);

    reset (F2);
    reset (F3);
    reset (F4);
    rewrite (F5);

    EqnNum := 6;

(   Read in vehicle data.   )

    readln (F2);
    readln (F2, Line, TypeVehicle);
    readln (F2, Line, SprungMass, GVN);
    readln (F2, Line, IPitch);
    readln (F2, Line, IRoll);
    readln (F2, Line, NumberOfTracks, NumberOfProfiles);

    for J:=1 to NumberOfProfiles do
        begin
            read (F2, Line, DiffRWRadii[J]);
            if (DiffRWRadii[J] > 5) then halt;
            for I:=1 to DiffRWRadii[J] do
                read (F2, Terr[J]^Wheel[I]);
            readln (F2);
        end;
    end;

```

```

for I:=0 to NumberOfTracks do
begin
  readln (F2, Line, NumberOfTrackedWheels[I]);
end;

for I:=1 to NumberOfTracks do
begin
  readln (F2, Line, UnitWeight[I], Elasticity[I]);
  readln (F2, Line, TrackLength[I]);
  readln (F2, Line, Sprocket[I]);
  readln (F2, Line, SpridlerRadius[I,1], SpridlerRadius[I,2]);
  readln (F2, Line, XSpridler[I,1], XSpridler[I,2]);
  readln (F2, Line, YSpridler[I,1], YSpridler[I,2]);
  readln (F2, Line, ZSpridler[I,1], ZSpridler[I,2]);
end;

for I:=0 to NumberOfTracks do
begin
  for J:=1 to NumberOfTrackedWheels[I] do
begin
  readln (F2, Line, TypeSusp[I,J]);
  readln (F2, Line, RoadArm[I,J], ProfileCode[I,J]);
  readln (F2, Line, WheelRadius[I,J]);
  readln (F2, Line, WheelWeight[I,J]);
  readln (F2, Line, XSpring[I,J], YSpring[I,J], ZSpring[I,J]);
  readln (F2, Line, SpringLength[I,J], SpringConst[I,J]);
  readln (F2, Line, BumpStop[I,J], BumpConst[I,J]);
  readln (F2, Line, XDamp[I,J], YDamp[I,J], ZDamp[I,J]);
  readln (F2, Line, DampConst[I,J], VMax[I,J]);
end;
end;

readln (F2, Line, Y[1], Y[2], Y[3]);

for I:=0 to NumberOfTracks do
begin
  if (NumberOfTrackedWheels[I] <> 0) then
begin
  read (F2, Line);
  for J:=1 to NumberOfTrackedWheels[I] do
    read (F2, XWheel[I,J]);
  readln (F2);
end;
end;

( Read in simulation control values. )

```



```

readln (F3, Accuracy, Iterate, StepSize, PrintOut, Vel, Auto);

gotoxy (2, 2);
write ('Reading terrain data.                                     ');

( Read in terrain profile and hubpath data. )

for K:=1 to NumberOfProfiles do
begin
  readln (F4);
  readln (F4, Terr[K]^Limit);

  read (F4, X1, Terr[K]^Y[0]^1);
  for I:=1 to DiffRWRadii[K]-1 do
    read (F4, Terr[K]^Y[I]^1);
  readln (F4, Terr[K]^Y[DiffRWRadii[K]]^1);

  for J:=2 to Terr[K]^Limit do
begin
  read (F4, X2, Terr[K]^Y[0]^J);
  for I:=1 to DiffRWRadii[K]-1 do
    read (F4, Terr[K]^Y[I]^J);
  readln (F4, Terr[K]^Y[DiffRWRadii[K]]^J);
end;

  end;

( Calculate constant interval between terrain data points. )

  Terr[K]^Interval := (X2 - X1) / (Terr[K]^Limit - 1);
  readln (F4);
end;

( Calculate aerodynamic and rolling resistances acting on the
| vehicle based on the given vehicle type.
| )
Resist (TypeVehicle, SprungMass, Vel, Ra, Rr);

( Calculate the initial length of the suspension dampers based
| on the vehicle's equilibrium position.
| )
InitLength (Y, NumberOfTracks, NumberOfTrackedWheels, XWheel,
            WheelRadius, XDamp, YDamp, ZDamp, DampLength);

gotoxy (2, 2);
write ('Solving equations of motion...                               ');

( Solve equations of motion. )

RungeKutta (StepSize, SprungMass, IPitch, IRoll, Rr, Ra,
            Vel, NumberOfTracks, EqnNum, PrintOut,

```

```

        NumberOfTrackedWheels,
        UnitWeight, Elasticity, TrackLength, Sprocket,
        SpridlerRadius, XSpridler, YSpridler, ZSpridler,
        RoadArm, WheelRadius, WheelWeight, XSpring, YSpring,
        ZSpring, SpringLength, SpringConst, BumpStop, BumpConst,
        XDamp, YDamp, ZDamp, DampConst, VMax, TypeSusp, Y,
        DampLength, F5, BColour2, FColour2, BColour3, FColour3);

close (F2);
close (F3);

sound (523);
delay (60);
nosound;
delay (60);
sound (587);
delay (60);
nosound;

textcolor (FColour3 + 128);
gotoxy (11, 22);
writeln ('Success:');
textcolor (FColour3);
gotoxy (11, 23);
write ('Simulation complete. Press any key to continue... ');

( Keep the process status screen displayed until the user hits any
| keyboard key. Then reset screen and cursor, and append the
| equilibrium data to the vehicle data file.
)

if (Auto = 1) then
    repeat until keypressed;

textbackground (BColour1);
textcolor (FColour1);

CursorSize (CursorSmall, true);

clrscr;

close (F5);
end.

```

```

unit Sup_Proc;

( (c) 1990 Michael Campbell Galway
)

interface

uses Data;

procedure Error (I : integer);
function ArcSinh (X : real) : real;
function Sgn (Number : real) : integer;
function Sinh (X : real) : real;
function Cosh (X : real) : real;
function Atan (X : real) : real;
function Acos (X : real) : real;
function Asin (X : real) : real;
function Tan (X : real) : real;
procedure Coord (AbsRel : Alphabet;
                 Heave, CosRoll, SinRoll, CosPitch, SinPitch,
                 Time, Vel : real;
                 var Xa, Ya, Za, Xr, Yr, Zr : real);
procedure Resist (TypeVehicle : Alphabet;
                 GrossWeight, Vel : real;
                 var AeroForce, RollForce : real);

implementation

procedure Error (I : integer);

( This procedure handles error situations by reporting problem and
  | stopping.
)

var
    Message : string;

begin
    Message := 'Error.';

    if (I = 1) then Message := 'Error 1: unable to compute lower tangent'
                               + ' between two wheels.';
    if (I = 2) then Message := 'Error 2: attempted to take arccosine of '
                               + 'number greater than one.';
    if (I = 3) then Message := 'Error 3: attempted to take arcsine of '
                               + 'number greater than one.';

    writeln;
    writeln ('!!!!!!! ERROR !!!!!!!!');

```

```

    writeln;
    writeln (Message);
    writeln;
    writeln ('Program halted.', 1);
    halt;
end;

```

```

function Tan (X : real) : real;

```

```

(   This function calculates the tangent of a number (Turbo Pascal has
|   no tangent function).
)

```

```

begin
    Tan := sin (X) /cos (X);
end;

```

```

function ArcSinh (X : real) : real;

```

```

(   This function calculates the inverse hyperbolic sine of a number.  )

```

```

begin
    ArcSinh := ln (X +sqrt (sqr(X) +1));
end;

```

```

function Sgn (Number : real) : integer;

```

```

(   Determines the algebraic sign of a real number.  )

```

```

var
    SigPnt : ^byte;
    SigBit : byte;

begin
    SigPnt := ptr (seg (Number), ofs (Number) + $05);
    SigBit := SigPnt^;
    Sgn := -((SigBit shr 7) shl 1) +1;
end;

```

```

function Sinh (X : real) : real;

```

```

(   This function finds the hyperbolic sine of a real number.  )

```

```

begin

```

```

    Sinh := 0.5 *(exp (X) -exp(-X));
end;

```

```

function Cosh (X : real) : real;

```

```

(   This function finds the hyperbolic cosine of a real number.   )

```

```

begin

```

```

    Cosh := 0.5 *(exp (X) +exp (-X));
end;

```

```

function Atan (X : real) : real;

```

```

(   This function finds the arctangent of a real number.   )

```

```

begin

```

```

    Atan := arctan (X);
end;

```

```

function Acos (X : real) : real;

```

```

(   This function returns the arccosine of a real number.   )

```

```

begin

```

```

    Acos := pi * 0.5;
    if (abs (X) > 1) then Error (2);
    if (X <> 0) then Acos := arctan (sqrt (1 -sqr (X)) /X);
end;

```

```

function Asin (X : real) : real;

```

```

(   This function returns the arcsine of a real number.   )

```

```

begin

```

```

    Asin := Sgn (X) *pi *0.5;
    if (abs (X) > 1) then exit;
    if (abs (X) <> 1) then Asin := arctan (X /sqrt (1 -sqr (X)));
end;

```

```

procedure Coord (AbsRel : Alphabet;
                Heave, CosRoll, SinRoll, CosPitch, SinPitch,
                Time, Vel : real;
                var Xa, Ya, Za, Xr, Yr, Zr : real);

(   This procedure converts global coordinates to local coordinates
|   and vice versa.
|
|   AbsRel...direction of conversion.
|   Heave....vertical displacement of vehicle cg from equilibrium.
|   Roll.....roll angle of vehicle.
|   Pitch....pitch angle of vehicle.
|   Time.....elapsed simulation time.
|   Vel.....vehicle velocity.
|   Xa.....absolute horizontal coordinate.
|   Xr.....relative horizontal coordinate.
|   Ya.....absolute vertical coordinate.
|   Yr.....relative vertical coordinate.
|   Za.....absolute lateral coordinate.
|   Zr.....relative lateral coordinate.
)

var
    ZrYr, XaVT, XaYa, YaHeave : real;
begin
    if AbsRel = 'R->A' then

(   If converting from relative to absolute coordinates, do the
|   following:
|
|   begin
|       ZrYr := Zr *SinRoll -Yr *CosRoll;
|       Xa := Xr *CosPitch +ZrYr *SinPitch +Vel *Time;
|       Ya := Xr *SinPitch -ZrYr *CosPitch +Heave;
|       Za := Yr *SinRoll +Zr *CosRoll;
|   end else

begin

(   If converting from absolute to relative coordinates, do the
|   following:
|
|   XaYa := (Xa -Vel *Time) * SinPitch +(Heave -Ya) *CosPitch;
|   Xr := (Xa -Vel *Time) *CosPitch +(Ya -Heave) *SinPitch;
|   Yr := -XaYa *CosRoll +Za *SinRoll;
|   Zr := XaYa *SinRoll +Za *CosRoll;
|   end;
end;
end;

```

```

procedure Resist (TypeVehicle : alphabet;
                 GrossWeight, Vel : real;
                 var AeroForce, RollForce : real);

( This procedure calculates motion resistance on the vehicle due to
| aerodynamic drag and rolling resistance. These motion resistance
| terms form part of the constraint equation used to determine track
| tension.
|
| AeroForce.....aerodynamic drag acting on the vehicle (newtons).
| Af.....frontal area of the vehicle (square metres).
| Alphabet.....character variable type.
| Cd.....vehicle drag coefficient.
| fR.....rolling resistance coefficient (ie rolling
|           resistance divided by gross vehicle weight).
| GrossWeight....gross vehicle weight (kg);
| Rho.....density of air at STP (1.29 kg/cubic metre).
| RollForce.....rolling resistance acting against vehicle
|               (newtons).
| TypeVehicle....vehicle type.
| Vel.....vehicle velocity (m/s).
)

var
  Af, Cd, fR, Rho : real;

begin
  if (TypeVehicle = 'light-tracked') then
    fR := 0.03 + 0.0017 * Vel;

  if (TypeVehicle = 'heavy-tracked') then
    fR := 0.038 + 0.0017 * Vel;

  if (TypeVehicle = 'automobile') then
    fR := 0.015;

  if (TypeVehicle = 'truck') then
    fR := 0.012;

  if (TypeVehicle = 'motorcycle') then
    fR := 0.015;

  RollForce := fR * GrossWeight;

```

```

if (TypeVehicle = 'light-tracked') then
begin
  Cd := 1.17;
  Af := 6;
end;

if (TypeVehicle = 'heavy-tracked') then
begin
  Cd := 1.17;
  Af := 8;
end;

if (TypeVehicle = 'automobile') then
begin
  Cd := 0.4;
  Af := 2.3;
end;

if (TypeVehicle = 'truck') then
begin
  Cd := 0.9;
  Af := 6;
end;

if (TypeVehicle = 'motorcycle') then
begin
  Cd := 1.8;
  Af := 0.9;
end;

rho := 1.29;
AeroForce := 0.5 *rho *Cd *Af *sqr (Vel);

end;

begin
end.

```



```

unit SuspProc;

( (c) 1990 Michael Campbell Galway
)

interface

uses Data, Sup_Proc;

procedure InitLength (Y : EqnVector;
                     NumberOfTracks : integer;
                     NumberOfTrackedWheels : ITrackVector;
                     XWheel, WheelRadius,
                     XDamp, YDamp, ZDamp : WheelMatrix;
                     var DampLength : WheelMatrix);
procedure Spring (X, Y, XWheel, YWheel, SpringConst, BumpConst, Length,
                  BumpStop : real;
                  TypeSusp : Alphabet;
                  var ForceX, ForceY : real);
procedure Damper (X, Y, XWheel, YWheel, DeltaT, VMax, DampConst : real;
                  var Length, ForceX, ForceY : real);
procedure Pivot (Wheeled : integer;
                 A, B, C, D, Delta, Dx, Dy, Gamma, M, Ra, Rr, Sx, Sy,
                 Theta, T2, W : real;
                 N : integer;
                 TypeSusp, Sprocket : Alphabet;
                 var Gx, Gy, GZ, T1, R : real);

implementation

procedure InitLength (Y : EqnVector;
                     NumberOfTracks : integer;
                     NumberOfTrackedWheels : ITrackVector;
                     XWheel, WheelRadius,
                     XDamp, YDamp, ZDamp : WheelMatrix;
                     var DampLength : WheelMatrix);

( This procedure determines the initial length of the dampers based
| on the equilibrium wheel positions found in the Equilibrium
| Processor, and on the hull orientation. InitLength is called once
| at the beginning of a simulation.
|
| DampLength.....initial length of damper.
| I.....counter.
| J.....counter.
| NumberOfTrackedWheels.....number of wheels in a track.
| NumberOfTracks.....number of tracks on a vehicle.

```

```

| WheelRadius.....radius of roadwheel.
| Xa.....absolute location of hull-damper
|           attachment.
| XDamp.....relative location of hull-damper location.
| XWheel.....absolute equilibrium location of
|           wheel centre.
| Y.....heave, roll, and pitch of hull.
| Ya.....absolute location of hull-damper
|           attachment.
| YDamp.....relative location of hull-damper location.
| Za.....absolute location of hull-damper
|           attachment.
| ZDamp .....relative location of hull-damper location.
|
)

var
  I, J      : integer;
  CosRoll, CosPitch, SinRoll, SinPitch, Xa, Ya, Za : real;

begin

  CosRoll := cos (Y[3]);
  SinRoll := sin (Y[3]);
  CosPitch := cos (Y[2]);
  SinPitch := sin (Y[2]);

  for I:=0 to NumberOfTracks do
    begin
      for J:=1 to NumberOfTrackedWheels[I] do
        begin

          ( Find absolute (global) coordinates of hull damper attachment. )

            Coord ('R->A', Y[I], CosRoll, SinRoll, CosPitch, SinPitch,
              0, 0, Xa, Ya, Za,
              XDamp[I,J], YDamp[I,J], ZDamp[I,J]);

          ( Calculate damper length based on absolute hull-damper attachment
          | point, and on absolute wheel centre location. (Note that wheel
          | centre height is assumed to be wheel radius since wheel is
          | incompressible.
          |
          )

            Damplength[I,J] := sqrt (sqr (Xa -XWheel[I,J])
              +sqr (Ya -WheelRadius[I,J]));

          end;
        end;
      end;
    end;
  end;
end;

```

```

procedure Spring (X, Y, XWheel, YWheel, SpringConst, BumpConst, Length,
    BumpStop : real;
    TypeSusp : alphabet;
    var ForceX, ForceY : real);

```

```

( This procedure calculates the forces generated by a spring.
|
| alphabet.....variable type.
| BumpConst.....spring constant of bumpstop.
| BumpStop.....for torsion bar, angle of bumpstop above horizontal;
|               for helical spring, distance to bumpstop from
|               spring end.
| ForceX... ..for torsion bar, spring torque; for helical spring,
|               horizontal component of spring force.
| ForceY.....for helical spring, vertical component of
|               spring force.
| Length.....for torsion bar, free angle of road arm above
|               horizontal; for helical spring, undeformed length
|               of spring.
| SpringAngle....angle of road-arm above the horizontal.
| SpringConst....spring constant.
| SpringForce....spring force developed.
| SpringLength...instantaneous length of helical spring.
| SpringTorque...torque developed by torsion bar.
| TypeSusp.....type of suspension (torsion bar or helical spring).
| X.....spring/hull horizontal attachment point.
| XWheel.....horizontal location of wheel hub.
| Y.....spring/hull vertical attachment point.
| YWheel.....vertical location of wheel hub.
|
)

```

```

var

```

```

    SpringAngle, SpringForce, SpringLength, SpringTorque : real;

```

```

begin

```

```

    if (TypeSusp = 'torsion') then

```

```

( For a torsion bar, find the deformed spring angle, and calculate
| the resultant torque. Evaluate whether the bumpstop has been
| contacted. If so, add the torque due to bumpstop compression.
|
)

```

```

    begin

```

```

        SpringAngle := atan ((Y - YWheel) / (X - XWheel));

```

```

        if ((SpringAngle > 0) and (Y > YWheel)) then

```

```

            SpringAngle := -pi + SpringAngle

```

```

else
    if ((SpringAngle < 0) and (Y < YWheel)) then
        SpringAngle := -pi + SpringAngle;
    SpringTorque := SpringConst *(SpringAngle -Length);
    if SpringAngle < BumpStop then
        SpringTorque := SpringTorque +BumpConst
            *(-BumpStop +SpringAngle);
    ForceX := SpringTorque;
(
    SpringTorque := 2 *SpringConst *(BumpStop -Length);
    if (ForceX < SpringTorque) then ForceX := SpringTorque;
    if (ForceX > 0) then ForceX := 0;)
end;

if (TypeSuspension = 'helical') then

(
    For a helical spring suspension, find the spring force, and check
    | whether the bumpstop has been contacted. If so, add its spring
    | force. Find the component forces in the x and y directions.
)

begin
    SpringLength := sqrt (sqr (X -XWheel) + sqr (Y -YWheel));
    if Y < YWheel then SpringLength := -SpringLength;
    SpringForce := SpringConst *(Length -SpringLength);
    if SpringLength < Length -BumpStop then
        SpringForce := SpringForce +BumpConst
            *(Length -BumpStop -SpringLength);
    if (X = XWheel) then
        ForceX := 0
    else
        ForceX := SpringForce /SpringLength *(X -XWheel);

    if (Y = YWheel) then
        ForceY := SpringForce
    else
        ForceY := SpringForce /SpringLength *(Y -YWheel);
end;
end;

procedure Damper (X, Y, XWheel, YWheel, DeltaT, VMax, DampConst : real;
    var Length, ForceX, ForceY : real);

(
    This procedure calculates the forces generated by a damper.
    |
    | DampConst....damping constant of damper.
    | DampForce....damping force developed by a damper.

```

```

| DampLength...instantaneous damper length.
| DampVel.....damper velocity.
| DeltaT.....integration time.
| ForceX.....horizontal component of damping force.
| ForceY.....vertical component of damping force.
| Length.....previous length of a telescopic damper.
| VMax.....blow-off velocity of damper.
| X.....telescopic damper/hull horizontal attachment point.
| XWheel.....horizontal location of wheel hub.
| Y.....telescopic damper/hull vertical attachment point.
| YWheel.....vertical location of wheel hub.
|
)

```

```
var
```

```
    DampForce, DampLength, DampVel : real;
```

```
begin
```

```

(   For a telescopic damper mounted between the vehicle hull and the
|   roadwheel hub, find damper velocity based on its previous length,
|   check that the blow-off velocity is not exceeded, calculate the
|   damping force, and decompose the force into components in the x and
|   y directions.
)

```

```

DampLength := sqrt (sqr (X -XWheel) +sqr (Y -YWheel));
if Y < YWheel then DampLength := -DampLength;
DampVel := (Length -DampLength) /DeltaT;
if (abs (DampVel) > VMax) then DampVel := sgn (DampVel) * VMax;
DampForce := DampConst * (DampVel);
ForceY := (Y -YWheel) /DampLength *DampForce;
ForceX := (X -XWheel) /abs (DampLength) *DampForce;
Length := DampLength;

```

```
end;
```

```
procedure Pivot (Wheeled : integer;
```

```

    A, B, C, D, Delta, Dx, Dy, Gamma, M, Ra, Rr, Sx, Sy,
    Theta, T2, W : real;
    N : integer;
    TypeSusp, Sprocket : alphabet;
    var Gx, Gy, G2, T1, R : real);

```

```

(   This procedure employs Gaussian elimination with pivotal
|   condensation to solve for the forces acting on the wheel. The
|   forces are:
|

```

- a. spring forces;
- b. damper forces;
- c. track tension;
- d. shear force at the wheel/ground interface;
- e. normal force at the wheel/ground interface;
- f. rolling resistance;
- g. aerodynamic drag; and
- h. wheel-assembly weight.

The equations used are:

- a. sum of forces in the x direction is zero;
- b. sum of forces in the y direction is zero;
- c. sum of moments in the xy plane is zero; and
- d. tractive effort in the x direction must be equal and opposite to motion resistance acting on the vehicle (normalized with respect to the number of wheels on the ground).

A.....horizontal location of wheel hub.
 Angle1.....track angle.
 Angle2.....track angle.
 B.....vertical location of wheel hub.
 C.....horizontal location of spring/hull attachment point.
 D.....vertical location of spring/hull attachment point.
 Delta.....rear track angle.
 Dx.....horizontal damper force on hull.
 Dy.....vertical damper force on hull.
 Gamma.....front track angle.
 Gx.....horizontal reaction force on hull.
 Gy.....vertical reaction force on hull.
 G1.....horizontal reaction or total reaction.
 G2.....vertical reaction or reaction moment.
 I.....index.
 Imax.....maximum index.
 J.....index.
 K.....index.
 M.....torsion bar torque.
 MX.....array of linear equation elements.
 Matrix.....variable type.
 Max.....value of pivot entry.
 N.....number of wheels in contact with ground.
 NumEqn.....number of simultaneous equations to be solved.
 R.....normal force at ground-wheel contact.
 Ra.....aerodynamic resistance acting on vehicle.
 Rr.....rolling resistance acting on vehicle.
 Sign.....short-cut.
 Sprocket...flags whether drive sprocket is at front or rear.
 Sx.....horizontal component of spring force.

4 of/de 4

MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS
STANDARD REFERENCE MATERIAL 1010a
(ANSI and ISO TEST CHART No. 2)

```

| Sy.....vertical component of spring force.
| Temp.....holding variable for moving matrix rows.
| Theta.....ground slope.
| TypeSusp..torsion bar or helical spring.
| T1.....track tension at wheel front.
| T2.....track tension at wheel rear.
| W.....wheel assembly weight.
| Wheeled....0 if no tracks, otherwise not 0.
|
)

type
  Matrix = array [1..4, 1..5] of real;

var
  I, Imax, J, K, NumEqn, Sign          : integer;
  AA, BB, CC, DD, EE, FF : real;
  Alpha, Beta, Angle1, Angle2, G1, Max, Temp : real;
  CosAlpha, CosTheta, SinAlpha, SinTheta : real;
  MX                                     : Matrix;

begin

  ( Assign values to linear equations matrix. )

  CosTheta := cos (Theta);
  SinTheta := sin (Theta);

  if Wheeled <> 0 then
    begin
      Sign := -1;

  ( If the drive sprocket is at the front, the wheel equations are
  | solved from the rear wheel to the front wheel.
  |
  | If the drive sprocket is at the rear, the wheel equations are
  | solved from the front wheel to the rear wheel.
  |
  )

    if (Sprocket = 'Front') then
      begin
        Sign := 1;
        Angle1 := Gamma;
        Angle2 := Delta;
      end else
        begin
          Sign := -1;
          Angle1 := Delta;

```



```

    Angle2 := Gamma;
end;

( Matrix initiation and solution differ somewhat between swinging
| arm and pillar type suspensions.
)

if (TypeSuspension = 'torsion') then
begin
(
    Beta := atan ((D - B) / (A - C));
    if ((Beta > 0) and (D > B)) then
        Beta := -pi + Beta
    else
        if ((Beta < 0) and (D < B)) then
            Beta := -pi + Beta;
    Gx := -Dx;
    Gy := (M / sqrt (sqr (A - C) +sqr (B - D)) -Gx *cos (Beta)) /sin (Beta);
    AA := -Sign *cos (Angle1);
    BB := SinTheta;
    CC := -(Rr +Ra) /N -Sign *T2 *cos (Angle2);
    DD := sin (Angle1);
    EE := CosTheta;
    FF := Dy +Gy +W +T2 *sin (Angle2);
    R := ((FF -CC *DD /AA) / (EE -BB *DD /AA));
    T1 := CC -R *BB /AA;)

(
    T1 := Sign *(M / (C -A) +Dx *(D -B) / (C -A) -Dy -(Rr +Ra) /N
        *(1 /tan (Theta) +(D -B) / (C -A)) -W +T2 /SinTheta
        *(1 +Sign *sin (Angle2) *SinTheta -Sign *(cos (Beta)
        *SinTheta))) *SinTheta / (1 -Sign *sin (Angle1)
        *SinTheta -Sign *(D -B) / (C -A) *cos (Angle1)
        *SinTheta);
    Gx := -Dx +(Rr +Ra) /N +Sign *T1 *cos (Angle1)
        -Sign *T2 *cos (Angle2);
    Gy := (Gx *(D -B) -M) / (C -A);
    R := (Sign *(T2 -T1) *CosTheta -(Rr +Ra) /N) /SinTheta;
    exit;)
    MX[1,1] := -1;
    MX[1,2] := 0;
    MX[1,3] := Sign *(cos (Angle1) -CosTheta);
    MX[1,4] := -SinTheta;
    MX[1,5] := Dx +Sign *T2 *(cos (Angle2) -CosTheta);

    MX[2,1] := 0;
    MX[2,2] := -1;
    MX[2,3] := sin (angle1) -Sign *SinTheta;
    MX[2,4] := CosTheta;
    MX[2,5] := Dy -T2 *(sin (Angle2) +Sign *SinTheta) +W;

```

```

MX[3,1] := -(B -D);
MX[3,2] := -(C -A);
MX[3,3] := 0;
MX[3,4] := 0;
MX[3,5] := M;

MX[4,1] := 0;
MX[4,2] := 0;
MX[4,3] := -Sign *CosTheta;
MX[4,4] := -SinTheta;
MX[4,5] := -Sign *T2 *CosTheta +(Rr -Ra) /N;

end;

if (TypeSusp = 'helical') then
begin
if (D = 0) then
Alpha := pi / 2
else
Alpha := atan ((A -C) /(D -B));
CosAlpha := cos (Alpha);
SinAlpha := sin (Alpha);
MX[1,1] := -CosAlpha;
MX[1,2] := 0;
MX[1,3] := Sign *(cos (Angle1) -CosTheta);
MX[1,4] := SinTheta;
MX[1,5] := Dx +Sx +Sign *T2 *(cos (Angle2) -CosTheta);

MX[2,1] := -SinAlpha;
MX[2,2] := 0;
MX[2,3] := sin (angle1) -Sign *SinTheta;
MX[2,4] := CosTheta;
MX[2,5] := Dy +Sy -T2 *(sin (Angle2) +Sign *SinTheta) +W;

MX[3,1] := sqrt (sqr (A-C) +sqr (B -D));
MX[3,2] := 1;
MX[3,3] := 0;
MX[3,4] := 0;
MX[3,5] := 0;

MX[4,1] := 0;
MX[4,2] := 0;
MX[4,3] := -Sign *CosTheta;
MX[4,4] := SinTheta;
MX[4,5] := -Sign *T2 *CosTheta +(Rr -Ra) /N;

end;
NumEqn := 4;
end;

```

```
(  Untracked suspensions have a smaller matrix since they have one
|  fewer unknown variables.
)
```

```
if Wheeled = 0 then
  begin
```

```
    if (TypeSusp = 'torsion') then
      begin
        (  Beta := atan ((D - B) / (A - C));
          if ((Beta > 0) and (D > B)) then
            Beta := -pi + Beta
          else
            if ((Beta < 0) and (D < B)) then
              Beta := -pi + Beta;
            )

        if (N = 0) then
          begin
            R := -1;
            exit;
          end;

        Gx := -Dx - (Rr + Ra) / N;
        Gy := (-Gx * (D - B) - M) / (C - A);
        R := (Gy + Dy + W) * CosTheta - (Rr + Ra) / N * SinTheta;
        exit;
      end;
```

```
    if (TypeSusp = 'helical') then
      begin
        if (D = B) then
          Alpha := pi / 2
        else
          Alpha := atan ((A - C) / (D - B));
        CosAlpha := cos (Alpha);
        SinAlpha := sin (Alpha);
        MX[1,1] := -CosAlpha;
        MX[1,2] := 0;
        MX[1,3] := SinTheta;
        MX[1,4] := Dx + Sx;

        MX[2,1] := -SinAlpha;
        MX[2,2] := 0;
        MX[2,3] := CosTheta;
        MX[2,4] := Dy + Sy + W;

        MX[3,1] := sqrt (sqr (A - C) + sqr (B - D));
        MX[3,2] := -1;
```

```

    MX[3,3] := 0;
    MX[3,4] := 0;

    end;
    NumEqn := 3;
end;

( Put augmented matrix of system of linear equations in row-echelon
| order.
)

for K:=1 to NumEqn do
    begin
        Max := 0;
        Imax := K;

    ( Find matrix row which has highest value for first non-zero element.
    | This element is the pivot entry.
    )

        for I:=K to NumEqn do
            begin
                if abs(MX[I,K]) > Max then
                    begin
                        Max := abs(MX[I,K]);
                        Imax := I;
                    end;
            end;

        ( Move pivot entry to the top of the rows tested. )

            if (MX[Imax, K] <> 0) then
                begin
                    Max := 1 / MX[Imax, K];
                    for I:=K to NumEqn+1 do
                        begin
                            Temp := MX[K,I];
                            MX[K,I] := MX[Imax,I];
                            MX[Imax,I] := Temp;
                            MX[K,I] := MX[K,I] * Max;
                        end;
                end;

            ( Reduce rows under pivot entry row by forcing zeros under the pivot
            | entry.
            )

                for I:=K+1 to NumEqn do
                    begin

```

```

    Temp := MX[I,K];
    for J:=K to NumEqn+1 do
        begin
            MX[I,J] := MX[I,J] -Temp *MX[K,J];
        end;
    end;
end;

( Solve by back substitution. )

if Wheeled = 0 then
begin
    if (MX[3,3] <> 0) then
        R := MX[3,4] / MX[3,3]
    else
        R := 0;
    G2 := MX[2,4] -R *MX[2,3];
    G1 := MX[1,4] -R *MX[1,3] -G2 *MX[1,2];
end;

if Wheeled <> 0 then
begin
    if (MX[4,4] <> 0) then
        R := MX[4,5] / MX[4,4]
    else
        R := 0;
    T1 := MX[3,5] -R *MX[3,4];
    G2 := MX[2,5] -R *MX[2,4] -T1 *MX[2,3];
    G1 := MX[1,5] -R *MX[1,4] -T1 *MX[1,3] -G2 *MX[1,2];
end;

if (TypeSusp = 'helical') then
begin
    Gx := G1 *CosAlpha;
    Gy := G1 *SinAlpha;
end else
begin
    Gx := G1;
    Gy := G2;
end;

end;

begin
end.

```

```

unit TerrProc;

( (c) 1990 Michael Campbell Galway
)

interface

uses Data, Sup_Proc;

procedure Terrain (Heave, Roll, Pitch, Time, Vel, WheelRadius,
                  XSpring, YSpring, ZSpring, RoadArm : real;
                  TypeSusp : Alphabet;
                  var GroundWheelContact, GroundWheelSlope,
                      XWheel, YWheel : real;
                  var Flag : integer;
                  SpringLength, BumpStop : real;
                  PCode : integer);

function Profile (x : real; PCode : integer) : real;

function WheelPath (X : real; PCode, WheelCode : integer) : real;

implementation

procedure Terrain (Heave, Roll, Pitch, Time, Vel, WheelRadius,
                  XSpring, YSpring, ZSpring, RoadArm : real;
                  TypeSusp : Alphabet;
                  var GroundWheelContact, GroundWheelSlope,
                      XWheel, YWheel : real;
                  var Flag : integer;
                  SpringLength, BumpStop : real;
                  PCode : integer);

( This procedure finds the wheel centre location in global terms
| for a given hull position and time. This is done by finding
| the intersection of the wheel centre locus (as defined by
| vehicle orientation and suspension geometry) with the hubpath
| locus (previously determined by the WheelPath Generation
| Processor). This is faster than finding the highest point of
| tangency between the wheel and the terrain. Analytical methods
| are used to find the root corresponding to the difference
| between the wheel centre and hubpath loci. On locating the wheel
| centre, the procedure then determines the point of ground-wheel
| tangency (found from the wheel radius and the normal to the
| hubpath) and the slope at the point of tangency. A swinging
| arm (torsion bar) suspension may not have an intersection
| point with the hubpath (as may a helical spring type suspension
| under unusual conditions) in which case the spring element is

```

```
| set to its free (uncompressed) position, thus defining the
| wheel centre locatirn, and a flag is set to indicate a wheel off
| the ground. For tracked vehicles, the point of tangency and
| the slope at the point of tangency for off-ground wheels are
| then computed within the Derive procedure.
```

```
|
| A.....peak allowable amplitude of roadarm motion.
| Accuracy.....criterion for convergence.
| BumpStop.....angle to bumpstop.
| F.....independent variable.
| Flag.....flags off-ground wheel.
| F1.....independent variable.
| GroundWheelContact...point of ground-wheel tangency.
| GroundWheelSlope.....angle at ground wheel tangent point.
| N.....feedback variable.
| Heave.....not used.
| I.....counter.
| Iterate.....maximum number of iterations to convergence.
| R.....gain variable; reduces gain.
| RoadArm.....roadarm length/caster angle.
| Roll.....not used.
| P.....gain variable; increases gain.
| Pitch.....hull pitch displacement.
| Sign.....defines direction to root.
| SpringLength.....free spring length/angle.
| S1.....root bypass check.
| S2.....root bypass check.
| Time.....not used.
| TypeSuspns.....torsion bar or helical spring suspension.
| Vel.....not used.
| WheelCode.....index to correct hubpath.
| WheelRadius.....roadwheel radius.
| X.....independent search variable.
| XSpring.....absolute location of hull-spring attachment.
| XWheel.....absolute location of wheel centre.
| X1.....independent search variable.
| YSpring.....absolute location of spring-hull attachment.
| YWheel.....absolute location of wheel centre.
| ZSpring.....not used.
|
| )
```

```
procedure WheelCoder (WheelRadius : real;
                     PCode : integer;
                     var WheelCode : integer);
```

```
( This procedure identifies which hubpath should be used for a given
| wheel, and returns an index (WheelCode) to the calling procedure.
```

```

|
| 1.....counter.
| Terr^.Wheel....wheel radii corresponding to hubpaths.
| WheelCode.....index to hubpath.
| WheelRadius....radius of wheel in question.
|
|
)

var
  I : integer;

begin
  for I:=1 to 5 do
    begin
      ( When the difference between the given wheel radius and a hubpath
      | wheel radius is zero, the correct hubpath has been identified.
      )
      if (WheelRadius-Terr[PCode]^Wheel[I] = 0) then
        begin
          WheelCode := I;
          exit;
        end;
      end;
    end;
  end;

var
  A, B, C, Intercept, Radical, Slope, X, X1, X2, Y1, Y2      : real;
  Inclination                                                : real;
  I, ICount, IStart, Irate, Sign, WheelCode                : integer;
  ExitFlag                                                    : boolean;

Label 20;

begin
  Flag := 0;

  ( Find out which hubpath to use for the given wheel radius. )

  WheelCoder (WheelRadius, PCode, WheelCode);

  if (TypeSuspension = 'torsion') then
    begin
      ICount := trunc ((XSpring + RoadArm) / Terr[PCode]^Interval) + 1;

      ExitFlag := false;
    end;

```



```

X := XSpring - RoadArm;

Sign := 1;
A := WheelPath (XSpring, PCode, WheelCode);
if (YSpring <= A) then
  Sign := -1;

if (X > 0) then
  IStart := trunc (X / Terr(PCode)^.Interval)
else
  IStart := trunc (X / Terr(PCode)^.Interval) - 1;

for I:=IStart to ICount do
  begin

    if (I <= 0) then
      begin
        Slope := 0;
        Intercept := WheelRadius;
      end else
      begin
        Y1 := WheelPath (Terr(PCode)^.Interval *(I - 1), PCode, WheelCode);
        Y2 := WheelPath (Terr(PCode)^.Interval *I, PCode, WheelCode);
        Slope := (Y2 - Y1) / Terr(PCode)^.Interval;
        Intercept := Y2 - Slope *(Terr(PCode)^.Interval *I);
      end;

    A := 1 +sqr (Slope);
    B := 2 *Slope *(Intercept -YSpring) -2 *XSpring;
    C := sqr (XSpring) +sqr (Intercept -YSpring) -sqr (RoadArm);

    Radical := sqr (B) -4 *A *C;

    if (Radical >= 0) then
      begin
        X1 := (-B -sqrt (Radical)) /(2 *A);

        if (I <= 0) then
          begin
            XWheel := X1;
            YWheel := WheelPath (X1, PCode, WheelCode);
            GroundWheelSlope := Slope;
            GroundWheelContact := X1 +WheelRadius *sin (arctan (Slope));
            ExitFlag := true;
          end;

        if ((X1 <= Terr(PCode)^.Interval *I)
          and (X1 >= Terr(PCode)^.Interval *(I-1))) then
          begin

```

```

    XWheel := X1;
    YWheel := WheelPath (X1, PCode, WheelCode);
    GroundWheelSlope := Slope;
    GroundWheelContact := X1 + WheelRadius * sin (arctan (Slope));
    ExitFlag := true;
end;

if (not ExitFlag) then
begin
    X2 := (-B + sqrt (Radical)) / (2 * A);
    if ((X2 <= Terr[PCode].Interval * I)
        and (X2 >= Terr[PCode].Interval * (I-1))) then
        begin
            XWheel := X2;
            YWheel := WheelPath (X1, PCode, WheelCode);
            GroundWheelSlope := Slope;
            GroundWheelContact := X2 + WheelRadius * sin (arctan (Slope));
            ExitFlag := true;
        end;
end;

if (ExitFlag) then
begin
    Inclination := arctan ((YSpring - YWheel) / (XSpring - XWheel));
    if ((Inclination > 0) and (YSpring > YWheel)) then
        Inclination := -pi + Inclination
    else
        if ((Inclination < 0) and (YSpring < YWheel)) then
            Inclination := -pi + Inclination;
        if ((SpringLength + Pitch) < Inclination) then goto 20;
    end;
end;

end;
end;

( If the intersection point was not found,
| set the off-ground flag to zero and calculate wheel
| centre locations based on free spring length/angle.
)
20: Flag := 1;

if (Sign = -1) then
begin
    YWheel := YSpring + RoadArm * sin (-pi);
    XWheel := XSpring + RoadArm * cos (-pi);
    GroundWheelSlope := 0;
    GroundWheelContact := XWheel;
    write'n ('Flag2', XWheel);

```

```

    Flag := 0;
    exit;
end;

writeln ('FLAG1', xwheel);
Ywheel := YSpring + RoadArm * sin (SpringLength + Pitch);
Xwheel := XSpring + RoadArm * cos (SpringLength + Pitch);
exit;
end;

if (TypeSusp = 'helical') then
begin
    ICount := trunc (SpringLength * 2 / Terr[PCode].Interval);

    X := XSpring;

    if (X > 0) then
        IStart := trunc (X / Terr[PCode].Interval) + 1
    else
        IStart := trunc (X / Terr[PCode].Interval) - 1;

    for I:=IStart+ICount downto IStart-ICount do
        begin

            if ((I > 0) and (I < Terr[PCode].Limit)) then
                begin
                    Y1 := Terr[PCode].Y[WheelCode]^I;
                    Y2 := Terr[PCode].Y[WheelCode]^(I+1);
                    Slope := (Y2 - Y1) / Terr[PCode].Interval;
                    Intercept := Y2 - Slope * (Terr[PCode].Interval * I);
                end else
                if (I <= 0) then
                    begin
                        Slope := 0;
                        Intercept := WheelRadius;
                    end else
                    begin
                        Slope := 0;
                        Intercept := Terr[PCode].Y[WheelCode]^(Terr[PCode].Limit);
                    end;

            A := tan (RoadArm + Pitch - 0.5 * pi);
            B := YSpring - A * XSpring;

            X1 := (Intercept - B) / (A - Slope);

            if (I <= 0) then
                begin
                    XWheel := X1;

```

```

    YWheel := WheelPath (X1, PCode, WheelCode);
    GroundWheelSlope := Slope;
    GroundWheelContact := X1 + WheelRadius * sin (arctan (Slope));
    exit;
end;

if ((X1 <= Terr[PCode].Interval * I)
    and (X1 >= Terr[PCode].Interval * (I-1))) then
begin
    XWheel := X1;
    YWheel := WheelPath (X1, PCode, WheelCode);
    GroundWheelSlope := Slope;
    GroundWheelContact := X1 + WheelRadius * sin (arctan (Slope));
    exit;
end;

end;

( If the intersection point was not found,
| set the off-ground flag to zero and calculate wheel
| centre locations based on free spring length/angle.
)

Flag := 1;

YWheel := YSpring + SpringLength * sin (RoadArm + Pitch);
XWheel := XSpring + SpringLength * cos (RoadArm + Pitch);
exit;
end;

end;

```

```

function WheelPath (x : real; PCode, WheelCode : integer) : real;

```

```

( This function returns the wheelpath vertical displacement for a
| given horizontal location along the profile, and for a given
| hubpath index. Linear interpolation is used.
|
| I.....index.
| J.....index.
| Terr.Interval...constant horizontal distance between hubpath data.
| Terr.Limit.....total number of hubpath data.
| Terr.Y.....hubpath height.
| WheelCode.....index of hubpath.
| WheelPath.....hub height at horizontal location.
| X.....horizontal location.
|

```

```

)

var
  I, J : longint;
  Part : real;
  Gimpy : real;
begin
  ( For horizontal locations preceding the start of the terrain profile,
  | set height equal to initial profile height.
  )
  if (x <= 0) then
    WheelPath := Terr[PCode]^Y[WheelCode]^[1]

  ( For horizontal locations following the start of the terrain profile,
  | proceed.
  )
  else begin

  ( Calculate the index of the data point preceding the horizontal
  | location. (This is possible because the terrain data are spaced
  | at equal intervals.
  )
    Part := x / Terr[PCode]^Interval;
    I := trunc (Part) +1;

  ( If the calculated index is greater than the number of terrain data,
  | set the hub height equal to the hub height at the end of the
  | profile.
  )
    if (I >= Terr[PCode]^Limit) then
      WheelPath := Terr[PCode]^Y[WheelCode]^[Terr[PCode]^Limit]
    else

  ( Use linear interpolation to find the hub height.  )

      WheelPath := (Terr[PCode]^Y[WheelCode]^[I+1]
                    -Terr[PCode]^Y[WheelCode]^[I])
                    *(Part -I) +Terr[PCode]^Y[WheelCode]^[I+1];

  end;
end;

function Profile (x : real; PCode : integer) : real;

  ( This function is similar to the WheelPath function, but finds the
  | actual profile height rather than the hubpath height.
  |

```



```

unit TrakProc;

( (c) 1990 Michael Campbell Galway
)

interface

uses Data, Sup_Proc, TerrProc;

procedure PointCircleTangent (Radius, XCentre, XPoint, YCentre,
                             YPoint : real;
                             var Xtan1, Ytan1, Angle : real);
procedure CircleCircleTangent (Radius1, Radius2, XCentre1, XCentre2,
                               YCentre1, YCentre2 : real;
                               var Xtan1, Xtan2, Ytan1, Ytan2,
                               Angle : real);
Procedure TrackSegment (XFront, XRear, RearTangent, FrontTangent,
                        YRearWheel, XRearWheel, YFrontWheel,
                        XFrontWheel, FrontRadius, RearRadius : real;
                        PCode : integer;
                        var ThetaFront, ThetaRear, TrackUsed : real);
procedure Lowertrack (GroundWheelContact, GroundWheelSlope,
                     WheelRadius, XWheel, YWheel : WheelMatrix;
                     Heave, CosRoll, SinRoll, CosPitch, SinPitch,
                     Time, Vel : real;
                     SpridlerRadius, XSpridler, YSpridler,
                     ZSpridler : SpridlerMatrix;
                     NumberOfTracks : integer;
                     NumberOfTrackedWheels : ITrackVector;
                     var LowerTrackLength : TrackArray;
                     var FrontTrackAngle, RearTrackAngle : WheelMatrix;
                     var LowerSpridlerAngle : SpridlerMatrix);
procedure Catenary (X1, Y1, X2, Y2, S, UnitWeight, Elasticity : real;
                   var XC, YC, R, Angle1, Angle2,
                   Tension1, Tension2 : real);
procedure UpperTrack (Heave, CosRoll, SinRoll, CosPitch, SinPitch,
                     Time, Vel : real;
                     NumberOfTracks : integer;
                     LowerTrackLength, TrackLength,
                     UnitWeight, Elasticity : TrackArray;
                     WheelRadius, xWheel, yWheel : WheelMatrix;
                     NumberOfTrackedWheels : ITrackVector;
                     SpridlerRadius, XSpridler, YSpridler,
                     ZSpridler : SpridlerMatrix;
                     var Angle, TotTension : CatMatrix;
                     var Support : IWheelMatrix;
                     var N : ITrackVector);

```

implementation

```

procedure PointCircleTangent (Radius, XCentre, XPoint, YCentre,
                             YPoint : real;
                             var Xtan1, Ytan1, Angle : real);

( This procedure determines the line that is tangent to a circle
| given a point on the line exterior to the circle, and returns the
| point of intersection of the line with the circle, as well as the
| angle of tangency. The formulae were taken from:
|
| McCullough, M.K. and E.J.Haug. "Terra-Dynamics of High
| Mobility Track Vehicles," Technical Report 85-22.
| Centre for Computer Aided Design, University of Iowa.
| December 1985.
|
| A.....slope of line joining circle centre and external point.
| Angle.....angle between horizontal and tangency line.
| B.....ratio of radius to distance from circle centre to point.
| FactorA....convenient grouping of terms.
| FactorB....convenient grouping of terms.
| Radius.....radius of circle.
| XCentre....horizontal location of centre of circle.
| XPoint ...horizontal location of external point.
| Xtan1.....horizontal location of tangency point on circle.
| YCentre....vertical location of centre of circle.
| YPoint.....vertical location of external point.
| Ytan1.....vertical location of tangency point on circle.
|
)

var
  A, B, FactorA, FactorB : real;

begin

( Ensure that XCentre is never coincident with XPoint since this would
| cause A to be infinite (this is anyway physically impossible).
|
)
  if (XCentre = XPoint) then XPoint := XCentre + 0.000001;

( Find the tangency point. )

  A := (YCentre - YPoint) / (XCentre - XPoint);
  B := Radius / (XCentre - XPoint);
  FactorA := (-B + A * sqrt (abs (1 + sqr (A) - sqr (B)))) / (1 + sqr (A));
  FactorB := (-B * A - sqrt (abs (1 + sqr (A) - sqr (B)))) / (1 + sqr (A));
  Xtan1 := XCentre + Radius * FactorA;

```



```

Ytan1 := YCentre +Radius *FactorB;

if (FactorA > 1) then
  Angle := arctan (A)
else
  Angle := Asin (FactorA);
end;

procedure CircleCircleTangent (Radius1, Radius2, XCentre1, XCentru2,
                               YCentre1, YCentre2 : real;
                               var Xtan1, Xtan2, Ytan1, Ytan2,
                               Angle : real);

( This procedure determines the line that is tangent to two circles,
| and returns the points of intersection of the line with the two
| circles, as well as the angle of tangency. The formulae were taken
| from:
|
|   McCullough, M.K. and E.J.Haug. "Terra-Dynamics of High
|   Mobility Track Vehicles," Technical Report 85-22.
|   Centre for Computer Aided Design, University of Iowa.
|   December 1985.
|
|   A.....slope of line joining the centres of two circles.
|   Angle.....angle between horizontal and tangency line.
|   B.....ratio of difference of radii to distance between centres.
|   FactorA....convenient grouping of terms.
|   FactorB....convenient grouping of terms.
|   Radius1....radius of first circle.
|   Radius2....radius of second circle.
|   XCentre1...horizontal location of centre of first circle.
|   XCentre2...horizontal location of centre of second circle.
|   Xtan1.....horizontal location of tangency point on first circle.
|   Xtan2.....horizontal location of tangency point on second circle.
|   YCentre1...vertical location of centre of first circle.
|   YCentre2...vertical location of centre of second circle.
|   Ytan1.....vertical location of tangency point on first circle.
|   Ytan2.....vertical location of tangency point on second circle.
|
| )
var
  A, B, FactorA, FactorB : real;

begin
  ( Ensure that circle centres are never coincident, as this would

```

```

| cause A and B to become infinite (in practice this is
| anyway physically impossible).
)
if (XCentre1 = XCentre2) then XCentre1 := XCentre2 + 0.000001;

( Solve for tangency points. )

A := (YCentre2 - YCentre1) / (XCentre2 - XCentre1);
B := (Radius1 - Radius2) / (XCentre2 - XCentre1);
FactorA := (B + A * sqrt(1 + sqr(A) - sqr(B))) / (1 + sqr(A));
FactorB := (B * A - sqrt(1 + sqr(A) - sqr(B))) / (1 + sqr(A));
Xtan1 := XCentre1 + Radius1 * FactorA;
Xtan2 := XCentre2 + Radius2 * FactorA;
Ytan1 := YCentre1 + Radius1 * FactorB;
Ytan2 := YCentre2 + Radius2 * FactorB;

if (FactorA > 1) then
  Angle := ArcTan(A)
else
  Angle := Asin(FactorA);

( If Ytan1 is greater than the centre of the first circle, then the
| wrong tangent has been determined and the program should halt.
)
if (Ytan1 > YCentre1) then
  begin
    Error(1);
    if (FactorA = 0) then FactorA := FactorA;
  end;
end;

Procedure TrackSegment (XFront, XRear, RearTangent, FrontTangent,
  YRearWheel, XRearWheel, YFrontWheel,
  XFrontWheel, FrontRadius, RearRadius : real;
  PCode : integer;
  var ThetaFront, ThetaRear, TrackUsed : real);

( This procedure calculates the length of track between the wheel-
| ground contact points of two adjacent wheels, taking into account
| terrain topography and wheel geometry.
|
| Front refers to the foremost wheel.
|
| The distance between the ground-wheel contact points (or tangency
| points) of two adjacent is sectioned. The slopes of the terrain
| between these points are found. The slope of the track is determined
| from these slopes subject to the following conditions:

```

```

a. the slope of the track must always decrease once free of the
   wheels; and

b. the slope of the track cannot be greater than the slope of the
   ground at the ground-wheel tangency point at the rearmost
   wheel, and cannot be less than the slope of the ground at the
   ground-wheel tangency point at the frontmost wheel.

A.....a calculated parameter used in finding circle-
   line tangency points.
B.....a calculated parameter used in finding circle-
   line tangency points.
DeltaX.....constant horizontal length of search segments.
I.....index of slope.
J.....array containing index of frontmost section point
   for a given slope.
K.....index of section points.
RearRadius.....radius of rear wheel.
RearTangent.....slope at rear ground-wheel tangency point.
RearX.....track-wheel tangency point for rear wheel.
FrontRadius.....radius of front wheel.
FrontTangent.....slope at front ground-wheel tangency point.
FrontX.....track-wheel tangency point for front wheel.
Section.....number of segments between rear and front ground-
   wheel tangency points.
Slope.. .....slope of line between two section points. Slope
   of track between two section points.
ThetaRear.....angle between track-wheel tangency point and
   ground-wheel tangency point.
ThetaFront.....angle between track-wheel tangency point and
   ground-wheel tangency point.
Theta1.....angle between vertical and track-wheel tangency
   point.
Theta2.....angle between vertical and ground-wheel tangency
   point.
TrackUsed.....length of track between the two ground-wheel
   tangency points.
XRear.....ground-wheel tangency point for rear wheel.
XRearWheel.....rear wheel centre.
XFront.....ground-wheel tangency point for front wheel.
XFrontWheel.....front wheel centre.
Y.....terrain height at a section point.
YRearWheel.....rear wheel centre.
YFrontWheel.....front wheel centre.

```

```

type

```

```

    Matrix = array [0..20] of real;

```

```

IMatrix = array [0..20] of integer;

var
  Slope, Y                               : Matrix;
  J                                       : IMatrix;
  I, K, Section                          : integer;
  DeltaX, Dummy, FrontX, RearX, Theta1, Theta2, XPoint : real;

begin
  ( The horizontal distance between the front and rear wheels' ground
  | tangency points is divided into 15 sections (or any other arbitrary
  | number of sections.
  )

  TrackUsed := 0;
  Section := 15;
  DeltaX := (XFront -XRear) /Section;
  RearX := XRear;
  FrontX := XFront;
  J[0] := 1;
  J[1] := 2;

  ( Find the terrain height at each subdivision.  )

  for I := 0 to Section do
    begin
      Y[I+1] := Profile (XRear +DeltaX *I, PCode);
      if ((XRear +DeltaX *I) < (XRear + RearRadius)) then
        if (sqrt (sqr (XRear +DeltaX *I -XRearWheel)
          +sqr (Y[I+1] -YRearWheel)) <= RearRadius) then
          begin
            Y[I+1] := YRearWheel -sqrt (sqr (RearRadius)
              -sqr (XRear +DeltaX *I -XRearWheel));
            RearX := XRear +DeltaX *I;
            J[0] := I + 1;
            J[1] := I + 2;
          end;
        if ((XRear +DeltaX *I) > (XFront - FrontRadius)) then
          if (sqrt (sqr (XRear +DeltaX *I -XFrontWheel)
            +sqr (Y[I+1] -YFrontWheel)) <= FrontRadius) then
            begin
              Y[I+1] := YFrontWheel -sqrt (sqr (FrontRadius)
                -sqr (XRear +DeltaX *I -XFrontWheel));
              FrontX := XRear +DeltaX *I;
            end;
          end;
    end;

  ( Initialize variables.  )

```

```

if (DeltaX = 0) then Error (2);
Slope[1] := (Y[J[1]] - Y[J[0]]) / DeltaX;
I := 1;

for K := J[1] to Section+1 do
  begin

( Find the slope between adjacent subdivisions. )

    Slope [I] := (Y[K] - Y[J[I-1]]) / ((K - J[I-1]) * DeltaX);

( If the slope between the ground-wheel contact point and the first
| subdivision is greater than the ground slope at the contact point,
| the track would be inside the wheel. This is not physically
| possible. The track must therefore wrap itself around the wheel
| before departing for the first subdivision point. The new point of
| track departure must be found.
)

    if (I = 1) and (Slope[I] > RearTangent) then
      begin

        XPoint := XRear + (J[1] - 1) * DeltaX;
        PointCircleTangent (RearRadius, XRearWheel, XPoint, YRearWheel,
                           Y[J[1]], RearX, Y[1], Dummy);

        end;

( The track must wrap itself around the wheel or it will pass through
| the wheel. Find the new departure point.
)

    if (K = Section+1) and (Slope[I] < FrontTangent) then
      begin

        XPoint := XRear + (J[I-1] - 1) * DeltaX;
        PointCircleTangent (FrontRadius, XFrontWheel, XPoint,
                           YFrontWheel, Y[J[I-1]], FrontX,
                           Y[Section+1], Dummy);

        end;

( If the slope in one subdivision is greater than a preceding
| subdivision, the track is concave. Since the track cannot support
| bending moments, this is not possible. Therefore, the track cannot
| intersect the subdivision point joining the two slopes, and that
| point must be removed from the analysis.
)

    while (I > 1) and (Slope[I] >= Slope[I-1]) do
      begin
        I := I - 1;
        Slope[I] := (Y[K] - Y[J[I-1]]) / ((K - J[I-1]) * DeltaX);

```

```

    if (I = 1) and (Slope[I] > RearTangent) then
        begin
            XPoint := XRear + (J[I] - 1) * DeltaX;
            PointCircleTangent (RearRadius, XRearWheel, XPoint,
                                YRearWheel, Y[J[I]], RearX, Y[I],
                                Dummy);
        end;

    if (K = Section+1) and (Slope[I] < FrontTangent) then
        begin
            XPoint := XRear + (J[I-1] - 1) * DeltaX;
            PointCircleTangent (FrontRadius, XFrontWheel, XPoint,
                                YFrontWheel, Y[J[I-1]], FrontX,
                                Y[Section+1], Dummy);
        end;

    J[I] := K;
end;
J[I] := K;
I := I + 1;
end;

( If I=2, the track forms a straight tangent to the two roadwheels.
| Find the intersection points of the tangent.
)

if I = 2 then
    begin
        CircleCircleTangent (FrontRadius, RearRadius, XFrontWheel,
                                XRearWheel, YFrontWheel, YRearWheel,
                                FrontX, RearX, Y[J[I]], Y[I], Dummy);
    end;

( Find the angle between the ground-wheel contact point and the track-
| wheel contact point. Find the track arc subtended by this angle.
)

Theta1 := Asin ((FrontX - XFrontWheel) / FrontRadius);
Theta2 := Asin ((XFront - XFrontWheel) / FrontRadius);
ThetaFront := Theta2 - Theta1;
TrackUsed := FrontRadius * ThetaFront;

( Repeat the above process for the other wheel. Add the track arc to
| the summation of track used.
)

Theta1 := Asin ((XRearWheel - RearX) / RearRadius);
Theta2 := Asin ((XRearWheel - XRear) / RearRadius);
ThetaRear := Theta2 - Theta1;
TrackUsed := TrackUsed + RearRadius * ThetaRear;

( Sum the track between adjacent subdivisions. )

```

```

if I = 2 then
  begin
    TrackUsed := TrackUsed +sqrt (sqr (FrontX -RearX) +sqr (Y[J[1]]
      -Y[1]));
  end;

if I > 2 then
  begin
    TrackUsed := TrackUsed +sqrt (sqr (FrontX -(XRear +DeltaX *(J[I-2]
      -1))) +sqr (Y[Section+1] -Y[J[I-2]]));
    TrackUsed := TrackUsed +sqrt (sqr (XRear +DeltaX *(J[I] -1)
      -RearX) +sqr (Y[J[I]] -Y[1]));
  end;

for K := 2 to I-2 do
  begin
    TrackUsed := TrackUsed +sqrt (sqr (DeltaX *(J[K] -J[K-1]))
      +sqr (Y[J[K]] -Y[J[K-1]]));
  end;
end;

procedure Lowertrack (GroundWheelContact, GroundWheelSlope,
  WheelRadius, XWheel, YWheel : WheelMatrix;
  Heave, CosRoll, SinRoll, CosPitch, SinPitch,
  Time, Vel : real;
  SpridlerRadius, XSpridler, YSpridler,
  ZSpridler : SpridlerMatrix;
  NumberOfTracks : integer;
  NumberOfTrackedWheels : ITrackVector;
  var LowerTrackLength : TrackArray;
  var FrontTrackAngle, RearTrackAngle : WheelMatrix;
  var LowerSpridlerAngle : SpridlerMatrix);

( This procedure calculates the length of track on the ground between
| the top of the front spridler and the top of the rear spridler.
|
| Angle.....angle of track between spridler and wheel.
| FrontTrackAngle.....the angle between the track and the
|                      horizontal at the front of the wheel.
| GroundWheelContact.....the horizontal coordinate of the ground-
|                      wheel contact point.
| GroundWheelSlope.....the slope at the ground-wheel contact point.
| I.....an index.
| J.....an index.

```

```

| LowerSpridlerAngle.....angle between the track and the horizontal
|                          below the spridler.
| LowerTrackLength.....tracklength on the ground between spridlers.
| NumberOfTrackedWheels...number of roadwheels in a given track.
| NumberOfTracks.....number of tracks on the vehicle.
| RearTrackAngle.....angle between the track and the horizontal
|                          at the rear of the wheel.
| SegmentLength.....length of track segment between ground-wheel
|                          contact points of adjacent wheels.
| SpridlerMatrix.....variable type.
| SpridlerRadius.....radius of a given spridler.
| WheelMatrix.....variable type.
| TrackVector.....variable type.
| WheelRadius.....radius of given roadwheel.
| XSpridler.....horizontal coordinate of given spridler.
| XTan1.....horizontal coordinate of track-spridler
|              tangency point.
| XTan2.....horizontal coordinate of track-wheel
|              tangency point.
| Xwheel.....horizontal coordinate of given roadwheel.
| YSpridler.....vertical coordinate of given spridler.
| YTan1.....vertical coordinate of track-spridler
|              tangency point.
| YTan2.....vertical coordinate of track-wheel
|              tangency point.
| Ywheel.....vertical coordinate of given roadwheel.
|
)

```

```

var
  I, J                      : integer;
  Angle, SegmentLength, Xa, XTan1, XTan2 : real;
  Ya, YTan1, YTan2, Za      : real;

begin
  ( Repeat for each track on the vehicle. )

  for I := 1 to NumberOfTracks do
    begin
      LowerTrackLength[I] := 0;

  ( Find length of track between each pair of adjacent wheels, and sum.
  )
      for J := 1 to round (NumberOfTrackedWheels[I]) - 1 do
        begin
          TrackSegment (GroundWheelContact[I,J],
                        GroundWheelContact[I,(J+1)],
                        GroundWheelSlope[I,(J+1)],
                        GroundWheelSlope[I,J], Ywheel[I,J+1],

```



```

        XWheel[I,J+1], YWheel[I,J], XWheel[I,J],
        WheelRadius[I,J], WheelRadius[I,J+1],
        ProfileCode[I,J],
        RearTrackAngle[I,J], FrontTrackAngle[I,J+1],
        SegmentLength);
    LowerTrackLength[I] := LowerTrackLength[I] + SegmentLength;
end;

( Find track length from front spridler to front roadwheel. )

Coord ('R->A', Heave, CosRoll, SinRoll, CosPitch, SinPitch,
      Time, Vel, Xa, Ya, Za,
      XSpridler[I,1], YSpridler[I,1], ZSpridler[I,1]);

CircleCircleTangent (SpridlerRadius[I,1], WheelRadius[I,1],
                    Xa, XWheel[I,1], Ya,
                    YWheel[I,1], XTan1, XTan2, YTan1, YTan2,
                    Angle);
FrontTrackAngle[I,1] := Angle;
LowerSpridlerAngle[I,1] := Angle;

( Add straight length of track between spridler and wheel. )

LowerTrackLength[I] := LowerTrackLength[I] + sqrt (sqr (XTan1
    -XTan2) +sqr (YTan1 -YTan2));
( Add track arc between top of spridler and track-spridler contact
  | point.
  )
LowerTrackLength[I] := LowerTrackLength[I] +(pi -Angle)
    *SpridlerRadius[I,1];

( Add track arc between ground-wheel contact point and track-wheel
  | contact point.
  )
LowerTrackLength[I] := LowerTrackLength[I]
    +(-GroundWheelSlope[I,1] +Angle)
    *WheelRadius[I,1];

( Find track length from rear spridler to rear roadwheel. )

Coord ('R->A', Heave, CosRoll, SinRoll, CosPitch, SinPitch,
      Time, Vel, Xa, Ya, Za,
      XSpridler[I,2], YSpridler[I,2], ZSpridler[I,2]);

CircleCircleTangent (SpridlerRadius[I,2], WheelRadius[I,J+1],
                    Xa, XWheel[I,J+1],
                    Ya, YWheel[I,J+1], XTan1, XTan2,
                    YTan1, YTan2, Angle);
RearTrackAngle[I,J+1] := Angle;

```

```

    LowerSpridlerAngle[I,2] := Angle;

( Add straight track segment. )

    LowerTrackLength[I] := LowerTrackLength[I] +sqrt (sqr (XTan1
        -XTan2) +sqr (YTan1 -YTan2));

( Add track arc for spridler. )

    LowerTrackLength[I] := LowerTrackLength[I] +(pi +Angle)
        *SpridlerRadius[I,2];

( Add track arc for roadwheel. )

    LowerTrackLength[I] := LowerTrackLength[I]
        +(-GroundWheelSlope[I,J+1] +Angle)
        *WheelRadius[I,J+1];

end;
end;

procedure Catenary (X1, Y1, X2, Y2, S, UnitWeight, Elasticity : real;
    var XC, YC, R, Angle1, Angle2,
        Tension1, Tension2 : real);

( This procedure calculates the total tension present at the
| suspension points of a pseudo-catenary, as well as the tangents.
| The pseudo-catenary is defined as being the circular arc subtended
| by the suspension points with arc length equal to the length of
| suspended track. The tensions are determined by a force balance.
| The circular approximation of the catenary is valid for small
| catenary sags, and small inclinations of the suspension points.
| Huyghen's approximation formula is used to relate known parameters
| to the unknowns explicitly.
|
| Alpha.....angle with respect to horizontal of line connecting
| pseudo-catenary centre with front suspension point.
| Angle1.....tangent to pseudo-catenary at front suspension point.
| Angle2.....tangent to pseudo-catenary at rear suspension point.
| Beta.....arc angle subtended by suspension points.
| D.....straightline distance between suspension points.
| R.....radius of circular pseudo-catenary.
| S.....length of suspended track.
| Tension1.....total tension at front suspension point.
| Tension2.....total tension at rear suspension point.
| Theta.....angle between line intersecting suspension points
| and the horizontal.

```

```

|   UnitWeight....unit weight of track (kg/m).
|   Weight.....total weight of suspended track.
|   XC.....horizontal coordinate of centre of circular
|           pseudo-catenary.
|   X1.....horizontal coordinate of front suspension point.
|   X2.....horizontal coordinate of rear suspension point.
|   YC.....vertical coordinate of centre of circular
|           pseudo-catenary.
|   Y1.....vertical coordinate of front suspension point.
|   Y2.....vertical coordinate of rear suspension point.
|
|
)

var
  Alpha, Beta, D, Theta, Weight : real;

begin
  ( Evaluate known inclination and distance between suspension points. )

  D := sqrt (sqr (X1 -X2) + sqr (Y1 -Y2));
  Theta := arctan ((Y1 -Y2) /(X1 -X2));
  Weight := UnitWeight * S;

  if (D >= 0.99 *S) then
    begin
      Beta := 2 *pi -4 *Asin (0.99);

      if (Beta < 0.3011364) then Beta := 0.3011364;

    ( Solve for radius and centre of arc. )

      R := 1.010101 *D /Beta;
      Angle1 := Theta + Beta /2;
      Angle2 := Theta - Beta /2;
      Alpha := pi /2 + Angle1;
      XC := X1 +R *cos (Alpha);
      YC := Y1 +R *sin (Alpha);

    ( Use force balance to evaluate tensions. )

      Tension1 := Weight /(sin (Angle1) -cos (Angle1) *tan (Angle2));
      Tension2 := Weight /(cos (Angle2) *tan (Angle1) -sin (Angle2));
      Tension1 := Elasticity *(1.010101 *D -S) +Tension1;
      Tension2 := Elasticity *(1.010101 *D -S) +Tension2;
      exit;
    end;

  ( Use Huyghen's approximation formula to solve for subtended arc

```

```

| angle.
>
  Beta := 2 *pi -4 *Asin (4 *D /(3 *S +D));

  if (Beta < 0.3011364) then Beta := 0.3011364;

( Solve for radius and centre of arc.  )

  R := S / Beta;
  Angle1 := Theta + Beta /2;
  Angle2 := Theta - Beta /2;
  Alpha := pi /2 + Angle1;
  XC := X1 +R *cos (Alpha);
  YC := Y1 +R *sin (Alpha);

( Use force balance to evaluate tensions.  )

  Tension1 := Weight /(sin (Angle1) -cos (Angle1) *tan (Angle2));
  Tension2 := Weight /(cos (Angle2) *tan (Angle1) -sin (Angle2));
end;

procedure UpperTrack (Heave, CosRoll, SinRoll, CosPitch, SinPitch,
  Time, Vel : real;
  NumberOfTracks : integer;
  LowerTrackLength, TrackLength,
  UnitWeight, Elasticity : TrackArray;
  WheelRadius, xWheel, yWheel : WheelMatrix;
  NumberOfTrackedWheels : ITrackVector;
  SpridlerRadius, XSpridler, YSpridler,
  ZSpridler : SpridlerMatrix;
  var Angle, TotTension : CatMatrix;
  var Support : IWheelMatrix;
  var N : ITrackVector);

( This procedure determines track tensions in the upper track at
| suspension points.
|
| Angle.....angles made by catenary with supports.
| Distance.....distance between pseudo-catenary centre and
|               possible interference point.
| I.....a counter.
| II.....a counter.
| Interference.....flag to indicate assumed catenary sits below
|                  a roadwheel.
| IWheelMatrix.....variable type.
| ITrackVector.....variable type.
| J.....a counter.

```

```

| K.....a counter.
| LowerTrackLength.....length of track on the ground between TDCs
|                       of front and rear spridlers.
| N.....number of catenaries in upper loop.
| NumberOfTrackedWheels...number of roadwheels in a given track.
| NumberOfTracks.....number of tracks on a vehicle.
| R.....radius of circular pseudo-catenary.
| S.....length of track in each catenary.
| Secant.....straightline distance between catenary
|             supports; used in apportioning track length.
| SecantTotal.....sum of straightline distances between
|             catenary supports.
| Support.....array of indices of roadwheels supporting
|             catenary.
| TotTension.....total tension at catenary supports.
| TrackLength.....total length of track.
| TrackMatrix.....variable type.
| TrackVector.....variable type.
| UnitWeight.....weight of track per unit length.
| WheelRadius.....radius of roadwheel.
| WheelSupport.....array of flags indicating roadwheels
|             supporting catenary.
| TrackVector.....variable type.
| XC.....centre of circular pseudo-catenary.
| xSpridler.....horizontal position of spridler.
| xWheel.....horizontal position of centre of roadwheel.
| X1.....horizontal location, front catenary support.
| X2.....horizontal location, rear catenary support.
| YC.....centre of circular pseudo-catenary.
| ySpridler.....vertical position of spridler.
| yWheel.....vertical position of centre of roadwheel.
| Y1.....vertical location, front catenary support.
| Y2.....vertical location, rear catenary support.
)

```

```

var
  I, II, Interference, J, K                : integer;
  WheelSupport                             : IWheelVector;
  S, Secant                                : WheelVector;
  Distance, R, SecantTotal, Xa, XC, X1, X2, Ya, YC : real;
  Y1, Y2, Za                               : real;

```

```
begin
```

```
  for I:=1 to NumberofTracks do
```

```
(   Repeat for each track.   )
```

```

begin

( Initialize. The first support is 0, the front spridler. The
| second support is the number of tracked wheels in the track plus
| one (the rear spridler). Set the support flags for all road-
| wheels to zero since they aren't supporting the catenary. Set
| the length of track in the catenary equal to the difference of the
| total track length and the length of track on the ground. Set the
| interference flag to one to ensure at least one check of roadwheel
| interference.
)

Support[I,1] := 0;
Support[I,2] := NumberOfTrackedWheels[I] + 1;
for II:=1 to NumberOfTrackedWheels[I] do
    WheelSupport[II] := 0;
S[I] := TrackLength[I] - LowerTrackLength[I];
K := 2;
Interference := 1;

while Interference = 1 do

( Begin roadwheel interference check. )

begin
    Interference := 0;

    for II:=1 to K-1 do

( Find the pseudo-catenary (subtended arc) parameters for all K-1
| catenaries. For the first catenary, the first support is the
| front spridler. For the last catenary, the rear spridler is the
| second support. All intermediate catenaries have roadwheels as
| supports. Procedure Catenary returns pseudo-catenary parameters.
)

begin
    if Support[I,II] = 0 then
        begin
            Coord ('R->A', Heave, CosRoll, SinRoll, CosPitch, SinPitch,
                Time, Vel, Xa, Ya,
                Za, xSpridler[I,1], ySpridler[I,1],
                zSpridler[I,1]);

            X1 := Xa;
            Y1 := Ya +SpridlerRadius[I,1];
        end else
            begin
                X1 := xWheel[I,Support[I,II]];
                Y1 := yWheel[I,Support[I,II]]
                    +WheelRadius[I,Support[I,II]];
            end;
end;

```

```

if Support[I,I+1] = NumberOfTrackedWheels[I]+1 then
  begin
    Coord ('R->A', Heave, CosRoll, SinRoll, CosPitch, SinPitch,
          Time, Vel, Xa, Ya,
          Za, xSpridler[I,2], ySpridler[I,2],
          zSpridler[I,2]);
    X2 := Xa;
    Y2 := Ya +SpridlerRadius[I,2];
  end else
    begin
      X2 := xWheel[I,Support[I,I+1]];
      Y2 := yWheel[I,Support[I,I+1]]
          +WheelRadius[I,Support[I,I+1]];
    end;

  Catenary (X1, Y1, X2, Y2, S[I], UnitWeight[I],
            Elasticity[I], XC, YC, R, Angle[I,I,1],
            Angle[I,I,2], TotTension[I,I,1],
            TotTension[I,I,2]);

  for J:=Support[I,I]+1 to Support[I,I+1]-1 do
    begin
      ( Check all the roadwheels between the two supports to ensure they
        | do not interfere with the pseudo-catenary (ie check that the
        | distance between each roadwheel top and the pseudo-catenary (arc)
        | centre is greater than the arc radius). If they do, flag them, and
        | set the interference flag to 1.
        )

      Distance := sqrt (sqr (xWheel[I,J] -XC)
                       +sqr (yWheel[I,J] +WheelRadius[I,J] -YC));

    end;
  end;

  if Interference = 1 then

    ( If the interference flag was set to 1, restart with the two known
      | catenary supports (the spridlers), and add more supports by checking
      | the support flags in WheelSupport.
      )

    begin
      K := 2;
      for II:=1 * NumberOfTrackedWheels[I] do
        begin
          if WheelSupport[II] = 1 then
            begin
              Support[I,K] := II;
              K := K + 1;
            end;
          end;
        end;
      end;
    end;
  end;

```

```

    end;
end;

( The last support is the second spridler. )

Support[I,K] := NumberOfTrackedWheels[I] + 1;

( Distribute the track to each of the new K-1 catenaries on the basis
| of the catenary secants joining each pair of supports.
)

Coord ('R->A', Heave, CosRoll, SinRoll, CosPitch, SinPitch,
      Time, Vel, Xa, Ya,
      Za, xSpridler[I,1], ySpridler[I,1],
      zSpridler[I,1]);

Secant[I] := sqrt (sqr (xSpridler[I,1]
      -xWheel[I,Support[I,2]])
      +sqr (Ya +SpridlerRadius[I,1]
      -yWheel[I,Support[I,2]]
      -WheelRadius[I,Support[I,2]]));

for I1:=2 to K-2 do
    Secant[I1] := sqrt (sqr (xWheel[I,Support[I,I1]]
      -xWheel[I,Support[I,I1+1]])
      +sqr (yWheel[I,Support[I,I1]]
      +WheelRadius[I,Support[I,I1]]
      -yWheel[I,Support[I,I1+1]]
      -WheelRadius[I,Support[I,I1+1]]));

Coord ('R->A', Heave, CosRoll, SinRoll, CosPitch, SinPitch,
      Time, Vel, Xa, Ya,
      Za, xSpridler[I,2], ySpridler[I,2],
      zSpridler[I,2]);

Secant[K-1] := sqrt (sqr (Xa
      -xWheel[I,Support[I,K-1]])
      +sqr (Ya +SpridlerRadius[I,2]
      -yWheel[I,Support[I,K-1]]
      -WheelRadius[I,Support[I,K-1]]));

SecantTotal := 0;
for I1:=1 to K-1 do SecantTotal := SecantTotal + Secant[I1];

for I1:=1 to K-1 do
    begin
        S[I1]:=Secant[I1] /SecantTotal *(TrackLength[I]
            -LowerTrackLength[I1]);
    end;
end;

```



```
        end;  
    end;  
  
    ( End of calculations for track. Calculate number of catenaries).  
    )  
    N(1) := K - 1;  
    end;  
end;  
  
begin  
end.
```

APPENDIX 7

SOURCE CODE FOR RIDE-Q.EXE

```

unit Common;

( (c) 1990 Michael Campbell Galway
)

interface

type
  Orient = (FRONTWARD, SIDEWAYS);
  Axis   = (SEAT_VERTICAL,
            SEAT_LATERAL,
            SEAT_FORE_AFT,
            FEET_VERTICAL,
            FEET_LATERAL,
            FEET_FORE_AFT,
            BACK_VERTICAL,
            BACK_LATERAL,
            BACK_FORE_AFT);

  BigArray  = array [0..10900] of real;
  BigPointer = ^BigArray;
  OutArray  = array [1..3,1..3] of real;
  Spectrum  = array [0..19] of real;
  Code      = (B, C, D);

implementation

begin
end.

```

```
unit FastFT;
```

```
( (c) 1990 Michael Campbell Galwey
)
```

```
interface
```

```
uses Common;
```

```
procedure FFT (FFTOut : string;
               DeltaInterval, Interval1 : real;
               J : integer;
               Direction : string;
               var xReal, xImag : BigPointer;
               var N : integer);
```

```
implementation
```

```
procedure FFT (FFTOut : string;
               DeltaInterval, Interval1 : real;
               J : integer;
               Direction : string;
               var xReal, xImag : BigPointer;
               var N : integer);
```

```
( This subroutine computes the discrete Fourier transform, or the
  inverse discrete Fourier transform of a function using fast Fourier
  transform (FFT) technique.
```

VARIABLES:

Arg.....FFT variable.

C.....FFT variable.

Convert.....factor to convert from the discrete FT to the
 approximate continuous FT. This is equal to the
 constant interval between input data.

DeltaInterval....Sample spacing.

DeltaTransform...Sample spacing in transform domain.

Direction.....indicates whether transformation is from the time
 domain to the frequency domain or vice versa.

Exponent.....the number of samples must be equal to a whole
 power of '2'. This is the power of '2' equal to
 the number of samples. If 'Exponent' is not a
 whole number, zeros must be added to the
 sample set.

f1.....file with input data.

f2.....file with transformed output data.

I.....counter.

Interval1.....Lowest time/frequency for input data.
 Interval2.....Second lowest time/frequency for input data.
 J.....counter.
 K.....counter.
 L.....counter.
 N.....number of samples taken of the continuous
 function.
 Nu.....the number of samples must be equal to a whole
 power of '2'. 'Nu' is this power.
 Nu1.....Nu - 1.
 N2.....half the number of samples.
 P.....FFT variable.
 S.....FFT variable.
 Shift.....True if data is time or frequency shifted
 (ie data begins at time or frequency less than
 zero; this is always true for frequency data);
 otherwise false.
 ShiftArg.....Common term of exponential involved in shift
 correction.
 Temp.....temporary holder.
 Timg.....FFT variable.
 TReal.....FFT variable.
 X.....full argument for shift correcting exponential.
 ximg.....imaginary data.
 xReal.....real data.

The FFT algorithm is based upon an ALGOL algorithm given in:

Brigham, E.O. The Fast Fourier Transform. Prentice-Hall, Inc.,
 Englewood Cliffs, New Jersey. 1974.

)

```

function BitRev (J, Nu : integer) : integer;
var
  I, J1, J2, K : integer;
begin
  J1 := J;
  K := 0;
  for I:=1 to Nu do
    begin
      J2 := J1 div 2;
      K := K * 2 + (J1 - 2 * J2);
      J1 := J2;
    end;
  BitRev := K;
end;

```

```

var
  I, K, L, Nu, Nu1, N2 : integer;
  Arg, C, Convert, DeltaTransform, Exponent,
    Interval2, P, S, ShiftArg, Temp, Timg, TReal, X : extended;
  f1, f2 : text;
  Shift : boolean;
  label 100;

begin
  Shift := false;

  if (FFTOut <> 'DEFAULT') then
    begin:
      assign (F2, FFTOut);
      rewrite (F2);
    end;

  ( If going from time to frequency domain, assume time data is real.  If
    going from frequency to time domain, assume frequency data is complex.
    Frequency data must be conjugated to use the forward FFT algorithm.
  )

  for I:=0 to J do
    begin
      if (Direction = 'T->F') then
        ximag^[I] := 0
      else
        ximag^[I] := -ximag^[I];
    end;

  ( Check whether the number of data is an integer power of two.  If not,
    find the next largest power of two.  )

  Exponent := ln (J) /ln (2.0);
  if (frac (Exponent) < 0.5) and (frac (Exponent) > 0.0) then
    Exponent := Exponent + 1;
  Nu := round (Exponent);
  N := round (exp (Nu *ln (2.0)));

  DeltaTransform := 1 /DeltaInterval /N;
  Convert := DeltaInterval;

  ( Check for shifting along the time or frequency axis (the FFT assumes

```

```

the signal starts at the origin).  )

if (Interval1 < 0) then
  begin
    Shift := true;
    ShiftArg := 2 *pi *abs (Interval1) /DeltaInterval /N;
  end;

( If the number of input data is not a power of two, pad the data out to
the next largest power of two with zeros (these will only increase the
resolution of the transform.  )

for I:=J to N-1 do
  begin
    xReal^[I] := 0.0;
    xImag^[I] := 0.0;
  end;

( The forward FFT algorithm follows:  )

N2 := N div 2;
Nu1 := Nu -1;
K := 0;
for L:=1 to Nu do
  begin
100:   for I:=1 to N2 do
      begin
        P := BitRev (K div round (exp (Nu1 *ln (2.0))), Nu);
        Arg := 2 *pi *P / int (N);
        C := cos (Arg);
        S := sin (Arg);
        TReal := xReal^[K+N2] *C +xImag^[K+N2] *S;
        TImag := xImag^[K+N2] *C -xReal^[K+N2] *S;
        xReal^[K+N2] := xReal^[K] -TReal;
        xImag^[K+N2] := xImag^[K] -TImag;
        xReal^[K] := xReal^[K] +TReal;
        xImag^[K] := xImag^[K] +TImag;
        K := K + 1;
      end;
    K := K + N2;
    if K < N then goto 100;
    K := 0;
    Nu1 := Nu1 - 1;
    N2 := N2 div 2;
  end;
for K:=0 to N-1 do
  begin

```

```

I := BitRev (K, Nu);
if I > K then
  begin
    TReal := xReal^K;
    TImag := xImag^K;
    xReal^K := xReal^I;
    xImag^K := xImag^I;
    xReal^I := TReal;
    xImag^I := TImag;
  end;
end;

if (Direction = 'T->F') then
  begin

( Correct frequency domain data for time shifting. )

I := 1;

for J:=0 to N-1 do
  if (J > round (N / 2) - 1) then I := -1;
  K := J + I * round (N / 2);
  if (Shift = true) then
    begin
      X := ShiftArg * K;
      if (abs (cos (X)) <= 0.000001) then
        begin
          Temp := xImag^K;
          xImag^K := xReal^K / sin (X);
          xReal^K := -Temp / sin (X);
        end else
          if (abs (sin (X)) <= 0.000001) then
            begin
              xImag^K := xImag^K / cos (X);
              xReal^K := xReal^K / cos (X);
            end else
              begin
                xImag^K := xReal^K * sin (X) + xImag^K * cos (X);
                xReal^K := xReal^K / cos (X) - xImag^K
                  * sin (X) / cos (X);
              end;
        end;
end;

for K:=round (N/2) to (N-1) do
  begin
    xReal^K := xReal^K * Convert;
    xImag^K := xImag^K * Convert;
    if (FFTOut <> 'DEFAULT') then

```



```

        writeln (f2, DeltaTransform *(K-N), ' ', xReal^[K],
            ' ', xImag^[K]);
    TReal := XReal^[K];
    TImag := XImag^[K];
    XReal^[K] := XReal^[K-round(N/2)];
    XReal^[K-round(N/2)] := TReal;
    XImag^[K] := XImag^[K-round(N/2)];
    XImag^[K-round(N/2)] := TImag;
end;

for K:=round (N/2) to (N-1) do
    begin
        XReal^[K] := XReal^[K] * Convert;
        XImag^[K] := XImag^[K] * Convert;
        if (FFTOut <> 'DEFAULT') then
            writeln (f2, DeltaTransform *(K-N/2) , ' ', xReal^[K],
                ' ', xImag^[K]);
        end;
    end else
        begin
            for K:=0 to (N - 1) do
                begin
                    xImag^[K] := -xImag^[K];
                end;
            end;
        end;
    end;

( Correct time domain data for frequency shifting. )

    if (Shift = true) then
        begin
            X := ShiftArg * K;
            if (abs (cos (X)) <= 0.000001) then
                begin
                    Temp := xReal^[K];
                    xReal^[K] := xImag^[K] /sin (X);
                    xImag^[K] := -Temp /sin (X);
                end else
                    if (abs (sin (X)) <= 0.000001) then
                        begin
                            xImag^[K] := xImag^[K] /cos (X);
                            xReal^[K] := xReal^[K] /cos (X);
                        end else
                            begin
                                xReal^[K] := xReal^[K] *cos (X) +xImag^[K] *sin (X);
                                xImag^[K] := xImag^[K] /cos (X) -xReal^[K]
                                    *sin (X) /cos (X);
                            end;
                    end;
                end;
            end;
        end;
    end;
end;

```

```

XReal^[K] := XReal^[K] * Convert;
XImag^[K] := XImag^[K] * Convert;

if (FFTOut <> 'DEFAULT') then
  writeln (f2, DeltaTransform *K, ' ', xReal^[K],
          ' ', xImag^[K]);
end;
end;

if (FFTOut <> 'DEFAULT') then close (F2);
end;

begin
end.

```

```

unit PostProc;

( (c) 1990 Michael Campbell Galway
)

interface

uses Common, FastFT;

function Even (X : integer) : boolean;

procedure MoveAxes (Position : Orient;
                    Xr, Yr, Zr : real;
                    Heave, Pitch, Roll, VHeave, VPitch, VRoll,
                    AHeave, APitch, ARoll : real;
                    var Ax, Ay, Az : real);

procedure Power (Motion : Axis;
                DeltaTime, TimeStop, TimeStart : real;
                NumFFT : integer;
                FName : string;
                Hr, Hi : BigPointer;
                var AbPower : real);

procedure Griffin (Motion : Axis;
                  DeltaTime, TimeStop, TimeStart : real;
                  NumFFT, NumTime : integer;
                  FName : string;
                  RMag, IMag : BigPointer;
                  var Crest, ERMS, VDV, T15 : real;
                  var Comfort : integer);

procedure ISO2631 (Motion : Axis;
                  DeltaTime, TimeStop, TimeStart : real;
                  NumFFT, NumTime : integer;
                  FName : string;
                  RMag, IMag : BigPointer;
                  var Crest, RMSWeighted, TExplim, TDecPro,
                  TRedCom : real;
                  var RMS : Spectrum);

implementation

function Even (X : integer) : boolean;

( This function determines whether an integer is even (TRUE)
| or odd (FALSE). Allowance has been made for loss of precision
| in real division.
)

```

```

begin
  if (abs (frac (X / 2)) < 0.25) then
    Even := true
  else
    Even := false;
end;

```

```

procedure MoveAxes (Position : Orient;
                    Xr, Yr, Zr : real;
                    Heave, Pitch, Roll, VHeave, VPitch, VRoll,
                    AHeave, APitch, ARoll : real;
                    var Ax, Ay, Az : real);

```

```

( This procedure calculates vertical, lateral, and fore-aft
| accelerations for a point on the vehicle with axes fixed to
| the vehicle. This means that even though a driver or passenger
| be pitched or rolled with the vehicle, the accelerations acting
| along his caudocephalid, anteroposterior, and lateral axes may
| be determined. Since ride quality criteria are defined in terms
| of acceleration inputs along these axes, the accelerations must
| be given with respect to these axes, not the globally fixed
| axes outside the vehicle. The user must indicate whether the
| driver or passenger is facing ahead (usually) or to either side
| (as section members in a personnel carrier might).
|
| AHeave.....acceleration along global heave axis.
| APitch.....acceleration around global pitch axis.
| ARoll.....acceleration around global roll axis.
| ATemp.....temporary variable.
| Ax.....acceleration along lateral or anteroposterior axis.
| Ay.....acceleration along caudocephalid axis.
| Az.....acceleration along lateral or anteroposterior axis.
| Heave.....global heave displacement.
| Pitch.....global pitch displacement.
| Position...driver/passenger orientation.
| Roll.....global roll displacement.
| VHeave.....global heave velocity.
| VPitch.....global pitch velocity.
| VRoll.....global roll velocity.
| Xr.....longitudinal point of acceleration input.
| Ya.....absolute vertical point of acceleration input.
| Yr.....vertical point of acceleration input.
| Zr.....lateral point of acceleration input.
|
)

```

```

var
  Ya, ATemp : real;

begin
  ( Find vertical difference in CG and point of interest in global
  | coordinate system.
  )
  Ya := Yr *cos (Roll) -Zr *sin (Roll);

  ( Calculate accelerations at point in all three axes, making sure
  | to include centripetal effects.
  )
  Ax := AHeave *sin (Pitch) -sqr (VPitch) *Xr -APitch *Ya;
  Ay := (AHeave *cos (Pitch) -sqr (VPitch) *Ya +APitch *Xr)
        *cos (Roll) -sqr (VRoll) *Yr +ARoll *Zr;
  Az := (AHeave *cos (Pitch) -sqr (VPitch) *Ya +APitch *Xr)
        *sin (Roll) -sqr (VRoll) *Zr -ARoll *Yr;

  ( If the driver/passenger is sitting sideways, interchange
  | lateral and fore-aft accelerations.
  )
  if (Position = SIDEWAYS) then
    begin
      ATemp := Ax;
      Ax := Az;
      Az := ATemp;
    end;
  end;

  procedure Power (Motion : Axis;
    DeltaTime, TimeStop, TimeStart : real;
    NumFFT : integer;
    FName : string;
    Hr, Hi : BigPointer;
    var AbPower : real);

  ( This procedure calculates the average absorbed power due to
  | acceleration in a certain axis, given the FFT of the acceleration-
  | time history.
  |
  | AbPower.....average absorbed power in an axis.
  | DeltaTime....sampling rate in time.
  | FName.....file name for absorbed PSD output.

```

```

| F1.....absorbed power weighting parameter.
| F2.....absorbed power weighting parameter.
| F3.....absorbed power weighting parameter.
| F4.....absorbed power weighting parameter.
| Hertz.....frequency.
| Hr.....real part of FFT.
| Hi.....imaginary part of FFT.
| KP.....absorbed power weighting parameter.
| K2.....total weighting at frequency.
| K0.....absorbed power weighting parameter.
| K1.....absorbed power weighting parameter.
| L.....counter.
| Magnitude....absorbed PSD.
| Motion.....axis of acceleration.
| NumFFT.....number of points in FFT.
| Omega.....radian frequency.
| Omega2.....radian frequency squared.
| Omega4.....radian frequency to the fourth power.
| Omega6.....radian frequency to the sixth power.
| Ten.....natural log of ten, used to compute powers of ten.
| TimeStart....time at start of run.
| TimeStop.....time at end of run.
| Weight.....weighting for Simpson integration.
|
|
)

```

```

var
  F1, F2, F3, F4, Hertz, KP, KTotal, K2, K0, K1      : real;
  Magnitude, Omega, Omega2, Omega4, Omega6, Ten, Weight : real;
  L                                                    : integer;
  FN1                                                  : text;

```

```
begin
```

```
( Initialize parameters. )
```

```

  Ten := ln(10.0);
  K1 := 1.356;
  AbPower := 0;

```

```
( Open file to which to write absorbed PSD if requested by user. )
```

```

  if (FName <> 'DEFAULT') then
  begin
    assign (FN1, FName);
    rewrite (FN1);
  end;

```

```
( Weight each component of the FFT according to frequency. )
```

```

for L:=0 to (NumFFT-1) do
  begin
    Hertz := (L - NumFFT / 2) / DeltaTime / NumFFT;

    ( Zero frequency will cause a domain error. )

    if (Hertz = 0) then Hertz := 0.0000001;

    Omega := 2 * Pi * abs (Hertz);
    Omega2 := exp (2 * ln (Omega));
    Omega4 := exp (4 * ln (Omega));
    Omega6 := exp (6 * ln (Omega));

    ( Filter acceleration-time history at 30 Hz (low-pass). This is a
    | standard procedure reflecting the rapid fall-off in human
    | response to vibration above 30 Hz.
    )

    if (abs (Hertz) > 30) then
      begin
        Hr^[L] := 0;
        Hi^[L] := 0;
      end;

    ( Calculate frequency weighting for vertical motion. )

    if (Motion = SEAT_VERTICAL) then
      begin
        F1 := -0.10245296 * exp ( -9 * Ten) * Omega6
              +0.17583343 * exp ( -5 * Ten) * Omega4
              -0.44600722 * exp ( -2 * Ten) * Omega2 +1;
        F2 := 0.12881887 * exp ( -7 * Ten) * Omega4
              -0.93394367 * exp ( -4 * Ten) * Omega2 +0.10543059;
        F3 := -0.45416156 * exp ( -9 * Ten) * Omega6
              +0.37667129 * exp ( -5 * Ten) * Omega4
              -0.56104406 * exp ( -2 * Ten) * Omega2 +1;
        F4 := -0.21179193 * exp (-11 * Ten) * Omega6
              +0.5172811 * exp ( -7 * Ten) * Omega4
              -0.17946748 * exp ( -3 * Ten) * Omega2 +0.10543059;
        K0 := 4.35373;
      end;

    ( Calculate frequency weighting for vertical foot motion. )

    if (Motion = FEET_VERTICAL) then
      begin
        F1 := 0.58657402 * exp ( -7 * Ten) * Omega4
              -0.18824531 * exp ( -2 * Ten) * Omega2 +1;
        F2 := -0.18706955 * exp ( -4 * Ten) * Omega2 +0.074036789;

```

```

F3 := 0.33913154 *exp ( -6 *Ten) *Omega4
      -0.23697592 *exp ( -2 *Ten) *Omega2 +1;
F4 := 0.17013499 *exp ( -8 *Ten) *Omega4
      -0.39439090 *exp ( -4 *Ten) *Omega2 +0.074036789;
K0 := 1.182;
end;

( Calculate frequency weighting for fore-aft motion. )

if (Motion = SEAT_FORE_AFT) then
begin
  F1 := 1;
  F2 := 0.219106;
  F3 := -0.0185309 *Omega2 +1;
  F4 := -0.000618934 *Omega2 +0.219106;
  K0 := 4.3532;
end;

( Calculate frequency weighting for lateral motion. )

if (Motion = SEAT_LATERAL) then
begin
  F1 := 0.24052124 *exp ( -3 *Ten) *Omega4
      -0.066974483 *Omega2 +1;
  F2 := 0.57384538 *exp ( -5 *Ten) *Omega4
      -0.50170413 *exp ( -2 *Ten) *Omega2 +0.33092592;
  F3 := -0.14979958 *exp ( -5 *Ten) *Omega6
      +0.10088882 *exp ( -2 *Ten) *Omega4
      -0.10108617 *Omega2 +1;
  F4 := -0.17137490 *exp ( -7 *Ten) *Omega6
      +0.53137351 *exp ( -4 *Ten) *Omega4
      -0.011096507 *Omega2 +0.33092592;
  K0 := 4.353;
end;

K2 := K1 *K0 *(F1 *F4 -F2 *F3) /(sqr (F3) +Omega2 *sqr (F4));

( Find weighted PSD (absorbed PSD). )

Magnitude := K2 *(sqr (Hr^[L]) +sqr (Hi^[L]));

if (FName <> 'DEFAULT') then
  writeln (FN1, Hertz, ' ', Magnitude);

( Use Simpson's method to integrate area beneath absorbed PSD. )

if (Even (L)) then
  Weight := 2
else

```



```

    Weight := 4;
    if (L = 0) then Weight := 2;

(   Include conversion factor from m/s^2 to ft/sec^2.   )

    AbPower := AbPower +Weight *Magnitude /sqr (0.3048);

end;

(   Divide total absorbed energy by time of run to calculate
|   average absorbed power.
)
AbPower := AbPower /(3 *NumFFT *(TimeStop -TimeStart) *DeltaTime);

if (FName <> 'DEFAULT') then close (FN1);
end;

```

```

procedure Griffin (Motion : Axis;
                   DeltaTime, TimeStop, TimeStart : real;
                   NumFFT, NumTime : integer;
                   FName : string;
                   RMag, IMag : RealPointer;
                   var Crest, ERMS, VDV, T15 : real;
                   var Comfort : integer);

(   This procedure determines ride characteristics using Griffin's
|   Vibration Dose Value approach. Note that this requires
|   calculation of the IFFT of the frequency weighted FFT.
|
|   Average.....average acceleration in time domain.
|   Comfort.....comfort index based on rms acceleration.
|   Crest.....crest factor of weighted acceleration.
|   DeltaTime....sampling rate in time.
|   ERMS.....estimated rms acceleration.
|   Factor.....weighting factor due to frequency.
|   FName.....file name for weighted acceleration data.
|   Hertz.....frequency of signal.
|   IMag.....imaginary part of acceleration.
|   L.....counter.
|   Magnitude....weighted acceleration in time.
|   MaxPeak.....maximum acceleration.
|   MinPeak.....minimum acceleration.

```

```

| Motion.....axis of analysis.
| NumDummy.....dummy variable.
| NumFFT.....number of frequency samples.
| NumTime.....number of time samples.
| Peak.....peak variation of acceleration from mean.
| RMag.....real part of acceleration.
| TimeStop.....time at end of run.
| TimeStart.....time at start of run.
| T15.....time to 15 VDV.
| VDV.....vibration dose value.
| Weight.....weighting for Simpson integration.
| Weighting....code for weighting.
|
)

var
  Average, Factor, Hertz, Magnitude, MaxPeak, MinPeak,
  Peak, Weight : real;
  L, NumDummy : integer;
  Weighting : Code;

begin
  ( Determines, according to the acceleration axis, which weightings
  | to use.
  )
  if ( (Motion = SEAT_VERTICAL)
    or (Motion = FEET_FORE_AFT)
    or (Motion = FEET_VERTICAL)
    or (Motion = FEET_LATERAL )) then Weighting := B;

  if ( (Motion = BACK_FORE_AFT)) then Weighting := C;

  if ( (Motion = SEAT_FORE_AFT)
    or (Motion = SEAT_LATERAL )
    or (Motion = BACK_VERTICAL)
    or (Motion = BACK_LATERAL )) then Weighting := D;

  ( Weight each component of the acceleration-time history FFT.  )

  for L:=0 to (NumFFT-1) do
    begin
      Hertz := abs ((L-NumFFT/2) /DeltaTime /NumFFT);

  ( Band pass components between 0.5 and 80 Hz, and reject all others.
  | These frequencies come directly from ISO 2631.
  )
    if ((Hertz < 0.5) or (Hertz > 80)) then
      begin

```

```

    RMag^[L] := 0;
    IMag^[L] := 0;
    Factor := 1;
end;

if (Weighting = B) then
begin
    if ((Hertz >= 0.5) and (Hertz < 2.0)) then Factor := 0.4;
    if ((Hertz >= 2.0) and (Hertz < 5.0)) then
        Factor := 0.2 * Hertz;
    if ((Hertz >= 5.0) and (Hertz < 16.0)) then Factor := 1.0;
    if ((Hertz >= 16.0) and (Hertz < 80.0)) then
        Factor := 16 / Hertz;
end;

if (Weighting = C) then
begin
    if ((Hertz >= 0.5) and (Hertz < 8.0)) then Factor := 1.0;
    if ((Hertz >= 8.0) and (Hertz < 80.0)) then
        Factor := 8.0 / Hertz;
end;

if (Weighting = D) then
begin
    if ((Hertz >= 0.5) and (Hertz < 2.0)) then Factor := 1.0;
    if ((Hertz >= 2.0) and (Hertz < 80.0)) then
        Factor := 2.0 / Hertz;
end;

( Weight the components. )
    RMag^[L] := RMag^[L] * sqrt (Factor);
    IMag^[L] := IMag^[L] * sqrt (Factor);
end;

( Calculate the inverse FFT of the weighted FFT. Note that a real
| time signal produces an even frequency signal, and that an even
| frequency signal must produce a real time signal. Since positive
| and negative frequencies have been weighted symmetrically, the
| IFFT must be real (ie have zero imaginary component). However,
| due to machine precision, truncation of the time signal, etc.,
| the IFFT has a small imaginary component. This is included in
| the following analysis, but could probably be ignored.
| )
    FFT (FName, 1/DeltaTime/NumFFT, (-NumFFT/2)/DeltaTime/NumFFT,
        NumFFT-1, 'F->T', RMag, IMag, NumDummy);

( Set up parameters for VDV analysis. )

VDV := 0;

```

```

MinPeak := 999999;
MaxPeak := -999999;
Average := 0;

for L:=0 to NumTime do
  begin

( Calculate the quad of the frequency weighted acceleration-time
| history. Note the inclusion of the small imaginary component.
)
    Magnitude := sqr (sqr (RMag^[L]) +sqr (IMag^[L]));

( Find the maximum, minimum, and average accelerations. Note that
| for convenience the small imaginary component has been ignored.
)
    if (RMag^[L] >= MaxPeak) then MaxPeak := RMag^[L];
    if (RMag^[L] <= MinPeak) then MinPeak := RMag^[L];
    Average := Average + RMag^[L];

( Use Simpson's method to integrate quad acceleration. )

    if (Even (L)) then
      Weight := 2
    else
      Weight := 4;

    if (L = 0) then Weight := 2;
    VDV := VDV +Weight *Magnitude;
  end;

( Calculate average acceleration, and find greatest deviation from
| the average.
)
  Average := Average / NumTime;
  if (abs (Average -MinPeak) > abs (MaxPeak -Average)) then
    Peak := abs (Average -MinPeak)
  else
    Peak := abs (MaxPeak - Average);

( Calculate the Vibration Dose Value, the estimated RMS acceleration,
| the time to 15 VDV in seconds, and the crest factor for the
| weighted acceleration.
)
  VDV := sqrt (sqrt (VDV /(3 *NumTime) *(TimeStop -TimeStart)));
  ERMS := VDV /(1.4 *sqrt (sqrt (TimeStop -TimeStart)));
  if (VDV = 0) then
    T15 := 0
  else
    T15 := sqr (sqr (15 /VDV)) *(TimeStop -TimeStart);

```

```

if (ERMS = 0) then
  Crest := 0
else
  Crest := Peak / ERMS;

( Calculate a comfort code based on the estimated RMS value of
| acceleration. This ranges from "not uncomfortable" (0) to
| "extremely uncomfortable" (5). These ratings are meaningful
| for crest factors less than 6.
)

if (ERMS < 0.315) then
  Comfort := 0
else
  if (ERMS < 0.57) then
    Comfort := 1
  else
    if (ERMS < 0.9) then
      Comfort := 2
    else
      if (ERMS < 1.4) then
        Comfort := 3
      else
        if (ERMS < 2.25) then
          Comfort := 4
        else
          Comfort := 5;

end;

procedure ISO2631 (Motion : Axis;
  DeltaTime, TimeStop, TimeStart : real;
  NumFFT, NumTime : integer;
  FName : string;
  RMag, IMag : BigPointer;
  var Crest, RMSWeighted, TExplim, TDecPro,
  TRedCom : real;
  var RMS : Spectrum);

var
  Acc, Average, Cube2, Factor, FCentre, FLower, FUpper : real;
  Hertz, Magnitude, MinPeak, MaxPeak, Peak, Weight : real;
  Count, FCIndex, FCCheck, L, NumDummy : integer;
  Weighting : Code;

begin

```

```

( Set up parameters for centre frequency calculations. )

Cube2 := ln (2) /3;
FCIndex := round (ln (0.5 /DeltaTime) / Cube2);
FCentre := exp (FCIndex *Cube2);
FUpper := sqrt (exp (Cube2));
FLower := sqrt (exp (-Cube2));
Count := 0;

( Only calculate the centre frequency accelerations for the negative
| frequencies (the FFT is an even function). Remember to multiply
| the accelerations by two to account for the positive frequencies
| in the spectrum.
)
L := 0;
while (L < (NumFFT/2)) do
  begin
    Hertz := abs ((L -NumFFT /2) /DeltaTime /NumFFT);

( Check that we haven't moved into a new third-octave band. )

    FCCheck := round (ln (Hertz) /Cube2);
    if (FCCheck <> FCIndex) then
      begin

( If we have started a new third octave band, do the final
| calculation of rms acceleration based on Simpson's method.
)
        if (FCIndex <= 19) then
          RMS[FCIndex] := sqrt ((RMS[FCIndex]
                                +2 *(sqr (RMag^[L]) +sqr (IMag^[L])))
                                /(3 *(Count+1)) *(FUpper -FLower) *FCentre
                                /(TimeStop -TimeStart));

( If FCCheck is less than zero, then we are beyond the first third-
| octave band (centre frequency = 1 Hz). Reset L so that the next
| 'while' test will fail.
)
        if (FCCheck < 0) then
          L := round (NumFFT /2) + 1;

( Set up parameters for next third-octave band. )

        FCIndex := FCCheck;
        Count := 0;
        FCentre := exp (FCIndex *Cube2);
      end;

( Use Simpson's method to integrate the PSD across the third-octave

```

```

| band.
)
    if (Even (Count)) then
        Weight := 2
    else
        Weight := 4;

    if (Count = 0) then Weight := 1;

( Band-pass frequencies between 0.5 and 80 Hz, and reject all
| others.
)
    if ((Hertz < 0.5) or (Hertz > 80)) then
        begin
            Weight := 0;
        end;

( Check range of centre frequency index to prevent error. )

    if ((FCIndex >= 0) and (FCIndex <= 19)) then
        RMS[FCIndex] := RMS[FCIndex] + 2 *Weight
            *(sqr (RMagL) +sqr (IMagL));

    inc (Count);
    inc (L);
end;

( Determine which weighting and which 1 minute reference
| acceleration to use.
)
    if (Motion = SEAT_VERTICAL) then
        begin
            Weighting := 8;
            Acc := 2.8;
        end;

    if ( (Motion = SEAT_FORE_AFT)
        or (Motion = SEAT_LATERAL )) then
        begin
            Weighting := 0;
            Acc := 2.0;
        end;

( Weight each component of the acceleration-time history FFT. )

for L:=0 to (NumFFT-1) do
    begin
        Hertz := abs ((L -NumFFT /2) /DeltaTime /NumFFT);

```

```

( Band pass components between 0.5 and 80 Hz, and reject all
| others.
)
    if ((Hertz < 0.5) or (Hertz > 80)) then
        begin
            RMag^[L] := 0;
            IMag^[L] := 0;
            Factor := 1;
        end;

    if (Weighting = B) then
        begin
            if ((Hertz >= 0.5) and (Hertz < 1.0)) then Factor := 0.5;
            if ((Hertz >= 1.0) and (Hertz < 4.0))
                then Factor := 0.185 * Hertz + 0.33;
            if ((Hertz >= 4.0) and (Hertz < 8.0)) then Factor := 1.0;
            if ((Hertz >= 8.0) and (Hertz < 80.0)) then
                Factor := 8 / Hertz;
        end;

    if (Weighting = D) then
        begin
            if ((Hertz >= 0.5) and (Hertz < 2.0)) then Factor := 1.0;
            if ((Hertz >= 2.0) and (Hertz < 80.0)) then
                Factor := 2.0 / Hertz;
        end;

( Weight the components. )

    RMag^[L] := RMag^[L] * sqrt (Factor);
    IMag^[L] := IMag^[L] * sqrt (Factor);
end;

( Calculate the inverse FFT of the weighted FFT. Note that a real
| time signal produces an even frequency signal, and that an even
| frequency signal must produce a real time signal. Since positive
| and negative frequencies have been weighted symmetrically, the
| IFFT must be real (ie have zero imaginary component). However,
| due to machine precision, truncation of the time signal, etc.,
| the IFFT has a small imaginary component. This is included in
| the following analysis, but could probably be ignored.
)
    FFT (FName, 1/DeltaTime/NumFFT, (-NumFFT/2)/DeltaTime/NumFFT,
        NumFFT-1, 'F->T', RMag, IMag, NumDummy);

( Set up parameters for ISO analysis. )

RMSWeighted := 0;
MinPeak     := 999999;

```



```

MaxPeak      := -999999;
Average      := 0;

for L:=0 to NumTime do
  begin

( Calculate the square of the frequency weighted acceleration-time
| history. Note the inclusion of the small imaginary component.
)
    Magnitude := sqr (RMag^[L]) +sqr (IMag^[L]);

( Find the maximum, minimum, and average accelerations. Note that
| for convenience the small imaginary component has been ignored.
)
    if (RMag^[L] >= MaxPeak) then MaxPeak := RMag^[L];
    if (RMag^[L] <= MinPeak) then MinPeak := RMag^[L];
    Average := Average + RMag^[L];

( Use Simpson's method to integrate square acceleration.  )

    if (Even (L)) then
      Weight := 2
    else
      Weight := 4;

    if (L = 0) then Weight := 2;
    RMSWeighted := RMSWeighted +Weight *Magnitude;
  end;

( Calculate average acceleration, and find greatest deviation from
| the average.
)
  Average := Average / NumTime;
  if (abs (Average -MinPeak) > abs (MaxPeak -Average)) then
    Peak := abs (Average -MinPeak)
  else
    Peak := abs (MaxPeak - Average);

( Calculate the weighted RMS acceleration, crest factor for the
| weighted acceleration, and times to exposure limit, fatigue-
| decreased proficiency, and reduced comfort boundaries.
)
  RMSWeighted := sqrt (RMSWeighted /(3 *NumTime));
  if (RMSWeighted = 0) then
    begin
      Crest := 0;
      TExLim := 0;
      TDecPro := 0;
      TRedCom := 0;
    end
  end

```

```
end else
  begin
    Crest := Peak / RMSWeighted;
    TExplim := 10 *sqr (Acc *2 /RMSWeighted);
    TDecPro := 10 *sqr (Acc /RMSWeighted);
    TRedCom := 10 *sqr (Acc /3.15 /RMSWeighted);
  end;
end;

begin
end.
```

```

*****
|
|               program Ride-Q
|
|       (c) 1990   Michael Campbell Galway
|
|       The purpose of this program is to quantify the ride quality of a
|       vehicle using three ride quality criteria.  These are:
|
|       a.  absorbed power, a concept developed at the Tank Automotive
|           Command (TACOM), an agency of the United States Department
|           of the Army, and widely used in military ride analysis;
|
|       b.  ISO 2631, a system adopted from the U.S. Army by the
|           International Organization for Standardization, and commonly
|           referred to by researchers in all fields concerned with
|           human vibration response; and
|
|       c.  Griffin's vibration dose value (VDV), a recent response to
|           some of the shortcomings of ISO 2631, particularly with
|           regard to high-crest factor acceleration environments.
|           Griffin's VDV has been applied in some British ride
|           models.
|
|       All of the criteria require a vehicle's acceleration-time history,
|       which here are generated by the Ride Prediction Model's main
|       processor.
|
|       BColour1.....background colour of process status screen.
|       BColour2.....background colour of process status box.
|       BColour3.....background colour of solution boxes.
|       Colour.....0 if monochrome, 1 if colour adapter.
|       FColour1.....text colour.
|       FColour2.....text colour.
|       FColour3.....text colour.
|       FName1.....vehicle data file name.
|       FName2.....control file name.
|       F1.....meta-control file.
|       F2.....vehicle data file.
|       F3.....process control file.
|       GraphDriver.....type of graphics driver detected.
|       GraphMode.....graphics mode detected.
|       I.....a counter.
|       J.....a counter
|
| *****

```

```
uses Common, crt, Environ, FastFT, graph, PostProc, Sup_Proc;
```

```
var
```

```

Auto, Colour, EqnNum, GraphDriver      : integer;
GraphMode, I, Iterate, J, K, L, M      : integer;
N, NN, OrientationCode                  : integer;
Comfort                                 : array [1..3,1..3] of integer;
BColour1, BColour2, BColour3            : byte;
FColour1, FColour2, FColour3            : byte;

F1, F2, F3, F4, F5, F6, F7, F8         : text;
FileA, FileB, FileC                    : string;
FName2, FName3, FName4, FName5          : string;
FName6, FName7, FName8                  : string;

AHeave, APitch, ARoll, DeltaTime         : real;
Dist, Dummy, Heave, Hertz, Magnitude, Pitch : real;
Roll, Time, Time0, TimeDecPro            : real;
T15VDV, TotAbPower, TotRMS, TotVDV       : real;
VHeave, VPitch, Vroll, Weight            : real;
Ai                                         : BigPointer;

AbPower, ERMS, GCrest, ISOCrest          : OutArray;
RMSWeighted, TDecPro, TExplim            : OutArray;
TRedCom, T15, VDV                        : OutArray;

RMS                                       : array [1..3,1..3] of Spectrum;

X, Y, Z                                  : array [1..3] of real;
A                                         : array [1..3] of BigPointer;

Orientation                              : Orient;
Motion                                   : array [1..3,1..3] of Axis;
Back, Feet                               : boolean;
```

```
label 10;
```

```
begin
```

```
( Make space on the heap for acceleration data. )
```

```

new (A[1]);
new (A[2]);
new (A[3]);
new (Ai);
```

```

( Initialize window information (for display). Determine whether
| monochrome graphics adapter is present or if user requested
| monochrome override (by using command line parameter 'C'). Set
| the colour flag Colour appropriately to 1 or zero.
|
| Remove the cursor so it won't appear on the process status screen.
)

```

```

InitWindowInfo;
detectgraph (GraphDriver, GraphMode);
Colour := 0;
if (paramcount > 0) then Colour := 1;
if ((BooleanMono = true) or (graphdriver = 7)) then Colour := 2,

```

```

CursorSize (CursorInvisible, BooleanMono);

```

```

( Set background colours and text colours for the process status
| screen. Note that colours are automatically set to black or white
| for monochrome displays. Colours are:
|
| 0....black
| 1....blue
| 3....cyan
| 7....gray
| 15...white
)

```

```

case Colour of
  0 : begin
    BColour1 := 1;
    BColour2 := 3;
    BColour3 := 7;

    FColour1 := 15;
    FColour2 := 0;
    FColour3 := 0;
    end;
  1 : begin
    BColour1 := 0;
    BColour2 := 0;
    BColour3 := 15;

    FColour1 := 15;
    FColour2 := 15;
    FColour3 := 0;
    end;
  2 : begin
    BColour1 := 0;
    BColour2 := 0;
    BColour3 := 15;

```

```

        FColour1 := 15;
        FColour2 := 7;
        FColour3 := 0;
        end;
    end;

( Set up the background for the process status screen (title
| line, exit message box).
)
    textcolor (FColour1);
    textbackground (BColour1);
    clrscr;

    gotoxy (17, 2);
    write ('Ride Prediction Model - Ride Quality Criteria');
    gotoxy (17,3);
    write ('      (c) 1990  Michael Campbell Galway');
    OpenWindow (8, 19, 72, 25, BColour1, FColour1, single);
    gotoxy (2, 2);
    write ('Process Message:');
    textbackground (BColour3);
    OpenWindow (10, 22, 69, 23, BColour3, FColour3, none);

    textcolor (FColour3 + 128);
    gotoxy (2, 1);
    write ('Working:');
    gotoxy (2, 2);
    textcolor (FColour3);
    write ('Initializing...           ');

( Set up process status box.  )

    OpenWindow (13, 4, 68, 16, BColour2, FColour2, double);
    gotoxy (11, 3);
    write ('Computing FFT...');
    gotoxy (11, 5);
    write ('Computing absorbed power...');
    gotoxy (11, 7);
    write ('Computing Griffins VDV...');
    gotoxy (11, 9);
    write ('Computing ISO 2631...');
    gotoxy (6, 11);
    write ('Percent completed of FULL analysis:');

    OpenWindow (56, 14, 61, 14, BColour3, FColour3, none);
    gotoxy (3, 1);
    write (0, '%');

```

```

window (1, 1, 80, 25);

( Open standard meta-control file Meta.CNT, and read names of
| data files. Open files for reading or writing as appropriate.
)

assign (F1, 'Meta.CNT');
reset (F1);

( File assignments are as follows:
|
|   FName2 - run-time parameters      [input]
|   FName3 - time history data        [input]
|   FName4 - PSD file                 [output]
|   FName5 - absorbed PSD file        [output]
|   FName6 - Griffin weighted time history [output]
|   FName7 - ISO weighted time history [output]
|   FName8 - output report file       [output]
)

readln (F1);
readln (F1, FileA);
readln (F1, FName2);
readln (F1, FileB);
readln (F1, FileC);
readln (F1, FName3);
readln (F1, FName4);
readln (F1, FName5);
readln (F1, FName6);
readln (F1, FName7);
readln (F1, FName8);

assign (F2, FName2);
assign (F3, FName3);
assign (F8, FName8);

reset (F2);

( If the report file exists, append new information to it.
| If the file doesn't exist, create it.
)

($I-)
append (F8);
if (ioresult = 2) then rewrite (F8);
($I+)

( Read seat, feet, and back coordinates from data file. The indices
| are assigned as follows:
|
|   index 1 = seat;

```

```
|    index 2 = feet;  
|    index 3 = back;  
}
```

```
readln (F2, Dummy, Dummy, Dummy, Dummy, Dummy, Auto);
readln (F2, X(1), Y(1), Z(1));
readln (F2, X(2), Y(2), Z(2));
readln (F2, X(3), Y(3), Z(3));
readln (F2, OrientationCode);
```

```
if (OrientationCode = 0) then Orientation := FRONTWARD;
if (OrientationCode = 1) then Orientation := SIDEWAYS;
```

```
( Initialize motion direction flags. )
```

```

Motion[1,1] := SEAT_VERTICAL;
Motion[1,2] := SEAT_LATERAL;
Motion[1,3] := SEAT_FORE_AFT;
Motion[2,1] := FEET_VERTICAL;
Motion[2,2] := FEET_LATERAL;
Motion[2,3] := FEET_FORE_AFT;
Motion[3,1] := BACK_VERTICAL;
Motion[3,2] := BACK_LATERAL;
Motion[3,3] := BACK_FORE_AFT;

```

```

for I:=1 to 3 do
  for J:=1 to 3 do
    begin
      AbPower      [I,J] := 0;
      GCrest       [I,J] := 0;
      ERMS         [I,J] := 0;
      VDV          [I,J] := 0;
      T15          [I,J] := 0;
      Comfort      [I,J] := 0;
      ISOCrest     [I,J] := 0;
      RMSweighted  [I,J] := 0;
      TExplim      [I,J] := 0;
      TDecPro      [I,J] := 0;
      TRedCom      [I,J] := 0;
    end
  end
end

```

```

for M:=0 to 19 do
  RMS [I,J,M] := 0;
end;

```

```
Feet := true;
Back := true;
```

(If the coordinates for the back or feet are identical to the seat
| coordinates, the user is indicating foot or back inputs are to be


```

| ignored.

if ((X[2] = X[1]) and (Y[2] = Y[1]) and (Z[2] = Z[1])) then
    Feet := false;

if ((X[3] = X[1]) and (Y[3] = Y[1]) and (Z[3] = Z[1])) then
    Back := false;

( Do separate analyses for the seat, feet, and back. )

for I:=1 to 3 do
    begin

        reset (F3);

( If the user has indicated that foot or back inputs are not to
| be considered, then skip the analysis for feet or back.
)

        if ((I = 2) and (Feet = false)) then goto 10;
        if ((I = 3) and (Back = false)) then goto 10;

        if (I = 1) then
            begin
                gotoxy (11, 23);
                write ('Seat analysis - reading acceleration time history...');
            end;

        if (I = 2) then
            begin
                gotoxy (11, 23);
                write ('Feet analysis - reading acceleration time history...');
            end;

        if (I = 3) then
            begin
                gotoxy (11, 23);
                write ('Back analysis - reading acceleration time history...');
            end;

( Reset the pointer of the time-history file to the file beginning
| for each analysis.
)

        K := 0;

( Read in the first record to find the initial time. )

        readln (F3, Time0, Dist, Heave, Pitch, Roll, VHeave,
                VPitch, VRoll, AHeave, APitch, ARoll);
        MoveAxes (Orientation, X[I], Y[I], Z[I], Heave, Pitch, Roll,

```

```

VHeave, VPitch, VRoll, AHeave, APitch, ARoll,
A[3]^K, A[1]^K, A[2]^K);

while (not eof (F3)) do
begin
K := K + 1;

( Until the end of the file is reached, input time history data.
| Use K to count the number of records input.
)

readln (F3, Time, Dist, Heave, Pitch, Roll, VHeave,
VPitch, VRoll, AHeave, APitch, ARoll);

( Switch from the global coordinates to the driver/passenger
| coordinates. Calculate vertical, lateral, and fore-aft
| accelerations taking into account centripetal and angular
| accelerations, as well as the global vertical acceleration.
)

MoveAxes (Orientation, X[I], Y[I], Z[I], Heave, Pitch, Roll,
VHeave, VPitch, VRoll, AHeave, APitch, ARoll,
A[3]^K, A[1]^K, A[2]^K);

end;

DeltaTime := abs (Time -Time0) /(K);

( Do an analysis for each of the vertical, lateral, and fore-aft
| directions.
)

for J:=1 to 3 do
begin

gotoxy (57, 14);
write (((I -1) *100 div 3 +(J -1) *100 div 9):3, '%');

if (J = 1) then
begin
gotoxy (27, 23);
write ('examining vertical direction... ');
end;

if (J = 2) then
begin
gotoxy (27, 23);
write ('examining lateral direction... ');
end;

if (J = 3) then
begin
gotoxy (27, 23);

```

```

        write ('examining fore-aft direction... ');
    end;

( Find the power spectral density (PSD) for the acceleration in each
| axis. (Actually, the PSD is the square of the magnitude of
| the FFT, not calculated at this point.
)

    OpenWindow (52, 6, 57, 6, BColour3, FColour3+128, none),
    gotoxy (1,1);
    write (' WAIT ');
    FFT (FName4, DeltaTime, Time0, N, 'T->F', A[J], Ai, N);
    OpenWindow (52, 6, 57, 6, BColour2, FColour3, none);

    if (Motion[I,J] < FEET_LATERAL) then
        begin
            OpenWindow (52, 8, 57, 8, BColour3, FColour3+128, none);
            gotoxy (1,1);
            write (' WAIT ');
            Power (Motion[I,J], DeltaTime, Time, Time0, N,
                    FName5, A[J], Ai, AbPower[I,J]);
            OpenWindow (52, 8, 57, 8, BColour2, FColour3, none);
        end;

    OpenWindow (52, 10, 57, 10, BColour3, FColour3+128, none);
    gotoxy (1,1);
    write (' WAIT ');
    Griffin (Motion[I,J], DeltaTime, Time, Time0, N, K,
            FName6, A[J], Ai, GCrest[I,J], ERMS[I,J], VDV[I,J],
            T15[I,J], Comfort[I,J]);
    OpenWindow (52, 10, 57, 10, BColour2, FColour3, none);

    if (Motion[I,J] < FEET_VERTICAL) then
        begin
            OpenWindow (52, 12, 57, 12, BColour3, FColour3+128, none);
            gotoxy (1,1);
            write (' WAIT ');
            ISO2631 (Motion[I,J], DeltaTime, Time, Time0, N, K,
                    FName7, A[J], Ai, ISOCrest[I,J], RMSWeighted[I,J],
                    TExLim[I,J], TDecPro[I,J], TRedCom[I,J],
                    RMS[I,J]);
            OpenWindow (52, 12, 57, 12, BColour2, FColour3, none);
        end;

    textbackground (BColour3);
    window (1, 1, 80, 25);
end;
10:end;

```

```

gotoxy (57, 14);
write ('100%');

gotoxy (11, 23);
write ('Calculating final ride parameters... ');

Tc:AbPower := 0;
TotVDV := 0;
TotRMS := -0.96 *sqr (RMSWeighted[1,1]);

for I:=1 to 3 do
  for J:=1 to 3 do
    begin
      TotAbPower := TotAbPower + AbPower[I,J];

      if (Motion[I,J] < FEET_VERTICAL) then
        Weight := 1
      else
        if (Motion[I,J] < FEET_LATERAL) then
          Weight := 0.4
        else
          if (Motion[I,J] < BACK_VERTICAL) then
            Weight := 0.25
          else
            if (Motion[I,J] < BACK_LATERAL) then
              Weight := 0.4
            else
              if (Motion[I,J] < BACK_FORE_AFT) then
                Weight := 0.5
              else
                Weight := 0.8;

      TotVDV := TotVDV +sqr (sqr (VDV[I,J] *Weight));

      TotRMS := TotRMS +sqr (1.4 *RMSWeighted[I,J]);
    end;

TotVDV := sqrt (sqrt (TotVDV));
if (TotVDV = 0) then
  T15VDV := 0
else
  T15VDV := sqr (sqr (15 /TotVDV)) *(Time -Time0);

TotRMS := sqrt (TotRMS);
if (TotRMS = 0) then
  TimeDecPro := 0
else
  TimeDecPrc := sqr (2.8 /TotRMS) *10;

```

```

( Append ride quality information to the output report. )

gotoxy (11, 23);
write ('Creating ride quality report... ');

writeln (F8, '*****'+
'*****');
writeln (F8, '                RIDE QUALITY REPORT');
writeln (F8, '*****'+
'*****');

writeln (F8);
writeln (F8, 'Input Files...');
writeln (F8, 'Vehicle parameters      : ', FileA);
writeln (F8, 'Run-time parameters      : ', FName2);
writeln (F8, 'Terrain profile data     : ', FileB);
writeln (F8, 'Hub path data            : ', FileC);
writeln (F8);
writeln (F8, 'Output Files...');
writeln (F8, 'Time-history              : ', FName3);
writeln (F8, 'PSD                      : ', FName4);
writeln (F8, 'Absorbed PSD              : ', FName5);
writeln (F8, 'Griffin weighted time-history : ', FName6);
writeln (F8, 'ISO weighted time-history  : ', FName7);
writeln (F8, 'Ride quality report       : ', FName8);

writeln (F8);
writeln (F8, 'A.  ABSORBED POWER:');
writeln (F8, '      Seat, Vertical -', AbPower[1,1]:8:1, ' watts');
writeln (F8, '      Seat, Lateral -', AbPower[1,2]:8:1, ' watts');
writeln (F8, '      Seat, Fore-Aft -', AbPower[1,3]:8:1, ' watts');
writeln (F8, '      Feet, Vertical -', AbPower[2,1]:8:1, ' watts');
writeln (F8, '      Sum of Average Absorbed Powers -',
TotAbPower:8:1, ' watts');

writeln (F8);
writeln (F8, 'B.  ISO 2631:');
writeln (F8);
writeln (F8, '  I.  Third Octave Band Analysis:');
writeln (F8);
writeln (F8, '  fc (Hz)      Seat      Seat      Seat');
writeln (F8, '                Vertical  Lateral  '+
'Fore-Aft');

writeln (F8);
writeln (F8, '  1.0 ', RMS[1,1,0]:15:2, RMS[1,2,0]:15:2,
RMS[1,3,0]:16:2);
writeln (F8, '  1.25 ', RMS[1,1,1]:15:2, RMS[1,2,1]:15:2,
RMS[1,3,1]:16:2);
writeln (F8, '  1.6 ', RMS[1,1,2]:15:2, RMS[1,2,2]:15:2,
RMS[1,3,2]:16:2);
writeln (F8, '  2.0 ', RMS[1,1,3]:15:2, RMS[1,2,3]:15:2,

```

```

      RMS(1,3,3):16:2);
writeln (F8, ' 2.5 ', RMS(1,1,4):15:2, RMS(1,2,4):15:2,
      RMS(1,3,4):16:2);
writeln (F8);
writeln (F8, ' 3.15', RMS(1,1,5):15:2, RMS(1,2,5):15:2,
      RMS(1,3,5):16:2);
writeln (F8, ' 4.0 ', RMS(1,1,6):15:2, RMS(1,2,6):15:2,
      RMS(1,3,6):16:2);
writeln (F8, ' 5.0 ', RMS(1,1,7):15:2, RMS(1,2,7):15:2,
      RMS(1,3,7):16:2);
writeln (F8, ' 6.3 ', RMS(1,1,8):15:2, RMS(1,2,8):15:2,
      RMS(1,3,8):16:2);
writeln (F8, ' 8.0 ', RMS(1,1,9):15:2, RMS(1,2,9):15:2,
      RMS(1,3,9):16:2);
writeln (F8);
writeln (F8, ' 10.0 ', RMS(1,1,10):15:2, RMS(1,2,10):15:2,
      RMS(1,3,10):16:2);
writeln (F8, ' 12.5 ', RMS(1,1,11):15:2, RMS(1,2,11):15:2,
      RMS(1,3,11):16:2);
writeln (F8, ' 16.0 ', RMS(1,1,12):15:2, RMS(1,2,12):15:2,
      RMS(1,3,12):16:2);
writeln (F8, ' 20.0 ', RMS(1,1,13):15:2, RMS(1,2,13):15:2,
      RMS(1,3,13):16:2);
writeln (F8, ' 25.0 ', RMS(1,1,14):15:2, RMS(1,2,14):15:2,
      RMS(1,3,14):16:2);
writeln (F8);
writeln (F8, ' 31.5 ', RMS(1,1,15):15:2, RMS(1,2,15):15:2,
      RMS(1,3,15):16:2);
writeln (F8, ' 40.0 ', RMS(1,1,16):15:2, RMS(1,2,16):15:2,
      RMS(1,3,16):16:2);
writeln (F8, ' 50.0 ', RMS(1,1,17):15:2, RMS(1,2,17):15:2,
      RMS(1,3,17):16:2);
writeln (F8, ' 63.0 ', RMS(1,1,18):15:2, RMS(1,2,18):15:2,
      RMS(1,3,18):16:2);
writeln (F8, ' 80.0 ', RMS(1,1,19):15:2, RMS(1,2,19):15:2,
      RMS(1,3,19):16:2);
writeln (F8);
writeln (F8, 'Notes:');
writeln (F8, ' 1. fc = centre frequency of third-octave band. ');
writeln (F8, ' 2. Accelerations are rms acceleration over the '+
      ' third-octave ');
writeln (F8, ' band in m/s/s. ');
writeln (F8);
writeln (F8);
writeln (F8, ' 11. Overall analysis: ');
writeln (F8);
writeln (F8, '
      A(rms)  CF      T-EL      T-FDP'
      '+'      T-RC ');
writeln (F8);

```

```

writeln (F8, 'Seat, Vertical ', RMSWeighted[1,1]:7:1,
          ISOCrest[1,1]:7:1,
          TExplim[1,1]:9:1, TDecPro[1,1]:9:1, TRedCom[1,1]:9:1);
writeln (F8, 'Seat, Lateral ', RMSWeighted[1,2]:7:1,
          ISOCrest[1,2]:7:1,
          TExplim[1,2]:9:1, TDecPro[1,2]:9:1, TRedCom[1,2]:9:1);
writeln (F8, 'Seat, Fore-Aft ', RMSWeighted[1,3]:7:1,
          ISOCrest[1,3]:7:1,
          TExplim[1,3]:9:1, TDecPro[1,3]:9:1, TRedCom[1,3]:9:1);
writeln (F8);
writeln (F8, '          Total ', TotRMS:8:1, TimeDecPro:25:1);
writeln (F8);
writeln (F8, 'Notes:');
writeln (F8, '  1. RMS accelerations in m/s.s. ');
writeln (F8, '  2. CF is the crest factor of the frequency '+
          'weighted rms');
writeln (F8, '          acceleration. ');
writeln (F8, '  3. T-EL is the time to the exposure limit boundary '+
          'in');
writeln (F8, '          minutes. ');
writeln (F8, '  4. T-FDP is the time to the fatigue-decreased '+
          'proficiency');
writeln (F8, '          boundary in minutes. ');
writeln (F8, '  5. T-RC is the time to the reduced comfort '+
          'boundary in');
writeln (F8, '          minutes. ');
writeln (F8);
writeln (F8, 'C. GRIFFIN', #39, 'S VIBRATION DOSE VALUE:');
writeln (F8);
writeln (F8, '          A(rms)  CI   CF   VDV   Time to '+
          '15 VDV');
writeln (F8);
writeln (F8, 'Seat, Vertical', ERMS[1,1]:8:1, Comfort[1,1]:4,
          GCrest[1,1]:7:1, VDV[1,1]:8:1, T15[1,1]:13:1);
writeln (F8, 'Seat, Lateral ', ERMS[1,2]:8:1, Comfort[1,2]:4,
          GCrest[1,2]:7:1, VDV[1,2]:8:1, T15[1,2]:13:1);
writeln (F8, 'Seat, Fore-Aft', ERMS[1,3]:8:1, Comfort[1,3]:4,
          GCrest[1,3]:7:1, VDV[1,3]:8:1, T15[1,3]:13:1);
writeln (F8);
writeln (F8, 'Feet, Vertical', ERMS[2,1]:8:1, Comfort[2,1]:4,
          GCrest[2,1]:7:1, VDV[2,1]:8:1, T15[2,1]:13:1);
writeln (F8, 'Feet, Lateral ', ERMS[2,2]:8:1, Comfort[2,2]:4,
          GCrest[2,2]:7:1, VDV[2,2]:8:1, T15[2,2]:13:1);
writeln (F8, 'Feet, Fore-Aft', ERMS[2,3]:8:1, Comfort[2,3]:4,
          GCrest[2,3]:7:1, VDV[2,3]:8:1, T15[2,3]:13:1);
writeln (F8);
writeln (F8, 'Back, Vertical', ERMS[3,1]:8:1, Comfort[3,1]:4,
          GCrest[2,1]:7:1, VDV[3,1]:8:1, T15[3,1]:13:1);
writeln (F8, 'Back, Lateral ', ERMS[3,2]:8:1, Comfort[3,2]:4,

```

```

        GCrest[2,2]:7:1, VDV[3,2]:8:1, T15[3,2]:13:1);
writeln (F8, 'Back, Fore-Aft', ERMS[3,3]:8:1, Comfort[3,3]:4,
        GCrest[2,3]:7:1, VDV[3,3]:8:1, T15[3,3]:13:1);
writeln (F8);
writeln (F8, '          Total', TotVDV:27:1, T15VDV:13:1);
writeln (F8);
writeln (F8, 'Notes:');
writeln (F8, '  1. A(rms) is the rms frequency weighted '+
        'acceleration in');
writeln (F8, '          m/s/s as estimated from the vibration '+
        'dose value.');
```

writeln (F8, ' 2. CI is a comfort index based on rms '+	'acceleration, as');
writeln (F8, ' follows:');	
writeln (F8, ' CI = 0	not uncomfortable');
writeln (F8, ' CI = 1	a little uncomfortable');
writeln (F8, ' CI = 2	fairly uncomfortable');
writeln (F8, ' CI = 3	uncomfortable');
writeln (F8, ' CI = 4	very uncomfortable');
writeln (F8, ' CI = 5	extremely uncomfortable.');

```

writeln (F8);
writeln (F8, '  3. CF is the crest factor of the frequency '+
        'weighted');
writeln (F8, '          acceleration.');
```

writeln (F8, ' 4. Vibration dose value (VDV) is measured in '+	'm/s^1.75.');
writeln (F8, ' 5. The time taken to reach 15 VDV is measured '+	'in seconds.');
writeln (F8, ' 6. The total values are based on the root sum '+	'of quads of');
writeln (F8, ' the individual values.');	

```

close (F8);

sound (523);
delay (60);
nosound;
delay (60);
sound (587);
delay (60);
nosound;

textcolor (FColour3 + 128);
gotoxy (11, 22);
writeln ('Success:');
textcolor (FColour3);
gotoxy (11, 23);
write ('Ride quality calculated. Press any key... ');

```



```
( Keep the process status screen displayed until the user hits any  
| keyboard key. Then reset screen and cursor, and append the  
| equilibrium data to the vehicle data file.  
)
```

```
if (Auto = 1) then  
    repeat until keypressed;
```

```
textbackground (BColour1);  
textcolor (FColour1);
```

```
CursorSize (CursorSmall, true);
```

```
clrscr;  
end.
```

APPENDIX 8

SAMPLE RIDE QUALITY REPORTS

Notes on Appendix 8:

The ride quality reports here are for two different scenarios. The first tabulates the ride quality obtained for an M113A1 travelling at 8 km/h over the sine wave course with low damping rate. The second gives the ride results for the same vehicle crossing the 150 mm half-round obstacle at 24 km/h.

 RIDE QUALITY REPORT

Input Files...

Vehicle parameters : C:\TPASCAL\VEH\M113-A1.DLO
 Run-time parameters : C:\TPASCAL\DATA\APCDATA8.DAT
 Terrain profile data : C:\TPASCAL\TERR\SINE.DAT
 Hub path data : C:\TPASCAL\TERR\M113SINE.DAT

Output Files...

Time-history : D:\RUN\SINE-DLO.DAT
 PSD : DEFAULT
 Absorbed PSD : DEFAULT
 Griffin weighted time-history : DEFAULT
 ISO weighted time-history : DEFAULT
 Ride quality report : D:\RUN\SINE-DLO.REP

A. ABSORBED POWER:

Seat, Vertical - 9.0 watts
 Seat, Lateral - 0.0 watts
 Seat, Fore-Aft - 0.3 watts
 Feet, Vertical - 0.0 watts
 Sum of Average Absorbed Powers - 9.2 watts

B. ISO 2631:

1. Third Octave Band Analysis:

fc (Hz)	Seat Vertical	Seat Lateral	Seat Fore-Aft
1.0	0.24	0.00	0.02
1.25	0.38	0.00	0.04
1.6	0.24	0.00	0.03
2.0	0.40	0.00	0.05
2.5	0.41	0.00	0.06
3.15	0.47	0.00	0.05
4.0	0.54	0.00	0.07
5.0	0.60	0.00	0.08
6.3	0.71	0.00	0.08
8.0	0.77	0.00	0.09
10.0	0.82	0.00	0.10
12.5	0.97	0.00	0.11
16.0	1.09	0.00	0.13
20.0	1.22	0.00	0.15
25.0	1.37	0.00	0.17

31.5	1.53	0.00	0.19
40.0	1.71	0.00	0.22
50.0	1.98	0.00	0.25
63.0	2.31	0.00	0.29
80.0	2.00	0.00	0.23

Notes:

1. f_c = centre frequency of third-octave band.
2. Accelerations are rms acceleration over the third-octave band in m/s/s.

11. Overall analysis:

	A(rms)	CF	T-EL	T-FDP	T-RC
Seat, Vertical	1.5	3.5	146.1	36.5	3.7
Seat, Lateral	0.0	0.0	0.0	0.0	0.0
Seat, Fore-Aft	0.1	2.9	11482.7	2870.7	289.3
Total	1.5			36.1	

Notes:

1. RMS accelerations in m/s/s.
2. CF is the crest factor of the frequency weighted rms acceleration.
3. T-EL is the time to the exposure limit boundary in minutes.
4. T-FDP is the time to the fatigue-decreased proficiency boundary in minutes.
5. T-RC is the time to the reduced comfort boundary in minutes.

C. GRIFFIN'S VIBRATION DOSE VALUE:

	A(rms)	CI	CF	VDV	Time to 15 VDV
Seat, Vertical	1.8	4	3.2	6.0	1154.2
Seat, Lateral	0.0	0	0.0	0.0	0.0
Seat, Fore-Aft	0.2	0	2.4	0.7	5595751.7
Feet, Vertical	0.0	0	0.0	0.0	0.0
Feet, Lateral	0.0	0	0.0	0.0	0.0
Feet, Fore-Aft	0.0	0	0.0	0.0	0.0

Back, Vertical	0.0	0	0.0	0.0	0.0
Back, Lateral	0.0	0	0.0	0.0	0.0
Back, Fore-Aft	0.0	0	0.0	0.0	0.0
Total			6.0		1153.9

Notes:

1. A(rms) is the rms frequency weighted acceleration in m/s/s as estimated from the vibration dose value.
2. CI is a comfort index based on rms acceleration, as follows:

CI = 0	not uncomfortable
CI = 1	a little uncomfortable
CI = 2	fairly uncomfortable
CI = 3	uncomfortable
CI = 4	very uncomfortable
CI = 5	extremely uncomfortable.
3. CF is the crest factor of the frequency weighted acceleration.
4. Vibration dose value (VDV) is measured in $m/s^{1.75}$.
5. The time taken to reach 15 VDV is measured in seconds.
6. The total values are based on the root sum of quads of the individual values.

 RIDE QUALITY REPORT

Input Files...

Vehicle parameters : C:\TPASCAL\VEN\M113-A1.DLO
 Run-time parameters : C:\TPASCAL\DATA\APCDAT24.DAT
 Terrain profile data : C:\TPASCAL\TERR\BUMP.DAT
 Hub path data : C:\TPASCAL\TERR\M113BUMP.DAT

Output Files...

Time-history : D:\RUN\BMP2-DLO.DAT
 PSD : DEFAULT
 Absorbed PSD : DEFAULT
 Griffin weighted time-history : DEFAULT
 ISO weighted time-history : DEFAULT
 Ride quality report : D:\RUN\BMP2-DLO.REP

A. ABSORBED POWER:

Seat, Vertical - 0.6 watts
 Seat, Lateral - 0.0 watts
 Seat, Fore-Aft - 0.2 watts
 Feet, Vertical - 0.0 watts
 Sum of Average Absorbed Powers - 0.8 watts

B. ISO 2631:

1. Third Octave Band Analysis:

fc (Hz)	Seat Vertical	Seat Lateral	Seat Fore-Aft
1.0	0.00	0.00	0.00
1.25	0.41	0.00	0.01
1.6	0.51	0.00	0.02
2.0	0.00	0.00	0.00
2.5	0.70	0.00	0.03
3.15	1.23	0.00	0.06
4.0	1.08	0.00	0.07
5.0	1.90	0.00	0.13
6.3	2.47	0.00	0.20
8.0	3.14	0.00	0.29
10.0	4.08	0.00	0.45
12.5	4.73	0.00	0.61
16.0	5.21	0.00	0.82
20.0	4.13	0.00	0.96
25.0	1.71	0.00	0.75

31.5	1.71	0.00	0.19
40.0	4.29	0.00	0.55
50.0	4.92	0.00	0.64
63.0	2.81	0.00	0.22
80.0	1.78	0.00	0.15

Notes:

1. fc = centre frequency of third-octave band.
2. Accelerations are rms acceleration over the third-octave band in m/s/s.

11. Overall analysis:

	A(rms)	CF	T-EL	T-FDP	T-RC
Seat, Vertical	5.4	7.7	10.6	2.6	0.3
Seat, Lateral	0.0	0.0	0.0	0.0	0.0
Seat, Fore-Aft	0.4	4.8	926.6	231.6	23.3
Total	5.5			2.6	

Notes:

1. RMS accelerations in m/s/s.
2. CF is the crest factor of the frequency weighted rms acceleration.
3. T-EL is the time to the exposure limit boundary in minutes.
4. T-FDP is the time to the fatigue-decreased proficiency boundary in minutes.
5. T-RC is the time to the reduced comfort boundary in minutes.

C. GRIFFIN'S VIBRATION DOSE VALUE:

	A(rms)	CI	CF	VDV	Time to 15 VDV
Seat, Vertical	1.2	3	2.4	1.8	7388.3
Seat, Lateral	0.0	0	0.0	0.0	0.0
Seat, Fore-Aft	0.2	0	2.3	0.3	10426898.4
Feet, Vertical	0.0	0	0.0	0.0	0.0
Feet, Lateral	0.0	0	0.0	0.0	0.0
Feet, Fore-Aft	0.0	0	0.0	0.0	0.0

Back, Vertical	0.0	0	0.0	0.0	0.0
Back, Lateral	0.0	0	0.0	0.0	0.0
Back, Fore-Aft	0.0	0	0.0	0.0	0.0
Total			1.8		7383.0

Notes:

1. A(rms) is the rms frequency weighted acceleration in m/s/s as estimated from the vibration dose value.
2. CI is a comfort index based on rms acceleration, as follows:

CI = 0	not uncomfortable
CI = 1	a little uncomfortable
CI = 2	fairly uncomfortable
CI = 3	uncomfortable
CI = 4	very uncomfortable
CI = 5	extremely uncomfortable.
3. CF is the crest factor of the frequency weighted acceleration.
4. Vibration dose value (VDV) is measured in $m/s^{1.75}$.
5. The time taken to reach 15 VDV is measured in seconds.
6. The total values are based on the root sum of quads of the individual values.

END

0/9·03·92

FIN