79669

CHRYSLER / UMTRI

Wind-Steer Vehicle Simulation

Reference Manual
Version 1.4
(Volume II of II)

Report No. UMTRI-90-19-2

M. W. Sayers
C. C. MacAdam
Y. Guy

May 1990

UMTRI The University of Michigan
Transportation Research Institute

Technical Report Documentation Page

1. Report No. UMTRI-90-19-2	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle Chrysler / UMTRI Wind-Steer Vehicle Simulation — Reference Manual, Version 1.4 (Volume II)		5. Report Date May 1990 6. Performing Organization Code
7. Author(s) M. W. Sayers, C. C. MacAdam, Y. Guy		8. Performing Organization Report No. UMTRI-90-19-2
9. Performing Organization Name and Address The University of Michigan Transportation Research Institute 2901 Baxter Road, Ann Arbor, Michigan 48109		10. Work Unit No. (TRAIS) 11. Contract or Grant No. 2000533
12. Sponsoring Agency Name and Address Chrysler Motors Corporation Highland Park, Michigan		13. Type of Report and Period Covered 6/86 - 3/90 14. Sponsoring Agency Code
15. Supplementary Notes Chrysler Challenge Fund Project 2000533, "**Vehicle Crosswind Stability.**" Chrysler Corp Personnel: James H. Frye, Mark Gleason, John Pointer.		

16. Abstract

The Wind-Steer model is a time-based simulation of the handling performance of a passenger car in response to steer inputs from a driver and external wind conditions. The simulation includes the aerodynamic properties of the vehicle, a closed-loop driver model, and vehicle chassis characteristics.

This is volume 2 of 2.

17. Key Words computer model, simulation, vehicle dynamics, vehicle model, aerodynamics, passenger car, steering system, driver model, crosswind	18. Distribution Statement No restrictions.		
19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of Pages 127	22. Price

COPYRIGHT NOTICE

ACKNOWLEDGEMENTS

This document and associated work are part of a research project supported at The University of Michigan Transportation Research Institute (UMTRI) by The Chrysler Motors Corporation under the Chrysler Challenge Fund Project 2000533 entitled, "Vehicle Crosswind Stability." The Challenge Fund Program is administered at Chrysler by Mr. James H. Frye. Technical support has been generously provided by Chrysler engineering staff, in particular Messrs. Mark Gleason and John Pointer of the aerodynamics group. Additional engineering assistance has been provided by Messrs. Don VanDis and Fred Winsor on various topics related to vehicle chassis and steering dynamics.

The model and simulation code were originally developed by Mr. Yoram Guy and L. Segel of the Engineering Research Division (ERD) at UMTRI in 1986 and 1987. Subsequent additions and modifications have been performed by M. Sayers and C. MacAdam (also at UMTRI) during 1987 through 1990.

TABLE OF CONTENTS

Reference Manual (Volume II)

1. GUIDE TO THIS MANUAL

This manual is the second of two volumes documenting the Chrysler/UMTRI Wind-Steer Vehicle Simulation, Version 1.4. This manual provides detailed background material for the model that is of interest in order to understand the modeling assumptions or to modify the computer code.

The companion volume (Volume I) explains how to use the software.

The material in this volume is presented in four chapters and two appendices, which contain the following material:

Chapter 2 **Nomenclature**
This chapter defines all of the symbols used in the rest of the volume.

Chapter 3 **Equations of Motion**
This chapter describes modeling assumptions and presents the differential equations that are solved numerically in the Wind-Steer simulation program.

Chapter 4 **Programming Details**
This chapter discusses the programming details necessary for understanding and modifying the computer code.

Appendix A **Driver Model**
This appendix contains reprints of two technical papers that describe the driver steering control model contained in the program.

Appendix B **Source Code**
This appendix lists the FORTRAN 77 source code used in implementing the model on both Apple Macintosh and IBM PC / compatibles personal computers.

2. NOMENCLATURE

This chapter defines all of the symbols used in Chapter 3 and in the rest of this volume.

Subscripts

1 = Front Axle   2 = Rear Axle   L = Left   R = Right

Variables and Parameters

Symbols refer to parameters unless they are identified as being variable

A_y = [*Variable*] Lateral acceleration of vehicle center-of-mass, perpendicular to longitudinal vehicle axis and parallel to ground

a = Distance from front axle to total vehicle center of mass

a_s = Distance from front axle to center of mass of the sprung mass

b = Distance from total vehicle center of mass to rear axle

C_p = Steering boost coefficient

C_v = Steering damping coefficient

C_α = [*Variable*] Cornering stiffness for slip, defined as $\partial F_Y/\partial\alpha$

C_γ = [*Variable*] Cornering stiffness for camber, defined as $\partial F_Y/\partial\gamma$

$C_{M\alpha}$ = [*Variable*] Aligning stiffness, defined as $\partial M_Z/\partial\alpha$

D_{j1}, D_{j2} = Damping coeficient for jounce for front and rear shock absorbers

D_{r1}, D_{r2} = Damping coeficient for rebound for front and rear shock absorbers

F_D = [*Variable*] Suspension jounce / rebound damping force (additional subscripts indicate which wheel)

F_Y = [*Variable*] Tire-generated side force (additional subscripts indicate which wheel)

F_{YA} = [*Variable*] Aerodynamic side force

F_{ZA} = [*Variable*] Aerodynamic vertical force

h_1, h_2 = Height of nominal front and rear roll centers

h_{sm} = Nominal height of sprung-mass center of mass

h_{ra} = [*Variable*] Vertical distance between the sprung-mass center of mass and the instant roll axis

h_{rc1}, h_{rc2} = [*Variable*] Vertical distance between center-of-mass of the sprung mass and the instant front and rear roll centers

I_{xs} = [*Variable*] Instant moment of inertia of sprung mass about roll axis

I_{xx} = Moment of inertia of sprung mass about longitudinal (x) axis

I_{xz} = Cross product of inertia of sprung mass for x, z directions

I_{zz} = Yaw moment of inertia for entire vehicle

K_{Aux1}, K_{Aux2} = Auxiliary roll rate (beyond rate due to vertical springs), for front and rear axles, including the effects of tire compliance

K_{rr1}, K_{rr2} = Auxiliary roll rate (beyond rate due to vertical springs), for front and rear axles (without effects of tire compliance)

K_{S1}, K_{S2} = Vertical spring stiffness, for front and rear suspensions (one wheel)

K_{T1}, K_{T2} = Vertical tire stiffness, for front and rear tires (one tire)

K_{ϕ} = Total roll stiffness of suspensions and tires acting on sprung mass

L = Wheelbase (a + b)

$M_{cf, cb}$ = [*Variable*] Total steering system coulomb-friction on "forward" or "backward" path, resolved as a motion-resisting moment about the front-wheel kingpins

M_v = [*Variable*] Steering system viscous damping, resolved as a motion-resisting moment about the front-wheel kingpins

M_p = [*Variable*] Steering moment servo-boost component, resolved as a motion-assisting or motion-resisting moment about the front-wheel kingpins

M_G = [*Variable*] Front-wheel steering moment component, less viscous damping and boost

M_m = [*Variable*] Upper steering shaft "manual" moment (also controlling servo valve)

M_{XA} = [*Variable*] Aerodynamic roll moment acting on vehicle

M_{YA} = [*Variable*] Aerodynamic pitch moment acting on vehicle

M_Z = [*Variable*] Tire aligning moment (additional subscripts indicate which wheel is referenced)

M_{ZA} = [*Variable*] Aerodynamic yaw moment acting on vehicle

m_s = Sprung mass

m = Total mass

p = [*Variable*] Roll rate

Q = Aerodynamic pressure, $\rho V_A^2 / 2$

q = [*Variable*] Pitch rate

r = [*Variable*] Yaw rate

t_1, t_2 = Half-track distances for front and rear of vehicle (centerline of vehicle to centerline of tire)

V = Vehicle speed (constant)

V_A = [*Variable*] Air speed, relative to vehicle

V_{wind} = [*Variable*] Absolute wind speed

w = [*Variable*] Vertical velocity of sprung mass

X = [*Variable*] absolute (inertial) X coordinate of vehicle center-of-mass

Y = [*Variable*] absolute (inertial) Y coordinate of vehicle center-of-mass

y_{ra} = [*Variable*] Lateral distance between instant roll axis and sprung-mass center of mass

y_{rc1}, y_{rc2} = [*Variable*] Lateral distance between center-of-mass of the sprung mass and the instant front and rear roll centers

α = [*Variable*] Tire slip angle (subscripts indicate referenced tire)

α_{1o}, α_{2o} = Static tire slip angles for front and rear axles

β = [*Variable*] Vehicle slip angle

β_a = [*Variable*] Aerodynamic slip angle

δ_G = [*Variable*] Front-wheel steering angle displacement, before adjusting for lash

δ_{FW} = [*Variable*] Average front-wheel steering angle displacement

δ_{Lash} = Total steering system lash resolved to an angle about front-wheel kingpins

ε_2 = Roll steer coeficient for beam-type rear suspension

ϕ = [*Variable*] Roll of sprung mass relative to baseline trim condition

γ = [*Variable*] Tire camber angle (subscripts indicate referenced tire)

γ_{1o}, γ_{2o} = Static tire camber angles for front and rear axles

μ_1, μ_2 = Nondimensional parameters that reduce the effective suspension stiffness to account for tire vertical compliance

θ = [*Variable*] Pitch of sprung mass relative to baseline trim condition

ρ = Density of air **OR**

ρ = [*Variable*] Instantaneous path curvature of vehicle, at the center of mass

ψ = [*Variable*] Vehicle yaw (heading) angle relative to inertial frame

ψ_{wind} = [*Variable*] Absolute wind direction (180° from meteorology convention)

z = [*Variable*] Vertical displacement of vehicle sprung mass

z_{1L} = [*Variable*] Vertical displacement at left front suspension point

z_{1R} = [*Variable*] Vertical displacement at right front suspension point

z_{2L} = [*Variable*] Vertical displacement at left rear suspension point

z_{2R} = [*Variable*] Vertical displacement at right rear suspension point

I_{ss} = Steering wheel / upper column rotational inertia

K_{SC} = Steering column stiffness

K_{SL} = Steering linkage stiffness (one side)

K_{SS} = Effective (lumped) steering system stiffness based on K_{SC} and K_{SL}

GR = Overall gear ratio of steering system

δ_{sw} = [*Variable*] Steering wheel rotational displacement

δ_{fw}' = [*Variable*] δ_{sw} / GR

C_L = Aerodynamic lift coefficient

C_D = Aerodynamic drag coefficient

C_M = Aerodynamic pitch moment coefficient

K_L = Aerodynamic coefficient for lift force variation due to β_a^2

K_D = Aerodynamic coefficient for drag force variation due to β_a^2

K_Y = Aerodynamic side force coefficient

K_N = Aerodynamic yaw moment coefficient

K_R = Aerodynamic roll coefficient

K_M = Aerodynamic coefficient for pitch moment variation due to β_a^2

A = Aerodynamic cross sectional area

3. EQUATIONS OF MOTION

This chapter describes modeling assumptions and presents the differential equations that are solved numerically in the Wind-Steer simulation program.

3.1 Degrees of Freedom

The vehicle model is intended to accurately predict handling behavior for moderate levels of cornering (less than 0.3 g's) and frequencies less than 5 Hz. The vibrational characteristics of the unsprung masses are not included in the model because they involve frequencies (typically 10 — 15 Hz) well above the bandwidth of interest for steering response. Forces and moments acting on the body through the suspensions depend on the suspension deflections. The model predicts suspension deflections by using a quasi-static force balance between the tire vertical force and the vertical suspension force. That is, vertical tire forces predicted by the models are determined solely by the motions of the sprung mass.

The vehicle is modeled as a rigid body that is free to pitch, roll, and yaw, and to translate laterally and vertically. The forward speed is constant. Movements of the four wheels are treated quasi-statically by considering the compliance of the suspensions and tires. Altogether, the vehicle has five dynamic degrees of freedom and is essentially an extension of Segel's original three-degree-of-freedom model which included yaw, roll, and lateral translation.[1] A dynamic steering system option adds a sixth degree of freedom.

Even though the unsprung masses do not have dynamical degrees of freedom, the equations of motion nonetheless account for the distribution of mass and rotational inertia between the sprung and unsprung masses. In bounce, pitch, and roll, the sprung and unsprung masses are coupled through the compliant suspension elements. In yaw and lateral translation, they are coupled through suspension elements that are essentially rigid. Thus, the force and moment balances for yaw and lateral acceleration involve the mass and rotational inertia of the *entire* vehicle. The force and moment balances for bounce, pitch, and roll involve the mass and rotational inertia of *only* the sprung mass.

The dynamics of the steering system are also included as an option. When selected, the input to the steering system is a *torque* acting on the steering wheel. The steering wheel torque may be specified as an open-loop time-history input, or, it can instead be calculated by the resident closed-loop driver model during path-following maneuvers.

The constant-speed vehicle model includes a total of six dynamic degrees of freedom. Table 3.1.1. defines the twelve state variables that describe the kinematics of the vehicle. They follow the SAE recommended practice sign convention.[2]

· Independent steer, camber, and vertical motions are included for each wheel. These high-frequency motions are treated as being in static equilibrium at the low frequencies of interest.

3.2 Body Equations

Kinematical Relationships

The derivatives of the inertial X and Y coordinates of the vehicle center of mass are related to the constant forward speed and vehicle rotation:

$$\dot{X} = V \cos(\psi + \beta) \qquad (3.2.1)$$

[1] Segel, L. "Theoretical Prediction and Experimental Substantiation of the Response of the Automobile to Steering Control." *Proceedings* of the Automobile Division of the Institution of Mechanical Engineers, No. 7, 1956-57.

[2] Vehicle Dynamics Terminology, Handbook Supplement, SAE Recommended Practice, SAE J670e, June 1978.

$$\dot{Y} = V \sin(\psi + \beta) \tag{3.2.2}$$

Table 3.1.1. State variables.

Symbol	Description
X	inertial forward coordinate of vehicle center of mass
Y	inertial lateral coordinateof vehicle center of mass
z	inertial vertical coordinate of vehicle sprung mass
ϕ	Euler roll angle of sprung mass
θ	Euler pitch angle of sprung mass
ψ	Euler yaw angle of total vehicle
p	Roll angle rate of sprung mass (in body-axis coordinate system)
q	Pitch angle rate of sprung mass (in body-axis coordinate system)
r	Yaw angle rate of sprung mass (in body-axis coordinate system)
β	Side slip angle of vehicle mass center
w	Vertical displacement rate of sprung mass (in body axis coordinate system)
δ_{FW}	Average steer angle of front wheels

The following four state variables are speeds defined as derivatives of other state variables:

$$w = \dot{z} \tag{3.2.3}$$

$$p = \dot{\phi} \tag{3.2.4}$$

$$q = \dot{\theta} \tag{3.2.5}$$

$$r = \dot{\psi} \tag{3.2.6}$$

Two useful variables that are derived from the yaw rotation rates are the lateral acceleration and the path curvature of the vehicle center of mass:

$$A_y = \frac{V\,(r + \dot{\beta})}{g} \tag{3.2.7}$$

$$\rho = \frac{r + \dot{\beta}}{V} \tag{3.2.8}$$

Force / Moment Equilibrium Equations

The following sums combine the external forces and moments applied the tires and the aerodynamic effect:

$$\sum F_Y = F_{Y_{1L}} + F_{Y_{1R}} + F_{Y_{2L}} + F_{Y_{2R}} + F_{YA} \tag{3.2.9}$$

$$\sum F_Z = F_{Z_{1L}} + F_{Z_{1R}} + F_{Z_{2L}} + F_{Z_{2R}} - F_{ZA} \tag{3.2.10}$$

$$\sum M_Z = M_{Z_{1L}} + M_{Z_{1R}} + M_{Z_{2L}} + M_{Z_{2R}} + M_{ZA} + a\,(F_{Y_{1L}} + F_{Y_{1R}}) - b\,(F_{Y_{2L}} + F_{Y_{2R}}) + \left(a - \frac{L}{2}\right) F_{YA} \tag{3.2.11}$$

Five equilibrium equations can be written for this vehicle model by balancing the applied forces and moments with D'Alembert's forces and torques. The summation of lateral force and yaw moment are applied about the entire vehicle, whereas the pitch and roll moments and the vertical force are applied only for the sprung mass. As implied by the form of the following equations, these relations are used to evaluate the acceleration variables.

$$I_{XS}\,\dot{p} = -I_{XZ}\,\dot{r} - m_s\,h_{ra} V\,(r + \dot{\beta}) + m_s\,g\,y_{ra} - K_\phi\,\phi + t_1(F_{D_{1L}} - F_{D_{1R}}) + t_2(F_{D_{2L}} - F_{D_{2R}}) + M_{XA} - h_{sm}\,F_{YA} \tag{3.2.12}$$

$$\dot{\beta} = \frac{-m_s\,h_{ra}\,\dot{p} + \sum F_Y}{m\,V} - r \tag{3.2.13}$$

$$\dot{r} = \frac{-I_{XZ}\,\dot{p} + \sum M_Z}{I_{ZZ}} \tag{3.2.14}$$

$$\dot{w} = \frac{m\,g - \sum F_Z}{m_s} \tag{3.2.15}$$

$$\dot{q} = \frac{a\,[2\,\mu_1\,K_{S1}\,(z - a\,\theta) + F_{D_{1L}} + F_{D_{1R}}] - b\,[2\,\mu_2\,K_{S2}\,(z + b\,\theta) + F_{D_{2L}} + F_{D_{2R}}] + M_{YA} + \left(\frac{L}{2} - a_s\right) F_{ZA}}{I_{YS}} \tag{3.2.16}$$

As written above, the first three of these equations are coupled in such a way that they cannot be evaluated sequentially in a computer program. That is, the terms $\dot{p}$, $\dot{\beta}$, and $\dot{r}$ appear on both sides of eqs. 3.2.12 through 3.2.14. By substituting eqs. 3.2.13 and 3.2.14 into 3.2.12, an alternative expression for $\dot{p}$ is obtained which is not dependent on $\dot{\beta}$ or $\dot{r}$:

$$\dot{p} = \frac{m_s\, g\, y_{ra} - \frac{I_{XZ}}{I_{ZZ}} \sum M_Z - K_\phi\, \phi + M_{XA} - h_{sm}\, F_{YA} - \frac{m_s h_{ra}}{m} \sum F_Y + t_1 \left(F_{D_{1L}} - F_{D_{1R}}\right) + t_2 \left(F_{D_{2L}} - F_{D_{2R}}\right)}{I_{XS} - \frac{m_s^2 h_{ra}^2}{m} - \frac{I_{XZ}^2}{I_{ZZ}}} \qquad (3.2.17)$$

This expression is used (rather than eq. 3.2.12) to evaluate $\dot{p}$. The known value of $\dot{p}$ is then used in eqs. 3.2.13 and 3.2.14 to evaluate $\dot{\beta}$ and $\dot{r}$.

3.3 Aerodynamic Forces and Moments

The aerodynamic slip angle (β_a) and speed (V_A) are defined as follows:

$$V_{ax} = V_{wind} \cos\left(\psi_{wind} - \psi\right) - V \cos(\beta) \qquad (3.3.1)$$

$$V_{ay} = V_{wind} \sin\left(\psi_{wind} - \psi\right) - V \sin(\beta) \qquad (3.3.2)$$

$$V_a = \sqrt{V_{ax}^2 + V_{ay}^2} \qquad (3.3.3)$$

$$\beta_a = \tan^{-1}\left(\frac{V_{ax}}{V_{ay}}\right) - \pi; \; V_{ay} > 0 \qquad (3.3.4)$$

$$\beta_a = \tan^{-1}\left(\frac{V_{ax}}{V_{ay}}\right) + \pi; \; V_{ay} < 0 \qquad (3.3.5)$$

The areodynamic forces and moments are the following:

$$Q = \frac{\rho\, V_A^2}{2} \qquad (3.3.6)$$

$$F_{XA} = Q\, A\, (C_{D0} + K_D\, \beta_a^2) \qquad (3.3.7)$$

$$F_{YA} = -Q\, A\, K_Y\, \beta_a \qquad (3.3.8)$$

$$F_{ZA} = -Q\, A\, (C_{L0} + K_L\, \beta_a^2) \qquad (3.3.9)$$

$$M_{XA} = -Q\, A\, L\, K_R\, \beta_a \qquad (3.3.10)$$

$$M_{YA} = -Q\, A\, L\, (C_{M0} + K_M\, \beta_a^2) \qquad (3.3.11)$$

$$M_{ZA} = -Q\, A\, L\, K_N\, \beta_a \qquad (3.3.12)$$

3.4 Suspension / Wheel Terms

Vertical Displacements

The tire slip and camber angles are influenced by the following suspension deflections

$$z_{1L} = z - a\theta - t_1 \phi \tag{3.4.1}$$

$$z_{1R} = z - a\theta + t_1 \phi \tag{3.4.2}$$

$$z_{2L} = z + b\theta - t_2 \phi \tag{3.4.3}$$

$$z_{2R} = z + b\theta + t_2 \phi \tag{3.4.4}$$

The above expressions neglect vertical tire deflection. The effects of tire compliance are treated below by defining "effective stiffness" and "effective damping" values.

Effective Stiffness and Damping Values

All suspension springs in the vehicle model are linear. These include the vertical spring rates at each wheel, the auxiliary roll stiffness for the front and rear axles, and the tire vertical spring rates. The vertical motions of the wheels (acting against the tire vertical stiffness) is not computed in this model. Instead, the tire compliance values are used to lower the spring and damping rates of the suspension so that the vertical force, roll moment, and pitch moment acting on the sprung mass take into account the tire vertical deflections.

Effects of vertical spring and damper coefficients are reduced by the proportion of the overall vertical wheel movement that is due to the tire compliance

$$\mu_1 = \frac{K_{T1}}{K_{T1} + K_{S1}} \tag{3.4.5}$$

$$\mu_2 = \frac{K_{T2}}{K_{T2} + K_{S2}} \tag{3.4.6}$$

The effective auxiliary roll stiffnesses for the front and rear axles are also reduced due to tire compliance

$$K_{Aux1} = \frac{2\, t_1^2\, K_{T1}\left(2\, t_1^2\, K_{S1} + K_{rr1}\right)}{2\, t_1^2\left(K_{T1} + K_{S1}\right) + K_{rr1}} - 2\, \mu_1 t_1^2\, K_{S1} \tag{3.4.7}$$

$$K_{Aux2} = \frac{2\, t_2^2\, K_{T2}\left(2\, t_2^2\, K_{S2} + K_{rr2}\right)}{2\, t_2^2\left(K_{T2} + K_{S2}\right) + K_{rr2}} - 2\, \mu_2\, t_2^2\, K_{S2} \tag{3.4.8}$$

A single stiffness applies to the roll motions of the sprung mass.

$$K_f = 2\, m_1\, K_{S1}\, t_1^2 + 2\, m_2\, K_{S2}\, t_2^2 + K_{Aux1} + K_{Aux2} \tag{3.4.9}$$

Vertical Damping Forces

A bi-directional shock absorber model is used. A linear damping coefficient is used with different values for jounce and rebound, as indicated by the subscripts j/r. The nondimensional coefficients μ_1 and μ_2 (see eqs. 3.4.5 and 3.4.6) are used to reduce the suspension motion by the amount of the tire deflection.

$$F_{D_{1L}} = \mu_1 D_{(j/r)1} [w - a\theta - t_1 p] \tag{3.4.10}$$

$$F_{D_{1R}} = \mu_1 D_{(j/r)1} [w - a\theta + t_1 p] \tag{3.4.11}$$

$$F_{D_{2L}} = \mu_2 D_{(j/r)2} [w + b\theta - t_2 p] \tag{3.4.12}$$

$$F_{D_{2R}} = \mu_2 D_{(j/r)2} [w + b\theta + t_2 p] \tag{3.4.13}$$

Vertical Ground Loads

The tire forces and moments are influenced by vertical load. The vertical loads for each tire are defined as follows:

$$F_{Z_{1L}} = \frac{1}{2}\left[\frac{b\,m\,g}{L} - \frac{K_{Aux1}\,\phi - h_{rc1}(F_{Y_{1L}} + F_{Y_{1R}})}{t_1}\right] + F_{D_{1L}} + \mu_1 z_{1L} K_{S1} \tag{3.4.14}$$

$$F_{Z_{1R}} = \frac{1}{2}\left[\frac{b\,m\,g}{L} + \frac{K_{Aux1}\,\phi - h_{rc1}(F_{Y_{1L}} + F_{Y_{1R}})}{t_1}\right] + F_{D_{1R}} + \mu_1 z_{1R} K_{S1} \tag{3.4.15}$$

$$F_{Z_{2L}} = \frac{1}{2}\left[\frac{a\,m\,g}{L} - \frac{K_{Aux2}\,\phi - h_{rc2}(F_{Y_{2L}} + F_{Y_{2R}})}{t_2}\right] + F_{D_{2L}} + \mu_2 z_{2L} K_{S2} \tag{3.4.16}$$

$$F_{Z_{2R}} = \frac{1}{2}\left[\frac{a\,m\,g}{L} + \frac{K_{Aux2}\,\phi - h_{rc2}(F_{Y_{2L}} + F_{Y_{2R}})}{t_2}\right] + F_{D_{2R}} + \mu_2 z_{2R} K_{S2} \tag{3.4.17}$$

3.5 Roll Axis

The suspension kinematics are simplified by assuming that the sprung mass rotates about a roll axis. To extend this representation, the axis is permitted to move as a function of roll angle. The roll axis is located by two points, each in the vertical plane containing each axle. These points are defined by static heights located on the longitudinal centerline of the vehicle, h_1 and h_2. Movements of these two points are introduced as vertical and lateral components, h_{rc} and y_{rc}, which are defined as quadratic functions of roll angle in coordinates fixed in the (rolling) sprung mass (see Section 2.1 in Volume I). The (rolled) vertical and lateral distances between the center of the sprung mass and the roll axis are defined as

$$h' = h_{rc1} + \frac{a_s}{L}(h_{rc2} - h_{rc1}) \tag{3.5.1}$$

$$y' = y_{rc1} + \frac{a_s}{L}(y_{rc2} - y_{rc1}) \tag{3.5.2}$$

These dimensions are projected into a non-rolling frame to yield the offsets

$$h_{ra} = h' - y' \phi \tag{3.5.3}$$

$$y_{ra} = y' + h' \phi \tag{3.5.4}$$

An instant roll moment of inertia is defined for the sprung mass to include the effect of the offset of the center of mass relative to the roll axis.

$$I_{xs} = I_{xx} + (y_{ra}^2 + h_{ra}^2)\, m_s \tag{3.5.5}$$

3.6 Tire Slip / Camber / Steer Equations

The tire side force and aligning moment are modeled as being linear with respect to slip and inclination angles. However, as was explained in Section 2.3 of Volume I, the coefficients can be functions of vertical load. The static offset in steer and inclination angle are assumed to have equal magnitudes but opposite signs for the two suspensions on an axle. The same is true for kinematics effects. That is, the change in steer and inclination due to suspension deflection on the two sides of the vehicle are modeled as having opposite signs. Steer and inclination are also influenced by forces and moments applied by the tires, due to suspension compliance. The compliance effects are the same on either side of the vehicle.

The right-hand side of the vehicle should be used as the reference in applying conventional SAE kinematics and compliance coefficients.

Suspension Kinematics (Independent Suspensions)

The changes in steer and inclination due to suspension kinematics for axle i are described as follows for independent suspensions:

$$\delta_{K_{iL}} = C_{\delta z1_i}\, z_{iL} + C_{\delta z1_i}\, z_{iL}^2 \tag{3.6.1}$$

$$\delta_{K_{iR}} = -C_{\delta z1_i}\, z_{iR} - C_{\delta z1_i}\, z_{iR}^2 \tag{3.6.2}$$

$$\gamma_{K_{iL}} = C_{\gamma z1_i}\, z_{iL} + C_{\gamma z2_i}\, z_{iL}^2 \tag{3.6.3}$$

$$\gamma_{K_{iR}} = -C_{\gamma z1_i}\, z_{iR} - C_{\gamma z2_i}\, z_{iR}^2 \tag{3.6.4}$$

Suspension Kinematics (Beam Rear Axle)

For a beam rear axle, the inclination angle remains constant. The influence of suspension kinematics on steer are accomodated by a linear roll-steer coefficient:

$$\delta_{K_{2R}} = \delta_{K_{2L}} = \varepsilon_R\, \phi \tag{3.6.5}$$

Left-Front Suspension

The slip angle is defined as follows for the left-front suspension:

$$\alpha_{1L} = \alpha_{1o} + \beta + \frac{r\,a}{V} - \delta_{FW} + \delta_{K_{1L}} - K_{\delta CF_1} F_{Y_{1L}} - K_{\delta CM_1} M_{Z_{iL}} \tag{3.6.6}$$

where α_{1L} is the slip angle, α_{1o} is the static toe-out, δ_{FW} is the steer angle predicted by the steering model, $\delta_{K_{1L}}$ is the steer due to bump-steer, $K_{\delta CF_1}$ is the lateral-force compliance-steer coefficient for the front axle, and $K_{\delta CM_1}$ is the aligning-torque compliance coefficient.

If the dynamic steering system is being used, separate right- and left-hand front-wheel steer angles are computed within the steering system model. The aligning torque compliances are also accounted for in the steering system model. If the steering model is not being used, δ_{FW} is equal to $\delta_{SW}/_{GR}$ and the aligning torque compliances are included as shown.

The inclination angle for the left-front suspension is:

$$\gamma_{1L} = -\gamma_{1o} + \phi + \gamma_{K_{1L}} - K_{\gamma CF_1} F_{Y_{1L}} \tag{3.6.7}$$

where γ_{1L} is the inclination angle, γ_{1o} is the static camber, $\gamma_{K_{1L}}$ is the portion of the inclination angle due to suspension deflection, and $K_{\gamma CF_1}$ is a lateral-force compliance-inclination coefficient for the front axle.

Because the slip and inclination angles are influenced by tire side force and aligning moment, which are in turn developed by slip and inclination , the above equations are not suitable for sequential evaluation. For the specific tire model described here, it is in fact possible to obtain closed-form solutions for slip and inclination , because the explicit force and moment equations are linear with respect to slip and inclination. The simulataneous linear equations can be solved analytically.

A problem with this approach is that the formulation is tied to the specifics of the tire model. Changing the tire model (for example, to allow slip angles into the nonlinear regime) becomes difficult. Therefore, a more generalized formulation is used which is independent of the tire model characteristics. The simultaneous equations are solved with a Newton-Raphson iteration method, which is valid for both linear and nonlinear equations. The Newton-Raphson method is well established, and an existing computer alogorithm was incorporated into the Wind-Steer simulation.[1]

The Newton-Raphson method requires that a set of two simultaneous equations be put into the form of error functions:

$$f_1\ (x_1, x_2, \dots x_n) = 0$$

---

[1] Press, W.H. et al., *Numerical Recipes: the Art of Scientific Computing*. Cambridge University Press, 1986.

$$f_2(x_1, x_2, \ldots x_n) = 0 \tag{3.6.8}$$

where x_1 and x_2 are independent variables and f_1 and f_2 are nonlinear error functions of those variables that are identically zero when the equations are satisfied. In matrix form, this is

$$\underline{f}(\underline{x}) = 0 \tag{3.6.9}$$

where $\underline{f}$ is defined as $[f_1, f_2]^T$ and $\underline{x}$ is defined as $[x_1, x_2]^T$. To find the values for x_1 and x_2 which satisfy eq. 3.6.9, the Newton-Raphson algorithm requires: (1) a subroutine to compute $\underline{f}$, given $\underline{x}$, and (2) a subroutine to compute $\underline{J}$, given $\underline{x}$, where $\underline{J}$ is the 2x2 Jacobian, whose elements are defined as

$$J_{ij} = -\frac{\partial f_i}{\partial x_j} \tag{3.6.10}$$

The above equations for slip and camber are easily put into the form required for the Newton-Raphson computation:

$$\underline{x} = [\alpha_{1L}, \gamma_{1L}]^T \tag{3.6.11}$$

Error functions are obtained by moving all terms to the right-hand side of eqs 3.6.6 and 3.6.7:

$$0 = f_1 = -\alpha_{1L} + \alpha_{1o} + \beta + \frac{r\,a}{V} - \delta_{FW} + \delta_{K_{1L}} - K_{\delta CF_1} F_{Y_{1L}} - K_{\delta CM_1} M_{Z_{iL}}$$

$$0 = f_2 = -\gamma_{1L} - \gamma_{1o} + \phi + \gamma_{K_{1L}} - K_{\gamma CF_1} F_{Y_{1L}} \tag{3.6.12}$$

The Jacobian is obtained by applying eq. 3.6.10 to eq. 3.6.12:

$$J_{11} = 1 + K_{\delta CF_1}\frac{\partial F_Y}{\partial \alpha} + K_{\delta CM_1}\frac{\partial M_Z}{\partial \alpha} \qquad J_{12} = K_{\delta CF_1}\frac{\partial F_Y}{\partial \gamma}$$

$$J_{21} = K_{\gamma CF_1}\frac{\partial F_Y}{\partial \alpha} \qquad J_{22} = 1 + K_{\gamma CF_1}\frac{\partial F_Y}{\partial \gamma} \tag{3.6.13}$$

Right-Front Suspension

A similar formulation is used for the right-front suspension:

$$\alpha_{1R} = -\alpha_{1o} + \beta + \frac{r\,a}{V} - \delta_{FW} + \delta_{K_{1R}} - K_{\delta CF_1} F_{Y_{1R}} - K_{\delta CM_1} M_{Z_{iR}} \tag{3.6.14}$$

$$\gamma_{1R} = \gamma_{1o} + \phi + \gamma_{K_{1R}} - K_{\gamma CF_1} F_{Y_{1R}} \tag{3.6.15}$$

The Newton-Raphson formulation for the right-front suspension is nearly identical to the one used for the left. In this case, the variables are

$$\underline{x} = [\alpha_{1R}, \gamma_{1R}]^T \tag{3.6.16}$$

the error functions are:

$$0 = f_1 = -\alpha_{1R} - \alpha_{1o} + \beta + \frac{r\,a}{V} - \delta_{FW} + \delta_{K_{1R}} - K_{\delta CF_1} F_{Y_{1R}} - K_{\delta CM_1} M_{Z_{iR}}$$

$$0 = f_2 = -\gamma_{1R} + \gamma_{1o} + \phi + \gamma_{K_{1R}} - K_{\gamma CF_1} F_{Y_{1R}} \qquad (3.6.17)$$

The Jacobian matrix is identical to that used for the left-front suspension (eq. 3.6.13).

Left-Rear Independent Suspension

The equations for an independent left-rear suspension are:

$$\alpha_{2L} = \alpha_{2o} + \beta - \frac{r\,b}{V} + \delta_{K_{2L}} - K_{\delta CF_2} F_{Y_{2L}} - K_{\delta CM_2} M_{Z_{2L}} \qquad (3.6.18)$$

$$\gamma_{2L} = -\gamma_{2o} + \phi + \gamma_{K_{2L}} - K_{\gamma CF_2} F_{Y_{2L}} \qquad (3.6.19)$$

The independent variables for the Newton-Raphson computation are:

$$\underline{x} = [\alpha_{2L}, \gamma_{2L}]^T \qquad (3.6.20)$$

Error functions are developed from eqs 3.6.18 and 3.6.19:

$$0 = f_1 = -\alpha_{2L} + \alpha_{2o} + \beta - \frac{r\,b}{V} + \delta_{K_{2L}} - K_{\delta CF_2} F_{Y_{2L}} - K_{\delta CM_2} M_{Z_{2L}}$$

$$0 = f_2 = -\gamma_{2L} - \gamma_{2o} + \phi + \gamma_{K_{2L}} - K_{\gamma CF_2} F_{Y_{2L}} \qquad (3.6.21)$$

The Jacobian is obtained by applying eq. 3.6.10 to eq. 3.6.21:

$$J_{11} = 1 + K_{\delta CF_2} \frac{\partial F_Y}{\partial \alpha} + K_{\delta CM_2} \frac{\partial M_Z}{\partial \alpha} \qquad J_{12} = K_{\delta CF_2} \frac{\partial F_Y}{\partial \gamma}$$

$$J_{21} = K_{\gamma CF_2} \frac{\partial F_Y}{\partial \alpha} \qquad J_{22} = 1 + K_{\gamma CF_2} \frac{\partial F_Y}{\partial \gamma} \qquad (3.6.22)$$

Right-Rear Independent Suspension

The equations for an independent right-rear suspension are:

$$\alpha_{2R} = -\alpha_{2o} + \beta - \frac{r\,b}{V} + \delta_{K_{2R}} - K_{\delta CF_2} F_{Y_{2R}} - K_{\delta CM_2} M_{Z_{2R}} \qquad (3.6.23)$$

$$\gamma_{2R} = \gamma_{2o} + \phi + \gamma_{K_{2R}} - K_{\gamma CF_2} F_{Y_{2R}} \qquad (3.6.24)$$

The independent variables for the Newton-Raphson computation are:

$$\underline{x} = [\alpha_{2R}, \gamma_{2R}]^T \qquad (3.6.25)$$

and the error functions are:

$$0 = f_1 = -\alpha_{2R} - \alpha_{2o} + \beta - \frac{r\,b}{V} + \delta_{K_{2R}} - K_{\delta CF_2} F_{Y_{2R}} - K_{\delta CM_2} M_{Z_{2R}}$$

$$0 = f_2 = -\gamma_{2R} + \gamma_{2o} + \phi + \gamma_{K2R} - K_{\gamma CF_2} F_{Y2R} \quad (3.6.26)$$

The Jacobian defined in eq. 3.6.22 also applies to the right-rear suspension.

Beam Rear Axle

For a beam rear axle, linkage compliance can permit the axle to steer in response to applied side force and aligning moment. The attachment of the wheels to the axle is assumed to be rigid, and the axle is assumed to have negligible roll compliance. These assumptions lead to the following expressions for the slip and camber angles.

$$\alpha_{2L} = \alpha_{2o} + \beta - \frac{r\,b}{V} + \delta_{K_{2L}} - K_{\delta CF_2}(F_{Y_{2L}} + F_{Y_{2R}}) - K_{\delta CM_2}(M_{Z_{2L}} + M_{Z_{2R}}) \quad (3.6.27)$$

$$\alpha_{2R} = \alpha_{2L} - 2\,\alpha_{2o} \quad (3.6.28)$$

$$\gamma_{2L} = -\gamma_{2o} \qquad \gamma_{2R} = \gamma_{2o} \quad (3.6.29)$$

In this case, only one independent variable is needed for the Newton-Raphson computation:

$$\underline{x} = [\alpha_{2L}]^T \quad (3.6.30)$$

the error function is:

$$0 = f_1 = -\alpha_{2L} + \alpha_{2o} + \beta - \frac{r\,b}{V} + \delta_{K_{2L}} - K_{\delta CF_2}(F_{Y_{2L}} + F_{Y_{2R}}) - K_{\delta CM_2}(M_{Z_{2L}} + M_{Z_{2R}}) \quad (3.6.31)$$

and the Jacobian is:

$$J_{11} = 1 + 2\,K_{\delta CF_2}\frac{\partial F_Y}{\partial \alpha} + 2\,K_{\delta CM_2}\frac{\partial M_Z}{\partial \alpha} \quad (3.6.32)$$

3.7 Power-Assisted Steering System

The steering system model is similar to work published by Segel and MacAdam[1], except that the high-frequency degree of freedom associated with wobble of the front wheels is left out because the high frequencies are outside the bandwidth of interest.

---

[1] Segel, L. "On the Lateral Stability and Control of the Automobile as Influenced by the Dynamics of the Steering System." *Journal of Engineering for Industry (Transactions of ASME)*, Vol. 88, Series B, No. 3, August 1966.

Segel, L. and MacAdam, C. C. "The Influence of the Steering System on the Directional Response to Steering," *Proceedings* of the 10th IAVSD Symposium of the Dynamics of Vehicles on Roads and Tracks, Prague Cz, 1987.

The following equations for the dynamic steering system model are based on the diagram of Figure 3.7.1. The dynamics for the upper portion of the steering system are given by:

$$I_{ss}\, d^2 (\delta_{sw}) / dt^2 = M + K_{ss} (\delta_{fw} - \delta_{fw}') / GR$$

$$- C_{ss}\, d (\delta_{sw}) / dt - C_F \,\mathrm{sign}[d (\delta_{sw}) / dt] \tag{3.7.1}$$

where,

$$\delta_{fw}' = \delta_{sw} / GR \tag{3.7.2}$$

and C_{ss} , C_F are parameters representing viscous and coulomb friction.

The "no-lash" front wheel angle, δ_{fw}, is determined from the quasi-static relationship accross the lumped compliance K_{ss} and current value of δ_{fw}' as:

$$\delta_{fw} = \delta_{fw}' + H (1 - C_B) / K_{ss} \tag{3.7.3}$$

The lumped compliance, K_{ss}, is given by the serial combination of the upper column compliance K_{sc} and the two lower linkage compliances K_{sL} as:

$$2 K_{sc} K_{sL} GR^2 / (GR^2 K_{sc} + 2 K_{sL})$$

C_B is the power boost (percent / 100) contribution from the pump and, H, the tire aligning torques of both front tires, is given by:

$$H = 2 C_\alpha (x_p+x_m)[(v+ar)/U - \delta_{fw}] / K_{ss} \tag{3.7.4}$$

x_p and x_m are the pneumatic and mechanical trails, respectively, of the front tires/wheels. C_α is the front tire cornering stiffness.

Substituting 3.7.4 into 3.7.3 and solving for δ_{fw} yields:

$$\delta_{fw} = [\delta_{fw}' + 2 C_\alpha (x_p+x_m) (1 - C_B) (v+ar) / (U K_{ss})] /$$

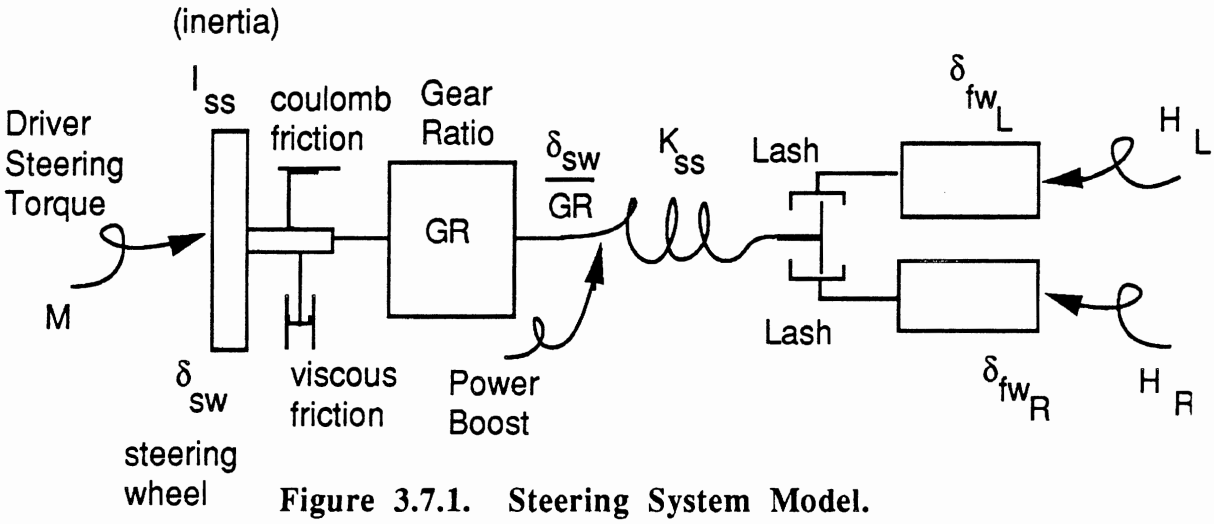

Figure 3.7.1. Steering System Model.

$$[\, 1 + 2\, C_\alpha\, (x_p + x_m)\, (1 - C_B) / K_{ss}] \qquad (3.7.5)$$

Substituting 3.7.5 into the differential equation 3.7.1 results in:

$$I_{ss}\, d^2(\delta_{sw}) / dt^2 = M + K_{ss}[\, \mathbf{A}\, \delta_{sw} - \mathbf{B}\, (v+ar)\,] / GR^2$$

$$- C_{ss}\, d\, (\delta_{sw}) / dt - C_F\, \mathrm{sign}[d\, (\delta_{sw}) / dt] \qquad (3.7.6)$$

where,

$$\mathbf{A} = 1 - 1 / [\, 1 + 2\, C_\alpha\, (x_p + x_m)\, (1 - C_B) / K_{ss}\,]$$

and,

$$\mathbf{B} = 2\, C_\alpha\, (x_p + x_m)\, (1 - C_B)\, GR \; /$$

$$\{\, [\, 1 + 2\, C_\alpha\, (x_p + x_m)\, (1 - C_B) / K_{ss}]\, U\, K_{ss}\, \}$$

The left and right front wheel angles, δ_{fwL} and δ_{fwR}, are obtained from equation (3.7.5) using left/right parameter values of tire cornering stiffness and inclusion of the wheel lash.

4. PROGRAMMING DETAILS

This chapter describes how the Wind-Steer program operates. It is intended for programmers who wish to modify the program, or port it to a new computer.

4.1 Machine Dependencies

The Wind-Steer program is written completely in Fortran 77. The standard does not recognize any hardware-specific aspects of a computer, such as the screen, keyboard, or clock. To make the program a more productive tool, it does make use of a few machine-specific features for the versions that run on the IBM PC, the Apple Macintosh, and MTS (The University of Michigan mainframe computer). These are:

- The output file contains the time and date for the simulation, which is provided by a subroutine called TIMDAT. The subroutine TIMDAT should be modified to work on the computer for which the program will be used. If time and date information is not available, the subroutine can be made inoperative.

  The Macintosh version uses external subroutines provided with the compiler, TIME and DATE. These must be linked with the rest of the program if it is re-compiled for the Macintosh.

- The Fortran i/o unit number for the "terminal" (i.e., the keyboard and screen) should be set to the proper value expected by the compiler. Most compilers, including all three that have been used to date, permit an asterisk * to be used to specify the screen and keyboard.

- Simulation progress is shown on the screen in the PC and Mac versions. This involves interacting with the screen. This is done in the subroutine OUTPUT and should be modified to work on the new computer, or deleted.

  The IBM version uses the subroutine SETCUR from an UMTRI library of Fortran extensions. This library must be linked with the rest of the program for use on the IBM PC.

- Writing of binary data has been done differently for every system so far. The MTS version uses an MTS subroutine, WRITE, to put binary data into an ordinary file. The PC version opens a separate file with access type set to a nonstandard type BINARY. The Mac version uses a separate file with access set to UNFORMATTED. Both the Mac and the PC versions of the program produce binary files with no structure—just a stream of binary data.

- The source code is contained in a large file with the main program and all of the subroutine modules, and in nine small "include files" which are merged with the main file during compilation. The INCLUDE command is not standard Fortran, and is handled differently by each compiler.

4.2 Structure of Program

The operation of this program follows that of many programs that use numerical integration to simulate a dynamic system, and can be summarized by the following steps:

1. Read input data. This function is performed by the subroutine INDATA.
2. Initialize variables and constants derived from input data. This function is performed by the subroutine INIT.
2. Establish name(s) of output file(s) and write header data (number of channels, names, etc.) This function is performed by the subroutine OPNOUT.
3. Perform the numerical integration using a "loop," in which the differential equations are solved numerically for time T, and T is increased in small increments DT. The differential equations are written in the form:

   $$\dot{Y}_i = dY_i/dt = f(Y_1, Y_2, \dots Y_n, t) \qquad (4.2.1)$$

   where Y_i is a state variable, $i = 1, 2, \dots n$, and n = number of equations.

   The function indicated above as f is named FUNCTN in the Fortran Wind-Steer program.

   The integration from time T to T+DT is performed using a modified Euler method, sometimes called a second-order Runge-Kutta. Specifically, the integration of each state variable is accomplished as follows:

   $$Y'_i = Y_i(T) + DT/2 \cdot f(Y_1, Y_2, \dots Y_n, T) \qquad (4.2.2)$$

   $$Y_i(T+DT) = Y_i(T) + DT \cdot f(Y'_1, Y'_2, \dots Y'_n, T+DT/2) \qquad (4.2.3)$$

Note that *f* (FUNCTN) is evaluated twice for each integration step: once at the start, and a second time at the midpoint of the time interval. All of the equations that represent the vehicle are contained in FUNCTN and in several auxiliary subprograms that are used by FUNCTN. (These additional routines are named AIRACT, FDAMP, ROLLAX, STEER, TIRES2, SUM, etc.)

At some multiple of DT, values of interest are written into the output file by the subroutine OUTPUT.

4. Print the success or failure of the simulation and close any open files.

4.3 Program Modules

This section describes the modules that make up the Wind-Steer program. The subprograms are shown below in alphabetical order with a listing of their arguments and common block references.

AIRACT(YAW, BETA, VYAW)

Update air velocity, sideslip, and magnitudes of forces and moments in common block /AERO/.

→ YAW	real*4	Yaw angle of vehicle.
→ BETA	real*4	Sideslip angle of vehicle.
→ VYAW	real*4	Yaw rate of vehicle.

Common Blocks: GLBL PARS AERO

BEAM(ALPH, DFDX, FNEG)

Compute 2x2 Jacobian and a 2-element error array for beam rear suspension.

→ ALPH	real*4	2-element array. 1=left slip, 2=right slip
← DFDX	real*4	2x2 array, df/dx (partial derivatives)
← FNEG	real*4	2 negative error functions in equations

Common Blocks: SUSP TSOLVE VARS

Functon DFYDA(ALPHA, GAMMA, FZ, AXLE)

Compute cornering stiffness as a function of FZ. In this version (linear default) the arguments ALPHA and GAMMA are not used.

→ ALPHA	real*4	slip angle
→ GAMMA	real*4	camber angle
→ FZ	real*4	vertical load
→ AXLE	integer	axle number (1 or 2)

← DFYDA	real*4	partial derivative of Fy with respect to alpha.

Common Blocks: TIRE

Functon DFYDG (ALPHA, GAMMA, FZ, AXLE)

Compute camber stiffness as a function of FZ. In this version (linear default) the arguments ALPHA and GAMMA are not used.

→ ALPHA	real*4	slip angle
→ GAMMA	real*4	camber angle
→ FZ	real*4	vertical load
→ AXLE	integer	axle number (1 or 2)
← DMZDA	real*4	partial derivative of Mz with respect to alpha.

Common Blocks: TIRE

Functon DMZDA(ALPHA, GAMMA, FZ, AXLE)

Compute aligning stiffness as a function of FZ. In this version (linear default) the arguments ALPHA and GAMMA are not used.

→ ALPHA	real*4	slip angle
→ GAMMA	real*4	camber angle
→ FZ	real*4	vertical load
→ AXLE	integer	axle number (1 or 2)
← DFYDA	real*4	partial derivative of Fy with respect to alpha.

Common Blocks: TIRE

DRIVE1 (DFW)

Read driver model parameters.

← DFW	real	initial average front wheel angle = 0

Common Blocks: DRVMOD GLBL PARS

DRIVER (X, Y, DFW, DFWNOW)

Calculates closed-loop driver steering control angle.

→ X	real	current time
→ Y	real	driver model state vector
← DFW	real	calculated average front wheel angle.
→ DFWNOW	real	current average front wheel angle.

Common Blocks: AERO GLBL PARS DRVMOD

DRIVET (X, Y, DRTORQ, DRTNOW)

Calculates closed-loop driver steering wheel control torque.

→ X	real	current time
→ Y	real	driver model state vector
← DRTORQ	real	calculated steering wheel torque.
→ DRTNOW	real	current steering wheel torque.

Common Blocks: AERO GLBL PARS DRVTOR VARS

DRIVGO

Initialize driver model parameters for steering angle version of driver model.

Common Blocks: DRVMOD GLBL PARS VARS TIRE

DRIVGT

Initialize driver model parameters for torque version of driver model.

Common Blocks: DRVTOR GLBL PARS VARS TIRE

ECHO

Echo parameter values to file to verify that the input was interpreted correctly.

Common Blocks: GLBL PARS MNVR SUSP TIRE AERO
PRNT VARS

FDAMP (VZ, VROLL, VPITCH, FD)

Compute the damping force for all four wheels.

→ VZ	real*4	vertical velocity of vehicle sprung mass c.g.
→ VROLL	real*4	roll velocity of vehicle sprung mass.
→ VPITCH	real*4	pitch velocity of vehicle sprung mass.
← FD	real*4	2 x 2 matrix of damping forces at each wheel.

Common Blocks: SUSP

This subroutine uses different rates for jounce and rebound. The sign convention is that jounce → positive damping force.

FUNCTN (T, Y, YP)

Compute derivatives of state variables in the vehicle/steering model.

→ T	real*4	Time (independent variable of integration)
→ Y	real*4	1-D array of 13 state variables
← YP	real*4	1-D array of 13 derivatives: yp(i) = dy(i) / dt

Common Blocks: GLBL PARS SUSP AERO VARS TIRE

Subroutine FUNCTN contains the equations of motion for the 5-d.o.f vehicle model and a 1-d.o.f steering system model.

Functon FWIND (T)

This function provides a wind profile as a function of time. The default version uses filtered random noise to generate a random wind with a PSD that falls off at 6 dB/oct. It can be replaced to provide a different type of profile.

→ T	real*4	time

Common Blocks: GLBL

GMPRD (A, B, R, N, M, L)

Calculates the product of two matrices.

→ A	real	N x M input matrix
→ B	real	M x L input matrix
← R	real	N x L output matrix equal to product of A and B
→ N	integer	row dimension of A
→ M	integer	column dimension of A and row dimension of B
-> L	integer	column dimension of B

INDATA

Prompt user for name of input file, then open file and read input data.

Common Blocks: DRVMOD GLBL PARS MNVR SUSP TIRE AERO PRNT

INDSUS(IX, DFDX, FNEG)

Compute 2x2 Jacobian and 2-element error array for a tire on an independent suspension. This routine is used by the Newton-Raphson solver, MNEWT.

→ X	real*4	2-element array. x(1)=slip, x(2)=camber
← DFDX	real*4	2x2 array, df/dx (partial derivatives)

← FNEG	real*4	2 negative error functions in equations

Common Blocks: SUSP TSOLVE

INIT

Initialize input-based values and non-zero variables.

Common Blocks: GLBL PARS SUSP AERO VARS PRNT

Function LENSTR(STRING)

Count characters in left-justified string.

→ STRING	char	left-justified string
← LENSTR	integer	number of significant characters in STRING

MAIN—WIND

Main program module that controls the wind & handling simulation.

Common Blocks: DRVMOD GLBL PARS SUSP VARS AERO PRNT MNVR

MNEWT(X, USRFUN, IERR)

This routine is based on MNEWT from the Numerical Recipes library. It has been "hard-wired" for the Chrysler vehicle handling model. The error threshold is less than .01 deg

↔ X	real*4	2-element array with variables to solve for.
→ USRFUN	sub.	subroutine to provide ALPHA and BETA arrays.
← IERR	integer	error code. 0=OK, 1=didn't converge.

OPNOUT

Write header portion of the output ERD file, and compute constants used later.

Common Blocks: GLBL PARS

OUTPUT (T, Y, YP)

Write predicted response variables into output file and show progress on screen.

→ T	real	Time.
→ Y	real	1-D array with state variables of system.
→ YP	real	1-D array with derivatives of state variables.

Common Blocks: GLBL PARS VARS AERO

Function POLY4(COEF, FZ)

evaluate 4-th order polynomial

→ COEF	real	array of 4 coeficients.
→ FZ	real	load (independent variable).

ROLLAX (ROLL, YROLAX, HROLAX, IXSRA)

Compute instantaneous lateral and vertical distances of the sprung mass c.g. from the roll axis.

→ ROLL	real*4	Roll angle of sprung mass.
← YROLAX	real*4	Lateral distance (in a non-rolling frame) between c.g. of sprung mass and roll axis.
← HROLAX	real*4	Horizontal distance (in a non-rolling frame) between c.g. of sprung mass and roll axis.
← IXSRA	real*4	Moment of inertia of the sprung-mass about the instantaneous roll axis.

Common Blocks: PARS SUSP

Function STEER(T)

Return steering wheel angle or steering wheel torque as function of time.

← STEER	real*4	Steering wheel angle.
→ T	real*4	Time.

Common Blocks: MNVR VARS GLBL PARS DRVMOD

The angle (or torque) is determined by one of three methods, dependent upon the variable NSTEER in the common block MNVR: (1) if NSTEER < 0, the UMTRI driver model is used; (2) if NSTEER = 0, a sinusoidal function is used; and (3) if NSTEER > 0, a table look-up is used.

Function SUM(MATRIX)

Sum values in a 4-element matrix.

→ MATRIX	real*4	matrix with 4 elements (2 x 2), (4 x 1), or (1 x 4).
← SUM	real*4	Sum of values in matrix.

TABLE(M, N, X, Y, Z, Q)

Table look-up routine.

→ M	integer	index of X-Y table (arrays) at which to start search
→ N	integer	index of X-Y table (arrays) at which to end search
→ X	real	N-array of abscissa table values
→ Y	real	N-array of ordinate table values
→ Z	real	scalar abscissa value
← Q	real	scalar ordinate value of X-Y table corresponding to Z

TIMDAT(TIMEDT)

Obtain the current time and date.

← TIMEDT	char*24	String containing time and date.

Functon TIREFY (ALPHA, GAMMA, FZ, AXLE)

Compute tire side force.

→ ALPHA	real*4	slip angle
→ GAMMA	real*4	camber angle
→ FZ	real*4	vertical load
→ AXLE	integer	axle number (1 or 2)
← TIREFY	real*4	tire side force

Common Blocks: TIRE

Functon TIREMZ (ALPHA, GAMMA, FZ, AXLE)

Compute tire aligning torque.

→ ALPHA	real*4	slip angle
→ GAMMA	real*4	camber angle
→ FZ	real*4	vertical load
→ AXLE	integer	axle number (1 or 2)
← TIREMZ	real*4	tire aligning torque

Common Blocks: TIRE

Functon TIRES (BETA, V, VYAW, ROLL

This subroutine solve simultaneous equations for slip and camber angles and tire forces and moments.

→ BETA	real*4	slip angle
→ V	real*4	speed
→ VYAW	real*4	yaw rate

→ ROLL	real*4	roll angle

Common Blocks: TSOLVE SUSP VARS PARS

TRAJ (X, XT, YT, YPATH)

Obtains the previewed lateral path position (relative to the vehicle heading).

→ X	real	forward preview distance
→ XT	real	x-coordinates of path in vehicle axis system at X ahead
→ YT	real	y-coordinates of path in vehicle axis system at X ahead
→ YPATH	real	lateral offset of path from vehicle at X ahead

TRANS

Calculates transition matrix for driver model internal vehicle model. (without steering system)

Common Blocks: DRVMOD

TRANST

Calculates transition matrix for driver model internal vehicle model. (with steering system)

Common Blocks: DRVTOR PARS GLBL TIRE VARS

WHEELZ (Z, ROLL, PITCH)

Update matrices in the common block /VARS/ based on the new position of the sprung mass.

→ Z	real*4	Vertical position of sprung mass c.g. (in).
→ ROLL	real*4	Roll angle of sprung mass (rad)
→ PITCH	real*4	Pitch angle of sprung mass (rad)

Common Blocks: SUSP VARS

The matrices ZW, FZ, KNMSTR, KNMCBR in common /VARS/ are updated. The quantities computed for each wheel are: vertical displacement, normal ground load, bump-steer angle and bump-camber angle for each wheel, relative to static trim. roll-center heights are assumed fixed relative to the road for the calculation of lateral load transfer.

4.4 The Output File

There are at least two reasons why one might wish to modify the existing format of the output file created by the Wind-Steer program: (1) to add or delete variables of interest, or (2) to set the format to match established post-processing software other than the software used within ERD at UMTRI.

Method Used to Write Time Histories

The code for writing the output file is contained in two program modules: (1) OPNOUT opens the output file and writes the header information, and (2) OUTPUT writes the values of output variables at discrete time intervals. Only these two subroutines need to be modified. (In reading the following descriptions, it may be helpful to also view the source code listings for those subroutines, contained in Appendix B.)

Most of the the code in subroutine OPNOUT assigns names to character variables. Then, at the bottom of the subroutine, those variables are written into the output file in the format required for an ERD header. Similarly, most of the code in OUTPUT assigns values to elements in a REAL array. Then, at the bottom of the subroutine, those variables are written into the output file in the format required for an ERD header. It is essential that the one-to-one correspondence is maintained between labels for variables and values for the variables. As long as the two forms of data are properly paired, the number of variables and their order really doesn't matter.

Both subroutines use a variable called NCHAN to identify the channel number being considered. For each value of NCHAN, the following assignments are made in OPNOUT:

- a 32-character name for the variable of interest is assigned to the character*32 Fortran array element LONGNM (NCHAN), e.g., "Input Steer Angle"
- an 8-character name for the variable of interest is assigned to the character*8 Fortran array element SHORTN (NCHAN), e.g., "Steer In"
- a 32-character generic name for the variable of interest is assigned to the character*32 Fortran array element GENNM (NCHAN), e.g., "Steer Angle"
- an 8-character name for the units of the variable of interest is assigned to the character*8 Fortran array element UNITNM (NCHAN), e.g., "deg"
- a 32-character generic name for the rigid body associated with the variable of interest is assigned to the character*32 Fortran array element RIGBOD (NCHAN), e.g., "Input"

In subroutine OUTPUT, for each value of NCHAN, an appropriate value is assigned to the array element BUFFER (NCHAN).

At the bottom of each subroutine, the value of NCHAN is equal to the total number of channels that are written into the output file.

The channel definitions are grouped such that variables that apply to the input or the entire vehicle are handled first. Variables that apply to each wheel (suspension and tire variables) are handled in two nested DO loops. The outer loop goes from the front axle to the rear, and the inner loop goes from the left side to the right. Thus, each block of code within the loops gets executed four times.

Deleting Variables

To delete a variable, a block of code is removed from the OPNOUT subroutine and a corresponding block is removed from OUTPUT. The block of code in OPNOUT begins with comments describing the variable, then the statement "NCHAN = NCHAN + 1," and then five assignment statements for element NCHAN of arrays LONGNM, SHORTN, UNITNM, GENNM, and RIGBOD. Delete all of these lines or comment them out (insert a C in column 1 of each line so that the line is ignored by the Fortran compiler). Identify the corresponding assignment statement in OUTPUT and delete also (or comment it out). It is usually necessary to modify some of the lines following the deleted line in OUTPUT so that the following values are put into lower indexed elements of the array BUFFER.

For example, suppose we want to delete the Z deflection of the vehicle body. The block of code in subroutine OPNOUT that provides the labels is the following:

```
      ...
      UNITNM (NCHAN) = UDIST
      RIGBOD (NCHAN) = THISRB
C
C  Z Position
C
      NCHAN = NCHAN + 1
      LONGNM (NCHAN) = 'Z Position, Sprung Mass cg'
      SHORTN (NCHAN) = 'Z cg'
      GENNM (NCHAN) = 'Z Position'
      UNITNM (NCHAN) = UDISP
      RIGBOD (NCHAN) = THISRB
C
C  Roll Angle
C
      NCHAN = NCHAN + 1
      LONGNM (NCHAN) = 'Roll Angle'
      ...
```

The underlined lines would be deleted. The code in subroutine OUTPUT that includes this variable is the following:

```
C
C  Body position variables
C
      BUFFER (NCHAN + 1) = Y(1) / ININFT
      BUFFER (NCHAN + 2) = Y(2) / ININFT
      BUFFER (NCHAN + 3) = Y(3)
      BUFFER (NCHAN + 4) = Y(4) * TODEG
      BUFFER (NCHAN + 5) = Y(5) * TODEG
      BUFFER (NCHAN + 6) = Y(6) * TODEG
      NCHAN = NCHAN + 6
```

From viewing the definitions of the Y array, it turns out the Y(3) is the Z variable. The underlined code would be modified as follows:

```
C
C  Body position variables
C
      BUFFER (NCHAN + 1) = Y(1) / ININFT
      BUFFER (NCHAN + 2) = Y(2) / ININFT
      BUFFER (NCHAN + 3) = Y(4) * TODEG
      BUFFER (NCHAN + 4) = Y(5) * TODEG
      BUFFER (NCHAN + 5) = Y(6) * TODEG
      NCHAN = NCHAN + 5
```

The line that set the value in the buffer was deleted, and the following lines were modified so that at the end of the block NCHAN was incremented by 5, rather than 6 as before.

Adding Variables

To add a variable, a new block of code is added to subroutine OPNOUT and a corresponding block is added to OUTPUT. The code added to OPNOUT should (1) provide labels for element NCHAN of the arrays LONGNM, SHORTN, UNITNM, GENNM, and RIGBOD, and (2) the variable NCHAN should be properly incremented. The code added to OUTPUT should (1) provide the value of the new variable and put it into the element NCHAN of the array BUFFER, and (2) the variable NCHAN should be properly incremented. The location of the added code defines where the new variable is situated relative to the existing output variables. The only restriction is the the order of channels in BUFFER must match the order of the labels in each of the character arrays.

Changing the Format of the Output File

As the Wind-Steer program exists at UMTRI, the output file follows the ERD format. The numerical values of the output variables can be written in binary form, or in text form using a Fortran FORMAT that was specified in line 5 of the input file. The existing flexibility should be sufficient to accommodate any desired formats for the output. For example, if a plotting program expects to find columns of numbers separated by commas, the following FORMAT could be put into line 5 of the input file:

(100(F10.2,1X))

If the existing flexibility is not sufficient, the code that writes into the output file can be replaced as needed. (It lies at the bottom of the OUTPUT subroutine.)

The header portion of the file is more likely to cause problems with post-processing software. The code that writes the header is contained in the bottom of the OPNOUT subroutine, and is shown (partially) below:

```
C
C  Write standard ERD file heading.
C
      WRITE(IOUT,'(A)') 'ERDFILEV2.00'
      WRITE(IOUT,410) NCHAN, NSAMP, NRECS, NBYTES, NUMKEY, DT*IPRINT
  410 FORMAT(5(I6,','),E13.6)
  411 FORMAT (A8,255A8)
  412 FORMAT (A8, 31A32 : 2(/'&1000   ', 31A32))
```

```
      WRITE(IOUT,'(A,A)') 'TITLE    ', TITLE
      WRITE(IOUT,411) 'SHORTNAM', (SHORTN(J), J=1, NCHAN)
      ...
      WRITE (IOUT, '(A,A)') 'HISTORY Input file was ', FNREAD
      WRITE (IOUT, '(A)') 'END'
      ...
```

This is the only code that is modified to change the form of the file header. Most of the code above this section consists of statements that assign labels to arrays of character variables. Some of those labels can be printed in a different format if desired. For example, suppose that a plotter expects to find labels enclosed in double quotes on the first line, followed by numbers separated by commas. Also suppose that the short labels (8 characters or less) are the appropriate length for the plotter. Then the existing code could be replaced with the following:

```
C
C  Write 1-line heading with labels enclosed in double-quotes and
C  separated by commas.  e.g., "Time", "Steer In", ...
C
      WRITE(IOUT,411) 'SHORTNAM', (SHORTN(J), J=1, NCHAN)
  411 FORMAT (100('"',A8,'"',1X)
      ...
```

4.5 Changing the Tire Model

The formulation used to incorporate the tire side force and aligning torque into the vehicle dynamics is independent of the formulation used to define the side force and aligning torque as functions of slip and camber angle. (See Section 3.6.) The tire model is defined within five subprogram modules: DFYDA, DFYDG, DMZDA, TIREFY, and TIREMZ. The first two define partial derivatives of side force with respect to alpha and gamma, the third defines the partial derivative of aligning torque with respect to alpha, and the last two compute side force and aligning torque.

To change the tire model, these five modules are replaced. No other changes should be made in the program, with the possible exceptions of the input and echo routines if the new model requires additional input parameters.

APPENDIX A — DRIVER MODEL

This appendix contains copies of two technical papers which fully document the concepts implemented in the computer code used to represent the driver model closed-loop steering control process. Additional documentation is provided by comments contained in the computer code itself; see subroutines DRIVGO, DRIVE1, TRANS, DRIVER, AND TRAJ.

The material is reproduced here by permission of the *IEEE Transactions on Systems, Man, and Cybernetics* journal (Copyright IEEE), and the *ASME Journal of Dynamic Systems, Measurement, and Control* (Copyright ASME).

Application of an Optimal Preview Control for Simulation of Closed-Loop Automobile Driving

CHARLES C. MACADAM

Abstract—An optimal preview control method is applied to the automobile path following problem. The technique is first used to examine the straight-line regulatory driving task and results compared with similar experimental measurements. The method is further demonstrated by closed-loop simulation of an automobile driver/vehicle system during transient lane-change maneuvers. The computer simulation results are compared with equivalent vehicle test measurements.

I. INTRODUCTION

THIS PAPER presents example applications (to the automobile path following problem) of a general method of control synthesis presented in [1]. The method is demonstrated here by simulation of a closed-loop automobile/driver system and the results compared with driver/vehicle test measurements. Results for the optimal preview control are also discussed within the context of manual control pursuit tracking task findings.

The control technique demonstrated herein is designed for application to linear time-invariant systems utilizing preview control strategies for regulation or tracking tasks. A common example of this type of control strategy occurs during normal automobile path following in which drivers "look-ahead" to follow a desired path. Human operators, as part of various man–machine systems, typically employ preview control strategies to control and stabilize such systems. It is widely recognized that human operators are capable of controlling and adapting to a wide variety of dynamical systems, many of which are vehicles with preview-oriented control requirements such as automobiles, bicycles, and complex aircraft [2]–[8]. Clearly human control of most vehicles would not be possible without some training by the operator to acquire an understanding of the vehicle response to various control inputs. While a certain portion of this training serves to identify and reinforce learned open-loop responses for repeated and familiar control task scenarios, the remainder frequently serves to identify and reinforce the operator's understanding or "feel" of the vehicle response to control inputs continually in use for closed-loop regulation and/or pursuit needs. It is in this latter control category for general linear system representations capable of preview control strategies, that the method presented in [1] can find particular application. As will be demonstrated in this paper, application to the automobile path following problem produces substantive agreement when compared with driver/vehicle experimental measurements for both straight-line regulatory driving and transient lane-change maneuvers.

Manuscript received October 10, 1980; revised March 2, 1981.
The author is with the Highway Safety Research Institute of the University of Michigan, Ann Arbor, MI 48109.

II. THE OPTIMAL PREVIEW CONTROL

Before applying the optimal preview control of [1] to the automobile path following problem, the main results and symbol definitions contained therein are briefly reviewed in this section for later reference. As derived in [1], for the linear system

$$\dot{x} = Fx + gu \tag{1}$$

$$y = m^T x \tag{2}$$

where

- x $n \times 1$ state vector,
- y scalar output related to the state by the $n \times 1 m^T$ constant observer vector transpose,
- F constant $n \times n$ system matrix,

and

- g constant $n \times 1$ control coefficient vector,

the optimal control $u^0(t)$ which minimizes a special form of the local performance index,

$$J \triangleq \frac{1}{T}\int_t^{t+T}\{[f(\eta) - y(\eta)]W(\eta - t)\}^2\, d\eta \tag{3}$$

over the current preview interval $(t, t+T)$ where

- W arbitrary weighting function over the preview interval

and

- f previewed input,

is given by

$$u^0(t) = \left[\int_t^{t+T}\left\{f(\eta) - m^T\left[I + \sum_{n=1}^{\infty}\frac{F^n(\eta - t)^n}{n!}\right]x(t)\right\}\cdot\left\{(\eta - t)m^T\left[I + \sum_{n=1}^{\infty}\frac{F^n(\eta - t)^n}{(n+1)!}\right]g\right\}W(\eta - t)\, d\eta\right] \Big/ \left[\int_t^{t+T}\left\{(\eta - t)m^T\left[I + \sum_{n=1}^{\infty}\frac{F^n(\eta - t)^n}{(n+1)!}\right]g\right\}^2 \cdot W(\eta - t)\, d\eta\right] \tag{4}$$

where I is the identity matrix. For the special case of $W(\eta - t) = \delta(T^*)$, the Dirac delta function for $0 < T^* \leq T$, (4) simplifies to

$$u^0(t) = \frac{f(t+T^*) - m^T\left[I + \sum_{n=1}^{\infty} \frac{F^n(T^*)^n}{n!}\right]x(t)}{T^* m^T\left[I + \sum_{n=1}^{\infty} \frac{F^n(T^*)^n}{(n+1)!}\right]g} \tag{5}$$

$$= [f(t+T^*) - y_0(t+T^*)]/(T^*K), \tag{6}$$

the single-point preview control version of (4), where

$$K \triangleq m^T\left[I + \sum_{n=1}^{\infty} \frac{F^n(T^*)^n}{(n+1)!}\right]g.$$

Equation (6) represents a proportional controller with gain inversely related to the preview interval T^* and operating on the error between the previewed input $f(t + T^*)$ and $y_0(t + T^*)$, that portion of the previewed output deriving from the state vector's current initial condition. Likewise (4) can be interpreted as a proportional controller operating on a similar error averaged and weighted over the preview interval $(t, t + T)$ by the additional terms appearing in (4).

It is also shown in [1] that the optimal solution $u^0(t)$ can be expressed in terms of any current nonoptimal $u(t)$ and correspondingly nonzero preview output error $\epsilon(t)$ as

$$u^0(t) = u(t) + \frac{\int_t^{t+T} \epsilon(\eta)A(\eta)W(\eta - t)\,d\eta}{\int_t^{t+T} A^2(\eta)W(\eta - t)\,d\eta} \tag{7}$$

where

$$A(\eta) \triangleq (\eta - t)m^T\left[I + \sum_{n=1}^{\infty} \frac{F^n(\eta - t)^n}{(n+1)!}\right]g$$

$$\epsilon(\eta) \triangleq f(\eta) - m^T\phi(\eta, t)x(t) - u(t)A(\eta)$$

$$\phi(\eta, t) \triangleq I + \sum_{n=1}^{\infty} \frac{F^n(\eta - t)^n}{n!}.$$

For the special case of $W(\eta - t) = \delta(T^*)$, as before, (7) reduces to

$$u^0(t) = u(t) + \frac{\epsilon(t + T^*)}{T^* \cdot K}. \tag{8}$$

The formulation expressed by (7) can be useful in describing systems which do not achieve, though closely approximate, the defined optimal system behavior. Such cases may arise from limitations in achieving the precise optimal control due to time lags or dynamic properties inherent in the controller and not accounted for *a priori* in the optimization. The next two sections adopt this view for the car/driver man–machine system in an attempt to describe and explain actual closed-loop driving behavior.

Finally, it was also shown in [1] that information concerning stability of the closed-loop system utilizing the optimal preview control of (4) or (7) is provided by the characteristic roots of the constant matrix

$$[F - gc^T] \tag{9}$$

where

$$c^T = \frac{m^T \int_0^T \phi(\eta, 0)\left\{\eta m^T\left[I + \sum_{n=1}^{\infty} \frac{F^n(\eta)^n}{(n+1)!}\right]g\right\}W(\eta)\,d\eta}{\int_0^T \left\{\eta m^T\left[I + \sum_{n=1}^{\infty} \frac{F^n(\eta)^n}{(n+1)!}\right]g\right\}^2 W(\eta)\,d\eta}.$$

For the special case of $W(\eta) = \delta(T^*)$, (9) becomes

$$F - \left\{gm^T\left[I + \sum_{n=1}^{\infty} \frac{F^n(T^*)^n}{n!}\right] \Big/ (T^* \cdot K)\right\}. \tag{10}$$

III. Application to Manual Control Pursuit Tracking Tasks as Represented by Straight-Line Automobile Driving

The most well-known and characteristic property exhibited by human operators in tracking tasks is the transport delay deriving from perceptual and neuromuscular mechanisms. By introducing this inherent delay property *a posteriori* in the optimal preview control formulation, excellent agreement can be demonstrated between typical manual control pursuit tracking task results and the resulting optimal preview controller modified to include the inherent transport delay (heretofore referred to as the "modified" optimal preview control).

For reasons of clarity and notational simplicity, the discussion in this section will make use only of (8), the single-point preview control version of (7). Equation (8) can be represented by the block diagram of Fig. 1, where $G(s) = [Is - F]^{-1}g$ represents the controlled element vector transfer function, and $u(t)$, the current control, is related to the optimal control $u^0(t)$ by a transfer function $H(s)$ (previously assumed equal to one in the derivation of the optimal control $u^0(t)$). The introduction of the $H(s)$ transfer function is useful in describing systems which function (or are presumed to do so) in an error minimization fashion, but fail to achieve the precise optimal control due to an inherent limitation within the controller or control process itself, e.g., delays resulting from processor calculations and sample hold operations in digital systems, or perceptual/neuromuscular lags in the case of a human controller. By letting $H(s) = e^{-s\tau}$, those actual delay limitations displayed by human operators during tracking tasks can be approximated by the parameter τ, an effective transport lag. By incorporating this approximation and noting then that the transfer function relating $u(t)$ and $\epsilon(t + T^*)$ is $e^{-s\tau}/(1 - e^{-s\tau})KT^*$, Fig. 1 reduces to Fig. 2, a single-loop pursuit tracking formulation. The open-loop transfer function $Y_0(s)$ relating $y(t + T^*)$ and $\epsilon(t + T^*)$ is given by

$$Y_0(s) = \frac{e^{-s\tau}}{1 - e^{-s\tau}}\left[1 + \frac{m^T\phi(t + T^*, t)G(s)}{KT^*}\right]. \tag{11}$$

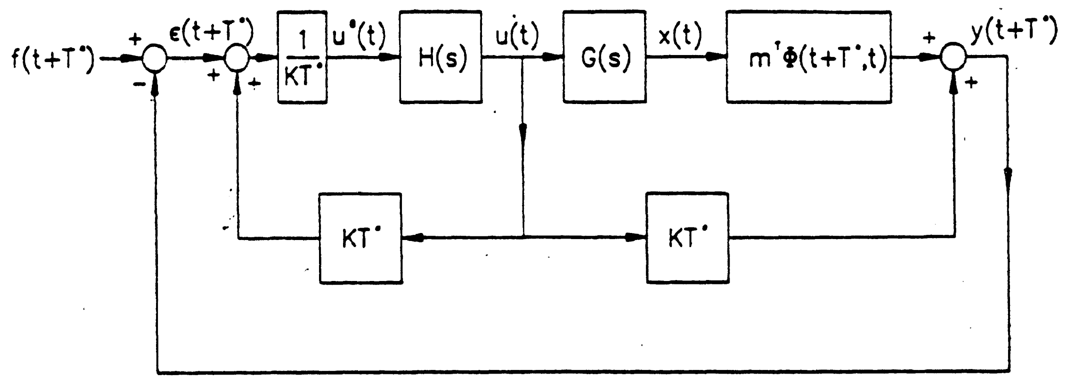

Fig. 1. Block diagram for the single-point preview control.

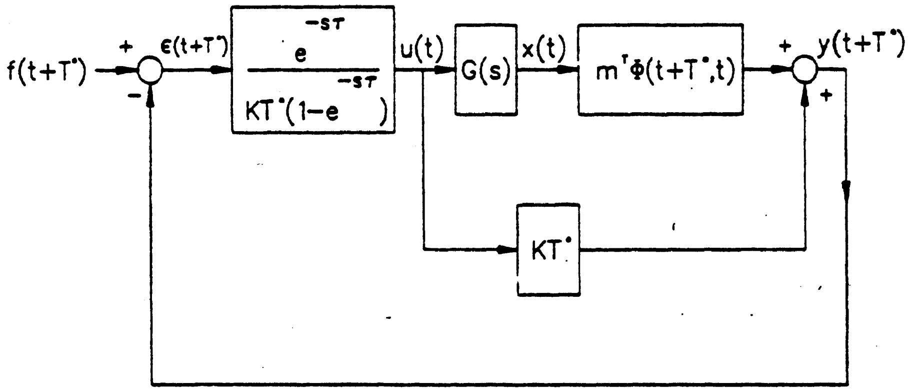

Fig. 2. Equivalent block diagram for the single-point preview control, $H(s) = e^{-s\tau}$.

The stability of this system is determined by the characteristic roots of $1 + Y_0(s)$, or equivalently,

$$1 + e^{-s\tau} m^T \phi(t + T^*, t) G(s)/KT^* = 0. \qquad (12)$$

To test the utility of this model by comparison with experimental findings, open-loop gain/phase frequency response results measured by Weir *et al.* [9, Fig. 12-C] for an automobile straight-line regulatory control task are presented in Figs. 3 and 4. These experimental results represent the open-loop frequency response relating the driver's output (presumably an estimate of future lateral position) to an assumed error, derived by the driver, between the previewed input (straight road ahead) and the driver's output. Since this may be categorized as a form of linear pursuit tracking, the formulation of (11) is accommodated. Also shown in Figs. 3 and 4 is the frequency response calculation for (11) with parameters $T^* = 3.0$ (s) and $\tau = 0.26$ (s). The model output $y(t + T^*)$ is the estimated vehicle lateral position at time $t + T^*$; the input $f(t + T^*) \equiv 0$ is the lateral displacement of the previewed path. The automobile (F, g) dynamics used in (11) appear in Appendix I-A and duplicate those identified in [9]. The values of T^* and τ were selected to fit the experimental data as closely as the single-point model would permit. As can be seen, the model and experimental results display excellent agreement. Not only does the preview model reproduce the -6 db/octave slope of the familiar manual control "crossover" model [2], [8] gain characteristic, but also the peaking phase characteristic usually displayed in manual control task experimental data of this kind.

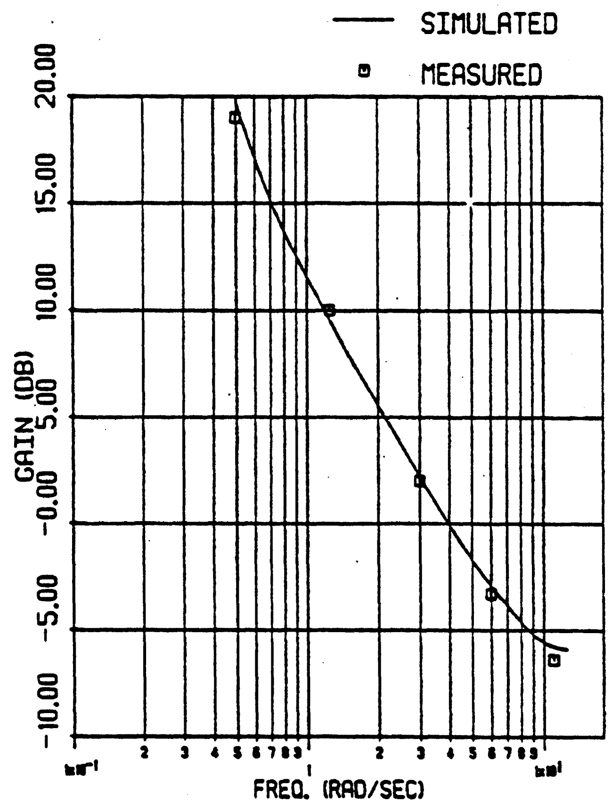

Fig. 3. Frequency response gain comparison.

The model parameters T^* and τ appearing in (11) represent the average preview time used by the driver and his/her effective transport lag associated with this particular control task. The values of T^* and τ used here fall well within the range identified by other investigators studying straight-line automobile driving [10]–[12] and human operator tracking performance [2], [4], [9].

Interestingly, for the relatively simple control task of typical straight-line automobile regulation as discussed here,

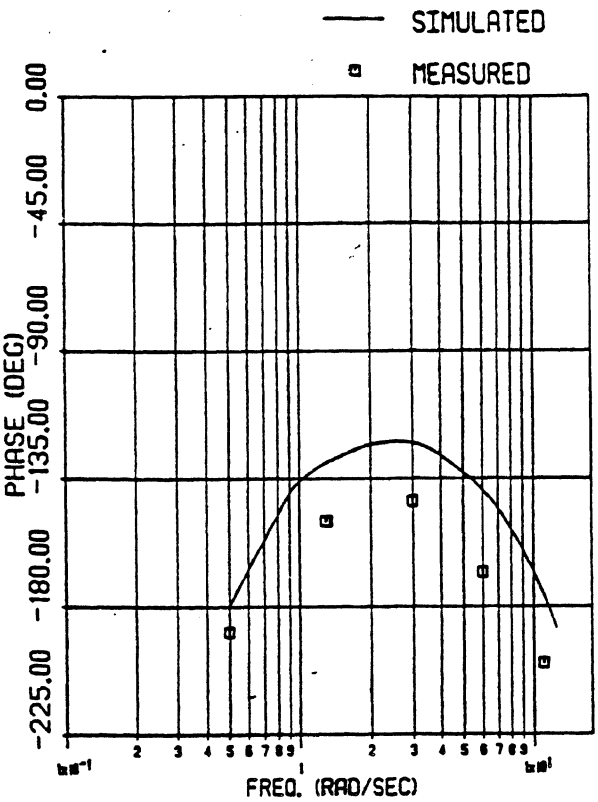

Fig. 4. Frequency response phase comparison.

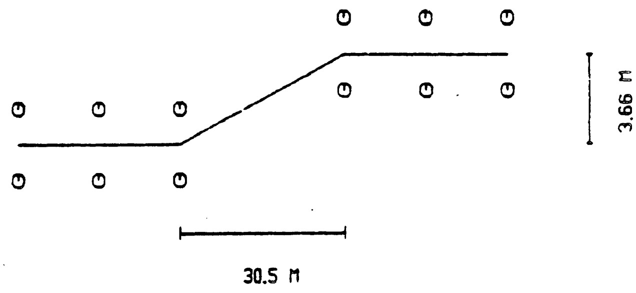

Fig. 5. Lane-change test course.

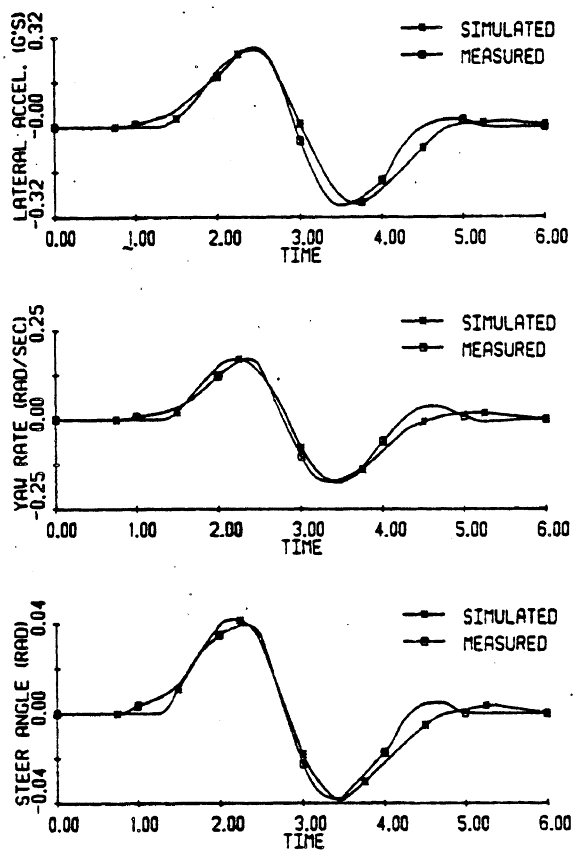

Fig. 6. Closed-loop simulation/test result comparison.

the vehicle dynamics portion of the total transfer function (11) does not play a dominant role except at very low frequencies. As a result, the open-loop transfer function gain characteristic (11) is closely approximated by the human operator term, $e^{-s\tau}/(1-e^{-s\tau}) \approx e^{-s\tau}/\tau s$. Such a result would support the well-known fact that tracking task test results for simple automobile regulation [8], [9] can generally be approximated by the "cross-over" model form $Ce^{-s\tau}/s$ (C being the "cross-over" gain constant) in the vicinity of the cross-over frequency. Moreover, in such cases where the above approximation does hold, $1/\tau$ becomes C in the "cross-over" model representation.

For the simple manual control pursuit tracking task, as represented here by straight-line automobile regulation, the modified optimal preview controller, even employed in only a single-point form $[W(\eta - t) = \delta(T^*)]$, appears to accurately mimic human control behavior. It might, therefore, seem reasonable to conjecture that human operator strategy during simple pursuit tracking (or at least straight-line automobile regulation) is closely akin to an optimal preview error minimization process which ignores or is unaware of transport delay mechanisms inherent in the control processor. A more stringent test of this hypothesis is offered in the following section wherein transient automobile path following is examined using the modified optimal preview control model in its complete form.

Application of the Optimal Preview Control for Simulation of Closed-Loop Transient Automobile Path Following

The previous section addressed the applicability of the optimal preview control to the problem of preview regulation and the effects of an inherent transport delay within the controller. Using straight-line automobile regulation as an example, the single-point preview model was compared with experimental results within the frequency domain. In this section application to the tracking problem is demonstrated using the general preview control model (7), with an inherent transport time delay to simulate a closed-loop automobile/driver path following maneuver. Results from the model are compared with time history measurements from corresponding full-scale vehicle tests.

The specific closed-loop maneuver examined here required an automobile driver to perform a standard 3.66 m (12-ft) lane-change within a distance of 30.5 m (100 ft) at a vehicle speed of approximately 26.8 m/s (60 mi/h). The initiation and completion of the lane change was constrained by 3.05-m wide (10 ft) cone-marked lanes (Fig. 5). The test vehicle was a standard American compact with measured parameter values shown in Appendix I-B. A representative test result for this vehicle/driver combination appears in Fig. 6, showing recorded-time histories of lateral acceleration, yaw rate, and front-wheel steer angle [13].

Also shown in Fig. 6 are computer simulation results using the optimal preview control (7) with an assumed human operator transport delay term $e^{-s\tau}$ relating $u^0(t)$ and $u(t)$. The transport lag term is included here, as in the previous section, to approximate the principal human operator lag effects. The calculation of (7), steer angle, seen in Fig. 6 is for values of $\tau = 0.2$ (s) and $T = 1.3$ (s) using ten equally spaced points in the preview interval to approximate the integral. The values of T and τ were selected to closely fit the test measurements. The (F, g) automobile dynamics model is the same two-degree-of-freedom model appearing in Appendix I-A, evaluated for the parameter values identified in Appendix I-B. The previewed input $f(\eta)$ appearing in (7) represents the desired lateral path deviation and was obtained during the simulation using the simple straight-line path segments shown in Fig. 5 as input.

As seen from Fig. 6, excellent agreement can be obtained between the experimental results and simulation predictions using the two numerical parameters (τ, T) and a simple straight-line path input. Variations in the value of τ primarily influenced the closed-loop system damping; larger values producing reduced damping. Variations in the value of T influenced control (steering) amplitude as well as damping; larger values of T producing lower control amplitude and increased damping.

Finally, Fig. 7 shows a comparison of the preview model predictions and measured test results for a modified set of vehicle dynamics (F, g). The same vehicle was employed but with modifications to its mass center and rear tires so as to produce a new set of parameter values listed in Appendix I-C. As shown in Fig. 7 the principal change in the closed-loop response from Fig. 6 is an increased steering gain (lower steering amplitude for the same nominal maneuver) and decreased damping. Larger values of τ (0.3) and T (1.55) were required in the calculation of (7), shown as steer angle in Fig. 7, to better approximate the reduced damping and smaller amplitude steering control. A comparison of computed vehicle path trajectories, corresponding to the baseline and modified vehicle responses shown in Figs. 6 and 7, appears in Fig. 8.

Characteristic roots for each of the closed-loop systems, as calculated from the constant matrix (13), are shown in Fig. 9. The matrix (13) (see Appendix I-D) is similar to that given by (9) but includes the influence of the transport lag term $e^{-s\tau}$ approximated by the first-order Padé polynomial

$$\frac{1-\frac{\tau}{2}s}{1+\frac{\tau}{2}s}$$

$$\left[\begin{array}{c|c} F & g \\ \hline c^T\left(F-\frac{2}{\tau}I\right) & c^Tg-\frac{2}{\tau} \end{array}\right]. \tag{13}$$

Note that the reduced damping in the driver/vehicle responses, displayed in Figs. 7 and 8, is equivalently represented by the corresponding closed-loop characteristic root locations shown in Fig. 9.

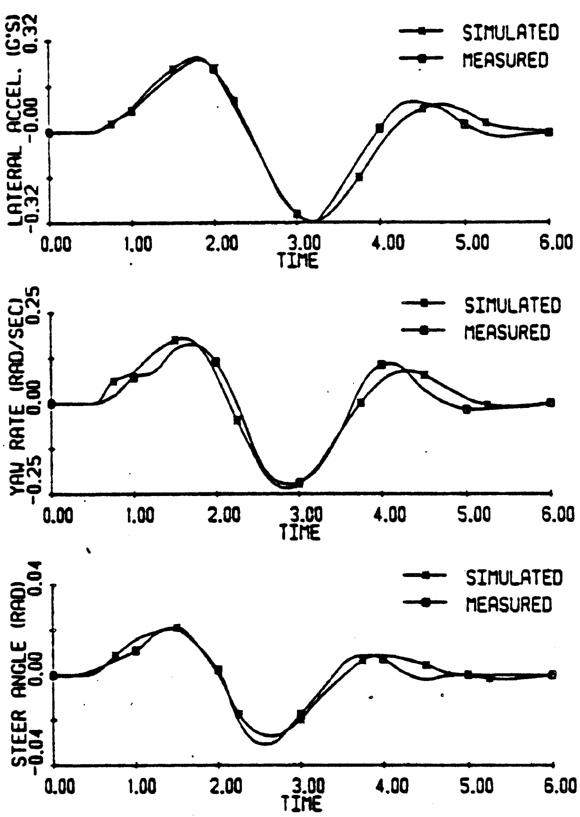

Fig. 7. Closed-loop simulation/test result comparison—modified vehicle.

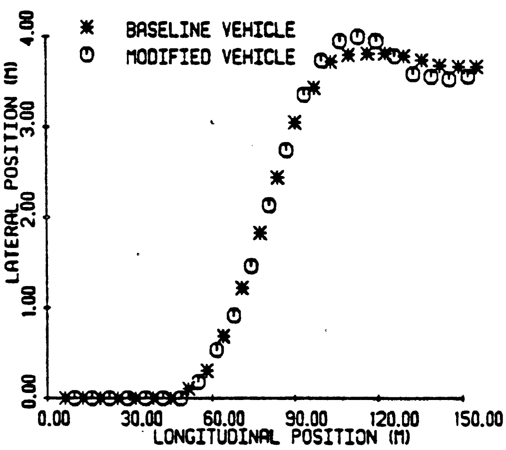

Fig. 8. Simulated path trajectories.

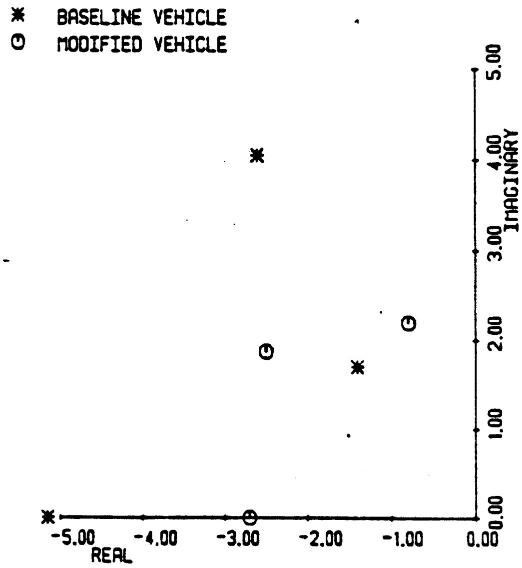

Fig. 9. Characteristic roots of the baseline and modified closed-loop systems.

These results and those of the previous section demonstrate useful application of the optimal preview model in simulation of closed-loop automobile driving. The principal conclusion concerning these results is that driver steering control strategy during path following can be accurately represented as a time-lagged optimal preview control. Similar applications and extensions to problems in other fields are clearly suggested by the results shown here.

Conclusion

The optimal preview control model, applied here to the closed-loop automobile path following problem, offers a useful and direct method for representing closed-loop behavior of linear driver/vehicle systems. It is suggested that driver automobile steering control strategy during path following can be viewed as a time-lagged optimal preview control process.

The general linear system formulation of the preview control methodology, demonstrated here, permits application to a broad range of problems relating to man-machine systems.

Appendix I

A. Vehicle Dynamics

The linear dynamical equations of an automobile for lateral and yaw motions are

$$\dot{y} = v + U\psi \tag{A1}$$

$$\dot{v} = \left[-2(C_{\alpha_F} + C_{\alpha_R})/mU\right]v + \left[2(bC_{\alpha_R} - aC_{\alpha_F})/mU - U\right]r + (2C_{\alpha_F}/m)\delta_{FW} \tag{A2}$$

$$\dot{r} = \left[2(bC_{\alpha_R} - aC_{\alpha_F})/IU\right]v + \left[-2(a^2C_{\alpha_F} + b^2C_{\alpha_R})/IU\right]r + (2aC_{\alpha_F}/I)\delta_{FW} \tag{A3}$$

$$\dot{\psi} = r \tag{A4}$$

where

- y inertial lateral displacement of the vehicle mass center,
- v lateral velocity in the vehicle body axis system,
- r yaw rate about the vertical body axis,
- ψ vehicle heading angle, and
- δ_{FW} front tire steer angle, control variable.

The parameters appearing in (A1)–(A4) are

- U forward vehicle velocity,
- $C_{\alpha_F}, C_{\alpha_R}$ front and rear tire cornering coefficients,
- a, b forward and rearward locations of tires from the vehicle mass center, and
- m, I vehicle mass and rotational inertia.

The above equations can be expressed in matrix notation as

$$\dot{x} = Fx + g\delta_{FW} \tag{A5}$$

where

$$x = \begin{Bmatrix} y \\ v \\ r \\ \psi \end{Bmatrix}$$

$$F = \begin{bmatrix} 0 & 1 & 0 & U \\ 0 & A_1 & B_1 & 0 \\ 0 & A_2 & B_2 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \qquad g = \begin{Bmatrix} 0 \\ C_1 \\ C_2 \\ 0 \end{Bmatrix}$$

and

$$A_1 = -2(C_{\alpha_F} + C_{\alpha_R})/mU$$
$$B_1 = 2(bC_{\alpha_R} - aC_{\alpha_F})/mU - U$$
$$C_1 = 2C_{\alpha_F}/m$$
$$A_2 = 2(bC_{\alpha_R} - aC_{\alpha_F})/IU$$
$$B_2 = -2(a^2C_{\alpha_F} + b^2C_{\alpha_R})/IU$$
$$C_2 = 2aC_{\alpha_F}/I.$$

The calculation of (11) appearing in Figs. 3 and 4 used the following parameter values identified in [9] for vehicle D

- a = 1.41 m (4.63 ft)
- b = 1.41 m (4.63 ft)
- m = 2016 kg (138 slug)
- I = 4013 m·N·s² (2960 ft·lb·s²)
- U = 22.3 m/s (73.3 ft/s)
- C_{α_F} = 25 266 N/rad (5 680 lb/rad)
- C_{α_R} = 70 933 N/rad (15 960 lb/rad).

The constant observer vector $m^T = (1,0,0,0)$ provided the vehicle lateral position y.

B. Baseline Vehicle Parameter Values

The vehicle parameter values listed below and used in the calculations appearing in Fig. 6 were derived from vehicle wheelbase/weight measurements and steady-state, constant-steer vehicle test results [13]

- a = 1.37 m (4.5 ft)
- b = 1.22 m (4.0 ft)
- m = 1563 kg (107 slug)
- I = 2712 m·N·s² (2 000 ft·lb·s²)
- U = 25.9 m/s (85 ft/s)
- C_{α_F} = 19 438 N/rad (4 370 lb/rad)
- C_{α_R} = 33 628 N/rad (7 560 lb/rad).

The weighting function W appearing in (7) was selected as constant 1.0 over the ten-point preview interval.

C. Modified Vehicle Parameter Values

The vehicle parameters of Appendix I-B were altered to those values shown in this section by a rearward shift in the vehicle mass center and a decrease in rear tire inflation pressures

$a = 1.43$ m (4.7 ft)
$b = 1.16$ m (3.8 ft)
$m = 1753$ kg (120 slug)
$I = 2712\ \mathrm{m \cdot N \cdot s^2}$ ($2000\ \mathrm{ft \cdot lb \cdot s^2}$)
$U = 25.9$ m/s (85 ft/s)
$C_{\alpha_F} = 20\,906$ N/rad (4700 lb/rad)
$C_{\alpha_R} = 29\,536$ N/rad (6640 lb/rad).

The closed-loop calculation using these parameter values appears in Fig. 7.

D. Stability of the Closed-Loop Optimal Preview-Controlled System Including a Transport Time Lag

Given the system

$$\dot{x} = Fx + gu \tag{A6}$$

$$u = e^{-s\tau} u^0 \tag{A7}$$

$$u^0 = -c^T x \tag{A8}$$

where F, g, u^0, and c^T are defined in (1), (4), and (9). If the transport time lag $e^{-s\tau}$ is approximated by the first-order Padé polynomial,

$$\frac{1 - \frac{\tau}{2}s}{1 + \frac{\tau}{2}s}, \tag{A9}$$

(A7) becomes

$$\dot{u} = \frac{2}{\tau}(-u + u^0) - \dot{u}^0. \tag{A10}$$

Substitution of

$$u^0 = -c^T x$$

and

$$\dot{u}^0 = -c^T[Fx + gu]$$

into (A10) produces the closed-loop state equation

$$\left\{ \frac{\dot{x}}{\dot{u}} \right\} = \left[\begin{array}{c|c} F & g \\ \hline c^T\left(F - \frac{2}{\tau}I\right) & c^T g - \frac{2}{\tau} \end{array} \right] \left\{ \frac{x}{u} \right\} \tag{A11}$$

equivalent of (A6)–(A8). For small τ, stability of the time-lagged optimal preview-controlled system is provided by the characteristic roots of the system matrix appearing in (A11).

REFERENCES

[1] C. C. MacAdam, "An optimal preview control for linear systems," *J. Dynamic Systems, Measurement, Control*, Sept., 1980.

[2] D. T. McRuer, *et al.*, "New approaches to human-pilot/vehicle analysis," Systems Technology, Inc., Tech. Rep., AFFDL-TR-67-150, Feb. 1968.

[3] W. W. Wierwille, G. A. Gagne, and J. R. Knight, "An experimental study of human operator models and closed-loop analysis methods for high-speed automobile driving," *IEEE Trans. Hum. Factors Electron.*, vol. HFE-8, no. 3, pp. 187–201, Sept. 1967.

[4] K. Tanaka, N. Goto, and K. Washizu, "A comparison of techniques for identifying human operator dynamics utilizing time series analysis," in *Proc. Twelfth Annu. Conf. Manual Control*, Univ. of Illinois, Urbana, IL, May 25–27, 1976, pp. 673–693.

[5] D. H. Weir, "Motorcycle handling dynamics and rider control and the effect of design configuration on response and performance," Ph.D. dissertation, Univer. of California, Los Angeles, CA 1972.

[6] S. Ben-Ari and J. R. Ellis, "The control of an articulated semitrailer vehicle," in *Vehicle Safety Legislation—Its Engineering and Social Implications*. London: Mechanical Engineering Publications Limited, 1975.

[7] D. L. Kleinman, S. Baron, and W. H. Levison, "An optimal control model of human response, part I: Theory and validation," *Automatica*, vol. 6, pp. 357–369, 1970.

[8] D. T. McRuer *et al.*, "New results in driver steering control models," *Human Factors*, vol. 19, pp. 381–397, Aug. 1977.

[9] D. H. Weir, R. J. DiMarco, and D. T. McRuer, "Evaluation and correlation of driver/vehicle data," vol. II, Final Tech. Rep., National Highway Traffic Safety Admin., DOT-HS-803-246, Apr. 1977.

[10] R. G. Mortimer and C. M. Jorgeson, "Eye fixations of drivers as affected by highway and traffic characteristics and moderate doses of alcohol," in *Proc. Sixteenth Annu. Meeting, Human Factors Society*, Oct. 17–19, 1972, pp. 86–92.

[11] M. Kondo and A. Ajimine, "Driver's sight point and dynamics of the driver-vehicle-system related to it," SAE Paper No. 680104, Automotive Engineering Congress, Detroit, MI, Jan. 8–12, 1968.

[12] D. A. Gordon, "Experimental isolation of drivers' visual input," *Public Roads*, vol. 33, pp. 266–273, 1966.

[13] "Comparison of vehicle test procedures," Contract DRDA 781433, Highway Safety Research Institute, University of Michigan, 1978.

Journal of Dynamic Systems, Measurement and Control

Technical Briefs

An Optimal Preview Control for Linear Systems

C. C. MacAdam[1]

A technique for synthesizing closed-loop control of linear time-invariant systems during tracking of previewed inputs is presented. The derived control is directly dependent upon the properties of the controlled system and is obtained by minimization of a defined previewed output error.

I Introduction

This paper presents a general method of control synthesis applicable to linear time-invariant systems utilizing preview control strategies for regulation or tracking tasks. A common example of this type of dynamical behavior occurs during normal automobile path following in which drivers "look-ahead" to follow a desired path. A frequent source of preview control strategies in various man-machine systems is, of course, the human operator. It is widely recognized that human operators are capable of controlling and adapting to a wide variety of dynamical systems, many of which are vehicles with preview-oriented control requirements such as automobiles, bicycles, and complex aircraft [1-7]. Although this paper does not offer evidence as to the utility of the proposed control synthesis for man-machine systems involving preview strategies, it is suggested that the method presented here can be applied to such problems. Portions of the work by Tomizuka [8], which treated a similar problem, indicated useful application of optimal preview control methods in representing man-machine dynamical behavior.

The particular method presented in this paper is directly applicable to general linear system representations assumed to incorporate preview control strategies that depend only upon knowledge of the current values of the state and control. The optimal control is derived by minimization of a performance index that is defined as a mean squared preview output error. It will be shown that the derived control function is not arbitrary or independent but depends directly upon the dynamical properties of the controlled system.

II Statement of the Problem

Given the linear system

$$\dot{\mathbf{x}} = F\mathbf{x} + \mathbf{g}u \tag{1}$$

$$y = \mathbf{m}^T\mathbf{x} \tag{2}$$

[1] Research Associate, University of Michigan, Highway Safety Research Institute, Ann Arbor, Mich. 48109

Contributed by the Dynamic Systems and Control Division of THE AMERICAN SOCIETY OF MECHANICAL ENGINEERS. Manuscript received at ASME Headquarters, July 9, 1980.

where,

$\mathbf{x}$ is the $n \times 1$ state vector

y is the scalar output related to the state by the $n \times 1$ $\mathbf{m}^T$ constant observer vector transpose

F is the constant $n \times n$ system matrix

and

$\mathbf{g}$ is the constant $n \times 1$ control coefficient vector

find the control, $u(t)$, which minimizes a local performance index,

$$J \triangleq \frac{1}{T}\int_t^{t+T} \{[f(\eta) - y(\eta)]W(\eta - t)\}^2 d\eta \tag{3}$$

over the current preview interval $(t, t+T)$, where,

W is an arbitrary weighting function over the preview interval

and f is the previewed input.

The performance index given by (3) represents the weighted mean squared error between the previewed input and the previewed output as defined below.

The previewed output, $y(\eta)$, is related to the present state, $\mathbf{x}(t)$, by

$$y(\eta) = \mathbf{m}^T\phi(\eta,t)\mathbf{x}(t) + \int_t^{\eta} \mathbf{m}^T\phi(\eta,\xi)\mathbf{g}u(\xi)d\xi \tag{4}$$

where,

$$\phi(\eta,t) = \exp[F(\eta - t)]$$

is the transition matrix of the system F [9].

If $u(t)$ is assumed selected on the basis of a constant previewed control, $u(\xi) = u(t)$, equation (4) simplifies to

$$y(\eta) = \mathbf{m}^T\phi(\eta,t)\mathbf{x}(t) + u(t)\int_t^{\eta} \mathbf{m}^T\phi(\eta,\xi)\mathbf{g}d\xi \tag{5}$$

and the performance index, (3), can be written as

$$J = \frac{1}{T}\int_t^{t+T} \left\{\left[f(\eta) - \mathbf{m}^T\phi(\eta,t)\mathbf{x}(t) - u(t)\int_t^{\eta} \mathbf{m}^T\phi(\eta,\xi)\mathbf{g}d\xi\right]^2 \cdot W(\eta - t)\right\} d\eta \tag{6}$$

The above assumption simply requires the resulting optimization to reflect a control strategy dependent only upon current values of the state and control. This assumption is, in part, motivated by the potential application to those man-machine systems, wherein, it is assumed the human operator is limited in deriving or having knowledge a priori of more complex or optimal control waveforms over the preview interval.

The necessary condition for minimization of J, defined by uation (6), with respect to the control, $u(t)$, is provided by $'/du = 0$, or

$$\frac{dJ}{\ _{l}} = \frac{2}{T}\int_{t}^{t+T}\Bigg\{\Big[f(\eta)-\mathbf{m}^{T}\phi(\eta,t)\mathbf{x}(t)$$

$$-u(t)\int_{t}^{\eta}\mathbf{m}^{T}\phi(\eta,\xi)\mathbf{g}d\xi\Big]\Bigg\}$$

$$\cdot\left\{\int_{t}^{\eta}\mathbf{m}^{T}\phi(\eta,\xi)\mathbf{g}d\xi\right\}W(\eta-t)d\eta=0 \tag{7}$$

Equating $\phi(\eta,\xi)$ with $\exp[F(\eta-\xi)] = I + \sum_{n=1}^{\infty} F^{n}\frac{(\eta-\xi)^{n}}{n!}$,

ıere I is the identity matrix, and performing the $d\xi$ integrations, (7) becomes

$$\frac{J}{\ _{l}} = \frac{2}{T}\int_{t}^{t+T}\Bigg\{f(\eta)-\mathbf{m}^{T}\Big[I+\sum_{n=1}^{\infty}\frac{F^{n}(\eta-t)^{n}}{n!}\Big]\mathbf{x}(t)$$

$$(\eta-t)\mathbf{m}^{T}\Big[I+\sum_{n=1}^{\infty}\frac{F^{n}(\eta-t)^{n}}{(n+1)!}\Big]\mathbf{g}u(t)\Bigg\}\Bigg\{(\eta-t)\mathbf{m}^{T}$$

$$\cdot\Big[I+\sum_{n=1}^{\infty}\frac{F^{n}(\eta-t)^{n}}{(n+1)!}\Big]\mathbf{g}\Bigg\}W(\eta-t)d\eta=0 \tag{8}$$

ˈlving (8) for $u(t)$ yields

$$u^{0}(t)=\Bigg[\int_{t}^{t+T}\Bigg\{f(\eta)-\mathbf{m}^{T}\Big[I+\sum_{n=1}^{\infty}\frac{F^{n}(\eta-t)^{n}}{n!}\Big]\mathbf{x}(t)\Bigg\}$$

$$\Bigg\{(\eta-t)\mathbf{m}^{T}\Big[I+\sum_{n=1}^{\infty}\frac{F^{n}(\eta-t)^{n}}{(n+1)!}\Big]\mathbf{g}\Bigg\}W(\eta-t)d\eta\Bigg]$$

$$/\Bigg[\int_{t}^{t+T}\Bigg\{(\eta-t)\mathbf{m}^{T}\Big[I+\sum_{n=1}^{\infty}\frac{F^{n}(\eta-t)^{n}}{(n+1)!}\Big]\mathbf{g}\Bigg\}^{2}W(\eta-t)d\eta\Bigg] \tag{9}$$

ᐯ..ıere $u^{0}(t)$ represents the optimal solution. For the special case of $W(\eta-t) = \delta(T^{*})$, the Dirac delta function for $0 \le T^{*} \le T$, (9) simplifies to

$$u^{0}(t)=\frac{f(t+T^{*})-\mathbf{m}^{T}\Big[I+\sum_{n=1}^{\infty}\frac{F^{n}(T^{*})^{n}}{n!}\Big]\mathbf{x}(t)}{T^{*}\mathbf{m}^{T}\Big[I+\sum_{n=1}^{\infty}\frac{F^{n}(T^{*})^{n}}{(n+1)!}\Big]\mathbf{g}} \tag{10}$$

$$=[f(t+T^{*})-y_{0}(t+T^{*})]/(T^{*}K) \tag{11}$$

ıere

$$K \triangleq \mathbf{m}^{T}\Big[I+\sum_{n=1}^{\infty}\frac{F^{n}(T^{*})^{n}}{(n+1)!}\Big]\mathbf{g}.$$

Equation (11) represents a proportional controller with gain inversely related to the preview interval, T^{*}, and operating on the error between the previewed input, $f(t+T^{*})$, and $y_{0}(t+T^{*})$, that portion of the previewed output deriving from the state vector's current initial condition. Likewise, equation (9) can be interpreted as a proportional controller operating on a similar error averaged and weighted over the preview interval $(t, t+T)$ by the additional terms appearing in equation (9).

The optimal solution, $u^{0}(t)$, can also be expressed in terms of any current non-optimal $u(t)$ and correspondingly nonzero preview output error, $\epsilon(t)$, by writing equation (9) as

$$u^{0}(t)=\Bigg[\int_{t}^{t+T}\Big\{f(\eta)-\mathbf{m}^{T}\phi(\eta,t)\mathbf{x}(t)-u(t)A(\eta)\Big\}$$

$$\cdot A(\eta)W(\eta-t)d\eta+u(t)\int_{t}^{t+T}A^{2}(\eta)W(\eta-t)d\eta\Bigg]$$

$$/\Bigg[\int_{t}^{t+T}A^{2}(\eta)W(\eta-t)d\eta\Bigg] \tag{12}$$

or

$$u^{0}(t)=u(t)+\frac{\int_{t}^{t+T}\epsilon(\eta)A(\eta)W(\eta-t)d\eta}{\int_{t}^{t+T}A^{2}(\eta)W(\eta-t)d\eta} \tag{13}$$

where

$$A(\eta)\triangleq(\eta-t)\mathbf{m}^{T}\Big[I+\sum_{n=1}^{\infty}\frac{F^{n}(\eta-t)^{n}}{(n+1)!}\Big]\mathbf{g}$$

$$\epsilon(\eta)\triangleq f(\eta)-\mathbf{m}^{T}\phi(\eta,t)\mathbf{x}(t)-u(t)A(\eta)$$

$$\phi(\eta,t)\triangleq \mathrm{I}+\sum_{n=1}^{\infty}\frac{F^{n}(\eta-t)^{n}}{n!}$$

For the special case of $W(\eta-t) = \delta(T^{*})$, as before, equation (13) reduces to

$$u^{0}(t)=u(t)+\frac{\epsilon(t+T^{*})}{T^{*}\cdot K} \tag{14}$$

The formulation expressed by equation (13) can be useful in describing systems which do not achieve, though closely approximate, the optimal system behavior. Such cases may arise from limitations in achieving the precise optimal control due to time lags or dynamic properties inherent in the controller and not accounted for *a priori* in the optimization.

While equations (9) and (13) are equivalent mathematically, the latter deomonstrates an explicit relationship between the derived optimal control and the previewed output error function appearing in the performance index of the original problem formulation. Simply stated, the current control level is modified only in response to a nonzero function of the previewed output error, and, in this sense, analogous to an integral controller.

Finally, dependence of the derived optimal control upon the system $(F, \mathbf{g})$ properties is clearly demonstrated by the explicit presence of F and $\mathbf{g}$ in equations (9) and (13). Furthermore, information concerning stability of the closed-loop system utilizing the optimal preview control of equation (9) or (13) is provided by the characteristic roots of the constant matrix

$$F-\frac{\mathbf{g}\mathbf{m}^{T}\int_{0}^{T}\Big\{\Big[I+\sum_{n=1}^{\infty}\frac{F^{n}(\eta)^{n}}{n!}\Big]\Big\}\Big\{\eta\mathbf{m}^{T}\Big[I+\sum_{n=1}^{\infty}\frac{F^{n}(\eta)^{n}}{(n+1)!}\Big]\mathbf{g}\Big\}W(\eta)d\eta}{\int_{0}^{T}\Big\{\eta\mathbf{m}^{T}\Big[I+\sum_{n=1}^{\infty}\frac{F^{n}(\eta)^{n}}{(n+1)!}\Big]\mathbf{g}\Big\}^{2}W(\eta)d\eta}$$

or

$$[F-\mathbf{g}\mathbf{c}^T] \tag{15}$$

where

$$\mathbf{c}^T=\frac{\mathbf{m}^T\int_0^T\left\{\left[I+\sum_{n=1}^{\infty}\frac{F^n(\eta)^n}{n!}\right]\right\}\left\{\eta\mathbf{m}^T\left[I+\sum_{n=1}^{\infty}\frac{F^n(\eta)^n}{(n+1)!}\right]\mathbf{g}\right\}W(\eta)\,d\eta}{\int_0^T\left\{\eta\mathbf{m}^T\left[I+\sum_{n=1}^{\infty}\frac{F^n(\eta)^n}{(n+1)!}\right]\mathbf{g}\right\}^2W(\eta)\,d\eta}$$

resulting from the substitution of (9) into (1). For the special case of $W(\eta)=\delta(T^*)$, (15) becomes

$$F-\left\{\mathbf{g}\mathbf{m}^T\left[I+\sum_{n=1}^{\infty}\frac{F^n(T^*)^n}{n!}\right]/(T^*\cdot K)\right\} \tag{16}$$

III Summary

The optimal preview control model presented here offers a useful and direct method for representing closed-loop behavior of linear systems utilizing preview control strategies. The derived control is directly related to the properties of the linear system and the previewed input. Further, the method is formulated in terms of general linear system representations, thereby permitting applications to a wide variety of problems.

References

1 McRuer, D.T., et al., "New Approaches to Human-Pilot/Vehicle Analysis," Systems Technology, Inc., Tech. Rept. AFFDL-TR-67-150, Feb. 1968.

2 Wierwille, W.W., Gagne, G.A., and Knight, J.R., "An Experimental Study of Human Operator Models and Closed-Loop Analysis Methods for High-Speed Automobile Driving," *IEEE Trans. on Human Factors in Electronics*, Vol. HFE-8, No. 3, Sept. 1967, pp. 187-201.

3 Tanaka, K., Goto, N., and Washizu, K., "A Comparison of Techniques for Identifying Human Operator Dynamics Utilizing Time Series Analysis," *Proceedings of the Twelfth Annual Conference on Manual Control*, University of Illinois, Urbana, Ill., May 25-27, 1976, pp. 673-693.

4 Weir, D.H., "Motorcycle Handling Dynamics and Rider Control and the Effect of Design Configuration on Response and Performance," Ph.D. thesis, University of California, Los Angeles, 1972.

5 Ben-Ari, S., and Ellis, J.R., "The Control of an Articulated Semitrailer Vehicle," *Vehicle Safety Legislation—Its Engineering and Social Implications*, Mechanical Engineering Publications Limited, London, 1975.

6 Kleinman, D.L., Baron, S., and Levison, W.H., "An Optimal Control Model of Human Response, Part I: Theory and Validation," *Automatica*, Vol. 6, 1970, pp. 357-369.

7 McRuer, D.T., et al., "New Results in Driver Steering Control Models," *Human Factors*, Vol. 19, Aug. 1977, pp. 381-397.

8 Tomizuka, M., "The Optimal Finite Preview Problem and Its Application to Man-Machine Systems," Dissertation, MIT, Cambridge, Mass., Sept., 1973.

9 D'Angelo, H., *Linear Time-Varying Systems: Analysis and Synthesis*, Allyn and Bacon, Boston, 1970.

Asymptotic Theory of Freight Car Hunting

A. M. Whitman[1]

A simple formula is derived for the hunting speed of a freight car from an 8 degree of freedom linear model using asymptotic techniques. A comparison is made between the approximation and exact (numerical) solutions. The two agree within 10 percent for parameter values typical of present designs.

[1] Director of Research, Railroad Dynamics, Inc., Ardmore, Pa.

Present Address: Associate Professor, University of Pennsylvania, Philadelphia, Pa.

Contributed by the Dynamic Systems and Control Division of THE AMERICAN SOCIETY OF MECHANICAL ENGINEERS. Manuscript received at ASME Headquarters, July 9, 1980.

Introduction

The purpose of the present paper is twofold. The first is to obtain an analytic expression for the critical speed of a multidegree of freedom model of a freight car which is simple enough to convey physical insight into the hunting problem while at the same time complex enough to have validity for realistic vehicles. The second is to illustrate the simplification which can be effected in problems of this type by employing asymptotic methods. These methods are model independent and rely on the fact that the creep forces dominate the motion.

Previous work has included analytical studies of simple vehicles [1-2] and numerical solutions for realistic vehicles [3-4]. The present work can be viewed as a generalization and formal mathematical justification of the former, which although cleverly done are ad hoc by nature and seem to be restricted to systems with few degrees of freedom, and a specialization of the latter, giving the same results in the region of validity of the expansion but being restricted by nature to specific regions in parameter space. The utility of the present work is in the simple result which it yields. From this one can obtain physical insight into the phenomenon as well as easily calculable answers.

Model Description

We consider a model of the lateral dynamics of a freight car composed of a rigid car body pinned at either end to a truck. The pin connection transmits a linear damping moment (constant c'_f) between the car body and the truck. Each truck, see Fig. 1, is composed of 2 wheelsets, two rigid sideframes connected by ball joints to each wheelset, and a bolster, which contains the car connection (centerplate) at its midpoint, is constrained to move parallel to each wheelset by means of frictionless slotted pins in each sideframe, and is restrained from moving freely in that direction by 2 linear springs (constant k each) and dampers (constant c each) at each end. In the real system this restraint is provided by the shear stiffness of the bolster springs, whose primary function is to support the car weight, and the sliding of the friction wedges laterally. Further, because the springs and dampers are separated by a distance d, there is a moment tending to square the truck due to both the springs (constant $4kd^2$) and the dampers constant $4cd^2$). In addition, the bolster has mounted symmetrically with respect to the centerplate, constant contact sidebearings (constant $\bar{k}_B$ each) whose function is to provide a torsional spring restraint for the bolster relative to the car body (constant $2\bar{k}_B w^2$). Actually the sidebearings also transmit a damping moment between the bolster and the car body (constant $2c_B w^2$); however, this has the same form as the centerplate moment and can be combined with it. There are eight degrees of freedom in this model and we will take as our independent coordinates x^F, ψ^F, β^F, u^F, x^R, ψ^R, β^R, u^R. Here the superscripts represent the front and rear truck coordinates, x is the axial displacement of the truck centroid relative to the track center line, ψ the yaw angle of each wheelset of the truck as a result of the kinematic constraint, β the trail angle of the truck, and u the bolster displacement relative to the truck centroid. The equations of motion, which have been derived elsewhere [5] and which are quite similar to others which have been discussed in the literature [4], are written here in dimensionless form in terms of sum and difference coordinates,

APPENDIX B — SOURCE CODE

This appendix lists the Fortran source code for the Wind-Steer model. Variables in common blocks are defined in separate "include" files, which are listed separately from the program subroutines at the end of the appendix.

Main Program Listing

```
C
C       CHRYSLER/UMTRI VEHICLE CROSSWIND STABILITY PROJECT
C
C 5-D.O.F. VEHICLE + 2-D.O.F. STEERING SYSTEM + CLOSED-LOOP DRIVER MODEL
C
C                   VERSION 1.4 - APRIL 1990
C
C  (c) The Regents of The University of Michigan, All Rights Reserved
C
C  Written by Yoram Guy, 6-30-87                        (Phase 1;  v 0.70)
C  Modified by M. Sayers, 4-26-88    (mainframe to PC versions; v 0.80)
C  Modified by C. MacAdam, 5-19-88     (driver model installed; v 0.83)
C  Modified by M. Sayers, 8-28-88 (changed eqs. of motion,
C                                             new integrator; v 0.85)
C  Modified by C. MacAdam, 9-7-88 (driver model and wind profile
C                                                 additions;  v 0.90)
C  Modified by M. Sayers,12-14-88    (cosmetics, changed input; v 0.91)
C
C  Modified by C. MacAdam,1-30-89  (steering system, revised tire eqns
C   and params for SAE conventions, torque-option driver model;  v 1.0)
C
C  Modified by C. MacAdam, 5-30-89  (static wind sensitivity eqns and
C   summary output added to echo file;  random crosswind input;
C   additional output plot variables: e.g., aver workload, etc.; v 1.3)
C
C  Modified by M. Sayers, 4-10-90         (large angle tire eqs.; v1.4)
C======================================================================
C
C  MACHINE DEPENDENCIES:
C  ---------------------
C  Most of the following code is standard Fortran 77 and is independent
C  of the implementation, EXCEPT:
C
C      (1) "include" files are not standard and must be referenced
C            as needed for a specific compiler.
C      (2) The terminal is referenced as unit * in READ and WRITE
C           statements involving the user.  (Although not "standard,"
C           this works with most compilers and probably is OK.)
C
C  Otherwise, all machine-specific sections of code are identified by
```

```
C  comments that begin with "C--".  This file includes the code
C  needed for
C
C      (1) the Microsoft Fortran compiler for the IBM PC
C      (2) the Absoft Fortran compiler for the Apple Macintosh
C      (3) the FortranVS compiler for the UM mainframe (MTS) system
C
C=====================================================================
C
C  PROGRAM SECTIONS:
C  -----------------
C  MAIN -- Controls "flow" of program and performs num. integration
C
C  AIRACT(T, YAW, BETA, VYAW) -- handle aerodynamic forces and moments
C  BEAM(ALPH, DFDX, FNEG) -- equations for beam suspension
C  BLOCK DATA -- initializes variables in COMMON blocks
C  Function DFYDA (ALPHA, GAMMA, FZ, AXLE) -- partial deriv: dFy/dAlpha
C  Function DFYDG(ALPHA, GAMMA, FZ, AXLE) -- partial deriv: dFy/dGamma
C  Function DMZDA(ALPHA, GAMMA, FZ, AXLE) -- partial deriv: dMz/dAlpha
C  DRIVE1(DFW) -- Read driver model parameters
C  DRIVER(X, Y, DFW, DFWNOW) -- compute closed-loop steer angle
C  DRIVET(X, Y, DRTORQ, DRTNOW) -- compute closed-loop steer torque
C  DRIVGO -- initialize driver model (steer-angle version)
C  DRIVGT -- initialize driver model (steer-torque version)
C  ECHO -- create output file with echo of input parameters
C  FDAMP(VZ, VROLL, VPITCH, FD) -- compute damping force for 4 wheels
C  FUNCTN(T,Y,YP) -- computes YP derivatives given T and Y
C  Function FWIND(T) -- provide cross-wind as function of time
C  GMPRD(A, B, R, N, M, L) -- multiply two matrices
C  INDATA -- read input data and converts units
C  INDSUS(X, DFDX, FNEG) -- equations for independent suspension
C  INIT -- computes constants used in simulation
C  Function LENSTR(STRING) -- no. of characters in string
C  MNEWT(X,USRFUN,IERR) -- algebraic equation solver
C  OPNOUT -- create output file and write header
C  OUTPUT(T, Y, YP) -- write simulation variables into file at time T
C  Function POLY4(COEF, FZ) -- evaluate 4th-order polynomial of Fz
C  ROLLAX(ROLL, YROLAX, HROLAX, IXSRA) -- roll axis kinematics
C  Function STEER(T) -- provides steering wheel angle as function of T
C  Function SUM(MATRIX) -- sums 4 elements of matrix
C  TABLE(M, N, X, Y, Z, Q) -- table look-up routine.
C  TIMDAT(TIMEDT) -- produce string with time and date
C  Function TIREFY(ALPHA, GAMMA, FZ, AXLE) -- compute Fy for tire
C  Function TIREMZ(ALPHA, FZ, AXLE) -- compute Mz for tire
C  TIRES(BETA, V, VYAW, ROLL) -- compute slip, camber, forces for tires
C  TRAJ(X, XT, YT, YPATH)  -- compute lat. disp. of previewed path
C  TRANS -- Compute transition matrix for driver model (angle version)
C  TRANST -- Compute transition matrix for driver model (torque version)
C  WHEELZ(Z, ROLL, PITCH) -- handle wheel kinematics
C
C=====================================================================
C
C  LIST OF SYMBOLS:
C  ----------------
C
C  I/O SYMBOLS
C
C     IREAD - unit number for input data
```

```
C     IECHO - unit number for output file with echo of input data
C     IOUT  - unit number for simulation output file
C
C
C  SIMULATION PARAMETERS
C
C     DT      -  time step for numerical integration
C     TEND    -  end time of simulation
C     IPRINT  -  print interval (every i-th point is save in output file)
C     KSYWND  -  wind heading angle (of velocity vector)
C     AIRHO      air density
C     V       -  vehicle speed
C     VWIND   -  wind speed
C     WINDKY  -  wind key: >= 0  => time history wind profile input:
C                                   windky is num of (T,VW) table pairs.
C
C                          < 0  => call user function "FWIND" for
C                                   profile input.
C
C  GLOBAL TIME-VARIABLES
C
C     T          -  time
C     Y(13)      -  array of 13 state variables
C     AY         -  vehicle lateral acceleration (ignoring roll-accel.)
C     RHO        -  path curvature
C     BETAIR     -  aerodynamic sideslip angle
C     VAX,VAY    -  x,y components of air velocity (axles reference)
C
C  Position  Speed   Accel.
C  --------  -----   ------
C     XG,     VXG                -  X of total cg (inertial reference)
C     YG,     VYG                -  Y of total cg (inertial reference)
C     BETA,   VBETA              -  ground sideslip (BETA = VY / V)
C     Z,      VZ,     AZ         -  sprung mass cg vertical
C     ROLL,   VROLL,  AROLL      -  roll
C     PITCH,  VPITCH, APITCH     -  pitch
C     YAW,    VYAW,   AYAW       -  yaw
C     SW,     VSW,    ASW        -  steering wheel angle
C     FW                         -  front wheel steer angle
C
C  GLOBAL INTEGERS  -  INDICES AND FLAGS
C
C     UNITS  -  (CHAR*1) 'E' = English (ft, lbm, deg), otherwise metric
C     NUMKEY -  1 = binary MAC, 2 = BINARY PC, 5 = text output
C     NAXLE  -  1 = front, 2 = rear
C     NSIDE  -  1 = left, 2 = right
C     NSTEER -  >0  ---> steer table    (no. of T,SW pairs)
C            -  =0  ---> sine function  (harmonic SW)
C            -  <0  ---> driver model   (-no. of XPNT,YPNT pairs)
C
C  VEHICLE AERODYNAMIC, STEERING-SYSTEM, AND GENERAL PARAMETERS
C
C     AREA   -  VEHICLE CROSS-SECTION AREA (IN Y-Z PLANE)
C     QZERO  -  DENSITY * AREA / 2
C     KY     -  AERODYNAMIC SIDE FORCE COEFFICIENT
C     CL0,KL -  AERODYNAMIC DOWN FORCE (-LIFT) COEFFICIENTS
C     KR     -  AERODYNAMIC ROLL MOMENT COEFFICIENT
C     CM0,KM -  AERODYNAMIC PITCH MOMENT COEFFICIENTS
```

```
C     KN      -  AERODYNAMIC YAW MOMENT COEFFICIENT
C     CD0,KD  -  AERODYNAMIC DRAG FORCE COEFFICIENTS
C
C     CBOOST  -  STEERING POWER-BOOST COEFFICIENT
C     CFSS    -  STEERING SYSTEM COULOMB FRICTION MOMENT
C     GR      -  STEERING-SYSTEM OVERALL KINEMATIC RATIO
C     GRTODG  -  GR (ABOVE) * TODEG
C     ISS     -  STEERING-SYSTEM MOMENT OF INERTIA - LUMPED AT STEER-WHL
C     KSC     -  STEERING-COLUMN STIFFNESS
C     KSL     -  STEERING LINKAGE STIFFNESS
C     SSKEY   -  STEERING-SYSTEM KEY TRIGGERING USE OF DYN ST SYS MODEL
C     XTRAIL  -  FRONT WHEEL MECHANICAL TRAIL
C
C     HCGTTL  -  TOTAL STATIC CG HEIGHT ABOVE GROUND
C     HCGSP   -  STATIC SPRUNG-MASS CG HEIGHT ABOVE GROUND
C     XCGSP   -  STATIC SPRUNG-MASS CG DISTANCE FROM FRONT AXLE
C     XWBCGS  -  SPRUNG-MASS CG DISTANCE AHEAD OF HALF-WHEELBASE POINT
C     XWBCGT  -  TOTAL CG DISTANCE AHEAD OF HALF-WHEELBASE POINT
C     WHLRAD  -  TIRE ROLLING RADIUS = ASSUMED UNSPRUNG-MASS CG HEIGHT
C     IXSCG   -  SPRUNG-MASS X-X MOMENT OF INERTIA (X-X THRU SPRUNG CG)
C     IXSRA   -  SPRUNG-MASS MOMENT OF INERTIA ABOUT ROLL AXIS
C     IXZ     -  SPRUNG-MASS XZ PRODUCT OF INERTIA
C     IYS     -  SPRUNG-MASS PITCH MOMENT OF INERTIA
C     IZZ     -  TOTAL YAW MOMENT OF INERTIA
C     KROLL   -  TOTAL ROLL STIFFNESS
C     MASS    -  TOTAL MASS
C     SPMASS  -  SPRUNG MASS
C     SPWGHT  -  SPRUNG WEIGHT
C     WEIGHT  -  TOTAL WEIGHT
C     USWGHT  -  UNSPRUNG WEIGHT
C     WRATIO  -  FRONT-AXLE NORMAL (GROUND) LOAD FRACTION OF TOTAL WEIGHT
C     WB      -  WHEELBASE
C     CSROLL  -  REAR BEAM-AXLE ROLL-STEER COEFFICIENT
C
C  PER-AXLE PARAMETERS  -  INDEXED (AXLE)
C
C     XAXLE   -  DISTANCE FROM TOTAL CG TO AXLE (NEGATIVE FOR REAR)
C     TRACK   -  NOMINAL TRACK WIDTH (ASSUMED CONSTANT)
C     H0ROLC  -  STATIC ROLL CENTER HEIGHT ABOVE GROUND
C     HCGSRC  -  STATIC ROLL CENTER HEIGHT BELOW SPRUNG CG (<0 IF ABOVE)
C     FZ0WHL  -  TIRE/ROAD STATIC NORMAL LOAD AT EACH WHEEL
C     KZ      -  SUSPENSION VERTICAL (RIDE) STIFFNESS AT EACH WHEEL
C     KZAXLE  -  SUSPENSION VERTICAL (RIDE) STIFFNESS (2 X KZ)
C     KAUX    -  SUSPENSION AUXILIARY ROLL STIFFNESS
C     KTIRE   -  TIRE VERTICAL STIFFNESS
C     CZJNCE  -  DAMPING COEFFICIENT IN JOUNCE AT EACH WHEEL
C     CZRBND  -  DAMPING COEFFICIENT IN REBOUND AT EACH WHEEL
C     CSFY    -  FY (CORNERING-FORCE) COMPLIANCE-STEER COEFFICIENT
C     CSMZ    -  ALIGNING-MOMENT COMPLIANCE-STEER COEFFICIENT
C     CCFY    -  FY COMPLIANCE-CAMBER COEFFICIENT (0 FOR BEAM AXLE)
C
C  KINEMATIC Z-POLYNOMIAL COEFFICIENTS (2 X 2) - INDEXED (AXLE,POWER)
C
C     CSZ     -  BUMP-STEER COEFFICIENTS
C     CCZ     -  BUMP-CAMBER COEFFICIENTS
C     YROLCF  -  R.C. LATERAL DISP. VS ROLL IN SPRUNG MASS COEFFICIENTS
C     HROLCF  -  R.C. VERTICAL DISP. VS ROLL IN SPRUNG MASS COEFFICIENTS
C
```

```
C  PER-WHEEL (2 X 2 ARRAY) VARIABLES  -  INDEXED (AXLE,SIDE)
C
C     ALFA   -  TIRE SLIP ANGLE
C     GAMMA  -  TIRE CAMBER ANGLE
C     ALFA0  -  STATIC TIRE SLIP ANGLE
C     GAMMA0 -  STATIC TIRE CAMBER ANGLE
C     FY     -  TIRE CORNERING FORCE DUE TO SLIP AND CAMBER
C     MZ     -  TIRE ALIGNING MOMENT
C     FD     -  SUSPENSION VERTICAL DAMPING FORCE
C     FZ     -  TIRE/ROAD NORMAL LOAD
C     ZW     -  SUSPENSION DYNAMIC VERTICAL DISPLACEMENT
C     KNMCBR -  KINEMATIC (BUMP/ROLL) STEER ANGLE
C     KNMCBR -  KINEMATIC (BUMP/ROLL) CAMBER ANGLE
C
C  TIRE FZ-POLYNOMIAL COEFFICIENTS (4 X 2)  -  indexed (AXLE,POWER)
C
C     CALFA  -  cornering-stiffness Fz-polynomial coefficients
C     CGAMMA -  camber-stiffness Fz-polynomial coefficients
C     CALIGN -  aligning-stiffness Fz-polynomial coefficients
C
C
C SINUSOIDAL STEER PARAMETERS  (for equation see function STEER)
C
C     TSWBGN -  global time at steer start (prior to which: SW = 0)
C     TSWEND -  global time at steer end   (after which: SW is frozen)
C     TSWPRD -  length of period     (sec)
C     SWPHSE -  time phase lead      (deg, e.g. +90 ---> cosine)
C     SWAMPL -  amplitude            (steering wheel deg)
C     SWSHFT -  amplitude zero shift (steering wheel deg)
C
C==================================================================
C==================================================================
C==================================================================
C
C   MAIN PROGRAM
C   ------------
C
      IMPLICIT REAL (K,M)
      EXTERNAL FUNCTN
      REAL Y(13), YP(13), YM(13)
      CHARACTER AGAIN
      INTEGER*2 HOUR, MIN, SEC, I100
C
      include DRVMOD.inc
      include GLBL.inc
      include PARS.inc
      include SUSP.inc
      include AERO.inc
      include VARS.inc
      include PRNT.inc
      include mnvr.inc
C
      DATA T/0.0/, Y/13*0.0/
      PI = 4.0 * ATAN(1.0)
C
C   Read input data (includes opening all i/o files)
C
      CALL INDATA
```

```
      CALL INIT
C
C  Initialize Driver Model Vehicle Parameters:
C
      IF(NSTEER .LT. 0) THEN
        IF (ABS(SSKEY) .LE. 0.001) THEN
          CALL DRIVGO
        ELSE
          IF(NSTEER .GT. -100) CALL DRIVGT
        ENDIF
      ENDIF
C
C   Set up output file with simulated time histories
C
      CALL OPNOUT
C
C Start by evaluating derivatives and printing variables at t=0
CMD--Use function TIME for Mac (1 line)
      CALL TIME (ISEC1)
CMD--Use function GETTIM for IBM PC (2 lines)
*      CALL GETTIM (HOUR, MIN, SEC, I100)
*      ISEC1 = 3600*HOUR + 60*MIN + SEC + I100*.01
      CALL FUNCTN (T, Y, YP)
      CALL OUTPUT (T, Y, YP)
C
C Integration loop. Continue until printout time reaches final time.
C Begin each step by allowing subroutines to update internal variables.
C Then use two evaluations of the derivatives to integrate over the
C step.
C
      NLOOP = TEND/DT/IPRINT+1
      DT2 = DT / 2.
      DO 40 ILOOP=1,NLOOP
        DO 30 INNER=1,IPRINT
          DO 10 I=1,NEQN
            YM(I) = Y(I) + DT2 * YP(I)
   10     CONTINUE
          CALL FUNCTN (T+DT2, YM, YP)
          DO 20 I=1,NEQN
            Y(I) = Y(I) + DT * YP(I)
   20     CONTINUE

          T = T + DT
          CALL FUNCTN (T, Y, YP)
   30   CONTINUE
        CALL OUTPUT (T, Y, YP)
        IF (T .GE. TEND) go to 50
   40 CONTINUE
   50 CONTINUE
CMD--Use function TIME for Mac (1 line)
      CALL TIME (ISEC2)
CMD--Use function GETTIM for IBM PC (2 lines)
*      CALL GETTIM (HOUR, MIN, SEC, I100)
*      ISEC2 = 3600*HOUR + 60*MIN + SEC + I100*.01

* End of integration loop.  Print final status of run

      WRITE (*, *) ' Termination at time =', T, ' sec.'
```

```
      WRITE (*,*) ' Computation efficiency: ', (ISEC2 - ISEC1) / T,
     &            ' sec/sim. sec'
      WRITE (*,*) ' '

      CLOSE(IOUT)
      PAUSE 'Done'
      END
************************************************************************
      SUBROUTINE AIRACT(T, YAW, BETA, VYAW)
************************************************************************
C   Subroutine  AIRACT  updates air velocity and sideslip, and the
C   magnitudes of all corresponding aerodynamic forces and moments
C   in the common block /AERO/
C
      IMPLICIT REAL (K,M)
C
      include GLBL.inc
      include PARS.inc
      include AERO.inc
C
C  Look up wind magnitude from TABLE, or, get from user-defined "FWIND"
C  function.  TABLE and FWIND return VWIND in units of kmh or mph.
C
      VWIND = 0.0
      IF(WINDKY .GT. 0) THEN
        CALL TABLE(1, WINDKY, TWIND, WINMAG, T, VWIND)
      ELSE
        VWIND = 0.0
        IF (WINDKY .LT. 0) VWIND = FWIND(T)
      ENDIF
C
C  CONVERT VWIND TO INTERNAL UNITS OF M/SEC OR IN/SEC:
C
      VWIND = VWIND / KMHMPH
C
C   CALCULATE AIR SLIP AND VELOCITY:
C
      RELKSY = KSYWND - YAW
      VAX = ( VWIND * COS(RELKSY) - V * COS(BETA)) / ININFT
      VAY = ( VWIND * SIN(RELKSY) - V * SIN(BETA)) / ININFT
      VAY = VAY - XWBCGS * VYAW / ININFT
      VA2 = VAX * VAX + VAY * VAY
      VA = SQRT(VA2)
      BETAIR = 0.0
      IF(VAY .GT. 0.0) BETAIR = (ATAN2(VAY, VAX) - PI) * TODEG
      IF(VAY .LT. 0.0) BETAIR = (ATAN2(VAY, VAX) + PI) * TODEG
      BETA2 = BETAIR * BETAIR
C
C   CALCULATE AERODYNAMIC FORCES AND MOMENTS ACTING
C   AT GROUND LEVEL, AT HALF WHEELBASE POINT:
C
      CY = -KY * BETAIR
      FYA = QZERO * CY * VA2
C
      CL = CL0 + KL * BETA2
      FZA = -QZERO * CL * VA2
C
      CR = -KR * BETAIR
```

```
      MXA = QZERO * WB * CR * VA2
C
      CM = CM0 + KM * BETA2
      MYA = QZERO * WB * CM * VA2
C
      CN = -KN * BETAIR
      MZA = QZERO * WB * CN * VA2
C
      CD = CD0 + KD * BETA2
      FDRAG = QZERO * CD * VA2
C
C   RESOLVE MOMENTS ABOUT SPRUNG OR TOTAL CG, AS APPROPRIATE:
C
      MXA = MXA - HCGSP * FYA
      MYA = MYA + XWBCGS * FZA
      MZA = MZA - XWBCGT * FYA
C
      RETURN
      END
************************************************************************
      SUBROUTINE BEAM(ALPH, DFDX, FNEG)
************************************************************************
* This subroutine computes a 2x2 Jacobian and a 2-element error
* array for beam rear suspension.
* It is called in turn by the Newton-Raphson solver, MNEWT.
*
* --> ALPH  real  2-element array. 1=left slip, 2=right slip
* <-- DFDX  real  2x2 array, df/dx (partial derivatives)
* <-- FNEG  real  2 negative error functions in equations
*
      IMPLICIT REAL (K,M)
      REAL ALPH(2), DFDX(2,2), FNEG(2)
      include VARS.inc
      include SUSP.inc
      include TSOLVE.inc

* 2x2 Jacobian

      DFDX(2,1) = CSFY(2)*DFYDA(ALPH(1), -GAMMA0(2), FZ(2,1), 2)
     &          + CSMZ(2)*DMZDA(ALPH(1), -GAMMA0(2), FZ(2,1), 2)
      DFDX(1,1) = DFDX(2,1) + 1.
      DFDX(1,2) = CSFY(2)*DFYDA(ALPH(2), GAMMA0(2), FZ(2,2), 2)
     &          + CSMZ(2)*DMZDA(ALPH(2), GAMMA0(2), FZ(2,2), 2)
      DFDX(2,2) = DFDX(1,2) + 1.

* Compute and save tire forces and moments.

      SAVEFY(1) = TIREFY(ALPH(1), -GAMMA0(2), FZ(2,1), 2)
      SAVEFY(2) = TIREFY(ALPH(2),  GAMMA0(2), FZ(2,2), 2)
      SAVEMZ(1) = TIREMZ(ALPH(1), FZ(2,1), 2)
      SAVEMZ(2) = TIREMZ(ALPH(2), FZ(2,2), 2)

* Negative of error function.

      FNEG(1) =  - CSFY(2)* (SAVEFY(1) + SAVEFY(2))
     &           - CSMZ(2)* (SAVEMZ(1) + SAVEMZ(2)) - ALPH(1) + BIAS(1)
      FNEG(2) = FNEG(1) - ALFA0(2)
      FNEG(1) = FNEG(1) + ALFA0(2)
```

```
      RETURN
      END
***********************************************************************
      BLOCK DATA
***********************************************************************
* Initialize variables in common blocks.
C
      IMPLICIT REAL (K,M)
C
      include GLBL.inc
      include PARS.inc
      include MNVR.inc
      include SUSP.inc
      include TIRE.inc
      include AERO.inc
      include VARS.inc
      include PRNT.inc
C
      DATA NEQN/13/, NSTEER/1/, TODEG/1.0/, SW/0.0/, FW/2*0.0/, AY/0.0/
      DATA RHO/0.0/, KROLL/0.0/, CSROLL/0.0/, CSZ/4*0.0/, CCZ/4*0.0/
      DATA ALFA/4*0.0/, GAMMA/4*0.0/, FY/4*0.0/, MZ/4*0.0/, FD/4*0.0/
      DATA ZW/4*0.0/, YROLCF/4*0.0/, HROLCF/4*0.0/
      DATA KNMSTR/4*0.0/, CPLSTR/4*0.0/, TTLSTR/4*0.0/, KNMCBR/4*0.0/
      DATA YOUTDR/13*0.0/, STORQ/0.0/, MMCOL/0.0/
C
      DATA TSWBGN/0.0/, TSWEND/0.0/, SWAMPL/0.0/, TSWPRD/0.0/
      DATA SWPHSE/0.0/, SWSHFT/0.0/, DRLAG/0.0/, DRPREV/0.0/
      DATA VA/0.0/, BETAIR/0.0/, FYA/0.0/, FZA/0.0/, FZ/4*0.0/
      DATA MXA/0.0/, MYA/0.0/, MZA/0.0/, FDRAG/0.0/
      DATA XPNT/999*0.0/, YPNT/999*0.0/, SLOPE/999*0.0/
C
      DATA G/9.81/, ININFT/1/, KMHMPH/3.6/, UOMEGA/'rad/sec'/
      DATA UDISP/'m'/, UDIST/'m'/, UANGL/'rad'/, UVELFT/'m/s'/
      DATA UFORC/'N'/, UTORQ/'m-N'/, KINEM/.TRUE./, BEAM/.TRUE./
      DATA LINE/-1/, NPAGE/1/, INDX/0/, BLNK12/'            '/
      DATA FNREAD /' '/
C
      END
***********************************************************************
      FUNCTION DFYDA(ALPHA, GAMMA, FZ, AXLE)
***********************************************************************
* This version (UMTRI default) computes cornering stiffness as
* a cubic function of FZ.

* In this version, the arguments ALPHA and GAMMA are not used.

* --> alpha  real    slip angle
* --> gamma  real    camber angle
* --> FZ     real    vertical load
* --> AXLE   integer axle no. (1 or 2)
* <-- DFYDA  real    partial derivative of Fy with respect to alpha
*
      include TIRE.inc
      INTEGER AXLE
      DFYDA = POLY4(CALFA(1, AXLE), FZ)
      RETURN
      END
***********************************************************************
```

```
      FUNCTION DFYDG(ALPHA, GAMMA, FZ, AXLE)
************************************************************************
* This version (UMTRI default) computes camber stiffness as a
* cubic function of FZ.

* In this version, the arguments ALPHA and GAMMA are not used.

* --> alpha  real    slip angle
* --> gamma  real    camber angle
* --> FZ     real    vertical load
* --> AXLE   integer axle no. (1 or 2)
* <-- DFYDG  real    partial derivative of Fy with respect to gamma
*
      include TIRE.inc
      INTEGER AXLE
      DFYDG = POLY4(CGAMMA(1, AXLE), FZ)
      RETURN
      END
************************************************************************
      FUNCTION DMZDA(ALPHA, GAMMA, FZ, AXLE)
************************************************************************
* This version (UMTRI default) computes aligning stiffness as a cubic
* function of FZ.

* In this version, the arguments ALPHA and GAMMA are not used.

* --> alpha  real    slip angle
* --> gamma  real    camber angle
* --> FZ     real    vertical load
* --> AXLE   integer axle no. (1 or 2)
* <-- DMZDA  real    partial derivative of Mz with respect to alpha
*
      include TIRE.inc
      INTEGER AXLE
      DMZDA = POLY4(CALIGN(1, AXLE), FZ)
      RETURN
      END
C***********************************************************************
C***********************************************************************
C
C DRIVE1: Reads Driver Model (Path, Preview, Lag) Parameters->unit IREAD
C
C===========Author and Modification Section============================
C
C  Author:       C. C. MacAdam
C
C  Date written: 05/19/88
C
C  Written on:
C
C  Modifications:
C
C======================================================================
C
C===========Algorithm Description======================================
C
C  Purpose and use:
C
```

```
C  Error conditions:
C
C  Machine dependencies: none
C
C  Called By: INDATA
C
C======================================================================
C
      SUBROUTINE DRIVE1(DFW)
      SAVE
C
C===========Variable Descriptions======================================
C
C---Arguments passed:
C
C  DFW...steer angle of front tires [or average] (rad)
C
C
C---COMMON blocks------------------------------------------------------
C
      include drvmod.inc
      include pars.inc
      include glbl.inc
C
C---DRIV.BLK common block variables------------------------------------
--
C
C  CAF...total cornering stiffness of tires on left front susp (lb/rad)
C  CAR...total cornering stiffness of tires on left rear susp (lb/rad)
C  WHBS..wheelbase of vehicle (center-line of front & rear susp) (ft)
C  WF....static load on front suspension (lb)
C  WR....static load on rear suspension (lb)
C  U.....initial velocity (ft/sec)
C
C---DRVST1.BLK common block variables
C
C   GRAV.....gravitational constant
C   TICYCL...driver model sample time (sec)
C   TSS......minimum preview time (sec)
C   DMAX.....upper bound on front wheel angle steer (rad)
C   XP,YP....x-y path coords(SAE) wrt inertial coords [input] (ft)
C   TAUMEM...driver transport time dealy [input parameter] (sec)
C   TFF......driver model preview time [input parameter] (sec)
C   RM.......vehicle mass (slug)
C   A........distance from c.g. to front suspension center-line (ft)
C   B........distance from c.g. to rear suspension center-line (ft)
C   RI.......total vehicle yaw inertia (slug-ft)
C   PSIO.....current yaw angle reference value (rad)
C   NTF......number of points in the preview time interval
C   NP.......number of points in the x-y trajectory table
C   TLAST....last time driver model calulated a steer value (sec)
C   DFWLST...last value of steer calculated by driver model (rad)
C   TILAST...last sample time driver model calulated a steer value (sec)
C   DMEM.....2-dim array (time & steer history) used in delay calculat'n
C   XT,YT....transformation of XP,YP in vehicle body axes (ft)
C
C---Local variables----------------------------------------------------
C
```

```
C  WGHT..total static weight on front and rear suspsensions (lb)
C  DFW...steer angle of front tires [or average] (rad)
C
C---Functions and subroutines------------------------------------------
C
C      None
C
C=====================================================================
C
C============Process Block============================================
C
      GRAV = 32.2
      TICYCL = 0.0099
      TSS = 0.0
      DMAX = 0.2
C
      DO 40 J = 1, NP
        READ(IREAD,30) XPDR(J), YPDR(J)
   30   FORMAT(2F12.4)
   40 CONTINUE
      READ(IREAD,60) TAUMEM, TFF
   60 FORMAT(F12.4)
C
      PSI0 = 0.0
      NTF = 10
      DO 80 J = 1, NP
        XT(J) = XPDR(J) * COS(PSI0) + YPDR(J) * SIN(PSI0)
        YT(J) = -XPDR(J) * SIN(PSI0) + YPDR(J) * COS(PSI0)
   80 CONTINUE
      TLAST = 0.
      DFWLST = 0.
      TILAST = 0.
      DFW = 0.
      DO 90 I = 1, 100
        DMEM(I,1) = 0.
   90 DMEM(I,2) = -1.
      RETURN
      END
C*********************************************************************
C
C  Closed-Loop Steer Calculation
C
C DRIVER: Computes closed-loop steering control during the simulation
C
C============Author and Modification Section==========================
C
C  Author:       C. C. MacAdam
C
C  Date written: 05/19/88
C
C  Written on:
C
C  Modifications:
C
C=====================================================================
C
C============Algorithm Description====================================
C
```

```
C  Purpose and use:
C
C  Error conditions:
C
C  References:
C
C              MacAdam, C.C.  "Development of Driver/Vehicle Steering
C                    Interaction Models for Dynamic Analysis," Interim
C                    Technical Report, U.S. Army Tank Automotive Command
C                    Contract No. DAAE07-85-C-R069, The University of
C                    Michigan Transportation Research Institute Report
C                    No. UMTRI-86-41, July 1986.
C
C              MacAdam, C.C.  "Application of an Optimal Preview Control
C                    for Simulation of Closed-Loop Automobile Driving,"
C                    IEEE Transactions on Systems, Man, and Cybernetics,
C                    Vol. 11, June 1981.
C
C              MacAdam, C.C.  "An Optimal Preview Control for Linear
C                    Systems," Journal of Dynamic Systems, Measurement,
C                    and Control, ASME, Vol. 102, No. 3, September 1980.
C
C
C  Machine dependencies: none
C
C  Called By: STEER (function)
C
C=========================================================================
C
      SUBROUTINE DRIVER(X, Y, DFW, DFWNOW)
      SAVE
C
C============Variable Descriptions========================================
C
C---Arguments passed:
C
C  X.......time in the simulation (sec)
C  Y.......current state vector obtained from WIND/STEER
C  DFW.....closed-loop steering control returned to WIND/STEER
C  DFWNOW..current steering angle [average] of front wheels,
C        after effects of roll-steer, compliance, etc.
C
      DIMENSION Y(5), YC(5)
      DIMENSION DUMV11(4)
      DIMENSION DUMV1(4), VECM(4)
      DIMENSION DUMM1(4,4), DUMM2(4,4)
      DIMENSION FFV(4)
C
C---COMMON blocks---------------------------------------------------------
C
      include drvmod.inc
      include pars.inc
      include aero.inc
      include glbl.inc
C
C---DRIV.BLK common block variables--------------------------------------
--
C
```

```
C  CAF...total cornering stiffness of tires on left front susp (lb/rad)
C  CAR...total cornering stiffness of tires on left rear susp (lb/rad)
C  WHBS..wheelbase of vehicle (center-line of front & rear susp) (ft)
C  WF....static load on front suspension (lb)
C  WR....static load on rear suspension (lb)
C  U.....initial velocity (ft/sec)
C
C---DRVST1.BLK common block variables
C
C   GRAV.....gravitational constant
C   TICYCL...driver model sample time (sec)
C   TSS......minimum preview time (sec)
C   DMAX.....upper bound on front wheel angle steer (rad)
C   XP,YP....x-y path coords(SAE) wrt inertial coords [input] (ft)
C   TAUMEM...driver transport time dealy [input parameter] (sec)
C   TFF......driver model preview time [input parameter] (sec)
C   RM.......vehicle mass (slug)
C   A........distance from c.g. to front suspension center-line (ft)
C   B........distance from c.g. to rear suspension center-line (ft)
C   RI.......total vehicle yaw inertia (slug-ft)
C   PSIO.....current yaw angle reference value (rad)
C   NTF......number of points in the preview time interval
C   NP.......number of points in the x-y trajectory table
C   TLAST....last time driver model calulated a steer value (sec)
C   DFWLST...last value of steer calculated by driver model (rad)
C   TILAST...last sample time driver model calulated a steer value (sec)
C   DMEM.....2-dim array (time & steer history) used in delay calculat'n
C   XT,YT....transformation of XP,YP in vehicle body axes (ft)
C
C---TRSSTR.BLK common block variables
C
C  TTT.......transition matrix at 10 discrete points in preview interval
C  TTT1......integral of trans matrix wrt preview time
C  GV.........vector of control gain coefficients
C
C---Local variables------------------------------------------------------
C
C   YC.......local (body-axis based) copy of state vector Y
C   VECM.....observer vector - lateral displacement from state vector
C   DUMV1....work vector
C   DUMV11...   "
C   DUMM1....work matrix
C   DUMM2....   "
C   T........time in the simulation (sec)
C   EPSI.....yaw angle between body axis and current index value, PSI0
C   PSIO.....current nominal value of yaw angle used for linearization
C   NP.......number of points in x-y path table
C   XP,YP....x-y inertial path table [input] (ft)
C   XT,YT....x-y path table transformed to body axis [PSIO] system (ft)
C   EPSY2....cumulative preview path error squared
C   EPSY.....mean squared value of cumulative preview path error
C   TSUM.....scalar work quantity
C   SSUM.....scalar work quantity
C   DFWLST...steering control from last calculation (rad)
C   TJI......preview time ahead from present time value (sec)
C   I,J,K....integer counters
C   XCAR.....preview distance ahead in feet (ft)
C   X0.......present forward postion of vehicle c.g. (ft)
```

```
C   TTAB.....current time less the driver delay, TAUMEM. Used to access
C         the delayed driver response stored in DMEM array. (sec)
C   S1.......scalar work quantity
C   T1.......scalar work quantity
C   EP.......previewed path error (ft)
C   FFV......aerodynamic lateral accel and yaw accel "sensory" vector
C
C---Functions and subroutines------------------------------------------
C
         EXTERNAL TRAJ, GMPRD
C
C====================================================================
C
C===========Process Block============================================
C
C
        DATA VECM /1.0, 3*0.0/
C
C  Update Aerodynamic accel (force/moment) vector for driver model:
C
        FFV(1) = 0.0
        FFV(2) = FYA / RM * WEIGHT / SPWGHT
        FFV(3) = MZA / ININFT/ RI
        FFV(4) = 0.0
C
    1   T = X
        EPSI = ABS(Y(4) - PSI0)
        DO 10 I = 1, 5
   10   YC(I) = Y(I)
C         IF (EPSI .LE. .0002) GO TO 30
C
C  Update Coordinate Transformation
C
        PSI0 = Y(4)
        DO 20 J = 1, NP
          XT(J) = XPDR(J) * COS(PSI0) + YPDR(J) * SIN(PSI0)
   20   YT(J) = -XPDR(J) * SIN(PSI0) + YPDR(J) * COS(PSI0)
C
   30   Y0 = -Y(5) * SIN(PSI0) + Y(1) * COS(PSI0)
        X0 = Y(5) * COS(PSI0) + Y(1) * SIN(PSI0)
        YC(1) = Y0
        YC(4) = Y(4) - PSI0
        EPSY2 = 0.
        TSUM = 0.
        SSUM = 0.
        DFW = DFWLST
C
C  Return if time from last calculation less than sample interval
C
        IF (T - TILAST .LE. TICYCL) RETURN
C
C
C  Update tire cornering stiffnesses and vehicle velocity
C  and recalculate transition matrix:  Not Used Presently
C  ***  COMMENTED OUT  ***
C
C       CAFTEM = (CCAF1*FFZL1+CCAF2*FFZL2) / (FFZL1+FFZL2)
C       CARTEM = (CCAR1*FFZL3+CCAR2*FFZL4) / (FFZL3+FFZL4)
```

```
C      CAF = CAFTEM
C      CAR = CARTEM
C      UTEMP = DMVELC
C      U = UTEMP
C      CALL TRANS
C
C  Loop to calculate optimal preview control per References 2 & 3:
C   (NTF points within the preview interval)
C
      DO 50 I = 1, NTF
        TJI = (TFF - TSS) / NTF * I + TSS
        DO 40 J = 1, 4
          DO 40 K = 1, 4
            DUMM1(J,K) = TTT1(J,K,I)
   40   DUMM2(J,K) = TTT(J,K,I)
        CALL GMPRD(VECM, DUMM1, DUMV11, 1, 4, 4)
        CALL GMPRD(VECM, DUMM2, DUMV1, 1, 4, 4)
        CALL GMPRD(DUMV1, YC, T1, 1, 4, 1)
C
C  Get observed path input, YPATH, within preview interval at XCAR ft:
C
        XCAR = X0 + U * TJI
        CALL TRAJ(XCAR, XT, YT, YPATH)
C
        CALL GMPRD(DUMV11, GV, S1, 1, 4, 1)
        CALL GMPRD(DUMV11, FFV, DYAERO, 1, 4, 1)
C
C  EP is the previewed path error at this preview point.
C
        EP = T1 + S1 * DFWNOW + DYAERO - YPATH
        TSUM = TSUM + EP * S1
        SSUM = SSUM + S1 * S1
C
C  Cumulative preview error calculation (unrelated to control)
C
        EPSY2 = EPSY2 + EP * EP * (TFF - TSS) / NTF
C
   50 CONTINUE
C
C  Cumulative preview error calculation (unrelated to control)
C
      EPSY = SQRT(EPSY2) / (TFF - TSS)
C
C  Optimal value - no delay yet.
C
      DFW = -TSUM / SSUM + DFWNOW
C
C  Maximum steer bound set at DMAX (arbitrary)
C
      IF (ABS(DFW) .GT. DMAX) DFW = DMAX * SIGN(1.,DFW)
C
C  Store steer history and corresponding times in DMEM.
C  Retrieve steer delayed by TAUMEM sec and return as
C  delayed driver steer control, DFW.
C
      DO 60 J = 1, 2
        DO 60 I = 1, 99
          DMEM(101 - I,J) = DMEM(100 - I,J)
```

```
   60 CONTINUE
      DMEM(1,1) = DFW
      DMEM(1,2) = T
      TTAB = T - TAUMEM
      DO 70 I = 1, 99
        IJK = I
        IF (DMEM(I + 1,2) .LE. TTAB .AND. DMEM(I,2) .GE. TTAB)
     1      GO TO 90
   70 CONTINUE
      WRITE (*,80)TAUMEM,DFW,X
   80 FORMAT ('0', '***** TAUMEM PROBABLY TOO LARGE *****',
     &        /,3(1X,G12.6))
      STOP
   90 DFW = 0.0
      IF(T .GE. TAUMEM) DFW = DMEM(IJK,1)
C
C  Save steer and time values for next calulation.
C
      DFWLST = DFW
      TLAST = X
      TILAST = X
      RETURN
      END
C***********************************************************************
C
C  Closed-Loop Steer Calculation
C
C DRIVET: Computes closed-loop steering TORQUE control during the simul
C
C============Author and Modification Section============================
C
C  Author:       C. C. MacAdam
C
C  Date written: 01/30/89
C
C  Written on:
C
C  Modifications:
C
C=======================================================================
C
C============Algorithm Description======================================
C
C  Purpose and use:
C
C  Error conditions:
C
C  References:
C
C              MacAdam, C.C.  "Development of Driver/Vehicle Steering
C                    Interaction Models for Dynamic Analysis," Final
C                    Technical Report, U.S. Army Tank Automotive Command
C                    Contract No. DAAE07-85-C-R069, The University of
C                    Michigan Transportation Research Institute Report
C                    No. UMTRI-88-53, December 1988.
C
C              MacAdam, C.C.  "Application of an Optimal Preview Control
C                    for Simulation of Closed-Loop Automobile Driving,"
```

```
C                   IEEE Transactions on Systems, Man, and Cybernetics,
C                   Vol. 11, June 1981.
C
C              MacAdam, C.C.  "An Optimal Preview Control for Linear
C                   Systems," Journal of Dynamic Systems, Measurement,
C                   and Control, ASME, Vol. 102, No. 3, September 1980.
C
C
C  Machine dependencies: none
C
C  Called By: STEER (function)
C
C=====================================================================
C
      SUBROUTINE DRIVET(X, Y, DRTORQ, DRTNOW)
      SAVE
C
C============Variable Descriptions====================================
C
C---Arguments passed:
C
C  X.......time in the simulation (sec)
C  Y.......current state vector obtained from WIND/STEER
C  DRTORQ.....closed-loop TORQUE control returned to WIND/STEER
C  DRTNOW.....current steering TORQUE
C
      DIMENSION Y(7), YC(7)
      DIMENSION DUMV11(6)
      DIMENSION DUMV1(6), VECM(6)
      DIMENSION DUMM1(6,6), DUMM2(6,6)
      DIMENSION FFV(6)
C
C---COMMON blocks-----------------------------------------------------
C
      include drvtor.inc
      include pars.inc
      include aero.inc
      include glbl.inc
      include vars.inc
C
C---DRIV.BLK common block variables-----------------------------------
C
C  CAF...total cornering stiffness of tires on left front susp (lb/rad)
C  CAR...total cornering stiffness of tires on left rear susp (lb/rad)
C  WHBS..wheelbase of vehicle (center-line of front & rear susp) (ft)
C  WF....static load on front suspension (lb)
C  WR....static load on rear suspension (lb)
C  U.....initial velocity (ft/sec)
C
C---DRVST1.BLK common block variables
C
C   GRAV.....gravitational constant
C   TICYCL...driver model sample time (sec)
C   TSS......minimum preview time (sec)
C   DMAX.....upper bound on front wheel angle steer (rad)
C   XP,YP....x-y path coords(SAE) wrt inertial coords [input] (ft)
C   TAUMEM...driver transport time dealy [input parameter] (sec)
C   TFF......driver model preview time [input parameter] (sec)
```

```
C   RM.......vehicle mass (slug)
C   A........distance from c.g. to front suspension center-line (ft)
C   B........distance from c.g. to rear suspension center-line (ft)
C   RI.......total vehicle yaw inertia (slug-ft)
C   PSIO.....current yaw angle reference value (rad)
C   NTF......number of points in the preview time interval
C   NP.......number of points in the x-y trajectory table
C   TLAST....last time driver model calulated a steer value (sec)
C   STLST...last value of steer calculated by driver model (rad)
C   TILAST...last sample time driver model calulated a steer value (sec)
C   DMEM.....2-dim array (time & steer history) used in delay calculat'n
C   XT,YT....transformation of XP,YP in vehicle body axes (ft)
C
C---TRSSTR.BLK common block variables
C
C  TTT.......transition matrix at 10 discrete points in preview interval
C  TTT1......integral of trans matrix wrt preview time
C  GGV.........vector of control gain coefficients
C
C---Local variables-----------------------------------------------------
C
C   YC.......local (body-axis based) copy of state vector Y
C   VECM.....observer vector - lateral displacement from state vector
C   DUMV1....work vector
C   DUMV11...    "
C   DUMM1....work matrix
C   DUMM2....    "
C   T........time in the simulation (sec)
C   EPSI.....yaw angle between body axis and current index value, PSI0
C   PSIO.....current nominal value of yaw angle used for linearization
C   NP.......number of points in x-y path table
C   XP,YP....x-y inertial path table [input] (ft)
C   XT,YT....x-y path table transformed to body axis [PSIO] system (ft)
C   EPSY2....cumulative preview path error squared
C   EPSY.....mean squared value of cumulative preview path error
C   TSUM.....scalar work quantity
C   SSUM.....scalar work quantity
C   DFWLST...steering control from last calculation (rad)
C   TJI......preview time ahead from present time value (sec)
C   I,J,K....integer counters
C   XCAR.....preview distance ahead in feet (ft)
C   X0.......present forward postion of vehicle c.g. (ft)
C   TTAB.....current time less the driver delay, TAUMEM. Used to access
C         the delayed driver response stored in DMEM array. (sec)
C   S1.......scalar work quantity
C   T1.......scalar work quantity
C   EP.......previewed path error (ft)
C   FFV......aerodynamic lateral accel and yaw accel "sensory" vector
C            & power boost influence
C
C---Functions and subroutines-------------------------------------------
C
      EXTERNAL TRAJ, GMPRD
C
C======================================================================
C
C============Process Block=============================================
C
```

```
C
      DATA VECM /1.0, 5*0.0/
      DATA STLST /0.0/
C
C  Update Aerodynamic accel (force/moment) vector for driver model:
C
      FFV(1) = 0.0
      FFV(2) = FYA / RM * WEIGHT / SPWGHT
      FFV(3) = MZA / ININFT/ RI
      FFV(4) = 0.0
      FFV(5) = 0.0
      FFV(6) = 0.0
C
  1   T = X
      EPSI = ABS(Y(4) - PSI0)
      DO 10 I = 1, 7
  10  YC(I) = Y(I)
C       IF (EPSI .LE. .0002) GO TO 30
C
C  Update Coordinate Transformation
C
      PSI0 = Y(4)
      DO 20 J = 1, NP
        XT(J) = XPDR(J) * COS(PSI0) + YPDR(J) * SIN(PSI0)
  20  YT(J) = -XPDR(J) * SIN(PSI0) + YPDR(J) * COS(PSI0)
C
  30  Y0 = -Y(7) * SIN(PSI0) + Y(1) * COS(PSI0)
      X0 = Y(7) * COS(PSI0) + Y(1) * SIN(PSI0)
      YC(1) = Y0
      YC(4) = Y(4) - PSI0
      EPSY2 = 0.
      TSUM = 0.
      SSUM = 0.
      DRTORQ = STLST
C
C  Return if time from last calculation less than sample interval
C
      IF (T - TILAST .LT. TICYCL) RETURN
C
C
C  Update tire cornering stiffnesses and vehicle velocity
C  and recalculate transition matrix:  Not Used Presently
C  ***  COMMENTED OUT  ***
C
C       CAFTEM = (CCAF1*FFZL1+CCAF2*FFZL2) / (FFZL1+FFZL2)
C       CARTEM = (CCAR1*FFZL3+CCAR2*FFZL4) / (FFZL3+FFZL4)
C       CAF = CAFTEM
C       CAR = CARTEM
C       UTEMP = DMVELC
C       U = UTEMP
C       CALL TRANST
C
C  Loop to calculate optimal preview control per References 2 & 3:
C   (NTF points within the preview interval)
C
      DO 50 I = 1, NTF
        TJI = (TFF - TSS) / NTF * I + TSS
        DO 40 J = 1, 6
```

```
         DO 40 K = 1, 6
           DUMM1(J,K) = TTTT1(J,K,I)
   40    DUMM2(J,K) = TTTT(J,K,I)
         CALL GMPRD(VECM, DUMM1, DUMV11, 1, 6, 6)
         CALL GMPRD(VECM, DUMM2, DUMV1, 1, 6, 6)
         CALL GMPRD(DUMV1, YC, T1, 1, 6, 1)
C
C  Get observed path input, YPATH, within preview interval at XCAR ft:
C
         XCAR = X0 + U * TJI
         CALL TRAJ(XCAR, XT, YT, YPATH)
C
         CALL GMPRD(DUMV11, GGV, S1, 1, 6, 1)
         CALL GMPRD(DUMV11, FFV, DYAERO, 1, 6, 1)
C
C  EP is the previewed path error at this preview point.
C
         EP = T1 + S1 * DRTNOW + DYAERO - YPATH
         TSUM = TSUM + EP * S1
         SSUM = SSUM + S1 * S1
C
C  Cumulative preview error calculation (unrelated to control)
C
         EPSY2 = EPSY2 + EP * EP * (TFF - TSS) / NTF
C
   50 CONTINUE
C
C  Cumulative preview error calculation (unrelated to control)
C
      EPSY = SQRT(EPSY2) / (TFF - TSS)
C
C  Optimal value - no delay yet.
C
      DRTORQ = -TSUM / SSUM + DRTNOW
C
C  Maximum steer bound set at STMAX (arbitrary)
C
      IF (ABS(DRTORQ) .GT. STMAX) DRTORQ = STMAX * SIGN(1.,DRTORQ)
C
C  Store torque history and corresponding times in DMEM.
C  Retrieve steer delayed by TAUMEM sec and return as
C  delayed driver torque control, DRTORQ.
C
      DO 60 J = 1, 2
        DO 60 I = 1, 99
          DMEM(101 - I,J) = DMEM(100 - I,J)
   60 CONTINUE
      DMEM(1,1) = DRTORQ
      DMEM(1,2) = T
      TTAB = T - TAUMEM
      DO 70 I = 1, 99
        IJK = I
        IF (DMEM(I + 1,2) .LT. TTAB .AND. DMEM(I,2) .GE. TTAB)
     1     GO TO 90
   70 CONTINUE
      WRITE (*,80)TAUMEM,DRTORQ,X
   80 FORMAT ('0', '***** TAUMEM PROBABLY TOO LARGE *****',
     &        /,3(1X,G12.6))
```

```
      STOP
   90 DRTORQ = 0.0
      IF(T .GE. TAUMEM) DRTORQ = DMEM(IJK,1)
C
C  Save steer and time values for next calculation.
C
      STLST = DRTORQ
      TLAST = X
      TILAST = X
      RETURN
      END
C*********************************************************************
C*********************************************************************
C
C   ***  CHRYSLER Initialization Entry for the Driver Model  ***
C
C DRIVGO: Intializes driver model vehicle-based parameters from COMMONs
C
C============Author and Modification Section===========================
C
C  Author:       C. C. MacAdam
C
C  Date written: 05/19/88
C
C  Written on:    Mac II
C
C  Modifications:
C
C=====================================================================
C
C============Algorithm Description=====================================
C
C  Purpose and use:
C
C  Error conditions:
C
C  References:
C
C              MacAdam, C.C.  "Development of Driver/Vehicle Steering
C                    Interaction Models for Dynamic Analysis," Interim
C                    Technical Report, U.S. Army Tank Automotive Command
C                    Contract No. DAAE07-85-C-R069, The University of
C                    Michigan Transportation Research Institute Report
C                    No. UMTRI-86-41, July 1986.
C
C              MacAdam, C.C.  "Application of an Optimal Preview Control
C                    for Simulation of Closed-Loop Automobile Driving,"
C                    IEEE Transactions on Systems, Man, and Cybernetics,
C                    Vol. 11, June 1981.
C
C              MacAdam, C.C.  "An Optimal Preview Control for Linear
C                    Systems," Journal of Dynamic Systems, Measurement,
C                    and Control, ASME, Vol. 102, No. 3, September 1980.
C
C
C  Machine dependencies: none
C
C  Called By: INDATA
```

```
C
C=========================================================================
C
      SUBROUTINE DRIVGO
      SAVE
C
C============Variable Descriptions========================================
C
C---Arguments passed:  None
C
C
C---COMMON blocks---------------------------------------------------------
C
      include drvmod.inc
      include pars.inc
      include glbl.inc
      include tire.inc
      include vars.inc
C
C---DRIV.BLK common block variables-------------------------------------
--
C
C  CAF...total cornering stiffness of tires on left front susp (lb/rad)
C  CAR...total cornering stiffness of tires on left rear susp (lb/rad)
C  WHBS..wheelbase of vehicle (center-line of front & rear susp) (ft)
C  WF....static load on front suspension (lb)
C  WR....static load on rear suspension (lb)
C  U.....initial velocity (ft/sec)
C
C---DRVST1.BLK common block variables
C
C   GRAV.....gravitational constant
C   TICYCL...driver model sample time (sec)
C   TSS......minimum preview time (sec)
C   DMAX.....upper bound on front wheel angle steer (rad)
C   XP,YP....x-y path coords(SAE) wrt inertial coords [input] (ft)
C   TAUMEM...driver transport time dealy [input parameter] (sec)
C   TFF......driver model preview time [input parameter] (sec)
C   RM.......vehicle mass (slug)
C   A........distance from c.g. to front suspension center-line (ft)
C   B........distance from c.g. to rear suspension center-line (ft)
C   RI.......total vehicle yaw inertia (slug-ft)
C   PSIO.....current yaw angle reference value (rad)
C   NTF......number of points in the preview time interval
C   NP.......number of points in the x-y trajectory table
C   TLAST....last time driver model calulated a steer value (sec)
C   DFWLST...last value of steer calculated by driver model (rad)
C   TILAST...last sample time driver model calulated a steer value (sec)
C   DMEM.....2-dim array (time & steer history) used in delay calculat'n
C   XT,YT....transformation of XP,YP in vehicle body axes (ft)
C
C---Local variables-------------------------------------------------------
C
C  A.....distance from c.g. to front suspension center-line (ft)
C  B.....distance from c.g. to rear suspension center-line (ft)
C  WGHT..total static weight on front and rear suspsensions (lb)
C  RM....total static mass (slug)
C  DFW...steer angle of front tires [or average] (rad)
```

```
C
C---Functions and subroutines-----------------------------------------------
C
      EXTERNAL TRANS
C
C==========================================================================
C
C============Process Block=================================================
C
C
      WGHT = WEIGHT
      B = WRATIO * WB / 12.
      A = (1. - WRATIO) * WB / 12.
      RM = WGHT / GRAV
      WHBS = A + B
      WF = WGHT * B / WHBS
      WR = WGHT * A / WHBS
C     RI = A * B * RM
      RI = IZZ / 12.
C
C  Initial Tire Cornering Stiffnesses for Driver Model (lb/rad):
C  (flip sign from SAE convention to positive values here)
C
      CAF = 0.0
      CAR = 0.0
      DO 30 NAXLE = 1, 2
        DO 20 NSIDE = 1, 2
          CALF = 0.0
          DO 10 NPOWER = 1, 4
            CALF = CALF
     1           + CALFA(NPOWER,NAXLE) * FZ(NAXLE,NSIDE)**(NPOWER-1)
   10     CONTINUE
         IF(NAXLE .EQ. 1) CAF = CAF - 0.5 * CALF
         IF(NAXLE .EQ. 2) CAR = CAR - 0.5 * CALF
   20   CONTINUE
   30 CONTINUE
C
C  Speed in ft/sec:
C
      U = V * KMHMPH * 88. / 60.
C
C
C  Call TRANS to Calculate Transition Matrix
C
      CALL TRANS
C
      RETURN
      END
C**************************************************************************
C**************************************************************************
C
C   ***  CHRYSLER Initialization Entry for the Driver Model  ***
C
C DRIVGT: Intializes driver model vehicle-based parameters from COMMONs
C
C============Author and Modification Section===========================
C
C  Author:       C. C. MacAdam
```

```
C
C  Date written: 01/30/89
C
C  Written on:    Mac II
C
C  Modifications:
C
C=====================================================================
C
C============Algorithm Description=====================================
C
C  Purpose and use:
C
C  Error conditions:
C
C  References:
C
C              MacAdam, C.C.  "Development of Driver/Vehicle Steering
C                    Interaction Models for Dynamic Analysis," Final
C                    Technical Report, U.S. Army Tank Automotive Command
C                    Contract No. DAAE07-85-C-R069, The University of
C                    Michigan Transportation Research Institute Report
C                    No. UMTRI-88-53, December 1988.
C
C              MacAdam, C.C.  "Application of an Optimal Preview Control
C                    for Simulation of Closed-Loop Automobile Driving,"
C                    IEEE Transactions on Systems, Man, and Cybernetics,
C                    Vol. 11, June 1981.
C
C              MacAdam, C.C.  "An Optimal Preview Control for Linear
C                    Systems," Journal of Dynamic Systems, Measurement,
C                    and Control, ASME, Vol. 102, No. 3, September 1980.
C
C
C  Machine dependencies: none
C
C  Called By: INDATA
C
C=====================================================================
C
      SUBROUTINE DRIVGT
      SAVE
C
C============Variable Descriptions=====================================
C
C---Arguments passed:  None
C
C
C---COMMON blocks-----------------------------------------------------
C
      include drvtor.inc
      include pars.inc
      include glbl.inc
      include tire.inc
      include vars.inc
C
C---DRIV.BLK common block variables------------------------------------
--
```

```
C
C  CAF...total cornering stiffness of tires on left front susp (lb/rad)
C  CAR...total cornering stiffness of tires on left rear susp (lb/rad)
C  WHBS..wheelbase of vehicle (center-line of front & rear susp) (ft)
C  WF....static load on front suspension (lb)
C  WR....static load on rear suspension (lb)
C  U.....initial velocity (ft/sec)
C
C---DRVST1.BLK common block variables
C
C   GRAV.....gravitational constant
C   TICYCL...driver model sample time (sec)
C   TSS......minimum preview time (sec)
C   DMAX.....upper bound on front wheel angle steer (rad)
C   XP,YP....x-y path coords(SAE) wrt inertial coords [input] (ft)
C   TAUMEM...driver transport time dealy [input parameter] (sec)
C   TFF......driver model preview time [input parameter] (sec)
C   RM.......vehicle mass (slug)
C   A........distance from c.g. to front suspension center-line (ft)
C   B........distance from c.g. to rear suspension center-line (ft)
C   RI.......total vehicle yaw inertia (slug-ft)
C   PSIO.....current yaw angle reference value (rad)
C   NTF......number of points in the preview time interval
C   NP.......number of points in the x-y trajectory table
C   TLAST....last time driver model calulated a steer value (sec)
C   DFWLST...last value of steer calculated by driver model (rad)
C   TILAST...last sample time driver model calulated a steer value (sec)
C   DMEM.....2-dim array (time & steer history) used in delay calculat'n
C   XT,YT....transformation of XP,YP in vehicle body axes (ft)
C
C---Local variables---------------------------------------------------
C
C  A.....distance from c.g. to front suspension center-line (ft)
C  B.....distance from c.g. to rear suspension center-line (ft)
C  WGHT..total static weight on front and rear suspsensions (lb)
C  RM....total static mass (slug)
C  DFW...steer angle of front tires [or average] (rad)
C
C---Functions and subroutines-----------------------------------------
C
      EXTERNAL TRANS
C
C====================================================================
C
C===========Process Block============================================
C
C
      WGHT = WEIGHT
      B = WRATIO * WB / 12.
      A = (1. - WRATIO) * WB / 12.
      RM = WGHT / GRAV
      WHBS = A + B
      WF = WGHT * B / WHBS
      WR = WGHT * A / WHBS
C     RI = A * B * RM
      RI = IZZ / 12.
      STMAX = 1000.
C
```

```
C  Initial Tire Cornering Stiffnesses for Driver Model (lb/rad):
C  (flip sign from SAE convention to positive values here)
C
      CAF = 0.0
      CAR = 0.0
      DO 30 NAXLE = 1, 2
        DO 20 NSIDE = 1, 2
          CALF = 0.0
          DO 10 NPOWER = 1, 4
            CALF = CALF
     1           + CALFA(NPOWER,NAXLE) * FZ(NAXLE,NSIDE)**(NPOWER-1)
   10     CONTINUE
         IF(NAXLE .EQ. 1) CAF = CAF - 0.5 * CALF
         IF(NAXLE .EQ. 2) CAR = CAR - 0.5 * CALF
   20   CONTINUE
   30 CONTINUE
C
C  Speed in ft/sec:
C
      U = V * KMHMPH * 88. / 60.
C
C
C  Call TRANS to Calculate Transition Matrix
C
      CALL TRANST
C
      RETURN
      END
**********************************************************************
      SUBROUTINE ECHO
**********************************************************************
* Echo parameter values to file to verify that the input was
* interpreted correctly

      include drvmod.inc
      include GLBL.inc
      include PARS.inc
      include MNVR.inc
      include SUSP.inc
      include TIRE.inc
      include AERO.inc
      include PRNT.inc
      include VARS.inc
      CHARACTER*32 FNECHO
      CHARACTER*24 TIMEDT
      LOGICAL ISIT
C
C Get name of echo file from user.  Delete old file if it exists.
C
      FNECHO = ' '
      WRITE(*, '(A\)') ' Name of (optional) parameter echo file: '
      READ(*, '(A)') FNECHO
      IF (FNECHO .EQ. ' ') THEN
        RETURN
      ELSE
        INQUIRE (FILE=FNECHO, EXIST=ISIT)
        IF (ISIT) THEN
          OPEN (IECHO, FILE=FNECHO)
```

```
          CLOSE (IECHO, STATUS='DELETE')
        END IF
        OPEN (IECHO, FILE=FNECHO, STATUS='NEW')
      END IF

      WRITE(IECHO,'(A/)')
     &' ECHO FROM WIND/HANDLING SIMULATION, V1.4'
      WRITE(IECHO, '(A, A/)') ' Input file: ', FNREAD

      CALL TIMDAT (TIMEDT)
      WRITE (IECHO, '(A,A/)') ' Run made at ', TIMEDT

      WRITE(IECHO,'(A,A/)') ' TITLE: ', TITLE
      WRITE(IECHO,'(A/)')
     &     ' GENERAL SIMULATION INFORMATION:'
      IF (UNITS .EQ. 'E' .OR. UNITS .EQ. 'e') THEN
        WRITE(IECHO,'(T5, A)') 'English Units'
      ELSE
        WRITE(IECHO,'(T5, A)') 'Metric Units'
      END IF
      WRITE(IECHO, '(T5,A,A)') 'Output format: ', FRMT

      WRITE(IECHO,'(T5,''V, TEND, DT:'',T30, 3G14.5)') V, TEND, DT
      IF (IPRINT .EQ. 1) THEN
        WRITE(IECHO, '(T5,A)') 'Write to file every time step'
      ELSE
        WRITE(IECHO, '(T5,A,I2,A)') 'Write to file every ', IPRINT,
     &  ' steps'
      END IF
      WRITE(IECHO,'(T5, ''KSYWND, AIRHO:'', T30, 2G14.5)')
     1          KSYWND, AIRHO
      IF(WINDKY .GE. 0) THEN
         WRITE(IECHO,'(/A/)') ' WIND MAGNITUDE TIME HISTORY INPUT:'
         DO 32, J=1, WINDKY
            WRITE(IECHO, '(3X, 2G14.5)') TWIND(J), WINMAG(J)
   32    CONTINUE
      ELSE
        WRITE(IECHO,'(/A/)')
     &     ' Wind input defined by user function FWIND'
      ENDIF
C
      IF (NSTEER .EQ. 0) THEN
        WRITE(IECHO,'(/A/)') ' SINUSOIDAL STEER:'
        WRITE(IECHO,'(T8,''TSWBGN, TSWEND:'',T30,2G14.5)') TSWBGN,
     &     TSWEND
        WRITE(IECHO,'(T8,''SWSHFT, SWAMPL:'',T30,2G14.5)') SWSHFT,
     &     SWAMPL
        WRITE(IECHO,'(T8,''TSWPRD, SWPHSE:'',T30,2G14.5)') TSWPRD,
     &     SWPHSE
      ELSE IF (NSTEER .LT. 0 .AND. NSTEER .GT. -100) THEN
        WRITE(IECHO,'(/A/)') ' DRIVER MODEL INPUT:'
        WRITE(IECHO,'(T5,''DRLAG, DRPREV:'',T30,2G14.5)') TAUMEM, TFF
        WRITE(IECHO,'(/T5,A/)') 'X-Y path coordinates:'
        DO 35, J=1, ABS(NSTEER)
          WRITE(IECHO, '(3X, 2G14.5)') XPDR(J), YPDR(J)
   35   CONTINUE
      ELSE
        IF (NEQN .EQ. 11) THEN
```

```
         WRITE(IECHO,'(/A/)') ' STEER TABLE - time(sec), sw(deg):'
         DO 40, J=1, ABS(NSTEER)
           WRITE(IECHO, '(3X, 2G14.5)') XPNT(J), YPNT(J)
   40    CONTINUE
       ELSE
         IF(NSTEER .GT.-100) THEN
         WRITE(IECHO,'(/A/)') ' STEER TORQUE TABLE - time(sec), storq
     & (in-lbs):'
         DO 42, J=1, ABS(NSTEER)
           WRITE(IECHO, '(3X, 2G14.5)') XPNT(J), YPNT(J)
   42    CONTINUE
         ENDIF
       ENDIF
      END IF
C
C   Total vehicle and sprung mass parameters:
C
      WRITE(IECHO,'(/A/)') ' TOTAL VEHICLE AND SPRUNG MASS PARAMETERS:'
      WRITE(IECHO,'(T5, ''WEIGHT, SPWGHT, WRATIO:'', T30, 3G14.5)')
     1          WEIGHT, SPWGHT, WRATIO
      WRITE(IECHO,'(T5, ''IXSCG, IYS, IZZ, IXZ:'', T30, 4G14.5)')
     1          IXSCG, IYS, IZZ, IXZ
      WRITE(IECHO,'(T5, ''WB, WHLRAD, HCGTTL:'', T30, 3G14.5)')
     1          WB, WHLRAD, HCGTTL
C
C   Aerodynamic parameters:
C
      WRITE(IECHO,'(/A/)') ' AERODYNAMIC PARAMETERS:'
      WRITE(IECHO,'(T5, ''AREA:'', T30, G14.5)') AREA
      WRITE(IECHO,'(T5, ''KY, KR, KN:'', T30, 3G14.5)') KY, KR, KN
      WRITE(IECHO,'(T5, ''CL0, KL:'', T30, 2G14.5)') CL0, KL
      WRITE(IECHO,'(T5, ''CM0, KM:'', T30, 2G14.5)') CM0, KM
      WRITE(IECHO,'(T5, ''CD0, KD:'', T30, 2G14.5)') CD0, KD
C
C   Steering system:
C
      WRITE(IECHO,'(/A/)') ' STEERING SYSTEM:'
      WRITE(IECHO,'(T5, ''ISS, KSC, DLASH, KSL:'', T30, 4G14.5)') ISS,
     & KSC, DLASH, KSL
      WRITE(IECHO,'(T5, ''GR, XTRAIL, CSS, SWSTOP:'', T30, 4G14.5)') GR,
     & XTRAIL, CSS, SWSTOP
      WRITE(IECHO,'(T5, ''CBOOST, SSKEY, CFSS:'', T30, 3G14.5)')
     1          CBOOST, SSKEY, CFSS
C
      IF (KINEM) THEN
        WRITE (IECHO, '(/A)')
     &     ' NKINEM <> 0 -- Use full kinematics model'
      ELSE
        WRITE (IECHO, '(/A)')
     &     ' NKINEM = 0 -- Use simple kinematics model'
      END IF
C
      IF (BEAM) THEN
        WRITE (IECHO, '(/A)') ' BEAM <> 0 -- Beam rear suspension'
      ELSE
        WRITE (IECHO, '(/A)') ' BEAM = 0 -- Independent rear suspension'
      END IF
C
```

```
      DO 80, NAXLE=1, 2
C
C    Suspension and tire data:
C
        WRITE(IECHO,'(/'' AXLE NUMBER'', I2,
     1                //T5,''Suspension and tire data'')') NAXLE
        WRITE(IECHO,'(T7, ''TRACK, H0ROLC:'', T30, 2G14.5)')
     1            TRACK(NAXLE), H0ROLC(NAXLE)
        WRITE(IECHO,'(T7, ''KZ, KAUX:'', T30, 2G14.5)')
     1            KZ(NAXLE), KAUX(NAXLE)
        WRITE(IECHO,'(T7, ''CZJNCE, CZRBND:'', T30, 2G14.5)')
     1            CZJNCE(NAXLE), CZRBND(NAXLE)
        WRITE(IECHO,'(T7, ''ALFA0, GAMMA0:'', T30, 2G14.5)')
     1            ALFA0(NAXLE), GAMMA0(NAXLE)
C
C       Kinematic coefficients:
C
        IF (KINEM) THEN
          WRITE(IECHO,'(/T5,A)') 'Kinematic coefficients:'
          WRITE(IECHO,'(T7, A, T30, 2G14.5)')
     &         'YROLCF:', YROLCF(NAXLE,1), YROLCF(NAXLE,2)
          WRITE(IECHO,'(T7, A, T30, 2G14.5)')
     &         'HROLCF:',  HROLCF(NAXLE,1), HROLCF(NAXLE,2)
          IF (BEAM .AND. NAXLE .EQ. 2) THEN
            WRITE(IECHO,'(T7, A, T30, G14.5)')
     &             'Rear axle roll steer: ', CSROLL
          ELSE
            WRITE(IECHO,'(T7, A, T30, 2G14.5)')
     &              'CSZ: ', CSZ(NAXLE,1), CSZ(NAXLE,2)
          END IF
          WRITE(IECHO,'(T7, A, T30, 2G14.5)')
     &           'CCZ: ', CCZ(NAXLE,1), CCZ(NAXLE,2)
        END IF
C
C       Compliance coefficients:
C
        WRITE(IECHO,'(/T5,A)') 'Compliance coefficients:'
        WRITE(IECHO,'(T7, ''CSFY, CSMZ, CCFY:'', T30, 3G14.5)')
     1            CSFY(NAXLE), CSMZ(NAXLE), CCFY(NAXLE)
C
C       Tire coefficients (positive stiffness input values assumed):
C
        WRITE(IECHO,'(/T5,A)') 'Tire stiffness coefficients:'
        WRITE(IECHO,'(T7, ''CALFA:'', T16, 4G14.5)')
     1           (CALFA(J,NAXLE),J=1,4)
        WRITE(IECHO,'(T7, ''CGAMMA:'', T16, 4G14.5)')
     1           (CGAMMA(J,NAXLE),J=1,4)
        WRITE(IECHO,'(T7, ''CALIGN:'', T16, 4G14.5)')
     1           (CALIGN(J,NAXLE),J=1,4)
        WRITE(IECHO,'(T7, ''KTIRE:'', T16, G14.5)') KTIRE(NAXLE)
   80 CONTINUE
C
C
C   Aerodynamic Center of Pressure, Vehicle Mass Center, & Neutral Steer
C   Point Calculations:
C
       FZ(1,1) = .5 * WEIGHT * WRATIO
       FZ(1,2) = .5 * WEIGHT * WRATIO
```

```
      FZ(2,1) = .5 * WEIGHT * (1. - WRATIO)
      FZ(2,2) = .5 * WEIGHT * (1. - WRATIO)
C
      CA1 =  POLY4(CALFA(1,1), FZ(1,1))
      CA2 =  POLY4(CALFA(1,2), FZ(2,1))
C
      AA = WB * (1. - WRATIO)
      BB = WB * WRATIO
C
      REDUC1 = 1.0
      REDUC2 = 1.0
C
C  Check for steering system usage:
C
      IF(SSKEY .NE. 0.0) THEN
C
C  Pneumatic + Mechanical trail, total steering system compliance:
C
        XP =  POLY4(CALIGN(1,1), FZ(1,1)) / POLY4(CALFA(1,1), FZ(1,1))
        XPM = XP + XTRAIL
C
C  Power Boost (cboost is percentage/100 contribution by pump):
C  cboost = 0 -> no power steering boost
C
        CB = 1. - CBOOST
C
C  Reduction factor to reflect presence of steering system
C  for a non-rolling vehicle:   (include both tires => 2.0 factor)
C
        REDUC1 = KSS / ( KSS + 2.0 * CA1 * XPM * CB )
C
C  End of steering system-related parameter calculations.
C
      ENDIF
C
C  Reduction factor to reflect rolling & tire camber:
C
C    Serial spring (suspension & tire):
C
        RKFP = 1. / (1. / KZ(1) + 1. / KTIRE(1))
        RKRP = 1. / (1. / KZ(2) + 1. / KTIRE(2))
C
C  Roll stiffness of vehicle:
C
        RKPHI = (RKFP * (TRACK(1))**2 + RKRP * (TRACK(2))**2)/ 2./57.3
     &  + KAUX(1) + KAUX(2)
C
        REDUC2 = 1.0 - (HCGTTL * WEIGHT /(RKPHI * FZ(1,1)) + CCFY(1)) *
     &  POLY4(CGAMMA(1,1), FZ(1,1)) + HCGTTL * WEIGHT * TRACK(1) /
     &  (2. * RKPHI * 57.3)
     &  * CA1 / FZ(1,1) * CSZ(1,1) + CA1 * CSFY(1) -
     &  POLY4(CGAMMA(1,1), FZ(1,1)) * HCGTTL * WEIGHT * TRACK(1) /
     &  (2. * RKPHI * 57.3 * FZ(1,1)) * CCZ(1,1)
C
C  TOTAL Front Reduction:
C
        REDUCF = REDUC1 * REDUC2
C
```

```
C  Effective front tire cornering stiffness following reductions:
C
         CA1 = CA1 * REDUCF
C
C  Rear Reduction due to aligning torque compliance:
C  (need to add rear roll ster effect still) 4/24/89
C
         XPR = POLY4(CALIGN(1,2), FZ(2,1)) / POLY4(CALFA(1,2), FZ(2,1))
         REDUC3 = 1.0 / (1.0 + CSMZ(2) * CA2 * XPR)
C
C  Rear Reduction due to camber & Fy compliances:
C
         REDUC4 = 1.0 - CCFY(2) * POLY4(CGAMMA(1,2), FZ(2,1))
     &   + CA2 * CSFY(2)
C
C  TOTAL Rear Reduction:
C
         REDUCR = REDUC3 * REDUC4
C
C  Effective rear tire cornering stiffness following reductions:
C
         CA2 = CA2 * REDUCR
C
C  c.g. to neutral steer pt (<0 for understeer, or, nsp behind c.g.) :
C
       XCGNS = (AA * CA1 - BB * CA2) / (CA1+CA2)
C
C  Center of Pressure (CP) ahead of mid-wheelbase:
C
       XL2CP = KN / KY * WB
C
C  c.g. ahead of mid-wheelbase:
C
       XL2CG = (BB - AA) / 2.
C
C  mid-wheelbase pt to neutral steer pt (>0 => ahead of L/2):
C
       XL2NS = XL2CG + XCGNS
C
C  neutral steer pt to center of pressure (>0 => cp ahead of nsp):
C
       XNSCP = XL2CP - XL2CG - XCGNS
C
C  Understeer gradient (deg/g):
C
       UNDSTR = FZ(1,1) / CA1 - FZ(2,1) / CA2
C
C
C  Passive Wind Sensitivity (Rss & its approx due to constant crosswind)
C  of 10 degrees of aero sideslip:
C  First calculate crosswind force due to 1 degree of wind sideslip:
C
       UU = V * 88. / 60.
       FA = 0.5 * AIRHO * UU**2 * KY * AREA
C
C  WSENS = Rss (due to 1 degree of crosswind)
C   (vehicle reference area differences accounted for in FA)
C
```

```
      WSENS = 1. / (57.3 * 2.* CA1 * CA2 / (CA1 + CA2) * (AA + BB)**2 /
     & (12**2) / UU + WEIGHT / 32.2 * UU * (-XCGNS/12.) ) * XNSCP / 12.
     & * 57.3 * FA
C
C  Simplified wind sensitivity (WSIMP):
C
      WSIMP = 1. / (WEIGHT / 32.2 * UU * (-XCGNS / 12.) ) *
     & XNSCP / 12. * 57.3 * FA
C
C
      WRITE(IECHO,'(/T5,A)') '*** Summary Calculations ***'
      WRITE(IECHO,'(/T5, ''Front C-alpha (lb/deg):'', T60, F10.3)') CA1
      WRITE(IECHO,'(T5, ''Rear  C-alpha (lb/deg):'', T60, F10.3)') CA2
      IF(SSKEY .NE. 0.0) THEN
      WRITE(IECHO,'(T5, ''Pneumatic & mechanical trail (in):'', T60,
     & F10.3)') XPM
      WRITE(IECHO,'(T5, ''Steering system stiffness (in-lb/deg):'', T60,
     & F10.3)') KSS
      WRITE(IECHO,'(T5, ''Steering Syst Compl Reduction Factor:'', T60,
     & F10.3)') REDUC1
      ENDIF
      WRITE(IECHO,'(T5, ''Front Rolling & Camber Reduction Factor:'',
     & T60, F10.3)') REDUC2
      WRITE(IECHO,'(T5, ''Total Front C-alpha Reduction Factor:'', T60,
     & F10.3)') REDUCF
      WRITE(IECHO,'(T5, ''Rear Camber & Fy Compl Reduction Factor:'',
     & T60, F10.3)') REDUC3
      WRITE(IECHO,'(T5, ''Rear Align Torque Compl Reduction Factor:'',
     & T60, F10.3)') REDUC4
      WRITE(IECHO,'(T5, ''Total Rear C-alpha Reduction Factor:'', T60,
     & F10.3)') REDUCR
        WRITE(IECHO,'(T5, ''Understeer Gradient (deg/g):'',
     & T60, F10.3)') UNDSTR
      WRITE(IECHO,'(T5, ''Distance from c.g FORWARD to neutral steer pt
     & (in),  XCGNS:'', T70, F10.3)') XCGNS
      WRITE(IECHO,'(T5, ''Distance from WB/2 FORWARD to center of
     & pressure (in), XL2CP:'', T70, F10.3)') XL2CP
      WRITE(IECHO,'(T5, ''Distance from WB/2 FORWARD to c.g (in),
     & XL2CG:'', T70, F10.3)') XL2CG
      WRITE(IECHO,'(T5, ''Distance from WB/2 FORWARD to neutral
     & steer pt (in), XL2NS:'', T70, F10.3)') XL2NS
      WRITE(IECHO,'(T5, ''Distance from neut steer pt FORWARD to
     & cent of press (in), XNSCP:'', T70, F10.3)') XNSCP
      WRITE(IECHO,'(T5, ''Wind Sensitivity (deg/s of yaw rate
     & / deg of wind sideslip), WSENS:'',  T70, F10.3)') WSENS
      WRITE(IECHO,'(T5, ''SIMPLIFIED (less accurate) Wind Sensitivity
     & Measure, WSIMP:'', T70, F10.3)') WSIMP
C
      CLOSE(IECHO)
      RETURN
      END
*********************************************************************
      SUBROUTINE FDAMP(VZ, VROLL, VPITCH, FD)
*********************************************************************
C   SUBROUTINE FDAMP RETURNS FD, THE DAMPING FORCE ACTING AT EACH WHEEL
C   -- ACCOUNTING FOR SEPARATE JOUNCE AND REBOUND COEFFICIENTS.
C   POLARITY:      NET JOUNCE VELOCITY  ==> POSITIVE FD
C   ---------      NET REBOUND VELOCITY ==> NEGATIVE FD
```

```
C
      IMPLICIT REAL (K,M)
      REAL FD(2,2)
C
      include SUSP.inc
C
      DO 20, NAXLE = 1, 2
        DO 10, NSIDE = 1, 2
          VDAMP = VZ - XAXLE(NAXLE) * VPITCH
     1            + .5 * TRACK(NAXLE) * VROLL * (-1)**NSIDE
          IF (VDAMP .GT. 0.0) THEN
            FD(NAXLE,NSIDE) = CZJNCE(NAXLE) * VDAMP
          ELSE
            FD(NAXLE,NSIDE) = CZRBND(NAXLE) * VDAMP
          END IF
   10   CONTINUE
   20 CONTINUE
C
      RETURN
      END
**********************************************************************
      SUBROUTINE FUNCTN(T, Y, YP)
**********************************************************************
C   SUBROUTINE  FUNCTN  DEFINES THE EQUATIONS OF MOTION FOR THE 5 D.O.F
C   VEHICLE + THE 2 D.O.F STEERING SYSTEM (A SECOND ORDER SYSTEM FOR
C   THE STEERING WHEEL INERTIA/COLUMN AND FIRST-ORDER SYSTEM FOR THE
C   LOWER WHEEL ROTATIONAL MOTION (NO WHEEL INERTIA):
C   YP(I) = F(Y, T), WHERE Y, YP ARE VECTORS, AND YP(I) = DY(I)/DT.
C   STATE-VECTOR Y AND T ARE PASSED TO  FUNCTN, AND VECTOR YP RETURNED.
C
      SAVE VSW1, VSW2
      IMPLICIT REAL (K,M)
      REAL IXSRA, Y(13), YP(13)
C
      include GLBL.inc
      include PARS.inc
      include SUSP.inc
      include AERO.inc
      include VARS.inc
      include TIRE.inc
      DATA VSW1, VSW2 /2*0.0/
      DATA ALF1, ALF2, ALF1PR, ALF2PR /4 * 0.0/
C
C
C   CONVERT VECTOR Y INTO NAMES
C
      XG     = Y(1)
      YG     = Y(2)
      Z      = Y(3)
      ROLL   = Y(4)
      PITCH  = Y(5)
      YAW    = Y(6)
      VROLL  = Y(7)
      VPITCH = Y(8)
      VYAW   = Y(9)
      BETA   = Y(10)
      VZ     = Y(11)
C
```

```
C  Steering System STATE Variables:
C
      IF (NEQN .EQ. 13) THEN
        VSW     = Y(12)
        SW      = Y(13)
      ENDIF
C
C  GET CURRENT STEERING WHEEL ANGLE OR STEERING WHEEL TORQUE
C  CONTROL INPUTS:  (depending upon inclusion, or not, of steering sys)
C
      CONTRL = STEER(T)
      IF (NEQN .EQ. 11) THEN
        SW = CONTRL
      ENDIF
C
      IF (NEQN .EQ. 13) THEN
        STORQ = CONTRL
      ENDIF
C
C  CALCULATE CURRENT GEOMETRY AND FORCES
C
      CALL ROLLAX(ROLL, YROLAX, HROLAX, IXSRA)
      CALL FDAMP(VZ, VROLL, VPITCH, FD)
      CALL WHEELZ(Z, ROLL, PITCH)
      CALL TIRES(BETA, V, VYAW, ROLL)
      CALL AIRACT(T, YAW, BETA, VYAW)
C
C                        EQUATIONS OF MOTION
C
C   (I)  ROLL MOMENT:
C
      MSHR = SPMASS * HROLAX
      SUMFY =. SUM(FY) + FYA
      SUMMZ = SUM(MZ) + MZA + XAXLE(1)*(FY(1,1) + FY(1,2))
     &                     + XAXLE(2)*(FY(2,1) + FY(2,2))

      AROLL = (SPWGHT * YROLAX - IXZ / IZZ * SUMMZ + MXA
     &        - KROLL * ROLL - MSHR / MASS * SUM(FY)
     &        + .5 * TRACK(1)*(FD(1,1) - FD(1,2))
     &        + .5 * TRACK(2)*(FD(2,1) - FD(2,2)))
     &        / (IXSRA - MSHR * MSHR / MASS - IXZ*IXZ/IZZ)
C
C   (II)  LATERAL FORCE:
C
      VBETA = (SUMFY - MSHR * AROLL) / (MASS * V) - VYAW
C
C   (III)  YAW MOMENT:
C
      AYAW = (SUMMZ - IXZ * AROLL) / IZZ
C
C   (IV)  VERTICAL FORCE:
C
      AZ = (WEIGHT + FZA - SUM(FZ)) / SPMASS
C
C   (V)  PITCH MOMENT:
C
      APITCH = (XAXLE(1) * (KZAXLE(1) * (Z - XAXLE(1) * PITCH)
     &         + FD(1,1) + FD(1,2))
```

```
     &          + XAXLE(2) * (KZAXLE(2) * (Z - XAXLE(2) * PITCH)
     &          + FD(2,1) + FD(2,2)) + MYA) / IYS
C
C
C
C   (VI)  POWER-STEERING SYSTEM (Lash & Coulomb Friction included):
C
      IF (NEQN .EQ. 13) THEN
C
      XP1 = - POLY4(CALIGN(1,1), FZ(1,1)) / POLY4(CALFA(1,1), FZ(1,1))
      XP2 = - POLY4(CALIGN(1,1), FZ(1,2)) / POLY4(CALFA(1,1), FZ(1,2))
      XP = (XP1 + XP2) * 0.5

      CA1 = - POLY4(CALFA(1,1), FZ(1,1))
      CA2 = - POLY4(CALFA(1,1), FZ(1,2))
      CA = (CA1 + CA2) * 0.5

      XPM = XP + XTRAIL
C
C  Power Boost (cboost is percentage/100 contribution by pump):
C  cboost = 0 -> no power steering boost
C
      CB = 1. - CBOOST

      EXPR = 1. + 2. * XPM * CB * (CA / TODEG) / KSS

      ASW = (STORQ - KSS / (GR**2) * ((1. - 1. / EXPR) * SW - 2. * XPM
     & * CA * GR * CB * (BETA + XAXLE(1) * VYAW / V) / (EXPR * KSS)) )
     & / ISS - CSS * VSW / ISS
C
      ASW = ASW * TODEG
C
C          Update column "wrap-up" torque, mmcol = m - iss * asw:
C          (measured in tests)
C
                  MMCOL = STORQ - ISS * ASW / TODEG
C
C
C     Add coulomb friction and check for polarity change:
C
      IF(ABS(VSW) .GT. 0.01 .AND. VSW2 .NE. 0.0) THEN
          ASW = ASW - SIGN( (CFSS / ISS * TODEG), VSW)
          IF(SIGN(1.,VSW) .NE. SIGN(1.,VSW1) .AND.
     &        SIGN(1.,VSW) .EQ. SIGN(1.,VSW2) ) THEN
               VSW = 0.0
               ASW = 0.0
               VSW1 = 0.0
               VSW2 = 0.0
               Y(14) = 0.0
          ENDIF
      ELSE
         IF(ABS(ASW) .GT. (CFSS / ISS * TODEG)) THEN
            ASW = ASW - SIGN( (CFSS / ISS * TODEG), ASW)
       ELSE
           ASW = 0.0
         ENDIF
      ENDIF
C
```

```
      VSW2 = VSW1
      VSW1 = VSW
C
C  Check for violation of steering wheel stop (limit) setting:
C
      IF(ABS(SW) .GT. SWSTOP) THEN
        SW = SIGN(SWSTOP, SW)
        VSW = 0.0
        ASW = 0.0
      ENDIF
C
C  Front Wheel Angles:
C
      FW(1) = SW / GR / (1. + (XP1 + XTRAIL) * CB * CA1 / TODEG / (KSS
     & / 2.)) + (XP1 + XTRAIL) * CA1 * CB / (KSS / 2.) * (BETA  +
     & XAXLE(1) * VYAW / V) / (1. + (XP1 + XTRAIL) * CB * CA1 / TODEG
     & / (KSS / 2.) )

      FW(2) = SW / GR / (1. + (XP2 + XTRAIL) * CB * CA2 / TODEG / (KSS
     & / 2.)) + (XP2 + XTRAIL) * CA2 * CB / (KSS / 2.) * (BETA  +
     & XAXLE(1) * VYAW / V) / (1. + (XP2 + XTRAIL) * CB * CA2 / TODEG
     & / (KSS / 2.) )
C
C  Include the lash (deg):
C
        IF(ABS(DLASH) .GT. 0.001) THEN
          ALF1PR = (BETA + XAXLE(1) * VYAW / V) * TODEG - FW(1)
          IF(ABS(ALF1PR) .GT. DLASH) THEN
            ALF1 = ALF1PR - SIGN(DLASH, ALF1PR)
            FW(1) = BETA + XAXLE(1) * VYAW / V - ALF1 / TODEG
          ELSE
            FW(1) = BETA + XAXLE(1) * VYAW / V
          ENDIF
C
          ALF2PR = (BETA + XAXLE(1) * VYAW / V) * TODEG - FW(2)
          IF(ABS(ALF2PR) .GT. DLASH) THEN
            ALF2 = ALF2PR - SIGN(DLASH, ALF2PR)
            FW(2) = BETA + XAXLE(1) * VYAW / V - ALF2 / TODEG
          ELSE
            FW(2) = BETA + XAXLE(1) * VYAW / V
          ENDIF
C
        ELSE
C
C  no lash:  (to radians)
C
          FW(1) = FW(1) / TODEG
          FW(2) = FW(2) / TODEG
C
        ENDIF
C
C
C  End of steering system calculations.
C
      ENDIF
C
C
C  INERTIAL DISPLACEMENTS OF TOTAL CG:
```

```
C
      VDIR = YAW + BETA
      VXG = V * COS(VDIR)
      VYG = V * SIN(VDIR)
C
C   LATERAL ACCELERATION OF TOTAL CG (W/O CONTRIBUTION OF ROLL-ACCEL.):
C
      AY = (VYAW + VBETA) * V / G
C
C   Path curvature:
C
      RHO = (VBETA + VYAW) / V
C
C   Convert names into array YP
C
      YP(1)  = VXG
      YP(2)  = VYG
      YP(3)  = VZ
      YP(4)  = VROLL
      YP(5)  = VPITCH
      YP(6)  = VYAW
      YP(7)  = AROLL
      YP(8)  = APITCH
      YP(9)  = AYAW
      YP(10) = VBETA
      YP(11) = AZ
C
C  Steering System STATE Variables:
C
      IF (NEQN .EQ. 13) THEN
        YP(12) = ASW
        YP(13) = VSW
      ENDIF
C
C   Copy array Y into common block for use by driver model
C
      DO 150, J = 1, 13
        YOUTDR(J) = Y(J)
  150 CONTINUE
C
      RETURN
      END
************************************************************************
      FUNCTION FWIND (T)
      SAVE
************************************************************************
C  This function is an optional user-defined subroutine used to
C  calculate or define a wind profile in lieu of entering time history
C  wind profiles.  It is called when the WINDKY parameter is entered as
C  a negative integer;  a positive entry for WINDKY forces a table
C  look-up instead.
C
C  Time, T, is passed to the subroutine;  the wind magnitude, FWIND, is
C  returned.
C
      include GLBL.inc
C
C  (user-defined code)
```

```
C
*  Current wind:  random number generator passed through a first order
*  filter.  This scheme makes the rms of the wind sensitive to the
*  integration step size.  Currently "sized" for DT = 0.02 seconds.
*
      DATA XX, YY, VV, XL, YL, TL /6*0.0/
*
*  Set dc component of wind (mph):
*
      DC = 20.0
*
*  Initialize fwind filter:
*
      IF(T .LE. 0.0) THEN
        XX = RAN3(-1) - 0.5
        CALL FILTER(T, XX, YY, VV, XL, YL, TL)
      ENDIF
*
      XX = RAN3(1) - 0.5
      CALL FILTER(T, XX, YY, VV, XL, YL, TL)
      FWIND = YY + DC
      RETURN
      END
C*********************************************************************
C*********************************************************************
C
C  *** Matrix Product  Subroutine ***
C
C GMPRD: Computes matrix product
C
C===========Author and Modification Section===========================
C
C  Author:       IBM Scientific Subroutine
C
C  Date written:
C
C  Written on:
C
C  Modifications:  C. MacAdam
C
C=====================================================================
C
C===========Algorithm Description=====================================
C
C  Purpose and use:  R = A B
C
C  Error conditions:
C
C  Machine dependencies: none
C
C  Called By: DRIVER
C
C=====================================================================
C
      SUBROUTINE GMPRD(A, B, R, N, M, L)
C
C===========Variable Descriptions=====================================
C
```

```
C---Arguments passed:
C
C   A.....N x M matrix
C   B.....M x L matrix
C   R.....N x L resultant matrix = A B product
C   N.....integer row dimension of A
C   M.....integer column dimension of A (or row dimension of B)
C   L.....integer column dimension of B
C
      DIMENSION A(N*M), B(M*L), R(N*L)
C
C---COMMON blocks-------------------------------------------------------
C
C     None
C
C---COMMON Variables----------------------------------------------------
C
C     None
C
C---Local variables-----------------------------------------------------
C
C  IR, IK, M, K, L, IR, JI, J, N, IB, IK, etc ......integer counters
C
C---Functions and subroutines-------------------------------------------
C
C      None
C
C======================================================================
C
C===========Process Block===============================================
C
      IR = 0
      IK = -M
      DO 10 K = 1, L
        IK = IK + M
        DO 10 J = 1, N
          IR = IR + 1
          JI = J - N
          IB = IK
          R(IR) = 0.
          DO 10 I = 1, M
            JI = JI + N
            IB = IB + 1
   10 R(IR) = R(IR) + A(JI) * B(IB)
      RETURN
      END
**********************************************************************
      SUBROUTINE INDATA
**********************************************************************
C   (1) Get file names from the user,
C   (2) connect the files to their Fortran i/o units,
C   (3) read the dataset from unit IREAD,
C   (4) echo the parameter values to unit IECHO,
c   (5) and, perform the necessary conversions of physical units.
C
      IMPLICIT REAL (K,M)
      LOGICAL ISIT
C
```

```
      include drvmod.inc
      include GLBL.inc
      include PARS.inc
      include MNVR.inc
      include SUSP.inc
      include TIRE.inc
      include AERO.inc
      include PRNT.inc
C
C   Get input file name from user
C
      WRITE(*,'(///A/A/A/A/A/A)')
     &     ' CHRYSLER-UMTRI CROSSWIND STABILITY PROJECT',
     &     ' WIND / STEER SIMULATION - Version 1.4, April 1990',' ',
     &     ' Copyright (c) The Regents of The University of Michigan',
     &     ' 1987-1990, Ann Arbor, Michigan.  All Rights Reserved.',' '
C
  100 WRITE(*, '(A\)') ' Name of input file: '
      READ(*, '(A)') FNREAD
      INQUIRE (FILE=FNREAD, EXIST=ISIT)

      IF (.NOT. ISIT) THEN
        WRITE (*, '(A, A, A)') ' File "', FNREAD,
     &                 '" does not exist.  Try again.'
        GO TO 100
      END IF
      OPEN(IREAD, ERR=100, STATUS='OLD', FILE=FNREAD)
C
C   Read general simulation and maneuver parameters:
C
      READ(IREAD,'(//A)') TITLE
      READ(IREAD,'(A)') UNITS
      READ(IREAD,'(A)') FRMT
      DO 3 I=1,10
        IF (FRMT(I:I) .NE. ' ') THEN
          FRMT = FRMT(I:)
          GO TO 4
        END IF
    3 CONTINUE
    4 CONTINUE
C
CMD--Use NUMKEY=1 for Mac, 2 for IBM PC
      IF (FRMT(:1) .NE. '(') THEN
        NUMKEY = 1
        FRMT = 'Binary'
      ELSE
        NUMKEY = 5
      END IF
C)
      READ(IREAD,530) V, TEND, DT
      READ(IREAD,520) IPRINT
      READ(IREAD,530) KSYWND, AIRHO
      READ(IREAD,520) WINDKY
      VWIND = 0.0
      IF(WINDKY .GE. 0) THEN
        DO 5 J = 1, WINDKY
          READ(IREAD,530) TWIND(J), WINMAG(J)
    5   CONTINUE
```

```
      ELSE
        VWIND = FWIND(T)
      ENDIF
C
      READ(IREAD,520) NSTEER
      IF (NSTEER .EQ. 0) THEN
        READ(IREAD,530) TSWBGN, TSWEND
        READ(IREAD,530) SWSHFT, SWAMPL
        READ(IREAD,530) TSWPRD, SWPHSE
      ENDIF

      IF (NSTEER .LT. 0 .AND. NSTEER .GT. -100) THEN
        NP = -NSTEER
        CALL DRIVE1(SW)
      ENDIF
      IF(NSTEER .GT. 0) THEN
        DO 10, J=1, ABS(NSTEER)
        READ(IREAD,530) XPNT(J), YPNT(J)
   10   CONTINUE
      ENDIF
C
C   Total vehicle and sprung mass parameters:
C
      READ(IREAD,530) WEIGHT, SPWGHT, WRATIO
      READ(IREAD,540) IXSCG, IYS, IZZ, IXZ
      READ(IREAD,530) WB, WHLRAD, HCGTTL
C
C   Aerodynamic parameters:
C
      READ(IREAD,530) AREA
      READ(IREAD,530) KY, KR, KN
      READ(IREAD,530) CL0, KL
      READ(IREAD,530) CM0, KM
      READ(IREAD,530) CD0, KD
C
C   Steering system:
C
      READ(IREAD,540) ISS, KSC, DLASH, KSL
      READ(IREAD,540) GR, XTRAIL, CSS, SWSTOP
        IF (SWSTOP .EQ. 0.) SWSTOP = 1000.
      READ(IREAD,530) CBOOST, SSKEY, CFSS
C
C  Calculate equivalent single steering system stiffness based on input
C  values for the steering column, steering linkage, and gear ratio:
C
      KSS = 2.*GR*GR*KSC*KSL / (2.*KSL + GR*GR*KSC)
C
C   Suspension and tire data:
C
      READ(IREAD,520) NKINEM
      IF (NKINEM .EQ. 0) KINEM = .FALSE.
      READ(IREAD,520) NBEAM
      IF (NBEAM .EQ. 0) BEAM = .FALSE.
C
      DO 30, NAXLE=1, 2
C
        READ(IREAD,530) TRACK(NAXLE), H0ROLC(NAXLE)
        READ(IREAD,530) KZ(NAXLE), KAUX(NAXLE)
```

```
        READ(IREAD,530) CZJNCE(NAXLE), CZRBND(NAXLE)
        READ(IREAD,530) ALFA0(NAXLE), GAMMA0(NAXLE)
C
C       KINEMATIC COEFFICIENTS:
C
        IF (KINEM) THEN
          READ(IREAD,530) YROLCF(NAXLE,1), YROLCF(NAXLE,2)
          READ(IREAD,530) HROLCF(NAXLE,1), HROLCF(NAXLE,2)
          IF (BEAM .AND. NAXLE .EQ. 2) THEN
            READ(IREAD,530) CSROLL
          ELSE
            READ(IREAD,530) CSZ(NAXLE,1), CSZ(NAXLE,2)
          END IF
          READ(IREAD,530) CCZ(NAXLE,1), CCZ(NAXLE,2)
        END IF
C
C       Compliance coefficients:
C
        READ(IREAD,530) CSFY(NAXLE), CSMZ(NAXLE), CCFY(NAXLE)
C
C       Tire stiffness coefficients:
C
        READ(IREAD,540) (CALFA(J,NAXLE),J=1,4)
        READ(IREAD,540) (CGAMMA(J,NAXLE),J=1,4)
        READ(IREAD,540) (CALIGN(J,NAXLE),J=1,4)
        READ(IREAD,530) KTIRE(NAXLE)
C
   30 CONTINUE
C
      CLOSE(IREAD)
C
C    Change from metric to English units, if specified
C
      IF (UNITS .EQ. 'E' .OR. UNITS .EQ. 'e') THEN
        G = 386.1
        ININFT = 12
        KMHMPH = 0.056818
        TODEG = 180.0 / PI
        UDISP = 'in'
        UDIST = 'ft'
        UANGL = 'deg'
        UVELFT = 'ft/s'
        UOMEGA = 'deg/sec'
        UFORC = 'lb'
        UTORQ = 'in-lb'
      END IF
C
C    General simulation and maneuver parameters:
C
C      Include steering system dynamics only if non-zero damping:
C
      IF (ABS(SSKEY) .LT. 0.001) NEQN = NEQN - 2
C
      WRITE(*,'('' '',A//)') TITLE

      IF (IECHO .GT. 0) CALL ECHO
C
      V = V / KMHMPH
```

```
      VWIND = VWIND / KMHMPH
      KSYWND = KSYWND / TODEG
      GRTODG = GR * TODEG

C
      DO 80, NAXLE=1, 2
        KAUX(NAXLE) = KAUX(NAXLE) * TODEG

C     With english units, SW, KSC, KSL stay in deg, while FW, VFW
C     stay in rad (with metric units, all are in rad, and todeg = 1)
C
        ALFA0(NAXLE) = ALFA0(NAXLE) / TODEG
        GAMMA0(NAXLE) = GAMMA0(NAXLE) / TODEG
        IF (NSTEER .EQ. 0) SWPHSE = SWPHSE / TODEG
C
C Convert polynomial coefficients from deg to rad:
C
        IF (KINEM) THEN
          DO 50, NPOWER = 1, 2
            YROLCF(NPOWER,NAXLE) = YROLCF(NPOWER,NAXLE) * TODEG **
     &                             NPOWER
            HROLCF(NPOWER,NAXLE) = HROLCF(NPOWER,NAXLE) * TODEG **
     &                             NPOWER
            CSZ(NPOWER,NAXLE) = CSZ(NPOWER,NAXLE) / TODEG
            CCZ(NPOWER,NAXLE) = CCZ(NPOWER,NAXLE) / TODEG
   50     CONTINUE
        END IF
C
C       Compliance coefficients:
C
        CSFY(NAXLE) = CSFY(NAXLE) / TODEG
        CSMZ(NAXLE) = CSMZ(NAXLE) / TODEG
        CCFY(NAXLE) = CCFY(NAXLE) / TODEG
C
C       Change CALFA polarity to conform with SAE conventions
c       and convert polynomial coefficients from deg to rad
C
        DO 70, NPOWER = 1, 4
          CALFA(NPOWER,NAXLE)  = -CALFA(NPOWER,NAXLE) * TODEG
          CGAMMA(NPOWER,NAXLE) = CGAMMA(NPOWER,NAXLE) * TODEG
          CALIGN(NPOWER,NAXLE) = CALIGN(NPOWER,NAXLE) * TODEG
   70   CONTINUE
   80 CONTINUE
C
      RETURN
C
  520 FORMAT (BN, I4)
  530 FORMAT (3F12.0)
  540 FORMAT (4F12.0)
C
      END
************************************************************************
      SUBROUTINE INDSUS(X, DFDX, FNEG)
************************************************************************
* This subroutine computes a 2x2 Jacobian and a 2-element error
* array for a tire on an independent suspension.
* It is called in turn by the Newton-Raphson solver, MNEWT.
*
```

```
* --> X      real  2-element array. x(1)=slip, x(2)=camber
* <-- DFDX  real  2x2 array, df/dx (partial derivatives)
* <-- FNEG  real  2 negative error functions in equations
*
* In addition to the arguments, it deals with variables FZTEMP, AXLE,
* and BIAS from the common block /TSOLVE/.  These apply for
* a particular wheel, and are set in the subroutine TIRES.

      IMPLICIT REAL (K,M)
      REAL X(2), DFDX(2,2), FNEG(2)
      include TSOLVE.inc
      include SUSP.inc

      ALPHA = X(1)
      GAMMA = X(2)

      CA = DFYDA(ALPHA, GAMMA, FZTEMP, AXLE)
      CM = DMZDA(ALPHA, GAMMA, FZTEMP, AXLE)
      CG = DFYDG(ALPHA, GAMMA, FZTEMP, AXLE)

* 2x2 Jacobian

      DFDX(1,1) = CSFY(AXLE)*CA + CSMZ(AXLE)*CM + 1.
      DFDX(2,1) = CCFY(AXLE)*CA
      DFDX(1,2) = CSFY(AXLE)*CG
      DFDX(2,2) = CCFY(AXLE)*CG + 1.

* Negative of error function.

      SAVEFY(1) = TIREFY(ALPHA, GAMMA, FZTEMP, AXLE)
      SAVEMZ(1) = TIREMZ(ALPHA, FZTEMP, AXLE)
      FNEG(1) =  -ALPHA + BIAS(1) - CSFY(AXLE)*SAVEFY(1)
     &          - CSMZ(AXLE)*SAVEMZ(1)
      FNEG(2) =  -GAMMA + BIAS(2) - CCFY(AXLE)*SAVEFY(1)
      RETURN
      END
***********************************************************************
      SUBROUTINE INIT
***********************************************************************
C
C   Initialize input-based values and non-zero variables
C
      IMPLICIT REAL (K,M)
      include GLBL.inc
      include PARS.inc
      include SUSP.inc
      include AERO.inc
      include VARS.inc
      include PRNT.inc

      MASS = WEIGHT / G
      SPMASS = SPWGHT / G
      USWGHT = WEIGHT - SPWGHT
      XAXLE(2) = - WB * WRATIO
      XAXLE(1) = WB + XAXLE(2)
      FZ0WHL(1) = .5 * WEIGHT * WRATIO
      FZ0WHL(2) = .5 * WEIGHT * (1 - WRATIO)
      XCGSP = WB * (2 * FZ0WHL(2) - .5 * USWGHT) / SPWGHT
```

```
      XWBCGS = .5 * WB - XCGSP
      XWBCGT = .5 * WB - XAXLE(1)
      HCGSP = (HCGTTL * WEIGHT - WHLRAD * USWGHT) / SPWGHT
      ROLLVR = .5 * TRACK(1) / HCGTTL
      QZERO = AIRHO * AREA / 2
      KROLL = 0.
C
      DO 20, NAXLE = 1, 2
C
C    Approximate effects of tire stiffness + damping in suspension:
C
        TRKSQR = .5 * TRACK(NAXLE)**2
        SUMKZ = KZ(NAXLE) + KTIRE(NAXLE)
C
C    Reduce overall damping coefficients for negligible tire damping:
C
        CZJNCE(NAXLE) = CZJNCE(NAXLE) * KTIRE(NAXLE) / SUMKZ
        CZRBND(NAXLE) = CZRBND(NAXLE) * KTIRE(NAXLE) / SUMKZ
C
C    Total vertical suspension rate at wheel (parallel springs):
C
        KZSSP = KZ(NAXLE) + KAUX(NAXLE) / TRKSQR
C
C    Overall vertical rate (suspension and tire in series):
C
        KZTTL = KZSSP * KTIRE(NAXLE) / (KZSSP + KTIRE(NAXLE))
C
C    KZ <--- overall vertical rate without auxiliary roll stiffness:
C
        KZ(NAXLE) = KZ(NAXLE) * KTIRE(NAXLE) / SUMKZ
C
C    Adjusted auxiliary roll rate (in parallel with kz):
C
        KAUX(NAXLE) = (KZTTL - KZ(NAXLE)) * TRKSQR
C
C    Effective roll stiffness and axle vertical stiffness
C
        KROLL = KROLL + KZ(NAXLE) * TRKSQR + KAUX(NAXLE)
        KZAXLE(NAXLE) = 2 * KZ(NAXLE)
        HCGSRC(NAXLE) = HCGSP - H0ROLC(NAXLE)
        DO 10, NSIDE = 1, 2
          ALFA(NAXLE,NSIDE) = -(-1)**NSIDE * ALFA0(NAXLE)
          GAMMA(NAXLE,NSIDE) = -(-1)**NSIDE * GAMMA0(NAXLE)
          FZ(NAXLE,NSIDE) = FZ0WHL(NAXLE)
          KNMSTR(NAXLE,NSIDE) = 0.0
          KNMCBR(NAXLE,NSIDE) = 0.0
   10   CONTINUE
   20 CONTINUE
      RETURN
      END
**********************************************************************
      FUNCTION LENSTR (STRING)
**********************************************************************
*  count characters in left-justified string.   M. Sayers, 8-9-87

      CHARACTER*(*) STRING
      N = LEN (STRING)
      DO 10 L = N, 1, -1
```

```
        IF (STRING(L:L) .NE. ' ' .AND. STRING(L:L) .NE. char(3)) THEN
          LENSTR = L
          RETURN
        END IF
   10 CONTINUE
      LENSTR = 1
      RETURN
      END
************************************************************************
      SUBROUTINE MNEWT(X,USRFUN,IERR)
************************************************************************
* This routine is based on MNEWT from the Numerical Recipes library. It
* has been "hard-wired" for the Chrysler vehicle handling model.  The
* error threshold is less than .01 deg

      PARAMETER (N=2, NTRIAL=10, TOLX=.0000001, TOLF=.000001)
      DIMENSION X(N),ALPHA(N,N),BETA(N),INDX(N)
      IERR=0
      DO 13  K=1,NTRIAL
        CALL USRFUN(X,ALPHA,BETA)
        ERRF=0.
        DO 11 I=1,N
          ERRF=ERRF+ABS(BETA(I))
11      CONTINUE
        IF(ERRF.LE. TOLF)RETURN
        TMP1 = ALPHA(2,1)/ALPHA(2,2)
        TMP2 = BETA(2)/ALPHA(2,2)
        BETA(1)=   (BETA(1) - ALPHA(1,2)*TMP2)
     &            / (ALPHA(1,1) - ALPHA(1,2)*TMP1)
        BETA(2)=TMP2 - TMP1*BETA(1)

        ERRX=0.
        DO 12 I=1,N
          ERRX=ERRX+ABS(BETA(I))
          X(I)=X(I)+BETA(I)
12      CONTINUE
        IF(ERRX.LE.TOLX)RETURN
13    CONTINUE
      IERR=1
      RETURN
      END
************************************************************************
      SUBROUTINE OPNOUT
************************************************************************
C   SUBROUTINE OPNOUT INITIALIZES THE OUTPUT ERD FILE.
C
      IMPLICIT REAL (K,M)
      CHARACTER*32 LONGNM(66), GENNM(66), RIGBOD(66), THISRB
      CHARACTER*32 FNOUT
      CHARACTER*24 TIMEDT
      CHARACTER*8 SHORTN(66), UNITNM(66)
      CHARACTER*4 LORR(2)
      CHARACTER*1 AXLE(2), SIDE(2)
      INTEGER NCHAN
      LOGICAL ISIT
C
      include GLBL.inc
      include PARS.inc
```

```
C
      DATA AXLE/'1','2'/, SIDE/'L','R'/, LORR/'Left','Rght'/
      DATA TIMEDT/' '/
C
  110 WRITE(*, '(A\)') ' Name of simulation output file: '
      READ(*, '(A)') FNOUT
      IF (FNOUT .NE. ' ') THEN
        INQUIRE (FILE=FNOUT, EXIST=ISIT)
        IF (ISIT) THEN
          OPEN (IOUT, FILE=FNOUT)
          CLOSE (IOUT, STATUS='DELETE')
        END IF
        OPEN (IOUT, FILE=FNOUT, STATUS='NEW')
        WRITE (*,*) ' '
      ELSE
        WRITE (*,*) 'Output file is required!'
        GO TO 110
      END IF
C
C  Start with 0 output channels
C
      NCHAN = 0
C
C  Time
C
      NCHAN = NCHAN + 1
      LONGNM (NCHAN) = 'Time'
      SHORTN (NCHAN) = 'Time'
      GENNM (NCHAN) = 'Time'
      UNITNM (NCHAN) = 'sec'
      RIGBOD (NCHAN) = 'Time'
C
C  Input Steer Angle
C
      NCHAN = NCHAN + 1
      LONGNM (NCHAN) = 'Input Steer Angle'
      SHORTN (NCHAN) = 'Steer in'
      GENNM (NCHAN) = 'Angle'
      UNITNM (NCHAN) = UANGL
      RIGBOD (NCHAN) = 'Input'
C
C  Input Steer Torque & Steering Wheel Velocity
C
      IF (SSKEY .NE. 0.0) THEN
        NCHAN = NCHAN + 1
        LONGNM (NCHAN) = 'Input Steer Torque'
        SHORTN (NCHAN) = 'SW Torq'
        GENNM (NCHAN) = 'Torque'
        UNITNM (NCHAN) = UTORQ
        RIGBOD (NCHAN) = 'Input'
C
        NCHAN = NCHAN + 1
        LONGNM (NCHAN) = 'Steering Wheel Velocity'
        SHORTN (NCHAN) = 'SW Vel'
        GENNM (NCHAN) = 'SW Vel'
        UNITNM (NCHAN) = UOMEGA
        RIGBOD (NCHAN) = 'Input'
      END IF
```

```
C
      THISRB = 'Body'
C
C  X Position
C
      NCHAN = NCHAN + 1
      LONGNM (NCHAN) = 'X Position, Sprung Mass cg'
      SHORTN (NCHAN) = 'X cg'
      GENNM (NCHAN) = 'X Position'
      UNITNM (NCHAN) = UDIST
      RIGBOD (NCHAN) = THISRB
C
C  Y Position
C
      NCHAN = NCHAN + 1
      LONGNM (NCHAN) = 'Y Position, Sprung Mass cg'
      SHORTN (NCHAN) = 'Y cg'
      GENNM (NCHAN) = 'Y Position'
      UNITNM (NCHAN) = UDIST
      RIGBOD (NCHAN) = THISRB
C
C  Z Position
C
      NCHAN = NCHAN + 1
      LONGNM (NCHAN) = 'Z Position, Sprung Mass cg'
      SHORTN (NCHAN) = 'Z cg'
      GENNM (NCHAN) = 'Z Position'
      UNITNM (NCHAN) = UDISP
      RIGBOD (NCHAN) = THISRB
C
C  Roll Angle
C
      NCHAN = NCHAN + 1
      LONGNM (NCHAN) = 'Roll Angle'
      SHORTN (NCHAN) = 'Roll'
      GENNM (NCHAN) = 'Roll'
      UNITNM (NCHAN) = UANGL
      RIGBOD (NCHAN) = THISRB
C
C  Pitch Angle
C
      NCHAN = NCHAN + 1
      LONGNM (NCHAN) = 'Pitch Angle'
      SHORTN (NCHAN) = 'Pitch'
      GENNM (NCHAN) = 'Pitch'
      UNITNM (NCHAN) = UANGL
      RIGBOD (NCHAN) = THISRB
C
C  Yaw Angle
C
      NCHAN = NCHAN + 1
      LONGNM (NCHAN) = 'Yaw Angle'
      SHORTN (NCHAN) = 'Yaw'
      GENNM (NCHAN) = 'Yaw'
      UNITNM (NCHAN) = UANGL
      RIGBOD (NCHAN) = THISRB
C
C  Roll Rate
```

```
C
      NCHAN = NCHAN + 1
      LONGNM (NCHAN) = 'Roll Rate'
      SHORTN (NCHAN) = 'p'
      GENNM (NCHAN) = 'Roll Rate'
      UNITNM (NCHAN) = UOMEGA
      RIGBOD (NCHAN) = THISRB
C
C  Pitch Rate
C
      NCHAN = NCHAN + 1
      LONGNM (NCHAN) = 'Pitch Rate'
      SHORTN (NCHAN) = 'q'
      GENNM (NCHAN) = 'Pitch Rate'
      UNITNM (NCHAN) = UOMEGA
      RIGBOD (NCHAN) = THISRB
C
C  Yaw Rate
C
      NCHAN = NCHAN + 1
      LONGNM (NCHAN) = 'Yaw Rate'
      SHORTN (NCHAN) = 'r'
      GENNM (NCHAN) = 'Yaw Rate'
      UNITNM (NCHAN) = UOMEGA
      RIGBOD (NCHAN) = THISRB
C
C  Body Slip Angle
C
      NCHAN = NCHAN + 1
      LONGNM (NCHAN) = 'Vehicle Slip Angle'
      SHORTN (NCHAN) = 'slip'
      GENNM (NCHAN) = 'Angle'
      UNITNM (NCHAN) = UANGL
      RIGBOD (NCHAN) = THISRB
C
C  X Velocity, Sprung Mass cg
C
      NCHAN = NCHAN + 1
      LONGNM (NCHAN) = 'X Velocity, Sprung Mass cg'
      SHORTN (NCHAN) = 'X dot'
      GENNM (NCHAN) = 'X Velocity'
      UNITNM (NCHAN) = UVELFT
      RIGBOD (NCHAN) = THISRB
C
C  Y Velocity, Sprung Mass cg
C
      NCHAN = NCHAN + 1
      LONGNM (NCHAN) = 'Y Velocity, Sprung Mass cg'
      SHORTN (NCHAN) = 'Y dot'
      GENNM (NCHAN) = 'Y Velocity'
      UNITNM (NCHAN) = UVELFT
      RIGBOD (NCHAN) = THISRB
C
C  Z Velocity, Sprung Mass cg
C
      NCHAN = NCHAN + 1
      LONGNM (NCHAN) = 'Z Velocity, Sprung Mass cg'
      SHORTN (NCHAN) = 'w cg'
```

```
      GENNM (NCHAN) = 'Z Velocity'
      UNITNM (NCHAN) = UDISP // '/s'
      RIGBOD (NCHAN) = THISRB
C
C  Lateral Acceleration
C
      NCHAN = NCHAN + 1
      LONGNM (NCHAN) = 'Lateral Acceleration at cg'
      SHORTN (NCHAN) = 'Ay cg'
      GENNM (NCHAN) = 'Lateral Acceleration'
      UNITNM (NCHAN) = 'g''s'
      RIGBOD (NCHAN) = THISRB
C
C  Vehicle Path Curvature
C
      NCHAN = NCHAN + 1
      LONGNM (NCHAN) = 'Vehicle Path Curvature'
      SHORTN (NCHAN) = 'Rho cg'
      GENNM (NCHAN) = 'Vehicle Path Curvature'
      UNITNM (NCHAN) = '1/' // UDIST
      RIGBOD (NCHAN) = THISRB
C
C  Aerodynamic Drag Force
C
      NCHAN = NCHAN + 1
      LONGNM (NCHAN) = 'Aerodynamic Drag Force'
      SHORTN (NCHAN) = 'Fx Aero'
      GENNM (NCHAN) = 'Force'
      UNITNM (NCHAN) = UFORC
      RIGBOD (NCHAN) = THISRB
C
C  Aerodynamic Side Force
C
      NCHAN = NCHAN + 1
      LONGNM (NCHAN) = 'Aerodynamic Side Force'
      SHORTN (NCHAN) = 'Fy Aero'
      GENNM (NCHAN) = 'Force'
      UNITNM (NCHAN) = UFORC
      RIGBOD (NCHAN) = THISRB
C
C  Aerodynamic Down Force
C
      NCHAN = NCHAN + 1
      LONGNM (NCHAN) = 'Aerodynamic Down Force'
      SHORTN (NCHAN) = 'Fz Aero'
      GENNM (NCHAN) = 'Force'
      UNITNM (NCHAN) = UFORC
      RIGBOD (NCHAN) = THISRB
C
C  Aerodynamic Roll Moment
C
      NCHAN = NCHAN + 1
      LONGNM (NCHAN) = 'Aerodynamic Roll Moment'
      SHORTN (NCHAN) = 'Mx Aero'
      GENNM (NCHAN) = 'Moment'
      UNITNM (NCHAN) = UTORQ
      RIGBOD (NCHAN) = THISRB
C
```

```
C  Aerodynamic Pitch Moment
C
      NCHAN = NCHAN + 1
      LONGNM (NCHAN) = 'Aerodynamic Pitch Moment'
      SHORTN (NCHAN) = 'My Aero'
      GENNM (NCHAN) = 'Moment'
      UNITNM (NCHAN) = UTORQ
      RIGBOD (NCHAN) = THISRB
C
C  Aerodynamic Yaw Moment
C
      NCHAN = NCHAN + 1
      LONGNM (NCHAN) = 'Aerodynamic Yaw Moment'
      SHORTN (NCHAN) = 'Mz Aero'
      GENNM (NCHAN) = 'Moment'
      UNITNM (NCHAN) = UTORQ
      RIGBOD (NCHAN) = THISRB
C
C  Air Speed
C
      NCHAN = NCHAN + 1
      LONGNM (NCHAN) = 'Air Speed'
      SHORTN (NCHAN) = 'V Air'
      GENNM (NCHAN) = 'Speed'
      UNITNM (NCHAN) = UVELFT
      RIGBOD (NCHAN) = 'Input'
C
C  Cross Wind
C
      NCHAN = NCHAN + 1
      LONGNM (NCHAN) = 'Crosswind'
      SHORTN (NCHAN) = 'VyWind'
      GENNM (NCHAN) = 'Speed'
      UNITNM (NCHAN) = UVELFT
      RIGBOD (NCHAN) = 'Input'
C
C  Steering Work
C
      NCHAN = NCHAN + 1
      LONGNM (NCHAN) = 'Steering Work'
      SHORTN (NCHAN) = 'StWork'
      GENNM (NCHAN) = 'Speed'
      UNITNM (NCHAN) = 'in-lb'
      RIGBOD (NCHAN) = 'Input'
C
C  Average Steering Power
C
      NCHAN = NCHAN + 1
      LONGNM (NCHAN) = 'Average Steering Power'
      SHORTN (NCHAN) = 'StPower'
      GENNM (NCHAN) = 'Speed'
      UNITNM (NCHAN) = 'in-lb/s'
      RIGBOD (NCHAN) = 'Input'
C
C  Aerodynamic Slip Angle
C
      NCHAN = NCHAN + 1
      LONGNM (NCHAN) = 'Aerodynamic Slip Angle'
```

```
      SHORTN (NCHAN) = 'Slip Air'
      GENNM (NCHAN) = 'Angle'
      UNITNM (NCHAN) = UTORQ
      RIGBOD (NCHAN) = THISRB
C
C   Tire/Wheel variables.  There are 2 nested loops here: the outer
C   indexed the axle, and the inner indexes the side.
C
      DO 100, NAXLE = 1, 2
        DO 80 NSIDE = 1, 2
          THISRB = SIDE(NSIDE) // ' side, Axle ' // AXLE(NAXLE)
C
C  Steer of road wheel
C
          NCHAN = NCHAN + 1
          LONGNM (NCHAN) = 'Total Steer, ' // THISRB
          SHORTN (NCHAN) = SIDE(NSIDE) // ' Str ' // AXLE(NAXLE)
          UNITNM (NCHAN) = UANGL
          GENNM (NCHAN) = 'Angle'
          RIGBOD (NCHAN) = THISRB
C
C  Tire slip angle
C
          NCHAN = NCHAN + 1
          LONGNM (NCHAN) = 'Slip Angle, '// THISRB
          SHORTN (NCHAN) = SIDE(NSIDE) // ' Alph ' // AXLE(NAXLE)
          UNITNM (NCHAN) = UANGL
          GENNM (NCHAN) = 'Angle'
          RIGBOD (NCHAN) = THISRB
C
C  Tire camber angle
C
          NCHAN = NCHAN + 1
          LONGNM (NCHAN) = 'Camber Angle, '// THISRB
          SHORTN (NCHAN) = SIDE(NSIDE) // ' Gamm ' // AXLE(NAXLE)
          UNITNM (NCHAN) = UANGL
          GENNM (NCHAN) = 'Angle'
          RIGBOD (NCHAN) = THISRB
C
C  Tire side force
C
          NCHAN = NCHAN + 1
          LONGNM (NCHAN) = 'Side Force, '// THISRB
          SHORTN (NCHAN) = SIDE(NSIDE) // ' Fy ' // AXLE(NAXLE)
          UNITNM (NCHAN) = UFORC
          GENNM (NCHAN) = 'Force'
          RIGBOD (NCHAN) = THISRB
C
C  Tire Aligning Moment
C
          NCHAN = NCHAN + 1
          LONGNM (NCHAN) = 'Aligning Moment, ' // THISRB
          SHORTN (NCHAN) = SIDE(NSIDE) // ' Mz ' // AXLE(NAXLE)
          UNITNM (NCHAN) = UTORQ
          GENNM (NCHAN) = 'Moment'
          RIGBOD (NCHAN) = THISRB
C
C  Tire vertical force
```

```
C
          NCHAN = NCHAN + 1
          LONGNM (NCHAN) = 'Load, ' // THISRB
          SHORTN (NCHAN) = SIDE(NSIDE) // ' Fz ' // AXLE(NAXLE)
          UNITNM (NCHAN) = UFORC
          GENNM (NCHAN) = 'Force'
          RIGBOD (NCHAN) = THISRB
C
C  Suspension Displacement
C
          NCHAN = NCHAN + 1
          LONGNM (NCHAN) = 'Vert Disp, ' // THISRB
          SHORTN (NCHAN) = SIDE(NSIDE) // ' Z ' // AXLE(NAXLE)
          UNITNM (NCHAN) = UDISP
          GENNM (NCHAN) = 'Displacement'
          RIGBOD (NCHAN) = THISRB
C
C  Suspension Damping Force
C
          NCHAN = NCHAN + 1
          LONGNM (NCHAN) = 'Damping Force, ' // THISRB
          SHORTN (NCHAN) = SIDE(NSIDE) // ' Fdmp ' // AXLE(NAXLE)
          UNITNM (NCHAN) = UFORC
          GENNM (NCHAN) = 'Force'
          RIGBOD (NCHAN) = THISRB
   80   CONTINUE
  100 CONTINUE
C
C   Write Header Info for ERD file
C
C   Set parameters needed to write header
C   NUMKEY = 1 for 32-bit floating-point binary, 5 for Text
C
      NSAMP = TEND / DT / IPRINT + 1
      NRECS = NSAMP
      IF (NUMKEY .NE. 5) THEN
        NBYTES = 4*NCHAN
      ELSE
        NBYTES = 1
      END IF
C
C  Write standard ERD file heading.
C
      WRITE(IOUT,'(A)') 'ERDFILEV2.00'
      WRITE(IOUT,410) NCHAN, NSAMP, NRECS, NBYTES, NUMKEY, DT*IPRINT
  410 FORMAT(5(I6,','),E13.6)
  411 FORMAT (A8,255A8)
  412 FORMAT (A8, 31A32 : 2(/'&1000    ', 31A32))
      WRITE(IOUT,'(A,A)') 'TITLE    ', TITLE
      WRITE(IOUT,411) 'SHORTNAM', (SHORTN(J), J=1, NCHAN)
      WRITE(IOUT,412) 'LONGNAME', (LONGNM(J), J=1, NCHAN)
      WRITE(IOUT,411) 'UNITSNAM', (UNITNM(J), J=1, NCHAN)
      WRITE (IOUT, 412) 'GENNAME ', (GENNM(J), J=1, NCHAN)
      WRITE (IOUT, 412) 'RIGIBODY', (RIGBOD(J), J=1, NCHAN)
C
      WRITE (IOUT, '(A)') 'TRUCKSIMWind/Steer'
C
      IMPH = NINT (V * KMHMPH)
```

```
      IF (UNITS .NE. 'E' .AND. UNITS .NE. 'e')
     &   IMPH = NINT (V * KMHMPH / 1.61)
      WRITE (IOUT, '(A,I5)') 'SPEEDMPH', IMPH
C
      IF (NUMKEY .EQ. 5) WRITE(IOUT, '(A,A)') 'FORMAT  ',FRMT
C
      CALL TIMDAT (TIMEDT)

      WRITE (IOUT, '(A,A)')
     &  'HISTORY Data generated with Wind/Steer model at ', TIMEDT
      WRITE (IOUT, '(A,A)') 'HISTORY Input file was ', FNREAD
      WRITE (IOUT, '(A)') 'END'
C
C  If this is a Mac or PC, and data will be binary, then close header
C  and create binary file.  The following line is used to disable the
C  creating of a second file for MTS.
C
C--Use this only for the MTS version.
C       RETURN
  501 CONTINUE

      IF (NUMKEY .NE. 5) THEN
        CLOSE (IOUT)
        LNAME = LENSTR(FNOUT)
        FNOUT = FNOUT (:LNAME) // '.BIN'
        INQUIRE (FILE=FNOUT, EXIST=ISIT)
        IF (ISIT) THEN
          OPEN (IOUT, FILE=FNOUT)
          CLOSE (IOUT, STATUS='DELETE')
        END IF
C--The following 2 lines are for the Mac version.
        OPEN (IOUT, FILE=FNOUT, STATUS='NEW', ACCESS='SEQUENTIAL',
     &        FORM='UNFORMATTED')
C--The following 2 lines are for the PC version.
*        OPEN (IOUT, STATUS='NEW', ACCESS='SEQUENTIAL',
*     &        FORM='BINARY')
      END IF

      RETURN
      END
***********************************************************************
      SUBROUTINE OUTPUT(T, Y, YP)
***********************************************************************
C   Subroutine  OUTPUT  copies (at each output step) the simulation
C   variables into a buffer array in a sequence as specified in the
C   header, and outputs the buffer into the erd-file in binary or
C   text form.
C
      SAVE
      IMPLICIT REAL (K,M)
      INTEGER*2 LEN2, IBMROW, IBMCOL
      REAL BUFFER(66), Y(*), YP(*)
C
      include GLBL.inc
      include PARS.inc
      include VARS.inc
      include AERO.inc
C
```

```
C  Fill the ERD buffer.  The following code should create a one-to-one
C  match between the label sets created in routine OPNOUT and output
C  variables put into the array BUFFER.
C
      NCHAN = 0
C
C  Time and inputs
C
      BUFFER (NCHAN + 1) = T
      BUFFER (NCHAN + 2) = SW
      NCHAN = NCHAN + 2
      IF (SSKEY .NE. 0.0) THEN
        NCHAN = NCHAN + 1
        BUFFER (NCHAN) = STORQ
        NCHAN = NCHAN + 1
        BUFFER (NCHAN) = VSW
      END IF
C
C  Body position variables
C
      BUFFER (NCHAN + 1) = Y(1) / ININFT
      BUFFER (NCHAN + 2) = Y(2) / ININFT
      BUFFER (NCHAN + 3) = Y(3)
      BUFFER (NCHAN + 4) = Y(4) * TODEG
      BUFFER (NCHAN + 5) = Y(5) * TODEG
      BUFFER (NCHAN + 6) = Y(6) * TODEG
      NCHAN = NCHAN + 6
C
C  Body speed variables
C
      BUFFER (NCHAN + 1) = Y(7) * TODEG
      BUFFER (NCHAN + 2) = Y(8) * TODEG
      BUFFER (NCHAN + 3) = Y(9) * TODEG
      BUFFER (NCHAN + 4) = Y(10) * TODEG
      BUFFER (NCHAN + 5) = YP(1) / ININFT
      BUFFER (NCHAN + 6) = YP(2) / ININFT
      BUFFER (NCHAN + 7) = Y(11)
      NCHAN = NCHAN + 7
C
C  Lateral Acceleration, Path Curvature
C
      BUFFER (NCHAN + 1) = AY
      BUFFER (NCHAN + 2) = RHO * ININFT
      NCHAN = NCHAN + 2
C
C   Aerodynamic variables
C
      BUFFER (NCHAN + 1) = FDRAG
      BUFFER (NCHAN + 2) = FYA
      BUFFER (NCHAN + 3) = FZA
      BUFFER (NCHAN + 4) = MXA
      BUFFER (NCHAN + 5) = MYA
      BUFFER (NCHAN + 6) = MZA
      BUFFER (NCHAN + 7) = VA
      BUFFER (NCHAN + 8) = VWIND / ININFT
      IF(T .GE. 0.1 .AND. SSKEY .NE. 0.0) THEN
         BUFFER (NCHAN + 9) = BUFFER (NCHAN + 9) + STORQ *
     &   (SW - SWLAST) / 57.3
```

```
         BUFFER (NCHAN + 10) = BUFFER (NCHAN + 9) / T
      ELSE
         BUFFER (NCHAN + 9) = 0.0
         BUFFER (NCHAN + 10) = 0.0
      ENDIF
      SWLAST = SW
      BUFFER (NCHAN + 11) = BETAIR
      NCHAN = NCHAN + 11
C
C   Tire/Wheel variables
C
      DO 100, NAXLE = 1, 2
        DO 80, NSIDE = 1, 2
          BUFFER (NCHAN + 1) = TTLSTR(NAXLE,NSIDE) * TODEG
          BUFFER (NCHAN + 2) = ALFA(NAXLE,NSIDE) * TODEG
          BUFFER (NCHAN + 3) = GAMMA(NAXLE,NSIDE) * TODEG
          BUFFER (NCHAN + 4) = FY(NAXLE,NSIDE)
          BUFFER (NCHAN + 5) = MZ(NAXLE,NSIDE)
          BUFFER (NCHAN + 6) = FZ(NAXLE,NSIDE)
          BUFFER (NCHAN + 7) = ZW(NAXLE,NSIDE)
          BUFFER (NCHAN + 8) = FD(NAXLE,NSIDE)
          NCHAN = NCHAN + 8
   80   CONTINUE
  100 CONTINUE
C
C  Write data to the file.
C
C--The next 3 lines are for the Mac
      IF (T .EQ. 0.) WRITE (*, '(A/7X,A)') ' Progress:','sec'
      CALL TOOLBX (Z'89409000',0,-11)
      WRITE (*, '(F6.2)') T
C--The next 11 lines are for the IBM PC
*      IF (T .EQ. 0.) THEN
*        IBMROW = 18
*        IBMCOL = 10
*        WRITE (*, '(/////A\)') ' '
*        CALL SETCUR (IBMROW, IBMCOL)
*        WRITE (*, '(A,12X,A\)') ' Progress:','sec'
*      END IF

*      IBMROW = 18
*      IBMCOL = 22
*      CALL SETCUR (IBMROW, IBMCOL)
*      WRITE (*, '(F6.2\)') T
C--End IBM PC stuff

      IF (NUMKEY .EQ. 5) THEN
        WRITE (IOUT, FRMT) (BUFFER(J),J=1, NCHAN)
      ELSE
C
C--This line is only for MTS
C        LEN2 = NBYTES
C        CALL WRITE (BUFFER, LEN2, 16384, LNUM, IOUT)
C
C--This line is for the Mac and the PC
        WRITE (IOUT) (BUFFER (J), J=1, NCHAN)
      END IF
C
```

```
      RETURN
      END
************************************************************************
      FUNCTION POLY4(COEF,FZ)
************************************************************************
* evaluate 4-th order polynomial

      REAL COEF(*)
      POLY4 = COEF(1) + COEF(2)*FZ + COEF(3)*FZ*FZ + COEF(4)*FZ**3
      RETURN
      END
************************************************************************
      SUBROUTINE ROLLAX(ROLL, YROLAX, HROLAX, IXSRA)
************************************************************************
C   Subroutine ROLLAX returns YROLAX and HROLAX, the dynamic lateral
C   and vertical distances of the sprung mass cg from the roll axis in
C   a non-rolling reference frame, and IXSRA, the sprung-mass moment of
C   inertia about the instantaneous roll axis, as functions of roll.
C   (Effects of roll-axis inclination the from x-x axis are neglected.)
C
      IMPLICIT REAL (K,M)
      REAL IXSRA
C
      include PARS.inc
      include SUSP.inc
C
C   For each axle, find dynamic r.c. displacements in sprung mass
C   with sprung cg as origin
C
      DO 40 NAXLE=1, 2
        YRC(NAXLE) = 0.0
        HRC(NAXLE) = HCGSRC(NAXLE)
        DO 20 NPOWER=1, 2
          YRC(NAXLE) = YRC(NAXLE) + YROLCF(NPOWER,NAXLE) * ROLL**NPOWER
          HRC(NAXLE) = HRC(NAXLE) + HROLCF(NPOWER,NAXLE) * ROLL**NPOWER
   20   CONTINUE
   40 CONTINUE
C
C   Find y and z projections of roll-axis distance from sprung cg
C   in sprung-mass (rolling) reference frame
C
      YRACG = YRC(1) + (YRC(2) - YRC(1)) * XCGSP / WB
      HRACG = HRC(1) + (HRC(2) - HRC(1)) * XCGSP / WB
C
C   Transform y and z projections into non-rolling frame
C   (Approximating:  cos(roll) = 1, sin(roll) = roll )
C
      YROLAX = YRACG + HRACG * ROLL
      HROLAX = HRACG - YRACG * ROLL
C
C   Calculate IXSRA based on ixscg and roll-axis arm (YRACG**2+HRACG**2)
C
      IXSRA = IXSCG + (YRACG * YRACG + HRACG * HRACG) * SPMASS
C
      RETURN
      END
*******************************************************************
      FUNCTION STEER(T)
```

```
*********************************************************************
C   Function  steer  returns the steering wheel-angle (deg), SW,
C   or steering wheel torque (in-lbs), STORQ,
C   as a function of T in one of 3 control modes:
C     (NSTEER > 0) -- use table look-up
C     (NSTEER = 0) -- sinusoid function
C     (NSTEER < 0) -- Driver model
C     (NSTEER < -100) -- sinusoidal torque sweep
C
      SAVE
      IMPLICIT REAL (K,M)
      include vars.inc
      include mnvr.inc
      include glbl.inc
      include pars.inc
      include drvmod.inc
C
      DIMENSION YDR(7)
C
      DATA DFW,DFWNOW /2*0.0/
      DATA DRTORQ,DRTNOW /2*0.0/
C
      IF (NSTEER) 100, 200, 300
C
C   Driver model:
C
  100 IF(ABS(SSKEY) .LE. 0.001) THEN
        YDR(1) = YOUTDR(2) / ININFT
        YDR(2) = YOUTDR(10) * V / ININFT
        YDR(3) = YOUTDR(9)
        YDR(4) = YOUTDR(6)
        YDR(5) = YOUTDR(1) / ININFT
        DFWNOW = (TTLSTR(1,1) + TTLSTR(1,2)) * 0.5
        CALL DRIVER(T, YDR, DFW, DFWNOW)
C
C  Add kinematic and compliance steer effects (prior time step) and
C  convert to degrees at steering wheel:
C
        STEER = ( DFW - (KNMSTR(1,1) + KNMSTR(1,2)) * 0.5 -
     1   (CPLSTR(1,1) + CPLSTR(1,2)) * 0.5 ) * GRTODG
C
C  No initial steering from driver during lag period:
C
        IF(T .LE. TAUMEM) STEER = 0.0
C
        RETURN
C
      ELSE
        IF(NSTEER .LT. -100) THEN
          W0   = 0.1 * 6.2832
          WMAX = 4.0 * 6.2832
          WW = (WMAX - W0) / 2.0 * (1. - COS(6.2832/25. * T) ) + W0
          STEER = 20. * SIN (WW * T)
          RETURN
        ELSE
          YDR(1) = YOUTDR(2) / ININFT
          YDR(2) = YOUTDR(10) * V / ININFT
          YDR(3) = YOUTDR(9)
```

```
          YDR(4) = YOUTDR(6)
          YDR(5) = YOUTDR(13) / TODEG
          YDR(6) = YOUTDR(12) / TODEG
          YDR(7) = YOUTDR(1) / ININFT
          DRTNOW = STORQ / ININFT
          CALL DRIVET(T, YDR, DRTORQ, DRTNOW)
          STEER = DRTORQ * ININFT
C
C  No initial torque from driver during lag period:
C
          IF(T .LE. TAUMEM) STEER = 0.0
C
          RETURN

        ENDIF
C
      ENDIF
C
C
C   Sinusoidal steer function:
C
  200 IF (T .LT. TSWBGN) THEN
        STEER = 0.0
      ELSE
        IF (T .LE. TSWEND) STEER = SWSHFT +
     1      SWAMPL * SIN(2*PI*(T-TSWBGN)/TSWPRD + SWPHSE)
      END IF
C                    (FOR T > TSWEND, STEER IS NOT CHANGED)
      RETURN
C
C   Steer table:
C
  300 IF (T .LT. XPNT(NSTEER)) GO TO 310
C
C     Steering angle past the end of the table retains end value:
C
      STEER = YPNT(NSTEER)
      RETURN
C
  310 IF (INDX .NE. 0) GO TO 330
C
C     First call  -  pre-compute elements in SLOPE array
C
      DO 320, J=1,NSTEER-1
        SLOPE(J) = (YPNT(J+1) - YPNT(J)) / (XPNT(J+1) - XPNT(J))
  320 CONTINUE
C
C     Increment interval J if t >= XPNT(J+1), else pop to interpolate:
C
  330 DO 340, J = 1, NSTEER-1
        INDX = J
        IF (T .GE. XPNT(J) .AND. T .LT. XPNT(J+1)) GO TO 350
  340 CONTINUE
C
  350 STEER = YPNT(INDX) + (T - XPNT(INDX)) * SLOPE(INDX)
C
C   INDX  will hold the number (index) of the 'active' table interval
C
```

```
      RETURN
      END
*******************************************************************
      FUNCTION SUM(MATRIX)
*******************************************************************
C   FUNCTION SUM PERFORMS A SUMMATION OF ALL COMPONENTS
C   OF A 2 X 2 MATRIX ("WHEEL" ARRAY)
C
      REAL MATRIX(2,2)
C
      SUM = MATRIX(1,1) + MATRIX(1,2) + MATRIX(2,1) + MATRIX(2,2)
      RETURN
      END
***********************************************************************
      SUBROUTINE TABLE(M, N, X, Y, Z, Q)
***********************************************************************
C  Table look-up routine.  Q = Y(X), FOR X = Z.  Search over range
C  X(M) -> X(N).
C
      DIMENSION X(*), Y(*)
C
      INC = 1
      DO 20 I = M, N, INC
        IF (Z .LE. X(I)) GO TO 30
   20 CONTINUE
      Q = Y(N)
      RETURN
   30 IF (I .NE. M .AND. Z .NE. X(I)) GO TO 40
      Q = Y(I)
      IF (I .EQ. M .AND. Z .LT. X(I)) Q = Y(M)
      RETURN
   40 Q = (Y(I)*(Z - X(I - INC)) - Y(I - INC)*(Z - X(I))) / (X(I) - X(I
     1- INC))
      RETURN
      END
*******************************************************************
      SUBROUTINE TIMDAT (TIMEDT)
*******************************************************************
C  Get date and time
C
C  <-- TIMEDT char*24   string containing time & date.
C
      CHARACTER*24 TIMEDT
      CHARACTER*36 MONTHS
      INTEGER*2 YEAR, MONTH, DAY, HOUR, MIN, SEC, I100
      MONTHS = 'JanFebMarAprMayJunJulAugSepOctNovDec'

C--The following 4 lines are for the IBM PC (using Microsoft
C--time and date functions)
*      CALL GETDAT (YEAR, MONTH, DAY)
*      CALL GETTIM (IHOUR, MIN, SEC, I100)
*      WRITE (TIMEDT, 100) IHOUR, MIN, MONTHS (MONTH*3-2:MONTH*3),
*     &       DAY, YEAR

C--get time for MTS version
C      CALL TIME(22, 0, TIMEDT)

C--The following 5 lines are for the Apple Mac
```

```
C--(using Absoft time & date functions)
      call date (m, iday, iyear)
      call time (isec)
      write (timedt, 100)
     &       isec/3600, mod (isec, 3600) / 60, months (m*3-2:m*3),
     &       iday, 1900 + iyear

  100 FORMAT (I2,':',I2.2,' on ',A3,I3,',',I5)
      RETURN
      END
*************************************************************************
      FUNCTION TIREFY(ALPHA, GAMMA, FZ, AXLE)
*************************************************************************
* This function computes tire side force.  This version (UMTRI) is
* linear in alpha and gamma, but nonlinear in Fz.

* --> alpha  real    slip angle
* --> gamma  real    camber angle
* --> FZ     real    vertical load
* --> AXLE   integer axle no. (1 or 2)
* <-- TIREFY  real   side force for tire
*
      include TIRE.inc
      INTEGER AXLE
      TIREFY =  POLY4(CALFA(1,AXLE), FZ)*ALPHA
     &        + POLY4(CGAMMA(1,AXLE), FZ)*GAMMA
      RETURN
      END
*************************************************************************
      FUNCTION TIREMZ(ALPHA, FZ, AXLE)
*************************************************************************
* This function computes tire aligning torque.  This version (UMTRI) is
* linear in alpha, but nonlinear in Fz.

* --> ALPHA  real    slip angle
* --> FZ     real    vertical load
* --> AXLE   integer axle no. (1 or 2)
* <-- TIREMZ  real   side force for tire
*
      include TIRE.inc
      INTEGER AXLE
      TIREMZ = POLY4(CALIGN(1, AXLE), FZ)*ALPHA
      RETURN
      END
*************************************************************************
      SUBROUTINE TIRES(BETA, V, VYAW, ROLL)
*************************************************************************
* This subroutine solve simultaneous equations for slip and camber
* angles and tire forces and moments.
*
      IMPLICIT REAL (K,M)
      EXTERNAL INDSUS, BEAM
      include TSOLVE.inc
      include SUSP.inc
      include VARS.inc
      include PARS.inc
      REAL X(2)
C
```

```
      DO 100 AXLE = 1, 2
        YWPART = BETA + VYAW * XAXLE(AXLE) / V
C
C Case for beam rear axle (no camber compliance, same steer compliance
C for both wheels):
C
        IF (AXLE .EQ. 2 .AND. BEAM) THEN
C
C Compute known part of alphas, then solve equations
C
          BIAS(1) = YWPART - KNMSTR(2, 1)
          X(1) = ALFA(2, 1)
          X(2) = ALFA(2, 2)
          CALL MNEWT(X, BEAM, IERR)
          IF (IERR .NE. 0) THEN
      WRITE(*,*) ' Error, Newton-Raphson iteration did not converge.'
      WRITE(*,*) ' There is a problem in the tire/suspension equations.'
      WRITE(*,*) ' Rear (beam) suspension.'
      WRITE(*,*) ' (Press Return)'
      PAUSE
      STOP
          END IF
          ALFA(2, 1) = X(1)
          ALFA(2, 2) = X(2)
          FY(2, 1) = SAVEFY(1)
          FY(2, 2) = SAVEFY(2)
          MZ(2, 1) = SAVEMZ(1)
          MZ(2, 2) = SAVEMZ(2)
C
C  Calculate compliance steer and "total" steer
C
          CPLSTR(2,1) = CSMZ(2) * (SAVEMZ(1) + SAVEMZ(2))
     &                + CSFY(2) * (SAVEFY(1) + SAVEFY(2))
          CPLSTR(2,2) = CPLSTR(2,1)
          TTLSTR(2,1) = KNMSTR(2,1) + CPLSTR(2,1)
          TTLSTR(2,2) = TTLSTR(2,1)
C
C Independent wheels, with coupling between camber and steer:
C
        ELSE
          DO 90 SIDE = 1, 2
C
C Check for steering angle (from steering system on front axle)
C
            IF(AXLE .EQ. 1) THEN
              IF (ABS(SSKEY) .GT. 0.001) THEN
                CSMZ(AXLE) = 0.0
                STRCON = FW(SIDE)
              ELSE
                STRCON = SW / GRTODG
              ENDIF
            ELSE
              STRCON = 0
            END IF
C
C Sign of camber, toe-in depends on side
C
            IF (SIDE .EQ. 1) THEN
```

```
              BIAS(1) = ALFA0(AXLE)
              BIAS(2) = -GAMMA0(AXLE)
            ELSE
              BIAS(1) = -ALFA0(AXLE)
              BIAS(2) = GAMMA0(AXLE)
            END IF
C
C Compute known part of alpha and gamma, then solve equations
C
            BIAS(1) = BIAS(1) + YWPART - STRCON - KNMSTR(AXLE,SIDE)
            BIAS(2) = BIAS(2) + ROLL + KNMCBR(AXLE,SIDE)
            FZTEMP = FZ(AXLE, SIDE)
            X(1) = ALFA(AXLE, SIDE)
            X(2) = GAMMA(AXLE, SIDE)
            CALL MNEWT(X, INDSUS, IERR)
            IF (IERR .NE. 0) THEN
      WRITE(*,*) ' Error, Newton-Raphson iteration did not converge.'
      WRITE(*,*) ' There is a problem in the tire/suspension equations.'
      WRITE(*,*) ' AXLE, SIDE=', AXLE, SIDE
      WRITE(*,*) ' (Press Return)'
      PAUSE
      STOP
            END IF
            ALFA(AXLE, SIDE) = X(1)
            GAMMA(AXLE, SIDE) = X(2)
            FY(AXLE, SIDE) = SAVEFY(1)
            MZ(AXLE, SIDE) = SAVEMZ(1)
C
C  Calculate compliance steer and "total" steer (kinem + compl + strcon
C  input):
C
            CPLSTR(AXLE,SIDE) = CSMZ(AXLE) * SAVEMZ(1)
     &                        + CSFY(AXLE) * SAVEFY(1)
            TTLSTR(AXLE,SIDE) = KNMSTR(AXLE,SIDE)
     &                          + CPLSTR(AXLE, SIDE)
     &                          + (2 - AXLE) * STRCON
   90     CONTINUE
        END IF
  100 CONTINUE
      RETURN
      END
***********************************************************************
C
C   *** Trajectory Subroutine ***
C
C TRAJ: Computes lateral displacent of previewed path as a table look-up
C
C===========Author and Modification Section===========================
C
C  Author:       C. C. MacAdam
C
C  Date written: 01/01/88
C
C  Written on:
C
C  Modifications:
C
C=====================================================================
```

```
C
C============Algorithm Description=======================================
C
C  Purpose and use:
C
C  Error conditions:
C
C  Machine dependencies: none
C
C  Called By: DRIVER
C
C=======================================================================
C
      SUBROUTINE TRAJ(X, XT, YT, YPATH)
      SAVE
C
C============Variable Descriptions======================================
C
C---Arguments passed:
C
C  ->X.......forward displacement (ft)
C  ->XT......longitudinal path coordinates (ft)
C  ->YT......lateral path coordinated corresponding to XT values (ft)
C  <-YPATH...lateral displacement of path corresponding to X, (ft)
C
C
      DIMENSION XT(*), YT(*)
C
C---Local variables-----------------------------------------------------
C
C  J.......integer counter
C  SLOPE...dYT/dXT of path at X
C
C---Functions and subroutines-------------------------------------------
C
C      None
C
C=======================================================================
C
C============Process Block==============================================
C
C  SEARCH FOR XI,XI+1:
      DO 10 J = 1, 99
        IF (X .GE. XT(J) .AND. X .LT. XT(J + 1)) GO TO 30
   10 CONTINUE
      WRITE (*,20)
   20 FORMAT ('0', 'X-SEARCH IN SUB. TRAJ FAILED.')
      STOP
   30 SLOPE = (YT(J + 1) - YT(J)) / (XT(J + 1) - XT(J))
      YPATH = YT(J) + SLOPE * (X - XT(J))
      RETURN
      END
C***********************************************************************
C
C  Transition Matrix Calculation.
C
C
C TRANS: Computes transition matrix of the linearized system
```

```
C
C============Author and Modification Section============================
C
C  Author:       C. C. MacAdam
C
C  Date written: 05/19/88
C
C  Written on:
C
C  Modifications:
C
C=======================================================================
C
C============Algorithm Description======================================
C
C  Purpose and use: Used by the driver model in predicting future states
C
C  Error conditions:
C
C  Machine dependencies: none
C
C  Called By: DRIVGO
C
C=======================================================================
C
      SUBROUTINE TRANS
      SAVE
C
C============Variable Descriptions======================================
C
C---Arguments passed:  None
C
      DIMENSION SV(4), SD(4), SVI(4)
C
C---COMMON blocks-------------------------------------------------------
C
      include drvmod.inc
C
C---DRIV.BLK common block variables-------------------------------------
C
C  CAF...total cornering stiffness of tires on left front susp (lb/rad)
C  CAR...total cornering stiffness of tires on left rear susp (lb/rad)
C  WHBS..wheelbase of vehicle (center-line of front & rear susp) (ft)
C  WF....static load on front suspension (lb)
C  WR....static load on rear suspension (lb)
C  U.....initial velocity (ft/sec)
C
C---DRVST1.BLK common block variables
C
C   GRAV.....gravitational constant
C   TICYCL...driver model sample time (sec)
C   TSS......minimum preview time (sec)
C   DMAX.....upper bound on front wheel angle steer (rad)
C   XP,YP....x-y path coords(SAE) wrt inertial coords [input] (ft)
C   TAUMEM...driver transport time dealy [input parameter] (sec)
C   TFF......driver model preview time [input parameter] (sec)
C   RM.......vehicle mass (slug)
C   A........distance from c.g. to front suspension center-line (ft)
```

```
C   B........distance from c.g. to rear suspension center-line (ft)
C   RI.......total vehicle yaw inertia (slug-ft)
C   PSIO.....current yaw angle reference value (rad)
C   NTF......number of points in the preview time interval
C   NP.......number of points in the x-y trajectory table
C   TLAST....last time driver model calulated a steer value (sec)
C   DFWLST...last value of steer calculated by driver model (rad)
C   TILAST...last sample time driver model calulated a steer value (sec)
C   DMEM.....2-dim array (time & steer history) used in delay calculat'n
C   XT,YT....transformation of XP,YP in vehicle body axes (ft)
C
C---TRSSTR.BLK common block variables
C
C  TTT.......transition matrix at 10 discrete points in preview interval
C  TTT1......integral of trans matrix wrt preview time
C  GV.........vector of control gain coefficients
C
C---Local variables-----------------------------------------------------
C
C   DELT.....time step in local Euler integration (sec)
C   A1.......lat accel coefficient of sideslip veloc in linearizd system
C   B1.......    "               yaw rate              "
C   A2.......yaw accel    "          sideslip vel          "
C   B2.......    "               yaw rate              "
C   C1.......steer control gain coefficient for lateral accel
C   C2.......steer control gain coefficient for yaw moment
C   ULAST....last value of forward velocity (ft/sec)
C   NBEG.....integer startin counter value
C   NEND1....integer ending counter value
C   NENDV....integer ending counter value
C   J........integer counter
C   SV.......state vector: y,v,r,yaw,x  [SAE]
C   SV1......integral of state vector
C   SD.......state vector derivative
C
C---Functions and subroutines-------------------------------------------
C
C      None
C
C=======================================================================
C
C============Process Block==============================================
C
C
      DELT = 0.01
      A1 = -2. * (CAF + CAR) / RM / U
      B1 = 2. * (CAR*B - CAF*A) / RM / U - U
      A2 = 2. * (CAR*B - CAF*A) / RI / U
      B2 = -2. * (CAR*B*B + CAF*A*A) / RI / U
      C1 = 2. * CAF / RM
      C2 = 2. * CAF / RI * A
      ULAST = U
      GV(1) = 0.
      GV(2) = C1
      GV(3) = C2
      GV(4) = 0.
      DO 70 J = 1, 4
        NBEG = TSS / DELT + 1
```

```
        NEND1 = (TFF + .001 - TSS) / NTF / DELT
        NENDV = NEND1
        DO 10 L = 1, 4
          SV(L) = 0.0
          SVI(L) = 0.0
   10   CONTINUE
        TIME = 0.
C
C  Initialize each state in turn to 1.0 and integrate (Euler).
C
        SV(J) = 1.0
        DO 60 I = 1, NTF
          DO 40 K = NBEG, NENDV
            SD(1) = SV(2) + U * SV(4)
            SD(2) = A1 * SV(2) + B1 * SV(3)
            SD(3) = A2 * SV(2) + B2 * SV(3)
            SD(4) = SV(3)
            DO 20 L = 1, 4
              SV(L) = SV(L) + SD(L) * DELT
   20       CONTINUE
            TIME = TIME + DELT
            DO 30 L = 1, 4
              SVI(L) = SVI(L) + SV(L) * DELT
   30       CONTINUE
   40     CONTINUE
C
C  Store "impulse" responses in TTT columns, integral in TTT1.
C  TTT is a NPT-point tabular transition matrix,  TTT1 is its integral.
C  (See References 2 & 3.)
C
          DO 50 L = 1, 4
            TTT(L,J,I) = SV(L)
            TTT1(L,J,I) = SVI(L)
   50     CONTINUE
          NBEG = NBEG + NEND1
          NENDV = NENDV + NEND1
   60   CONTINUE
   70 CONTINUE
      RETURN
      END
C**********************************************************************
C**********************************************************************
C
C  Transition Matrix Calculation.
C
C
C TRANST: Computes transition matrix of the linearized system (torque
C         version of the driver model)
C
C============Author and Modification Section============================
C
C  Author:       C. C. MacAdam
C
C  Date written: 01/30/89
C
C  Written on:
C
C  Modifications:
```

```
C
C===========================================================================
C
C===========Algorithm Description==========================================
C
C  Purpose and use: Used by the driver model in predicting future states
C
C  Error conditions:
C
C  Machine dependencies: none
C
C  Called By: DRIVGT
C
C===========================================================================
C
      SUBROUTINE TRANST
      SAVE
      REAL KSSL, ISSL
C
C===========Variable Descriptions==========================================
C
C---Arguments passed:  None
C
      DIMENSION SV(6), SD(6), SVI(6)
C
C---COMMON blocks-----------------------------------------------------------
C
      include drvtor.inc
      include pars.inc
      include glbl.inc
      include tire.inc
      include vars.inc
C
C---DRIV.BLK common block variables-------------------------------------
--
C
C  CAF...total cornering stiffness of tires on left front susp (lb/rad)
C  CAR...total cornering stiffness of tires on left rear susp (lb/rad)
C  WHBS..wheelbase of vehicle (center-line of front & rear susp) (ft)
C  WF....static load on front suspension (lb)
C  WR....static load on rear suspension (lb)
C  U.....initial velocity (ft/sec)
C
C---DRVST1.BLK common block variables
C
C   GRAV.....gravitational constant
C   TICYCL...driver model sample time (sec)
C   TSS......minimum preview time (sec)
C   DMAX.....upper bound on front wheel angle steer (rad)
C   XP,YP....x-y path coords(SAE) wrt inertial coords [input] (ft)
C   TAUMEM...driver transport time dealy [input parameter] (sec)
C   TFF......driver model preview time [input parameter] (sec)
C   RM.......vehicle mass (slug)
C   A........distance from c.g. to front suspension center-line (ft)
C   B........distance from c.g. to rear suspension center-line (ft)
C   RI.......total vehicle yaw inertia (slug-ft)
C   PSIO.....current yaw angle reference value (rad)
C   NTF......number of points in the preview time interval
```

```
C   NP.......number of points in the x-y trajectory table
C   TLAST....last time driver model calulated a steer value (sec)
C   DFWLST...last value of steer calculated by driver model (rad)
C   TILAST...last sample time driver model calulated a steer value (sec)
C   DMEM.....2-dim array (time & steer history) used in delay calculat'n
C   XT,YT....transformation of XP,YP in vehicle body axes (ft)
C
C---TRSSTR.BLK common block variables
C
C  TTTT.......transition matrix at 10 discrete points in preview
interval
C  TTTT1......integral of trans matrix wrt preview time
C  GGV.........vector of control gain coefficients
C
C---Local variables----------------------------------------------------
C
C   DELT.....time step in local Euler integration (sec)
C   A1.......lat accel coefficient of sideslip veloc in linearizd system
C   B1.......     "             yaw rate            "
C   A2.......yaw accel    "         sideslip vel        "
C   B2.......     "             yaw rate            "
C   C1.......steer control gain coefficient for lateral accel
C   C2.......steer control gain coefficient for yaw moment
C   ULAST....last value of forward velocity (ft/sec)
C   NBEG.....integer startin counter value
C   NEND1....integer ending counter value
C   NENDV....integer ending counter value
C   J........integer counter
C   SV.......state vector: y,v,r,yaw,x  [SAE]
C   SV1......integral of state vector
C   SD.......state vector derivative
C
C---Functions and subroutines------------------------------------------
C
C      None
C
C=====================================================================
C
C===========Process Block=============================================
C
C
      CSDAML = CSS * TODEG / ININFT
      KSSL = KSS * TODEG / ININFT
      XP = - POLY4(CALIGN(1,1), FZ(1,1)) / POLY4(CALFA(1,1), FZ(1,1)) /
     &  ININFT
      XM = XTRAIL / ININFT
      ISSL = ISS / ININFT
      CSSL = CSS * TODEG / ININFT
      DELT = 0.01
      A1 = - 2. * (CAF + CAR) / RM / U
      B1 = 2. * (CAR*B - CAF*A) / RM / U - U
      A2 = 2. * (CAR*B - CAF*A) / RI / U
      B2 = - 2. * (CAR*B*B + CAF*A*A) / RI / U
      C1 = 2. * CAF / RM
      C2 = 2. * CAF / RI * A
      D1 = 1. / GR / (1. + 2. * (XP + XM) * CAF * (1. - CBOOST) / KSSL)
      E1 = 2. * (XP + XM) * CAF * (1. - CBOOST) / (U * KSSL
     &     * (1. + 2. * (XP + XM) * CAF * (1. - CBOOST) / KSSL) )
```

```
      F1 = A * E1
      A3 = 2. * (XP + XM) * CAF * (1. - CBOOST) / (GR * U * ISSL
     &     * (1. + 2. * (XP + XM) * CAF * (1. - CBOOST) / KSSL) )
      B3 = A * A3
      C3 = - KSSL / (GR**2) * (1. - 1. / (1. + 2. * (XP + XM)
     &     * CAF * (1. - CBOOST) / KSSL) ) / ISSL
      D3 = - CSSL / ISSL
C
      ULAST = U
      GGV(1) = 0.
      GGV(2) = 0.
      GGV(3) = 0.
      GGV(4) = 0.
      GGV(5) = 0.
      GGV(6) = 1. / ISSL
      DO 70 J = 1, 6
        NBEG = TSS / DELT + 1
        NEND1 = (TFF + .001 - TSS) / NTF / DELT
        NENDV = NEND1
        DO 10 L = 1, 6
          SV(L) = 0.0
          SVI(L) = 0.0
   10   CONTINUE
        TIME = 0.
C
C  Initialize each state in turn to 1.0 and integrate (Euler).
C
        SV(J) = 1.0
        DO 60 I = 1, NTF
          DO 40 K = NBEG, NENDV
            SD(1) = SV(2) + U * SV(4)
            SD(2) = (A1 + C1 * E1) * SV(2) + (B1 + C1 * F1) * SV(3)
     &              + C1 * D1 * SV(5)
            SD(3) = (A2 + C2 * E1) * SV(2) + (B2 + C2 * F1) * SV(3)
     &              + C2 * D1 * SV(5)
            SD(4) = SV(3)
            SD(5) = SV(6)
            SD(6) = A3 * SV(2) + B3 * SV(3) + C3 * SV(5) + D3 * SV(6)
C
            DO 20 L = 1, 6
              SV(L) = SV(L) + SD(L) * DELT
   20       CONTINUE
            TIME = TIME + DELT
            DO 30 L = 1, 6
              SVI(L) = SVI(L) + SV(L) * DELT
   30       CONTINUE
   40     CONTINUE
C
C  Store "impulse" responses in TTTT columns, integral in TTT1.
C  TTTT is a NPT-point tabular transition matrix,  TTT1 is its integral.
C  (See References 2 & 3.)
C
          DO 50 L = 1, 6
            TTTT(L,J,I) = SV(L)
            TTTT1(L,J,I) = SVI(L)
   50     CONTINUE
          NBEG = NBEG + NEND1
          NENDV = NENDV + NEND1
```

```
   60   CONTINUE
   70 CONTINUE
      RETURN
      END
*************************************************************************
      SUBROUTINE WHEELZ(Z, ROLL, PITCH)
*************************************************************************
C   Subroutine  wheelz  updates the matrices ZW, FZ, KNMSTR, KNMCBR in
C   common /VARS/ - namely: vertical displacement, normal ground load,
C   bump-steer angle and bump-camber angle for each wheel, relative to
C   static trim.
C   polarity:     jounce displacement  ==> positive ZW, FZ
C   ---------     rebound displacement ==> negative ZW, FZ
C
      IMPLICIT REAL (K,M)
      INTEGER SIDE
C
      include SUSP.inc
      include VARS.inc
C
      DO 30, NAXLE = 1, 2
        MOMENT = KAUX(NAXLE) * ROLL
     &           - H0ROLC(NAXLE) * (FY(NAXLE,1) + FY(NAXLE,2))
        DO 20, NSIDE = 1, 2
          IF (NSIDE .EQ. 1) THEN
            SIDE = 1
          ELSE
            SIDE = -1
          END IF
C
          THISZW = Z - XAXLE(NAXLE) * PITCH
     1              - SIDE * .5 * TRACK(NAXLE) * ROLL
          ZW(NAXLE,NSIDE) = THISZW
          FZ(NAXLE,NSIDE) = FZ0WHL(NAXLE) + THISZW * KZ(NAXLE)
     1        - SIDE * MOMENT / TRACK(NAXLE) + FD(NAXLE,NSIDE)
C
          IF (KINEM) THEN
            IF (NAXLE .EQ. 2 .AND. BEAM) THEN
              KNMSTR(2,NSIDE) = CSROLL * ROLL
            ELSE
              KNMSTR(NAXLE,NSIDE) = SIDE * (CSZ(1,NAXLE) * THISZW
     &                               + CSZ(2,NAXLE) * THISZW * THISZW)
              KNMCBR(NAXLE,NSIDE) = SIDE * (CCZ(1,NAXLE)
     &                   * THISZW  +  CCZ(2,NAXLE) * THISZW * THISZW)
            END IF
          END IF
   20   CONTINUE
   30 CONTINUE
C
      RETURN
      END
*
*
      SUBROUTINE FILTER(TIME, XNOW, YNOW, VNOW, XLAST, YLAST, TLAST)
      SAVE
*
*  First Order Filter:  y / x = a / (b + s)
*
```

```
*   TIME.....current time (sec)
*   XNOW.....current input signal to be filtered (input)
*   YNOW.....filtered value of XNOW signal (output)
*   VNOW.....first derivative of YNOW (output)
*   XLAST....value of XNOW at last filter computation (input)
*   YLAST....value of YNOW at last filter computation (input)
*   VLAST....value of VNOW at last filter computation (input)

*   TLAST....value of TIME at last filter computation (input)
*
*   Set Filter cutoff frequency (rad/s) and gain:
*
      BRKPT  = 1.5
      GAINDC = 200.0
*
      B = BRKPT
      A = GAINDC * B
*
*   UPDATE RETURN VALUES FROM LAST ENTRY IN CASE OF T<=0 RETURN
*
      YNOW = YLAST
      VNOW = VLAST
*
*   INITIALIZE FOR TIME ZERO
*
        IF(TIME .LE. 0.0) THEN
                YLAST = XNOW
                VLAST = 0.0
                XLAST = XNOW
                TLAST = 0.0
                YNOW = XNOW
                VNOW = 0.0
        END IF
*
        T = TIME - TLAST
*
        IF(T .LE. 0.0) RETURN
*
*
*   COMPUTE CONSTANTS IN RECURSION EXPRESSIONS
*
      C1 = (1.0 - T * B / 2.0) / (1.0 + T * B / 2.0)
      C2 = A * T / (2.0 + T * B)
*
*   CALCULATE FILTERED VALUE OF DISPLACEMENT & CURRENT VELOCITY:
*
           YNOW =  C1 * YLAST + C2 * (XNOW+XLAST)
           VNOW = A * XNOW - B * YNOW
*
*   UPDATE VALUES FOR NEXT ENTRY PRIOR TO RETURNING
*
        YLAST = YNOW
        XLAST = XNOW
        TLAST = TIME
*
        RETURN
*
        END
```

```
*
*
*
      FUNCTION RAN3(IDUM)
      SAVE
         IMPLICIT REAL*4(M)
         PARAMETER (MBIG=4000000.,MSEED=1618033.,MZ=0.,FAC=2.5E-7)
C       PARAMETER (MBIG=1000000000,MSEED=161803398,MZ=0,FAC=1.E-9)
      DIMENSION MA(55)
      DATA IFF /0/
      IF(IDUM.LT.0.OR.IFF.EQ.0)THEN
        IFF=1
        MJ=MSEED-IABS(IDUM)
        MJ=MOD(MJ,MBIG)
        MA(55)=MJ
        MK=1
        DO 11 I=1,54
          II=MOD(21*I,55)
          MA(II)=MK
          MK=MJ-MK
          IF(MK.LT.MZ)MK=MK+MBIG
          MJ=MA(II)
11      CONTINUE
        DO 13 K=1,4
          DO 12 I=1,55
            MA(I)=MA(I)-MA(1+MOD(I+30,55))
            IF(MA(I).LT.MZ)MA(I)=MA(I)+MBIG
12        CONTINUE
13      CONTINUE
        INEXT=0
        INEXTP=31
        IDUM=1
      ENDIF
      INEXT=INEXT+1
      IF(INEXT.EQ.56)INEXT=1
      INEXTP=INEXTP+1
      IF(INEXTP.EQ.56)INEXTP=1
      MJ=MA(INEXT)-MA(INEXTP)
      IF(MJ.LT.MZ)MJ=MJ+MBIG
      MA(INEXT)=MJ
      RAN3=MJ*FAC
      RETURN
      END
```

Include Files

```
C
C  -----------------------------> AERO:
C
      REAL KY, KL, KR, KM, KN, KSYWND, MXA, MYA, MZA, KD
      INTEGER WINDKY
      COMMON /AERO/ AIRHO, AREA, QZERO, KY, CL0, KL, KR, CM0, KM, KN,
     1              VWIND, KSYWND, VA, BETAIR, FYA, FZA, MXA, MYA, MZA,
     2              CD0, KD, FDRAG, WINDKY, TWIND(1000), WINMAG(1000)
      SAVE /AERO/
```

```
C
C -----------------------------> DRIV:
C
      COMMON /DRVST1/ GRAV,TICYCL,TSS,DMAX,XPDR(100), YPDR(100), TAUMEM,
     1        TFF, RM, A, B, RI, PSI0, NTF, NP, TLAST, DFWLST, TILAST,
     2        DMEM(100,2), XT(100), YT(100)
      SAVE/DRVST1/
      COMMON /DRIV/ CAF, CAR, WHBS, WF, WR, U
      SAVE/DRIV/
      COMMON /TRSSTR/ TTT(4,4,10), TTT1(4,4,10), GV(4)
      SAVE/TRSSTR/


C
C -----------------------------> DRIV:
C
      COMMON /DRVST1/ GRAV,TICYCL,TSS,DMAX,XPDR(100), YPDR(100), TAUMEM,
     1        TFF, RM, A, B, RI, PSI0, NTF, NP, TLAST, DFWLST, TILAST,
     2        DMEM(100,2), XT(100), YT(100)
      SAVE/DRVST1/
      COMMON /DRIV/ CAF, CAR, WHBS, WF, WR, U
      SAVE/DRIV/
      COMMON /TRSTOR/ TTTT(6,6,10), TTTT1(6,6,10), GGV(6), STMAX
      SAVE/TRSTOR/


C
C -----------------------------> GLBL:
C
      CHARACTER*80 TITLE
      CHARACTER*32 FNREAD, FRMT
      CHARACTER*8  UOMEGA, UTORQ, UANGL, UVELFT
      CHARACTER*2  UDISP, UDIST, UFORC
      CHARACTER*1  UNITS
      REAL KMHMPH, ININFT
      INTEGER NBYTES
      PARAMETER (IREAD=5, IECHO=7, IOUT=8)
      COMMON /GLBL/ NEQN, V, TEND, DT, NUMKEY, LNAME,
     &              IPRINT, PI, ININFT, KMHMPH, G, TODEG, TITLE,
     &              UOMEGA, UANGL, UVELFT, UTORQ, UDISP, UDIST, UFORC,
     &              FNREAD, FRMT, NBYTES, UNITS
      SAVE /GLBL/


C
C -----------------------------> MNVR:
C
      REAL XPNT(999), YPNT(999), SLOPE(999)
      COMMON /MNVR/ NSTEER, INDX, TSWBGN, TSWEND, SWAMPL, TSWPRD,
     1              SWPHSE, SWSHFT, DRLAG, DRPREV, XPNT, YPNT, SLOPE
      SAVE /MNVR/


C
C -----------------------------> PARS:
C
      REAL IXSCG, IXZ, IYS, IZZ, ISS, KSS
      REAL MASS, KSC, KSL, KROLL
```

```
      COMMON /PARS/ MASS, SPMASS, IXSCG, IXZ, IYS, IZZ, HCGTTL, WHLRAD,
     1              WEIGHT, SPWGHT, USWGHT, WRATIO, WB, GR, GRTODG,
     2              ISS, KSC, CBOOST, SSKEY, XTRAIL, KROLL, CFSS,
     3              XWBCGS, XWBCGT, XCGSP, HCGSP, DLASH, CSS, KSL, KSS,
     4              SWSTOP, XPM
      SAVE /PARS/

C
C ----------------------------> PRNT:
C
      CHARACTER*80 VNAMES, VUNITS
      CHARACTER*12 BLNK12
      REAL PRBUFF(0:50,5)
      COMMON /PRNT/ LINE, NPAGE, VNAMES, VUNITS, BLNK12, PRBUFF
      SAVE /PRNT/

C
C ----------------------------> SUSP:
C
      LOGICAL KINEM, BEAM
      REAL TRACK(2), XAXLE(2), KZ(2), KZAXLE(2), KAUX(2)
      REAL KTIRE(2), CZJNCE(2), CZRBND(2), ALFA0(2), GAMMA0(2)
      REAL FZ0WHL(2), H0ROLC(2), HCGSRC(2), CSFY(2), CSMZ(2), CCFY(2)
      REAL CSZ(2,2), CCZ(2,2), YROLCF(2,2), HROLCF(2,2), YRC(2), HRC(2)
      COMMON /SUSP/ KINEM, BEAM, CSROLL, TRACK, XAXLE, KZ, KZAXLE,
     1              KAUX, KTIRE, CZJNCE, CZRBND, ALFA0, GAMMA0, FZ0WHL,
     2              H0ROLC, HCGSRC, YROLCF, HROLCF, CSFY, CSMZ, CCFY,
     3              CSZ, CCZ, YRC, HRC
      SAVE /SUSP/

C
C ----------------------------> TIRE:
C
      REAL CALFA(4,2), CGAMMA(4,2), CALIGN(4,2)
      COMMON /TIRE/ CALFA, CGAMMA, CALIGN
      SAVE /TIRE/

* common block to share data between the tire-equation solver
* and the tire-matrix generator used by the Newton-Raphson routine.

      INTEGER AXLE, SIDE
      COMMON /TSOLVE/ AXLE, SIDE, BIAS(2), FZTEMP, SAVEFY(2), SAVEMZ(2)
      SAVE /TSOLVE/

C
C ----------------------------> VARS:
C
      REAL ALFA(2,2), GAMMA(2,2), FY(2,2), MZ(2,2), FD(2,2), FZ(2,2)
      REAL ZW(2,2), KNMSTR(2,2), CPLSTR(2,2), TTLSTR(2,2), KNMCBR(2,2)
      REAL YPOUT(13), YOUTDR(13), FW(2), MMCOL
      COMMON /VARS/ SW, FW, AY, RHO, ALFA, GAMMA, FY, MZ, FD, FZ, ZW,
     1              KNMSTR, CPLSTR, TTLSTR, KNMCBR, YPOUT, YOUTDR,
     2              STORQ, BOOST, MMCOL, VSW
```

```
SAVE /VARS/
```