Genetic algorithms and nonlinear programming for optimal low-thrust spacecraft trajectories

by

Lalitesh Kumar Katragadda

A Thesis Submitted to the

Graduate Faculty in Partial Fulfillment of the

Requirements for the Degree of

MASTER OF SCIENCE

Department: Aerospace Engineering and Engineering Mechanics
Major: Aerospace Engineering

Iowa State University
Ames, Iowa
1991

TABLE OF CONTENTS

NOMENCLATURE viii

ACKNOWLEDGEMENTS xii

ABSTRACT xiii

CHAPTER 1. INTRODUCTION 1

Projected Space Scenario 1

Low Thrust Transfers 2

Our Scope 4

CHAPTER 2. PROBLEM MODELING 6

Direct and Indirect Formulation 8

Continuous Thrust Earth-Mars Transfer 10

Maximum Energy Earth Escape 12

Optimal Earth-Moon Transfer 15

Normalization of the Variables 18

CHAPTER 3. NUMERICAL TECHNIQUES 21

Overview of Methods 21

Parameterization 22

The Indirect Method 25

Boundary Value Problem Solvers 26

Nonlinear Constrained Parameter Optimization 27

Numerical Integration 30

Error and Tolerance Scheduling 31

CHAPTER 4. GENETIC ALGORITHMS 33

Genetic Algorithms in Optimization 36

GAs in Optimal Control 40

Incomplete simulation for efficiency 42

Testing the GA 43

CHAPTER 5. COMPARISON OF MODELS AND METHODS . . 52

Effect of Models 52

Continuous thrust Mars transfer 53

Maximum energy Earth escape 55

Single coast Earth-Moon transfer 57

Algorithm Comparison and Maximum Accuracies 57

GAs and Low Accuracy Solutions 59

Model enhancement through GAs solutions 59

CHAPTER 6. PRESENTATION OF SOLUTIONS 61

CHAPTER 7. CONCLUSIONS AND SUGGESTIONS 72

Conclusions 72

Observations 74

Suggestions 75

REFERENCES 77

APPENDIX A. SOLAR AND SPACECRAFT PARAMETERS . . 79

APPENDIX B. DERIVATION OF OPTIMAL ESCAPE MODELS 81

APPENDIX C. DERIVATION OF THE THREE-BODY MODEL . 83

APPENDIX D. CODE LISTING 87

Main Program 87

Initial Data Module 99

Simulation Module 101

Colsys Interface 127

SQP Interface 142

Penalty Interface 147

FORTRAN Interface for the GA 155

Integration Module 157

The Genetic Algorithm 173

LIST OF TABLES

Table 4.1: Comparison of fitness scaling methods 39

Table 5.1: Comparison of CPU times and function evaluations 53

Table 5.2: Comparison of Polar and Cartesian coordinates for integration 53

Table 5.3: Comparison of indirect models for Mars transfer 54

Table 5.4: Comparison of direct and indirect formulations of the Mars transfer problem 55

Table 5.5: Comparison of indirect formulations of Maximum Energy Escape 56

Table 5.6: Direct solutions using BFGS and free cubic splines 56

Table 5.7: Maximum accuracies using different models and methods . . 58

Table 5.8: Best constraint tolerances using different models and methods 59

LIST OF FIGURES

Figure 2.1: Chosen polar coordinate system 10

Figure 2.2: Coordinates for the Earth-Moon System 15

Figure 3.1: Overview of problem formulation 22

Figure 3.2: Overview of possible Algorithms 23

Figure 4.1: Average log performance of DeJong's first function 46

Figure 4.2: Average performance of DeJong's second function 47

Figure 4.3: Average performance of DeJong's third function 48

Figure 4.4: Average performance of DeJong's fourth function 49

Figure 4.5: Average log performance of the modified F6 function 50

Figure 4.6: Average log performance of Goldberg's test function 51

Figure 6.1: Optimal Mars transfer trajectory 63

Figure 6.2: Control angle vs time for the optimal Mars transfer problem 64

Figure 6.3: Optimal Earth escape trajectory 65

Figure 6.4: Control angle vs time for the optimal escape problem 66

Figure 6.5: Comparison of control histories for direct and indirect methods 67

Figure 6.6: Average performance of the Earth-Moon problem 68

Figure 6.7: Average performance of the modified Moon model 69

Figure 6.8: Optimal Earth-Moon transfer trajectory 70

Figure 6.9: Optimal Earth-Moon control histories 71

NOMENCLATURE

$\hat{}$	Hat defines a corresponding unit vector
$\dot{}$	Dot denotes first time derivative
$\ddot{}$	Double dot denotes second time derivative
$\vec{}$	Top arrow defines a corresponding vector
$_1$	Subscript denoting secondary body constants
$_e$	Subscript denoting Earth physical constants
$_f$	Subscript denoting quantities at final time
$_k$	Subscript denoting quantities at k^{th} subproblem or iteration
$_m$	Subscript denoting Moon physical constants
$_{new}$	Subscript denoting the normalized variable
$_o$	Subscript denoting quantities at initial time
$_{ref}$	Subscript denoting a reference value for normalization
Au	Astronomical unit
$a(t)$	Thrust/mass of spacecraft at time t
C	Constraint vector of the nonlinear optimization problem
C_e	Vector of equality constraints
C_i	Vector of inequality constraints
d_1	Distance of the secondary body from the primary body

e	Specific energy of the spacecraft
E	Final specific energy of the spacecraft
F	Objective function of the nonlinear optimization problem
f	Vector of governing state differential equatons
H	Hamiltonian function
J	Performance index of the optimal control problem
K	Specific kinetic energy of the spacecraft
L	Integrand of the optimal control performance index
m	Spacecraft mass
R	Radius of the corresponding subscript
r	Radial position of the spacecraft from the chosen origin
T	Spacecraft thrust
t	Time variable
U	Specific potential energy of the spacecraft
$\mathbf{u}$	Control vector
u	Radial velocity of the spacecraft
v	Tangential velocity of the spacecraft
W	Weighting matrix of penalty multipliers for the constraints
$\mathbf{X}$	Design vector for the nonlinear optimization problem
$\mathbf{x}$	State vector
Λ	Vector of penalty multipliers
λ	Vector of lagrange multipliers for the optimal control states
μ	Gravitational constant (GM)
ν	Vector of constraint multipliers

ω	Angular velocity of the three body system
ψ	Vector of final state constraints
Φ	Penalty function for the nonlinear optimization problem
ϕ	Final time performance index of the optimal control problem
σ	Penalty constant
Theta	Control angle made by the thrust axis with respect to the positive tangential direction
θ	Clockwise angle made by the satellite with respect to the reference or the starting position

Abbreviations

BC	Boundary Conditions
BVP	Boundary Value Problem
BFGS	Broyden, Fletcher, Goldfarb and Shanno's hessian update
CPU	Central Processing Unit
DE	Differential Equations
DEC	Digital Equipment Corporation
GA	Genetic Algorithm
NASP	National AeroSpace Plane
NEM	Neighbouring Extremal Methods
N/A	Not Applicable
N/C	Not Convergant
PFM	Penalty Function Method
SQP	Sequential Quadratic Programming
SC	Stationarity Condition

2PBVP	Two Point Boundary Value Problem

ACKNOWLEDGEMENTS

I express my deep gratitude to Dr. Bion L. Pierson, who was my advisor and instructor. Many a time, his understanding of optimal controls and the trajectory problem helped me pull out of a mire of details. And the editorial net he provided during the preparation of this thesis is very much responsible for its current form. That I could finish my masters and this thesis in a year, is only due to his encouragement and the energy I derived from Master Yong Chin Pak's martial arts.

I would also like to thank Dr. James Cornette, Dr. Ping Lu and Dr. Roger Alexander for inspiring courses. My first and only exposure to genetic algorithms was a stimulating talk by Dr. S.S. Rao (at ADA, Bangalore), which consequently led to a part of this study.

Though it is not usual practice, I dedicate this work to the memory of my Biology teacher, Ms. Raktima Krishnaswamy, who inculcated in me an unshakable faith and wonder in the optimal and almost ideal nature of living systems and the process that evolves them.

Lalitesh

Lalitesh

December 8, 1991

ABSTRACT

Genetic algorithms and nonlinear programming for optimal low-thrust spacecraft trajectories

Lalitesh Kumar Katragadda

Under the supervision of Dr. Bion L. Pierson
Department of Aerospace Engineering and Engineering Mechanics
Iowa State University

The minimum thrust time problem for planetary transfer using low thrust spacecraft has assumed significance. However this problem is numerically sensitive. Three problems were chosen for study and testing different approaches. They are : a continuous thrust Mars transfer, maximum energy Earth escape, and a single-coast Earth-Moon transfer. Variations to the mathematical models gave limited success in providing better convergence. The multiplier penalty function approach gives better convergence for relatively poor initial guesses. Sequential Quadratic Programming showed convergence only with good initial guesses, while displaying ability to give high accuracy solutions. Genetic algorithms, in their first application to optimal trajectory problems, seem to offer the only general way to estimate the optimal trajectory, which was previously done using problem specific direct solutions. They succeeded in solving all the problems discussed with different thrust levels.

CHAPTER 1. INTRODUCTION

Projected Space Scenario

The next logical step in the human space adventure has been identified as the establishment and commercial utilization of a permanent Moon base. Manned ventures to Mars are also planned. In depth exploration of outer planets like Jupiter and Saturn is projected by sending satellites to permanently orbit these planets. These diverse goals require a launch system for low Earth orbit injection and a propulsion system capable of transferring satellites to other Solar bodies with low specific fuel consumption and at the same time permit flexible missions. Reusability of the propulsion module becomes a prerequisite for such an extended program from both flexibility and more importantly economic points of view. For example, a mission to the Moon and back would require carrying various cargo modules (including human) to the Moon, possibly landing on the Moon using the same propulsion system, and returning to Earth by a specified time. The mission would be dictated by this cargo and the time of launch and arrival. The effect of these changes on trajectory and fuel requirements cannot be ignored. A more exotic example would be the exploration of Jupiter; here, the satellite would be expected to change orbits to study features (features and hence orbits which cannot be predicted from Earth) and possibly transfer to an orbit around a planetary moon.

Recently a consensus has emerged as to the technology to apply to this problem. The low Earth orbit injection of the propulsion module, cargo and fuel is proposed to be carried out by a reusable launch system. Currently, the shuttle and its proposed updated derivatives are suitable. However, the proposed NASP (*National AeroSpace Plane*) is expected to take over the task and provide the full flexibility envisaged. The propulsion module is to be a nuclear powered system. These engines would be either a low thrust electric propulsion system or a medium to high thrust nuclear thermal systems. The terms low, medium and high refer to the thrust-to-weight ratio (which directly translates to g force due to the thruster) of the total spacecraft weight. Low thrust refers to levels below 0.1 g (usually 10^{-3} to 10^{-4} g's), medium spans the accelerations between 0.1 g to 1 g, and all levels higher than 1 thrust-to-weight constitute high thrust. This classification though adopted here is by no means standard. However, all references to low thrust in this thesis shall include medium and high thrust levels, since the problems associated arise from the same mathematical models (but have different numerical properties). Note that nuclear thermal rockets have been reported [1] to have thrust levels of the order of 1 g or greater and hence they can be used for landing missions on the Moon or Mars.

Low Thrust Transfers

Low thrust (also called *electrical propulsion* or *nuclear propulsion*) refers to a propulsion mode where the energy to eject the propellant is obtained from a source external to the propellant. Usually this energy source is a nuclear power plant or a chemical cell. Unlike conventional chemical propulsion, this energy is constant and more importantly virtually unlimited for the purposes of the engine. This en-

ables low but continuous mass ejection at a very high velocity for long durations. This implies a large saving in required propellant mass for the same total impulse. Given this power fixed, energy unlimited propulsion system, the trajectory planning task now translates to determining the orientation history of the thruster along with switching times of the mass flow (or thrust) to minimize spacecraft mass which directly translates into minimum cost. This involves considering a perturbed model of the spacecraft for prolonged periods of time requiring efficient trajectory integration, since we no longer have a closed form solution for the satellite orbit even as a patched conic approximation. The low thrust system has been studied for a variety of space missions, the chief of which can be classified as:

- Orbital transfer. Transferring from one elliptic orbit around a large central attracting body to another orbit around the same body to specify certain terminal conditions. These conditions can arise as a geostationary parking or other specified elliptic orbits.

- Hyperbolic escape/ capture. Achieving escape (or positive total specific energy) starting from an initial elliptic orbit which is usually a low circular parking orbit. The capture problem is to reach a specified elliptic orbit from a given escape condition. The effect of other bodies like the Moon or the Sun cannot be ignored for an accurate estimate.

- Interplanetary transfer. Transferring from one escape condition with respect to the 'first' planet to a hyperbolic end condition with respect to the 'second' planet under the influence of a third body (the sun). The Earth Moon transfer is included here since the problem is the same.

The objective in all these cases is almost always to minimize the amount of fuel mass spent. Variations can include an additional constant mass decrement. For example, on a manned mission, conservatively 1.5 kg per astronaut [2] are spent. Several variations of these problems are also introduced owing to the conditions imposed on the thruster. These could be:

- Continuous thrust on. The thruster is never switched off whereby the problem is to minimize total time of mission.
- Limited switching. The number of switchings, the minimum coasting (no thrust) time, the engine on interval or a combination of these could be constrained depending on the engine technology.
- Different thrust levels. An interesting variation is considering two levels of thrust; a constant high thrust and a much lower (and transient) thrust which manifests itself after the high thrust is switched off (possibly due to cooling down requirements).

Our Scope

The objective of this study in a broad outline has been to develop general algorithms to solve the minimum (engine on) time problem for interplanetary transfer. A few typical problems were deeply studied for a better understanding of the associated problems and a better physical feel. The chief concern has been to get initial estimates from which convergence to a desired accuracy is tractable using existing algorithms and the efficiency with which this can be done without specializing the

parameters to the problem (though this has also been done to understand the machination of some problems). This goal is made difficult by the fact that the full transfer problem is very sensitive, and the initial estimate itself needs to be reasonably accurate. Efficiency does not seem to be a primary concern since real time application is unlikely and the runs do not use excessive computation to begin with. But efficiency is a measure of the strength of an algorithm and associated problems like trajectory following and full mission planning systems will depend on some of the efficiency considerations, though these are problems not addressed in this thesis.

The approach to these problems has been two pronged. One is to develop and identify variations of the necessary conditions of optimality which show better numerical behavior. The other is to identify parameters and variations to existing algorithms which would find the optimal solution. All three attributes of a numerical algorithm, sensitivity, convergence and efficiency were studied. Memory was not considered since there is no dynamic memory growth and the code and data size are not significant.

CHAPTER 2. PROBLEM MODELING

This chapter elaborates on the problems chosen and the models used or developed for solving them. Each of the problems serves to increase the understanding of the complete problem by highlighting a few of its characteristics. The complete problem is defined as finding the optimal, minimum engine on time trajectory between two planetary bodies, given specific propulsion characteristics. Simplifications and approximations can be generally grouped into three categories:

1. Eliminate factors not significant in this particular problem:

   - Ignore solar pressure and radiation effects on spacecraft and propulsion system.
   - Ignore effects of minor bodies like asteroids and solar dust.

2. Remove details with little effect on problem complexity, though affecting results. It is felt in making such assumptions that the methods developed to solve the simplified problems will solve the unsimplified ones:

   - Coplanar orbits assumption. This assumption reduces the number of state and costate equations. The equations involved are at least as stable and the quantities ignored (z, $\dot{z}$) vary less in comparison to others.

- Restricted three body assumption. The two large masses are assumed to behave as fixed relative to each other with constant rotation. Relaxing this assumption would involve no major change in the problem formulation.
- Ideal engine performance is assumed. Transient engine behavior is also ignored. The engine is assumed to have constant mass flow and thrust when on, but no effect on the spacecraft when off.
- Navigation errors and other state estimation errors are ignored as constituting a control problem.

3. Reduce complexity of the problem to isolate specific features of the problem. The resulting problems are chosen as test beds for various methods and their variations in order to isolate or generate potential candidate codes for more difficult problems. The problems outlined below were chosen for this study :

   - Continuous thrust-on transfer from a given state to a specified state under gravitational influence. This is a simplistic problem which nevertheless provides confidence in the numerical methods, helps in weeding out unsuitable ones or, modifying them to solve problems of this nature. It also allows us to evaluate and verify support code like the numerical integration module.
   - Maximum energy escape from a low altitude parking orbit. This problem gives a fair idea of how escape trajectories of state and control angle will look like for the complete problem and enables comparisons of control parameterization effectiveness with the (indirect) optimal solution. Sensitivity is also a significant issue in this case.

- Escaping from low Earth parking orbit to low Moon parking orbit with restriction of a single engine-off phase. Besides giving estimates of the fuel consumption and a view of how an optimal trajectory is likely to look, this problem incorporates most of the difficulties of a complete problem.

Appendix A gives the values chosen for the constants not explicitly listed below. Some of them were simply adapted from a previous thesis [7] on a similar topic for comparison purposes.

Direct and Indirect Formulation

Given a system described by differential equations (*State DE*)

$$\dot{\mathbf{x}} = f(\mathbf{x}(t), \mathbf{u}(t), t); \quad \mathbf{x}(t_o), \quad t_o \leq t \leq t_f; \quad and \;\; \psi(\mathbf{x}(t_f), t_f) = 0 \tag{2.1}$$

with state $\mathbf{x}(t) \in \Re^n$ and constraints $\psi(\mathbf{x}(t_f), t_f) \in \Re^p$ $(p < n)$, an optimal control problem can be defined [3] as finding $\mathbf{u}(t)$ to maximize (or minimize)

$$J = \phi(\mathbf{x}(t_f), t_f) + \int_{t_o}^{t_f} L(\mathbf{x}(t), \mathbf{u}(t), t)\, dt \tag{2.2}$$

where the control input $\mathbf{u}(t) \in \Re^m$. Defining $\nu \in \Re^p$ as the multiplier for $\psi(\mathbf{x}(t_f), t_f)$ and the Hamiltonian as

$$H(\mathbf{x}, \mathbf{u}, t) = L(\mathbf{x}, \mathbf{u}, t) + \lambda^T f(\mathbf{x}, \mathbf{u}, t), \quad \lambda \in \Re^n;$$

we can apply variational analysis using the Lagrange multiplier approach to obtain (in addition to equations (2.1)) the following necessary conditions [4].

$$Costate\ DE \;\; : \;\; -\dot{\lambda} = f_{\mathbf{x}}^T \lambda + L_{\mathbf{x}}, \;\; t \leq t_f \tag{2.3}$$

$$Stationarity\ Condition(SC)\ :\ H_{\mathbf{u}} = L_{\mathbf{u}} + f_{\mathbf{u}}^T \lambda = 0 \tag{2.4}$$

$$Boundary\ Conditions(BC)\ :$$

$$\lambda^T d\mathbf{x}\mid_{t_o} - H d\mathbf{x}\mid_{t_o} = 0 \tag{2.5}$$

$$(\phi_{\mathbf{x}} + \psi_{\mathbf{x}}^T \nu - \lambda)^T \mid_{t_f} dx(t_f) + (\phi_t + \psi_t^T \nu + H)\mid_{t_f} dt_f = 0 \tag{2.6}$$

The solution of the necessary conditions gives the optimal control input subject to verifying the sufficient conditions. This approach is also known as the *indirect method* since the control is obtained from the costates which are not present in the problem statement.

Another solution method would be to parameterize the control time history $\mathbf{u}(t)$ using a chosen number of real values and find these along with other unknowns like initial and final times to minimize the performance index (2.2) while satisfying equations (2.1). This is known as the *direct method* for evident reasons. Some examples of parameterization would be splines, bezier fits, truncated taylor and fourier series. The parameterizations used for specific problems are discussed in Chapter 5.

The indirect method is generally known to yield a more accurate solution with low constraint tolerances, where as the direct method method is numerically more tractable but has suboptimal properties owing to restriction of the control time history scope by finite parameterization. Note that for the indirect case, solving for $\mathbf{u}(t)$ from the necessary conditions is equivalent to finding the initial costates $\lambda(t_o)$, which completely define the state and control trajectories; given time bounds and states. The primary objective is to obtain the indirect solution. Besides the advantages mentioned it also gives a *dynamic control law*, since $\mathbf{u}(t)$ is a function of state and costates, Equation (2.4). And the costates are governed by a known dynamic relation,

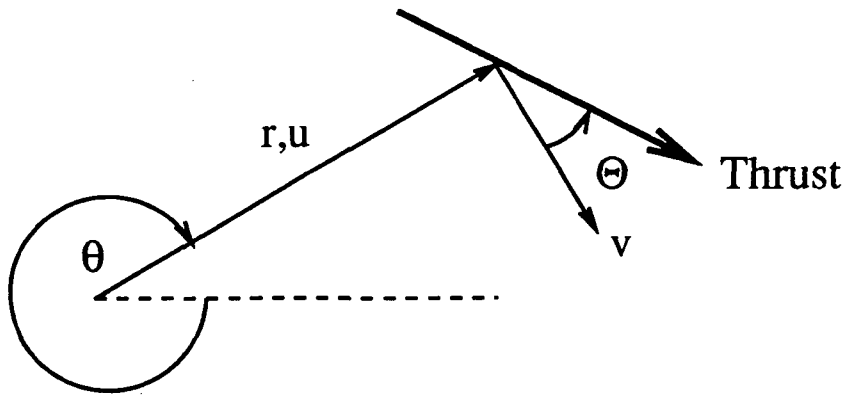


Figure 2.1: Chosen polar coordinate system

Equation (2.3).

Continuous Thrust Earth-Mars Transfer

This problem [5] involves solving for the minimum time coplanar transfer of a low thrust spacecraft from an approximate Earth escape condition to a similar condition with respect to Mars. All external forces except the Sun's gravitation and engine thrust are neglected. The orbits of Earth and Mars are assumed to be circular with the mean semi major axes for the radii. The two body approximation of the spacecraft in a polar coordinate system is given by the state equations:

$$\begin{aligned} \dot{r} &= u \\ \dot{u} &= \frac{v^2}{r} - \frac{\mu}{r^2} + a(t) \cdot \sin\Theta \\ \dot{v} &= -\frac{uv}{r} + a(t) \cdot \cos\Theta \end{aligned} \tag{2.7}$$

$$thrust\ acceleration,\ a(t) = \frac{T}{m_o - \dot{m}t}\ ,$$

$\mathrm{u}(t) = \Theta(t)$ is chosen with respect to the local horizon as in Figure 2.1. 'r' gives the radius, 'u','v' give the radial and circumferential velocities. The spacecraft's angular

position 'θ' is not considered since it is not specified and decoupled from the rest of the variables. The performance index $J = \int_{t_o}^{t_f} 1 \cdot dt$ and the normalized parameters [5] :

$$\mu = 1.0, \quad m_o = 1.0, \quad \dot{m} = 0.07487, \quad T = 0.1405, \quad T/\ weight\ |_{t_o} = 0.9 \times 10^{-4}$$

Initial state constraints, $x(t_o)$:

$$\begin{pmatrix} r_o \\ u_o \\ v_o \end{pmatrix} = \begin{pmatrix} 1.000 \\ 0.000 \\ \sqrt{\mu/r_o} \end{pmatrix} \quad \& \tag{2.8}$$

Final state constraints, $\psi(\mathbf{x}(t_f), t_f)$:

$$\begin{pmatrix} r_f \\ u_f \\ v_f \end{pmatrix} = \begin{pmatrix} 1.525 \\ 0.000 \\ \sqrt{\mu/r_f} \end{pmatrix} \tag{2.9}$$

For the indirect approach, equations (2.4),(2.3), respectively yield :

$$\tan\Theta = \lambda_u/\lambda_v$$

$$\begin{aligned} \dot{\lambda}_r &= \left(\frac{v^2}{r^2} - 2\frac{\mu}{r^3}\right) \cdot \lambda_u - \frac{uv}{r^2} \cdot \lambda_v \\ \dot{\lambda}_u &= -\lambda_r + \frac{v}{r} \cdot \lambda_v \\ \dot{\lambda}_v &= -2\frac{v}{r}\lambda_u + \frac{u}{r} \cdot \lambda_v \end{aligned} \tag{2.10}$$

The direct solution is now obtained by minimizing J while satisfying equations (2.7) and the state constraints as in (2.8,2.9).

The indirect form is solved by satisfying (2.7),(2.10) and (2.8). The final constraints however, assume different forms depending on the variations chosen. The constraints are derived from the boundary condition (2.6) after eliminating ν :

1. Original form. Terminal constraints are as in (2.9) and

$$H(t_f) = 0. \tag{2.11}$$

2. Fix the final time and maximize the final radius [3] by choosing $\phi(\mathbf{x}(t_f), t_f) = r(t_f)$. Solve the resulting subproblems by changing t_f until $(r(t_f) - r_f)$ is within tolerance. Each subproblem is an optimal control problem and the final one gives the solution. The final constraints for the subproblem are derived as :

$$\begin{aligned} u_f &= 0.0 \\ v_f &= \sqrt{\mu / r(t_f)} \\ \lambda_r(t_f) &= 1 + \frac{\lambda_v}{2}\sqrt{\frac{\mu}{r^3(t_f)}} \end{aligned} \tag{2.12}$$

3. Similar to the above problems. Except, final constraints are the last two constraints in (2.9) and the last constraint is modified to

$$\lambda_r(t_f) = 1 + \frac{\lambda_v}{2}\sqrt{\frac{\mu}{r_f^3}}, \tag{2.13}$$

where r_f is defined in (2.9). This shows faster convergence properties and retains optimal property since (2.13) linearly converges to (2.12) as $r(t_f) \to r_f$.

Maximum Energy Earth Escape

The objective here is to find the control input such that the satellite attains the maximum possible total energy. This is similar to a minimum time escape, and a problem of finding the minimum time required for a given total energy would

yield identical results. The state and costate equations are again given by equations (2.7),(2.10) and the performance index is the negative of the total specific energy with respect to Earth which would be constant in absence of propulsion [6]:

$$J = -\left(\frac{u^2(t_f) + v^2(t_f)}{2} - \frac{\mu}{r(t_f)}\right). \tag{2.14}$$

The initial conditions are given by:

$$\begin{pmatrix} r_o \\ u_o \\ v_o \end{pmatrix} = \begin{pmatrix} R_e + 315\ km \\ 0.0 \\ \sqrt{\frac{\mu_e}{r_o}} \end{pmatrix} \tag{2.15}$$

The values for R_e, μ_e and engine specifications are given in the aforementioned Appendix A. The direct problem is solved by choosing the parameterized control values to minimize index (2.14) while satisfying equations (2.7),(2.15). The indirect problem however is unbounded above when index (2.14) is minimized, indicating that this may be amenable as a maximization problem. Hence the performance index for indirect problems is:

$$J = \frac{u^2(t_f) + v^2(t_f)}{2} - \frac{\mu}{r(t_f)}, \tag{2.16}$$

which shows convergence to a maxima and validates the hypothesis. It is notable that the performance index will manifest itself only in the terminal costate constraints which are obtained from equations (2.6),(2.16):

$$\begin{aligned} \lambda_r(t_f) &= \frac{\mu}{r^2(t_f)} \\ \lambda_u(t_f) &= u(t_f) \\ \lambda_r(t_f) &= v(t_f) \end{aligned} \tag{2.17}$$

This indicates a high degree of sensitivity to initial costate values which will be the optimization variables. The physics of the problem however shows that the

performance index is an energy *integral.* The performance index is redefined as :

$$J = \frac{u^2(t_f) + v^2(t_f)}{2} - \frac{\mu}{r(t_f)} + \int_{t_o}^{t_f} a(t)\,(u \sin\Theta + v\cos\Theta)\,dt \tag{2.18}$$

Appendix B shows the corresponding derivations, including all the modified equations. The optimality condition and costates are now:

$$\tan\Theta = \frac{\lambda_u + u}{\lambda_v + v}$$

$$\begin{aligned}
\dot{\lambda}_r &= \left(\frac{v^2}{r^2} - 2\frac{\mu}{r^3}\right)\cdot\lambda_u - \frac{uv}{r^2}\cdot\lambda_v \\
\dot{\lambda}_u &= -\lambda_r + \frac{v}{r}\cdot\lambda_v - a(t)\cdot\sin\Theta \\
\dot{\lambda}_v &= -2\frac{v}{r}\lambda_u + \frac{u}{r}\cdot\lambda_v - a(t)\cdot\cos\Theta
\end{aligned} \tag{2.19}$$

The terminal constraints are unchanged. The total energy is effectively included twice in the new performance index. This change as seen later demonstrates superior stability and hence convergence. This also translates to less sensitivity to initial costates and an unusual scaling property.

Another independent variation is maximizing index (2.16) (or minimizing (2.14)) in lieu of satisfying the terminal constraints. One would expect that both forms (or a combination thereof) will lead to the same solution in the limit of convergence. But this is not the case as will be seen. The indirect Earth escape problem with its four variants can be summarized as follows by the necessary conditions needed to be satisfied:

- State equations (2.7)
- Initial conditions (2.15)
- Costate equations (2.10) OR (2.19)
- Final constraints (2.17) AND/OR minimize index (2.14)

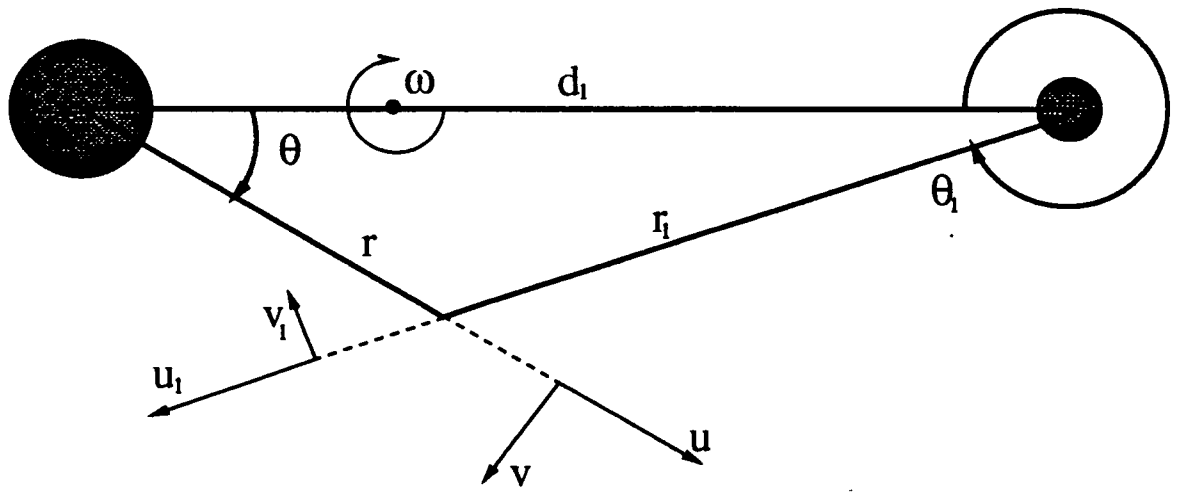


Figure 2.2: Coordinates for the Earth-Moon System

Optimal Earth-Moon Transfer

The objective here is to find the minimum engine on time for a spacecraft with only one allowable coast phase. The craft is initially in a low Earth parking orbit and the final desired state is a low Moon parking orbit. The Earth-Moon system for this problem is assumed to be acting as a restricted three body system with no influence of the sun; Figure 2.2.

The coordinate frame chosen is a Earth (or Moon) centered right handed, rotating polar coordinate frame with the principal axis fixed to the Earth-Moon center line. All angles are measured clockwise. The transformation from Earth centered to Moon centered rotating systems as described in Appendix C is given by:

$$
\begin{aligned}
r_1 \cos\theta_1 &= d_1 - r\cos\theta \\
r_1 \sin\theta_1 &= -r\sin\theta \\
r_1 &= \sqrt{r_1 \cos^2\theta_1 + r_1 \sin^2\theta_1}
\end{aligned}
$$

$$
\begin{aligned}
A &= v\sin\theta - u\cos\theta \\
B &= -(u\sin\theta + v\cos\theta) + d_1\omega \\
u_1 &= A\cos\theta_1 + B\sin\theta_1 \\
v_1 &= -A\sin\theta_1 + B\cos\theta_1
\end{aligned} \tag{2.20}
$$

The state equations are derived in Appendix C. They are simplified as:

$$
\begin{aligned}
\dot{r} &= u \\
\dot{u} &= \frac{v^2}{r} - \frac{\mu}{r^2} - \frac{\mu_1}{r_1^3}(r - d_1\cos\theta) - \frac{\mu_1}{d_1^3}d_1\cos\theta + r\omega^2 + 2v\omega + a(t)\sin\Theta \\
\dot{v} &= -\frac{uv}{r} + d_1\sin\theta\cdot\left(-\frac{\mu_1}{r_1^3} + \frac{\mu_1}{d_1^3}\right) - 2u\omega + a(t)\cos\Theta \\
\dot{\theta} &= \frac{v}{r}
\end{aligned} \tag{2.21}
$$

The costate equations are simplified as :

$$
\begin{aligned}
\dot{\lambda}_r &= -\lambda_u\left(-\frac{v^2}{r^2} + \frac{2\mu}{r^3} - \frac{\mu_1}{r_1^3} + T1\cdot\frac{\partial r_1^2}{\partial r} + \omega^2\right) - \lambda_v\left(\frac{uv}{r^2} + T2\cdot\frac{\partial r_1^2}{\partial r}\right) \\
\dot{\lambda}_u &= -\lambda_r - \lambda_v\left(-\frac{v}{r} - 2\omega\right) \\
\dot{\lambda}_v &= -\lambda_u\left(\frac{2v}{r} + 2\omega\right) + \lambda_v\frac{u}{r} - \frac{\lambda_\theta}{r} \\
\dot{\lambda}_\theta &= -\left[\mu_1\left(-\frac{d_1}{r_1^3} + \frac{1}{d_1^2}\right)\cdot(\lambda_u\sin\theta + \lambda_v\cos\theta)\right] - \frac{\partial r_1^2}{\partial\theta}(\lambda_u T1 + \lambda_v T2)
\end{aligned} \tag{2.22}
$$

Where

$$
\begin{aligned}
T1 &= 1.5\cdot\frac{\mu_1}{r_1^5}(r - d_1\cos\theta) \\
T2 &= 1.5\cdot\frac{\mu_1}{r_1^5}d_1\sin\theta \\
\frac{\partial r_1^2}{\partial r} &= 2(r - d_1\cos\theta) \\
\frac{\partial r_1^2}{\partial\theta} &= 2rd_1\sin\theta \\
\omega &= \sqrt{\frac{\mu + \mu_1}{d_1^3}}
\end{aligned}
$$

State constraints:

$$\begin{pmatrix} r_o \\ u_o \\ v_o \end{pmatrix} = \begin{pmatrix} R_e + 315\ km \\ 0.000 \\ \sqrt{\mu/r_o} \end{pmatrix} \quad \& \quad \begin{pmatrix} r_f \\ u_f \\ v_f \end{pmatrix} = \begin{pmatrix} R_m + 100\ km \\ 0.000 \\ \sqrt{\mu/r_f} \end{pmatrix} \tag{2.23}$$

Note that the initial constraints refer to the Earth centered coordinate system and the final time constraints refer to Moon centered coordinate system. The subscript '$_1$' in the state and costate equations refers to the secondary body. The other quantities refer to the primary body. The primary body is either the earth or the moon as chosen below. The mission is outlined as follows:

- Start (in Earth centered coordinate system) with engine on at $t_o = 0$ till unknown time t_1 and switch engine off. The initial state is defined in equation (2.23) and θ_o is an unknown.
- Coast till unknown time t_2.
- Transform state to Moon centered coordinate system using (2.20) and switch engine on till terminal constraints are reached at unknown time t_f

The objective of the indirect problem is to determine θ_o, t_1, t_2 and the control time history which is completely defined by the costates at t_1, and t_2 . The direct solution for this problem is not obtained since this problem is to demonstrate the code's effectiveness and our primary objective has been to obtain the indirect solution. Only the genetic algorithm was able to produce an initial guess. The other algorithms could not improve on this guess, since the final state is highly sensitive to any changes in the values at t_o. Hence starting with the final constraints and

integrating backwards until the engine is switched off, gave better results. The objective here is to match the resulting state with that of the forward integration phase. Kleuver [7] arrived at this conclusion with similar reasons. This will be referred to as *the modified three body model.* Further modifications gave better convergence:

- Since initial and final angular positions are free, the corresponding costates are zero. The costate λ_θ at t_o has been always found to be zero. However the other λ_θ is either zero or fairly constant, depending on the modification used.

- Using the modified three body model, the end time of the coast phase is determined so that the radial position matches that of the moon escape phase. This reduces a variable and increases convergence.

- The angular position at moon orbit can be determined by iterating so that it matches that of the coast phase. Though this increases computation, preliminary results show increased convergence, since the number of variables is reduced and now, only the velocities remain as the constraints.

Normalization of the Variables

Now we can proceed to normalize the variables in order to keep the quantities involved of the same magnitude to prevent loss of significant digits and associated numerical difficulties. This process is normally referred to as non-dimensional analysis and the approach is identical; but instead of changing the equations we proceed to change the values associated since this would give the flexibility to experiment with various reference systems and more importantly change the normalization factors during the course of the problem as the coordinate systems are changed.

The following reference systems were selected employing the thumb rule that all state variables should remain within one magnitude as far as possible. Note that in this process the non-dimensional time may assume values one or two magnitudes higher. But since time does not appear explicitly anywhere except in the mass equation in a non-additive form, there is no loss of precision due to this. All initial quantities are normalized with respect to the reference quantities according to their dimensional definitions as discussed later:

1. Earth-Mars transfer. The Earth-Sun distance (1 Au), the Sun's gravitational constant μ and the initial spacecraft mass were chosen as the reference parameters. All other quantities were dimensionally scaled according to these.

2. Earth escape problem. The Earth's radius, gravitational constant and the initial spacecraft mass were the reference units.

3. Earth-Moon problem. Same as above. However several other possibilities exist which have not been explored.

The scaling is done as follows:

$$L : length, \qquad T : time, \quad M : mass$$

$$Given \qquad L_{ref}, \quad \mu_{ref} \quad and \quad M_{ref}$$

$$T_{ref} = \sqrt{\frac{L_{ref}^3}{\mu_{ref}}}$$

$$\mu_{new} = \mu \cdot \frac{T_{ref}^2}{L_{ref}^3}$$

$$
\begin{aligned}
t_o,\ t_f,\quad t_{new} &= \frac{t}{t_{ref}} \\
R_e,\ \ldots, L_{new} &= \frac{L}{L_{ref}} \\
m_{new} &= \frac{m}{M_{ref}} \\
\dot{r},\ u,\ v_{new} &= v \cdot \frac{T_{ref}}{L_{ref}} \\
\ddot{u},\ \dot{v}_{new} &= \dot{v} \cdot \frac{T_{ref}^2}{L_{ref}} \\
\theta &: \ nondimensional \\
\omega_{new} &= \omega \cdot T_{ref}
\end{aligned}
\tag{2.24}
$$

CHAPTER 3. NUMERICAL TECHNIQUES

Overview of Methods

This chapter outlines the algorithms used to solve the optimal control problems presented in Chapter 2. Classifying the various methods needs a further distinction between the problem formulation and the numerical algorithm used.

- The problem formulation depends on on whether the necessary conditions are applied to the direct or the indirect problem. The indirect version can be solved by implicitly satisfying a combination of the necessary conditions. This gives rise to three major forms [3]. Parameterization of the control defines the form in the direct method. Any form can use all of the mathematical models of the problem falling in its domain of definition.

- Two examples of the numerical algorithm used are SQP (*Sequential Quadratic Programming*) and collocation schemes. Each algorithm can be used to solve more than one problem arising from more than one problem formulation. Conversely, more than one algorithm can be used to solve the same problem.

Figure 3.1 gives an overview of the possible formulations, along with the scope of various algorithms. Figure 3.2 summarizes the possible algorithms. The following sections describe the salient features of the formulations and related algorithms

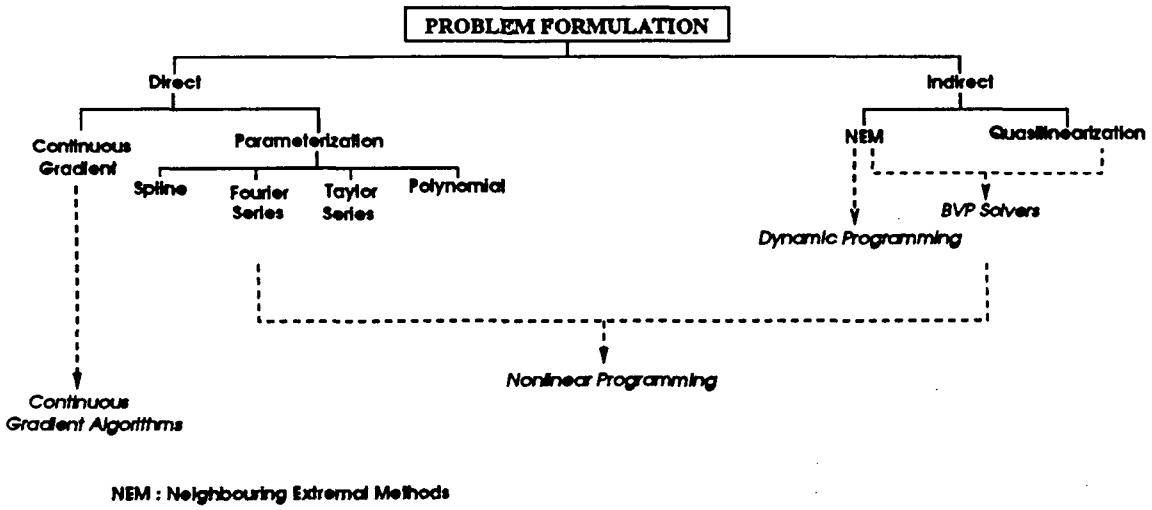
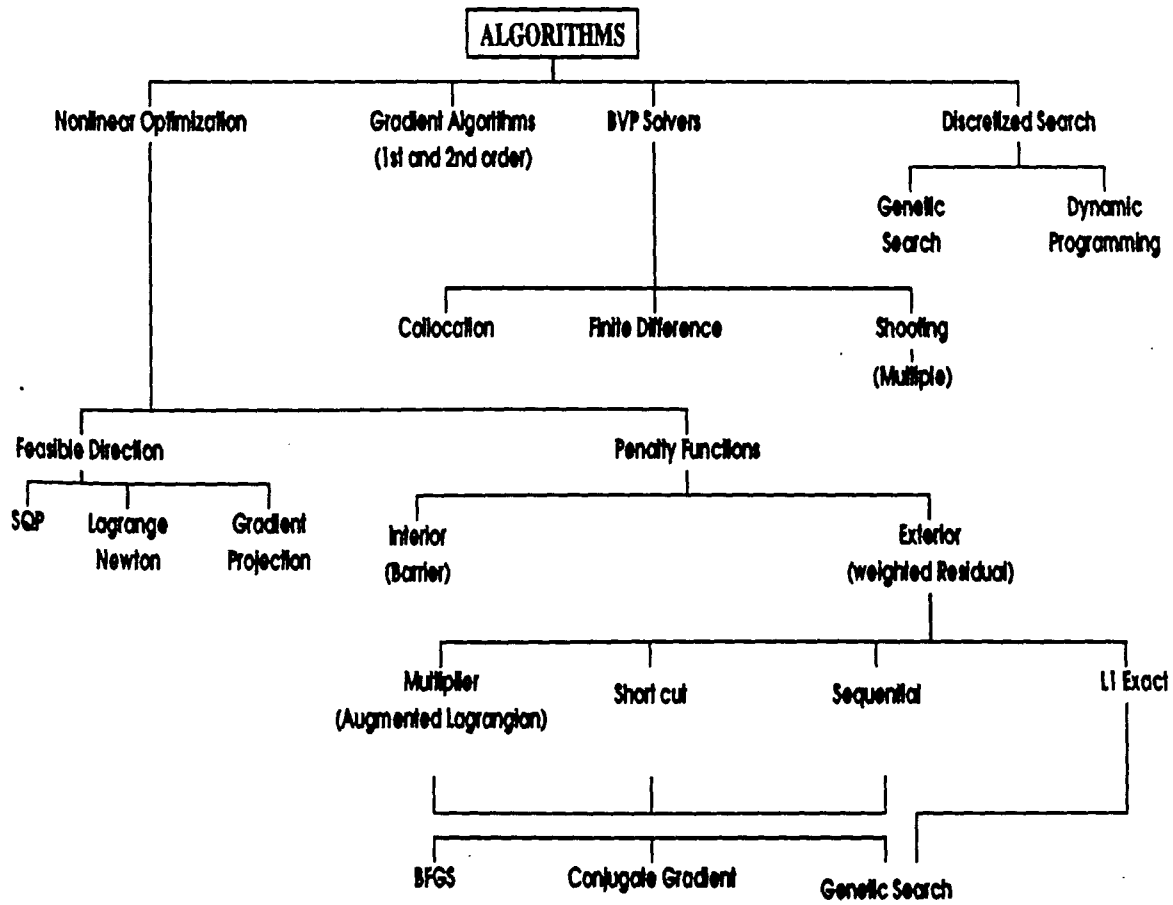


Figure 3.1: Overview of problem formulation

tested with the low thrust problem. Others are defined briefly. Since, genetic search algorithms represent a relatively new field and apparently have not been explored for optimal trajectory problems, this topic is dealt with in more detail in Chapter 4.

Parameterization

The direct problem objective is to minimize a performance index while satisfying the state constraints. However, $\mathbf{u}(t) = \Theta(t)$ is a function of time and generally cannot be represented with a finite number of real values. This necessitates representing $\Theta(t)$ as a combination of known continuous functions with unknown coefficients or parameters. These parameters now become the design variables. The control space is hence discretized with the associated artificial stiffness or restrictions on the control space, hence loss of optimality. The choice of these functions is of primary importance since it affects:



BVP Two Point Boundary Value Problem

SQP Sequential Quadratic Program

BFGS Broyden, Fletcher, Goldfarb and Shanno

Figure 3.2: Overview of possible Algorithms

- Accuracy. The truncation error due to finite degrees of freedom of control. If for example the first five terms of the Taylor series are chosen, the terms in the optimal control corresponding to the higher order terms are lost. This places a theoretical bound on the accuracy of the solution.

- Precision. Adverse scaling of the coefficients due to improper function choice results in ill conditioning due to finite machine precision. For example, two fifth order polynomials represent the same control space from $0 < t < 10000$. Let one of polynomial be normalized with respect to [0,1] and the other be as is. The first three terms of the latter representation are lost on a machine with eight significant digits with half of the fourth term ineffective.

- Convergence. The problem may become very sensitive to changes in some coefficients and insensitive to others. For example, in $u(t) = a(b + ct)$, a change in a may result in loss of effectiveness of b and c. The apparent scaling of the control space may translate to an entirely different change in the solutions space due to high problem nonlinearity.

Mainly two parameterizations were tested:

1. Normalized polynomial in t' with $t' = t/(t_f - t_o)$. This method converges but shows poor convergence as compared to splines. One reason is that the higher the order the more weighting that is given to the right end of the time domain since a high degree polynomial is close to 0 until it is near 1.0 . Another reason seems to be that each coefficient affects the entire control space.

2. Free cubic spline; a spline with unspecified boundary slopes. This representation showed good convergence properties for the problems tested. But subop-

timality in representing the escape trajectory was observed since the optimal escape trajectory shows a large number of oscillations indicating the need for incorporating sinusoidal functions, like $\sin(a + bt)$, in the control.

Other parameterizations may easily be tested in the program framework. The results indicate that a carefully chosen set of functions can show sufficient optimality and good convergence.

The Indirect Method

The necessary conditions [2.1 (state DE), 2.3(costate DE), 2.4(SC), 2.5&2.6(BC)] constitute the two point boundary value problem (2PBVP). This problem can be solved by iterating on a nominal solution which implicitly satisfies one to three of these conditions. However, only three of the fifteen possibilities [3] have been conventionally explored. All the known algorithms iterate using successive linearization. The only other alternative is dynamic programming which can only solve very simple continuous domain problems due to exponential computation increase with refinement of domain discretization. Of the three aforementioned possibilities, two are indirect methods:

1. Neighboring Extremal Methods. The nominal solution satisfies the SC and DEs, leaving the BCs to be satisfied by iteration. Each trajectory is an extremal for some other problem in the neighborhood and hence the name. Guessing the initial unknowns in the states, costates, time intervals, and iterating to satisfy the terminal BCs are known as shooting methods. Several modifications like multiple shooting with discretized domains, unit solutions by perturbation and

backward sweep enable more stability, accuracy and other improvements. In general, these algorithms are highly sensitive but do give accurate results.

2. Quasilinearization. The nominal solutions satisfies the SC and possibly the BCs. The starting point is a guess for the state and/or costate history while satisfying some/or all the BCs. The resulting perturbation equations in state and costates give a sequence of linear two point boundary value problems.

The third possibility the *continuous gradient method*, is normally classified as a direct method. The nominal solution only satisfies the state and costate DEs. This involves a guessed control time history and iterating by integrating the state DEs forward and the costate DE backward to get a continuous gradient (H_u), which is used to satisfy the SC and BCs. High initial convergence is a property of these methods.

Boundary Value Problem Solvers

These are a class of algorithms which solve ordinary differential equations with constraints specified at more than one point of time. The necessary conditions derived for an optimal control problem lead to a 2PBVP, and hence fall in their domain. The collocation code, Colsys from Ascher, Christiansen and Russell [9] was the only two-point boundary problem solver tested. It would be classified as a Quasilinearization scheme. However, such a method has an inherent drawback of requiring a fixed final time. For example, to solve the variable time Earth-Mars transfer problem, the following scheme was used to change the end time t_f to meet the constraint of final radius. Each subproblem maximizes end radius, using a specified t_f:

1. Guess two end times and solve the subproblem for each of them.

2. Obtain a first estimate by linear interpolation against the constraint, as described in Moyer's [5] generalized Newton-Raphson approach. Solve for this estimate.

3. Use the two guesses and the first estimate solutions to get the next estimate for t_f by quadratic interpolation of constraint vs t_f, the *generalized Newtons method.* Solve using this estimate.

4. Check for convergence in steps 2 and 3. Otherwise continue from step 3.

The generalized Newton's method reduced execution time by more than half that obtained using the generalized Newton-Raphson technique. Other modifications, like using the previous solution as the initial solution for the new t_f, did not bring about major changes. Colsys failed to converge on the minimum energy escape problem. A suboptimal solution was obtained by splitting the $[t_o, t_f]$ time interval into a specified number (4 here) of parts. The problem was then solved for each of these intervals with the final state of the previous interval supplying the initial state for the next time interval. All the modifications described can be generally applied to other problems.

Nonlinear Constrained Parameter Optimization

This refers to a class of algorithms which minimize an objective function subject to linear and nonlinear, equality or inequality constraints. They are very flexible with respect to problem formulation modifications. Incorporating changes like variable initial and end times and bounds on state or control is easier when compared with

formulating these changes into an indirect problem or solving them using boundary value solvers. Conventional algorithms, however, need the objective function to be at least twice differentiable and convex in the region of the initial guess for guaranteed convergence. Extensive research [8] has made many problems solvable, regardless. The algorithms fall into two broad categories:

- Unconstrained minimizers where constraints are handled by penalty functions. These include descent methods like conjugate gradient and quasi-Newton methods like BFGS. BFGS is more widely used because of its superlinear local convergence, scaling properties enabling global descent and an efficient Hessian (the matrix of second derivatives) updating scheme which needs only the gradient to be evaluated at each step. BFGS was hence chosen for the penalty method. The objective is to find the design vector $\mathbf{X}$ to minimize $F(\mathbf{X})$ subject to the constraints $C_e(\mathbf{X}) = 0$ and $C_i(\mathbf{X}) \geq 0$. The following variants achieve the objective by minimizing a new unconstrained function Φ:

  1. Sequential penalty functions:

$$\Phi(\mathbf{X}) = F(\mathbf{X}) + \frac{1}{2}C^T W_k C, \quad C = 0 \;\; if \;\; C_i \geq 0; \quad k = 1..\infty \tag{3.1}$$

  Here, C includes both C_e and C_i, W_k is positive definite and the second norm $|W_k|_2 > |W_{k-1}|_2$. This sequence of unconstrained subproblems gives linear convergence and gives a theoretical optimum as $|W_k|_2$ uniformly tends to ∞. Usually, and for this study, W_k is chosen to be a diagonal matrix with equal coefficients in which case the penalty function reduces to:

$$\Phi(\mathbf{X}) = F(\mathbf{X}) + \frac{1}{2}\sigma_k C^T C; \quad k = 1..\infty, \quad \sigma_k > \sigma_{k-1} \tag{3.2}$$

2. Short cut penalty function. Using a single, large σ to solve a single subproblem:

$$\Phi(\mathbf{X}) = F(\mathbf{X}) + \frac{1}{2}C^T W C \tag{3.3}$$

For the W chosen above, this translates to:

$$\Phi(\mathbf{X}) = F(\mathbf{X})/\sigma + \frac{1}{2}C^T C \tag{3.4}$$

3. Multiplier penalty function:

$$\Phi(\mathbf{X}, \Lambda_k) = F(\mathbf{X}) - \Lambda_k^T C(\mathbf{X}) + \frac{1}{2}\sigma_k C^T C; \quad \Lambda_{k+1} = \Lambda_k - \sigma_k C(\mathbf{X}_k) \tag{3.5}$$

Where σ_k is increased only if the constraint satisfaction rate drops. This is known as the Powell-Hestenes multiplier update. Others, like Fletcher's update, use the BFGS Hessian to provide superlinear local convergence. The multiplier penalty function gives the advantage of obtaining the optimum in a finite number of subproblems with finite σ. This property [8] is affected by inducing an origin shift for the constraints which also moves the discontinuity in second derivatives due to inequality constraints away from the optimal solution.

4. L1 exact penalty function:

$$\Phi(\mathbf{X}) = F(\mathbf{X})/\sigma + \sum |C_e(\mathbf{X})| + \sum C_i(\mathbf{X})^-, \quad C_i(\mathbf{X})^- = \max(-C_i(\mathbf{X}), 0) \tag{3.6}$$

For a sufficiently high σ, this function gives single step-convergence to the constrained optimum [8]. Discontinuities in the gradient prevent the use of conventional algorithms. However, this function is ideally suited for genetic search algorithms, which does not use gradient information.

- Feasible direction schemes. Using a local quadratic model with linear constraints, we can either reduce the variable set by elimination to span the constraint free hyperspace or equivalently solve the problem using Lagrange multipliers [8]. A sequence of such problems is required for nonlinear functionals. These include *Sequential Quadratic Programming* (SQP) and the gradient projection methods. These algorithms, unlike the penalty functions possess quadratic (or superlinear) convergence properties by definition. Each subproblem is a quadratic model instead of a general nonlinear function. SQP, a well used and readily available code, was used for solving some of the defined problems.

Numerical Integration

All the nonlinear programming schemes and shooting methods require integrating a system of ordinary differential equations over the time domain for each value of the design vector or each iteration. In solving a full problem, this signifies a large number of these integrations, with stringent accuracy requirements to provide gradient information when required. Gear [10] gives an extensive discussion on various methods. Different problems have different lengths of time and accuracy requirements. Hence, to have both efficiency and flexibility, we need some kind of error control. Further efficiency accrues if the step size is changed dynamically. This is

possible by modifications to the widely used Runge-Kutta methods or the more recent multivalue methods. For this study, a variable step size and variable order multivalue method was coded for the following reasons:

- They are strongly stable, since they are predictor-corrector schemes.
- Unlike Runge-Kutta methods, an increase in order does not increase the number of evaluations per step. Even for orders as high as eight, the number of function (differential equation) evaluations is only three compared with four for a fourth order Runge-Kutta method.
- The overhead computation is comparable with Runge-Kutta. Step and order changes are computationally inexpensive.
- Increasing the maximum order entails adding additional coefficients only. The same code can also solve higher-order differential equations.

Error and Tolerance Scheduling

Some of the numerical methods applied require solving a set of subproblems to arrive at the solution, besides carrying out a numerical integration for each design vector. Since the subproblems are not the solutions, it is not necessary to solve them with the same accuracy and constraint tolerance. And the required accuracy for each integration can also be correspondingly scaled. Hence, a method was developed to start with coarse error and tolerances and later refine them until the required values are achieved. The initial value is chosen to at least yield convergence. However, very coarse initial tolerances will mean an increase in the number of subproblems. The method is outlined as follows:

1. Initialize the permissible error (in constraints or gradients). Set the integration error corresponding to this level. For example, a required gradient level of 10^{-2} means an integration error of 10^{-5} or lower. The constraint tolerances can be set independently but must be greater by at least one magnitude than the integration accuracy. Using penalty functions would change this strategy since the constraints and gradients are being scaled.

2. Solve the subproblem and estimate the errors. Decrease the error levels for the next subproblem by a determined amount (usually 0.1). If the error level is less than the required final level, set it to the specified level.

This method has been applied for some problems as a proof of concept, though the code structure allows a complete investigation. Similar strategies could be applied inside each subproblem. This was not done since it would constitute rewriting parts of standard code and would require extensive work and would detract from the focus of this thesis.

CHAPTER 4. GENETIC ALGORITHMS

Genetic algorithms (GAs) are randomized *population* based search techniques closely emulating the natural process of evolution. They are predominantly string or integer-based searches with each *member* of the population represented by a string of bits (*alleles*), alphabets or other enumerated forms. This string or member is known as the *chromosome.* The evolution process is punctuated by evolving a new population set from the previous set. Each of these population sets is known as a *generation.* Each member of the new population is derived from one or more members from the previous set. Hence, the new chromosome is the *child* of the *parent* chromosome(s) from the previous generation. This process of *reproduction* is driven by a *fitness* value associated with each chromosome. In this context, the chromosome is known as the *genotype* and the fitness which is the genotype's physical manifestation, is known as the *phenotype.* The problem specifics play a role in genetic algorithms only in decoding the chromosome and constructing its fitness value or phenotype. There are no restrictions on the domain of the decoded design space or the solution space. This flexibility and the robust nature of genetic algorithms makes them very powerful tools. Unlike dynamic programming and similar methods, they do not possess the curse of dimensionality. However, they are not as efficient as some of the specialized schemes like BFGS or SQP when applied to problems in

their domains. Hybridization or using specific problem properties to enhance GAs is known to restore efficiency without sacrificing too much robustness or flexibility.

The *adaptive* nature of these algorithms is used to:

- Search the solution space for a minimum (*optimization*).
- Continually adapt to a changing environment (*Classifier Systems*) like games or steady-state optimal control.

Hence, they can be used for a variety of problems like minimizing noisy functions, playing chess, designing gas turbines, and robot arm trajectory following. The primary references in this field are due to Holland [11] and his student Goldberg [13]. DeJong [12] did an extensive study on optimizing real-valued functions including near singular and discontinuous ones. Davis [14] gives a commentary on optimizing real-valued functions and a compilation of papers. One of first applications of GAs was in real time optimal control of pipeline scheduling [13]. There seems to be little work in the area of optimal control however, except for ongoing research on optimal robot arm trajectory following [14]. Rao [15] and Hajela [16] are investigating applications in aerospace design.

There are predominantly two processes which form the core of the evolutionary process:

- Crossover. A child produced using this process will have part(s) of its chromosome from one parent and the rest from the second parent. More than two parents are rarely used.
- Mutation. Mutation is a allele-based process, where the mutation of an allele implies replacing the existing value with a random value.

Both processes or *operators* are carried out with a specified probability of success. They affect a child only if they pass the probability test. Typical crossover probability is 0.8 per two children and a typical mutation probability is 0.01 per allele. Which means that on average 8 out of 10 children have been produced by the *mating* of more than one parent and 1 out of 100 alleles are mutated. The selection of parents is a weighted probability of their fitness. This process of evolution and the population-based nature is what differentiates genetic algorithms from the rest of the search and optimization techniques.

If pure crossover is used, the algorithm degenerates into a combinatorial search. If pure mutation is used, it degenerates into a random search. The primary construct being searched for by the GA is the best *schema*. A schema is a similarity template which can match more than one chromosome. For example, a bit string chromosome 100110 matches the schemata $10****$, $100*10$, 100110 and 61 more. The '$*$' represents the "don't care" logical value. Given a string of length l, there are 3^l possible schemata. A given chromosome matches 2^l schemata. The best solution is represented by a set of one or more best schema. Hence, the GA evolves the population by mixing schema of the superior individuals and weeding out unwanted schemata by assigning low survival to weak individuals. It is assumed that the superior individuals have more parts of the best schema. However, some good schemata may be masked in the weak individual and lost. Hence, mutation (and recently *diploidy*) is primarily responsible for maintaining a diverse pool of schemata. Crossover is used to combine existing ones. A host of operators based on these two basic ones have been developed to enhance reproduction.

Genetic Algorithms in Optimization

Our interest in genetic search is restricted to optimizing nonlinear functions with low noise. The solution is the best individual obtained from the entire search. By tradition, as in the code presented, GAs are used to maximize a function. A GA for optimization is described as follows:

1. Get an initial population from the user or by random string generation.

2. Decode the genotypes (strings) of the population and evaluate the fitness value (phenotype). In the GA code, a chromosome is a composite string where each binary substring represents a real number. The binary substring is decoded to an integer and then mapped to a given domain of real numbers.

3. Scale the fitness values so they are all positive. Several scaling techniques exist. Assign a survival probability to each individual in the population based on fitness. Usually, this probability is the fraction of an individual's fitness to total fitness.

4. Generate a new population. In general, a part of the population is cloned from the best of the previous population. The rest of it is generated by the reproduction process described above. The parents for reproduction are selected by random selection with probability as assigned in Step 3.

5. Check the termination criteria, for example, the number of new individuals produced, fitness difference between the best and the weakest, or computation time elapsed. If the process is not terminated, continue from step 2. Otherwise, return the best individual as the solution.

The GA code used in this study was written in C using a framework and data structures similar to ones used by Goldberg [13]. Several modifications were made to improve efficiency, mostly as suggested by Davis [14]:

- Reproduction. Steady state [14], without duplicate individuals. Using overlapping generations (delete last) to copy a fixed number of the best individuals alive from the previous generation. Also, making sure that no two individuals in the population are identical by string matching.

- Fitness scaling. Windowing (adding a constant to all the fitnesses), to make all fitnesses positive and to remove large common denominators. Optionally making the fitness difference between each two adjacent individuals uniform (*linear normalization*).

- Operators. Separation of mutation and crossover as separate operators. Adding new operators like two-point cross over and uniform list crossover [14].

- Parameterization. Interpolate *operator fitness* using given values.

Note that testing a genetic code involves averaging several runs of the code for the same initial parameters, because of their randomized, probabilistic nature. Further modifications to the genetic code were developed and tried during this investigation. Their efficacy could not be demonstrated because of limited computational power and time. The modifications are as follows:

1. To promote keeping the best individuals, sorting of the population based on fitness was done, and a specified number of superior individuals were retained as is.

2. Mutation rate in the literature refers to the average number of mutations per allele. However, empirical performance of GAs indicates the number of mutations per individual to be a better index. The rationale is that this index would give a consistent performance across a range of string lengths.

3. Reproduction without duplication. In the scheme described by Davis, each child has to be searched against the rest of the already produced population before being accepted. The high number of duplicates produced indicates a large number of searches of the order of the square the population size. This is justified when the fitness evaluation time is long, since the benefits accrued are outweighed by the extra computation. To get similar benefits for fitness functions with small evaluation times, a set of rules was developed to eliminate most of the duplicates. A survey of duplicates indicated that most duplicates are produced by crossover of parents without mutation when the crossed over material is identical. This can be detected in three stages:

   (a) Check if mutation occurs. If it does not and crossover has not taken place or both the parents were the same, then a duplicate child is found, and a sibling is discarded; both are discarded if the parent has been kept alive.

   (b) Next step is to do a string comparison against the parents and discard them if the parents are being kept alive.

   (c) Compare the child against all children produced by the parents.

   The last test has not been implemented. It is observed that a majority of duplicates are eliminated using the first two techniques. Or alternately, if duplication is allowed; the fitness value of the duplicate can be copied on to the

child, saving decoding and function evaluation. These tests, when performed before a full search, decreases the number of full searches. Further investigation as to the source of duplicates seems to be a promising field.

4. Fitness scaling by *modified windowing.* Windowing has the disadvantage that an individual very superior relative to the rest dominates reproduction, and soon brings about premature convergence. However, linear normalization makes convergence extraordinarily slow by destroying relative information, which is not acceptable due to high computation cost. Hence, a scaling method which uses the basic windowing and introduces an additional *specified* difference between each individual may prove beneficial. Typically, the difference is the average fitness. Table 4.1 demonstrates the effect of such a change. Hence, the modified windowing scheme retains the distribution without completely suppressing the weak individual's selection chances.

Table 4.1: Comparison of fitness scaling methods

A set of population fitnesses					
Original Fitness	-4.50	-3.20	0.00	10.10	100.00
Windowing	0.10	1.40	4.60	14.70	104.60
Selection Probability %	0.08	1.12	3.66	11.72	83.41
Linear Normalization	1.00	2.00	3.00	4.00	5.00
Selection Probability %	6.67	13.33	20.00	26.67	33.33
Modified Windowing	25.18	51.56	79.84	115.02	230.00
Selection Probability %	5.01	10.27	15.91	22.93	45.85

5. Combined operators. Instead of using segregated operators as suggested by Davis [14], combined operators like mutation with uniform list crossover were

tried out. Preliminary tests did not reveal any significant differences.

A new operator, named the adaptive template operator, was conceived by the author. First a template is generated by using the XOR binary operator on the best two (or more) chromosomes. The crossover is now carried out by random exchange of corresponding bits between two parents wherever the corresponding template bit is 1 and exchanging whole blocks of strings wherever a contiguous string of 0s appears in the template. The rationale is that the better chromosomes, especially in the later stage of evolution, show similarities due to accumulation of good schema or due to domination of a particular individual. And hence this process may promote the exchange of better schema while suppressing their disruption. A more rigid operator would be to retain all the bits corresponding to 0s and exchange the rest at random. This would also help maintain schema separated by other alleles, which would otherwise be disrupted. A more logical choice in this context would be to treat the chromosome as a circular string [13], since the string ends are arbitrary positions fixed by the chosen coding scheme.

None of these changes have been thoroughly tested. Since proving GA changes is an arduous task, extensive testing across a variety of problems is required to validate or reject the suggestions.

GAs in Optimal Control

The optimal trajectory problems presented in Chapter 2 present an opportunity to apply GAs to develop a general algorithm for generating initial trajectories for the

indirect method. The objective is to find a solution close enough to the optimal solution with enough digits of accuracy (normally two or three) to enable more efficient codes like SQP or one of the NEM codes to converge to the optimal solution. This objective has been successful with GAs. The literature surveyed so far fails to reveal a (general) method for obtaining the indirect solution.

Finding a feasible trajectory for optimal control problems in general, let alone the one under consideration, is reported [3] to be very difficult due to the sensitive nature of the costate equations. GAs also have the ability to find more than one optimal solutions or *niches*. This leads to the three chief uses of GAs for such problems:

- To find solutions to the given problem with practically no coding except for objective function evaluation. This can save enormous investigative and development time for sensitive problems.

- To add specific enhancements and code hybrid GAs to give robustness and superior local convergence.

- To aid in better problem understanding. Analytic tools have long been the major source for problem understanding, using simplified problems. Hence, there is limited meaning to be found (except by long experience) on quantities like costates. Genetic codes by virtue of finding several solutions and allowing ad hoc problem modifications can enhance this process. A demonstration is given in Chapter 5.

In solving the optimal trajectory problem, the unknown initial states, costates and time intervals are taken as the design variables. Each variable is represented by a string of a specified number of bits. It was found that using slightly more bits than

required for the anticipated accuracy gave better performance. Given a length of l_i bits for the ith variable, its domain is discretized to 2^{l_i} uniformly spaced real numbers. Goldberg's [13] simple genetic algorithm is not suitable for real world applications because of its low efficiency. However, if we incorporate the enhancements described above, the GA starts rivalling conventional algorithms in efficiency. However, unlike other methods, there was no problem the GA did not converge for. Some parametric adjustments were required to get more performance. Constraints are handled by the L1 exact penalty function (Equation (3.6)) described by Fletcher [8]. The unconstrained minima of this function have been proved to be the constrained minima of the problem. These functions exhibit slope discontinuities and therefore cannot be minimized using gradient based techniques. The constraints were sufficiently weighted by trial and error, in order to ensure boundedness of the function. Bounds on the domain of the design vector are implicitly handled by the decoding scheme used. Numerical integration is carried out with much lower accuracies as compared to the requirements of nonlinear programming methods, since gradient information is no longer required.

Incomplete simulation for efficiency

Genetic search permits function discontinuities and hence allows incomplete or coarse trajectory simulation using the physical know how of the problem. Hence the following modifications gave significant performance increase with no change in convergence:

- Terminate trajectory after the first engine on phase if the radius is less than the initial radius or if the radius is less than five times planet radius. The former

truncates trajectories spiraling down and the latter conservatively weeds out non-escape conditions.

- Terminate trajectory after the coast phase if the radius is more than half the 'interplanetary' distance d_1.

- For the escape problem, terminate the trajectory if the control angle has exceeded 0.3 radians before $(t_f - t_o)/10$ has elapsed, since the optimal escape is known to be very close to zero for almost the entire trajectory.

These modifications, by no means extensive, were deemed safe in terms of not restricting the flexibility of the genetic code.

Testing the GA

Validation of the GA coded was done using a few representative functions as shown in Table 4. The third column gives the range and the number of bits used for each variable. The functions include nonconvex, discontinuous, large search space and bad scaling properties to demonstrate some of the robustness and efficiency properties of the GA. Figures 4.1-4.6 show the performance of the various functions versus the number of function evaluations. All the performance curves were generated using the seed random number 0.1678943251 and are reproducible. Two performance indices are used. One is the average of the log of the difference between the optimal solution and the best solution at that point and the other is the average of the best solution in the population. The performance was averaged over fifty runs. The best possible value using the given discretization is also shown in the figures as a solid horizontal line. All except Goldberg's test function were solved using a population size of 100

Name	-Function	x_i range
DeJong's 1st function	$\sum_{1}^{3} x_i^2$	[-5.12,5.12] 10
DeJong's 2nd function	$100\left(x_1^2 - x_2\right)^2 + (1 - x_1)^2$	[-2.048,2.048] 12
DeJong's 3rd function	$\sum_{1}^{5} integer(x_i)$	[-5.12,5.12] 10
DeJong's 4th function	$\sum_{1}^{30} i x_i^4$	[-1.28,1.28] 8
Modified binary, F6	$\frac{\cos^2(\sqrt{x^2+y^2})}{1.0 + 0.001\,(x^2+y^2)^2}$	[-100,100] 22
Goldberg's example	x^{10}	[0.0,1.0] 30

and a steady state population size of 95. The first generation is produced by random initialization. Figure 4.1 shows the linear convergence rate on the logarithmic scale. At the end of 1000 evaluations an approximate log index of 3 implies that on average, the solution rapidly converges to within 0.001 of the optimal solution. Figure 4.2 gives the log performance for a badly scaled nonconvex function. The convergence is hence slightly slower (2.5 digits) since variable range and search spaces are much larger. Figure 4.3 shows the average performance for a five dimensional step function. Note that there is no local information available since the function is constant except at the discontinuities. Average performance was chosen since the function can only assume integer values. Observing the solutions showed that 24 and 25 were the only solutions produced, the latter (optimal) solution appearing more frequently. Figure 4.4 demonstrates the property of GAs in being able to efficiently search large

design spaces. The average solution produced is approximately -0.6 as compared to the optimal of 0.0. Figure 4.5 gives the log performance of the modified 'Binary F6' [14] function. This function is a two dimensional trigonometric function with a large number of local minima and maxima near the optimal solution (0.0,0.0). This function has been extensively tested by Davis [14]. The best performance given in these tests using binary representation was 3.5 digits versus 5 digits of accuracy given by this algorithm. This comparison indicates the efficiency of the algorithm being used. However real number encoding is noted to give higher convergence of up to 6.5 digits. Figure 4.6 demonstrates the high accuracy to which GAs can converge to. The design variable is very finely discretized and the function lies close to 0 in most of its domain and hence supplies very little information. It was seen that for 4000 evaluations, the GA always converged to the exact solution.

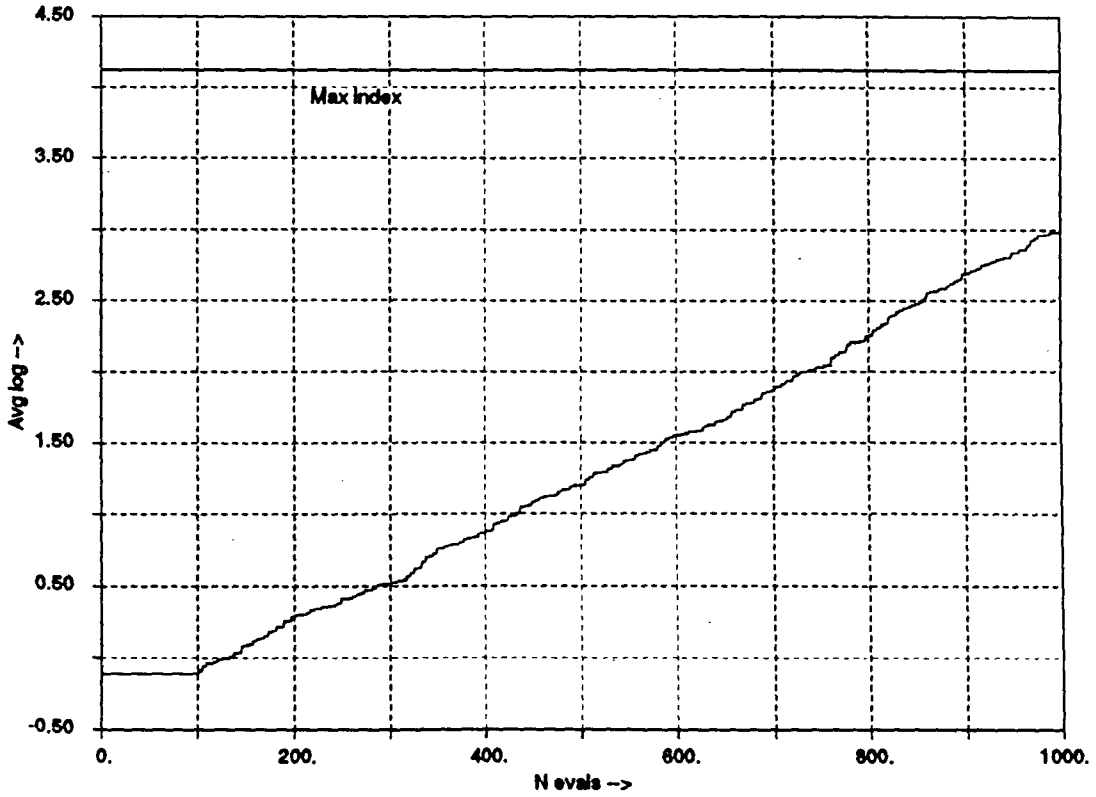


Figure 4.1: Average log performance of DeJong's first function

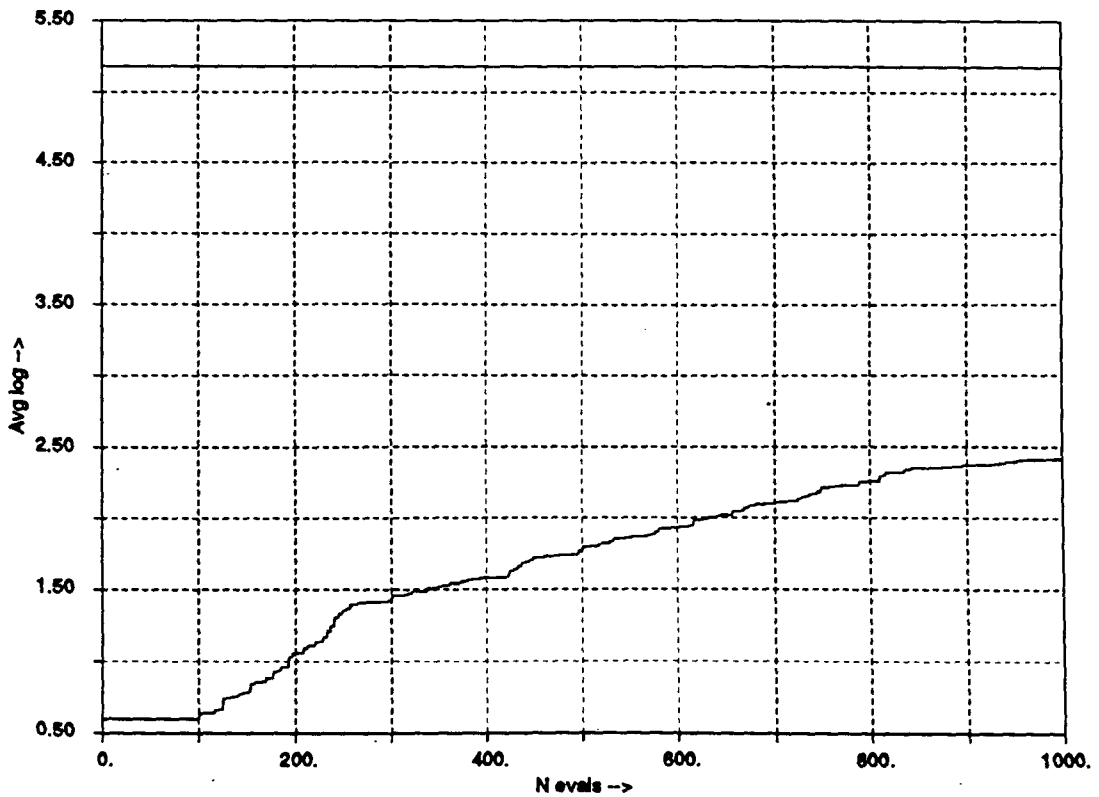


Figure 4.2: Average performance of DeJong's second function

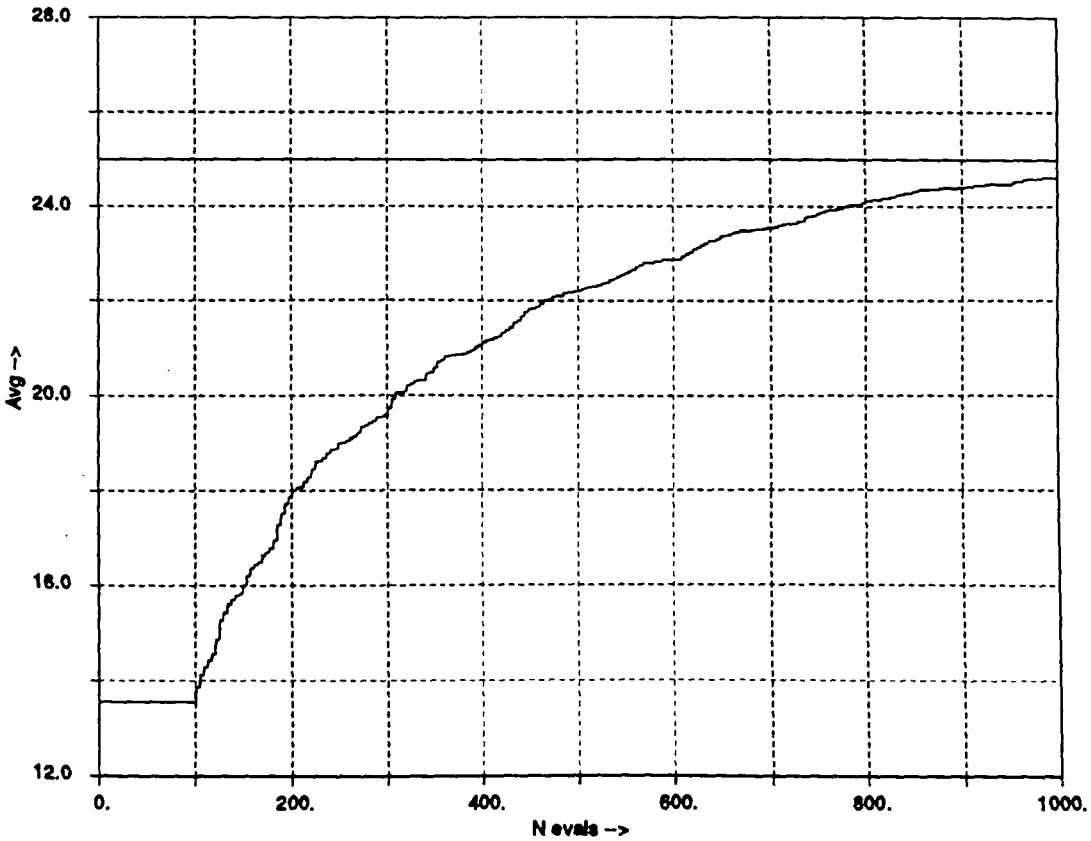


Figure 4.3: Average performance of DeJong's third function

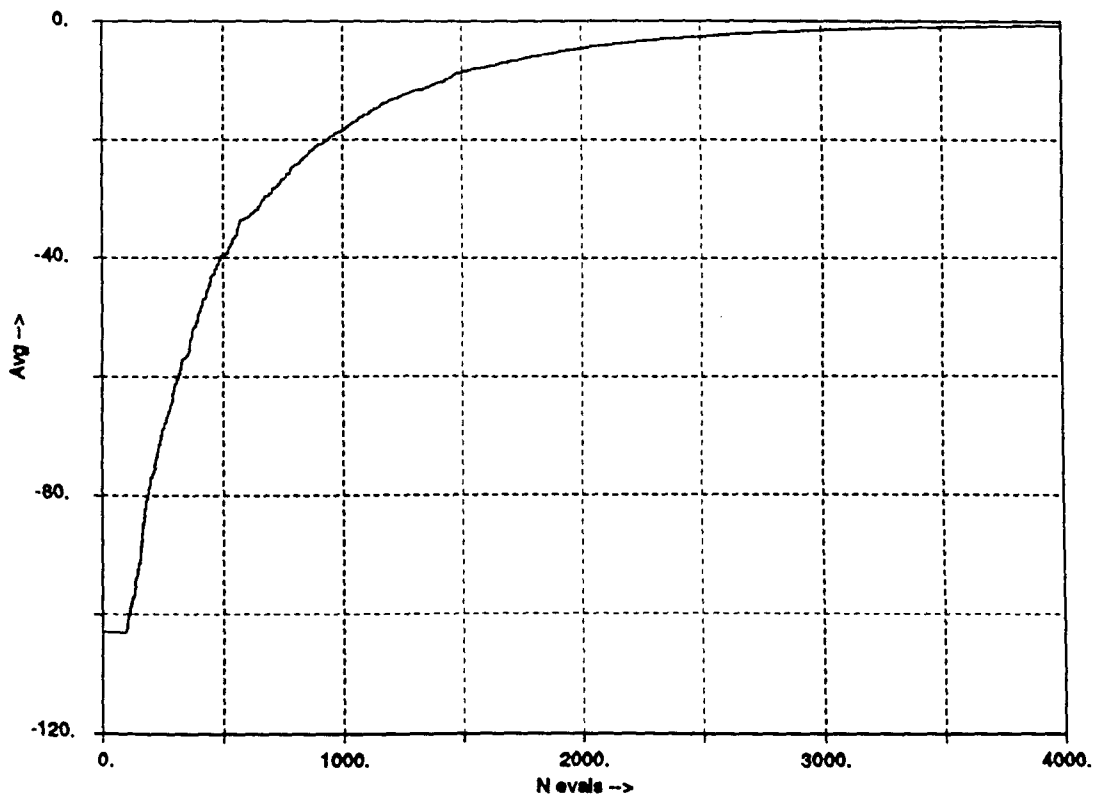


Figure 4.4: Average performance of DeJong's fourth function

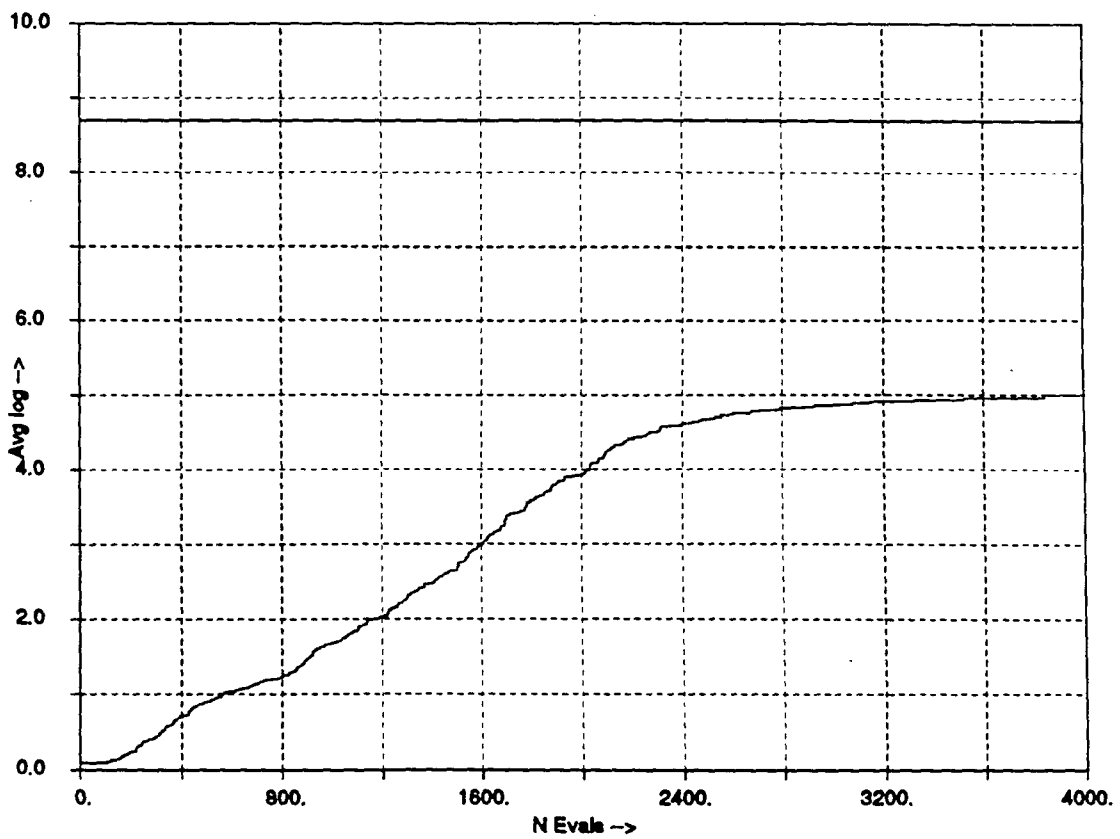


Figure 4.5: Average log performance of the modified F6 function

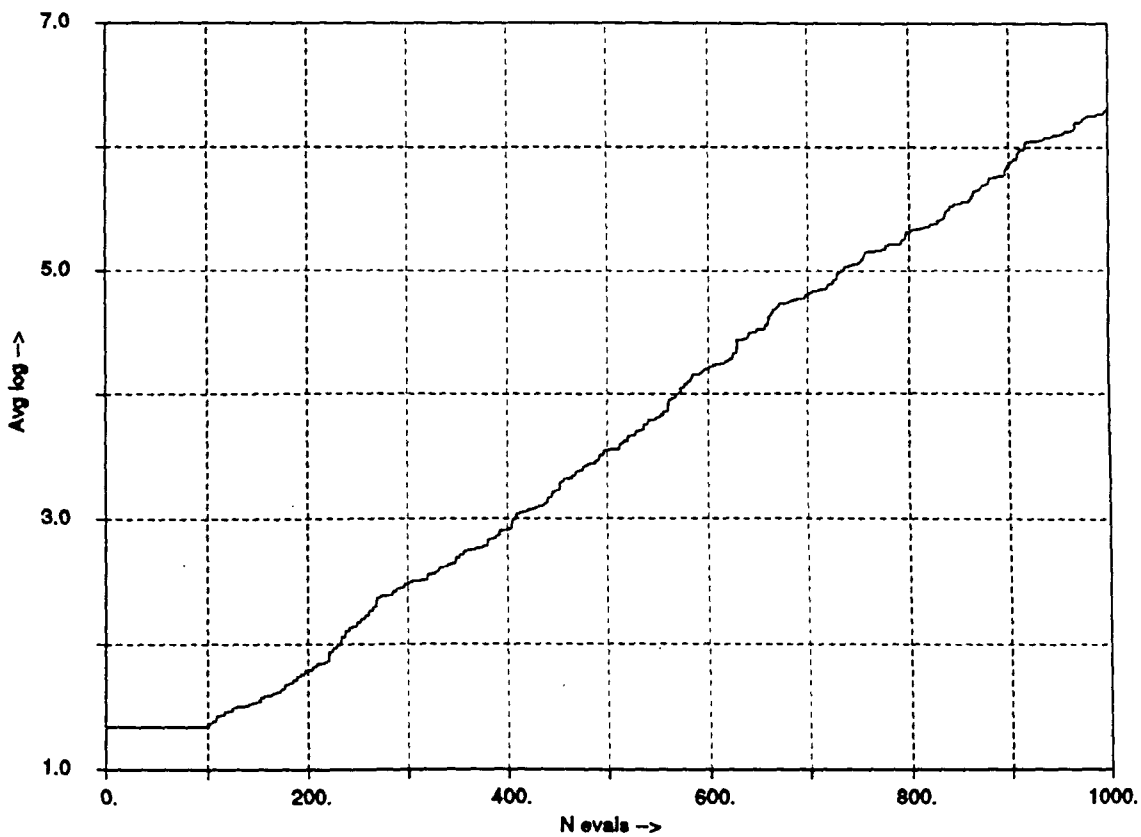


Figure 4.6: Average log performance of Goldberg's test function

CHAPTER 5. COMPARISON OF MODELS AND METHODS

This chapter gives numerical comparisons for the various models and algorithms discussed. The index of comparison was chosen to be the number of function evaluations. Here, 'function' refers to an evaluation of the right-hand sides of the state equations or the combined state-costate equations for the indirect models. This gives a good estimate of efficiency since most of the computation is in integrating the equations; this means a fixed overhead per function evaluation. However, the indirect problem when compared to the direct one requires integration of twice as many equations. This approximately translates to double the cost (of a direct method) for an indirect solution with an equal number of evaluations. Table 5.1 gives a comparison of CPU times and this index for the direct Mars transfer problem with various algorithms and number of spline points. Table 5.2 validates our choice of polar coordinates for numerical integration using the multivalue method described in Chapter 3.

Effect of Models

In comparing the efficacy of the mathematical models for the problems described in Chapter 2, a single algorithm will be used; usually the algorithm giving the best results for the particular problem is chosen.

Table 5.1: Comparison of CPU times and function evaluations

	PFM, 5	PFM, 6	PFM, 10	SQP, 10
CPU time (s), DEC 5000	42.92	46.485	69.37	78.812
Number of Evaluations	553088	599680	897840	1112630
Ratio $\times$ 1000 (s/eval)	0.077	0.077	0.077	0.070

Table 5.2: Comparison of Polar and Cartesian coordinates for integration

	Low Accuracy (10^{-4})	High Accuracy(10^{-9})
Polar	1484	5670
Cartesian	2255	6167

Continuous thrust Mars transfer

The Mars transfer problem can be solved using any of the models (2.11-2.13). However the original problem (2.11) cannot be solved using Colsys [9] since t_f is not fixed. Table 5.3 compares the other models. The three best strategies of tolerance scheduling are listed alongwith. The performance values of the overall better strategy (1), are listed. Performances with other strategies used are listed if better. The variant (2.13) with strategy (1) shows faster convergence against the best performance of variant (2.12) and shows that the former is superior. All Colsys solutions use two collocation points per interval and two initial intervals. The best overall formula for choosing initial intervals for subsequent subproblems was the lower of 16 and half the intervals required for the previous subproblem to converge. Table 5.4 compares direct and indirect formulations (2.11-2.13), using the multiplier penalty function (3.5). The initial guess provided was $t_f = 2.0$ and linearly interpolated control points between 1.0 and 6.0 radians. Note that the exact solution given by

Table 5.3: Comparison of indirect models for Mars transfer

Tolerance	10^{-3}	10^{-5}	10^{-6}	10^{-7}
Equation (2.12)	7072 2:6800	38704 3:16080	51060 3:19296	51060 2:49612
Equation (2.13)	5488 2:2520	11098 3:7660	19498 3:19168	26298

Tolerances	Initial	Increment
(1)	0.01	0.1
(2)	0.1	0.1
(3)	0.1	0.01

Colsys is 3.319309, which corresponds to 193.05 days. Choosing a lower end time than anticipated gives remarkable increases in performance due to reduction in integration per simulation. Integration accuracy of 10^{-6} (8 digits) was used. With a tolerance of 10^{-2}, it is possible to get performance of the order of 14000. This is however not useful since Colsys with the generalized Newton's iteration provides far higher accuracy with lower computation. Table 5.4 demonstrates the superior convergence and accuracy of using spline interpolation. But even using a 10-point spline does not give the accuracy of an indirect solution and takes far more computation. Hence we can conclude that direct solutions need only be used when indirect ones are not present. Since spline interpolation is continuous only up to its second derivative, the multivalue integrator halts due to use of higher derivatives. To overcome this problem, the step size was arbitrarily reduced by half each time the step size estimator using higher order derivatives failed more than once to bring error within tolerance, at the same time point.

Table 5.4: Comparison of direct and indirect formulations of the Mars transfer problem

	Tolerance: 10^{-3}	
	Perf.	t_f
IN : Equation (2.13)	153066	3.28
D : Polynomial , O(4)	505664	4.22
D : Taylors Series , O(4)	179868	3.55
D : Normalized Poly , O(4)	169151	3.49
D : Normalized Poly , O(5)	125619	3.46
D : Spline , 5 point	49647	3.38
D : Spline , 6 point	61370	3.33
D : Spline , 7 point	52733	3.33
D : Spline , 10 point	144139	3.32

IN: Indirect model; D: Direct Model

Maximum energy Earth escape

The four variations of the indirect model are compared in Table 5.5 using BFGS or the multiplier penalty method. The variation used is indicated along side. The state equations and initial conditions are unchanged. All the solutions used a 10^{-4} integration error and had a tolerance of 10^{-3} on gradient and constraints. The third set of initial costates, λ_{Colsys}, refers to the suboptimal solution obtained from Colsys as described in Chapter 3. As the table shows, the performance of Colsys is very high. For the optimal solution, the minimizing index (2.14) using costate Equation (2.19) evidently gives the best results *and* the fastest convergence. The modified costate has hence reduced sensitivity to initial values, increased the convergence rate, and gives the same solutions. However, the lower final energy obtained by satisfying constraints (2.17) is not expected since by definition of the necessary conditions, the solution is optimal. Note that even using tight tolerances and high precision

integration, the best solution by satisfying constraints (2.17) is 2.33×10^{-3}.

Direct solutions obtained for increasing number of spline points is given by Table 5.6. This shows the suboptimal nature of such solutions, even for a twenty point spline. Though there is marginal improvement with number of spline points, the increase in computational cost is not acceptable.

Table 5.5: Comparison of indirect formulations of Maximum Energy Escape

	$\lambda_o = (1.00 \times 3)$		$\lambda_o = (0.01 \times 3)$		$\lambda_o = \lambda_{Colsys}$	
	Perf.	$E \cdot 10^3$	Perf.	$E \cdot 10^3$	Perf.	$E \cdot 10^3$
Co : 2.10, C : 2.17	N/C		N/C		2:90285	2.34
	N/C		N/C		3:78754	2.33
Co : 2.10, M : 2.14	N/C		2:551668	-3.50	2:57189	2.96
	N/C		3:383788	-1.83	3:57151	2.96
Co : 2.19, C : 2.17	N/C		2:79020	2.72	2: 97176	1.64
	N/C		3:79088	2.72	3:115001	1.64
Co : 2.19, M : 2.14	2:63282	2.96	2:46209	2.96	2:25546	2.96
	3:54663	2.96	3:59409	2.96	3:24535	2.96

Co: Costate, C: Constraints, M: Minimization index
2: Dog Step line search, 3: Hookstep line search [8]
$\lambda_{Colsys} = (0.55714, -0.0046112, 0.61079)$, Performance=4560
$E_{Colsys} = 2.126 \times 10^{-3}$, $E = 6.75 \times 10^{-4}$
Energy in $\frac{\mu_e}{R_e}$ (per unit mass) as described in Chapter 2.

Table 5.6: Direct solutions using BFGS and free cubic splines

Degree	5	6	7	10	20
Energy$\cdot 10^3$	2.647	2.637	2.642	2.718	2.811
Performance	2:22907	2:29938	2:79427	2:54344	2:210034
	3:31315	3:49824	3:39326	3:53050	3:189636

Single coast Earth-Moon transfer

This problem could only be solved using the genetic code. Using the genetic algorithm's output for the initial guess and the *modified three body model* (Chapter 2), the multiplier method gave a small improvement. The solution did not improve after the first subproblem indicating that the multipliers are not helpful in this case. This solution used $3,34,519$ function evaluations and satisfied the constraints to an accuracy of $8 \cdot 10^{-2}$. Using other algorithms or the original set of constraints gave no improvement.

Algorithm Comparison and Maximum Accuracies

Table 5.7 lists the maximum accuracies successfully obtained for the Mars transfer problem using various algorithms and models. SQP gives the best accuracy with the indirect method. However, the best accuracies for the direct method are obtained from the multiplier penalty method. The multiplier method uses a constant but small increment in σ. The degeneration of SQP with increase in spline points can be attributed to the increase in the number of free variables in the constrained hyperspace. This causes SQP to do an unconstrained local search at each step using more variables. The extreme nonlinearity of the problem prevents such a search from being effective. Table 5.8 lists the best possible constraint satisfaction for the escape problem. As described above, maximum accuracy for the Earth-Moon problem is $2 \cdot 10^{-2}$.These restrictions on maximum accuracy are due to the limited gradient accuracy (which is half the possible integration accuracy) and possibly due to ill conditioning of the Hessian estimate. The question of maximum accuracy is not relevant to genetic algorithms since given sufficiently accurate integrals, they will eventually

evolve to the desired accuracy. Instead we look at the performance graphs of the GA in Chapter 6. We now list the best performance and the algorithm(s) used for all the three problems:

- The Mars transfer problem can be solved to five digit accuracy using Colsys in 7800 function evaluations.
- The Earth escape problem can be solved to four digit accuracy using Colsys and then the multiplier method, and the modified model in 19000 function evaluations.
- The Earth-Moon transfer problem can be solved to three digit accuracy using GAs and the multiplier method in about a million evaluations. Note that the constraint accuracy is in terms of the quantities normalized with respect to Earth radius and μ_e (Chapter 2).

Table 5.7: Maximum accuracies using different models and methods

	Colsys	PFM	Multiplier	SQP
Eq 2.13	10^{-8}	$2 \cdot 10^{-3}$	$4 \cdot 10^{-6}$	$3 \cdot 10^{-11}$
	3.319308	3.304817	3.319298	3.319308
Spline, 5*point*	N/A	$2 \cdot 10^{-6}$	$5 \cdot 10^{-9}$	$1 \cdot 10^{-8}$
		3.379104	3.379114	3.379114
Spline, 6*point*	N/A	$1 \cdot 10^{-7}$	$6 \cdot 10^{-9}$	10^{-3}
		3.29548	3.329549	4.5
Spline, 10*point*	N/A	$1 \cdot 10^{-7}$	$2 \cdot 10^{-7}$	10^{-3}
		3.320988	3.320988	3.83

Multiplier: Multiplier PFM.

Table 5.8: Best constraint tolerances using different models and methods

	Multiplier	SQP
Co : 2.10, Con : 2.17	$1 \cdot 10^{-2}$	N/C
Co : 2.19, Con : 2.17	$3 \cdot 10^{-3}$	N/C

GAs and Low Accuracy Solutions

It was observed that most of the computation time required by the nonlinear optimization techniques is lost in reaching a convex region containing the optimal solution. GAs were found to be an efficient and robust tool for such low accuracy solutions. Hence we briefly mention the relevant GA solutions obtained for the three problems. The Mars transfer problem was solved to two digit accuracy in about 5000 evaluations. Since this problem is well behaved, GAs are not relevant for this case. The Earth escape energy was optimized to 2.26×10^{-3} in 7588 evaluations. These figures are averaged over 50 runs with different initial random numbers. As discussed in the section above, the GA solution to the Earth-Moon problem is our only option. We now describe how the global property of GA solutions helped in developing a transformation for the initial costate representation which resulted in faster convergence.

Model enhancement through GAs solutions

For computation, the initial costate vector was conventionally represented as $\lambda_i = X(i)$. $\mathbf{X}$ is the variable set used in optimization. Solving the maximum energy Earth escape problem using the genetic algorithm gave different solutions from different runs. These solutions had the property of having similar energy values with

significantly different initial costates. Comparing the solutions however revealed a remarkable property of the optimal initial costates.

Though the first costate assumed a range of values, the corresponding third costate was always near this value. This implied a coupling between the first and third costates. Though desirable, this coupling is broken by the crossover process, slowing convergence. Hence a transformation of the variable set to the effect

$$\lambda_r = x(1); \quad \lambda_v = x(1) + x(3) \tag{5.1}$$

decoupled the variables. The genetic algorithm showed remarkably increased performance after this change. This can be attributed to the fact that now $x(3)$ has a lesser domain and is decoupled from $x(1)$. For the escape problem, GAs can now be directly used to give a solution of requested accuracy. For instance we need 75880 function evaluations to arrive at an average final energy of $2.934 \cdot 10^{-3}$. The low accuracy solution mentioned above used this transformation.

CHAPTER 6. PRESENTATION OF SOLUTIONS

This Chapter presents the optimal trajectories obtained. Figure 6.1 shows the optimal trajectory for the Earth-Mars transfer problem in polar coordinates. Figure 6.2 shows the optimal control time history. The optimal initial costate is given by $\lambda_{mars} = (1.87706, 0.928998, 2.02450)$ and a final time of 3.319308 (193.05 days). The control is observed to be accelerating the spacecraft in the first half of the trajectory and decelerating in the next half.

Figure 6.3 gives the optimal Earth escape trajectory. Figure 6.4 gives the optimal control time history and Figure 6.5 compares the indirect and the direct control solution histories using 20 spline points. As we can observe, the direct solution is trying to emulate the indirect, optimal solution but remains unsuccessful due to its restricted nature. However, towards the end when the oscillations reduce, the direct solution is very close to the indirect one. This also accounts for the energies being similar, since initially the control is near zero and has little effect on the final energy.

Figure 6.6 gives the average performance of genetic algorithm using the original Earth-Moon model. Figure 6.7 gives the performance for the modified model. The modified model shows improved convergence. The optimal trajectory for the initial thrust-to-weight ratio of 10^{-3} is given in Figure 6.8. It was observed that the multiplier method gave marginal improvement over the GA solution. The control history

in Figure 6.9 indicates that the Earth escape phase of the control is similar to the solution of the maximum energy escape problem. The optimal trajectory takes 2.252 days of initial thrusting, 0.479 days of final thrusting and 4.795 days of coasting. The optimal angles of departure ($\theta(t = 0)$) and arrival ($\theta_1(t = 7.526)$) are 6.128 and 5.569 radians, respectively. The Earth and Moon escape cosatates are, respectively, given by $\lambda_E = (0.9953, 0.03895, 1.1186, 0.0)$ and $\lambda_M = (0.6804, 0.05938, 0.8760, 0.0)$.

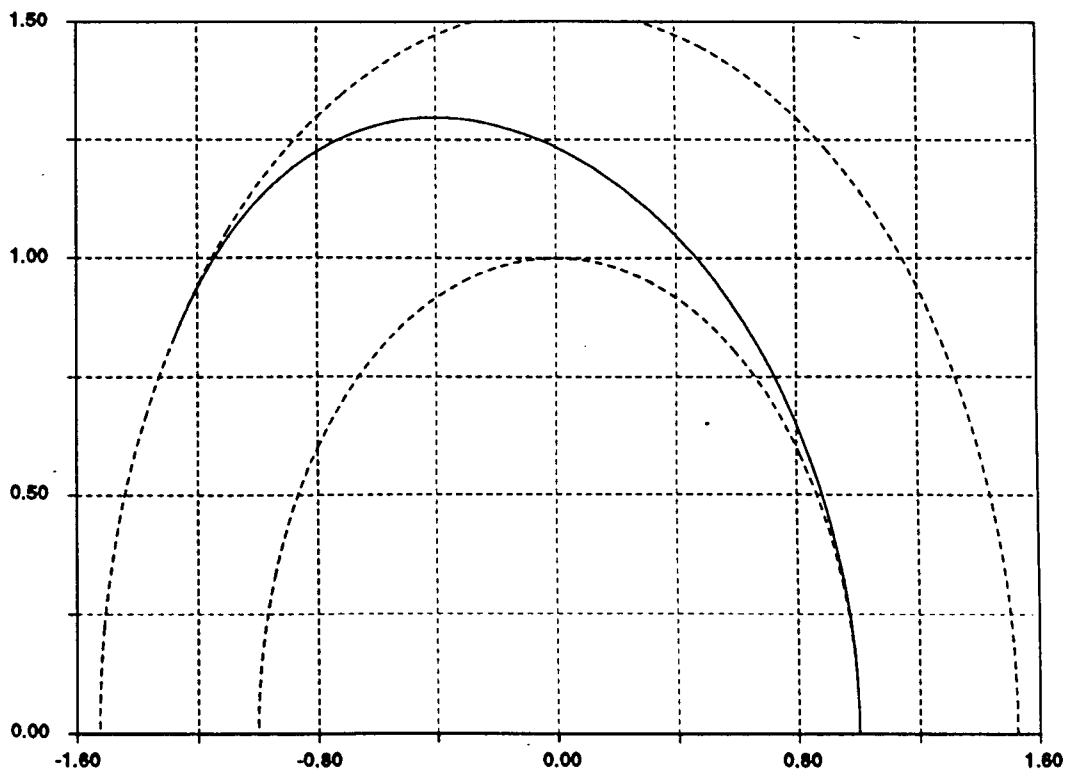


Figure 6.1: Optimal Mars transfer trajectory

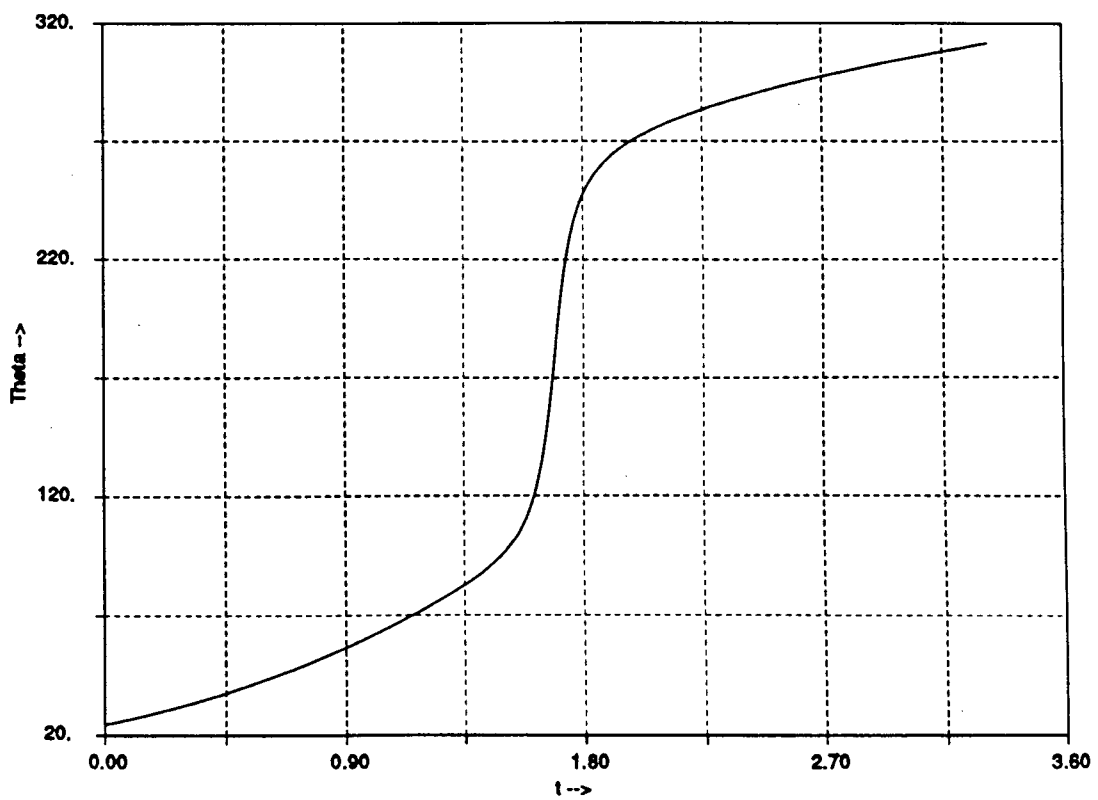


Figure 6.2: Control angle vs time for the optimal Mars transfer problem

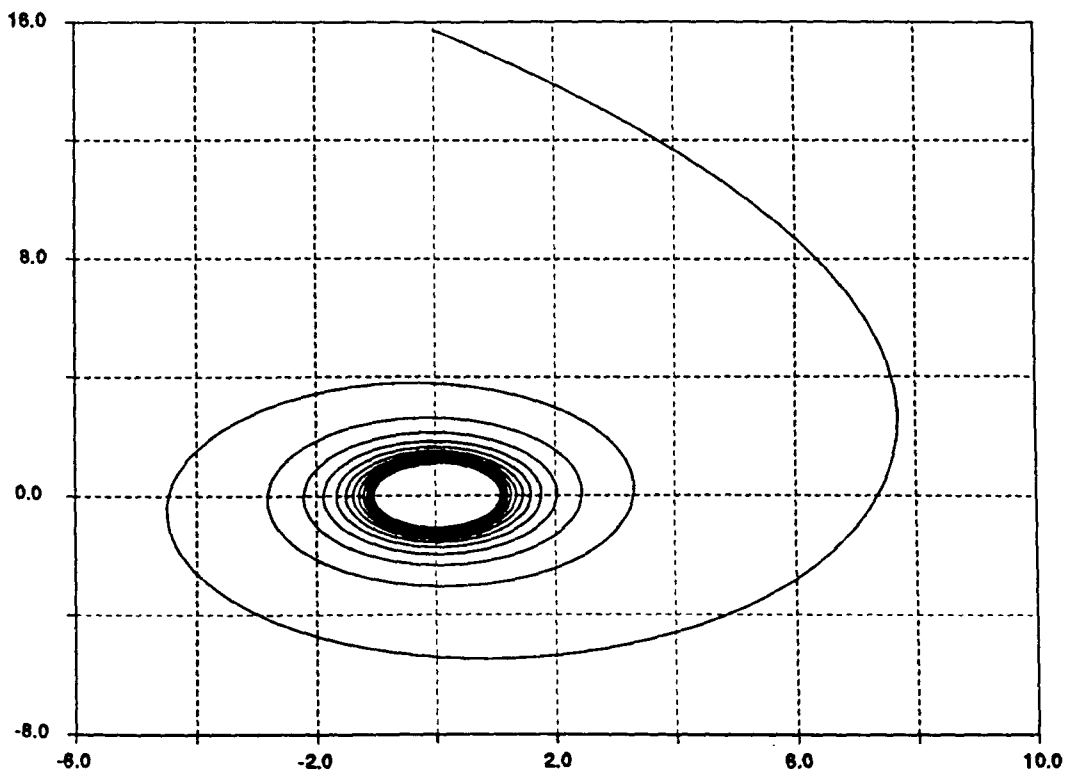


Figure 6.3: Optimal Earth escape trajectory

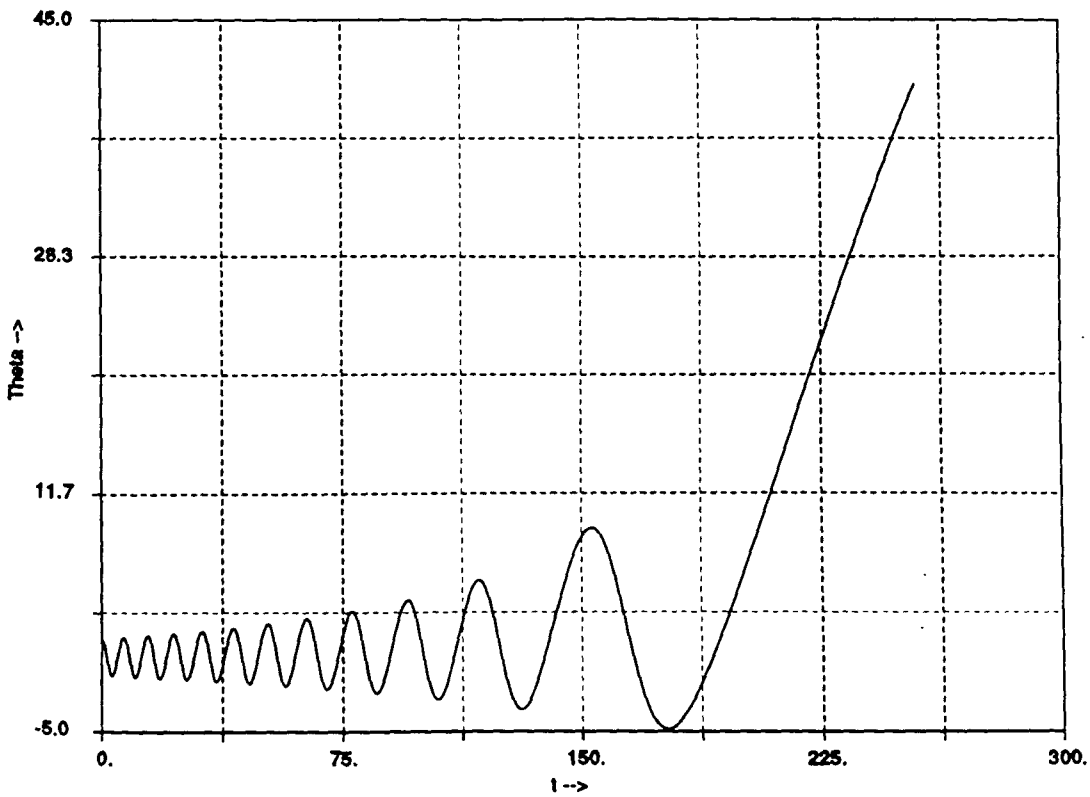


Figure 6.4: Control angle vs time for the optimal escape problem

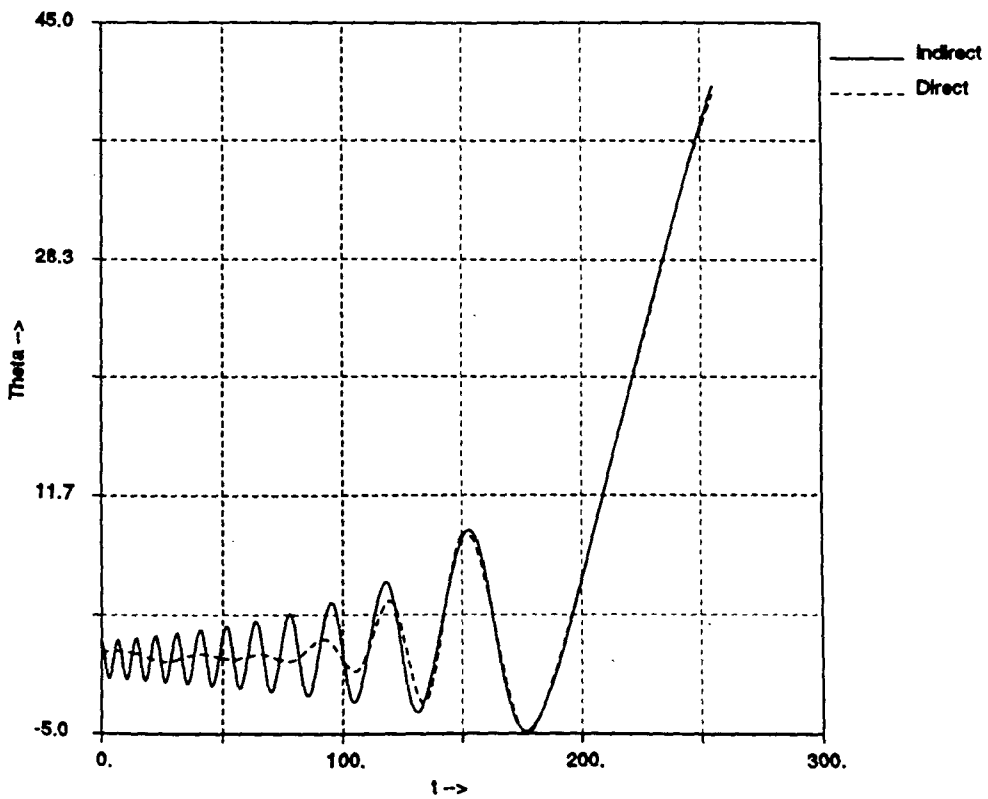


Figure 6.5: Comparison of control histories for direct and indirect methods

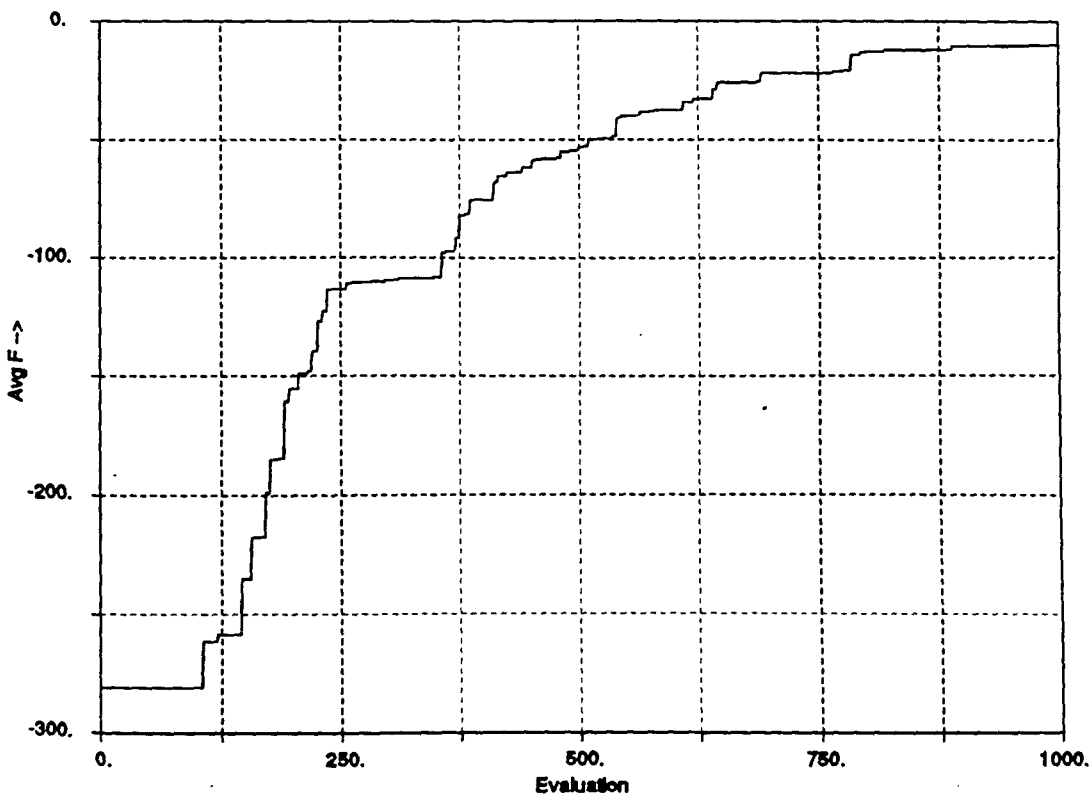


Figure 6.6: Average performance of the Earth-Moon problem

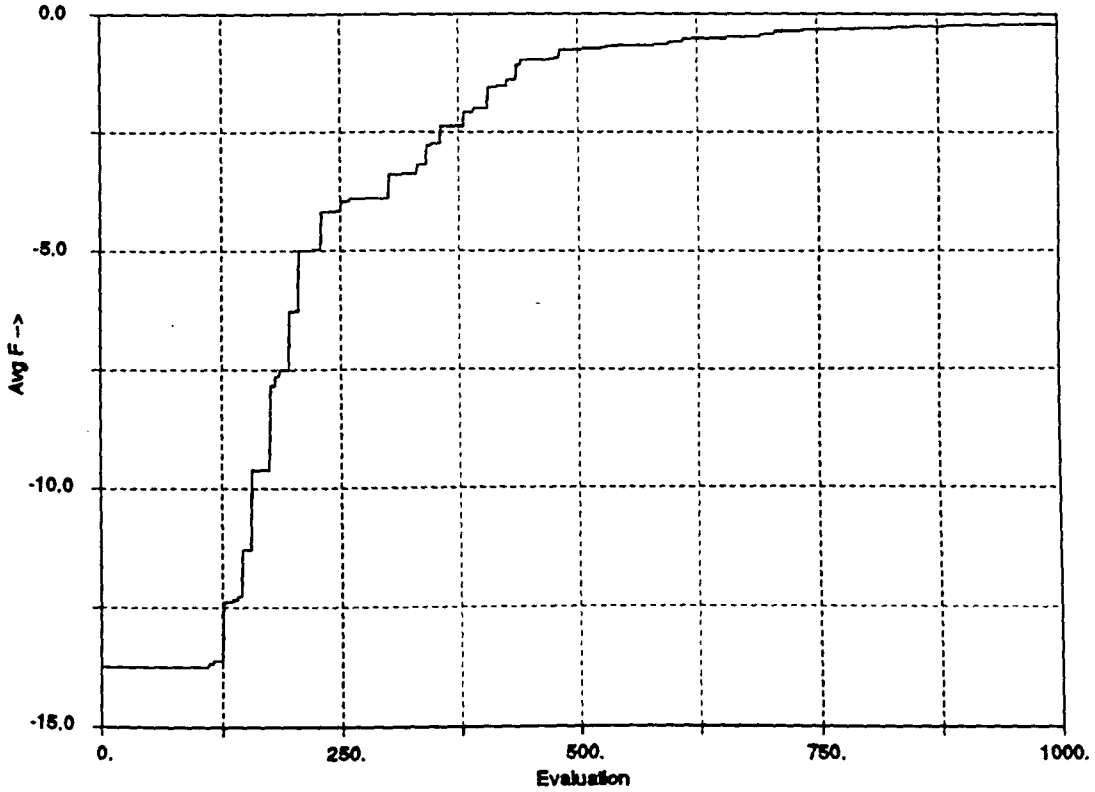


Figure 6.7: Average performance of the modified Moon model

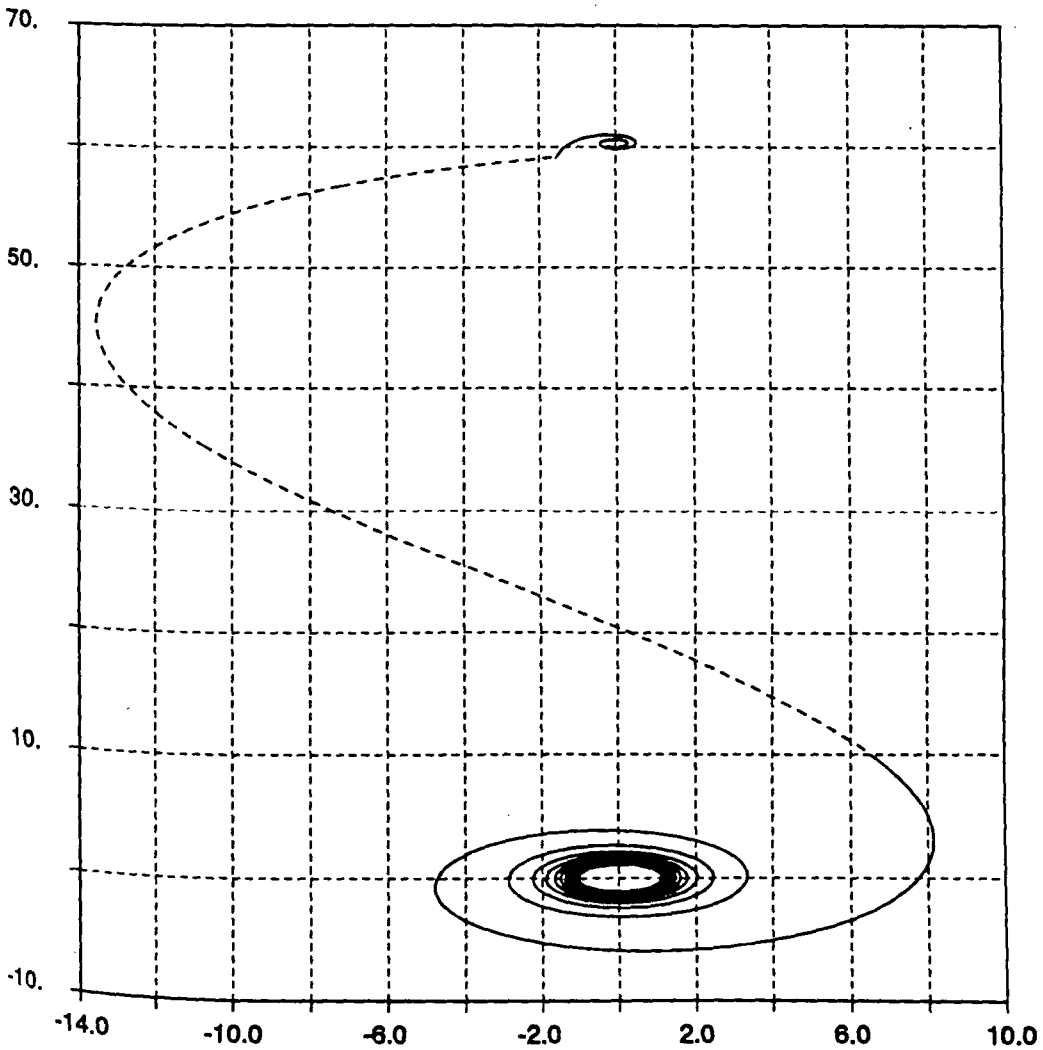


Figure 6.8: Optimal Earth-Moon transfer trajectory

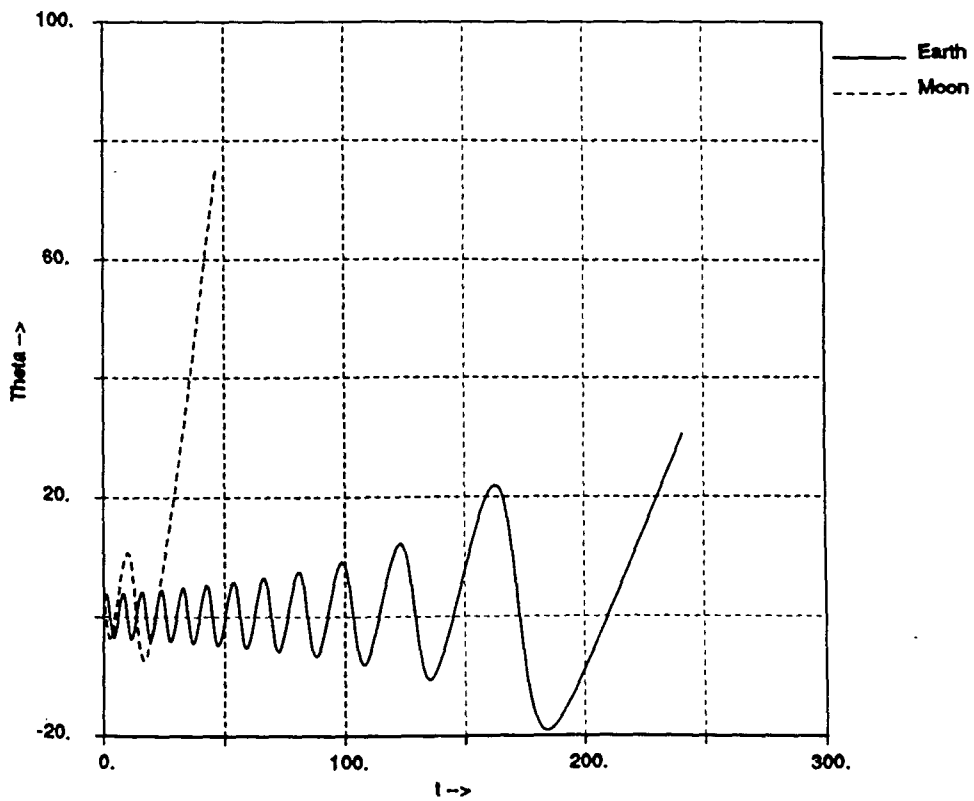


Figure 6.9: Optimal Earth-Moon control histories

CHAPTER 7. CONCLUSIONS AND SUGGESTIONS

In concluding the study; the objective of a constructing a general code for computing optimal trajectory estimates has been successful. However an equally if not more arduous task remains before the objective of a general code for optimal trajectory planning is realized. This would involve modifying the state and costate equations for three dimensional equations, using a standardized ephemeris and further hybridization along with better nonlinear optimization codes. We now summarize the relevant conclusions and observations of this investigation along with suggestions in light of the above objective.

Conclusions

- Genetic algorithms proved to be the only successful general solver of the optimal trajectory problems considered here.
- Conventional algorithms could not improve upon the solution of GA using the original model. However using the modified constraint set, we were able to obtain convergence of the conventional algorithms starting with the GAs estimate. As problems get more complex, dependence on hybrid GAs and modified models for the optimal solution is likely to increase.

- Though GAs can be used to completely solve the problem, conventional algorithms give faster convergence near the solution. But genetic search is inherently parallel since the new generation of individuals only depends on its predecessors, completely independent of the other individuals in the new population. Hence, on any parallel machine with sufficient processors, genetic algorithms will outperform conventional algorithms.

- Making ad hoc changes in the simulation (integration) module, does not effect convergence of the GA but does give better performance due to reduction in integration cost.

- Making modeling changes to tightly couple the problem objective into the costate equations desensitizes the problem. An example is the optimal Earth escape problem. The three-body model was rederived to incorporate the secondary body parameters as opposed to conventionally including only the primary body's effect. This model needs to be compared with a conventional model to test its efficacy.

- Constraint modifications improve convergence and performance as demonstrated by all three problems.

- When solving a series of subproblems, tolerance scheduling gives highly increased performance.

- The multiplier penalty method is robust relative to SQP and gives faster convergence when compared to the PFM. It requires lesser magnitude penalty constants and increments. This saves a lot of computation in increased integration

accuracy, since high weights require more significant digits.

- Splines offer suitable parameterization for problems without oscillations in the optimal control.

- SQP gives very high performance for the indirect method, at high accuracies. However, a close initial estimate is required.

Observations

- For low to medium integration accuracies (up to 10^{-8}), the number of significant digits for gradient evaluation can be assumed to be two greater than the specified local accuracy. This property was observed on all the three problems, indicating the stable nature of the state equations. This enables sufficient gradient accuracy with reduced computation.

- Colsys adapts by doubling or halving of the mesh size. This can give convergence problems. In solving the optimal escape problem the mesh size was repeatedly halved with no further decrease in the error. A better approach would have been to identify intervals of maximum error and adapt the mesh.

- Saving in significant digits is also achieved by multiplying the objective function by a constant less than 1, instead of weighting the constraints.

- The SQP and Collocation codes used are not state-of-the-art. Using better routines if available should yield better results.

Suggestions

- The genetic code needs to be further refined and is likely to give even better performance. Some untested variants are proposed in Chapter 4. Extensive testing of GAs is required to improve performance. Testing the GA needs averaging over a large number of seed random numbers. The machine used (DEC 5000) proved to be unsuitable for this task due to its computational limitations. The task is made further difficult by the plethora of options which can improve or destroy convergence.

- Investigation of better SQP and BVP solving codes is required to realize their full potential. The IMSL SQP code and the finite difference BVP solver are two such candidates.

- Using a parameterization of the form $\Theta(t) = a + bt + ct^2 + d\sin(e + ft)$ is likely to improve the performance for the direct method on problems similar to the escape problem.

- A method for getting initial estimates to control angles using a control law failed. This might be achieved using neural nets. But this is now not relevant to the current problem since optimal estimates are possible.

- The cylindrical coordinates seem to be the logical choice for the three dimensional problem. This would enable the current form of control definition and minimal changes in the state and costate equations.

- Though limited study was done to arrive at the reference normalization constants, a more analytic study along with a systematic comparison of the effect

of strategies to fix or vary the origin and normalization parameters needs to be carried out.

REFERENCES

[1] Borowski, S.K., "The Rationale/Benefits of Nuclear Thermal Rocket Propulsion for NASA'a Lunar Space Transportation System," Proceedings of 27th Joint Propulsion Conference, *AIAA* Paper 91-2052, Jun 1991.

[2] Nitta, Keiji, and Ohya, Haruhiko, "Lunar base extension program and closed loop life support," Acta Astronautica, Vol 23, Humans in Earth orbit and planetary exploration missions, Pergamon press, Oct 1990, pp. 253-262.

[3] Bryson, Arthur Earl Jr., and Ho, Yu-Chi, *Applied Optimal Control: Optimization, Estimation and Control*, Hemisphere Publishing Corporation, Washington, D.C., 1975.

[4] Lewis, Frank L., *Optimal Control*, John Wiley & Sons, New York, 1986.

[5] Moyer, Gardner H. and Pinkham, Gordon, "Several trajectory optimization techniques (Part II: Application)," *Computing Methods on Optimization Problems*, edited by Balakrishnan and Neustadt, Academic Press, New York, 1964.

[6] Battin, Richard H.,*An Introduction to the Mathematics and Methods of Astrodynamics*, AIAA Education Series, New York, 1987.

[7] Kleuver, Craig Allan, "Optimal and sub-optimal, low thrust, Earth-Moon trajectories using Sequential Quadratic Programming," M.S. Thesis, Iowa State University, Ames, 1990.

[8] Fletcher, R., *Practical Methods in Optimization*, John Wiley & Sons, New York, 1987.

[9] Ascher U., Christiansen J. and Russell R.D., "Colsys - A collocation code for boundary value problems," proceedings, *Conference for codes for bvp-s in ode-s*, Houston, Texas, 1978.

[10] Gear, William C.,*Numerical initial value problems in ordinary differential equations,* Prentice Hall, New Jersey, 1972.

[11] Holland, J. H., *Adaptation in Natural and Artificial Systems*, The University of Michigan Press, Ann Arbor, 1975.

[12] De Jong, K.A., "An analysis of the behavior of a class of adaptive systems," Doctoral Dissertation, University of Michigan, Ann Arbor, 1975.

[13] Goldberg, David E., *Genetic algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, Massachusetts, 1989.

[14] Davis, Lawerence, *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York, 1991.

[15] Rao, S.S., Pan, T.-S., Venkayya, V.B., "Optimal placement of actuators in actively controlled structures using genetic algorithms," AIAA Journal, Vol 29, Jun 1991, pp. 942-943.

[16] Hajela, Prabhakar, "Genetic search- An approach to the Nonconvex Optimization problem," AIAA Journal, Vol 28, Jul 1991, pp. 1205-1210.

APPENDIX A. SOLAR AND SPACECRAFT PARAMETERS

The following numerical values were used for the problems solved in this thesis:

Sun

$$\mu = 1.0$$
$$R_{ref} = 1.0Au$$

This was the system chosen to compare results with the reference problem [5]. One time unit then implies 58.16 days.

Earth

$$R_e = 6378.14453125km$$
$$R_o = 315km + R_e$$
$$R_b = 4670.71094km$$
$$\mu_e = 3.986011875 \times 10^5 km^3/s^2$$
$$Mean\ semi-major\ axis = 1.0Au$$

Where R_e refers to the mean Earth radius and R_b refers to the position of the Barycenter of the Earth-Moon system. R_o refers to the position of the low parking orbits

from which the escape trajectory begins. The final time for the maximum energy escape problem is 2.38 days.

Moon

$$
\begin{aligned}
R_m &= 1738.0km \\
R_o &= 100km + R_m, \quad (for\ Moon\ parking\ orbit) \\
d_1 &= 384400.0km \\
\mu_1 &= \mu \cdot \frac{R_b}{d_1 - R_b}
\end{aligned}
$$

Spacecraft

$$
\begin{aligned}
m_o &= 10^5 kg \\
\dot{m} &= 0.0299 kg/s \\
Thrust &= 2942.0N
\end{aligned}
$$

For the Mars transfer problem, the spacecraft parameters in the normalized coordinates chosen above are:

$$
\begin{aligned}
m_o &= 1.0 \\
\dot{m} &= 0.07487 \\
Thrust &= 0.1405
\end{aligned}
$$

APPENDIX B. DERIVATION OF OPTIMAL ESCAPE MODELS

The state equations of the satellite, escaping orbit, are given by Equations (2.7). The time rates of the kinetic energy $\dot{K}$ and potential energy $\dot{U}$ are given by:

$$\begin{aligned} \dot{K} &= u\dot{u} + v\dot{v} \\ \dot{U} &= -\frac{\mu}{r^2}u \end{aligned} \tag{B.1}$$

Hence the time rate of change of specific energy, $\dot{e}$ is given by:

$$\dot{e} = \dot{K} - \dot{U} \tag{B.2}$$

Substituting Equations (2.7,B.1) in the above gives:

$$\dot{e} = a(t) \cdot (u \sin\Theta + v \cos\Theta) \tag{B.3}$$

Using $\dot{e}$ as the functional whose integral has to be maximized and adding the total energy as an end time functional, we get the new performance index:

$$J = \frac{u^2(t_f) + v^2(t_f)}{2} - \frac{\mu}{r(t_f)} + \int_{t_o}^{t_f} a(t) \cdot (u \sin\Theta + v \cos\Theta) dt \tag{B.4}$$

The performance index is included twice to enable more stable costates and suitable terminal constraints.Substituting (B.4) in Equations (2.7) and (2.3) gives the governing new costate equations:

$$\tan\Theta = \frac{\lambda_u + u}{\lambda_v + v}$$

$$
\begin{aligned}
\dot{\lambda}_r &= \left(\frac{v^2}{r^2} - 2\frac{\mu}{r^3}\right) \cdot \lambda_u - \frac{uv}{r^2} \cdot \lambda_v \\
\dot{\lambda}_u &= -\lambda_r + \frac{v}{r} \cdot \lambda_v - a(t) \cdot \sin\Theta \\
\dot{\lambda}_v &= -2\frac{v}{r}\lambda_u + \frac{u}{r} \cdot \lambda_v - a(t) \cdot \cos\Theta
\end{aligned} \tag{B.5}
$$

Noting that the final time is fixed and applying the boundary conditions (2.6) gives the same terminal constraints:

$$
\begin{aligned}
\lambda_r(t_f) &= \frac{\mu}{r^2(t_f)} \\
\lambda_u(t_f) &= u(t_f) \\
\lambda_r(t_f) &= v(t_f)
\end{aligned} \tag{B.6}
$$

APPENDIX C. DERIVATION OF THE THREE-BODY MODEL

The three-body system is defined by a small mass under the influence of two large point-mass bodies. We refer to one of the large bodies as the primary and the other as the secondary. The coordinate system is defined by Figure 2.2. The primary body's parameters are defined without any subscript, and the secondary body is referred to by the subscript '$_1$'. The origin is at the primary. The motion of the satellite with respect to the rotating reference frame, fixed to the primary is given by adding the gravitational forces of the two bodies in the polar frame. To obtain its accelerations with respect to an inertial frame fixed to the primary, the acceleration of the origin (primary) is subtracted to give:

$$\begin{aligned}
\dot{r} &= u \\
\dot{u} &= \frac{v^2}{r} - \left(\frac{\mu}{r^2}\hat{r} + \frac{\mu_1}{r_1^2}\hat{r_1} + \frac{\mu_1}{d_1^2}\hat{d_1}\right)\cdot\hat{r} \\
\dot{v} &= -\frac{uv}{r} - \left(\frac{\mu_1}{r_1^2}\hat{r_1} + \frac{\mu_1}{d_1^2}\hat{d_1}\right)\cdot\hat{\theta} \\
\dot{\theta} &= \frac{v}{r}
\end{aligned} \tag{C.1}$$

Here the vectors $\vec{r}$ and $\vec{d_1}$ are pointing away from the origin. Hence, the vector quantities are given by:

$$\begin{aligned}
\vec{r} &= r\hat{r} \\
\vec{r_1} &= \vec{r} - \vec{d_1}
\end{aligned}$$

$$\begin{aligned}
\vec{d_1} &= d_1 \cos\theta \hat{r} - d_1 \sin\theta \hat{\theta} \\
\vec{r_1} &= (r - d_1 \cos\theta)\,\hat{r} + d_1 \sin\theta \hat{\theta}
\end{aligned} \tag{C.2}$$

Substituting vectors (C.2) in Equations (C.1) gives:

$$\begin{aligned}
\dot{r} &= u \\
\dot{u} &= \frac{v^2}{r} - \frac{\mu}{r^2} - \frac{\mu_1}{r_1^3}(r - d_1\cos\theta) - \frac{\mu_1}{d_1^3} d_1 \cos\theta \\
\dot{v} &= -\frac{uv}{r} - \frac{\mu_1}{r_1^3} d_1 \sin\theta + \frac{\mu_1}{d_1^3} d_1 \cos\theta \\
\dot{\theta} &= \frac{v}{r}
\end{aligned} \tag{C.3}$$

The centripetal and Coriolis acceleration components due to the barycenter rotation are given by:

$$\begin{aligned}
\ddot{\vec{r}} &= \ddot{\vec{r}}_{abs} - \dot{\vec{\omega}} \times \vec{r} - \vec{\omega} \times \vec{\omega} \times \vec{r} - 2\vec{\omega} \times \dot{\vec{r}} \\
\vec{\omega} &= \omega \hat{k} \\
\dot{\vec{\omega}} &= 0 \\
\omega &= \sqrt{\frac{\mu + \mu_1}{d_1^3}}, \quad [6] \\
\dot{\vec{r}} &= u\hat{r} + v\hat{\theta} \\
\Rightarrow \ddot{\vec{r}} &= \ddot{\vec{r}}_{abs} + r\omega^2 \hat{r} - 2u\omega\hat{\theta} + 2v\omega\hat{r}
\end{aligned} \tag{C.4}$$

Substituting Equation (C.3) in Equation (C.4) and adding terms for the thrust acceleration of the engine gives:

$$\begin{aligned}
\dot{r} &= u \\
\dot{u} &= \frac{v^2}{r} - \frac{\mu}{r^2} - \frac{\mu_1}{r_1^3}(r - d_1\cos\theta) - \frac{\mu_1}{d_1^3} d_1 \cos\theta + r\omega^2 + 2v\omega + a(t)\sin\Theta \\
\dot{v} &= -\frac{uv}{r} - \frac{\mu_1}{r_1^3} d_1 \sin\theta + \frac{\mu_1}{d_1^3} d_1 \cos\theta - 2u\omega + a(t)\cos\Theta \\
\dot{\theta} &= \frac{v}{r}
\end{aligned} \tag{C.5}$$

The costate equations are obtained from Equations (2.3) and (C.5) as follows:

$$
\begin{aligned}
\dot{\lambda}_r &= -\lambda_u\left(-\frac{v^2}{r^2}+\frac{2\mu}{r^3}-\frac{\mu_1}{r_1^3}+T1\cdot\frac{\partial r_1^2}{\partial r}+\omega^2\right)-\lambda_v\left(\frac{uv}{r^2}+T2\cdot\frac{\partial r_1^2}{\partial r}\right)\\
\dot{\lambda}_u &= -\lambda_r-\lambda_v\left(-\frac{v}{r}-2\omega\right)\\
\dot{\lambda}_v &= -\lambda_u\left(\frac{2v}{r}+2\omega\right)+\lambda_v\frac{u}{r}-\frac{\lambda_\theta}{r}\\
\dot{\lambda}_\theta &= -\lambda_u\left(-\frac{\mu_1}{r_1^3}d_1\sin\theta+T1\frac{\partial r_1^2}{\partial\theta}+\frac{\mu_1}{d_1^3}d_1\sin\theta\right)\\
&\quad -\lambda_v\left(-\frac{\mu_1}{r_1^3}d_1\cos\theta+T2\frac{\partial r_1^2}{\partial\theta}+\frac{\mu_1}{d_1^3}d_1\sin\theta\right)
\end{aligned}
\tag{C.6}
$$

where

$$
\begin{aligned}
T1 &= \frac{3}{2}\cdot\frac{\mu_1}{r_1^5}(r-d_1\cos\theta)\\
T2 &= \frac{3}{2}\cdot\frac{\mu_1}{r_1^5}d_1\sin\theta\\
\frac{\partial r_1^2}{\partial r} &= 2(r-d_1\cos\theta)\\
\frac{\partial r_1^2}{\partial\theta} &= 2rd_1\sin\theta
\end{aligned}
$$

These equations will be regrouped in a compact form in Chapter 2. If we need to switch to the secondary as our new primary, the velocity and position components need to be transformed. This done by choosing the connecting line between the primary and the secondary as the reference axis and grouping all quantities with respect to this. Now an additional d_1 is added to the horizontal component of position and an additional $d_1\omega$ is added to the vertical component of the velocity. These quantities are then resolved with respect to the new θ (previously θ_1), which is found by vector transformation. The resulting set of equations becomes (2.20). Using these

equations and swapping the values of μ and μ_1 switches the origin from the primary to the secondary which then becomes the new primary.

APPENDIX D. CODE LISTING

The code is modular in nature and allows rapid modifications and experimentation. During its development the necessity for object oriented syntax was felt. The genetic algorithm is written in C rather than FORTRAN to enable the use of structured variables and due to ease of string manipulation.

Main Program

This module is responsible for initializing data, ensuring data flow between the other modules, and unifying different problems and algorithms. It consists of the main program and input modules for GA (SGAINP), SQP (SQPINP), penalty methods using BFGS (PENINP), Colsys (COLSINP) and common data (GENINP).

```
C*******************************************************************
C                                        *                          *
C   AUTHOR : LALITESH KUMAR KATRAGADA *  SEP 10, 1991               *
C*******************************************************************
C   NERVE CENTER FOR CONTROLLING DATA FLOW
C   CAN SOLVE ALL PROBLEMS WITH ANY ALGORITHM, AND INITIAL VALUES
C   POSSIBLE TO PIPE OUTPUT OF ONE PROGRAM TO INPUT OF OTHERS
C   MODULES CAN BE ADDED INDEPENDENT OF OTHERS
C
C   PROBLEMS       :  MARS TRANSFER, ESCAPE, EARTH-MOON TRANSFER
C   ALGORITHMS     :  COLSYS, PFM, MULTIPLIER, GENETIC ALGORITHM
```

```
C   PLOTTING DATA :  OPTIONAL
C*****************************************************************
      PROGRAM CONTROL
      IMPLICIT NONE

C     INITYP    : INITIALIZATION TYPE
C     PROBTYP   : PROBLEM TYPE.
C     METHOD    : METHOD TO BE USED TO SOLVE IT.
C     FNTYPE    : MATH MODEL to USE. ; BC'S TO USE.
C
C     EPSO      : INITIAL EPSILON (FOR INTEGRATION)
C     EPSI      : EPS INCREMENT (IF DYNEPS)
C     EPSMIN    : MIN EPS TO USE.
C     DYNEPS    : DYNAMIC EPS CHANGE IF .TRUE.
C
C     DYNPROB   : > 1 SUBPROBLEMS if true
C     NPROBS    : No. of SUBPROBLEMS
C
C     TOLO      : INITIAL TOLERANCE
C     TOLI      : SAME AS ABOVE (IF DYNTOL)
C     TOLMIN    : MIN TOLERANCE.
C     DYNTOL    : DYNAMIC TOLERANCING IF .TRUE.
C
C     DYNTIME   : End TIME is NOT fixed.
C
C     FTOL      : Tolerance on function
C     CTOL      : Tolerance on gradient
C     GTOL      : Gradient Tolerance
C
C     FNTYPE    : State-Costate equation set.
C     CONTYPE   : Constraint Type
C     PARTYPE   : Type of Parameterization for Direct Method.
C                 1 : Polynomial
C                 2 : Normalized Polynomial
C                 3 : Taylors Series Polynomial
C                 4 : Spline
C                 5 : Normalized Taylors Series ?
C                 6 : Modified Sin Series ?
C
C     NOTE      : SET any unused value to 0, so that inadvertent
```

```
C               usage will generate error messages.
C
      INTEGER PROBTYP, INITYP, METHOD, OUTYP, MODEL, FNTYPE,CONTYPE,
     +        PARTYPE, PINDEX,NMAX, I, CMAX, MAXOPS, N1MAX
      PARAMETER (NMAX=40,CMAX=20,N1MAX=30)
      PARAMETER (MAXOPS=10)

      DOUBLE PRECISION TOLO,TOLI,TOLMIN, EPSO,EPSI,EPSMIN, EPS,
     +     FTOL,CTOL,GTOL
      INTEGER IFCNT,IGCNT,IGDCNT,NDIF,NRIGHT,MAXINT,NINT,IPR,K,
     +        INITSOL,SENSE,FLAG,NPROBS,NOUTPTS, N1,N,MCON,
     +        IFCNT1,IGCNT1,IHCNT1, SKIP,NSKIP,
     +        ITER,ITNLIM, DIGITS, MAXITR, NIN,NOUT,
     +        POPSIZ, NELITE, MAXGEN, NEVAL,NODUP,SCALE,NRUN
      DOUBLE PRECISION MEPS,mu, delta, TF, RF, RO,Thrust,mo,mdot,
     +                 To,Uo,Vo,t2init,DX(3,2),TDELTA,
     +     Too,TFF,X1(N1MAX),SCOEF(4,N1MAX),TX(N1MAX),
     +     Mu1,D1,OMEGA,DAY,Re,Rm,
     +     X(NMAX),FPLS,GPLS(NMAX),GRADTL,STEPTL,STEPMX,CNORM,CONTL,
     +     LINETA,BL(NMAX+CMAX),BU(NMAX+CMAX+1),
     +     BOUNDS(0:3*NMAX),OPFITS(0:3*MAXOPS),PCROSS,PMUT,
     +     SCMIN,SCMAX,RANDSEED

      LOGICAL DYNEPS,DYNTOL,START,MULINTS,DONE,DYNTIME,DYNPROB,
     +        STORE, PLOT
      CHARACTER*20 FILENAME
      INTEGER*1 CHR(20)
      EQUIVALENCE (FILENAME,CHR)
      COMMON /MVCOUNT/ ifcnt,igcnt, igdcnt
      COMMON /COUNT  / IFCNT1,IGCNT1,IHCNT1
      common /EXMARS / mu, delta, TF, RF, RO,Thrust,mo,mdot
      COMMON /PLANET / Mu1,D1,OMEGA,DAY,Re,Rm
      COMMON /escape / To,Uo,Vo
      COMMON /FNSPECS/ FNTYPE,CONTYPE,PARTYPE,PINDEX,METHOD,PLOT
      COMMON /ERSPECS/ EPS,FTOL,CTOL,GTOL
      COMMON /DIRCOMN/  X1,TX,SCOEF,Too,TFF,N1
      COMMON /COUT   / SKIP,NSKIP,STORE
      COMMON /CINOUT / NIN,NOUT

         NIN    = 5
```

```
      NOUT   = 6
      ifcnt  = 0
      igcnt  = 0
      igdcnt = 0
      IFCNT1 = 0
      igcnt1 = 0
      ihcnt1 = 0
      N      = 0
      MCON   = 0

      MEPS=4.D-16
      delta = sqrt(meps)
      NSKIP=4

      EPSI=0.1
      EPSO=1.D-2
      EPSMIN=1.D-5
      EPS=EPSO

      START=.TRUE.

      DYNEPS=.FALSE.
      DYNPROB=.FALSE.
      DYNTIME=.FALSE.
      NOUTPTS=301

      CALL GENINP (NIN,NOUT,N,N1,MCON,PROBTYP,FNTYPE,PINDEX,
     +             METHOD, CONTYPE,INITYP,PARTYPE)


      GOTO (1,2,3) INITYP
      WRITE(NOUT,*) 'INVALID INITIALIZATION CODE'
      STOP
1      CALL MARINIT (Mu,ro,to,rf,tf,t2init,Thrust,mo,mdot)
      GOTO 10
2      CALL EARINIT (Mu,ro,to,rf,tf,t2init,Thrust,mo,mdot)
       Uo=0.0
       Vo=SQRT(Mu/Ro)
      GOTO 10
3      CALL MOONINI (Mu,Mu1,D1,OMEGA,Ro,Rf,Thrust,mo,mdot
```

```
     +                    ,DAY,Re,Rm)
 10     CONTINUE


        WRITE(NOUT,*) 'GIVE EPS, NDIGITS'
        READ(NIN,*) EPS, DIGITS

        IF (METHOD.NE.4) THEN
           Write (NOUT,*) 'Give initial values : ',N
           READ (NIN, *)  (X(I), I=1,N)
         ENDIF

         PLOT=.FALSE.
        if (method.eq.6) THEN
          PLOT=.TRUE.
          Method=2
        ENDIF
        GOTO (21,22,23,24,22) METHOD
        WRITE(NOUT,*) 'INVALID METHOD CODE'
        STOP
 21     CONTINUE
           CALL COLSINP (NIN,NOUT,DYNPROB,DYNTIME,NPROBS,
     +                   NINT,K,MAXINT,EPSO,EPSI,EPSMIN)
           EPS=EPSO
           NDIF=6
           NRIGHT=NDIF/2
           IPR=0
           INITSOL=1
           SENSE=1
C     .DEFINE INITIAL AND TWO END TIME APPROXIMATIONS
           IF (DYNPROB) THEN
             TDELTA=(T2INIT-To)*1.00001/DBLE(NPROBS)
             TF=To+TDELTA
             Rf = Rf*Tf/(T2init-To)
           ENDIF
           DX(1,1)=TF
        GOTO 30
 22     CONTINUE
           ITER=0
           CALL PENINP(NIN,NOUT,GRADTL,STEPTL,STEPMX,CONTL,
```

```
     +           ITNLIM,MAXITR)
        GOTO 30
23      CONTINUE
           CALL SQPINP (NIN,NOUT,N,BL,BU,STEPTL,CONTL,LINETA)
           GOTO 30
24      CONTINUE
           WRITE(*,*) 'Give filename 10char long in quotes'
           READ(*,*) FILENAME
           CHR(11)=0
           CALL SGAINP(NIN,NOUT,N,BOUNDS,OPFITS,POPSIZ, NELITE,
     +                 MAXGEN, NEVAL,PCROSS, PMUT, RANDSEED,
     +                 NODUP, SCALE, SCMAX, SCMIN, NRUN)
30      CONTINUE



        DONE=.FALSE.
        FLAG=1
100     CONTINUE
        GOTO (11,12,13, 14,12) METHOD
11         CALL EXM1(EPS,NDIF,NRIGHT,MAXINT,NINT,K,IPR,INITSOL,
     +     SENSE,START,FLAG)
        GOTO 20
12        CONTINUE
          CNORM=10*CONTL
110       IF ((CNORM.GT.CONTL).AND.(ITER.LT.MAXITR)) THEN
            CALL PENSUB(N,MCON,X,FPLS,GPLS,DIGITS,
     +        GRADTL, STEPTL, STEPMX, CNORM,ITER,ITNLIM,START)
            WRITE(*,*) 'ITER,CNORM,FPLS,TFN',iter,cnorm,fpls,ifcnt
            write(*,*) 'gradient',(GPLS(i),i=1,n)
            write(*,*) 'X ',(X(i),i=1,n)
            IF ((ITER.EQ.MAXITR).AND.(CNORM.GT.CONTL)) THEN
              WRITE(NOUT,*) '**ERROR MAX ITERATIONS EXCEEDED'
              FLAG=-1
            ELSE
              FLAG=1
            ENDIF
            GOTO 110
          ELSE
            DONE=.TRUE.
```

```
          ENDIF
          CLOSE(1)
        GOTO 20
 13     CONTINUE
          FLAG=0
          CALL EXM3(N,X,BL,BU,MCON,DIGITS,MAXITR,
     +              LINETA,STEPTL,CONTL,FLAG)
          DONE=.TRUE.
        GOTO 20
 14     CONTINUE
          CALL FSGA(X,BOUNDS,OPFITS,POPSIZ,NELITE,MAXGEN,NEVAL,PCROS
     +     S,PMUT,RANDSEED,NODUP,SCALE,SCMAX,SCMIN,FILENAME,NRUN,MCON)
          DONE=.TRUE.
 20     CONTINUE


C       ...
        IF (FLAG.NE.1) THEN
          WRITE(NOUT,*) 'DOES NOT CONVERGE, FLAG=',FLAG
          DONE=.TRUE.
        ELSE
          IF (DYNPROB) THEN
            IF (METHOD.EQ.1) CALL  OUTCSYS(NOUTPTS,NDIF)
            CALL ADAPESC (NINT,DONE,T2INIT,TDELTA)
          ELSE
            IF (DYNTIME)
     +        CALL TADAPT (DX,EPS,NINT,DONE,T2INIT,EPSO,EPSI,EPSMIN)
            IF ((DONE).AND.(METHOD.EQ.1)) CALL OUTCSYS(NOUTPTS,NDIF)
          ENDIF
        ENDIF
        IF (.NOT.(DONE)) GOTO 100

      stop
      end
```

```
C*********************************************************************
      SUBROUTINE SGAINP(NIN,NOUT,N,BOUNDS,OPFITS,POPSIZ, NELITE,
     +                  MAXGEN, NEVAL,PCROSS, PMUT, RANDSEED,
     +                  NODUP, SCALE, SCMAX, SCMIN, NRUN)
      IMPLICIT NONE
      INTEGER N,POPSIZ,NELITE,MAXGEN,NEVAL,NODUP,
     +     SCALE,NRUN,MAXOPS,NOPS,I,NIN,NOUT
      PARAMETER (MAXOPS=20)
      DOUBLE PRECISION  BOUNDS(0:3*N),OPFITS(0:3*MAXOPS),PCROSS,PMUT,
     +                  RANDSEED,SCMAX,SCMIN
      CHARACTER*20 FILENAME

        WRITE(*,*) 'Give File in quotes for Operators and bounds'
        READ(*,*) FILENAME
        OPEN(UNIT=1,FILE=FILENAME,STATUS='OLD')
        READ(1,*) I, NOPS
        IF (I.NE.N) THEN
          WRITE(*,*) 'Improper file'
          STOP
        ENDIF
        BOUNDS(0)=N
        OPFITS(0)=NOPS
        READ(1,*) (OPFITS(I),I=1,NOPS)
        READ(1,*) (OPFITS(I+NOPS),I=1,NOPS)
        READ(1,*) (OPFITS(I+2*NOPS),I=1,NOPS)
        READ(1,*)
        READ(1,*) (BOUNDS(I),I=1,N)
        READ(1,*) (BOUNDS(I+N),I=1,N)
        READ(1,*) (BOUNDS(I+2*N),I=1,N)
        CLOSE(1)
        RANDSEED=0.1678943251

        SCALE=3
        NODUP=3
        SCMAX=1.0
        SCMIN=0.01
        NELITE = 95
        NEVAL=1000
        POPSIZ=100
```

```
      MAXGEN=1000
      PCROSS=0.80
      PMUT=0.004

      write(NOUT,*) 'Give rand seed :'
      read(NIN,*) randseed

      WRITE(NOUT,*) 'Number of times to run (for averaging) :'
      READ(NIN,*) NRUN

      Write(NOUT,*) 'Give POPSIZE,NELITE,MAXGEN,MAXEVAL'
      Read(NIN,*) POPSIZ,NELITE,MAXGEN,NEVAL
      Write(Nout,*) 'Give Pcross, Pmut'
      Read(*,*) pcross,pmut
      Write(Nout,*) 'Give fitness scale type, Scale Max,'
      Write(Nout,*) '     Scale min, Duplication type'
      Read(Nin,*) SCALE,SCMAX,SCMIN,NODUP
    RETURN
    END

    SUBROUTINE SQPINP (NIN,NOUT,N,BL,BU,STEPTL,CONTL,LINETA)
    IMPLICIT NONE
    INTEGER NIN,NOUT,N,I
    DOUBLE PRECISION BL(N),BU(N+1),STEPTL,CONTL,LINETA,BIG
C..     Relaxed Line search. 0:strict, 1:relaxed.
      WRITE(NOUT,*) 'GIVE Func Tol, Constr. Tol,',
     +    '  Line search. 0:strict -> 1:relaxed.'
      READ(NIN,*) STEPTL,CONTL,LINETA

      WRITE(NOUT,*) 'GIVE LOWER, UPPER BOUNDS ON SEPERATE LINES'
      WRITE(NOUT,*) 'EQUAL BOUNDS WILL BE TREATED AS INFINITE'
      BIG=1.D10
      READ(NIN,*) (BL(I),I=1,N)
      READ(NIN,*) (BU(I),I=1,N)
      DO 10 I=1,N
         IF (BL(I).EQ.BU(I)) THEN
           BL(I)=-BIG
           BU(I)=BIG
         ENDIF
 10   CONTINUE
```

```
  BU(N+1)=BIG
RETURN
END


SUBROUTINE PENINP(NIN,NOUT,GRADTL,STEPTL,STEPMX,CONTL,
+     ITNLIM,MAXITR)
IMPLICIT NONE
INTEGER NIN,NOUT,ITNLIM,MAXITR
DOUBLE PRECISION GRADTL,STEPTL,STEPMX,CONTL
   WRITE(NOUT,*)'Give Gradient Tol, Step tol, ',
+     'Max step allowed, Constraint Tol'
   READ (NIN, *)  GRADTL, STEPTL, STEPMX, CONTL
   WRITE(NOUT,*) 'Give Max iterations for each Subproblem :'
   READ (NIN, *)  ITNLIM
   WRITE (NOUT,*) 'Give max penalty subprobs to solve : '
   READ (NIN,*) MAXITR
RETURN
END


SUBROUTINE COLSINP (NIN,NOUT,DYNPROB,DYNTIME,NPROBS,NINT,
+                    K,MAXINT,EPSO,EPSI,EPSMIN)
LOGICAL DYNPROB,DYNTIME
INTEGER NPROBS,NINT,K,MAXINT
DOUBLE PRECISION EPSO,EPSI,EPSMIN
      WRITE(NOUT,*) 'GIVE (T/F) DYNTIME, DYNPROB,NPROBS'
      READ(NIN,*) DYNTIME,DYNPROB,NPROBS
      WRITE(NOUT,*) 'GIVE N INTERVALS ,MAX INTERVALS'
      READ(NIN,*) NINT,MAXINT
      WRITE(NOUT,*) 'GIVE NO. OF COLLOCATION PTS/INTERVAL'
      READ(NIN,*) K
      WRITE(NOUT,*) 'GIVE EPSo, EPSinc, Min EPS'
      READ(NIN,*) EPSO,EPSI,EPSMIN
RETURN
END

SUBROUTINE GENINP (NIN,NOUT,N,N1,MCON,PROBTYP,FNTYPE,PINDEX,
+     METHOD,CONTYPE,INITYP,PARTYPE)
```

```
      IMPLICIT NONE
      INTEGER NIN,NOUT,PROBTYP,FNTYPE,METHOD,N,N1,
     +     PINDEX,CONTYPE,INITYP,PARTYPE,MCON
         WRITE(NOUT,*) 'GENERALISED TRAJECTORY PLANNER & OPTIMIZER '
         WRITE(NOUT,*) 'Give Problem type (1:EMARS,2:ESCAPE,3:EMOON)'
         READ(NIN,*) PROBTYP
         WRITE(NOUT,*) 'GIVE FUNCTION TYPE M:1,2, E:3,4,5 M:6-8 '
         WRITE(NOUT,*) '     COLSYS 1) e-mars, 2) Escape'
         READ(NIN,*) FNTYPE
         WRITE(NOUT,*) 'Give Algorithm to use : ',
     +                '1:Colsys, 2:Multiplier PFM, 3:SQP, 4:GA, 5:PFM'
         READ(NIN,*) METHOD
         WRITE(NOUT,*) 'Give Constraint Type (for colsys)'
         WRITE(NOUT,*) '1) E-Mars 2) Version 2.,  3) Escape '
         WRITE(NOUT,*) 'For Esc : 1) TSC, 2) NONE '
         READ(NIN,*) CONTYPE
         WRITE(NOUT,*) 'Give initial Data code (1:Mars,2:Earth,3:EM)'
         READ(NIN,*) INITYP
         WRITE(NOUT,*) 'Give Parameterization (Direct Problems) &',
     +                 'No. of control parameters (in direct)'
         WRITE(NOUT,*) '1 : Polynomial     2 : Normalized Polynomial'
         WRITE(NOUT,*) '3 : Taylors Series 4 : Spline, 0 : NONE'
         READ(NIN,*) PARTYPE,N1

         GOTO (41,42,43,44,45,46,47,48,49) FNTYPE
         WRITE(*,*) 'MAIN:INVALID FNTYPE ',FNTYPE
         STOP
41          MCON=3
         goto 60
42          N=4
            MCON=3
         GOTO 50
43          MCON=0
         GOTO 60
44          N=3
            MCON=3
         GOTO 50
45          N=3
            MCON=3
         GOTO 50
```

```
46          N=12
            MCON=3
         GOTO 50
47          N=11
            MCON=3
         GOTO 50
48          N=11
            MCON=4
         GOTO 50
49          N=11
            MCON=3
         GOTO 50
60       CONTINUE
            GOTO (31,32) PROBTYP
            WRITE(NOUT,*) 'DIRECT METHOD INVALID, PROB : ',PROBTYP
            STOP
31            N=N1+1
              PINDEX=2
            GOTO 40
32            N=N1
              PINDEX=1
40          CONTINUE
50       CONTINUE

      RETURN
      END



      SUBROUTINE GMOON
      WRITE(*,*)' DUMMY SUB,GMOON'
      STOP
      END
```

Initial Data Module

The three routines MARINIT, EARINIT and MOONINI, respectively supply initial data for the Mars transfer, the Earth escape and the Earth-Moon transfer.

```
C-----------------------------------------------------------------
C                      INITIALIZATION MODULES
C-----------------------------------------------------------------
      SUBROUTINE MARINIT (Mu,ro,to,rf,tf,t2init,Thrust,mo,mdot)
      IMPLICIT NONE
      DOUBLE PRECISION Mu,ro,to,rf,tf,t2init,Thrust,mo,mdot
         TO=0.0
         tf=3.0
         T2INIT= 4.0
         rf=1.525
         ro=1.0
         mu = 1.0
         Thrust = 0.1405
         Mo = 1.0
         Mdot = 0.07487
      RETURN
      END

      SUBROUTINE EARINIT (Mu,ro,to,rf,tf,t2init,Thrust,mo,mdot)
      IMPLICIT NONE
      DOUBLE PRECISION Mu,ro,to,rf,tf,t2init,Thrust,mo,mdot,
     +                 Re,Day,Mass,Len
         Re   = 6378.14453125
         Len  = Re
         Day  = DBLE(0.5*24*3600)
         Mass = 1.0D05
         Ro = (315.0+Re)/Len
         mu = 3.986011875D05
         DAY = SQRT(Len**3/Mu)

         Mu = Day*Day/Len**3 * Mu
         TO=0.0
```

```
   T2INIT=2.38*24.*3600./Day
   TF=2.38*24.*3600./Day
   rf=15.0*Re/Len
   Mo = 1.D5/Mass
   Thrust = 2942.0/Mass* Day*Day/Len/1.D03
   Mdot = 0.0299*Day/Mass
RETURN
END


SUBROUTINE MOONINI (Mu,Mu1,D1,W,Ro,Rf,
+                    Thrust,mo,mdot,DAY,Re,Rm)
IMPLICIT NONE
DOUBLE PRECISION Mu,Mu1,D1,W,Ro,Rf,Thrust,mo,mdot,
+                 Rm,Re,Day,Mass,Len,TR
   Re   = 6378.14453125
   Rm   = 1738.0
   D1   = 384400.0
   TR   = 4670.71094
   Len  = Re
   Mass = 1.0D05
   mu   = 3.986011875D05
   Mu1  = Mu * (TR/(D1-TR))
   W    = SQRT((Mu+Mu1)/D1**3)
   DAY  = SQRT(Len**3/Mu)


   Ro = (315.0+Re)/Len
   Rf = (100.0+Rm)/Len
   Re = Re/Len
   Rm = Rm/Len
   Mu = Day*Day/Len**3 * Mu
   Mu1= Day*Day/Len**3 * Mu1
   W  = W*DAY
   D1 = D1/Len
   Mo = 1.D5/Mass
   Thrust = 2942.0/Mass* Day*Day/Len/1.D03
   Mdot = 0.0299*Day/Mass
RETURN
END
```

```
C-----------------------------------------------------------------
```

Simulation Module

This module contains the routine FPLANET which controls the other simulation routines for different problems. This enables problem specification in input. The routines FMARS, FMARS2, FESCAPE, FESCAPE2, FESCAPE3, E_MOON and E_MOON(2,3,4) setup the variables to call SOLPATH which calls the multivalue integrator, and evaluate the objective functions and constraints. These routines pass on the state or costate equations (MARS through ALLBODY) to be used as an argument to SOLPATH. Routines for parameterizing the control time history for the direct methods are THANGLE and INITANG. The other routines (OUT*) are the output routines called by the integrator at each successful step.

```
C*******************************************************************
C                              PROBLEM SIMULATIONS
C*******************************************************************

      FUNCTION FPLANET (N,X,I)
C     NOTE : FPLANET (N,X,0) MUST BE CALLED BEFORE ANY OTHER CALLS.
      IMPLICIT NONE
      INTEGER FNTYPE,CONTYPE,PARTYPE,PROBTYP,N,I,j,
     +     IFCNT,IGCNT,IHCNT,TFCNT,THTYPE
      DOUBLE PRECISION X(N),C(20),FPLANET,TMP(30),TMP1(30),
     +     FMARS,FMARS2,FESCAPE,FESCAPE2,E_MOON,E_MOON2,E_MOON3,
     +     FEscape3,THETA,E_MOON4
      CHARACTER*20 FILE
      LOGICAL INIT
      COMMON /FNSPECS/ FNTYPE,CONTYPE,PARTYPE,PROBTYP
      COMMON /CLOCAL / C
      COMMON /COUNT  / IFCNT,IGCNT,IHCNT
```

```
      COMMON /MVCOUNT/ TFCNT
      COMMON /POUT/ THETA,INIT,THTYPE,File
      COMMON /EXTEMP / TMP,TMP1

      FILE='plot.dat'
      IF (I.EQ.0) THEN
        IFCNT=IFCNT+1
        GOTO (1,2,3,4,5,6,7,8,9) FNTYPE
        WRITE(*,*) 'INVALID FUNCTION',FNTYPE
        STOP

1          THTYPE=1
           FPLANET = FMARS  (N,X,C)
        GOTO 10
2          THTYPE=3
           FPLANET = FMARS2 (N,X,C)
        GOTO 10
3          THTYPE=1
           FPLANET = FESCAPE  (N,X,C)
        GOTO 10
4          THTYPE=3
           FPLANET = FESCAPE2 (N,X,C)
        GOTO 10
5          THTYPE=2
           FPLANET = FESCAPE3 (N,X,C)
        GOTO 10
6          THTYPE=4
           FPLANET = E_MOON  (N,X,C)
        GOTO 10
7          THTYPE=4
           FPLANET = E_MOON2 (N,X,C)
        GOTO 10
8          THTYPE=4
           FPLANET = E_MOON3 (N,X,C)
        GOTO 10
9          THTYPE=4
           FPLANET = E_MOON4 (N,X,C)
10      CONTINUE
        do 11 j=1,n
11         tmp(j)=x(j)
```

```
C             write(*,*) FPLANET,'    ',C(1),c(2),c(3)
      ELSE
        do 12 j=1,n
           IF (tmp(j).NE.x(j)) then
             write(*,*) 'function mismatch ',x(j),tmp(j)
             stop
           endif
 12     continue

         FPLANET = C(I)
      ENDIF

      RETURN
      END

      FUNCTION FMARS (N,X,C)
C     Set N=N1+1, MCON=3
      IMPLICIT NONE
      INTEGER N,NDIF,MAXK,i,FNTYPE,CONTYPE,PARTYPE,PINDEX,METHOD
      PARAMETER (NDIF=3,MAXK=20)
      DOUBLE PRECISION X(N), MU, DELTA, tf,rf, ro,
     +       FMARS, Y(NDIF,MAXK), Thrust,mo,mdot,C(3)
      EXTERNAL MARS,OUT
      COMMON /FNSPECS/ FNTYPE,CONTYPE,PARTYPE,PINDEX,METHOD
      COMMON/EXMARS/MU,DELTA, TF, RF, RO,Thrust,mo,mdot
C     TMP CHECKS FOR ILLEGAL CALLS TO FMARS. MAY BE REMOVED.

      Tf   = X(1)
      Y(1,1)=RO
      Y(2,1)=0.0
      Y(3,1)=SQRT(MU/RO)
      CALL SOLPATH (Y,NDIF,MAXK,N,X,MARS,OUT,.FALSE.)
C     ... for Penalty Problem.
      IF (METHOD.EQ.3) THEN
        Fmars = X(1)*X(1)*0.1
      ELSE
        Fmars = X(1)*X(1)*0.001
      ENDIF

      C(1) = (Y(1,1)-RF)
```

```
      C(2) = Y(2,1)
      C(3) = (Y(3,1)-SQRT(MU/RF))
      RETURN
      END



      FUNCTION FMARS2 (N,X,C)
C     Set N=N1+1
      IMPLICIT NONE
      INTEGER N,NDIF,MAXK,j,TFCNT,FNTYPE,CONTYPE,
     +     PARTYPE,PINDEX,METHOD
      PARAMETER (NDIF=6,MAXK=10)
      DOUBLE PRECISION X(N), MU, DELTA, tf,rf, ro,
     +       FMARS2, Y(NDIF,MAXK), X1(10),
     +       Thrust,mo,mdot, C(3)
      EXTERNAL ALSTAT1,OUT
      COMMON/MVCOUNT/TFCNT
      COMMON /FNSPECS/ FNTYPE,CONTYPE,PARTYPE,PINDEX,METHOD
      COMMON/EXMARS/MU,DELTA, TF, RF, RO,Thrust,mo,mdot

      Y(1,1)=Ro
      Y(2,1)=0.0
      Y(3,1)=SQRT(MU/RO)
      Tf    =X(1)
      Y(4,1)=X(2)
      Y(5,1)=X(3)
      Y(6,1)=X(4)
      CALL SOLPATH (Y,NDIF,MAXK,N,X1,ALSTAT1,OUT,.FALSE.)
      IF (METHOD.EQ.3) THEN
        Fmars2 = Tf*Tf*0.1
      ELSE
        Fmars2 = Tf*Tf**0.001
      ENDIF

      C(1) = Y(2,1)
      C(2) = Y(3,1) - SQRT (MU/RF)
      C(3) = (Y(1,1)- RF)*4.
      RETURN
      END
```

```
      FUNCTION FESCAPE (N,X)
C     Set N=N1+1, MCON=0
      IMPLICIT NONE
      INTEGER N,NDIF,MAXK,TFCNT
      PARAMETER (NDIF=3,MAXK=10)
      DOUBLE PRECISION X(N), MU, DELTA, tf,rf, ro,
     +       FESCAPE, Y(NDIF,MAXK),
     +       Thrust,mo,mdot
      EXTERNAL MARS,OUT
      COMMON/MVCOUNT/TFCNT
      COMMON/EXMARS/MU,DELTA, TF, RF, RO,Thrust,mo,mdot

      Y(1,1)=RO
      Y(2,1)=0.0
      Y(3,1)=SQRT(MU/RO)
      CALL SOLPATH (Y,NDIF,MAXK,N,X,MARS,OUT,.FALSE.)
      FESCAPE = -0.5*(Y(2,1)*Y(2,1) + Y(3,1)*Y(3,1)) + MU/Y(1,1)
      RETURN
      END



      FUNCTION FESCAPE2 (N,X,C)
C     Set N=N1+1, MCON=0
      IMPLICIT NONE
      INTEGER N,NDIF,MAXK,j,TFCNT,FNTYPE,CONTYPE,PARTYPE,PINDEX
      PARAMETER (NDIF=6,MAXK=10)
      DOUBLE PRECISION X(N), MU, DELTA, tf,rf, ro,
     +       FESCAPE2, Y(NDIF,MAXK), X1(10),
     +       Thrust,mo,mdot, C(3)
      EXTERNAL ALSTAT1,OUTESC1,Out
      COMMON/MVCOUNT/TFCNT
      COMMON /FNSPECS/ FNTYPE,CONTYPE,PARTYPE,PINDEX
      COMMON/EXMARS/MU,DELTA, TF, RF, RO,Thrust,mo,mdot

      Y(1,1)=Ro
      Y(2,1)=0.0
```

```
      Y(3,1)=SQRT(MU/RO)
      Y(4,1)=X(1)
      Y(5,1)=X(2)
      Y(6,1)=X(3)
      CALL SOLPATH (Y,NDIF,MAXK,N,X1,ALSTAT1,OUT,.FALSE.)
      FESCAPE2 =-0.5*(Y(2,1)*Y(2,1)+Y(3,1)*Y(3,1))+MU/Y(1,1)
      IF (CONTYPE.EQ.1) THEN
        C(1)=Y(4,1)-MU/(Y(1,1)*Y(1,1))
        C(2)=Y(5,1)-Y(2,1)
        C(3)=Y(6,1)-Y(3,1)
        Fescape2=0.0
      ELSE
        c(1)=0.0
        c(2)=0.0
        c(3)=0.0
      ENDIF
      RETURN
      END

      FUNCTION FESCAPE3 (N,X,C)
C     Set N=N1+1, MCON=0
      IMPLICIT NONE
      INTEGER N,NDIF,MAXK,j,TFCNT,FNTYPE,CONTYPE,PARTYPE,PINDEX
      PARAMETER (NDIF=6,MAXK=10)
      DOUBLE PRECISION X(N), MU, DELTA, tf,rf, ro,
     +       FESCAPE3, Y(NDIF,MAXK), X1(10),
     +       Thrust,mo,mdot, C(3)
      EXTERNAL ALSTAT2,OUT,OUTESC2
      COMMON/MVCOUNT/TFCNT
      COMMON /FNSPECS/ FNTYPE,CONTYPE,PARTYPE,PINDEX
      COMMON/EXMARS/MU,DELTA, TF, RF, RO,Thrust,mo,mdot

      Y(1,1)=Ro
      Y(2,1)=0.0
      Y(3,1)=SQRT(MU/RO)
      Y(4,1)=X(1)
      Y(5,1)=X(2)
      Y(6,1)=X(3)
      CALL SOLPATH (Y,NDIF,MAXK,N,X1,ALSTAT2,OUT,.FALSE.)
      FESCAPE3 =-0.5*(Y(2,1)*Y(2,1)+Y(3,1)*Y(3,1))+MU/Y(1,1)
```

```
      Write(*,*) 'e=',Fescape3
      IF (CONTYPE.EQ.1) THEN
        C(1)=Y(4,1)-MU/(Y(1,1)*Y(1,1))
        C(2)=Y(5,1)-Y(2,1)
        C(3)=Y(6,1)-Y(3,1)
        Fescape3=0.0
      ELSE
        c(1)=0.0
        c(2)=0.0
        c(3)=0.0
      ENDIF
      RETURN
      END



      FUNCTION E_MOON2 (N,X,C)
C     Set N=11, MCON=3
      IMPLICIT NONE
      INTEGER N,NDIF,MAXK,j,NDIF2
      PARAMETER (NDIF=8,NDIF2=4,MAXK=10)
      DOUBLE PRECISION X(N), MU, DELTA, tf,rf, ro,
     +       E_MOON2, Y(NDIF,MAXK), X1(20), R1,TH1,U1,V1,
     +       Thrust,mo,mdot, C(3), Mu1,D1,W, Y2(NDIF2,MAXK),
     +       Smo,SMU,Smu1,DAY,DAY1,ENERGY,E,E0,Rs, Hamilt, Re,Rm
      EXTERNAL ALLBODY,OUT
      COMMON/EXMARS/MU,DELTA, TF, RF, RO,Thrust,mo,mdot
      COMMON/PLANET/Mu1,D1,W,DAY,Re,Rm
      ENERGY(W) = 0.5*(Y(2,1)**2+(Y(3,1)+W*Y(1,1))**2)-MU/Y(1,1)

      Smo  =Mo
      Smu  =Mu
      Smu1 =Mu1
      Day1  =DAY/(24.*3600.)
      Tf    =X(2)/DAY1
      Y(1,1)=Ro
      Y(2,1)=0.0
      Y(3,1)=SQRT(MU/RO)-W*Ro
      Y(4,1)=X(1)
```

```
      Y(5,1)=X(3)
      Y(6,1)=X(4)*(X(3)+X(5))
      Y(7,1)=X(3)+X(5)
      Y(8,1)=0.0
      EO=ENERGY(W)
      Rs=Y(1,1)
      CALL SOLPATH (Y,NDIF,MAXK,N,X1,ALLBODY,OUT,.FALSE.)

      E=ENERGY(W)
      IF (((E.LT.EO).OR.(Y(1,1).LT.Rs)).OR.
     +     (Y(1,1).LT.Ro*5.0)) THEN
        R1=Y(1,1)
        U1=Y(2,1)
        V1=Y(3,1)
        TH1=Y(4,1)
        CALL TRANSFORM(R1,TH1,U1,V1,D1,W,
     +       Y(1,1),Y(4,1),Y(2,1),Y(3,1))
        GOTO 100
      ENDIF
      RS=Y(1,1)

      Mo     = Mo-X(2)*Mdot
      Tf     = X(6)/DAY1
      Y2(1,1)=Y(1,1)
      Y2(2,1)=Y(2,1)
      Y2(3,1)=Y(3,1)
      Y2(4,1)=Y(4,1)
      CALL SOLPATH (Y2,NDIF2,MAXK,N,X1,ALLBODY,OUT,.FALSE.)

      CALL TRANSFORM(Y2(1,1),Y2(4,1),Y2(2,1),Y2(3,1),D1,W,
     +     Y(1,1),Y(4,1),Y(2,1),Y(3,1))
      IF ((Y2(1,1).LT.RS).OR.(Y(1,1).GT.D1*0.75)
     +           .OR.(Y(1,1).LT.Rm)) THEN
        GOTO 100
      ENDIF
      Mu     =SMu1
      Mu1    =Smu
      Tf     =X(11)/DAY1
      Y(5,1)=X(7)
      Y(6,1)=(X(9)+X(7))/X(8)
```

```
      Y(7,1)=X(9)+X(7)
      Y(8,1)=0.33
      CALL SOLPATH (Y,NDIF,MAXK,N,X1,ALLBODY,OUT,.FALSE.)
 100  continue
      Mo=Smo
      Mu=Smu
      Mu1=Smu1

      E_MOON2 = X(2)+X(11)
      C(1)=Y(1,1)-Rf
      C(2)=Y(2,1)
      C(3)=ABS(Y(3,1)+W*Y(1,1))-SQRT(MU1/Rf)
      C(1)=C(1)*20.
      C(2)=C(2)*5.
      C(3)=C(3)*5.
      RETURN
      END

      FUNCTION E_MOON (N,X,C)
C     Set N=12, MCON=3
      IMPLICIT NONE
      INTEGER N,NDIF,MAXK,j,NDIF2
      PARAMETER (NDIF=8,NDIF2=4,MAXK=10)
      DOUBLE PRECISION X(N), MU, DELTA, tf,rf, ro,
     +       E_MOON, Y(NDIF,MAXK), X1(20), R1,TH1,U1,V1,
     +       Thrust,mo,mdot, C(3), Mu1,D1,W, Y2(NDIF2,MAXK),
     +       Smo,SMU,Smu1,DAY,DAY1,ENERGY,E,EO,Rs, Hamilt
      EXTERNAL ALLBODY,OUT
      COMMON/EXMARS/MU,DELTA, TF, RF, RO,Thrust,mo,mdot
      COMMON/PLANET/Mu1,D1,W,DAY
      ENERGY(W) = 0.5*(Y(2,1)**2+(Y(3,1)+W*Y(1,1))**2)-MU/Y(1,1)

        Smo  =Mo
        Smu  =Mu
        Smu1 =Mu1
        Day1  =DAY/(24.*3600.)
        Tf    =X(2)/DAY1
        Y(1,1)=Ro
        Y(2,1)=0.0
        Y(3,1)=SQRT(MU/RO)-W*Ro
```

```
      Y(4,1)=X(1)

      Y(5,1)=X(3)
      Y(6,1)=X(4)*(X(3)+X(5))
      Y(7,1)=X(3)+X(5)
      Y(8,1)=X(6)
      E0=ENERGY(W)
      Rs=Y(1,1)
      CALL SOLPATH (Y,NDIF,MAXK,N,X1,ALLBODY,OUT,.FALSE.)

      E=ENERGY(W)
902   FORMAT ('Y ',8(G9.3,1X))
      IF ((E.LT.E0).OR.(Y(1,1).LT.Rs)) THEN
        R1=Y(1,1)
        U1=Y(2,1)
        V1=Y(3,1)
        TH1=Y(4,1)
        CALL TRANSFORM(R1,TH1,U1,V1,D1,W,
     +        Y(1,1),Y(4,1),Y(2,1),Y(3,1))
        GOTO 100
      ENDIF
      RS=Y(1,1)

      Mo    = Mo-X(2)*Mdot
      Tf    = X(7)/DAY1
      Y2(1,1)=Y(1,1)
      Y2(2,1)=Y(2,1)
      Y2(3,1)=Y(3,1)
      Y2(4,1)=Y(4,1)
      CALL SOLPATH (Y2,NDIF2,MAXK,N,X1,ALLBODY,OUT,.FALSE.)

      CALL TRANSFORM(Y2(1,1),Y2(4,1),Y2(2,1),Y2(3,1),D1,W,
     +      Y(1,1),Y(4,1),Y(2,1),Y(3,1))
      IF ((Y2(1,1).LT.RS).OR.(Y(1,1).GT.D1)) THEN
        GOTO 100
      ENDIF
      Mu    =SMu1
      Mu1   =Smu
      Tf    =X(12)/DAY1
      Y(5,1)=X(8)
```

```
      Y(6,1)=(X(10)+X(8))/X(9)
      Y(7,1)=X(10)+X(8)
      Y(8,1)=X(11)
      CALL SOLPATH (Y,NDIF,MAXK,N,X1,ALLBODY,OUT,.FALSE.)
100   continue
      Mo=Smo
      Mu=Smu
      Mu1=Smu1

      C(1)=Y(1,1)-Rf
      C(2)=Y(2,1)
      C(3)=ABS(Y(3,1)+W*Y(1,1))-SQRT(MU1/Rf)
      E_MOON = X(2)+X(12)
      C(1)=C(1)*10.
      C(2)=C(2)*5.
      C(3)=C(3)*5.
    RETURN
    END



    FUNCTION E_MOON3 (N,X,C)
C   Set N=11, MCON=3
    IMPLICIT NONE
    INTEGER N,NDIF,MAXK,j,NDIF2
    PARAMETER (NDIF=8,NDIF2=4,MAXK=10)
    DOUBLE PRECISION X(N), MU, DELTA, tf,rf, ro,
   +       E_MOON3, Y(NDIF,MAXK), X1(20), R1,TH1,U1,V1,
   +       Thrust,mo,mdot, C(4), Mu1,D1,W, Y2(NDIF2,MAXK),
   +       Smo,SMU,Smu1,Smdot,DAY,DAY1,ENERGY,E,E0,Rs, Re,Rm
    EXTERNAL ALLBODY,OUT
    COMMON/EXMARS/MU,DELTA, TF, RF, RO,Thrust,mo,mdot
    COMMON/PLANET/Mu1,D1,W,DAY,Re,Rm
    ENERGY(W) = 0.5*(Y(2,1)**2+(Y(3,1)+W*Y(1,1))**2)-MU/Y(1,1)

       Smo  =Mo
       Smu  =Mu
       Smu1 =Mu1
       Smdot=Mdot
       Day1  =DAY/(24.*3600.)
```

```
      Tf     =X(2)/DAY1
      Y(1,1)=Ro
      Y(2,1)=0.0
      Y(3,1)=SQRT(MU/RO)-W*Ro
      Y(4,1)=X(1)

      Y(5,1)=X(3)
      Y(6,1)=X(4)*(X(3)+X(5))
      Y(7,1)=X(3)+X(5)
      Y(8,1)=0.0
      EO=ENERGY(W)
      Rs=Y(1,1)
      CALL SOLPATH (Y,NDIF,MAXK,N,X1,ALLBODY,OUT,.FALSE.)

      E=ENERGY(W)
      Y2(1,1)=Y(1,1)
      Y2(2,1)=Y(2,1)
      Y2(3,1)=Y(3,1)
      Y2(4,1)=Y(4,1)
      Mo     = Mo-X(2)/Day1*Mdot
      Tf     = X(6)/DAY1
      CALL SOLPATH (Y2,NDIF2,MAXK,N,X1,ALLBODY,OUT,.FALSE.)
      CALL TRANSFORM(Y2(1,1),Y2(4,1),Y2(2,1),Y2(3,1),D1,W,
     +       R1,Th1,U1,V1)

      Mu     =SMu1
      Mu1    =Smu
      Mdot   =-Smdot
      Tf     =X(11)/DAY1
      Mo     =Mo-X(11)/Day1*Smdot
      Y(1,1)=Rf
      Y(2,1)=0.0
      Y(3,1)=SQRT(MU/Rf)-W*Rf
      Y(4,1)=X(10)
      Y(5,1)=X(7)
      Y(6,1)=(X(9)+X(7))*X(8)
      Y(7,1)=X(9)+X(7)
      Y(8,1)=0.0
      CALL SOLPATH (Y,NDIF,MAXK,N,X1,ALLBODY,OUT,.FALSE.)
      CALL  NORMANG(Y(4,1))
```

```
100     continue
        Mo=Smo
        Mu=Smu
        Mu1=Smu1
        Mdot=Smdot
        C(1)=Y(1,1)-R1
        C(2)=Y(2,1)-U1
        C(3)=ABS(Y(3,1)-V1)
        C(4)=ABS(Y(4,1)-Th1)

        E_MOON3 = (X(2)+X(11))*0.01
      RETURN
      END


      FUNCTION E_MOON4 (N,X,C)
C     Set N=11, MCON=3
      IMPLICIT NONE
      INTEGER N,NDIF,MAXK,j,NDIF2
      PARAMETER (NDIF=8,NDIF2=4,MAXK=10)
      DOUBLE PRECISION X(N), MU, DELTA, tf,rf, ro,
     +       E_MOON4, Y(NDIF,MAXK), X1(20), R1,TH1,U1,V1,
     +       Thrust,mo,mdot, C(4), Mu1,D1,W, Y2(NDIF2,MAXK),
     +       Smo,SMU,Smu1,Smdot,DAY,DAY1,ENERGY,E,E0,Rs, Re,Rm
      EXTERNAL ALLBODY,OUT
      COMMON/EXMARS/MU,DELTA, TF, RF, RO,Thrust,mo,mdot
      COMMON/PLANET/Mu1,D1,W,DAY,Re,Rm
      DOUBLE PRECISION THETA
      INTEGER THTYPE
      LOGICAL INIT
      CHARACTER*20 FILE
      COMMON /POUT/ THETA,INIT,THTYPE,File
      ENERGY(W) = 0.5*(Y(2,1)**2+(Y(3,1)+W*Y(1,1))**2)-MU/Y(1,1)

        Smo  =Mo
        Smu  =Mu
        Smu1 =Mu1
        Smdot=Mdot
        Day1  =DAY/(24.*3600.)
        Tf    =X(2)/DAY1
```

```
      Y(1,1)=Ro
      Y(2,1)=0.0
      Y(3,1)=SQRT(MU/RO)-W*Ro
      Y(4,1)=X(1)

      Y(5,1)=X(3)
      Y(6,1)=X(4)*(X(3)+X(5))
      Y(7,1)=X(3)+X(5)
      Y(8,1)=0.0
      EO=ENERGY(W)
      Rs=Y(1,1)
      FILE='plot1.dat'
      CALL SOLPATH (Y,NDIF,MAXK,N,X1,ALLBODY,OUT,.FALSE.)

      E=ENERGY(W)

      Y2(1,1)=Y(1,1)
      Y2(2,1)=Y(2,1)
      Y2(3,1)=Y(3,1)
      Y2(4,1)=Y(4,1)
      Mo    = Mo-X(2)/Day1*Mdot
      Tf    = X(6)/DAY1
C-------------------------------
      THTYPE=1
      THETA=0.0
      FILE='plot2.dat'
      CALL SOLPATH (Y2,NDIF2,MAXK,N,X1,ALLBODY,OUT,.FALSE.)
      CALL TRANSFORM(Y2(1,1),Y2(4,1),Y2(2,1),Y2(3,1),D1,W,
     +       R1,Th1,U1,V1)

      Mu    =SMu1
      Mu1   =Smu
      Mdot  =-Smdot
      Tf    =X(11)/DAY1
      Mo    =Mo-X(11)/Day1*Smdot
      Y(1,1)=Rf
      Y(2,1)=0.0
      Y(3,1)=SQRT(MU/Rf)-W*Rf
      Y(4,1)=X(10)
      Y(5,1)=X(7)
```

```
      Y(6,1)=(X(9)+X(7))*X(8)
      Y(7,1)=X(9)+X(7)
      Y(8,1)=0.0
C-------------------------------
      THTYPE=4
      FILE='plot3.dat'
      CALL SOLPATH (Y,NDIF,MAXK,N,X1,ALLBODY,OUT,.FALSE.)
      CALL  NORMANG(Y(4,1))
 100  continue
      Mo=Smo
      Mu=Smu
      Mu1=Smu1
      Mdot=Smdot
      C(1)=SQRT((Y(1,1)*COS(Y(4,1))-R1*COS(TH1))**2
     +        +(Y(1,1)*SIN(Y(4,1))-R1*SIN(TH1))**2)
      C(2)=Y(2,1)-U1
      C(3)=Y(3,1)-V1

      E_MOON4 = (X(2)+X(11))*0.01
    RETURN
    END


    SUBROUTINE TRANSFORM (R,THETA,U,V,D,W,R2,THETA2,U2,V2)
    IMPLICIT NONE
    DOUBLE PRECISION R,THETA,U,V,D,W,R2,THETA2,U2,V2,
     +                RSIN2,RCOS2,SINT,COST

      SINT=SIN(THETA)
      COST=COS(THETA)
      RSIN2=R*SINT
      RCOS2=D-R*COST
      R2 = DSQRT(RSIN2*RSIN2 + RCOS2*RCOS2)
      THETA2= -ATAN2(RSIN2,RCOS2)
      CALL NORMANG(THETA2)
      RCOS2=V*SINT-U*COST
      RSIN2=-(U*SINT+V*COST)+D*W
      SINT=SIN(THETA2)
      COST=COS(THETA2)
```

```
  U2=COST*RCOS2+SINT*RSIN2
  V2=-SINT*RCOS2+COST*RSIN2
RETURN
END

SUBROUTINE NORMANG (THETA)
IMPLICIT NONE
DOUBLE PRECISION THETA, TWOPI
  TWOPI=8.*ATAN(1.0D0)
  DO WHILE (THETA.GT.TWOPI)
     THETA=THETA-TWOPI
  ENDDO
  DO WHILE (THETA.LT.0.0)
     THETA=THETA+TWOPI
  ENDDO
RETURN
END

SUBROUTINE SOLPATH (Y,NDIF,MAXK,N,X,MARS,OUT,STORE1)
IMPLICIT NONE
INTEGER N,J,NDIF,MAXK,FAIL,P,K,FNTYPE,CONTYPE,PARTYPE,THTYPE,
+        FACTORIAL,TFCNT, SKIP,NSKIP, INDEX,PINDEX,METHOD
DOUBLE PRECISION T, X(N), MU, DELTA, MEPS,
+       tf,rf, ro,Thrust,mo,mdot, EPS,FTOL,CTOL,GTOL,
+       Y(NDIF,MAXK), DY(20), H,HMIN,HMAX,THETA
LOGICAL EXIT, STORE, STORE1, PLOT,INIT
PARAMETER (MEPS=1.2E-16)
COMMON /MVCOUNT/ TFCNT
COMMON /EXMARS / MU,DELTA, TF, RF, RO,Thrust,mo,mdot
COMMON /COUT   / SKIP,NSKIP,STORE
character*20 file
COMMON /POUT   / THETA,INIT,THTYPE,file

COMMON /ERSPECS/ EPS,FTOL,CTOL,GTOL
COMMON /FNSPECS/ FNTYPE,CONTYPE,PARTYPE,PINDEX,METHOD,PLOT
EXTERNAL OUT, FACTORIAL, MARS, OUTP

     STORE= STORE1
     P    = 1
     K    = P
```

```
      T    = 0.0
      H    = EPS
      HMIN = MEPS*100.
      HMAX = (Tf-T)/10.
      IF (PARTYPE.NE.0) CALL  INITANG (X(PINDEX),T,TF)

      CALL MARS (P,NDIF,T,Y,DY)
      DO 20 J=1,NDIF
         Y(J,P+1)=H**P/FLOAT(FACTORIAL(P))*DY(J)
20    CONTINUE

      IF (STORE) CALL STOREINIT (NDIF,NDIF,0)
      EXIT=.FALSE.

      SKIP=NSKIP

      IF (PLOT) THEN
        INIT=.TRUE.
        CALL OUTP (NDIF,K,NDIF,Y,T,EXIT)
        CALL MVAL (NDIF,NDIF,Y,T,Tf,H,HMAX,HMIN,EPS,K,P,
     +          MARS,OUTP,FAIL)
      ELSE
        CALL OUT (NDIF,K,NDIF,Y,T,EXIT)
        CALL MVAL (NDIF,NDIF,Y,T,Tf,H,HMAX,HMIN,EPS,K,P,
     +          MARS,OUT,FAIL)
      ENDIF

      IF (STORE) THEN
        CALL  STOREINIT (NDIF,NDIF,1)
        CALL  GETXVAL (TF,Y,1)
      ENDIF
      IF (PLOT) THEN
        WRITE(*,*) 'PLOT DONE',TFCNT
        write(*,*) (y(j,1),j=1,ndif)
        CLOSE(10)
        IF ((File.eq.'plot.dat').or.(file.eq.'plot3.dat')) STOP
      ENDIF

      DO 30 J=1,NDIF
         Y(J,2)=Y(J,2)/H**P*FLOAT(FACTORIAL(P))
```

```
 30         CONTINUE
            IF (FAIL.NE.0) THEN
              WRITE(*,*) 'INTEGRATION MULTIVAL FAILS :',FAIL, tfcnt
C              STOP
            ENDIF
      RETURN
      END


      SUBROUTINE MARS (P,MAX,T,Y,DY)
      implicit NONE
      COMMON /MVCOUNT/IFCNT
      INTEGER P,MAX,IFCNT
      DOUBLE PRECISION T, Y(MAX,P),DY(MAX), ThAngle,
     +    THR, MU, DELTA, THETA, TF, RF, RO, SIN, COS,
     +    Thrust,mo,mdot
      COMMON /POUT/ THETA
      common /EXMARS/ Mu, delta, tf, rf, ro,Thrust,mo,mdot

         IFCNT=IFCNT+1
C        ...CALCULATE SPECIFIC THRUST = "THRUST"
         THR=Thrust/(Mo - Mdot*t)
         Theta = THANGLE (T)
         DY(1) = Y(2,1)
         DY(2) = (Y(3,1)*Y(3,1) - MU/Y(1,1))/Y(1,1)
     +           + THR*SIN(THETA)
         DY(3) = -Y(2,1)*Y(3,1)/Y(1,1) + THR*COS(THETA)
      return
      End

      SUBROUTINE COSTATE (P,MAX,T,Y,DY)
      implicit NONE
      COMMON /MVCOUNT/IFCNT
      INTEGER P,MAX,IFCNT
      DOUBLE PRECISION T, Y(MAX,P),DY(MAX),X(20),Th,
     +                 Mu, delta, tf, rf, ro,Thrust,mo,mdot
      common  /EXMARS/ Mu, delta, tf, rf, ro,Thrust,mo,mdot

        IFCNT=IFCNT+1
        CALL GETXVAL (T,X,1)
```

```
      TH=THRUST/(Mo-Mdot*t)
      DY(1) = -(Y(2,1)*(-X(3)*X(3)+2*MU/X(1)) +
     +            Y(3,1)*X(2)*X(3))/(X(1)*X(1))
      DY(2) = -Y(1,1) + Y(3,1)*X(3)/X(1)
      DY(3) = -2*Y(2,1)*X(3)/X(1) + Y(3,1)*X(2)/X(1)
    RETURN
    END


    SUBROUTINE ALSTAT1 (P,MAX,T,Y,DY)
    implicit NONE
    COMMON /MVCOUNT/IFCNT
    INTEGER P,MAX,IFCNT
    DOUBLE PRECISION T, Y(MAX,P),DY(MAX), TH,SQ,a,b,
     +                 Mu, delta, tf, rf, ro,Thrust,mo,mdot
    common  /EXMARS/ Mu, delta, tf, rf, ro,Thrust,mo,mdot

      IFCNT=IFCNT+1
      TH=THRUST/(Mo - Mdot*t)
      a=Y(5,1)
      B=Y(6,1)
      SQ = SQRT(A*A + B*B)

      DY(1) = Y(2,1)
      DY(2) = (Y(3,1)*Y(3,1) - MU/Y(1,1))/Y(1,1) + TH*A/SQ
      DY(3) = -Y(2,1)*Y(3,1)/Y(1,1) + TH*B/SQ
      DY(4) = -(Y(5,1)*(-Y(3,1)*Y(3,1)+2.*MU/Y(1,1))
     +          + Y(6,1)*Y(2,1)*Y(3,1))/(Y(1,1)*Y(1,1))
      DY(5) = -Y(4,1) + Y(6,1)*Y(3,1)/Y(1,1)
      DY(6) = -2.*Y(5,1)*Y(3,1)/Y(1,1) +Y(6,1)*Y(2,1)/Y(1,1)
    RETURN
    END


    SUBROUTINE ALSTAT2 (P,MAX,T,Y,DY)
    implicit NONE
    COMMON /MVCOUNT/IFCNT
    INTEGER P,MAX,IFCNT
C   ... Note : Alstat2 differs from alstat1 only in definition
C   ...        of A,B & DY(5),DY(6): 1 extra term at end.
```

```
 DOUBLE PRECISION T, Y(MAX,P),DY(MAX), TH,SQ,a,b,
+                 Mu, delta, tf, rf, ro,Thrust,mo,mdot
 common  /EXMARS/ Mu, delta, tf, rf, ro,Thrust,mo,mdot

   IFCNT=IFCNT+1
   TH=THRUST/(Mo - Mdot*t)
   a=Y(2,1)+Y(5,1)
   B=Y(3,1)+Y(6,1)
   SQ = SQRT(A*A + B*B)

   DY(1) = Y(2,1)
   DY(2) = (Y(3,1)*Y(3,1) - MU/Y(1,1))/Y(1,1) + TH*A/SQ
   DY(3) = -Y(2,1)*Y(3,1)/Y(1,1) + TH*B/SQ
   DY(4) = -(Y(5,1)*(-Y(3,1)*Y(3,1)+2.*MU/Y(1,1))
+          + Y(6,1)*Y(2,1)*Y(3,1))/(Y(1,1)*Y(1,1))
   DY(5) = -Y(4,1) + Y(6,1)*Y(3,1)/Y(1,1) - TH*A/SQ
   DY(6) = -2.*Y(5,1)*Y(3,1)/Y(1,1)+Y(6,1)*Y(2,1)/Y(1,1)-TH*B/SQ
 RETURN
 END


 SUBROUTINE ALLBODY (P,MAX,T,Y,DY)
 implicit NONE
 COMMON /MVCOUNT/IFCNT
 INTEGER P,MAX,IFCNT
 DOUBLE PRECISION T, Y(MAX,P),DY(MAX), TH,SQ,a,b,
+                 Mu, delta, tf, rf, ro,Thrust,mo,mdot,
+       MU1,R1,SQR1,COS1,SIN1,D1,D1SQ,W,SINT,COST,
+       DR1R,DR1T,T1,T2
 common  /EXMARS/ Mu, delta, tf, rf, ro,Thrust,mo,mdot
 COMMON  /PLANET/ MU1,D1,W

   IFCNT=IFCNT+1
   IF (MAX.GT.4) THEN
     TH=THRUST/(Mo - Mdot*t)
     a=Y(6,1)
     B=Y(7,1)
     SQ   = SQRT(A*A + B*B)
   ELSE
     TH=0.0
```

```
      A=0.
      B=0.
      SQ = 1.
    ENDIF
    COST = COS(Y(4,1))
    SINT = SIN(Y(4,1))
    COS1 = Y(1,1)-D1*COST
    SIN1 = D1*SINT
    SQR1 = COS1*COS1+SIN1*SIN1
    R1   = SQRT(SQR1)
    D1SQ = D1*D1

    DY(1)= Y(2,1)

    DY(2)=(Y(3,1)*Y(3,1)-MU/Y(1,1))/Y(1,1)
+          - MU1*(COS1/SQR1/R1 + COST/D1SQ)
+          +W*(W*Y(1,1)+2.*Y(3,1)) + TH*A/SQ

    DY(3)=-Y(3,1)*Y(2,1)/Y(1,1)+MU1*SIN1*(-1./SQR1/R1+1./D1SQ/D1)
+         -2.*Y(2,1)*W + TH*B/SQ

    DY(4)=Y(3,1)/Y(1,1)

    IF (MAX.LE.4) RETURN

    DR1R = 2.*COS1
    DR1T = 2.*Y(1,1)*SIN1
    T2   = MU1/(SQR1*SQR1*R1)*1.5
    T1   = T2*COS1

    DY(5)=-Y(6,1)*((-Y(3,1)*Y(3,1)+2.*MU/Y(1,1))/Y(1,1)/Y(1,1)
+         +T1*DR1R + W*W)
+         -Y(7,1)*(Y(2,1)*Y(3,1)/Y(1,1)/Y(1,1)+T2*SIN1*DR1R)

    DY(6)=-Y(5,1) +Y(7,1)*(Y(3,1)/Y(1,1)+2.*W)

    DY(7)=-Y(6,1)*2.*(W+Y(3,1)/Y(1,1)) +Y(7,1)*Y(2,1)/Y(1,1)
+         -Y(8,1)/Y(1,1)

    DY(8)= -DR1T*(Y(6,1)*T1 + Y(7,1)*T2*SIN1)
```

```
     +          -MU1*(-D1/SQR1/R1+1./D1SQ)*(Y(6,1)*SINT+Y(7,1)*COST)

      RETURN
      END


        FUNCTION ThAngle (T)
C       ...RETURNS THRUST ANGLE OF THE THRUSTOR AT
C       ...THE GIVEN TIME. USES HORNERS ALGORITHM.
        IMPLICIT NONE
        INTEGER I,N, IER,N1MAX,FNTYPE,CONTYPE,PARTYPE
        PARAMETER (N1MAX=30)
        DOUBLE PRECISION T,To,TFF, THANGLE, X(N1MAX),Ts,
     +          SCOEF(4,N1MAX),TX(N1MAX),SPEVAL
        COMMON/DIRCOMN/X,TX,SCOEF,To,TFF,N
        COMMON /FNSPECS/ FNTYPE,CONTYPE,PARTYPE
        EXTERNAL SPEVAL
        GOTO (1,2,3,4) PARTYPE
        WRITE(*,*) 'IMPROPER PARTYPE',PARTYPE
        STOP
 1          ThAngle = X(N)
            DO 11 I=N-1,1,-1
               ThAngle = ThAngle*T + X(I)
 11         CONTINUE
        RETURN
 2          Ts = (T-To)/Tff
            ThAngle = X(N)
            DO 10 I=N-1,1,-1
               ThAngle = ThAngle*Ts + X(I)
 10         CONTINUE
        RETURN
 3          ThAngle = X(N)
            DO 12 I=N-1,1,-1
               ThAngle = ThAngle*T/DBLE(I+1) + X(I)
 12         CONTINUE
        RETURN
 4         ThAngle = SPEVAL (SCOEF,TX,N-1,T,IER)
           IF (IER.EQ.0) RETURN
           WRITE(*,*) 'IMPROPER XDATA : SPEVAL ',T,ier,TX(1),TX(N)
           STOP
```

```
      END

      SUBROUTINE INITANG (X,TOO,Tff)
      IMPLICIT NONE
      INTEGER N1,I,IER,NINT,N1MAX,FNTYPE,CONTYPE,PARTYPE,PINDEX
      PARAMETER (N1MAX=30)
      DOUBLE PRECISION X(N1),To,TF,X1(N1MAX),TFF,SCOEF(4,N1MAX),
     +        TX(N1MAX),DT,TOO
      COMMON/DIRCOMN/X1,TX,SCOEF,To,TF,N1
      COMMON /FNSPECS/ FNTYPE,CONTYPE,PARTYPE,PINDEX
      to=tOO
      tf=tff

      DO 10 I=1,N1
         X1(I)=X(I)
10    CONTINUE

      IF (PARTYPE.EQ.4) THEN
        NINT=N1-1
        DT=(TF-TO)/NINT
        TX(1)=TO
        DO 20 I = 2,N1-1
           TX(I) = TX(I-1)+DT
20      CONTINUE
        TX(N1)=TF
        CALL SPLINE (TX,X1,NINT,SCOEF, IER)
        IF (IER.NE.0) THEN
          WRITE(*,*) 'UNSUCCESSFUL SPLINE',IER
          STOP
        ENDIF
      ENDIF
      RETURN
      END

      SUBROUTINE OUT (YMAX,K,N,Y,T,EXIT)
      IMPLICIT NONE
      INTEGER K,N,YMAX,I,SKIP,NSKIP
      DOUBLE PRECISION Y(YMAX,K+1),T,THeta,PI
      LOGICAL EXIT,STORE,MODIFY
      COMMON /COUT/ SKIP,NSKIP,STORE
```

```
      IF (STORE) CALL STOREVAL (K,Y,T)
      IF ((SKIP.GE.NSKIP).OR.(EXIT)) THEN
        SKIP=0
      ELSE
        SKIP=SKIP+1
      ENDIF
    RETURN
  END


  SUBROUTINE OUTP (YMAX,K,N,Y,T,EXIT)
  IMPLICIT NONE
  INTEGER K,N,YMAX,I,SKIP,NSKIP,THTYPE
  DOUBLE PRECISION Y(YMAX,K+1),T,THeta,PI,THETA1,
+      ANGLE,TOLD,H,TEMP
  LOGICAL EXIT,STORE,MODIFY,INIT
  Character*20 file
  COMMON /COUT/ SKIP,NSKIP,STORE
  COMMON /POUT/ THETA,INIT,THTYPE,File
  COMMON /PLOCAL/ ANGLE,TOLD,H,TEMP
      IF (INIT) THEN
        OPEN(UNIT=10,FILE=file,STATUS='UNKNOWN')
        INIT=.FALSE.
        IF ((YMAX.EQ.4).OR.(YMAX.EQ.8)) THEN
          ANGLE=Y(4,1)
        ELSE
          ANGLE=0.0
          TOLD=T
        ENDIF
      ELSE
        H=T-TOLD
        TEMP=0.0
        DO 10 I=1,K+1,2
           TEMP=TEMP+Y(3,I)/DBLE(I)
           IF(I.LT.K+1) TEMP=TEMP-Y(3,I+1)/DBLE(I+1)
10      CONTINUE
        ANGLE=ANGLE+TEMP*H/Y(1,1)
        TOLD=T
      ENDIF
      Pi=4.*ATAN(1.)
```

```
          IF (THTYPE.EQ.1) THEN
            THETA1=THETA
          ELSEIF (THTYPE.EQ.2) THEN
            THETA1=ATAN2((Y(5,1)+Y(2,1)),(Y(6,1)+Y(3,1)))
          ELSEIF (THTYPE.EQ.3) THEN
            THETA1=ATAN2(Y(5,1),Y(6,1))
          ELSEIF (THTYPE.EQ.4) THEN
            THETA1=ATAN2(Y(6,1),Y(7,1))
          ELSE
            WRITE(*,*) 'OUTP ERROR ',THTYPE
            STOP
          ENDIF
          THETA1=THETA1*180/Pi
          IF ((YMAX.EQ.4).OR.(YMAX.EQ.8)) THEN
            WRITE(10,910) T,Y(1,1),Y(2,1),Y(3,1),
     +            Y(4,1),ANGLE,THETA1
          ELSE
            WRITE(10,910) T,Y(1,1),Y(2,1),Y(3,1),ANGLE,THETA1
          ENDIF
910       FORMAT(10(G20.10,1x))

          IF ((SKIP.GE.NSKIP).OR.(EXIT)) THEN
            SKIP=0
          ELSE
            SKIP=SKIP+1
          ENDIF
        RETURN
      END



      SUBROUTINE OUTESC2 (YMAX,K,N,Y,T,EXIT)
      IMPLICIT NONE
      INTEGER K,N,YMAX,I,SKIP,NSKIP
      DOUBLE PRECISION Y(YMAX,K+1),T,THeta,PI,B,EINT,THMAX
      LOGICAL EXIT,STORE,MODIFY
      COMMON /COUT/ SKIP,NSKIP,STORE
      COMMON /MVOUTL/ B,EINT
      COMMON /OUTCLOC/THMAX,MODIFY
          IF (T.LE.0.00001) THEN
```

```
        THMAX=0.0
        MODIFY=.TRUE.
      ENDIF
      IF (STORE) CALL STOREVAL (K,Y,T)
      IF ((SKIP.GE.NSKIP).OR.(EXIT)) THEN
        SKIP=0
        Theta=ATAN((Y(2,1)+Y(5,1))/(Y(6,1)+Y(3,1)))
        THMAX=MAX(THMAX,ABS(THETA))
        IF ((MODIFY).AND.(T.GT.B*0.1)) THEN
          MODIFY=.FALSE.
          IF (THMAX.GT..3) EINT=EINT*10.
        ENDIF
      ELSE
        SKIP=SKIP+1
      ENDIF
    RETURN
END



SUBROUTINE OUTESC1 (YMAX,K,N,Y,T,EXIT)
IMPLICIT NONE
INTEGER K,N,YMAX,I,SKIP,NSKIP
DOUBLE PRECISION Y(YMAX,K+1),T,THeta,PI,B,EINT,THMAX
LOGICAL EXIT,STORE,MODIFY
COMMON /COUT/ SKIP,NSKIP,STORE
COMMON /MVOUTL/ B,EINT
COMMON /OUTCLOC/THMAX,MODIFY
      IF (T.LE.0.00001) THEN
        THMAX=0.0
        MODIFY=.TRUE.
      ENDIF
      IF (STORE) CALL STOREVAL (K,Y,T)
      IF ((SKIP.GE.NSKIP).OR.(EXIT)) THEN
        SKIP=0
        Theta=ATAN(Y(5,1)/Y(6,1))
        THMAX=MAX(THMAX,ABS(THETA))
        IF ((MODIFY).AND.(T.GT.B*0.1)) THEN
          MODIFY=.FALSE.
```

```
            IF (THMAX.GT..3) EINT=EINT*10.
          ENDIF
        ELSE
          SKIP=SKIP+1
        ENDIF
      RETURN
    END
```

Colsys Interface

This module sets up and calls Colsys. Routines for the differential equations, constraints, initial solution and output are also included. The routines for the generalized Newton's method and tolerance scheduling are also present.

```
C*********************************************************************
C  Colsys interface Modules
C*********************************************************************

      SUBROUTINE EXM1(EPS,NCOMP,NREC,NMAX,NINT,K,IPR,INITSOL,
     +                SENSE,START,IFLAG)
c
c  problem Mars transfer - see Lewis : 247, Bryson & Ho : 66-68
c                          & Balakrishnan and Neustdat : p100
c

c     T0,TF,R0,RF : Initial,final times and radii
c     T1,R1, rold, told : store previous values of final time, radius
c     z(1..ncomp): vector storing variables,
c     See input description for further details.
c     eps, epso, epsi, epsmin : Tolerances for heuristic control
c     of tol.
c     m(1..ncomp) : stores derivative orders, 1 in our case
c     delta : stores diff. for derivative evaluation.
c
C
```

```
      implicit None
      integer maxdif,MAXF,MAXI
      parameter (maxdif=20, MAXF=120000, MAXI=6000)
      double precision zeta(maxdif), fspace(MAXF), tol(maxdif),
     +  z(maxdif), mu, x, fixpnt, Tf1,
     +  TO, TF, RF, RO, DELTA,
     +  EPS, FSAVE(MAXF), TSAVE, TOSAVE,
     +  Thrust,mo,mdot, Uo,Vo
      integer m(maxdif), ipar(11), ispace(MAXI), ltol(maxdif),NREC,
     +  mstar, ncomp, iflag, I, KD,KDM, ISAVE(MAXI),NDIMI,NDIMF,
     +  niter, NMAX, NINT, IPR,SENSE,
     +  INITSOL, K
      LOGICAL DONE,DYNAMIC,START
      common /EXMARS/ mu, delta, TF, RF, RO,Thrust,mo,mdot
      COMMON /INITSOL/ FSAVE,ISAVE,TSAVE,TOSAVE,NITER
      COMMON/escape/To,Uo,Vo
      COMMON /LARGE/fspace,ispace
      COMMON /LOCALCS/ ZETA,TOL,LTOL,MSTAR,IPAR,M
      external solutn,fsub,dfsub,gsub,dgsub

C     ...note : No. of Function evaluations slightly increases
C     ...       when delta is decreased to 5.e-13
C
c     NCOMP : no. of differential equations.
c     NREC : no. of right end bc's
c     K no. of collocation points per subinterval
C     nmax (max intervals)


      IF (START) THEN
C        . DEFINED REQUIRED FINAL RADIUS AND INITIAL RADIUS.
         TOSAVE=TO

c        orders
         mstar=0
         DO 10 I=1,NCOMP
            M(I)=1
            MSTAR=MSTAR+M(I)
 10      CONTINUE
```

```
c        a nonlinear problem
         ipar(1) = 1

         ipar(2) = K
c        initial uniform mesh of NINT subintervals, See ipar(3)
         ipar(8) = 0
c        dimension of real work array  fspace  :
         KD=K*NCOMP
         KDM=KD + MSTAR
         NDIMF = NMAX*(4+k+2*kd+(4+2*k)*mstar+(kdm-nrec)*(kdm+1))
         ipar(5) = NDIMF
c        dimension of integer work array  ispace  is CALCULATED.
         NDIMI=NMAX*(3 + KDM - NREC)
         ipar(6) = NDIMI
         IF ((NDIMI.GT.MAXI).OR.(NDIMF.GT.MAXF)) THEN
           WRITE(*,*) 'NDIMF,NDIMI',ndimf,ndimi
           WRITE(*,*) 'ERROR IN NMAX'
           STOP
         ENDIF

c        print (-1)full, (0)LITTLE, (1)NO output.
         ipar(7) = IPR
c        initial approximation for nonlinear iteration is provided
c        in  solutn
         ipar(9) = INITSOL
c        a sensitive problem
         ipar(10) = SENSE
c        no fixed points in the mesh
         ipar(11) = 0

         NITER = 0
         START=.FALSE.
      ENDIF

      ipar(3) = NINT
c         ...locations of side conditions
      zeta(1) = TO
      zeta(2) = TO
      zeta(3) = TO
      zeta(4) = TF
```

```
      zeta(5) = TF
      zeta(6) = TF
c     tolerances on  all components
      ipar(4) = ncomp
      do 20 i=1,ncomp
         ltol(i) = i
         tol(i) = eps
 20   continue

C        ... only place tf1 is used (bec tf is in common)
      Tf1=Tf
c     call  colsys
      call colsys (ncomp, m, TO, TF1, zeta, ipar, ltol,
     .     tol, fixpnt, ispace, fspace, iflag,
     .     fsub,dfsub,gsub,dgsub,SOLUTN)
      NITER=NITER+1
         if (iflag.eq.1) then
C        ...SAVE CURRENT SOLUTION
           DO 30 I=1,NDIMI
              ISAVE(I)=ISPACE(I)
 30        CONTINUE
           DO 40 I=1,NDIMF
              FSAVE(I)=FSPACE(I)
 40        CONTINUE
         endif
         TSAVE=TF
         ToSave=To

      RETURN
      end
c.....................................................................

      SUBROUTINE SOLUTN (T,X,DMVAL)
      IMPLICIT NONE
      INTEGER FNTYPE,CONTYPE
      DOUBLE PRECISION T, X(1), DMVAL(1)
      COMMON /FNSPECS/ FNTYPE,CONTYPE
         GOTO (1,2) FNTYPE
            WRITE(*,*) 'INVALID FN TYPE',FNTYPE
         STOP
```

```
 1          CALL MSOLUTN (T,X,DMVAL)
         RETURN
 2          CALL ESOLUTN (T,X,DMVAL)
         RETURN
      END

      subroutine Msolutn (t,x, dmval)
c     Initial estimate of solution for Mars transfer Problem.
      implicit none
      INTEGER MAXF,MAXI
      PARAMETER (MAXF=120000, MAXI=6000)
      INTEGER ISPACE(MAXI),NITER,IS5,IS6
      double precision x(6), t, dmval(6), Mu, delta, tf,rf,ro,
     +                 FSPACE(MAXF),TSAVE,TOSAVE,T1
C     Not used
     +                 ,Thrust,mo,mdot
      COMMON /INITSOL/ FSPACE,ISPACE,TSAVE,TOSAVE,NITER
      common /EXMARS/ Mu, delta, tf, rf, ro,Thrust,mo,mdot
         IF (NITER.GE.0) THEN
           x(1)=ro + (rf-ro)*t/tf
           dmval(1)=(rf-ro)/tf
           x(2)=0.0
           dmval(2)=0.0
           x(3)=sqrt(mu/x(1))
           dmval(3)=-0.5*sqrt(mu/x(1)**3)*dmval(1)
           x(4)=1.0
           dmval(4)=0.0
           if (t.le.tf/2.0) then
             x(5)=0.52
             x(6)=0.3
           else
             x(5)=-0.5
             x(6)=0.0
           endif
           dmval(5)=0.0
           dmval(6)=0.0
         ELSE
           IS6 = ISPACE(6) + 1
           IS5 = ISPACE(1) + 2
           T1=(T-TOSAVE)/(TF-TOSAVE)*(TSAVE-TOSAVE)
```

```
      CALL approx (ispace(5), T1, X, fspace(is6),fspace,ispace,
     1     fspace(is5), ispace(2), ispace(3), ispace(8),
     2     ispace(4), 1, DMVAL, 1)
    ENDIF

 return
 end



 subroutine Esolutn (t,x, dmval)
c Initial estimate of solution for Mars transfer Problem.
 implicit none
 INTEGER MAXF,MAXI
 PARAMETER (MAXF=120000, MAXI=6000)
 INTEGER ISPACE(MAXI),NITER,IS5,IS6
 double precision x(6), t, dmval(6), Mu, delta, tf,rf,ro,
+                 FSPACE(MAXF),TSAVE,TOSAVE,T1,
+                 To,Uo,Vo
C Not used
+                 ,Thrust,mo,mdot
 COMMON /INITSOL/ FSPACE,ISPACE,TSAVE,TOSAVE,NITER
 common /EXMARS/ Mu, delta, tf, rf, ro,Thrust,mo,mdot
 COMMON/escape/To,Uo,Vo
    IF (NITER.GE.0) THEN
      x(1)=ro + (rf-ro)*t/tf
      dmval(1)=(rf-ro)/tf
      x(2)=0.0
      dmval(2)=0.0
      x(3)=sqrt(mu/x(1))
      dmval(3)=-0.5*sqrt(mu/x(1)**3)*dmval(1)
      x(4)=1.0
      if (t.le.tf/2.0) then
        x(5)=0.52
        x(6)=0.3
      else
        x(5)=-0.5
        x(6)=0.0
      endif
      dmval(4)=0.0
```

```
          dmval(5)=0.0
          dmval(6)=0.0
        ELSE
          IS6 = ISPACE(6) + 1
          IS5 = ISPACE(1) + 2
          T1=T-To + TOSave
          CALL approx (ispace(5), T1, X, fspace(is6),fspace,ispace,
     1    fspace(is5), ispace(2), ispace(3), ispace(8),
     2    ispace(4), 1, DMVAL, 1)
        ENDIF

      return
      end
c.....................................................................
      subroutine fsub (t, x, f)
      implicit NONE
      DOUBLE PRECISION T, X(6), f(6)
      integer ifcnt, igcnt, igdcnt, FNTYPE,CONTYPE
      common /MVcount/ifcnt,igcnt, igdcnt
      COMMON /FNSPECS/ FNTYPE,CONTYPE
      ifcnt=ifcnt+1
        GOTO (1,2) FNTYPE
           WRITE(*,*) 'INVALID FUNCTION TYPE',FNTYPE
        STOP
 1        CALL FMARSIN (T,X,F)
        RETURN
 2        CALL FESC    (T,X,F)
        RETURN
      END

      subroutine fmarsin (t, x, f)
      implicit NONE
      DOUBLE PRECISION T, X(6), f(6), TH, MU, DELTA, SQ, TF, RF, RO,
     +                 THRUST, MO, MDOT
      common /EXMARS/ mu, delta, TF, RF, RO,Thrust,mo,mdot
        TH=THRUST/(Mo - Mdot*t)
        SQ = SQRT(X(5)*X(5) + X(6)*X(6))

        F(1) = X(2)
        F(2) = (X(3)*X(3) - MU/X(1))/X(1) + TH*X(5)/SQ
```

```
      F(3) = -X(2)*X(3)/X(1) + TH*X(6)/SQ

      F(4) = -(X(5)*(-X(3)*X(3)+2.*MU/X(1))
     +       + X(6)*X(2)*X(3))/(X(1)*X(1))
      F(5) = -X(4) + X(6)*X(3)/X(1)
      F(6) = -2.*X(5)*X(3)/X(1) + X(6)*X(2)/X(1)
      RETURN
     END



     subroutine fesc (t, x, f)
     implicit NONE
     DOUBLE PRECISION T, X(6), F(6), TH, MU, DELTA, SQ, TF, RF, RO,
     +                THRUST, MO, MDOT,a,b
     common /EXMARS/ mu, delta, TF, RF, RO,Thrust,mo,mdot

      TH=THRUST/(Mo - Mdot*t)
      a=X(5)+X(2)
      b=X(6)+X(3)
      SQ=SQRT(A*A+B*B)
      F(1) = X(2)
      F(2) = (X(3)*X(3) - MU/X(1))/X(1) + TH*A/SQ
      F(3) = -X(2)*X(3)/X(1) + TH*B/SQ

      F(4) = -(X(5)*(-X(3)*X(3)+2.*MU/X(1))
     +       + X(6)*X(2)*X(3))/(X(1)*X(1))
      F(5) = -X(4) + X(6)*X(3)/X(1) - TH*A/SQ
      F(6) = -2.*X(5)*X(3)/X(1) + X(6)*X(2)/X(1) - TH*B/SQ
     RETURN
     end
c.............................................................
     subroutine dfsub (t, x, df)
     implicit none
     double precision t, x(6), df(6,6), temp, FX(6), F(6), MU,DELTA,
C    ... not used
     +  tf, rf, ro,Thrust,mo,mdot
     INTEGER I,J
     common /exmars/ MU, DELTA, tf, rf, ro,Thrust,mo,mdot
        CALL FSUB (T, X, FX)
```

```
         DO 20 J=1,6
            TEMP=X(J)
            X(J)=X(J)+DELTA
            CALL FSUB (T, X, F)
            DO 10 I=1,6
               DF(I,J) = (F(I)-FX(I))/DELTA
 10         CONTINUE
            X(J)=TEMP
 20      CONTINUE
      return
      end
c.....................................................................

      SUBROUTINE GSUB (I, X, G)
      IMPLICIT NONE
      INTEGER I,FNTYPE,CONTYPE
      DOUBLE PRECISION X(1), G
      COMMON /FNSPECS/ FNTYPE, CONTYPE
         GOTO (1,2,3) CONTYPE
            WRITE(*,*) 'INVALID CONSTRAINT TYPE',CONTYPE
         STOP
 1          CALL MARGSUB (I,X,G)
         RETURN
 2          CALL MARGSB2 (I,X,G)
         RETURN
 3          CALL EARGSUB (I,X,G)
         RETURN
      END

      subroutine MARgsub (I, X, G)
      implicit NONE
      DOUBLE PRECISION X(6), G, MU, DELTA, TF, RF, RO
C     Not used
     +                  ,Thrust,mo,mdot
      integer i, ifcnt,igcnt, igdcnt
      COMMON /MVCOUNT/ifcnt,igcnt, igdcnt
      COMMON /exmars/ MU, DELTA, TF, RF ,RO, Thrust,mo,mdot
      IGCNT=IGCNT+1
      go to (1, 2, 3, 4, 5, 6), i
 1    g = X(1)- RO
```

```
      return
 2    g = X(2)
      return
 3    g = X(3) - SQRT(MU/RO)
      return
 4    G = X(2)
      return
 5    G = X(3) - SQRT(MU/Rf)
      return
 6    G = X(4)- 1. - X(6)*SQRT(MU)/(2*Rf**1.5)
      return
      end


      subroutine MARgsb2 (I, X, G)
      implicit NONE
      DOUBLE PRECISION X(6), G, MU, DELTA, TF, RF, RO
C     Not used
     +                  ,Thrust,mo,mdot
      integer i, ifcnt,igcnt, igdcnt
      COMMON /MVCOUNT/ifcnt,igcnt, igdcnt
      COMMON /exmars/ MU, DELTA, TF, RF ,RO, Thrust,mo,mdot
      IGCNT=IGCNT+1
      go to (1, 2, 3, 4, 5, 6), i
 1    g = X(1)- RO
      return
 2    g = X(2)
      return
 3    g = X(3) - SQRT(MU/RO)
      return
 4    G = X(2)
      return
 5    G = X(3) - SQRT(MU/X(1))
      return
 6    G = X(4)- 1. - X(6)*SQRT(MU)/(2*X(1)**1.5)
      return
      end


      subroutine EARgsub (I, X, G)
```

```
      implicit NONE
      DOUBLE PRECISION X(6), G, MU, DELTA, TF, RF, RO,
     +                 To,Uo,Vo
C     Not used
     +                 ,Thrust,mo,mdot
      integer i, ifcnt,igcnt, igdcnt
      COMMON /MVCOUNT/ifcnt,igcnt, igdcnt
      COMMON /exmars/ MU, DELTA, TF, RF, RO,Thrust,mo,mdot
      COMMON/escape/To,Uo,Vo
      IGCNT=IGCNT+1
      go to (1, 2, 3, 4, 5, 6), i
 1    g = X(1)- RO
      return
 2    g = X(2)- Uo
      return
 3    g = X(3) - Vo
      return
 4    G = X(4) + MU/(X(1)*X(1))
      G = X(4) - MU/(X(1)*X(1))
      return
 5    G = X(5) + X(2)
      G = X(5) - X(2)
      return
 6    G = X(6) + X(3)
      G = X(6) - X(3)
      return
      end
c.............................................................
      subroutine dgsub (i, X, dg)
      implicit NONE
      DOUBLE PRECISION X(6),dg(6), G, TEMP, MU, DELTA, GX, TF, RF, RO
C     Not used
     +   ,Thrust,mo,mdot
      integer i,j , ifcnt,igcnt, igdcnt
      COMMON /MVCOUNT/ifcnt,igcnt, igdcnt
      common /EXMARS/ Mu, delta, tf, rf, ro,Thrust,mo,mdot

      CALL GSUB (I,X,GX)
      do 10 j=1,6
         TEMP=X(J)
```

```
        X(J)=X(J)+DELTA
        CALL GSUB(I,X,G)
        DG(J)=(G-GX)/DELTA
        X(J)=TEMP
 10   CONTINUE
      IGDCNT=IGDCNT+1
      return
      end

c-----------------------------------------------------------------------


       FUNCTION ZERO (X1,Y1,X2,Y2,X3,Y3)
       IMPLICIT NONE
       DOUBLE PRECISION ZERO,X1,Y1,X2,Y2,X3,Y3, DX,A,B,C,
     +                  Y21,X21,Y32,X32,  TMP,S21,S32
       INTRINSIC SIGN,MAX,MIN

           Y21=(Y2-Y1)*1.e2
           Y32=(Y3-Y2)*1.e2
           X32=(X3-X2)*1.e2
           X21=(X2-X1)*1.e2

           TMP = (X21*Y32-X32*Y21)
           IF (ABS(TMP).LE.1.E-8) THEN
             write(*,*) 'straight line inter :'

C            ... SLOPE DY/DX
             DX = Y32/X32
             ZERO = -(Y3-DX*X3)/DX
           ELSE
             S21 = X21*(X2+X1)
             S32 = X32*(X3+X2)
             DX=X21*S32-X32*S21
             B=(Y21*S32-S21*Y32)/DX
             A=TMP/DX
             C=Y3-(B+A*X3)*X3
             ZERO = (-B+SQRT(B*B-4*A*C))*0.5/A
             WRITE(*,*) 'QUADRATIC INTERP'
           ENDIF
```

```
c$$$            WRITE(*,*) X1,Y1,X2,Y2,X3,Y3, A,B,C,ZERO
c$$$ 900        FORMAT(1X,3(G12.6,1X,G12.6,2X),1X,4G12.6)
c$$$            read(*,*)
      return
      END

c-----------------------------------------------------------------------

      SUBROUTINE TADAPT (DX,EPS,NINT,DONE,T2INIT,EPSO,EPSI,EPSMIN)
      IMPLICIT NONE
      INTEGER MAXF,MAXI,MAXDIF
      PARAMETER (MAXDIF=20, MAXF=120000, MAXI=6000)
      INTEGER NITER,NINT,ISPACE(MAXI),ISAVE(MAXI)
      DOUBLE PRECISION DX(3,2),Z(MAXDIF),T2INIT,mu, delta, TF,RF,RO,
     +     Thrust,mo,mdot, TOSAVE,TSAVE,FSPACE(MAXF),FSAVE(MAXF),
     +     EPSO,EPSI,EPSMIN,EPS,ZERO
      LOGICAL DONE
      EXTERNAL ZERO
      common /EXMARS/ mu, delta, TF, RF, RO,Thrust,mo,mdot
      COMMON /INITSOL/ FSAVE,ISAVE,TSAVE,TOSAVE,NITER
      COMMON /LARGE  / FSPACE,ISPACE
           CALL APPSLN (TF,Z,FSPACE,ISPACE)
           write(*,*) 'R,U,V : ',z(1),z(2),z(3)
           DX(3,2)=Z(1)-RF
           IF (ABS(Z(1)-RF).LE.EPSMIN) THEN
              DONE=.TRUE.
           ELSE
              IF (NITER.GT.1) THEN
C                 TF = T1 + (T1-TFOLD)*(RF-R1)/(R1-ROLD)
c                 .. the newton raphson method above has been
C                 .. replaced by quadratic interpolation.

                 TF = ZERO(DX(1,1),DX(1,2),DX(2,1),DX(2,2),
     +                     DX(3,1),DX(3,2))
                 CALL ADAPEPS(EPS,EPSO,EPSI,EPSMIN,Z(1)-Rf)
                 WRITE(*,*) 'NEW TF, EPS =',TF,EPS
c                ... set no. of interv to prev values
                 NINT = min(ispace(1)/2,16)
                 DX(1,1)=DX(2,1)
                 DX(1,2)=DX(2,2)
```

```
            DX(2,1)=DX(3,1)
            DX(2,2)=DX(3,2)
         ELSE
            DX(1,2)=Z(1)-RF
            DX(2,1)=DX(1,1)
            DX(2,2)=DX(1,2)
            TF = T2INIT
            EPS=EPSO
         ENDIF
         DX(3,1)=TF
      ENDIF
RETURN
END


SUBROUTINE ADAPEPS (EPS,EPSO,EPSI,EPSMIN,ERR)
IMPLICIT NONE
DOUBLE PRECISION EPS,EPSO,EPSI,EPSMIN,ERR
    EPS=MAX(EPSI*ABS(ERR),EPSMIN)
    EPS=MIN(EPS,EPSO)
RETURN
END


SUBROUTINE ADAPESC (NINT,DONE,T2INIT,TDELTA)
IMPLICIT NONE
INTEGER MAXF,MAXI,MAXDIF
PARAMETER (MAXDIF=20, MAXF=120000, MAXI=6000)
INTEGER NITER,NINT,ISPACE(MAXI),I
DOUBLE PRECISION DX(3,2),Z(MAXDIF),T2INIT,mu, delta, TF,RF,RO,
+      Thrust,mo,mdot, FSPACE(MAXF),TDELTA,To,Uo,Vo
LOGICAL DONE
COMMON/escape/To,Uo,Vo
common /EXMARS/ mu, delta, TF, RF, RO,Thrust,mo,mdot
COMMON /LARGE  / FSPACE,ISPACE

      IF (TF.GE.T2INIT) then
        DONE=.TRUE.
      ELSE
        CALL APPSLN (TO,Z,FSPACE,ISPACE)
```

```
          WRITE(*,*) 'SUBPROBLEM'
          Write(*,*) To,(Z(i),i=1,6)
          CALL APPSLN (TF,Z,FSPACE,ISPACE)
          Write(*,*) TF,(Z(i),i=1,6)
          Ro=Z(1)
          IF (Rf.LT.Ro) Rf=Ro+Rf

          TO=TF
          TF=TF+TDELTA
          IF (TF.GT.T2init) TF=T2INIT
          Uo=Z(2)
          Vo=Z(3)
        ENDIF

      RETURN
      END



      SUBROUTINE OUTCSYS (NPTS,NDIF)
      IMPLICIT NONE
      INTEGER MAXF,MAXI,MAXDIF,NPTS,NDIF
      PARAMETER (MAXDIF=20, MAXF=120000, MAXI=6000)
      INTEGER ISPACE(MAXI),I,J,ifcnt,igcnt, igdcnt,NITER,
     +     FNTYPE,CONTYPE,is5,is6
      DOUBLE PRECISION FSPACE(MAXF),TFS,TO,X,Z(MAXDIF),PI,dmval(6),
     +     mu, delta, TF, RF, RO,Thrust,mo,mdot,DEL, THETA,ANGLE
      COMMON /INITSOL/ FSPACE,ISPACE,TFS,TO,NITER
      common /EXMARS/ mu, delta, TF, RF, RO,Thrust,mo,mdot
      COMMON /MVCOUNT/ifcnt,igcnt, igdcnt
      COMMON /FNSPECS/ FNTYPE,CONTYPE
c       print values of the obtained approximate solution at points
        x = TO
        del = (TF-TO-1.e-5)/DBLE(NPTS-1)
        Pi = 4.0*ATAN(1.0)
        ANGLE=0.0
        CALL APPSLN (TF,Z,FSPACE,ISPACE)
        write(*,*) 'Energy = ',(Z(2)*Z(2)+z(3)*z(3))/2.0-Mu/z(1)
```

```
      write (6,201)
      do 555 i=1,NPTS
        IS6 = ISPACE(6) + 1
        IS5 = ISPACE(1) + 2
        CALL approx (ispace(5), x, z, fspace(is6), fspace, ispace,
     1  fspace(is5), ispace(2), ispace(3), ispace(8),
     2  ispace(4), 1, DMVAL, 1)
C         call appsln (x,z,fspace,ispace)
         IF (FNTYPE.EQ.2) THEN
           Theta = ATAN2((Z(2)+Z(5)),(Z(6)+Z(3)))
         ELSE
           Theta = ATAN2(Z(5),Z(6))
           IF (Theta.LT.0) THETA=THETA+2.*Pi
         ENDIF
         THETA = THETA*180./Pi
         if (i.gt.1) Angle=Angle+(z(3)-dmval(3)/2)/z(1)*del
         write (6,202) x, (z(j),j=1,6),THETA,Angle
         x = x + del
555   continue
      write(*,900) igcnt, igdcnt,IFCNT
    RETURN
900 format ('no. G, GD, F = ',3(I7))
201 format ('      t              r              u              v    ',
     .     '         Lr                Lu              Lv    ',
     .     '        Theta')
202 format ( F9.4, 1x, 8G15.6)
    END
```

SQP Interface

The main routine sets up and calls the NAG SQP code. It supplies routines for gradient evaluation which are listed below.

```
C*********************************************************************
C            Interface modules for NAG SQP, v.14c
C*********************************************************************

      SUBROUTINE EXM3(N,X,BL,BU,NCNLN,NDIGITS,ITMAX,
     +                ETA,FTOL,NCTOL,IFAIL)
C     E04VCF EXAMPLE PROGRAM TEXT
C     .. Parameters ..
      IMPLICIT NONE
      INTEGER         N, NCLIN, NCNLN, NCTOTL, NROWA, NROWJ, NROWR,
     *                LIWORK, LWORK, NMAX, NCMAX, NDIGITS
      PARAMETER       (NMAX=25,NCMAX=20,NROWA=NCMAX,
     *                  NROWJ=NCMAX,NROWR=NMAX,
     *                LIWORK=3*NMAX+2*NCMAX,LWORK=1000)
      DOUBLE PRECISION ZERO, ONE
      PARAMETER       (ZERO=0.0D0,ONE=1.0D0)
      INTEGER         NIN, NOUT
C     .. Local Scalars ..
      DOUBLE PRECISION EPSAF, EPSMCH, ETA, FTOL, OBJF
      INTEGER         I, IFAIL, ITER, ITMAX, MODE, MSGLVL, NSTATE
      LOGICAL         COLD, FEALIN, ORTHOG
C     .. Local Arrays ..
      DOUBLE PRECISION A(NROWA,NMAX), BL(NCMAX), BU(NCMAX), C(NROWJ),
     *                 CJAC(NROWJ,NMAX), CLAMDA(NCMAX),FEATOL(NCMAX),
     *                 OBJGRD(NMAX), R(NROWR,NMAX),WORK(LWORK),
     *                 X(N),tmpp(20),NCTOL,BIGBND
      INTEGER         ISTATE(NCMAX), IWORK(LIWORK)
      INTEGER N1MAX, IFCNT,IGCNT,IHCNT, TFCNT, N1
      PARAMETER (N1MAX=20)
      DOUBLE PRECISION DELTA,FVAL(0:20)

      COMMON /COUNT  / IFCNT,IGCNT,IHCNT
      COMMON /MVCOUNT/ TFCNT
      COMMON /EOLOCAL/ FVAL,DELTA,tmpp
      COMMON /CINOUT / NIN,NOUT

C     .. External Functions ..
      DOUBLE PRECISION X02AJF
      EXTERNAL        X02AJF
C     .. External Subroutines ..
```

```
      EXTERNAL      OBJFUN, E04VCF, E04ZCF, CONFUN, X04ABF
C     .. Intrinsic Functions ..
      INTRINSIC       ABS, SQRT
C     .. Executable Statements ..

      EPSMCH = 10.**(-NDIGITS)
C     ...Initialize Common
        NCLIN=0
        NCTOTL=N+NCLIN+NCNLN



      DELTA= SQRT(EPSMCH)
C     ...

      WRITE (NOUT,FMT=99999)
C
      CALL X04ABF(1,NOUT)
      BIGBND = BU(N+1)

C     * CHANGE MSGLVL TO A VALUE .GE. 5 TO GET INTERMEDIATE OUTPUT *
      MSGLVL = 20
      DO 20 I = 1, N+NCLIN
         FEATOL(I) = FTOL
 20   CONTINUE

      do 50 I=N+NCLIN+1,NCTOTL
         FEATOL(I) = NCTOL
         BL(I)=0.0
         BU(I)=0.0
 50   CONTINUE

C     SET THE ABSOLUTE PRECISION OF THE OBJECTIVE AT THE STARTING
C     POINT.
      NSTATE = 1
      MODE = 1

      EPSAF = EPSMCH
      COLD = .TRUE.
      FEALIN = .TRUE.
```

```
      ORTHOG = .TRUE.
C
C        SOLVE THE PROBLEM FROM A COLD START.
C
      IFAIL = -1
      CALL E04VCF(ITMAX,MSGLVL,N,NCLIN,NCNLN,NCTOTL,NROWA,NROWJ,
     *     NROWR,BIGBND,EPSAF,ETA,FTOL,A,BL,BU,FEATOL,CONFUN,
     *     OBJFUN,COLD,FEALIN,ORTHOG,X,ISTATE,R,ITER,C,CJAC,
     *     OBJF,OBJGRD,CLAMDA,IWORK,LIWORK,WORK,LWORK,IFAIL)
      IF (IFAIL.EQ.0) THEN
        WRITE(NOUT,*) 'SUCCESSFUL SOLUTION '
        IFAIL=1
      ELSE
          WRITE (NOUT,FMT=99994) IFAIL
          IF (IFAIL.GT.0) IFAIL=IFAIL+1
        END IF
        write(NOUT,*) 'x=',(x(i),i=1,n)
        write(NOUT,*) 'NFN, NO. FN :',ifcnt,tfcnt
      ENDIF
      STOP
C
99999 FORMAT (' MARS TRANSFER VERSION 3 ',/1X)
99996 FORMAT (/' INITIAL  X.',/(1X,7F10.2))
99995 FORMAT (/' E04VCF TERMINATED WITH  IFAIL =',I3)
99994 FORMAT (/' INCORRECT GRADIENTS.  IFAIL =',I3)
      END
C



      SUBROUTINE OBJFUN(MODE,N,X,OBJF,OBJGRD,NSTATE)
      IMPLICIT NONE
      INTEGER           MODE, N, NSTATE, I,j
      DOUBLE PRECISION  OBJF, OBJGRD(N), X(N), FVAL(0:20), DELTA,
     +                  TMP(20),c,CJac(10)
      COMMON /EOLOCAL/ FVAL,DELTA,TMP

c... for direct escape problem
C      CALL CONFUN(MODE,0,N,MAX(1,0),X,C,CJac,NSTATE)
        do 20 i=1,n
```

```
          if (tmp(i).ne.x(i)) then
            write(*,*) 'error,obj ',(x(j),tmp(j),j=1,n)
            stop
          endif
 20     continue
C     .. Executable Statements ..
        OBJF = FVAL(0)
        DO 10 I=1,N
           OBJGRD(I) = (FVAL(I)-OBJF)/DELTA
 10     CONTINUE
      RETURN
      END



      SUBROUTINE CONFUN(MODE,NCNLN,N,NROWJ,X,C,CJAC,NSTATE)

      IMPLICIT NONE
      INTEGER     MODE, N, NCNLN, NROWJ, NSTATE,
     +            I, J
      DOUBLE PRECISION  C(NROWJ), CJAC(NROWJ,N), X(N), FVAL(0:20),
     +                  TMPX, FPLANET,DELTA, TMP(20)
      COMMON /EOLOCAL/ FVAL,DELTA,tmp
      EXTERNAL FESCAPE, FESCAPE2, FMARS, FPLANET

        do 40 i=1,n
           tmp(i)=x(i)
 40     continue
        FVAL(0)=FPLANET(N,X,0)
        DO 10 J=1,NCNLN
           C(J)=FPLANET (N,X,J)
 10     CONTINUE

        DO 20 I=1,N
           TMPX=X(I)
           X(I)=X(I)+DELTA
           FVAL(I)=FPLANET(N,X,0)
           DO 30 J=1,NCNLN
              CJAC(J,I)=(FPLANET(N,X,J)-C(J))/DELTA
```

```
30        CONTINUE
          X(I)=TMPX
20      CONTINUE
      RETURN
      END
```

Penalty Interface

This modules contain the skeleton functions supplied to the minimizer. The main routine, PENSUB calls an implementation of BFGS. PENSUB also optionally calculates the penalty multipliers and(or) accelerates the solution using the Hessian.

```
C*******************************************************************
C                                    *
C     By : Lalitesh Kumar Katragadda *
C
C     To minimize a given function subject to a set of equality and
C     inequality Constraints Using the Penalty Function Method.
C     The library routine UNCMIN is used to solve the penalty
C     subproblems.
C
C     User documentation : To minimize a given function read protocol
C     in FPSKEL, follow examples given and create your routine.
C     Modify 1) PENSUB calling routine in main program,change MCON,N
C            2) Replace the function called IN FUNC by your own Func
C     Assume : That FUNC(N,X,0) will be called before any calls to
C              get the constraints. This gives one flexibility to
C              set constants and optimize constraint evaluation.
C
C     Interface : Some of the keyboard inputs have been suppressed
C                 for convenience, The user can (un)comment them any
C                 time if (repeated)no change is required. To comment
C                 a input, comment both prompt & read statement
C                 & make sure default is defined (or use values given
```

```
C                 by the default routine, see UNCMIN user's guide.
C
C*****************************************************************


C*****************************************************************
C CHANGES : 1) DLT = STEPMX
C           2) TYPSIZ = INITIAL X - Does not work

      FUNCTION FUNC (N,X,I)
      IMPLICIT NONE
      INTEGER N,I,MCON,IFCNT,IGCNT,IHCNT
      DOUBLE PRECISION X(N),SIGMA,FUNC,FPLANET,LAMBDA(10)
      COMMON/PFCOUNT/IFCNT,IGCNT,IHCNT
      COMMON/PENAL1/SIGMA,LAMBDA,MCON
           IF ((I.LT.0).OR.(I.GT.MCON)) THEN
              WRITE(*,*) '**********E R R O R**********'
              WRITE(*,*) '    IMPROPER CONSTRAINT NUMBER, FUNC,',MCON
              STOP
           ENDIF
           FUNC = FPLANET (N,X,I)
      RETURN
      END

      SUBROUTINE FPSKEL (N,X,F)
      IMPLICIT NONE
C
C     Skeleton Routine for formulating a Constrained Minimization
C     problem as a Penalty Problem.
C
C     Expected Name  : FUNC : FUNC(X,N,I), function to be optimized.
C     MCON           : No. of Constraints.
C     SIGMA          : Constraint weight.

      INTEGER IFCNT,IGCNT,IHCNT,MCON,I,N
      DOUBLE PRECISION X(N),F,SIGMA, LAMBDA(10),FUNC, TEMP,C
      COMMON/PFCOUNT/IFCNT,IGCNT,IHCNT
      COMMON/PENAL1/SIGMA,LAMBDA,MCON
C
C     FUNC : Function to be Optimized.
```

```
C                1) If equality constr.  => Value returned.
C                2) If inequu Constr => 0 or Value if not satisfied.
C     Function Value : FUNC(X,N,0), Gives Function value.
C     Constraint     : FUNC(X,N,I), Gives Ith Constraint.
C     MCON           : No. of Constraint equations
C     SIGMA          : Penalty weight.
C
C
      F=0.0
      TEMP= FUNC(N,X,0)
      DO 10 I=1,MCON
         C= FUNC(N,X,I)
         F=F+C**2
         TEMP=TEMP-C*LAMBDA(I)
 10   CONTINUE
      F= TEMP + SIGMA*0.5*F
      IFCNT=IFCNT+1
      RETURN
      END



      SUBROUTINE PENSUB(N,M1,X,FPLS,GPLS,DIGITS,
     +    GRADTL, STEPTL, STEPMX, CNORM,ITER,ITNLIM,START)
      IMPLICIT NONE
C
C     Refer to Uncmin Handout for explanation
C     of variables. Rest explained by input prompts.
C
C     CNORM   : Norm of the constraint vector C
C     SINC    : Factor by Which Sigma is incremented each iteration.
C     N       : No. of variables.
C     M1,MCON : Total No. of Constraints
C
      LOGICAL UPDATE, START
      INTEGER ITNLIM,N,M,I,LINMETH,HESMETH,
     $        IFCNT,IGCNT,IHCNT,METHOD,IEXP,MSG,NDIGIT,ILIM,
     $        IAGFLG,IAHFLG,IPR,ITRMCD, M1,MCON, ITER,
     $        IFCN, ICASE,J, DIGITS, NIN,NOUT,
     $        FNTYPE,CONTYPE,PARTYPE,PINDEX,FMETHOD
```

```
      PARAMETER (M=40)
      INTEGER IPVT(M)
      CHARACTER*15 LINC(3),HESC(0:1),FNAME
      DOUBLE PRECISION  X(N),TYPSIZ(M),XPLS(M),GPLS(N),
     $  A(M,M),WRK(M,8),GRADTL, STEPTL, STEPMX,FSCALE,DLT,DUM,
     $  STP,FPLS,Y1(100),Y2(100),Y3(100) ,SINC,SIGMA,
     $  CONTL, FUNC, CNORM, F, TEMP, TMP2,
     $  RNOISE,RCOND, DELTA,LAMBDA(10)
      EXTERNAL FCN
      COMMON  Y1, Y2, Y3
      COMMON /PFCOUNT/  IFCNT, IGCNT, IHCNT
      COMMON /PENAL1/SIGMA,LAMBDA,MCON
      COMMON /PSUB1/IFCN
      COMMON /LOCALPN/ TYPSIZ,LINMETH,HESMETH, UPDATE
      COMMON /CINOUT / NIN,NOUT
      COMMON /FNSPECS/ FNTYPE,CONTYPE,PARTYPE,PINDEX,FMETHOD
      DATA LINC/'LINE SEARCH','DOG STEP','HOOKSTEP'/
      DATA HESC/'NEWTONS METHOD','BFGS METHOD'/
      EXTERNAL  D1FCN,D2FCN,FPSKEL,FUNC

      MCON=M1
      IF (START.EQ..TRUE.) THEN
        START = .FALSE.
        DO 11 I = 1,N
           TYPSIZ(I) = 0.0
11      CONTINUE
        Write(NOUT,*) 'Give Global Step Strat : 1) Lin Srch. ',
     +        ' 2) Dogleg   3) Hookstep'
        READ (NIN,*) LINMETH

        Write(NOUT,*) 'Give Hessian method : 0) Finite Diff, 1) BFGS'
        READ (NIN,*) HESMETH

        WRITE(NOUT,*)'Sigma will be increased by SINC each iteration'
        Write (NOUT,*) 'Give Starting SIGMA value & SINC : '
        READ (NIN,*)  SIGMA, SINC

        WRITE(NOUT,*) 'Give output file name in quotes : '
        READ (NIN,*) FNAME
        WRITE(NOUT,*) 'ACCELERATE ? (TRUE/FALSE) '
```

```
        READ (NIN,*)  UPDATE
        DO 15 I=1,MCON
           LAMBDA(I)=0.0
 15     CONTINUE

C     Set Update (.true.) => acceleration step used else skippped.
        OPEN(UNIT=1,FILE=FNAME,STATUS='UNKNOWN')
      ENDIF


            WRITE (NOUT, 900)
C     ADJUST INITIAL VALUES OF X
C
            CALL DFAULT(N, X, XPLS, FSCALE, METHOD, IEXP, MSG,
     +           NDIGIT, ILIM, IAGFLG, IAHFLG, IPR, DLT, DUM,
     +           STP, DUM)
            IF( STEPMX .GT. 0.0D0 )  STP = STEPMX
            METHOD = LINMETH
            IAGFLG = 0
            IAHFLG = 0
            IEXP = HESMETH
            ILIM = ITNLIM
C           T1 = SECOND(DUM)
            NDIGIT=DIGITS
            DLT=STEPMX
            CALL OPTIF9(M, N, X, FPSKEL,
     +           D1FCN, D2FCN, TYPSIZ, FSCALE,
     +           METHOD, IEXP, MSG, NDIGIT, ILIM, IAGFLG, IAHFLG,
     +           IPR, DLT, GRADTL, STP, STEPTL,
     +           XPLS, FPLS, GPLS, ITRMCD, A, WRK)
            IF ((ITRMCD.NE.1).AND.(ITRMCD.NE.2)) THEN
               WRITE(*,*)'************ ERROR **********', itrmcd,msg
            ENDIF
            WRITE (NOUT, 901)  IFCNT, IGCNT, IHCNT
            WRITE(1,*)
            WRITE(1,902) LINC(METHOD),
     &           ';',HESC(IEXP),' NEVAL= ',IFCNT
            DO 20 I=1,N
               X(I)=XPLS(I)
               XPLS(I)=0.0
```

```
 20          CONTINUE
C            XPLS IS THE RIGHT HAND SIDE FOR ACCEL. EQUATION
             CNORM=0.0
             F = FUNC(N,X,0)
             WRITE(1,*) '    F = ',F
             DO 10 I=1,MCON
                TEMP = FUNC(N,X,I)
                IF (FMETHOD.EQ.2) THEN
                  LAMBDA(I)=LAMBDA(I)-SIGMA*TEMP
                ELSE
                  LAMBDA(I)=0.0
                ENDIF
                WRITE(1,*) '    C(X) ',I, ':',TEMP
                XPLS(N+I)= TEMP
                CNORM=MAX(CNORM,ABS(TEMP))
 10          CONTINUE
C             CNORM=CNORM**0.5
             ITER=ITER+1
             WRITE (1,903) ITER, SIGMA, CNORM
             WRITE (1,*) (X(I), I=1,N)
             IF (UPDATE.EQ..TRUE.) THEN
C********** START ACCELERATION, XPLS CONTAINS THE RIGHT HAND SIDE.
                RNOISE=1.0E-10
                DO 30 J=1,N
                   delta=sqrt(rnoise)*Max(x(j),1./Typsiz(j))
                   IFCN=0
                   ICASE=1
                   TEMP=X(J)
                   X(J)=X(J)+DELTA
C                  ... INITIALIZE BY CALLING FOR IFCN=0
                   CALL FCN (N,X,FPLS)
C                  Set up -A,-A(t)
                   DO 40 I=1,MCON
                      IFCN=I
                      CALL FCN (N,X,FPLS)
                      TMP2=(FPLS-XPLS(N+I))/DELTA
C                     ...Transpose(A)
                      A(N+I,J)= -TMP2
C                     ...Set up A
                      A(J,N+I)=-TMP2
```

```
40              CONTINUE
                X(J)=TEMP
30           CONTINUE
             IFCN=0
             FPLS=0.0
             DO 110 I=1,MCON
                FPLS=FPLS+XPLS(N+I)**2
110          CONTINUE
             FPLS= F + SIGMA*0.5*FPLS

             DO 120 I=1,N
                Y1(I)=(rnoise**(1./3.))*Max(x(I),1./Typsiz(I))
                TEMP=X(I)
                X(I)=X(I)+Y1(I)
                CALL FPSKEL(n,X,Y2(I))
                X(I)=TEMP
120          CONTINUE
             DO 130 I=1,N
                TEMP=X(I)
                X(I)=X(I)+Y1(I)
                DO 140 J=1,I
                   TMP2=X(J)
                   X(J)=X(J)+Y1(J)
                   CALL FPSKEL(N,X,F)
                   A(I,J)=((FPLS-Y2(I))+(F-Y2(J)))/(Y1(I)+Y1(J))
                   X(J)=TMP2
140             CONTINUE
                X(I)=TEMP
130          CONTINUE

C         lower triang hessian in place
             DO 50 I=1,N
                DO 60 J=I+1,N
                   A(I,J)=A(J,I)
60              CONTINUE
50           CONTINUE


             DO 80 I=N+1,N+MCON
                DO 80 J=N+1,N+MCON
```

```
                    A(I,J)=0.0
 80             CONTINUE
C               HESSIAN IN PLACE
C               SOLVE TO GET UPDATE
                CALL DGECO(A,M,N+MCON,IPVT,RCOND,Y1)
                CALL DGESL(A,M,N+MCON,IPVT,XPLS,0)
                DO 70 I=1,N
                   X(I)=X(I)+XPLS(I)
 70             CONTINUE
                write(NOUT,*) 'SOLVER FINISHED'
                CNORM=0.0
                F = FUNC(N,X,0)
                DO 90 I=1,MCON
                   TEMP = FUNC(N,X,I)
                   CNORM=CNORM+TEMP*TEMP
 90             CONTINUE
                WRITE(1,*) 'UPDATE, F= ',F,' ;F*= ',F+0.5*CNORM*SIGMA
                WRITE(1,*) (X(I), I=1,N)
                cnorm=0.0
                do 91 i=1,mcon
                   temp=func(n,x,i)
                   cnorm=max(cnorm,abs(temp))
 91             continue
C                CNORM=CNORM**0.5
                WRITE (1,*) 'CNORM = ',CNORM
                WRITE(1,*)
            ENDIF
            SIGMA=SIGMA*SINC

      RETURN
 900  FORMAT(1H1)
 901  FORMAT(23H OPTEST    #FCN  EVAL = , I10/
     +       23H OPTEST    GRAD  EVAL = , I10/
     +       23H OPTEST    HESN  EVAL = , I10)
C    +       23H OPTEST     EXEC TIME = , 1P, D12.4, 4H SEC )
 902  FORMAT (1X,A11,A1,A15,A8,I4)
 903  FORMAT (1X,I3,'th Iter, Sigma = ',F11.2,
     +                '; Norm(C) = ',f16.9)
 904  FORMAT (5(1X,F15.8))
```

```
      END

      SUBROUTINE D1FCN
      END

      SUBROUTINE D2FCN
      END

      SUBROUTINE FCN (N,X,F)
      IMPLICIT NONE
      INTEGER N,I
      DOUBLE PRECISION X(N),F,FUNC
      COMMON /PSUB1/I
      IF (I.EQ.0) THEN
         CALL FPSKEL(N,X,F)
      ELSE
         F = FUNC(N,X,I)
      ENDIF
      RETURN
      END
```

FORTRAN Interface for the GA

This routine sets up calls to the genetic algorithm written in C. This routine can also determine the average performance over a specified number of GA runs and output the history.

```
C*********************************************************************
C     FORTRAN INTERFACE FOR GENETIC ALGORITHM
C*********************************************************************

      SUBROUTINE FSGA(X,BOUNDS,OPFITS,POPSIZ, NELITE, MAXGEN,
     +     NEVAL,PCROSS, PMUT, RANDSEED, NODUP,
     +     SCALE, SCMAX, SCMIN, FILENAME,NRUN,MCON1)
```

```
      IMPLICIT NONE
      INTEGER IFCNT, IGCNT, IHCNT, TFCNT,NEVAL,STATSON, NODUP,
     +     MAXN, MAXOPS, I, MCON,MCON1
      INTEGER POPSIZ, MAXGEN, NELITE, SCALE, NRUN, TOTALF, TOTALIF
      PARAMETER (MAXN=40, MAXOPS=20)
      DOUBLE PRECISION PCROSS, PMUT, RANDSEED, SCMAX, SCMIN,
     +     HISTORY(0:5000),HISTORY2(0:5000),SIGMA,LAMBDA(10),
     +     BOUNDS(0:3*MAXN),OPFITS(0:3*MAXOPS),GPLANET,X(1)

      CHARACTER*20 FILENAME
      COMMON/MVCOUNT/TFCNT
      COMMON/COUNT/IFCNT,IGCNT,IHCNT
      COMMON/PENAL1/SIGMA,LAMBDA,MCON
      EXTERNAL GPLANET

      MCON=MCON1
      DO 10 I=1,NRUN
         HISTORY(0)=FLOAT(I)
         HISTORY2(0)=0.0
         IF (I.GE.NRUN) THEN
           STATSON=2
         ELSE
           STATSON=1
         ENDIF
         CALL SGA (GPLANET,X,BOUNDS,OPFITS,
     +         POPSIZ, NELITE, MAXGEN, NEVAL,
     +         PCROSS, PMUT, RANDSEED,
     +         NODUP, SCALE, SCMAX, SCMIN, STATSON,
     +         HISTORY, HISTORY2, FILENAME)
         RANDSEED=RANDSEED+1.E-5*2.**10
         TOTALF=TOTALF+TFCNT
         TOTALIF=TOTALIF+IFCNT
10    CONTINUE
      Write(*,*) 'Runs completed, avg performance TFN, IFN:',
     +            FLOAT(TOTALF)/NRUN,FLOAT(TOTALIF)/NRUN
      RETURN
      END
```

```
      SUBROUTINE GAPLANT (F,X,N)
      IMPLICIT NONE
      INTEGER N,I,IFCNT,TFCNT,MCON
      DOUBLE PRECISION X(N), F, FPLANET,sigma,lambda(10)
      COMMON/COUNT/IFCNT
      COMMON/MVCOUNT/TFCNT
      EXTERNAL FPLANET
      COMMON/PENAL1/SIGMA,LAMBDA,MCON
           F=-FPLANET(N,X,0)
           do 10 I=1,MCON
              F=F-ABS(FPLANET(N,X,I))
 10        CONTINUE
C           WRITE(*,*) 'EARTH MOON--->  ',F,' ',IFCNT,' ',TFCNT
      RETURN
      END
```

Integration Module

This is an implementation of Adams multivalue method described by Gear [10]. This implementation can also solve for higher order equations. A set of routines to store and efficiently interpolate the solution history are also included. A sample output routine explains the calling sequences. The output routine supplied to MULTIVAL is called at each successful step.

```
C-----------------------------------------------------------------
C                     NUMERICAL INTEGRATION MODULE
C-----------------------------------------------------------------

       SUBROUTINE GETMVAL(K,P,L)
       IMPLICIT NONE
       INTEGER K,P,I,J
       DOUBLE PRECISION SL(3,7,7+4),L(K+4)
```

```
      LOGICAL START
      COMMON /MVLOCAL/SL,START

C     K ORDER OF THE METHOD =>K+1 COEFFS
C     CODE THE DERIVATIVE ORDER OF THE METHOD
C     P=1=> F=Y', 2=>F=Y'' ... F=Y^(P)
C     IF START IS TRUE L IS NOT INITIALIZED, SL IS SET UP.

          IF (START.EQ..TRUE.) THEN
C             ...INITIALIZE IF FIRST TIME.
                SL(1,1,1)=1.
                SL(1,2,1)=0.5
                SL(1,2,3)=0.5
                SL(1,3,1)=5./12.
                SL(1,3,3)=3./4.
                SL(1,3,4)=1./6.
                SL(1,4,1)=3./8.
                SL(1,4,3)=11./12.
                SL(1,4,4)=1./3.
                SL(1,4,5)=1./24.
                SL(1,5,1)=251./720.
                SL(1,5,3)=25./24.
                SL(1,5,4)=35./72.
                SL(1,5,5)=5./48.
                SL(1,5,6)=1/120.
                SL(1,6,1)=95./288.
                SL(1,6,3)=137./120.
                SL(1,6,4)=5./8.
                SL(1,6,5)=17./96.
                SL(1,6,6)=1./40.
                SL(1,6,7)=1./720.
                SL(1,7,1)=19087./60480.
                SL(1,7,3)=49./40.
                SL(1,7,4)=203./270.
                SL(1,7,5)=49./192.
                SL(1,7,6)=7./144.
                SL(1,7,7)=7./1440.
                SL(1,7,8)=1./5040.
C verify below values.
                SL(2,2,1)=1.0
```

```
      SL(2,2,2)=1.0
      SL(2,3,1)=1.0007
      SL(2,3,2)=5./6.
      SL(2,3,4)=1./3.
      SL(2,4,1)=1./6.
      SL(2,4,2)=3./4.
      SL(2,4,4)=.5
      SL(2,4,5)=1./12.
      SL(2,5,1)=19./20.
      SL(2,5,2)=251./360.
      SL(2,5,4)=11./18.
      SL(2,5,5)=1./6.
      SL(2,5,6)=1./60.
      SL(2,6,1)=3./20.
      SL(2,6,2)=665./1008.
      SL(2,6,4)=25./36.
      SL(2,6,5)=35./144.
      SL(2,6,6)=1./24.
      SL(2,6,7)=1./360.
      SL(2,7,1)=863./6048.
      SL(2,7,2)=19087./30240.
      SL(2,7,4)=137./180.
      SL(2,7,5)=5./16.
      SL(2,7,6)=17./240.
      SL(2,7,7)=1./120.
      SL(2,7,8)=1./2520.

      SL(3,4,1)=.25
      SL(3,4,2)=.5
      SL(3,4,3)=1.25
      SL(3,4,5)=.25
      SL(3,5,1)=3./80.
      SL(3,5,2)=19./40.
      SL(3,5,3)=9./8.
      SL(3,5,5)=3./8.
      SL(3,5,6)=1./20.
C     ...INITIALIZE ERROR CONSTANTS
      SL(1,1,3)=2.0
      SL(1,1,4)=12.
      SL(1,1,5)=1.
```

```
                SL(1,2,4)=12.
                SL(1,2,5)=24.
                SL(1,2,6)=1.
                SL(1,3,5)=24.
                SL(1,3,6)=37.89
                SL(1,3,7)=2.
                SL(1,4,6)=37.89
                SL(1,4,7)=53.333
                SL(1,4,8)=1.
                SL(1,5,7)=53.333
                SL(1,5,8)=70.08
                SL(1,5,9)=0.3157
                SL(1,6,8)=70.08
                SL(1,6,9)=87.97
                SL(1,6,10)=0.07407
                SL(1,7,9)=87.97
                SL(1,7,10)=1.
                SL(1,7,11)=0.0139

C VERFIY BELOW VALUES.
                SL(2,1,3)=1.
                SL(2,1,4)=12.
                SL(2,1,5)=1.
                SL(2,2,4)=12.
C               2,2,5 IS 1/0.0, 6000 TAKEN AS KLUDGE.
                SL(2,2,5)=6000.
                SL(2,2,6)=0.5
                SL(2,3,5)=6000.
                SL(2,3,6)=240.
                SL(2,3,7)=2.0
                SL(2,4,6)=240.
                SL(2,4,7)=240.
                SL(2,4,8)=250.
                SL(2,5,7)=240.
                SL(2,5,8)=273.66516
                SL(2,5,9)=2.0
                SL(2,6,8)=273.66516
                SL(2,6,9)=318.31579
                SL(2,6,10)=0.333333
                SL(2,7,9)=318.31579
```

```
                SL(2,7,10)=369.1932
                SL(2,7,11)=0.0542986



                DO 10 I=1,3
                   DO 10 J=1,7
 10                   SL(I,J,I+1)=1.0
                START=.FALSE.
                RETURN
            ENDIF

C         ... DO INPUT ERROR CHECK
            IF (K.LT.P) THEN
C              ADD CONSTS FOR K=P+1,CHECK CONSTS, REFER GEARS COMMENT
               WRITE(*,*) 'NO. OF VALUES (K) INSUFFECIENT K>=P'
               STOP
            ELSEIF ((P.LT.1).OR.(P.GT.3)) THEN
               WRITE(*,*) 'UNAVAILABLE DERIVATIVE ORDER IN MULTIVAL'
               STOP
            ELSE IF ((((P.EQ.1).OR.(P.EQ.2)).AND.(K.GT.7)).OR.
     +                ((P.EQ.3).AND.(K.GT.5))) THEN
               WRITE(*,*) 'UNAVAILABLE ORDER IN MULTIVAL'
               STOP
            ENDIF

C         ...INITIALIZE ARRAY L AND ERROR CONSTANTS 1..K+1 CORRECTORS
C             K+2..K+4 : ERROR EVALUATION COEFFS.
            DO 20 I=1,K+4
               L(I)=SL(P,K,I)
 20         CONTINUE

       RETURN
       END


       SUBROUTINE MVAL (N,MAX1,Y,T,B,H,HMAX,HMIN,EPS,K,P,F,OUT,FAIL)

C*********************************************************************
```

```
C       AUTHOR : LALITESH KUMAR KATRAGADDA.      DATE : JUNE 11, 91  *
C                                                                    *
C       ROUTINE TO SOLVE A  SET OF  N ORDINARY DIFFERENTIAL EQUATIONS*
C       OF PTH ORDER, WITHIN AN ACCURACY OF EPS, USING VARIABLE STEP,*
C       ORDER MULTIVALUE METHOD.                                     *
C       RECOMMENDED INITIAL ORDER=P, SIZE H:SMALL(EPS)               *
C*********************************************************************
C       N   : NO OF Y'S, NC : NO OF CORRECTION STEPS                 *
C       T,B : INITIAL AND FINAL TIME                                 *
C       H,HMAX,HMIN : INITIAL, MAX AND MIN STEP SIZES.               *
C       K   : STARTING ORDER, THAT MANY DERIVATIVES REQUIRED.        *
C       MAX1: FIRST DIM FOR Y, GIVES MAX VECTOR SIZE.                *
C       P   : DERIVATIVE ORDER, >= 1                                 *
C       NC  : NO. OF CORRECTORS.                                     *
C       L   : CORRECTOR COEFF ARRAY(INCLUDEs ERROR EVALUATION COEFFS)*
C       F   : FUNCTION, GIVES USER SUPPLIED DERIVATIVES.             *
C       OUT : OUTPUT ROUTINE CALLED AT EACH STEP.                    *
C       IMAX: LOCAL MAX, >= DIMENSIONS OF MAX AND SECOND Y DIMENSION.*
C       DMAX: MAX DERIVATIVE ORDER, MAX VALUE OF P POSSIBLE.         *
C       DYP : LOCAL 1-D ARRAY TO STORE PREDICTOR PTH DERIVS.         *
C       DY  : LOCAL 1-D ARRAY TO STORE CURRENT DERIVATIVES.          *
C       YMAX: 1-D ARRAY OF SCALING FACTORS FOR ERROR CHECKS.         *
C       EPS : APPROX ERROR OVER THE WHOLE INTERVAL.                  *
C       ERR : MAX LOCAL TRUNCATION ERROR PERMITTED= EPS*H/(B-To)     *
C       ER, ERUP, ERDN : CEOFFECIENTS FOR STEP SIZE ESTIMATION       *
C       EINT: REL INTERVAL ERROR = EPS/(B-To)                        *
C       EMAX: MAX LOCAL ERROR.                                       *
C       ALPHA : FACTOR BY WHICH H IS DIVIDED.                        *
C       FAIL  : ERROR CODE. O IF SUCCESSFUL.                         *
C*********************************************************************

C       TO DO :
C               ASSOCIATE ZERO WITH MEPS.
C               ASSOCIATE OTHER CONSTANTS HMIN, CONVERGENCE WITH MEPS


      IMPLICIT NONE
      INTEGER N,K,P,I,J,M,NC,MAX1,FACTORIAL,IMAX,NMAX,
     +        NSTEP,KMAX, FAIL,KNEW,NFAIL
      PARAMETER (NMAX=40)
```

```
      PARAMETER (IMAX=15)
      PARAMETER (KMAX=7)
      DOUBLE PRECISION Y(MAX1,KMAX+1),L(IMAX),T,B,H,HMAX,HMIN,EPS,
     +     DYP(NMAX),DY(NMAX),G(NMAX),FACP, STOREL(3,7,7+4),
     +     ERR, ER,ERUP,ERDN, BND, ALPHA, YMAX(NMAX), EINT, EMAX,
     +     TALPHA,ALMIN,AL1, NK,NKDN,NKUP, HNEW,OLDDYP(NMAX),
     +     YOLD(NMAX,IMAX),TOLD,TF
      PARAMETER (ALMIN=1.E-2)
      LOGICAL START, CNVRG, EXIT
      COMMON /MVLOCAL/STOREL,START
      COMMON /MVOUTL/TF,ERR
      EXTERNAL OUT,F

      TF=B
      IF (B-T.LT.0.0) then
        WRITE(*,*) 'INValid time input'
        FAIL = -6
        return
      endif
      START=.TRUE.
      CALL GETMVAL (K,P,L)
      DO 10 I=1,N
 10      YMAX(I)=1.
C      EINT=EPS/(B-T)
      EINT=EPS
      ALPHA=1.0
      KNEW=K
      K=K-1
      ERR=EINT
C      ERR=EINT*H
      NSTEP=KNEW+1
      EXIT = .FALSE.

C     ... INITIALIZATION COMPLETE, BEGIN MAIN LOOP.

          DO WHILE (1.LT.2)

               FACP=H**P/FLOAT(FACTORIAL(P))
               IF (KNEW.NE.K) THEN
                  K=KNEW
```

```
            CALL GETMVAL(K,P,L)
            ER   = (L(K+2)*ERR)
            ERUP = (L(K+3)*ERR)
            ERDN = (L(K+4)*ERR)
            NK  = 1./FLOAT(K+1)
            NKUP= 1./FLOAT(K+2)
            NKDN= 1./FLOAT(K)
            BND  = ERR*.5*NKUP/FLOAT(N)
            IF (ERDN.EQ.0) THEN
               WRITE(*,*) 'ERROR OR STEP SIZE  TOO SMALL'
               WRITE(*,*) 'k,p,err,l(k+4) = ',k,p,err,l(k+4)
               FAIL=-5
               RETURN
            ENDIF
         ENDIF

         FAIL=0

         DO WHILE ((NSTEP.GT.0).AND.(FAIL.GE.0))
            NSTEP=NSTEP-1
            TOLD=T
            T=T+H
            DO 170 I=1,N
               DO 170 J=1,K+1
170               YOLD(I,J)=Y(I,J)

C        ...GET PREDICTOR BY PASCAL'S TRIANGLE
            DO 30 J=2,K+1
               DO 40 M=K,J-1,-1
                  DO 20 I=1,N
                     Y(I,M)=Y(I,M)+Y(I,M+1)
 20               CONTINUE
 40            CONTINUE
 30         CONTINUE
C        ...

C        ...SAVE PREDICTOR DERVIATIVES FOR ERROR EVALUATION.
            DO 70 I=1,N
               OLDDYP(I)=DYP(I)
               DYP(I)=Y(I,P+1)
```

```
 70                 CONTINUE

                    CNVRG=.FALSE.
                    NC=0
                    DO WHILE ((CNVRG.EQ..FALSE.).AND.(NC.LT.3))
                       CALL F(P,MAX1,T,Y,DY)
C                   ...DERIVATIVES EVALUATED, APPLY CORRECTOR.
                       CNVRG=.TRUE.
                       DO 80 I=1,N
                          DY(I)=FACP*DY(I)
                          G(I) =DY(I)-Y(I,P+1)
                          CNVRG= ((CNVRG.EQ..TRUE.).AND.
     +                         (ABS(G(I)).LE.(BND*YMAX(I))))
                          DO 60 J=1,P
C                            .G IS CORRECTOR REFERRED IN GEAR
 60                          Y(I,J)=Y(I,J)+G(I)*L(J)
                          Y(I,P+1)=DY(I)
 80                    CONTINUE
                       NC=NC+1
                    END DO
                    IF (CNVRG.EQ..FALSE.) THEN
c                        write(*,*) 'convergence fails'
                        ALPHA=4.0
                        AL1=0.25
                        FAIL=-2
                    ELSE

C                   ...COMPLETE CORRECTION
                       EMAX=0.0
                       DO 90 I=1,N
                          DYP(I)=Y(I,P+1)-DYP(I)
                          DO 100 J=P+2,k+1
 100                         Y(I,J)=Y(I,J)+DYP(I)*L(J)
                          EMAX=MAX(EMAX,ABS(DYP(I)/YMAX(I)))
 90                    CONTINUE

                       IF (EMAX.GT.ER) THEN
                          FAIL=-1
c                          write(*,*) 'step fails'
                       ELSE
```

```
            FAIL=0
            NFAIL=0
            IF (B-T.LE.EPS/1000.) then
              EXIT = .TRUE.
            ENDIF
            CALL OUT(MAX1,K,N,Y,T,EXIT)
            IF (EXIT.EQ..TRUE.) RETURN
         ENDIF
      ENDIF
   END DO

   IF (FAIL.LT.0) THEN
      NFAIL=NFAIL+1
      IF (H.LT.HMIN*1.0001) THEN
         WRITE(*,*) 'FATAL ERROR IN MULTIVAL'
         IF (FAIL.EQ.-1) THEN
            WRITE(*,*)'T,H=',T,H,
     +                'DIVERGENCE, EPS TOO SMALL'
         ELSE
            WRITE(*,*) 'ERROR - CORRECTOR INCONVERGANT'
         ENDIF
         RETURN
      ENDIF
      DO 180 I=1,N
         DO 180 J=1,K+1
180         Y(I,J)=YOLD(I,J)
      T=TOLD
   ENDIF



   KNEW=K
   IF (FAIL.GE.-1) THEN
      ALPHA= 1.2*(EMAX/ER)**NK
      AL1=1./ALPHA
      IF ((K.LT.KMAX).AND.(FAIL.EQ.0)) THEN
         EMAX=0.0
         DO 120 I=1,N
120      EMAX=MAX(EMAX,ABS((DYP(I)-OLDDYP(I))/YMAX(I)))
         TALPHA=1.4*(EMAX/ERUP)**NKUP
```

```
            IF (TALPHA.LT.ALPHA) THEN
               KNEW=K+1
               ALPHA=TALPHA
            ENDIF
         ENDIF
         IF (FAIL.EQ.-1) THEN
            DO 190 I=1,N
               DYP(I)=OLDDYP(I)
190         CONTINUE
         ENDIF

         IF (K.GT.P) THEN
            EMAX=0.0
            DO 130 I=1,N
130            EMAX= MAX(EMAX,ABS(Y(I,K+1)/YMAX(I)))
            TALPHA=1.3*(EMAX/ERDN)**NKDN
            IF (TALPHA.LT.ALPHA) THEN
               KNEW=K-1
               ALPHA=TALPHA
            ENDIF
         ENDIF
      ENDIF

      IF ((NFAIL.GT.2).AND.(FAIL.EQ.-1)) THEN
C          WRITE(*,*) 'MVAL,NF,K,KN,AL',NFAIL,K,KNEW,ALPHA
         ALPHA=MAX(ALPHA,2.0D0)
         AL1=0.25
      ENDIF

      ALPHA = MAX(ALPHA, ALMIN)
      ALPHA=1./ALPHA
      HNEW=H*ALPHA

      IF (HNEW.GT.HMAX) THEN
         HNEW=HMAX
         ALPHA=HNEW/H
      ENDIF

      IF (HNEW.LT.HMIN) then
        HNEW=HMIN
```

```
                ALPHA=HNEW/H
              ENDIF


              IF (((ALPHA-1.).GT.0.1).OR.
     +                ((ALPHA.LT.1.0).AND.(AL1.LT.0.95))) THEN

              ELSE
                ALPHA=1.0
                KNEW=K
                HNEW=H
              ENDIF

              IF (hnew*(knew+1)+t.gt.b) THEN
                HNEW = (B-T)/float(KNEW+1)
                ALPHA = HNEW/H
              ENDIF

              IF (ALPHA.NE.1.0) THEN
                H=HNEW
                TALPHA=ALPHA
                DO 150 J=2,K+1
                   DO 160 I=1,N
160                   Y(I,J)=Y(I,J)*TALPHA
                   TALPHA=TALPHA*ALPHA
150             CONTINUE
              ENDIF
              NSTEP=KNEW+1

              IF (KNEW.GT.K) THEN
                 DO 140 I=1,N
                    Y(I,KNEW+1)= DYP(I)*L(K+1)/FLOAT(K+1)*TALPHA
140              CONTINUE
              ENDIF

          END DO
      RETURN
      END

      FUNCTION FACTORIAL(P)
```

```
      IMPLICIT NONE
      INTEGER FACTORIAL,TEMP,P
      TEMP=P
      FACTORIAL=1
      DO 10 TEMP = 2,P,1
         FACTORIAL=FACTORIAL*TEMP
10    CONTINUE
      RETURN
     END

C***********SAMPLE OUTPUT ROUTINE **********************************
c$$$
c$$$       SUBROUTINE OUTDAT (MAX,K,N,Y,T,EXIT)
c$$$       IMPLICIT NONE
c$$$       INTEGER SKIP,NSKIP,U1,U2,N,I,K,MAX,TSTEP
c$$$       DOUBLE PRECISION Y(MAX,K+1),T
c$$$       LOGICAL EXIT
c$$$       COMMON /OUTC/SKIP,NSKIP,U1,U2,TSTEP
c$$$
c$$$       TSTEP=TSTEP+1
c$$$C       CALL STOREVAL(MAX,K,N,Y,T)
c$$$       IF ((SKIP.GE.NSKIP).OR.(EXIT)) THEN
c$$$          SKIP=1
c$$$          WRITE(U1,10) T,(Y(I,1),I=1,N)
c$$$          WRITE(U2,10) T,(Y(I,1),I=1,N)
c$$$10        FORMAT (5X,G13.6,3X,100(G17.9,2X))
c$$$       ELSE
c$$$          SKIP=SKIP+1
c$$$       ENDIF
c$$$       RETURN
c$$$       END
c$$$


C-----------------------------------------------------------------
C         MODULES FOR STORING, RETRIEVING INTERPOLATIOn DATA
C-----------------------------------------------------------------
      SUBROUTINE STOREINIT (NN,MAXY,CODE)
      IMPLICIT NONE
      INTEGER XMAX,TMAX, N,MAX,NN,MAXY, NSTEP, CODE
```

```
      PARAMETER (XMAX=40000)
      PARAMETER (TMAX=5000)
      INTEGER TI(0:TMAX,3), TEMP
      DOUBLE PRECISION X(XMAX)
      COMMON /HISTORY/ X,TI,NSTEP,N,MAX,TEMP

C      XMAX    : MAX SIZE OF VECTOR= N*TMAX*(AVGK+1)
C      TMAX    : MAX NO. OF TIME STEPS ANTICIPATED
C      X       : 1-D ARRAY STORING THE TRAJECTORY HISTORY
C      TI      : ARRAY CONTAINING TIME HISTORY
C
       IF (CODE.EQ.0) THEN
         N=NN
         MAX=MAXY
         NSTEP=0
         TI(0,1)=0
         TI(0,2)=0
         TI(0,3)=1
         X(1)=-1.D20
         TI(1,3)=2
       ELSE
         TEMP=1
         X(TI(NSTEP+1,3))=X(TI(NSTEP,3))+1.D20
       ENDIF
       RETURN
      END

      SUBROUTINE STOREVAL (K,Y,T)
      IMPLICIT NONE
      INTEGER MAX,K,N,XMAX,TMAX, NSTEP, INDEX, I,J
      PARAMETER (XMAX=40000)
      PARAMETER (TMAX=5000)
      INTEGER TI(0:TMAX,3),TEMP
      DOUBLE PRECISION Y(MAX,K+1),T, X(XMAX)
      COMMON /HISTORY/ X,TI,NSTEP,N,MAX,TEMP

C      XMAX    : MAX SIZE OF VECTOR= N*MAXT*(AVGK+1)
C      TMAX    : MAX NO. OF TIME STEPS ANTICIPATED
C      X       : 1-D ARRAY STORING THE TRAJECTORY HISTORY
C      TI      : ARRAY CONTAINING DATA ON ACCESSING X
```

```
C         (I,1) : K (order), (I,2) : no. of values for Ith step
C         (I,3) : Starting Index for ith step in X()
C         ..................................................

          NSTEP=NSTEP+1
          TI(NSTEP,1)=K
          INDEX=TI(NSTEP,3)
          TEMP=INDEX
          X(INDEX)=T
          INDEX=INDEX+1

          DO 10 J=1,K+1
             DO 10 I=1,N
                X(INDEX)=Y(I,J)
                INDEX=INDEX+1
 10       CONTINUE
c         ...set index for start of next step.
          TI(NSTEP+1,3)=INDEX
          TI(NSTEP,2)  =INDEX-TEMP
       RETURN
       END

       SUBROUTINE GETXVAL (T,Y,ND)
       IMPLICIT NONE
       INTEGER MAX,N, ND,XMAX,TMAX, TEMP, NSTEP, INDEX, I,J,K
       PARAMETER (XMAX=40000)
       PARAMETER (TMAX=5000)
       INTEGER TI(0:TMAX,3)
       DOUBLE PRECISION Y(MAX,ND),T, X(XMAX), DT, ALPHA
       COMMON /HISTORY/ X,TI,NSTEP,N,MAX,TEMP

           IF (T.GE.X(TI(TEMP,3))) THEN
             DO WHILE (T.GE.X(TI(TEMP+1,3)))
                TEMP=TEMP+1
             END DO
           ELSE
             DO WHILE (T.LT.X(TI(TEMP,3)))
                TEMP=TEMP-1
             END DO
           ENDIF
```

```
      INDEX=TI(TEMP,3)
      K=TI(TEMP,1)
      DT=T-X(INDEX)
      IF (DT.LT.0) THEN
C     ... THIS IF BLOCK to be elimnated after full testing.
         WRITE(*,*)'ERROR-IMPROPER VALUE OF TEMP OR T IN GETXVAL'
     +                  ,index
      ELSEIF (DT.LT.1.E-11) THEN
         DO 10 J=1,ND
            DO 10 I=1,N
               INDEX=INDEX+1
               Y(I,J)=X(INDEX)
 10      CONTINUE
      ELSE

         IF ((TEMP.LT.1).OR.(TEMP.GE.NSTEP)) THEN
           WRITE(*,*) 'GETXVAL ERROR  ',T,TEMP,
     +                  X(1),X(TI(NSTEP,3)),NSTEP
           IF (TEMP.EQ.NSTEP) THEN
             WRITE(*,*) 'GETXVAL WARNING, RANGE EXCEEDED, '
           ELSE
             STOP
           ENDIF
         ENDIF

         ALPHA=DT/(X(TI(TEMP+1,3))-X(INDEX))
         INDEX=TI(TEMP+1,3)
         DO 40 I=N,1,-1
            INDEX=INDEX-1
            Y(I,1)=X(INDEX)
 40      CONTINUE

         DO 30 J=K,1,-1
            DO 20 I=N,1,-1
               INDEX=INDEX-1
               Y(I,1)=X(INDEX)+Y(I,1)*ALPHA
 20         CONTINUE
 30      CONTINUE
```

```
   ENDIF
RETURN
END
```

The Genetic Algorithm

This algorithm implements the genetic algorithm. Each of the modules is commented and separated. The comments can be better understood by referring to Goldberg [13]. The data structure is more comprehensive and all parameters and variants are accessible through input. The specified include files are listed below alongwith comments.

```
/*****************************************************************/
/*   Genetic Algorithm with interface for Real valued functions    */
/*   AUTHOR : LALITESH KUMAR KATRAGADDA                            */
/*   See Goldberg for basci data structures                        */
/*****************************************************************/


#include <math.h>
#include <float.h>
#include <stdio.h>
#include <stdlib.h>
#define GAmaxpop 100
#define GAmaxstring 250
#define GAmaxn 30
#define GAmaxops 6
#define rinp "%lf"

typedef double real;
typedef struct { real v[GAmaxn+1];} vartype;
typedef unsigned char boolean;
typedef boolean allele;
```

```
typedef allele chromosome[GAmaxstring+1];
/* notice that means that the index [0..GAmaxstring], c- pah */
typedef struct {
                 chromosome chrom;
 vartype    x;
 real       fitness,funcval;
 int        parent1,parent2,xsite,optype;
      } individual;
typedef individual population[GAmaxpop+2];
typedef real oparray[GAmaxops+1];


population oldpop, newpop;
int        popsize, nelite, lchrom, gen, maxgen,maxeval, GAnvars;
real       pcross, pmutation;
int        nmutation, ncross, nfunc, scale, noduplicate,
           noperators, ops[GAmaxops+1],nbits[GAmaxn+1];
boolean    statson,output;
real       avg, max, min, sumfitness,
           scalemax, scalemin, scalesum,
           *history, *history2,
           llim[GAmaxn+1],rlim[GAmaxn+1];

oparray    opfitness, opfitini, opfitend;
real       (*objfunc)(vartype x);

/* scale : 0 (no scaling), 1 (windowing),
           2 (interpolation by increment) */
/* opfitness should sum to 1 */

#include </home/lalit/genetic/random.c>

#include </home/lalit/genetic/utility.c>

#include </home/lalit/genetic/interfac.c>

#include </home/lalit/genetic/stats.c>

#include </home/lalit/genetic/report.c>
```

```
#include </home/lalit/genetic/triops.c>

#include </home/lalit/genetic/sort.c>

#include </home/lalit/genetic/generate.c>

#include </home/lalit/genetic/initial.c>


void sga_ (real (*objfunc1)(vartype x),
   real x[], real bounds[], real opfits[],
   int *popsiz1,  int *nelite1,  int *maxgen1, int *mxevl1,
   real *pcross1, real *pmut1,   real *randseed,
           int *nodup, int  *scale1,  real *scmax,   real *scmin,
   int *stats1,real *hist1, real *hist2, char filename[10])
/* A Genetic Algorithm - GA - v2.0 */
{   FILE *out;
    int i,j,ngen;
    gen=0;
    j=0;
    GAnvars   =bounds[0];
    for (i=1; i<= GAnvars; i++)
      { nbits[i] = bounds[i];
j = j+nbits[i];
llim[i] = bounds[i+GAnvars];
rlim[i] = bounds[i+2*GAnvars];
      };
    if (j>GAmaxstring)
      { printf("******ERROR, string exceeded %d %d\n",GAmaxstring,j);
exit;
      };
    noperators=opfits[0];
    for (i=1; i<= noperators; i++)
      { ops[i]      = opfits[i];
opfitini[i] = opfits[i+noperators];
opfitend[i] = opfits[i+2*noperators];
      };

    objfunc=objfunc1;
    popsize = *popsiz1;
```

```
    maxgen = *maxgen1;
    maxeval= *mxevl1;
    pcross = *pcross1;
    pmutation  = *pmut1;
    nelite = *nelite1;
    scale = *scale1;
    scalemax = *scmax;
    scalemin = *scmin;
    history  = hist1;
    history2 = hist2;
    statson= *stats1;
    noduplicate = *nodup;
    output= ((*stats1 > 1)||(*stats1 <= 0));
    if (output) out = fopen(filename,"wt");
    initialize(out, *randseed);
    do {
        gen = gen+1;
generation();
statistics(popsize, popsize-nelite, nfunc, statson,
   &max, &avg, &min, &sumfitness, newpop,
   history, history2);
if (output) report(gen,out);
/* first sort according to fitness upto nelite individuals */
sort (nelite, popsize, newpop);
/* scale population as desired,full sorting may be required*/
scalepop(scale,popsize,max,avg,min,sumfitness,scalemax,
 scalemin,&scalesum, newpop);
setopfitness (noperators, maxeval,nfunc,
      opfitness, opfitini,opfitend);
        for (i=1; i<=popsize; i++) oldpop[i]=newpop[i];
      }
    while ((gen<maxgen)&&(nfunc<maxeval));

    if (nelite==0) sort (1, popsize, newpop);
    for (i=0;i<GAnvars; i++) x[i]=newpop[1].x.v[i];
    x[GAnvars]=newpop[1].funcval;

    if (statson)
      {
if (output) {
```

```
  fprintf(out,"\n\n\n                    HISTORY\n\n");
  for (i=1; i<=maxeval;i++)
    fprintf(out,"%d %13.10g    %13.10g\n",i,*(history+i)/
    *(history),*(history2+i)/ *(history));
};

printf("X : ");
for (i=1;i<=GAnvars;i++)
  printf("%9.7g ",newpop[1].x.v[i]);
printf("%13.10g \n",newpop[1].funcval);

      };

    if (output) fclose(out);
  }


/*****************************************************************/
/* Random Number generation Module
   including 1) advance_random, 2) warmup_random, 3) random,
             4) randomize,      5) flip,          6) rnd
*/

/* Global variables - don't use these var names in other code      */

real  oldrand[56];
int jrand;

void advance_random()
/* create next batch of random numbers */
{
   int jl;
   real  new_random;

   for ( jl=1; jl<=24; jl++ )
      {new_random = oldrand[jl] - oldrand[jl+31];
       if (new_random < 0.0) new_random = new_random+1.0;
       oldrand[jl] = new_random;
       }
```

```
   for ( jl=25; jl<=55; jl++ )
     {new_random = oldrand[jl] - oldrand[jl-24];
      if (new_random < 0.0) new_random = new_random+1.0;
      oldrand[jl] = new_random;
      }
 }


void warmup_random (real  random_seed)
/* Get random off and runnin */
{
    int jl,ii;
    real  new_random, prev_random;

    oldrand[55] = random_seed;
    new_random = 1.0e-9;
    prev_random = random_seed;
    for ( jl=1; jl<=54; jl++)
      {ii = (21*jl) % 55;
       oldrand[ii] = new_random;
       new_random = prev_random - new_random;
       if (new_random<0.0) new_random = new_random+1.0;
       prev_random = oldrand[ii];
     };
    advance_random(); advance_random(); advance_random();
    jrand=0;
  }

real  random()
/* Fetch a single random number between 0.0 and 1.0 - Subtractive
   Method See Knuth, D. (1969), v. 2 for details */
{
    jrand++;
    if (jrand<55)
      {jrand=1; advance_random(); };
    return(oldrand[jrand]);
  }

real rands_()
```

```
{
  return (random());
  }

boolean flip (real  probability)
/* Flip a biased coin - tru if heads */
{
    if (probability==1.0)
       return(1);
    else
       return(random() <= probability);
  }


int rnd (int low, int high)
/* Pick a random integer between low and high */
{
    int i;

    if (low >= high)
       i = low;
    else
      {i =  (high-low+1)*random()+low;
       if (i>high) i=high;
     };
    return(i);
  }


void randomize(real randomseed)
/* Get seed number for random and start it up */
{

  /* AUTOmate this using ftime, milliseconds */
    warmup_random(randomseed);
  }


/* Utility : Contains pause, page, repchar, skip, power */
```

```
void pause(int pauselength)
/* pause a while */
    {
#define maxpause 2500
     int i,jl;
     real  x;
     for (i=1; i<=pauselength; i++)
       for (jl=1; jl<=maxpause; jl++) x=x*1.0+0.1;
   }

void page(FILE *out)
    {fprintf(out,"\f\n\n\n");
     }

void repchar (FILE *out, char ch, int repcount)
/*repeatedly write ch to output device */
     {int j;
      for (j=1; j<=repcount; j++)
{fputc(ch,out);}
    }

void skip (FILE *out, int skipcount)
/* Skip Skipcount lines on device out */
    {int j;
     for (j=1; j<=skipcount; j++, fprintf(out,"/n"));}


real  power (real  x, real  y)
/* Raise x to the yth power */
   {return(exp(y*log(x)));}


/*****************************************************************/
/* INterface module : contains objfunc, decode */
/* change these for different problems         */


real goldberg_ (vartype x)
```

```
/* Fitness function f(x)=x**n */
/* lower lim : 0.0, ul : 1.0, 30 bits */

{
    nfunc++;
    return(power(x.v[1],10));
}


real objfunc2_ (vartype x)
/* Fitness function f(x)=x**n+y**2 */
/* same as for objfunc1, 30bits for y */
{
    nfunc++;
    return(power(x.v[1],10)+power(x.v[2],2));
}



real binf6_ (vartype x)
/* Binary F6 f(x,y)= */
/* lower lim -100.0, ul 100.0 */
/* x,y : 22 bits */

{   real temp,temp1;
    nfunc++;
    temp = x.v[1]*x.v[1] + x.v[2]*x.v[2];
    temp1 = cos(sqrt(temp));
    return(temp1*temp1/(1.0+0.001*temp*temp));
}


real dejong1_ (vartype x)
/* De Jong test function 1 */
/* Bounds [-5.12, 5.12] */
/* Ganvars 3, string length 10 for each */

{ nfunc++;
  return (-(x.v[1]*x.v[1] + x.v[2]*x.v[2]+x.v[3]*x.v[3]));
}
```

```
real dejong2_ (vartype x)
/* De Jong test function 2 */
/* Bounds [-2.048, 2.048] */
/* Ganvars 2, string length 12 for each */

{ real temp,temp1;
  nfunc++;
  temp = x.v[1]*x.v[1]-x.v[2];
  temp1 = 1-x.v[1];
  return (-100*temp*temp-temp1*temp1);
}

real dejong3_ (vartype x)
/* De Jong test function 3 */
/* Bounds [-5.12, 5.12] */
/* GAnvars 5, string length 10 for each */

{ int i;
  real temp;
  nfunc++;
  temp=0;
  for(i=1; i<=GAnvars; i++)
    temp=temp+ceil(x.v[i]);
  return (-temp);
}

real dejong4_ (vartype x)
/* De Jong test function 4 */
/* Bounds [-1.28,1.28] */
/* GAnvars 30, string length 8  for each */

{ int i;
  real temp,temp1;
  nfunc++;
  temp=0;
  for(i=1; i<=GAnvars; i++)
    { temp1 = x.v[i]*x.v[i];
      temp=temp+i*temp1*temp1;
    };
```

```
  return (-temp);
}

real dejong5_ (vartype x)
/* De Jong test function 5 */
/* Bounds [-65.536,65.536] */
/* GAnvars 2, string length 17  for each */

{ int i,j;
  real temp,temp1,temp2;
  nfunc++;
  temp=0.002;
  for(j=1; j<=25; j++)
    {
      for (i=1; i<=GAnvars; i++)
{
  if (i==j) temp1 = 1.0;
  else temp1= 0.0;
  temp1 = x.v[i]-temp1;
  temp1 = temp1*temp1*temp1;
  temp2 = temp1*temp1+ j;
};
      temp = temp + 1.0/temp2;
    };
  return (-temp2);
}


real gplanet_ (vartype x)
/* Fortran Planetary function */
/* uncomment gescape_ */

{   real f,y[GAmaxn];
    int n,i;
    nfunc++;
    n=GAnvars;
    for (i=1; y[i-1]=x.v[i], i<=n; i++);
    gaplant_ (&f,y,&n);
    return(f);
}
```

```
real emoon_ (vartype x)
/* Fortran Earth Moon transfer function */
/* GAnvars 12, bits 15,*/
/* uncomment gmoon_ */

{   real f,y[GAmaxn];
    int n,i,j;
    nfunc++;
    n=GAnvars;
    for (i=1,j=0; j <= n-1; j++)
      {
y[j]=x.v[i]; i++;
      };
    gmoon_ (&f,y,&n);
    return(f);
}

vartype decode (chromosome chrom, int lbits)
/* Decode string as unsigned binary integer- true=1, false=0 */
{
    int i,j,bit;
    long int temp,powerof2;
    vartype accum;

    powerof2=1.0;
    bit=0;

    for (i=1; i<=GAnvars; i++)
       {
 temp=0;
 powerof2=1;
         for (j=1; j<=nbits[i]; j++)
 {
   bit++;
   if (chrom[bit])
     temp=temp + powerof2;
```

```
  powerof2=powerof2*2;
 };
 powerof2=powerof2-1;
 accum.v[i]= (real)temp/ (real)powerof2*
                (rlim[i]-llim[i])+llim[i];
      };
    return(accum);
    }


/*****************************************************************/
/* stats : Statistics module                                     */

int GAkfunc=0;
/* dejong1,2,4    */
real GAsolution =0.0;
int GAmaxdig = 9;


/* dejong3
real GAsolution =25.0;
int GAmaxdig = 9;
*/

/* dejong5
real GAsolution = -1.2;
int GAmaxdig = 9;
*/

void statistics (int popsize, int ngen, int nfunc, boolean statson,
 real  *max, real  *avg, real  *min,
 real  *sumfitness, population pop,
 real ptr[], real ptr2[])
/* calculate population statistics */

    {int j,k;
     real temp;
     /* initialize */
     *sumfitness=pop[1].funcval;
```

```
     *min       =pop[1].funcval;
     *max       =pop[1].funcval;
     /* loop for max, min, sumfitness */
     for (j=2; j<=popsize; j++)
       {*sumfitness = *sumfitness+pop[j].funcval;
if (pop[j].funcval>*max) *max=pop[j].funcval;
        else if (pop[j].funcval<*min)  *min=pop[j].funcval;
      }
     /* calculate average */
     *avg = *sumfitness /popsize;
     if (statson)
       {
 if (GAkfunc>nfunc) {GAkfunc=0;};
 /* lawerence Davis's criteria */
 ptr2[0]= *max;
 for (k=GAkfunc+1,j=popsize-ngen+1; k<=nfunc; j++,k++)
   {
     temp = GAsolution-ptr2[0];
     if (temp<=0) ptr[k]=ptr[k]+GAmaxdig;
     else
       ptr[k]=ptr[k]-log10(temp);
     ptr2[k]=ptr2[k]+ptr2[0];
   };
GAkfunc=nfunc;
       };
   }



char *GASCALE[4] = {"None.","Windowing",
       "Constant increment","Window & Increment"};
void scalepop (int scale, int popsize, real  max, real  avg,
       real  min, real  sumfitness, real scalemax,
       real scalemin, real *scalesum, population pop)
/* scales population fitness from scalemax to scalemin */
      {
int i;
real scalefactor;

switch (scale) {
```

```
case 1 :
  scalefactor=(scalemax-scalemin)/(max-min);
  *scalesum = 0.0;
  for (i=1; i<=popsize; i++)
    {
      pop[i].fitness=(pop[i].funcval-min)*scalefactor
 +scalemin;
      *scalesum = *scalesum+pop[i].fitness;
    };
  /*note: *scalesum = scalefactor*(sumfitness-min*popsize)
    +scalemin*popsize; try this and check  */
  break;

case 2 : /* use only if fully sorted */
  scalefactor = (scalemax-scalemin)/popsize;
  pop[popsize].fitness=scalemin;
  *scalesum = pop[popsize].fitness;
  for (i=popsize-1; i>=1; i--)
    {
      pop[i].fitness=pop[i+1].fitness+ scalefactor;
      *scalesum = *scalesum+pop[i].fitness;
    }
  break;

case 3 : /* use only if fully sorted */
  scalefactor = (max-min)/popsize*scalemax;
  pop[popsize].fitness=scalemin*(max-min)/popsize
                       +scalefactor;
  *scalesum = pop[popsize].fitness;
  for (i=popsize-1; i>=1; i--)
    {
      pop[i].fitness=(pop[i].funcval-pop[i+1].funcval)
             + pop[i+1].fitness+scalefactor;
      *scalesum = *scalesum+pop[i].fitness;
    }
  break;

case 0 :
  *scalesum=sumfitness;
  for (i=1; i<=popsize; i++)
```

```
    pop[i].fitness = pop[i].funcval;
  break;
};
      }

void setopfitness (int noperators, int maxeval, int nfunc,
 oparray opfitness, oparray opfitini,oparray opfitend)
{int i;
 real factor,sum;
 sum=0.0;
 factor= (real) nfunc/ (real) maxeval;
 for (i=1; i<noperators; i++)
   {
     opfitness[i]=opfitini[i] + (opfitend[i]-opfitini[i])*factor;
     sum=sum+opfitness[i];
   };
 opfitness[noperators] = 1.0-sum;
}


/* Output module                              */
/* report.c : contains writechrom, report */

void writechrom(FILE *out, chromosome chrom, int lchrom)
/* Write a chromosome as a string of 1's (true's) & 0's(false's)*/
{    int j;
     for (j= lchrom; j>=1; j--)
          if (chrom[j])
    fputc('1',out);
          else
    fputc('0',out);
   }


void report(int gen, FILE *out)
/* Write the population report */
#define linelength 132
{    int j,k;
     repchar(out,'-',linelength); fprintf(out,"\n");
     repchar(out,' ',50); fprintf(out,"Population Report\n");
```

```
    repchar(out,' ',23); fprintf(out,"Generation %2d",gen-1);
    repchar(out,' ',57); fprintf(out;"Generation %2d\n",gen);
    fprintf(out,"\n");
    fprintf(out," #              x       fitness");
    fprintf(out,"       #  parents xsite");
    fprintf(out,"                 string                    x  ",
                "    fitness  \n");
    repchar(out,'-',linelength); fprintf(out," \n");

    if (!statson)
    for(j=1; j<= popsize; j++)
      {
fprintf(out,"    %2d  ",j); /*old string*/
fprintf(out," %9g %8g      |",oldpop[j].x.v[1],
oldpop[j].fitness);

    /*new string*/
fprintf(out,"%2d) %1d:(%2d,%2d)   %2d    ",j,
newpop[j].optype,newpop[j].parent1,
newpop[j].parent2,newpop[j].xsite);
writechrom(out,newpop[j].chrom,lchrom);
fprintf(out," %9g %8g  \n",newpop[j].x.v[1],
newpop[j].funcval);
      }
    else
      for(j=nelite+1; j<= popsize; j++)
      {
fprintf(out,"%2d) %1d:(%2d,%2d)   %2d    ",j,
newpop[j].optype,newpop[j].parent1,
newpop[j].parent2,newpop[j].xsite);
for (k=1;k<=GAnvars;k++)
fprintf(out," %9g ",newpop[j].x.v[k]);
fprintf(out," %8g \n",newpop[j].funcval);
      }

    repchar(out,'-',linelength); fprintf(out," \n");
    /* Generation statistics and accumulated values */
    fprintf(out," Note: Generation %2d & Accumulated Statistics: ",
                gen);
    fprintf(out," max= %6.4f, min= %7.5f, avg=%6.4f",max,min,avg);
```

```
    fprintf(out,", sum=%6.4f, numutation=%d, ncross= %d, NFN=%d \n",
           sumfitness, nmutation, ncross, nfunc);
    repchar(out,'-',linelength); fprintf(out," \n");
    page(out);
  }



/* Triops module */
/* Reproduction (select), Crossover (crossover), Mutation (mutatio*/
/* Plus four others                                               */


int  select(int popsize, real  sumfitness, population pop)
/* select a single individual via roulette wheel selection */

{    real  rand, partsum; /*random point on wheel, partial sum*/
     int j;               /*population index*/
     partsum=0.0; j=0; /*zero out pointer & accumulator*/
     rand = random()*sumfitness;
         /* wheel point calc. uses random[0..1]*/
     do{/*find wheel slot */
          j=j+1;
  partsum = partsum+pop[j].fitness;
        } while((partsum<rand)&&(j<popsize));
     /* return individual number */
     return(j);
   }

int  selectop(int noperators, real opfitness[])
/* select an operator via roulette wheel selection */

{    real  rand, partsum;
     int j;
     partsum=0.0; j= 0;
     rand = random();
     do{  j++;
  partsum = partsum+opfitness[j];
        } while((partsum<rand)&&(j<noperators));
     return(ops[j]);
```

```
   }

allele mutation(allele alleleval, real  pmutation, int *nmutation)
/* Mutate an allele w/ pmutation, count no. of mutations */
{    boolean mutate;
     mutate = flip(pmutation);
     if (mutate) {
         *nmutation= *nmutation+1;
 return(!alleleval); /*change bit value*/
       }
     else
        return(alleleval);
   }

/*************************************************************/

/** Operator : 0 **/
void crossover(chromosome parent1, chromosome parent2,
       chromosome child1,  chromosome child2,
       int *lchrom, int *ncross, int *nmutation, int *jcross,
       real  *pcross, real  *pmutation,
               boolean *new1, boolean *new2)
/* Cross two parent strings, place in two child strings */
{    int j,temp1,temp2;

     if (flip(*pcross)) {
         *jcross = rnd(1,*lchrom-1); /*cross between 1 & l-1 */
 *ncross = *ncross+1;
       }
     else
       {
         *jcross = *lchrom; /*force mutation */
 *new1   = 0;
 *new2   = 0;
       };

     temp1 = *nmutation;
     temp2 = *nmutation;
     /*1st exchange 1-1, 2-2  */
     for (j=1; j<=*jcross; j++)
```

```
      {
 child1[j] = mutation(parent1[j],*pmutation, &temp1);
 child2[j] = mutation(parent2[j],*pmutation, &temp2);
      };

    /* 2nd exchange, 1-2, 2-1 */
    if (jcross != lchrom) /* skip if xsite is lchrom--no xover*/
       for (j = *jcross+1; j <= *lchrom; j++)
  {
    child1[j] = mutation(parent2[j],*pmutation, &temp1);
    child2[j] = mutation(parent1[j],*pmutation, &temp2);
  };
    *new1 = (*new1 || (temp1 > *nmutation));
    *new2 = (*new2 || (temp2 > *nmutation));
    *nmutation = temp1 + (temp2-*nmutation);

   }


/** Operator : 1 **/
void pure_cross(chromosome parent1, chromosome parent2,
      chromosome child1,  chromosome child2,
      int *lchrom, int *ncross, int *jcross)
/* Cross two parent strings, place in two child strings */
{    int j;

     *jcross = rnd(1,*lchrom-1); /*cross between 1 & l-1 */
     *ncross = *ncross+1;

     /*1st exchange 1-1, 2-2  */
     for (j=1; j<=*jcross; j++)
       {
         child1[j] = parent1[j];
 child2[j] = parent2[j];
       };

     /* 2nd exchange, 1-2, 2-1 */
        for (j = *jcross+1; j <= *lchrom; j++)
  {
    child1[j] = parent2[j];
```

```
    child2[j] = parent1[j];
  };
   }
```

```
/** Operator : 2 **/
void  mutate (chromosome parent, chromosome child, int *lchrom,
      int *nmutation, real  *pmutation, boolean *new)

/* Mutate parent and produce a child. Clone with random changes */
{    int j,temp;

     temp = *nmutation;
     for (j=1; j<=*lchrom; j++)
       child[j] = mutation(parent[j],*pmutation, &temp);

     *new = (temp > *nmutation);
     *nmutation = temp;
   }
```

```
/** Operator : 3 **/
void uniform_cross(chromosome parent1, chromosome parent2,
       chromosome child1,  chromosome child2,
       int *lchrom, int *ncross)
/* Cross two parent strings, place in two child strings */
{    int j;

     *ncross = *ncross+1;

     for (j=1; j<=*lchrom; j++)
       if (flip(0.5))
 {
   child1[j] = parent1[j];
   child2[j] = parent2[j];
 }
```

```
      else
 {
   child1[j] = parent2[j];
   child2[j] = parent1[j];
 };
   }


/** Operator : 4 **/
void uniform_cross2(chromosome parent1, chromosome parent2,
       chromosome child1,  chromosome child2,
       int *lchrom, int *ncross, int *jcross)
/* Cross two parent strings, place in two child strings */
{    int j;

     *jcross = rnd(1,*lchrom-1); /*cross between 1 & l-1 */
     *ncross = *ncross+1;

     for (j=1; j<=*lchrom; j++)
       if ((j <= *jcross)||(flip(0.5)))
 {
   child1[j] = parent1[j];
   child2[j] = parent2[j];
 }
       else
 {
   child1[j] = parent2[j];
   child2[j] = parent1[j];
 };
   }


/** Operator : 5 **/
void multi_cross(chromosome parent1, chromosome parent2,
       chromosome child1,  chromosome child2,
       int *ncross, real *pcross,
       boolean *new1, boolean *new2)
/* Cross two parent strings, place in two child strings */
{    int i,j,jcross,bit;
     boolean new;
```

```
    bit=1;
    new=0;
    for (i=1; i<=GAnvars; i++)
      {
if (flip(*pcross))
  {
    jcross = rnd(1,nbits[i]-1); /*cross between 1 & l-1 */
    new=1;
  }
else
  jcross = nbits[i];

/*1st exchange 1-1, 2-2  */
for (j=1; j<=jcross; j++,bit++)
  {
    child1[bit] = parent1[bit];
    child2[bit] = parent2[bit];
  };

/* 2nd exchange, 1-2, 2-1 */
for (j = jcross+1; j <= nbits[i]; j++,bit++)
  {
    child1[bit] = parent2[bit];
    child2[bit] = parent1[bit];
  };
      };
    *new1 = (*new1 && new);
    *new1 = *new2;
  }


/** Operator : 6 **/
void uniform_cross3(chromosome parent1, chromosome parent2,
       chromosome child1,  chromosome child2,
       int *lchrom, int *ncross, int *nmutation,
       real  *pcross, real  *pmutation,
    boolean *new1, boolean *new2)
{    int j,temp1,temp2;
```

```
    temp1 = *nmutation;
    temp2 = *nmutation;
    for (j=1; j<=*lchrom; j++)
      if (flip(0.5))
 {
   child1[j] = mutation(parent1[j],*pmutation, &temp1);
   child2[j] = mutation(parent2[j],*pmutation, &temp2);
 }
      else
 {
   child1[j] = mutation(parent2[j],*pmutation, &temp1);
   child2[j] = mutation(parent1[j],*pmutation, &temp2);
 };
    *new1 = (*new1 || (temp1 > *nmutation));
    *new2 = (*new2 || (temp2 > *nmutation));
    *nmutation = temp1 + (temp2-*nmutation);
  }


/*****************************************************************/
/*     Sorting Module                                            */

  int GAcompare(individual *a, individual *b)
   {
if ((*a).funcval>(*b).funcval)
  return(-1);
else
  {
    if ((*a).funcval<(*b).funcval) return(1);
    else return(0);
  }
      }


  void sort (int nelite, int popsize, population pop)
    {
      int best,i;
      individual temp;
         if (nelite <= 0) exit;
```

```
 if (nelite==1) {
    best=1;
    for (i=2; i<=popsize; i++)
      if (pop[i].funcval > pop[best].funcval) best=i;
    temp = pop[1];
    pop[1] = pop[best];
    pop[best] = temp;
 }
 else
   qsort(&(pop[1]),popsize,sizeof(individual),GAcompare);
    }



/*******************************************************************/
/* Generation module                                               */


boolean GAnotequal (individual *test, individual *kid)
{
  int i,notequal;
  notequal=0;
  for (i=1;((i<=lchrom)&&(!notequal));i++)
    notequal = ((*test).chrom[i] ^ (*kid).chrom[i]);
  return(notequal);
}

boolean deliver_child(individual *kid,
      boolean newchild, int jcross,
      int mate1, int mate2, int npop)
/* inoldpop : true => match found in oldpop, not new */
{int i,j,notequal,jeq;
 boolean inoldpop;
 inoldpop = 0;
 notequal=1;
 switch (abs(noduplicate))
   {
   case 1 :
   case 2 :
   case 3 :
```

```
   jeq=mate1;
           if (!newchild)
     if (mate1<=nelite)
       {notequal=0; break;}
     else
       inoldpop=1;


   if (abs(noduplicate)<2) break;

   if (!GAnotequal(&oldpop[mate1],kid))
     {if (mate1<=nelite)
{inoldpop=0; notequal=0; break;}
     else
       inoldpop=1;
    };
   if (mate1 != mate2)
     if (!GAnotequal(&oldpop[mate2],kid))
       {jeq=mate2;
if (mate2<=nelite)
       {notequal=0; inoldpop=0; break;}
else
  inoldpop=1;
      };
   if (abs(noduplicate)<3) break;

   for (j=npop; ((j>=1)&&(notequal)); j--)
     if ((j>nelite)||((j!=mate1)&&(j!=mate2)))
       notequal = GAnotequal(&newpop[j],kid);
   if (!notequal) jeq=j;
 };

  if ((notequal)&&(!inoldpop))
    {
      /* decode string, evaluate fitness */
  (*kid).x = decode((*kid).chrom,lchrom);
  (*kid).funcval = objfunc((*kid).x);
    }
  else
    {
```

```
      if (inoldpop)
{
  (*kid).x = oldpop[jeq].x;
  (*kid).funcval = oldpop[jeq].funcval;
}
      else
{
  if (noduplicate > 0) return(0);
  (*kid).x = newpop[jeq].x;
  (*kid).funcval = newpop[jeq].funcval;
};
    };
  /* record parentage data on both children */
  (*kid).parent1 = mate1;
  (*kid).parent2 = mate2;
  (*kid).xsite   = jcross;
  return(1);
}


void generation()
/* Create a new generation through select, crossover, and mutation */
/* Note : Generation does not assume an even numbered popsize      */
/*        however if nelite>=1, sorted oldpop is assumed           */
{    int j,mate1, mate2, jcross, op;
     boolean newchild1,newchild2;

     /* copy to save the elite individuals from oldpop (sorted)*/
     for (j=1; j<=nelite; j++)
       newpop[j] = oldpop[j];

     /* generate the rest of the individuals */
     j=nelite+1;
     do {
       op = selectop (noperators, opfitness);
       mate1 = select(popsize, scalesum, oldpop);
       jcross=0;
       newpop[j].optype=op;
       switch (op)
 {
```

```
case 2 :
  break;
default :
  mate2 = select(popsize, scalesum, oldpop);
  newchild1 = (mate1 != mate2);
  newchild2 = newchild1;
  newpop[j+1].optype=op; break;
};

      switch (op)
{
case 0 :
  /* Crossover and mutation */
  crossover(oldpop[mate1].chrom, oldpop[mate2].chrom,
    newpop[    j].chrom, newpop[j+1  ].chrom,
    &lchrom,&ncross,&nmutation,&jcross,&pcross,
    &pmutation,&newchild1,&newchild2);
  break;
case 1 :
  pure_cross (oldpop[mate1].chrom, oldpop[mate2].chrom,
      newpop[    j].chrom, newpop[j+1  ].chrom,
      &lchrom,&ncross,&jcross);
  break;
case 2 :
  mutate (oldpop[mate1].chrom, newpop[j].chrom,
  &lchrom,&nmutation,&pmutation,&newchild1);
  break;
case 3 :
  uniform_cross (oldpop[mate1].chrom, oldpop[mate2].chrom,
 newpop[    j].chrom, newpop[j+1  ].chrom,
 &lchrom,&ncross);
  break;
case 4 :
  uniform_cross2 (oldpop[mate1].chrom, oldpop[mate2].chrom,
  newpop[    j].chrom, newpop[j+1  ].chrom,
  &lchrom,&ncross,&jcross);
  break;
case 5 :
  multi_cross (oldpop[mate1].chrom, oldpop[mate2].chrom,
newpop[    j].chrom, newpop[j+1  ].chrom,
```

```
&ncross, &pcross, &newchild1,&newchild2);
   break;
 case 6 :
   uniform_cross3(oldpop[mate1].chrom, oldpop[mate2].chrom,
  newpop[    j].chrom, newpop[j+1  ].chrom,
  &lchrom,&ncross,&nmutation,&pcross,
  &pmutation,&newchild1,&newchild2);
   break;

 default :
   printf ("invalid operator %d \n",op); break;
 };

       switch (op)
 {
 case 2 :
   if (deliver_child(&newpop[j],newchild1,
     0,mate1,mate1, j-1)) j++;
   break;
 default :
   if (deliver_child(&newpop[j],newchild1,
     jcross,mate1,mate2, j-1))
     {j++;}
   else
     newpop[j] = newpop[j+1];

   if ((j<=popsize)&&
       deliver_child(&newpop[j],newchild2,
     jcross,mate2,mate1, j-1)) j++;
 };

     } while(j<=popsize);
   }



/* initial : contains initdata, initpop, initreport, initialize */

void initdata (int *popsize, int *lchrom, int *maxgen,
        real *pcross, real *pmutation)
```

```
/* Interactive data inquiry and setup */
{
    printf ("        |-------------------------------|\n");
    printf ("        | Genetic Algorithm v2.0    -  GA|\n");
    printf ("        |-------------------------------|\n\n\n");
    printf ("***** Data entry and Initialization *****\n");
    printf ("Enter population size       : ");
    scanf ("%d",popsize);
    printf ("Enter elite population size       : ");
    scanf ("%d",nelite);
    printf ("Enter chromosome length      : ");
    scanf ("%d",lchrom);
    printf ("Enter max generations        : ");
    scanf ("%d",maxgen);
    printf ("Enter crossover probability  : ");
    scanf (rinp,pcross);
    printf ("Enter mutation probability   : ");
    scanf (rinp,pmutation);
  }


void initreport(FILE *out, real randseed)
{   int i;
    fprintf (out,"-------------------------------\n");
    fprintf (out,"A  Genetic Algorithm, V 2.0     \n");
    fprintf (out,"-------------------------------\n\n");


    fprintf (out,"      GA Parmaters \n");
    fprintf (out,"      ------------ \n\n");
    fprintf (out,"    Population size        : %d\n",popsize);
    fprintf (out,"    Elite population size  : %d\n",nelite);
    fprintf (out,"    Fitness Scaling        : %s\n",GASCALE[scale]);
    fprintf (out,"    Chromosome length      : %d  (",lchrom);
    for (i=1; i<=GAnvars; i++) fprintf (out," %d",nbits[i]);
    fprintf (out, " )\n");
    fprintf (out,"    Domain                 : ");
    for (i=1; i<=GAnvars; i++)
      fprintf (out,"x%d:[%g,%g]  ",i,llim[i],rlim[i]);
    fprintf (out,"\n");
```

```
  fprintf (out,"    Noduplicate mode        : %d\n",noduplicate);
  fprintf (out,"    Max generations         : %d\n",maxgen);
  fprintf (out,"    Max evaluations         : %d\n",maxeval);
  fprintf (out,"    Crossover probability   : %g\n",pcross);
  fprintf (out,"    Mutation probability    : %g\n",pmutation);
  fprintf (out,"    Operator fitnesses      : ");
  for (i=1; i<=noperators; i++)
    fprintf (out,"%d:[%g,%g] ",ops[i],opfitini[i],opfitend[i]);
  fprintf (out,"\n");
  fprintf (out,"    Random seed             : %11.9g\n\n\n\n",
           randseed);

 fprintf (out,"      Initial Generation Statistics\n");
 fprintf (out,"      ----------------------------\n");
 fprintf (out,"    Initial population maximum fitness = %g\n",max);
 fprintf (out,"    Initial populatiop average fitness = %g\n",avg);
 fprintf (out,"    Initial populatiop minimum fitness = %g\n",min);
 fprintf (out,"    Initial populatiop sum of  fitness = %g\n",
              sumfitness);
 fprintf (out,"\n\n\n\n\n\n\n\n");
}



/*************** Population initialization routines ***************/

 void initpop()
 /*  Initialize a population at random */
 {
   int j,jl;
   for (j=1; j<=popsize; j++)
     {for (jl=1; jl<=lchrom; jl++) oldpop[j].chrom[jl]=flip(0.5);
      oldpop[j].x = decode(oldpop[j].chrom, lchrom);
      oldpop[j].funcval = objfunc(oldpop[j].x);
      oldpop[j].parent1=0; oldpop[j].parent2=0; oldpop[j].xsite=0;
    };
 }


 /* geneinp ; initialization by file */
```

```
  void sinitpop()
  /* Initialize a population at random */
  {   FILE *inp;
      int j,jl;
      char schrom[31];
      inp = fopen("gene.ini","rt");
      for (j=1; j<=popsize; j++)
{
  fscanf(inp,"%s",schrom);
  /* printf("%s\n",schrom); */
  for (jl=1; jl<=lchrom; jl++)
    oldpop[j].chrom[jl]= (schrom[30-jl]=='1');
  oldpop[j].x = decode(oldpop[j].chrom, lchrom);
  oldpop[j].funcval = objfunc(oldpop[j].x);
  oldpop[j].parent1=0; oldpop[j].parent2=0; oldpop[j].xsite=0;
};
    }

/*****************************************************************/



void initialize(FILE *out, real randseed)
/* Initialization coordinator */
{   int i;
    lchrom=0;
    for (i=1; i<= GAnvars; i++)
      lchrom=lchrom+nbits[i];
    for (i=1; i<=noperators; i++)
      opfitness[i] = opfitini[i];
 /* initdata(); */
    randomize(randseed);
    nmutation=0;
    ncross=0;
    nfunc=0;
    initpop();
```

```
  statistics(popsize, popsize, nfunc, statson,
     &max, &avg, &min, &sumfitness, oldpop,
     history, history2);
  sort (nelite, popsize, oldpop);
  scalepop(scale,popsize,max,avg,min,sumfitness,scalemax,
   scalemin,&scalesum, oldpop);
  if (output) initreport(out,randseed);
}
```