A LÈVY FLIGHT BASED APPROACH TO SOLVING OPTIMAL
CONTROL PROBLEMS WITH INDIRECT METHODS

A Thesis in
Aerospace Engineering
by
Thomas Palazzo

© 2019 Thomas Palazzo

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

December 2019
The thesis of Thomas Palazzo was reviewed and approved* by the following:

Puneet. Singla
Associate Professor of Aerospace Engineering
Thesis Advisor

Robert G. Melton
Professor of Aerospace Engineering

Amy R. Pritchett
Professor of Aerospace Engineering
Head of the Department of Aerospace Engineering

*Signatures are on file in the Graduate School.
Abstract

Optimal control has received substantial attention in the field of aerospace engineering, particularly in trajectory optimization of space vehicles, where minimal-cost solutions to maneuver problems can save millions of dollars in mission costs. Solutions to optimal control problems come in two forms: the direct methods where the continuous problem is discretized and control input is specified at each point, or the indirect methods where the optimal control solution is derived mathematically from a set of necessary conditions. In the latter method, the solution constitutes a two-point boundary problem where an initial guess to the optimal solution must be supplied. Two-point boundary value problems are notoriously difficult to solve as their solution may be highly sensitive to the system’s initial conditions and there is often no physical intuition for generating a good initial guess. Despite this, indirect methods are preferred as they provide a way of understanding the sensitivity of the cost function to the system’s initial state through the costates.

This thesis provides a procedure for numerically solving optimization problems via indirect methods. To consistently generate good initial guess solutions, metaheuristic algorithms, that is, heuristic optimization algorithms with a natural metaphor, are used. These algorithms excel at arriving at close-to-optimal solutions to problems in the absence of a priori knowledge. The metaheuristic algorithms particle swarm optimization and the firefly algorithm are used as well as the Lèvy flight firefly algorithm, a modification to the firefly algorithm incorporating the Lèvy flight foraging hypothesis from optimal foraging theory. These metaheuristics are used to bypass the difficulty of indirect methods by obtaining a good guess of the optimal solution which can then be used to initialize a two-point boundary value problem solver. To show the efficacy of this approach, four optimal control problems of varying difficulty are presented with the metaheuristic/indirect method approach successfully deriving the optimal solution in each case.
# Table of Contents

List of Figures vii

List of Tables viii

Acknowledgments ix

Chapter 1
   Introduction 1

Chapter 2
   Optimal control 6
      2.1 Optimal control theory 6
         2.1.1 Boundary conditions 8
         2.1.2 Pontryagin’s minimum principle 9
      2.2 Solving optimal control problems 10
         2.2.1 Direct methods 10
         2.2.2 Indirect methods 11

Chapter 3
   Metaheuristics 13
      3.1 Particle swarm optimization 13
      3.2 Firefly algorithm 16
      3.3 Optimal foraging theory 19
      3.4 Lèvy flight firefly algorithm 23
         3.4.1 Testing firefly parameters 23
            3.4.1.1 Investigating randomness 24
            3.4.1.2 Investigating minimum attractiveness 25
            3.4.1.3 Investigating the Lèvy parameter 26
Chapter 4

Solving optimal control problems

4.1 Search methodology ................................................. 29
4.2 Zermelo’s problem .................................................. 31
  4.2.1 Metaheuristic results ......................................... 32
  4.2.2 Using guesses with bvp4c .................................... 39
  4.2.3 Comparison of algorithm performance ....................... 40
4.3 Minimum-time orbit injection ....................................... 44
  4.3.1 Metaheuristic results ......................................... 45
  4.3.2 Using guesses with Bvp4c .................................... 47
  4.3.3 Comparison of algorithm performance ....................... 49
4.4 Mars orbital transfer ................................................ 50
  4.4.1 Metaheuristic results ......................................... 53
  4.4.2 Performance comparisons .................................... 58

Chapter 5

Goddard rocket problem .............................................. 60

5.1 The Goddard rocket ................................................ 60
5.2 Metaheuristic results ............................................. 64
5.3 Performance comparisons ......................................... 67
5.4 Literature results ................................................. 68

Chapter 6

Conclusion ................................................................. 70

Bibliography ................................................................ 72
List of Figures

1.1 Methods for solving optimal control problems. 3

3.1 Comparison of a Lèvy flights versus a Brownian random walk. 21

3.2 Contours of the 2-d Ackley function. 24

3.3 Comparison of Lèvy Firefly Algorithm performance for different values of $\alpha$. 25

3.4 Comparison of Lèvy Firefly Algorithm performance for different values of $\beta_{\text{min}}$. 26

3.5 Comparison of Lèvy Firefly Algorithm performance for different values of $\lambda$. 27

4.1 The sensitivity of the Zermelo problem to changes in the initial guess. 33

4.2 Evolution of LFA guesses with iteration number. 34

4.3 Contour plots of the Zermelo problem fitness function. 36

4.4 Averaged state trajectory integrated from initial guesses generated in the $N = 1000$ monte carlo. 38

4.5 Averaged bvp4c solution for the four algorithms tested. The FA, LFA and PSO$_{30}$ all converged to the same solution. 40
4.6 Scatter plot showing each initial guess generated by the metaheuristics for the Zermelo problem. 43
4.7 Histograms of $\lambda(t_0)$ for the minimum-time orbit injection problem. 46
4.8 State variables for the minimum time orbit injection problem. 47
4.9 State variables for the minimum time orbit injection problem. 48
4.10 State variables for the minimum time orbit injection problem. 48
4.11 Coordinate system for the Earth-Mars orbital transfer. 51
4.12 Histograms of $\lambda(t_0)$ for the Earth-Mars orbital transfer problem. 54
4.13 Earth-mars orbital transfer trajectory for the firefly algorithm. 55
4.14 Earth-mars orbital transfer trajectory for the Lévy flight firefly algorithm. 56
4.15 Earth-mars orbital transfer trajectory for the particle swarm optimization. 57

5.1 Histograms of $\lambda(t_0)$ for the Goddard rocket problem. 65
5.2 State and co-state values for the Goddard Rocket problem. 67
5.3 Thrust and singular arc condition time history in the Goddard Rocket problem. 68
5.4 Left: Thrust time history obtained by Graichen et al. Right: Thrust time history obtained by Dolan et al. 69
List of Tables

3.1 Parameters used in the FA and LFA algorithms. Mu governs the Lévy distribution and is hence only used in LFA. 27

4.1 Results of an $N = 1000$ Monte Carlo simulation of the FA, LFA, and PSO algorithms for the Zermelo problem. 37

4.2 Zermelo problem results after $\lambda(t_0)$ guesses are passed through \texttt{bvp4c}. 39

4.3 Run-time and function evaluation statistics for each of the algorithms solving the Zermelo problem. 42

4.4 Metaheuristic results for the minimum time orbit injection problem. 47

4.5 Run-time and function evaluation statistics for the minimum-time orbit injection problem. 50

4.6 Mars orbital transfer final states of guesses generated by each algorithm. 54

4.7 Run-time statistics for the Low Thrust Mars transfer problem. 59

5.1 Terminal state values for the Goddard Rocket Problem. 66

5.2 Terminal constraint values for the Goddard Rocket Problem. 66

5.3 Optimal solutions to the Goddard Rocket problem. 66

5.4 Metaheuristic performance statistics for the Goddard Rocket problem. 68
Acknowledgments

First and foremost acknowledgement should go to Dr Singla. Thank you for your guidance, assistance and support in writing this thesis and a special thank you for allowing me to complete this from Australia in the company of my family despite the challenges this might have caused. Speaking of my family, I would like to thank them for their continued love and support. The roof over my head at the ‘Crescent Hill’ farm was greatly appreciated as were the cups of coffee that appeared mysteriously by my computer every morning before I woke up.

I am not going to name all of the friends who helped me get here but naturally I do have to signal out Frank. The pillows you gave me are probably the greatest gift I will ever receive. To the friends I have made in State College, thanks for helping with the homesickness and for letting me crash on your couches whenever I locked my house keys in my office. This thesis would not have been possible without you as I would have quite literally frozen to death in these harsh American winters.

As a final thank you I think it is important to acknowledge the staff at the West College Starbucks for keeping me juiced up. The coffee might have been mediocre but the company was damn fine. Cheers to you all.
Chapter 1
Introduction

Optimization: is the process of making the best or most effective use of a situation or resource [1]. An everyday realization of this topic might come in the form of simple questions on logistics: the cheapest airfare to Hawaii or the fastest route between Times Square and a restaurant in the West Village. Difference in discipline aside, these questions have a good deal of commonality with, for example, minimizing the fuel consumed by a rocket landing on the moon. In either case there is a quantity that must be minimized, a cost. There are variables that are subject to constraints, states. And there are decisions that can be made to affect the outcome, controls. These quantities are the foundation of optimization, and optimal control theory provides the mathematical tools and formalism necessary to derive the optimal solution to an incredibly diverse range of problems.

Notwithstanding the historical contributions of Newton and countless others, the original formulation of the optimal control problem in terms of costs and states was by Euler [2]. Together with Lagrange, a variational analysis approach utilizing Lagrange’s multipliers explored perturbations in the solution from an optimal curve. This analysis led to deriving the first-order necessary condition for optimality, today known as the Euler-Lagrange equation. Between the 18th and 20th centuries, Legendre (with significant contributions from Clebsch) would derive the second-order necessary conditions on optimality [2,3] in terms of the system Hamiltonian, a concept introduced by Hamilton when re-formulating classical mechanics in terms of the variational principles of Euler and Lagrange [2]. In the 20th century, optimal control established
its modern formalism. Pontryagin [4] famously established the Maximum Principle\(^1\) which fully derived the necessary conditions for optimality in terms of the system Hamiltonian. Simultaneously, Bellman would pioneer the mathematical optimization method known as dynamic programming in which optimization problems are solved by recursively dividing the problem into simpler sub-problems and proceeding optimally through each one. The result of this method is the sufficient condition for optimality obtained by solving what is now known as the Hamilton-Jacobi-Bellman equation [3,5,6]. Bellman’s method, though influential, succumbs to the “curse of dimensionality”, where for problems of sufficient dimension the computation time required to obtain a solution renders them intractable [5].

The result of Pontryagin’s necessary conditions is a two point boundary value problem (TPBVP) where the optimal control policy can be derived through Lagrange multipliers known as costates. In the absence of analytical solutions, guesses must be supplied for unknown information about the system’s initial state or costate [7,8]. Supplying these guesses is often unintuitive and the extreme sensitivity of the coupled differential equations to their initial state makes arriving at the set of optimal unknowns a great challenge. Because of this, optimal control is as much a mathematical science as it is a numerical one [3]. In most problems analytical solutions for parameter unknowns are unobtainable, and numerical methods are required to guess an initial state which, when integrated, satisfies the terminal boundary conditions and produces the minimum cost [9]. One such method is the shooting method, where a supplied initial guess is integrated and the deviation of the final state with respect to the boundary conditions is used to update the initial guess iteratively until agreement is achieved, in effect transforming the boundary value problem into the simpler initial value problem (IVP). Methods of this nature are commonly referred to as the indirect methods. While these methods have been successful in solving a number of contemporary optimization problems [9–11], their sensitivity to a “good” initial guess means that one generally has to guess a solution approximately equal to the unknown optimal solution to converge [11]. This is in many cases an insurmountable task as there is often no physical intuition for the

\(^1\)Due to a difference in sign convention, the maximum principle is also referred to as the minimum principle.
initial costate unknowns and they can, in principle, be any real number.

Modern algorithms, such as the multiple-shooting method [12,13], typically replace integrating the full interval from a guessed initial state and costate with dividing the problem into $N$ grid$^2$ points and guessing the state and costate at each of them. This dramatically increases the dimension of the problem, however the system becomes less prone to unstable trajectories as the integration is over a sufficiently short time period so that the instabilities cannot develop [13]. As a contrast to the indirect methods, **direct methods** discretize the optimal control problem to a grid and directly specify a guess for the state and control variables at each point. The continuous trajectory between points is then approximated by a sequence of polynomial segments [14,15]. The collocation method is one such example. In a similar vein, the pseudospectral method scales the time domain to $t \in [-1, 1]$, discretizes the interval using a Gauss-Lobatto grid and then generates a convergent sequence of interpolating functions, typically the Legendre-polynomials, between the grid points [14,16,17]. Both of these methods replace the computationally-intense numerical integration of shooting methods with some form of interpolation and consequently introduce a set of $\sim N$ continuity constraints which must also be satisfied. These more recent numerical techniques greatly enhance the robustness of each method, with the system no longer being as sensitive to a good initial guess. Figure 1.1 shows a summary of contemporary optimization techniques.

![Figure 1.1. Methods for solving optimal control problems. The dashed sections correspond to the scope of this work.](image)

However, neither direct nor indirect algorithms overcome the fundamental problem

$^2$Sometimes referred to as a mesh or collocation points.
of providing initial guesses for the solution to the optimal control problem. Heuristic optimization tools, like genetic algorithms or particle swarm optimization, can be used to bridge this gap [18,19]. Heuristic algorithms excel at arriving at good (not optimal) solutions to minimization problems when confronted with no apriori information. They generally achieve this through some randomized, multi-agent search with a feedback response that moves the search agents towards more optimal solutions. This can been used to create a coarse/fine approach where heuristic optimization is used until a sufficiently good guess is obtained, at which point a numerical optimization procedure can be invoked [20]. Such techniques have almost exclusively been used in conjunction with the direct methods [14,20–23]; however, by explicitly specifying the control policy, direct methods rob one of the ability to interpret the sensitivity of the cost function to the initial state [24]. Therefore, if a complete picture of the optimization problem is desired, indirect methods are required.

The objective of this thesis is to address this gap in the literature by interfacing the indirect methods with heuristic optimization tools to solve the optimal control problem. To this end, a general method for generating good initial guesses for the unknown Lagrange parameters utilizing heuristic optimization algorithms will be tested on a number of optimal control problems. As a secondary objective, a novel heuristic optimization algorithm, the Lévy Flight firefly algorithm, will be tested against particle swarm optimization and its progenitor, the firefly algorithm. This algorithm is inspired by optimal foraging theory which posits Lévy flights are employed by a myriad of species to optimize searches when foraging [25].

Chapter 2 will provide the mathematical basis for optimal control. The necessary conditions for optimality will be derived along with the boundary conditions required for solving the optimal control problem via indirect methods.

Chapter 3 describes the heuristic optimization algorithms used in this work. A historical discussion of particle swarm optimization and the firefly algorithm is given along side justification for the Lévy flight firefly algorithm in terms of optimal foraging theory. Examination of firefly algorithm parameters is also provided.

Chapter 4 presents the methodology used in this work to pair metaheuristics with indirect methods to solve optimal control problems. Solutions to the Zermelo problem, a minimum-time orbit injection problem and a low-thrust orbital transfer
are also presented with comparisons in the performance of each metaheuristic drawn.

Chapter 5 present a solution the Goddard Rocket problem. Discussion of the solutions generated by the metaheuristics will be given along with comparisons to literature solutions to the problem.
Chapter 2  |  Optimal control

This chapter mathematically derives the optimal control relationships which must be satisfied to ensure the optimality of a solution. The equations and conditions derived here will be used in Chapters 4–5 to derive the optimal control policy for each of the discussed problems.

2.1 Optimal control theory

In any optimization problem there is a particular quantity, or cost, $J$, that must be optimized. A complete description of a generic cost function in terms of $n$ state variables, $\mathbf{x}(t)^1$, and $m$ control variables $\mathbf{u}(t)$ is as follows

$$
\min J(\mathbf{x}, \mathbf{u}, t) = \Phi(\mathbf{x}(t_f), \mathbf{u}(t_f), t_f) + \int_{t_0}^{t_f} L(\mathbf{x}(t), \mathbf{u}(t), t) \, dt. \tag{2.1}
$$

The cost function consists of terminal costs, $\Phi(\mathbf{x}(t_f), \mathbf{u}(t_f), t_f)$ at final time and the time-varying costs given by the Lagrangian $L(\mathbf{x}(t), \mathbf{u}(t), t)$ evaluated between the initial and final times, $t_0$ and $t_f$, respectively. In addition to these terms, a system of differential constraints,

$$
\dot{\mathbf{x}} = f(\mathbf{x}(t), \mathbf{u}(t), t), \tag{2.2}
$$

may be imposed on the system to describe the time-evolution of the state variables.

---

1In this work, vectors will be indicated by the bold characters.
Such equations determine the time-evolution in terms of the present state \( x(t) \) and the control input \( u(t) \). These constraints can be augmented to the cost function utilizing the Lagrange multipliers, \( \lambda(t) \) and by substituting the Hamiltonian,

\[
H(x, u, t) = L(x(t), u(t), t) + \lambda^T f(x(t), u(t), t).
\]  \tag{2.3}

This results in the augmented cost function

\[
J_a(x, u, t) = \Phi(x(t_f), u(t_f), t_f) + \int_{t_0}^{t_f} \left( H(x, u, t) - \lambda^T \dot{x} \right) dt.
\]  \tag{2.4}

The \(-\lambda^T \dot{x}\) term Equation (2.4) can be integrated out by parts to yield

\[
J_a(x, u, t) = \Phi(x(t_f), u(t_f), t_f) - \lambda^T (t_f) x(t_f) + \lambda^T (t_0) x(t_0)
+ \int_{t_0}^{t_f} H(x, u, t) + \dot{\lambda}^T x dt.
\]  \tag{2.5}

This now contains the full dynamic information of the optimization problem as well as the initial and final boundary conditions, \( \lambda^T (t_0) x(t_0) \) and \( \lambda^T (t_f) x(t_f) \), respectively. For convenience, the functional dependencies will be omitted in the following derivations. The reader should be aware of the implicit time dependence of \( \lambda, x \) and \( u \) as well as the dependencies of the other terms in Equation (2.5).

Near a stationary point on the optimal path, the infinitesimal variation, \( \delta J_a \), in the cost function with respect to the variation in variables \( \lambda, x, u \) and \( t \) should be zero, that is

\[
\delta J_a = \left[ \frac{\partial \Phi}{\partial x} - \lambda^T \right] \delta x + \left[ \frac{\partial \Phi}{\partial t} + H \right] \delta t_f + [\lambda^T \delta x]_{t=t_0}
+ \int_{t_0}^{t_f} \left( \frac{\partial H}{\partial x} + \dot{\lambda}^T \right) \delta x + \frac{\partial H}{\partial u} \delta u \, dt = 0.
\]  \tag{2.6}

As each of the terms are independent, for this equation to be satisfied each of the individual terms must be zero. Considering the integral terms first, the coefficient of \( \delta x \) can be zero iff either \( \delta x(t) = 0 \) for all \( \delta t \) and control inputs \( \delta u(t) \) or if

\[
\dot{\lambda}^T = - \frac{\partial H}{\partial x}.
\]  \tag{2.7}
By similar reasoning this implies

$$\frac{\partial H}{\partial u} = 0.$$  \hspace{1cm} (2.8)

The above Equations (2.2), (2.7) and (2.8) form the set of necessary conditions. The remaining terms:

$$\left[\left(\frac{\partial \Phi}{\partial x} - \lambda^T\right)\delta x\right]_{t=tf} + \left[\lambda^T \delta x\right]_{t=t_0} + \left[\frac{\partial \Phi}{\partial t} + H\right] \delta t_f = 0$$  \hspace{1cm} (2.9)

provide information for the boundary conditions in the optimal control problem. However, the conclusions depend on whether the initial and final states ($x(t_0)$ and $x(t_f)$) are free or fixed, these cases will be examined in Section 2.1.1 below.

### 2.1.1 Boundary conditions

The boundary conditions in an optimal control problem can either be free or fixed. Further, the terminal time $t_f$ can also be free. This leads to several classes of problems which place different conditions on $\lambda(t_0)$ and $\lambda(t_f)$. Using Equation (2.9) and exploring the variations $\delta\{x_0, x_f, t_0, t_f\}$ the boundary conditions for the problem can be deduced. In this work, no problems in which the initial state and time are not specified will be covered so there will be no discussion of these cases (these terms will be strictly zero), for more details refer to Refs [7].

**Case 1**: Fixed final state. In this case the final state of the system is fully prescribed and therefore the variation $\delta x(t_f) = 0$. This reduces Equation (2.9) to:

$$x(t_f) = \hat{x}(t_f), \quad (n \text{ boundary conditions})$$  \hspace{1cm} (2.10)

where $\hat{x}(t_f)$ is the prescribed final state and $x(t_f)$ is the final state of the system on the optimal trajectory.

**Case 2**: Free final state. In this case the final system state is unspecified. This means that the variation $\delta x(t_f)$ is no longer strictly zero and is instead arbitrary. This means that to satisfy Equation (2.9) the following boundary condition is required:
\[ \lambda(t_f) = \left[ \frac{\partial \Phi}{\partial x} \right]_{t=t_f} \cdot \text{(n boundary conditions)} \]  \hspace{1cm} (2.11)

**Case 3**: Free final time. In the event the final time is free, the variation \( \delta t_f \) is no longer zero, therefore the following statement must be true for Equation (2.9) to be zero:

\[ H - \left[ \frac{\partial \Phi}{\partial t} \right]_{t=t_f} = 0. \text{ (1 boundary condition)} \]  \hspace{1cm} (2.12)

The cases above are not mutually exclusive, a free time problem could be formulated in which the final state is not specified in which case Equations (2.11) and (2.12) would be used. Or, perhaps only some of the state variables have boundary conditions prescribed at the final time (which could also be free). This means that for a given optimization problem, some combination of the conditions given above must be used to have enough boundary conditions to solve it. Together with the necessary conditions (restated below) almost every optimal control problem can be solved. There is, however, an additional class of problems where the necessary and boundary conditions are not sufficient. This occurs when the control parameters are bounded (e.g. rocket thrust) and the necessary condition \( \frac{\partial H}{\partial u} = 0 \) is not a function of \( u \). In this case no algebraic relationship can be obtained for \( u(t) \) and the *Pontryagin’s minimum principle* [4] must be used.

\[
\begin{align*}
\text{necessary conditions:} & \\
\dot{x} &= \frac{\partial H}{\partial x} = f, \hspace{1cm} n \text{ differential equations} \\
\dot{\lambda} &= -\frac{\partial H}{\partial x}, \hspace{1cm} n \text{ differential equations} \\
0 &= \frac{\partial H}{\partial u}, \hspace{1cm} m \text{ algebraic equations.}
\end{align*}
\]  \hspace{1cm} (2.13)

### 2.1.2 Pontryagin’s minimum principle

Suppose the control variable \( u(t) \) is bounded to some permissible region, (for example, the thrust generated by a rocket) then the necessary condition, \( \frac{\partial H}{\partial u} = 0 \), will no longer be sufficient to constrain the control variable to lie in its permissible region.
Pontryagin [4] showed that instead the condition

\[ H(x^*, u^*, \lambda^*, t) \leq H(x^*, u, \lambda^*, t) \quad \text{all admissible } u, \]  

(2.14)

where * denotes the optimal value and \( u = u^* + \delta u \) must be satisfied to ensure optimality. That is to say, variation in the optimal control at time \( t \) while the state and co-state continue to be optimal will increase the value of the Hamiltonian. This requirement is called Pontryagin’s minimum principle which states formally: “the Hamiltonian must be minimized over all admissible \( u \) for optimal values of the state and co-state”. This statement is a powerful tool and it is necessary for deriving the ‘bang-bang’ control profile.

2.2 Solving optimal control problems

For problems in classical physics, for example, the two body problem or a pendulum on a spring, an analytical solution to the optimization problem may be possible. Outside of these historically solved problems, numerical methods must be invoked to obtain solutions. Typically in optimal control, solution methods can be split into two categories: indirect and direct methods.

2.2.1 Direct methods

A direct method transforms the optimal control problem from solving a system of differential equations into a nonlinear programming problem. This is generally done by dividing the interval \([t_0, t_f]\) into \( N \) sub-intervals and providing a guess for the state \( x(t_i) \) and control \( u(t_i) \) at each point \( i \in [1, N] \), with some kind of interpolation between points. In the case of the collocation method the optimal control problem becomes a matter of minimizing the function

\[
\delta_i = x(t_{i+1}) - x(t_i) - \frac{h_i}{6} [f(x(t_i), u(t_i), t_i)
\]

\[ + f(x(t_{i+1}), u(t_{i+1}), t_{i+1}) + 4f(x(t_{i+1/2}), u(t_{i+1/2}), t_{i+1/2})], \]  

(2.15)
which measures the deviation between the parameterized state and its derivative at each point. The functional form of Equation (2.15) depends on the interpolation method used but in this formulation, $h_i$ is the width of the $i$’th interval, $x(t_i)$ is the state and $f$ are the state constraints evaluated at the start, middle and endpoints of the interval. Direct methods like the collocation method greatly reduce the amount of mathematical rigor necessary as only the state constraints are required to solve the problem, however, by parametrizing the control variables, all information about the costates is lost. Therefore, any opportunity to explore the sensitivity of the cost function to variations in the initial state variables is lost as well. For this reason, direct methods will not be used in this work.

2.2.2 Indirect methods

The indirect methods use the above necessary conditions (Equation (2.13)) and the minimum principle to completely define the equations of motion and optimal control policy for the system. This results in a two point boundary value problem where a system of $2n$ differential equations must be solved using the $2n$ boundary conditions derived from Equations (2.10), (2.11) and (2.12). However, the boundary conditions provide no information about the costates, $\lambda(t_0)$, resulting in $n$ unknown parameters for which an initial guess must be supplied. This is a significant challenge when using indirect methods as the necessary conditions are extraordinarily sensitive to the supplied initial guess. For a given initial guess, there is no guarantee the integrated necessary conditions will satisfy the terminal boundary conditions let alone produce the optimal solution to the problem.

To get around this, there are a number of different programming techniques that, through the use of some feedback response procedure, can iteratively improve upon an initial guess until the boundary conditions are satisfied within some numerical tolerance. The shooting method (see [11] or others) is one such example and it is given in Algorithm 1. While this algorithm can iteratively improve the initial guess, convergence is usually only achieved if the initial guess is approximately equal to the optimal solution [24].

This is the primary issue with indirect methods; a great deal of understanding of
the physical and mathematical relationships is required to obtain a good initial guess. Further, deriving the co-state equations and optimal control policy can take a large amount of work when problems have a highly coupled system state-equations and multiple control inputs. The following chapter presents the heuristic optimization algorithms that will be used to overcome the challenges of initial guess generation.

**Algorithm 1**: Pseudo code for a shooting method algorithm. The $\delta$ represents some correction factor derived from the error $\Delta$ and the system Jacobian.

**Result**: $\lambda(t_0)$

\[ i = 0; \]

**while** $\Delta > \epsilon$ **do**

\[ \text{Guess } \lambda^i(t_0); \]

\[ \text{Integrate necessary conditions from } t_0 \rightarrow t_f \text{ using } \hat{x}(t_0) \text{ and } \lambda(t_0); \]

\[ \text{Compute error } \Delta = x(t_f) - \hat{x}(t_f); \]

\[ \text{Update } \lambda^{i+1}(t_0) = \lambda^i(t_0) + \delta; \]

\[ i = i + 1; \]

**end**

\[ \lambda(t_0) = \lambda^i(t_0); \]
Chapter 3  |  Metaheuristics

This chapter will discuss two metaheuristics which will be used to generate guesses for $\lambda(t_0)$ to solve optimal control problems via indirect methods. Metaheuristics are a subset of mathematical optimization algorithms which excel at generating sufficiently good solutions to optimization problems in the absence of a priori solution information [26]. This work will focus on the Particle Swarm Optimization (PSO) [19] and the more recently developed Firefly Algorithm (FA) [27] metaheuristics to drive the search for the unknown parameters. The FA, still in its infancy, has had little to no application in optimal control theory; however, PSO has met significant success in a diverse range of disciplines. One such example is the field of trajectory optimization where it has been used to deduce time-optimal reorientation maneuvers [22, 28], multiple-burn rendezvous [29] and to compute optimal trajectories between Lagrange points in the circular restricted three body problem [20, 30].

3.1 Particle swarm optimization

Particle swarm optimization is a relatively recent field of interest in metaheuristic algorithms. In a trend with many other metaheuristics it has evolved from a biological analogy to model a particular behavior, in this case social interaction and the influence and spread of information throughout a large (swarm) population [19]. It has a number of formulations which treat the spread of information differently, these will be discussed below. The methodology is quiet simple, an objective function is defined which gives some feedback as to the quality of the information possessed by each member of the
population. Then a population of $N_p$ particles is initialized randomly in a bounded search region with $d$ coordinate axes representing the unknown parameters being optimized. Each particle’s position, $\{x_1, ..., x_d\}$, in this region is therefore a guess at the optimal values of these unknown free parameters. The particles then proceed to iteratively communicate and move in this region according to some set of rules with the goal of finding the best realization of the objective function. Iteration continues until a maximum number of steps is reached or some convergence criteria is satisfied.

Psuedo code for the $gbest$ PSO algorithm used in this work is given in Algorithm 2.

The most common (and original) implementation of PSO is the $gbest$ or, Global best PSO. Two variables $pbest$ and $gbest$ are defined which influence the movement of particles in the swarm. The first, $pbest$ (a $N_p \times 1$ vector), tracks the best solution each individual particle has obtained so far while the second, $gbest$, is the best solution obtained by any particle in the swarm. Information is shared globally with the whole swarm and the position of each particle is updated on each iteration according to the following policy:

$$
    v_i^{(t+1)} = \alpha v_i^{(t)} + r_1 \phi_1 (p_i - x_i^{(t)}) + r_2 \phi_2 (g - x_i^{(t)}),
$$

$$
    x_i^{(t+1)} = x_i^{(t)} + v_i^{(t+1)},
$$

(3.1)

where $v_i$ is the velocity of the $i$'th particle and $x_i$ is its position, $p$ and $g$ are short-hands for $pbest$ and $gbest$, respectively. The $r_{1,2}$ terms are random variables with uniform distribution on $[0, 1]$, $\phi_{1,2}$ are the associated acceleration coefficients (constant) and $\alpha$ is analogous to the inertial weight of the particle, scaling the particle’s velocity on the prior time-step [31]. In most standard implementations $\alpha = 0.7298$ and $\phi_1 = \phi_2 = 1.4961$ are used [31,32].

A simple variation of the $gbest$ PSO swarm is the local best or, $lbest$ implementation. In this case particles only compare against adjacent neighbors (adjacency can be defined numerous ways but typically a lattice, or topology, of particles is constructed) and a local best particle is used for comparison. The local best formulation tends to converge slower than $gbest$, however, it is less susceptible to becoming trapped in a local minimum [31]. This implementation is not necessarily exclusive with the $gbest$ method and both can be used in conjunction. A further variation is the fully informed particle swarm (FIPS) [33] in which $pbest$, $lbest$ and $gbest$ are replaced with
the average of the best solutions found by the particles. This formulation results in a more random search but has been shown in Ref [33] to improve the optimization. In this work, the canonical $g_{best}$ particle swarm will be used. In many of the problems examined in the later sections, there is no guarantee of an unique global minimum hence the benefits of a $l_{best}$ formulation will be wasted at the expense of increased run-time.

A common issue with PSO is that of stagnation, where particles converge to a point too quickly without sufficiently exploring the search medium. This manifests as an unchanging value in the objective function over successive iterations as the particles exploit the area they are in rather than exploring further to locate a more optimal solution. Balancing the exploration verses exploitation components of a metaheuristic algorithm is essential to ensuring a good problem solution is obtained on every use. The firefly algorithm presented below attempts to encourage exploration
Algorithm 2: Pseudo code for $g\text{best}$ Particle-Swarm Optimization.

**Result:** $g$

initialisation;

Define objective function $f(x)$, $x = (x_1, ..., x_d)^T$;

Randomly generate initial population $x_i$ ($i = 1, 2, ..., N_p$);

while $t \leq \text{MaxGeneration}$ do

    for $i = 1 : n$ all particles do

        Compute objective function $f(x_i)$ at each $x_i$;

        if $f(x_i) < p_i$ then

            $p_i = x_i$;

        end

        if $f(x_i) < g$ then

            $g = x_i$;

        end

    end

    for $i = 1 : n$ all particles do

        update position according to Equation (3.1);

    end

end

3.2 Firefly algorithm

The firefly algorithm constitutes a further iteration on the swarm optimization concept. First publication of this algorithm was by Yang [27], following this there have been numerous applications to a variety of disciplines; stock market forecasting [34], supply chain optimization [35] and unmanned air vehicle path planning [36] are some examples with a thorough summary of work-to-date by Fister et al in [37]. The firefly metaheuristic is inspired by the mating habits of fireflies, drawn to each other through the brightness of their lights. Similarly to PSO, the algorithm initializes with a population of $n$ fireflies randomly distributed in a bounded $d$-dimensional search space. The objective function is then used as an analogue for light intensity, $I$, with
\[ I = I_0 e^{-\gamma r_{ij}} \]  

(3.2)

where \( \gamma \) is the light absorption coefficient and \( r_{ij} \) is the euclidean distance between the \( i \)'th and \( j \)'th fireflies. Typically, as is the case in this work, it is simpler to just set \( I(x_i) = f(x_i) \) where \( f(x) \) is the objective function being optimized [38].

The movement of the fireflies is then governed by two components; the attractiveness of neighboring fireflies and a stochastic random walk component. The attractiveness is defined as

\[ \beta = (\beta_0 - \beta_{\text{min}}) e^{-\gamma r_{ij}^2} + \beta_{\text{min}}, \]  

(3.3)

while the random walk is given by

\[ W = \alpha \epsilon \otimes \text{sign}(U(0,1) - 1/2). \]  

(3.4)

The parameters \( \beta_0 \) and \( \beta_{\text{min}} \) provide a means of controlling the distance-varying and minimum attractiveness of each firefly, \( \alpha \) is a coefficient to scale the random contribution to the movement and \( \epsilon = |ub - lb| \) where \( ub \) and \( lb \) are the lower and upper bounds on the search space.

Lukasik and Żak performed a thorough investigation of firefly population, and the parameters \( \beta_0 \) and \( \gamma \) [39]. They conclude for most problems an optimal number of fireflies could not be definitively found, \( \gamma \in [0,10] \) was generally appropriate and that \( \beta_0 = 1 \) yielded the best results compared to other values of \( \beta_0 \) on \( [0,1] \). This testing was done with \( \alpha = \{0.001, 0.01, 0.1\} \) on 14 different benchmark problems with 100 trials per function [39]. No commentary is given on the parameter \( \beta_{\text{min}} \) which is neglected in all of Yang’s publications, however, in code\(^1\) given by Yang in [38] and last updated on mathworks in 2011, \( \beta_{\text{min}} \) is included as given Equation (3.3) with a suggested value of 0.2. It is unfortunate that no discussion of this parameter is given as it seems to greatly enhance the performance of the algorithm, particularly at low population numbers. Some analysis of \( \beta_{\text{min}} \) is given in this work in Section 3.4.1.

The firefly algorithm as used in this work is shown in Algorithm 3 and the distance

\(^1\)https://www.mathworks.com/matlabcentral/fileexchange/29693-firefly-algorithm
update step is given as

\[ x_i^{(t+1)} = x_i^{(t)} + \beta (x_j^{(t)} - x_i^{(t)}) + \alpha \epsilon \otimes \text{sign} (U(0, 1) - 1/2). \]  

(3.5)

In [38] Yang proposes scaling \( \alpha \) with iteration using \( \alpha^{(t+1)} = \alpha^{(t)} d\alpha \) where \( d\alpha \in [0.95, 0.97] \), however, in the aforementioned *mathworks* code, a more explicit relationship where \( d\alpha \) is a function of the maximum number of iterations:

\[ d\alpha = \left( \frac{10^{-4}}{0.9} \right)^{\frac{1}{\text{MaxGen}}}. \]  

(3.6)

Using either definition reduces the exploration of the search space as iteration progresses refining the exploitation around what should, at later iterations, be good solution candidates.

It should be noted that both PSO and FA are very similar in implementation. Both follow the same initialization procedure and the only significant difference in the pseudo-code is the nested *for* loop as part of the movement step that the firefly algorithm uses. While these differences seem cosmetic, and it has been argued by Weyland [40] that the differences are negligible, the results presented in Refs [27,38,39] suggest that the algorithm does constitute a significant improvement in performance, at least in solving benchmark optimization problems, relative to existing metaheuristics. However, one should take care in exploring the field of metaheuristics as there are a number of “novel” additions which *do* offer little to no innovation on the concept besides a trivial renaming of terms [41].

The reason for PSO’s success and why certain changes improve performance, are still unclear according to Kennedy [31]. Comparing the distance update procedure of FA and PSO in greater detail, each firefly compares against every other firefly, thus a fully connected topology, however, movement only occurs towards brighter (better solution) fireflies. This in effect creates a dynamic topology where some paths are ignored in favor of clusters of fireflies in regions with better solutions than the current region. Further, instead of being moved by historical performance and the global best solution (in *gbest*, for example), fireflies are moved toward every more attractive firefly, creating a movement similar to FIPS. There is no velocity or inertial component to the movement and because of the minimum attractiveness each firefly has for each
other, distant fireflies are still capable of improving the information they have when
the $(\beta_0 - \beta_{\text{min}}) e^{-\gamma r_{ij}^2}$ term becomes very small. Finally, the most significant change
is the random walk component which encourages exploration as fireflies move on a
trajectory towards better solutions. In this work, as a further iteration on the FA
concept, Optimal foraging theory (discussed below) will be used to improve upon the
random walk component of the firefly algorithm.

Algorithm 3: Pseudo code of the Firefly Algorithm [27].

Result: $x_1$

initialisation;
Define objective function $f(x)$, $x = (x_1, \ldots, x_d)^T$;
Randomly generate initial population $x_i$ $(i = 1, 2, \ldots, N_p)$;

while $t \leq \text{MaxGeneration}$ do
    for $i = 1 : n$ all $n$ particles do
        Compute objective function $f(x_i)$ at each $x_i$;
    end
    rank the fireflies;
    for $i = 1 : n$ all $n$ fireflies do
        for $j = 1 : n$ all $n$ fireflies do
            if $f(x_j) > f(x_i)$ then
                compute firefly separations, $r_{ij}$;
                move firefly $i$ towards $j$ using Equation (3.5);
            end
        end
    end
end

3.3 Optimal foraging theory

Movement, or spatial ecology seeks to explain the large-scale movements natural
entities undertake in the search for food, mates or any other imaginable need. As
an extension to this, optimal foraging theory states that organisms have evolved in
such a way that, when foraging, a search strategy is adopted that maximizes benefit
for the lowest cost input [42]. Stochastic optimal foraging theories, in which animals undertake random-walks, have met significant success in modeling the movements of predators in search of prey [25, 42]. Typically such models rely on the diffusive, Brownian motion (gaussian), to describe the movement of searchers [25]. However, these models fail to account for the directional persistence animal movements exhibit and will typically revisit the same site many times [43]. Recent studies have shown that search strategies may not be diffusive in nature but instead super-diffusive, with the movement of searchers governed instead by the Lèvy distribution [44–46], a fat-tailed exponential probability distribution. A comparison of movement according to a Lèvy flight and Brownian motion is shown in Figure 3.1 in which the same distance is traversed by each searcher. The Lèvy flight enables a searcher to cover a significantly larger area with the large steps being punctuated by long periods of intense exploitation of the new region. By comparison the Brownian distribution exploits mostly the same area repeatedly. It is for this reason that the Lèvy distribution should be amenable to a swarm optimization algorithm. Swarm members in the vicinity of a favorable solution will occasional jump large distances away. In the event the new region is better, the other swarm members will move towards them and if it is not it will be attracted back to its original position, likely utilizing a large number of smaller steps along the way.
Figure 3.1. (Top) Comparison of a Lèvy flights versus a Brownian random walk with the same distance traveled in each random walk. (Bottom) Distance traveled in each random walk as a function of step length.
The Lévy distribution features characteristic heavy-tails giving rise to large, but improbable, step lengths in between a number of small movements. The Lévy-flight foraging hypothesis is an optimal foraging theory which posits that searchers should adopt search strategies known as Lévy flights to optimize their yield [45–47]. The Lévy distribution has characteristic function

$$\phi(t) = \exp [it\nu - |ct|^\alpha (1 - i\beta \text{sign}(t)\Phi)],$$

with

$$\Phi = \begin{cases} \tan(t\pi/2), & \alpha \neq 1 \\ -(2/\pi)\ln|t|, & \alpha = 1 \end{cases}.$$  

However, it is more commonly described in terms of the exponent $\alpha$ and probability density function for the step length of the form

$$P(l) \sim l^{-\mu},$$

with $1 < \mu \leq 3$ and $\mu = \alpha + 1$. The parameters, $\nu$, $\beta$, $c$ and $\alpha$ represent the translation, asymmetry, scale, and distribution shape, respectively [25].

There is growing evidence of its efficacy in describing animal foraging patterns; a recent empirical study of 14 different aquatic species [46] showed significant agreement between the hypothesis and their observed foraging patterns exhibiting the heavy-tailed movement distributions associated with Lévy flights. Further, the hypothesis has been successfully applied to a number of different insect species [48–50] in which it is suggested the movement followed a $\mu \approx 2$ Lévy flight. The conclusions of these works are similar and form a more explicit restatement of the hypothesis: in the presence abundant targets Brownian motion is sufficient to describe foraging patterns, however, in regions with sparsely distributed targets, animals shift behavior to search according to Lévy flights. The free parameters in an optimal control problem can be thought of as targets in a sparsely populated region. Generally, there exists either a unique (i.e. sparse) set of optimal parameters, or an optimal relationship between parameter sets (e.g. $\lambda_1 = 2\lambda_2$) hence, a search method motivated by the Lévy flight foraging hypothesis should be the optimal means of finding these free parameters.
parameters. It then follows, that by changing the the uniform distribution in the original Firefly algorithm to a Lévy distribution, a more optimal method for searching for free parameters will be developed. This algorithm shall be referred to as the Lévy flight firefly algorithm and it is discussed in Section 3.4 below.

### 3.4 Lévy flight firefly algorithm

The Lévy Flight Firefly Algorithm (LFA) constitutes a small modification to the original FA algorithm with the random-walk component of Equation (3.5) being replaced with

\[
x_{i}^{(t+1)} = x_{i}^{(t)} + \beta (x_{j}^{(t)} - x_{i}^{(t)}) + \alpha \epsilon \times \text{Lévy}(\mu),
\]

where \( \text{Lévy}(\mu) \) represents a Lévy distribution with Lévy parameter \( \mu \). This change has been made in Ref [51] and in the Eagle Search Algorithm [52], another metaheuristic identical to the firefly algorithm but with the above substitution. Benchmarks in both these publications show an improvement in terms of performance compared to the base firefly algorithm despite being the simple substitution of a probability distribution, however, no application besides benchmarking functions has yet been attempted. Based on the work of Viswanathan, Lévy flights [25,47] occur for \( 1 < \mu < 3 \) (discussed in detail in Section 3.3), Yang in each of the aforementioned papers uses \( \mu = 1.2 \).

#### 3.4.1 Testing firefly parameters

The benchmarks performed in Ref [39] provide guidance on the parameters \( \gamma \) and \( \beta_0 \), however, no commentary on \( \alpha \) and \( \beta_{\text{min}} \) is given. Further, in Refs [51] and [52] little justification for the \( \mu = 1.2 \) assignment is given. This section will explore the changes in algorithm performance using a variety of parameter values for minimizing a 4-d Ackley function [53]. The Ackley function is an \( n \)-dimensional test function which features an infinite number of local minima and maxima and a unique global minimum located at the origin. A contour plot of the 2-d Ackley function is shown in Figure 3.2 and it is defined as
Figure 3.2. Contours of the 2-d Ackley function in the region \([x, y] \in [-4, 4]\)

Ackley\((x) = -a \exp \left[ -b \sqrt{\frac{1}{d} \sum_{i=1}^{d} x_i^2} \right] - \exp \left[ \frac{1}{d} \sum_{i=1}^{d} \cos c x_i \right] + a + \exp [1], \quad (3.11)\)

where \(a = 20, b = 1/5\) and \(c = 2\pi\) in the present tests. To reduce the number of tests will be performed with only one independent parameter while the rest will be held constant\(^2\).

### 3.4.1.1 Investigating randomness

The parameter \(\alpha\) weights the contribution of the random-walk component in the position update policy. Four different values of \(\alpha\) will be tested, \(\{0.01, 0.05, 0.1, 0.25\}\) and in all tests an iteration dependent \(\alpha^{(t)}\) will be used as defined in Equation (3.4). Figure 3.3 shows the objective function at each iteration for the four values of \(\alpha\) tested. It is clear that for small \(\alpha\) convergence is achieved rapidly but that too small

\(^2\)Seven different values of \(\alpha, \beta_{\text{min}}\) and \(\lambda\) are being tested at four different population numbers. A complete investigation would require 1372 different tests per benchmark function.
a value, as is the case for $\alpha = 0.001$ is detrimental to performance. The results of these tests were obtained from averaging 100 trials with $\beta_{\text{min}} = 0.05$, $\mu = 1.2$ and $N_p = 10, 20, 30$ and 40 fireflies. As firefly population increases, the point at which the $\alpha = 0.001$ line bottoms out improves. This is intuitive as the $\alpha$ parameter facilitates exploitation, for a large population, more exploitation will occur naturally hence, faster convergence will not impede exploitation.

![Graphs showing performance comparison](image)

**Figure 3.3.** Comparison of Lèvy Firefly Algorithm performance for different values of $\alpha$ when minimising a 4d Ackley function.

### 3.4.1.2 Investigating minimum attractiveness

The minimum attractiveness of the fireflies to each other is controlled by the parameter $\beta_{\text{min}}$. Though typically omitted from publications, the results in Figure 3.4 show the significant improvement in performance that can be gained by including a small amount of $\beta_{\text{min}}$ in the definition of firefly attractiveness. At low population numbers,
in this case $N_p = 10$, more $\beta_{\text{min}}$ is required to match the convergence of the higher-population number trials, however, it also appears that at low population numbers the minimum achieved is better, but not by a full order of magnitude. For large firefly populations, too much $\beta_{\text{min}}$ produces an undesired clustering of fireflies around local minima but does not degrade the mean performance below that of $\beta_{\text{min}} = 0$.

![Graphs showing comparison of Lévy Firefly Algorithm performance for different values of $\beta_{\text{min}}$](image)

**Figure 3.4.** Comparison of Lévy Firefly Algorithm performance for different values of $\beta_{\text{min}}$ using (a) 10, (b) 20, (c) 30 and (d) 40 fireflies to find the minimum of a 4d Ackley function.

### 3.4.1.3 Investigating the Lévy parameter

The Lévy distribution parameter $\mu$ controls the shape of the distribution. For $\mu \in (0, 2]$, Lévy flights occur with the upper limit corresponding to a Gaussian distribution [25]. The limiting $\mu = 0.25$ and 2.0 cases perform poorly for every population size, however, $\mu = 1.25$ and 1.5 perform consistently well supporting Yang’s use of $\mu = 1.2$. For comparison the FA, using $\alpha = 0.15$, is included.
Figure 3.5. Comparison of Lèvy Firefly Algorithm performance for different values of $\lambda$ when minimising a $4d$ Ackley function.

Table 3.1. Parameters used in the FA and LFA algorithms. $\mu$ governs the Lèvy distribution and is hence only used in LFA.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>0.25</td>
</tr>
<tr>
<td>$\beta_0$</td>
<td>1</td>
</tr>
<tr>
<td>$\beta_{\text{min}}$</td>
<td>0.05</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>1</td>
</tr>
<tr>
<td>$\mu$</td>
<td>1.5</td>
</tr>
</tbody>
</table>

A set of generally good firefly parameters are shown in Table 3.1. These have been found to give a reasonable balance of convergence rate with sufficient exploration to not become trapped in local minima. In the following chapters these parameter
values will be used to drive both the LFA and FA algorithms when generating guesses for \( \lambda(t_0) \) in each of the optimal control problems. In the next chapter, a method will be presented to handshake the optimal control problem discussed in Chapter 2 with the metaheuristics presented here.
Chapter 2 presented the mathematical tools required to solve optimal control problems in the form of the necessary conditions and the resulting two-point boundary value problem where guesses had to be supplied for the initial co-state values $\lambda(t_0)$. Chapter 3 defined the algorithms that will drive the search for the unknown optimal initial co-state values required to solve each problem. This chapter will combine that knowledge and develop a search methodology to find the optimal $\lambda(t_0)$. This is presented in Section 4.1. The subsequent sections will discuss two Zermelo-like optimization problems. Zermelo’s navigation problem (Section 4.2) deals with finding a path that minimizes the travel time between two points under the influence of an external force-field. The problem was originally proposed in the context of the optimal steering of a ship under the influence of some wind or current distribution. The generality of the Zermelo problem, that is, the motion of a controllable object under an external, spatially varying force field, gives rise to a number of analogue problems and extensions. The minimum-time orbit injection of a spacecraft starting on the lunar surface is just one example. This problem is presented in Section 4.3.

4.1 Search methodology

As stated above, the metaheuristics will be used to arrive at good guesses for $\lambda(t_0)$. In either of the algorithms discussed, a population of $n$ workers are initialized randomly...
throughout a search region. For most problems of interest, the search area will be a hyper-dimensional region with $d$ coordinate axes for each of the unknown $\lambda(t_0)$ values. The only remaining question is to determine the efficacy of each $\lambda(t_0)$ so that the metaheuristics can search for the optimal set. The easiest way to do this is to just integrate the necessary conditions (see Equation (2.13)) for a given initial guess and check whether all the constraints and boundary conditions are satisfied. In effect, perform a single iteration of the shooting method for each of the $N_p$ particles in the free parameter space every iteration of the metaheuristic. This is spelled out sequentially below:

1. Guess $\lambda(t_0)$ randomly for each particle
2. Integrate necessary conditions from $t_0$ to $t_f$
3. Compute fitness function, $f(\lambda(t_0))$
4. Update initial guesses using the metaheuristic’s update policy and repeat 2 – 4 until a convergence criteria is met or the maximum permitted iterations through the metaheuristic has been reached.

For most of the problems discussed in this work, the fitness function will be along the lines of

$$f(\lambda(t_0)) = \sum_{i=1}^{d} \left( \frac{x_i(\lambda(t_0), t_f) - \hat{x}_i(t_f)}{w_i} \right)^2, \quad (4.1)$$

where $x_i(\lambda(t_0), t_f)$ corresponds to the final state of the system obtained using the integrated $\lambda(t_0)$ guess, $\hat{x}(t_f)$ are the prescribed boundary conditions and terminal constraints and $w$ is a vector of weights. These weights do not necessarily have to be employed but can be convenient for normalizing the system if the terminal quantities are of differing scales (e.g km/s and radians). However, it may also be the case that certain terminal properties are correlated (e.g position and velocity) in which case weights could be adjusted to incentivize improvement in a particular quantity.

Generally speaking, metaheuristics are useful arriving at good solutions to a problem, however, they will rarely be able to find the optimal solution. In other
words, they will rarely be able to satisfy the problem constraints to numerical precision. Therefore, in this work, these algorithms will be used in conjunction with Matlab’s boundary value problem solver \texttt{bvp4c} [54]. The \texttt{bvp4c} solver is a fourth order accurate, collocation based method capable of delivering optimal solutions to a great deal of numerical precision. However, it is extremely sensitive to the initial guess and will typically diverge in the presence of poor initial guesses. Hence, to obtain the optimal solution to each of the discussed in this work, good free parameter guesses obtained via each metaheuristic will be used as initial problem solutions for \texttt{bvp4c}.

### 4.2 Zermelo’s problem

This section will explore finding the minimum time path of a ship with constant speed under the influence of a uniform current oriented parallel to the y-axis. As this is a time minimization problem with no terminal constraints, the cost function is simply

\[
\min J = \int_{t_0}^{t_f} dt
\]  

subject to the state constraints

\[
\begin{cases}
\dot{x} = f = V \left[ \cos \theta + u(y) \right] \\
\dot{y} = V \sin \theta
\end{cases}
\]  

where \( u(y) \) is the velocity contribution from the velocity field given by \( u(y) = \frac{y}{h} \), \( V \) is the ship speed and \( \theta \) is the steering angle. The system Hamiltonian can be determined from equation (2.3) and is as follows

\[
H = 1 + \lambda_x \left( V \left[ \cos \theta + u(y) \right] \right) + \lambda_y \left( V \sin \theta \right).
\]  

Using Equations 2.13 the necessary conditions for optimality are

\[
\frac{\partial H}{\partial \lambda} = \dot{x} = f,
\]  

31
\[-\frac{\partial H}{\partial x} = \dot{\lambda} = \begin{cases} \dot{x}_x = 0 \\ \dot{\lambda}_y = \lambda_x \frac{V}{h} \end{cases} \quad (4.6)\]

with optimal control policy

\[
\frac{\partial H}{\partial \theta} = 0
\]

\[
V(\lambda_y \cos \theta - \lambda_x \sin \theta) = 0
\]

\[
\tan \theta = \frac{\lambda_y}{\lambda_x}.
\]

Because this is a free final time problem an additional constraint, \(H(t_f) = 0\) is imposed to minimize \(H\). Values of \(V = 1\) and \(h = 1\) will be used with prescribed initial and final states

\[
\begin{bmatrix} \hat{x}(t_0) \\ \hat{y}(t_0) \end{bmatrix} = \begin{bmatrix} 3.6587 \\ -1.8637 \end{bmatrix}, \quad \begin{bmatrix} \hat{x}(t_f) \\ \hat{y}(t_f) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.
\]

(4.8)

Ref. [7] provide a closed-form analytical solution to this problem. The solution has \(J^* = 5.4641\) s with initial time Lagrange multipliers \(\lambda_x^*(t_0) = 0.5\) and \(\lambda_y^*(t_0) = -1.866\).

Zermelo’s problem is not a particularly complicated set of equations. In spite of this, the system is still extraordinarily sensitive to small changes in the initial conditions. Figure 4.1 shows three trajectories for different \(\lambda(t_0)\) values: the optimal solution (obtained analytically) and the optimal solution perturbed by \(\pm 2\%\) in each \(\lambda(t_0)\), with the final time held constant at the optimal value. These small changes result in a shift of \(\approx 0.6\) units from the required terminal position. In addition to this, the perturbed initial guesses do not converge to the optimal values when used in a shooting method algorithm (when using the optimal \(t_f\) guess)\(^1\).

### 4.2.1 Metaheuristic results

The Zermelo problem has three unknown parameters, \(\lambda_x(t_0), \lambda_y(t_0)\) and \(t_f\). The method given in Section 4.1 discussed the search for the best guess strictly in terms

\(^1\)The particular shooting method required a tolerance of \(10^{-5}\) be satisfied in under 500 iterations. It is possible that it would converge using another indirect method algorithm.
of the unknowns $\lambda(t_0)$, however, there is no reason the unknown $t_f$ (or in fact any free parameter) cannot be augmented to the search space and found in the same way. Using the FA, LFA and PSO algorithms with a maximum of 1000 iterations permitted through each algorithm, initial guesses for the three free parameters $\lambda_x(t_0), \lambda_y(t_0)$ and $t_f$ were obtained. For the firefly related algorithms $N_p = 8$ workers were used while for the PSO, $N_p = 8$ and $N_p = 30$ workers were used. Bounds of $[-4, 4]$ and $[0, 10]$ were placed on $\lambda_0$ and $t_f$, respectively, and the objective function was defined as $\min \epsilon = |x(t_f) - \hat{x}(t_f)|$ where $\hat{x}(t_f) = [0, 0, 0]$, the additional zero coming from the
$H(t_f) = 0$ constraint. To decrease run-time wasted on improving bad solutions, a proximity based convergence criteria which terminates evaluation when the norm of the standard deviation of best $d$ workers is less than $10^{-3}$ was used, explicitly:

$$
\left\| \sum_{i=1}^{d} \frac{(x_{i,j} - \mu_j)^2}{d-1} \right\| \leq 10^{-3} \quad j = 1, 2, 3 \quad (4.9)
$$

where $\mu$ is the mean position (in the free parameter space) of the best $d$ workers and the $j$ index corresponds to the free parameters. This is the same convergence criteria often used in the Nelder-Mead function minimization algorithm [55].

![Figure 4.2.](image)

Figure 4.2. Evolution of LFA guesses with iteration number.

Figure 4.2 shows the trajectory of the best $\lambda(t_0)$ obtained by the LFA as a function
of iteration number i.e the dotted black line corresponds to the zeroth iteration and the solid black line corresponds to the 500’th iteration. Convergence to a solution very close to the optimal value occurs at some point between 100 and 200 iterations with the final 300 slowly improving the $\lambda(t_0)$ until it closely matches the analytical solution. Using the same shooting method as above the 500’th iteration $\lambda(t_0)$ values converge in one iteration.

To test the viability of each metaheuristic (the FA, LFA and PSO), a $N = 1000$ Monte Carlo process is used. The averaged best guess results obtained from this are shown in Table 4.1. This table includes the best initial guesses for $\lambda(t_0)$ and $t_f$ (in terms of the optimal time) provided by each method. In the bounds specified there exists a local minimum for $t_f = 0$ which frequently traps PSO, the rate at which the algorithms avoid this minimum is shown in the convergence rate column in Table 4.1.

The contours of the $\lambda_x/\lambda_y$ space are plotted for fixed values of time in Figures 4.3. For the $t_f = 0$ s plot (top panel), the fitness function features a wide band where the fitness function is below 5. Compared with the $t_f = 5.4641$ s plot, this minimum is significantly better than the majority of solutions and is far easier to locate than the extremely narrow valley corresponding to the optimal solution so it is understandable that the fast-converging PSO trials would have a propensity to become stuck here. For contrast, the $t_f = 10$ s plot features overwhelmingly large fitness function values which explains why no trials become trapped at this bound.
The firefly algorithms significantly out-perform the PSO methods in terms of agreement with the analytical solution (\(J^*\)) and the convergence rate. In the case of PSO\(_8\) this is largely due to an insufficient number of particles being used in the search and while the mean \(\lambda(t_0)\) values are close to the analytical solution, the uncertainty is on the order of \(\pm 100\%\) so, in general, a reliable solution is not obtained when the algorithm terminates. Similar statements are applicable to PSO\(_{30}\) though as expected, the addition of 22 workers does improve the results. The inclusion of an 8 worker
particle swarm was to illustrate that the firefly algorithms can perform comparably to PSO with far fewer workers (8 workers is generally nonsensical for PSO). For future problems presented in this work a general convention of $N_p \approx 10d$ particles, where $d$ is the number of free parameters, will be adopted.

Table 4.1. Results of an $N = 1000$ Monte Carlo simulation of the FA, LFA, and PSO algorithms for the Zermelo problem. The PSO results have been filtered (indicated by *) to remove the unfeasible, $t_f = 0$ minimum which significantly corrupts the statistics. The raw statistics for these are also shown. The convergence rate corresponds to the number of trials which do not get trapped in the $t_f = 0$ local minimum.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>$\lambda_x$</th>
<th>$\lambda_y$</th>
<th>$J/J^*$</th>
<th>convergence rate (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FA$_8$</td>
<td>0.5±0.3</td>
<td>-1.8±0.4</td>
<td>0.99±0.08</td>
<td>99.4</td>
</tr>
<tr>
<td>LFA$_8$</td>
<td>0.501±0.009</td>
<td>-1.87±0.03</td>
<td>0.999±0.002</td>
<td>100</td>
</tr>
<tr>
<td>PSO$_8^*$</td>
<td>0.6±0.6</td>
<td>-2.2±1.2</td>
<td>0.9±0.3</td>
<td>–</td>
</tr>
<tr>
<td>PSO$_{30}^*$</td>
<td>0.6±0.3</td>
<td>-2.2±0.9</td>
<td>0.95±0.19</td>
<td>–</td>
</tr>
<tr>
<td>PSO$_8$</td>
<td>0.9±0.8</td>
<td>-2±2</td>
<td>0.6±0.5</td>
<td>62.5</td>
</tr>
<tr>
<td>PSO$_{30}$</td>
<td>0.7±0.5</td>
<td>-2.0±1.6</td>
<td>0.8±0.4</td>
<td>81.4</td>
</tr>
</tbody>
</table>

Figure 4.4 shows the mean $x$ and $y$ position obtained from integrating the state and costate equations for each $\lambda(t_0)$ guess (excluding $t_f = 0$ guesses), $3\sigma$ error bars are plotted every $t_f/10$ seconds. The lower panel of each figure shows the residual of $|[x, y]|$ position with respect to the analytical solution. As in Table 4.1, the LFA$_8$ algorithm performs the best with a mean residual of $\approx 10^{-4}$ and much smaller $3\sigma$ bounds compared to the other methods.
Figure 4.4. Averaged state trajectory integrated from initial guesses generated in the $N = 1000$ monte carlo. The lower panel of each figure shows the residual of the states with respect to the analytical solution. 3σ bounds are plotted and the prescribed initial and final states are indicated by the green and red points, respectively. The $x$-error bars in panel (c) have been omitted because they are too large to fit on a reasonable scale.
4.2.2 Using guesses with bvp4c

Table 4.2 shows the results of passing each of the Monte Carlo generated initial guesses through Matlab’s bvp4c function. For PSO₈, 38.2% of guesses result in a singular Jacobian error which is generally associated with a poor initial guess. The majority guesses which diverge are those in which the workers became trapped in the \( t_f = 0 \) minimum. With an increased number of particles, PSO₃₀ had a factor of two fewer singular Jacobian based failures and the mean cost function converged to the analytical optimum. The final \( x \) and \( y \) position are zero to working precision (with zero uncertainty) for all of the methods. Figure 4.5 shows the average trajectory of the successful bvp4c solutions for each of the methods as well as residuals with the analytic solution (3σ error bars plotted). The final state errors for FA₈, LFA₈ and PSO₃₀ are on the order of \( 10^{-7} \) with similar deviations at each time step, while PSO₈ has a mean error of \( 10^{-4} \), with a large error bar on the final time, and for states between \( t_0 \) and \( t_f \).

<table>
<thead>
<tr>
<th>algorithm</th>
<th>( x(t_f) )</th>
<th>( y(t_f) )</th>
<th>( t_f )</th>
<th>failures</th>
<th>warnings</th>
</tr>
</thead>
<tbody>
<tr>
<td>FA₈</td>
<td>0</td>
<td>0</td>
<td>5.464116±15</td>
<td>0.6%</td>
<td>0%</td>
</tr>
<tr>
<td>LFA₈</td>
<td>0</td>
<td>0</td>
<td>5.464116±15</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>PSO₈</td>
<td>0</td>
<td>0</td>
<td>5.4±4</td>
<td>38.2%</td>
<td>38.4%</td>
</tr>
<tr>
<td>PSO₃₀</td>
<td>0</td>
<td>0</td>
<td>5.464116±18</td>
<td>18.6%</td>
<td>69.1%</td>
</tr>
</tbody>
</table>
Figure 4.5. Averaged bvp4c solution for the four algorithms tested. The FA, LFA and PSO₃₀ all converged to the same solution.

4.2.3 Comparison of algorithm performance

The primary flaw in the PSO based methods owes to an increased propensity to become trapped in an unfavorable local minimum, this chance also exists for the firefly based methods, as seen in the 6 failures from the FA algorithm, although the
likelihood appears to be significantly lower. The addition of Lèvy flights to the firefly algorithm further reduced the number of failures (to 0%) with no appreciable increase in run-time, however, given the $N = 1000$ Monte Carlo sample size it is impossible to comment on whether the probability is entirely eliminated or whether this was just a coincidence. Figure 4.6 shows the distribution of each $\lambda(t_0)$ guess obtained by the metaheuristics. The $\text{FA}_8$ and $\text{LFA}_8$ trials are consistently distributed around the optimal solution of $[0.5, -1.866]$. The parabolic structure that a large number of $\text{PSO}_{30}$ guesses lie on correspond to those trapped in the $t_f = 0$ minimum. The reason this shape occurs can be seen in the contour plot given in Figure 4.3.

Statistics on run-time, function evaluations and $\text{bvp4c}$ success rates for each of the methods are shown in Table 4.3. The run-time for the $\text{bvp4c}$ solution was on the order of 0.3 seconds for the guesses given by each of the methods, with poorer guesses requiring more function evaluations therefore, relative to the swarm algorithms, the run-time of this component of solving the problem is negligible. The major benefit to the firefly based methods appears to be in the robustness of the initial guess. With significantly fewer failures, in both the guess generation stage and $\text{bvp4c}$ evaluation, the firefly methods were able to reliably solve the problem close to 100% of the time. However, this robustness came at significant cost in computation time, relative to the particle swarm optimization. The slower convergence of the algorithm, that is, the greater number of iterations required for the fireflies to become located in close proximity to each other resulted in an approximately three-fold increase in run-time due to greater number of function evaluations. This trade off seems worth while as the uncertainty associated with PSO guesses would necessitate repeated trials and analysis, eliminating the run-time benefit it offers.

It is worth noting that when the proximity convergence is removed and the PSO algorithms continued to the maximum permitted iterations, the results for $\text{PSO}_8$ and $\text{PSO}_{30}$ both improved dramatically. The minima found in the guess generation stage were generally, to numerical precision, the same as the analytical solution, however, the number of $\text{bvp4c}$ singular jacobian errors remained approximately the same. Therefore, there is no meaningful performance increase by removing the convergence criteria when compared with the firefly algorithm which was already capable of solving the problem close to 100% of the time as the primary purpose of
the metaheuristics is to obtain good guesses which converge to the optimal solution when paired with an indirect method. Removing the proximity criteria increased run-time to \( \approx 40 \text{s} \) and \( \approx 150 \text{s} \) for PSO_8 and PSO_{30}, respectively, while failing to match the robustness of the firefly methods. Removing the proximity convergence for the firefly algorithms increased run-time to be on par with PSO_8, as they will perform the same number of function evaluations, but this did little to improve upon the already good performance relative to the PSO. The statistics for the PSO methods without a proximity convergence criterion are shown in the lower section of Table 4.3.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Evals.</th>
<th>failures</th>
<th>warnings</th>
<th>run-time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FA_8</td>
<td>( 5.2 \pm 2 \times 10^3 )</td>
<td>0.6%</td>
<td>0%</td>
<td>15.7±11</td>
</tr>
<tr>
<td>LFA_8</td>
<td>( 4.4 \pm 2 \times 10^3 )</td>
<td>0%</td>
<td>0%</td>
<td>14.7±13</td>
</tr>
<tr>
<td>PSO_8</td>
<td>( 0.3 \pm 2 \times 10^3 )</td>
<td>38%</td>
<td>38%</td>
<td>0.9±7</td>
</tr>
<tr>
<td>PSO_{30}</td>
<td>( 1.3 \pm 8 \times 10^3 )</td>
<td>19%</td>
<td>69%</td>
<td>4±3</td>
</tr>
<tr>
<td>PSO^†_8</td>
<td>( 8 \times 10^3 )</td>
<td>34%</td>
<td>1%</td>
<td>40±14</td>
</tr>
<tr>
<td>PSO^†_{30}</td>
<td>( 30 \times 10^3 )</td>
<td>20%</td>
<td>0%</td>
<td>150±50</td>
</tr>
</tbody>
</table>

Table 4.3. Run-time and function evaluation statistics for each of the algorithms solving the Zermelo problem. Failures and warnings correspond to singular jacobian errors and final state residuals not satisfying the required tolerance, respectively. The PSO methods with the † correspond the the PSO methods with the convergence criteria removed.
Figure 4.6. Scatter plot showing each initial guess generated by the metaheuristics for the Zermelo problem. The top panel shows the initial guesses with the convergence criteria active while the bottom panel shows the guesses with it turned off. Guesses which lie along the ‘parabolic’ feature correspond to guesses trapped in the $t_f = 0$ minimum.
4.3 Minimum-time orbit injection

The purpose of this problem is to find the minimum time control history to transfer a spacecraft from the lunar surface to a circular orbit. In this problem a constant thrust acceleration and gravity approximation will be used making it a similar extension to the Zermelo problem, with gravitational acceleration taking on the role of the water current. Consequently, the same tangent steering law will be derived for the control variables in terms of the velocity costates. This is again a minimum-time problem with cost function given in Equation (4.2) subject to the state constraints

\[
\begin{cases}
\dot{x} \\
\dot{y} \\
\dot{u} \\
\dot{v}
\end{cases}
= f = \begin{cases}
u \\
v \\
 a \cos \beta \\
 a \sin \beta - g
\end{cases}
\]

The Hamiltonian for this problem is therefore

\[
H = 1 + \lambda_x u + \lambda_y v + \lambda_u (a \cos \beta) + \lambda_v (a \sin \beta - g).
\]

Using Equation (2.13), the co-state equations are given by

\[
\frac{\partial H}{\partial \mathbf{x}} = \begin{cases}
\dot{\lambda}_x = 0 \\
\dot{\lambda}_y = 0 \\
\dot{\lambda}_u = -\lambda_x \\
\dot{\lambda}_v = -\lambda_y
\end{cases}
\]

with the control parameter $\beta$ expressed in terms of the velocity costates:

\[
\frac{\partial H}{\partial \beta} = 0
\]

\[
\tan \beta = \frac{\lambda_v}{\lambda_u}
\]

From the problem description, the initial and final states are
\[
\begin{bmatrix}
\hat{x}(t_0) \\
\hat{y}(t_0) \\
\hat{u}(t_0) \\
\hat{v}(t_0)
\end{bmatrix}
= \begin{bmatrix}
0 \\
0 \\
0 \\
0
\end{bmatrix}, \quad \begin{bmatrix}
\hat{y}(t_f) \\
\hat{u}(t_f) \\
\hat{v}(t_f)
\end{bmatrix}
= \begin{bmatrix}
100 \\
\mathfrak{m}_{\text{moon}} \\
0
\end{bmatrix}.
\] (4.14)

In contrast to the Zermelo problem, the final \( x \) position of the spacecraft is completely free, which implies that the variation \( \delta x(t_f) \) is no longer strictly zero at the terminal time. Therefore to satisfy \( \delta H(t_f) = 0, \lambda_x(t_f) = 0 \) is required. Given that \( \hat{\lambda}_x = 0 \) this results in \( \lambda_x(t_0) = 0 \), reducing the free parameter space by one dimension to \([\lambda_y(t_0), \lambda_u(t_0), \lambda_v(t_0), t_f]\).

### 4.3.1 Metaheuristic results

The guess generation procedure for this problem is similar to that described in Section 4.2.1 but with one additional free parameter dimension. Bounds of \([-6, 6]\) will be placed on the \( \lambda(t_0) \) values with bounds of \([450, 650]\) seconds placed on the terminal time. The fitness function described in Equation (4.1) will be used with weights chosen to normalize the different units. To prevent solutions in which the spacecraft vectored thrust initially downward, paths which contained negative \( y \)-values were heavily penalized. This was achieved by including an additional term in the fitness function to reflect what proportion of the trajectory was spent below the lunar surface. These solutions were improbable but trivial to prevent in this way.

Table 4.4 shows the mean terminal state achieved by each of the algorithms, as well as the mean constraint violation. The final column shows the best result achieved by each algorithm, that is, the guess which produced the lowest constraint violation. Given there is a possibly infinite number of \( \lambda_y(t_0), \lambda_u(t_0) \) and \( \lambda_v(t_0) \) combinations which solve the problem, the mean and standard deviation of these parameters are not good descriptive statistics for the consistency of metaheuristic performance. Instead histograms of each the \( \lambda(t_0) \) values are shown in Figure 4.7 for each metaheuristics. Because of the bi-modal nature of these distributions, attention should instead be given to the relationships between these parameters. For each algorithm, the best solution exhibits the following relationships for the \( \lambda(t_0) \) values:
Figure 4.7. Histograms of $\lambda(t_0)$ values obtained by the FA (top), LFA (middle) and PSO (bottom) for the minimum-time orbit injection problem.

$$\frac{\lambda_y}{\lambda_u} \approx 0.008, \quad \frac{\lambda_y}{\lambda_v} \approx 0.003, \quad \frac{\lambda_u}{\lambda_v} \approx 0.38. \quad (4.15)$$

With each $\lambda$ having the same sign.

The mean position and velocity trajectories are shown in Figures 4.8–4.10 with $3\sigma$ bounds along side the optimal trajectories, shown in black, obtained using bvp4c (discussed below). The black arrows indicate the thrust direction at that point. For this problem, constraint satisfaction is generally poor, particularly for the $H(t_f)$ and $\dot{y}(t_f)$ constraints. If the weights are set to unity (instead of scaling by the units), each algorithm is capable of satisfying the $\dot{y}(t_f)$ to within a couple of ft, however, this comes at a significant cost to the other constraints. PSO performs better, relatively speaking, than in the Zermelo problem, satisfying the position and velocity constraints better than any of the other algorithms, though still performing worse than the LFA in terms of overall constraint satisfaction.
Figure 4.8. State variables obtained by integrating FA guesses for free parameters in the minimum time orbit injection problem. (a) Trajectory in x-y coordinates, (b) horizontal velocity as a function of time (c) vertical velocity as a function of time.

Table 4.4. Metaheuristic results for the minimum time orbit injection problem. The * quantities indicate the best λ values obtained by each algorithm and the ε column shows the terminal constraint violation.

<table>
<thead>
<tr>
<th>Alg.</th>
<th>y(t_f)</th>
<th>u(t_f)</th>
<th>v(t_f)</th>
<th>t_f</th>
<th>ε</th>
<th>λ_y*(t_0)</th>
<th>λ_u*(t_0)</th>
<th>λ_v*(t_0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FA_{12}</td>
<td>74±13</td>
<td>1.03±0.08</td>
<td>0.07±0.05</td>
<td>471±22</td>
<td>0.7±0.2</td>
<td>0.018</td>
<td>2.386</td>
<td>5.999</td>
</tr>
<tr>
<td>LFA_{12}</td>
<td>79±13</td>
<td>0.96±0.07</td>
<td>0.03±0.03</td>
<td>464±14</td>
<td>0.46±0.17</td>
<td>0.018</td>
<td>2.258</td>
<td>5.709</td>
</tr>
<tr>
<td>PSO_{40}</td>
<td>80±20</td>
<td>0.91±0.08</td>
<td>0.05±0.08</td>
<td>460±15</td>
<td>0.50±0.19</td>
<td>-0.017</td>
<td>-1.987</td>
<td>-5.629</td>
</tr>
<tr>
<td>x</td>
<td>100</td>
<td>0.863</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

4.3.2 Using guesses with Bvp4c

Despite the initial guesses having worse constraint satisfaction, the initial guesses generated using the metaheuristics have significantly fewer singular jacobian errors than in the Zermelo problem. The run-time and performance statistics for the metaheuristics and bvp4c are given in Table 4.5. The optimal set of \( \lambda(t_0) \) obtained is
Figure 4.9. State variables obtained by integrating LFA guesses for free parameters in the minimum time orbit injection problem. (a) Trajectory in x-y coordinates, (b) horizontal velocity as a function of time (c) vertical velocity as a function of time.

Figure 4.10. State variables obtained by integrating PSO guesses for free parameters in the minimum time orbit injection problem. (a) Trajectory in x-y coordinates, (b) horizontal velocity as a function of time (c) vertical velocity as a function of time.
\[
\begin{bmatrix}
\lambda_y(t_0) \\
\lambda_u(t_0) \\
\lambda_v(t_0)
\end{bmatrix} = 
\begin{bmatrix}
-1.59575 \\
-185.461 \\
-525.576
\end{bmatrix}, \text{ with } t_f = 479.1117,
\] 

(4.16)

which matches the relationships given in Equation 4.15, though with greater accuracy available, the exact relationships are \(\lambda_y/\lambda_u = 0.008604\), \(\lambda_y/\lambda_u = 0.003036\) and \(\lambda_y/\lambda_u = 0.352872\). The optimal trajectory plots are included in Figures 4.8–4.10 alongside the initial guess trajectory, the \texttt{bvp4c} solutions give numerical precision terminal boundary condition satisfaction.

### 4.3.3 Comparison of algorithm performance

The LFA continues to give the most consistent performance of any of the algorithms. The variations in run-time and functions evaluations are between a factor of three and four lower than the other algorithms, while resulting in zero failures and warnings when using the guesses with \texttt{bvp4c}. The FA performs the worst on this problem taking the longest time and averaging the lowest constraint satisfaction. This continues to suggest that the addition of the Lévy distribution does improve the performance of the FA. The PSO results are the most surprising, given the poor performance on the Zermelo problem. Run-time remains the best of any of the algorithms, but in addition, the number of bad guesses has decreased to almost none. This highlights the importance of the fitness function when using metaheuristics, expanding on what was mentioned earlier in Section 4.3.1, setting the weights to unity greatly altered the metaheuristic results. With unity weights, the number of singular jacobian failures for PSO guesses increased dramatically to 25\%. Still, even with a poor choice of fitness function, each metaheuristic was capable of solving the problem via an indirect method 75\% of the time. While the LFA is able to deliver good results, the PSO seems better suited to solving this problem as it is capable of matching the LFA performance in less time.
Table 4.5. Run-time and function evaluation statistics for the minimum-time orbit injection problem. Failures and warnings correspond to singular jacobian errors and final state residuals not satisfying the required tolerance, respectively.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>guess generation</th>
<th>failures</th>
<th>warnings</th>
<th>run-time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FA&lt;sub&gt;12&lt;/sub&gt;</td>
<td>9 ± 3 × 10&lt;sup&gt;3&lt;/sup&gt;</td>
<td>13.1%</td>
<td>0%</td>
<td>1.3±0.5 ×10&lt;sup&gt;2&lt;/sup&gt;</td>
</tr>
<tr>
<td>LFA&lt;sub&gt;12&lt;/sub&gt;</td>
<td>8.5 ± 0.7 × 10&lt;sup&gt;3&lt;/sup&gt;</td>
<td>0%</td>
<td>0%</td>
<td>1.2±0.2 ×10&lt;sup&gt;2&lt;/sup&gt;</td>
</tr>
<tr>
<td>PSO&lt;sub&gt;40&lt;/sub&gt;</td>
<td>5 ± 2 × 10&lt;sup&gt;3&lt;/sup&gt;</td>
<td>1.4%</td>
<td>0.7%</td>
<td>0.7±0.4 ×10&lt;sup&gt;2&lt;/sup&gt;</td>
</tr>
</tbody>
</table>

4.4 Mars orbital transfer

Trajectory optimization of low-thrust orbital maneuvers has generated a great deal of research interest dating from as early as the 1960s [56–62]. They are attractive for a number of reasons but most importantly, the significantly higher payload mass fraction compared to more conventional chemical propulsion systems. In recent years as these propulsion systems have entered the realms of reality, there have been a number of low-thrust missions to comets, asteroids and the outer planets with a number of future missions proposed. This chapter will examine a planar, low-thrust transfer from Earth orbit to Martian orbit. The thrust will be held constant in magnitude and vectored to achieve the orbital transfer in minimum time. This problem is in some ways another extension of the Zermelo problem where the constant vector field has been replaced with a single gravitational potential. The polar coordinate system is illustrated in Figure 4.11 with state variables: orbital radius, \( r \), polar angle, \( \theta \), radial velocity, \( v_r \) and angular velocity \( v_\theta \).
Figure 4.11. Coordinate system for the Earth-Mars orbital transfer.

In this problem a satellite with initial mass $m_0 = 4545.5$ kg, thrust $T = 3.787$ N and mass flow rate $\dot{m} = 6.787 \times 10^{-5}$ kg/s will transfer from and earth orbital radius to martian orbital radius with the goal of minimizing transit time. The cost function is therefore

$$\min J = \int_{t_0}^{t_f} dt$$

subject to the constraints

$$\begin{align*}
\dot{r} &= v_r, \\
\dot{\theta} &= \frac{v_\theta}{r}, \\
\dot{v}_r &= \frac{v_r^2}{r} - \frac{\mu}{r^2} + \frac{T \sin \beta}{m_0 - \dot{m} t}, \\
\dot{v}_\theta &= -\frac{v_r v_\theta}{r} + \frac{T \cos \beta}{m_0 - \dot{m} t}.
\end{align*}$$

This leads to the system Hamiltonian given below
\[ H = 1 + \lambda_r r + \lambda_\theta v_\theta \frac{v_\theta}{r} + \lambda_{v_r} \left( \frac{v_\theta^2}{r} - \frac{\mu}{r^2} + \frac{T \sin \beta}{m_0 - \dot{m} t} \right) \]
\[ + \lambda_{v_\theta} \left( - \frac{v_r v_\theta}{r} + \frac{T \cos \beta}{m_0 - \dot{m} t} \right). \]
\[ \text{(4.19)} \]

As in the prior chapters, the necessary conditions can be derived using Equation (2.13). The co-state equations are
\[
\frac{\partial H}{\partial \mathbf{x}} = \dot{\lambda} = \begin{cases} 
\dot{\lambda}_r = \lambda_\theta \frac{v_\theta}{r^2} - \lambda_{v_\theta} \frac{v_\theta}{r^2} - \lambda_{v_r} \left( \frac{2 \mu}{r^3} - \frac{v_\theta^2}{r^2} \right), \\
\dot{\lambda}_\theta = 0, \\
\dot{\lambda}_{v_r} = \lambda_{v_\theta} \frac{v_\theta}{r} - \lambda_r, \\
\dot{\lambda}_{v_\theta} = \lambda_{v_\theta} \frac{v_\theta}{r} - \frac{\lambda_\theta}{r} - 2 \lambda_{v_r} \frac{v_\theta}{r} \end{cases} \]
\[ \text{(4.20)} \]

and the optimal control policy can be solved in terms of \( \lambda_{v_r} \) and \( \lambda_{v_\theta} \)
\[
\tan(\beta) = \frac{\lambda_{v_r}}{\lambda_{v_\theta}} \]
\[ \text{(4.21)} \]

As this is a free time problem, optimality will occur by satisfying the additional constraint \( H(t_f) = 0 \). The initial and final states are
\[
\hat{\mathbf{x}}(t_0) = \begin{bmatrix} r(t_0) \\ v_r(t_0) \\ v_\theta(t_0) \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} \]
\[ \text{(4.22)} \]
\[
\hat{\mathbf{x}}(t_f) = \begin{bmatrix} r(t_f) \\ v_r(t_f) \\ v_\theta(t_f) \end{bmatrix} = \begin{bmatrix} 1.524 \\ 0 \\ 0.810 \end{bmatrix} \]
\[ \text{(4.23)} \]

in canonical units with \( \mu = 1 \) and length and time units \( \text{LU} = 1.496 \times 10^{11} \text{ m} \) and \( \text{TU} = 58.18 \text{ days} \), respectively. Note that in a free final time formulation of this problem, \( \theta(t_f) \) is no longer fixed (as it changes depending on the value of \( t_f \)) and hence, \( \lambda_\theta(t_f) = 0 \). Given that \( \lambda_\theta(t) = \text{const} \), \( \lambda_\theta \) is therefore zero for all \( t \). This completely decouples \( \dot{\theta} \) and \( \dot{\lambda}_\theta \) from the remaining equations which means for free final time problems these two equations can be ignored. This leaves free
parameters: $\nu_r(t_0)$, $\nu_r(t_0)$, $\nu_r(t_0)$ and $t_f$ for which guesses must be supplied. The original formulation of this problem is given in Bryson and Ho [7], however, instead of minimizing transit time they wished to maximize range.

### 4.4.1 Metaheuristic results

An additional solution to this problem is given by Rauwolf and Coverstone-Carrol in [63] where a Genetic Algorithm (see [18]) was used to generate guesses for the thrust profile throughout the orbital transfer, their results will be compared alongside the present work. Bounds of $[-6,6]$ have been placed on $\nu(t_0)$ and $[2.5,4.5]$ TU were placed on $t_f$. For the purposes of consistency, the fitness function from [63] is used:

$$
\epsilon = \frac{(r(t_f) - \hat{r}(t_f))^2}{0.01^2} + \frac{(v_r(t_f) - \hat{v}_r(t_f))^2}{0.01^2} + \frac{(v_r(t_f) - \hat{v}_\theta(t_f))^2}{0.01^2} + \frac{(H(t_f) - 0)^2}{3.5^2} \quad (4.24)
$$

Due to the large number of state, co-state and derived quantities obtained, plotting all of the figures for each of the methods is cumbersome. Figures will be limited to the FA$_{12}$, LFA$_{12}$ and PSO$_{40}$ results as the $N_p = 8$ worker firefly algorithms perform similarly but slightly worse than the $N_p = 12$ cases. Figures 4.13 - 4.15 show the trajectory guesses for the spacecraft as well as the state and co-state evolution while Table 4.6 shows the $t_f$ guess and $x(t_f)$ obtained by each algorithm along with the residual with respect to the prescribed terminal constraints. Both of the Lèvy flight firefly algorithms guess a final time of $\approx 182$ days, with the final states deviating on average by a norm of $10^{-3}$. The LFA in all categories outperforms the default firefly algorithm. The PSO results greatly undercut the other algorithm’s transit duration, however, this comes at the expense of poor constraint satisfaction and the highest variance of any method. For contrast, Rauwolf and Coverstone-Carrol obtain a trip duration of 199 days with worse $t_f$ residuals compared with the LFA and comparable deviation to the FA$_{12}$ method. The control profile is not given in [63] so it cannot be included in the following figures for comparison. Because there is no unique $\lambda(t_0)$ that solves the problem, the time evolution of the best co-state guess obtained by each of the algorithms will be presented. For information on the variance of the $\lambda(t_0)$

53
values consult Figure 4.12. Considering the ratios of each of the $\lambda(t_0)$ the optimal relationship for the initial co-state values appears to be:

$$\lambda_r/\lambda_{vr} \approx 4.1763, \quad \lambda_r/\lambda_{v\theta} \approx 6.9096, \quad \lambda_{vr}/\lambda_{v\theta} \approx 0.6044. \quad (4.25)$$

### Table 4.6. Mars orbital transfer final states of guesses generated by each algorithm. All quantities are in canonical length and time units defined in the text.

| Algorithm | $t_f$ | $r(t_f)$ | $v_r(t_f)$ | $v_\theta(t_f)$ | $|\mathbf{x}(t_f) - \hat{\mathbf{x}}(t_f)|$ |
|-----------|------|----------|------------|----------------|----------------------------------|
| FA$_8$    | 3.1±0.3 | 1.52±0.02 | 0.00±0.03  | 0.81±0.03      | 42(2)×10$^{-3}$                  |
| FA$_{12}$ | 3.1±0.2 | 1.52±0.01 | 0.000±0.017| 0.811±0.014    | 23(1)×10$^{-3}$                  |
| LFA$_8$   | 3.126±0.016 | 1.524±0.03 | 0.000±0.004| 0.810±0.005    | 7(3)×10$^{-3}$                   |
| LFA$_{12}$| 3.127±0.011 | 1.524±0.02 | 0.000±0.002| 0.8097±0.0014  | 3(1)×10$^{-3}$                   |
| PSO$_{40}$| 2.64±0.19 | 1.4±0.3   | 0.09±0.16  | 0.7±0.2        | 100(40)×10$^{-3}$                |
| GA        | 3.42    | 1.512     | 0.008      | 0.812          | 16×10$^{-3}$                     |
| $\hat{\mathbf{x}}(t_f)$ | –      | 1.524     | 0.000      | 0.810          | –                                |

**Figure 4.12.** Histograms of $\lambda(t_0)$ values obtained by the FA (top), LFA (middle) and PSO (bottom) for the Earth-Mars orbital transfer problem.
Figure 4.13. Earth-mars state and co-state trajectories for guesses obtained using the FA12. Panels (a) - (c) show the transfer trajectory in x-y coordinates as well as the radial and velocity components. The small black arrows on panel (a) indicate the thrust vectoring, sampled every $t_f/10$ days. Panels (d) - (f) show the co-state trajectory for the best $\lambda(t_0)$ guess obtained. In each plot the two lines are the bvp4c optimal solution (black/solid) obtained from the given initial guess (red/dashed).
Figure 4.14. Earth-mars state and co-state trajectories for guesses obtained using the LFA12 algorithm. Panels (a) - (c) show the transfer trajectory in x-y coordinates as well as the radial and velocity components. The small black arrows on panel (a) indicate the thrust vectoring, sampled every $t_f/10$ days. Panels (d) - (f) show the co-state trajectory for the best $\lambda(t_0)$ guess obtained. In each plot the two lines are the \texttt{bvp4c} optimal solution (black/solid) obtained from the given initial guess (blue/dashed).
Figure 4.15. Earth-mars state and co-state trajectories for guesses obtained using the PSO$_{40}$ algorithm. Panels (a) - (c) show the transfer trajectory in x-y coordinates as well as the radial and velocity components. The small black arrows on panel (a) indicate the thrust vectoring, sampled every $t_f/10$ days. Panels (d) - (f) show the co-state trajectory for the best $\lambda(t_0)$ guess obtained. In each plot the two lines are the bvp4c optimal solution (black/solid) obtained from the given initial guess (green/dashed).
4.4.2 Performance comparisons

On the whole, the metaheuristics were able provide good guesses to solve the problem, variations in solution quality were significantly lower than the minimum time orbit injection problem and a very high number of guesses were able to converge to the optimal solution. The firefly algorithms perform generally better than the PSO approach and produce initial guesses which, like in the Zermelo problem, closely replicate the optimal solution. The addition of Lèvy flights continues to improve the performance relative to the default firefly implementation, though this effect is less pronounced compared with the prior examples. Further, the addition of four fireflies did not appear to greatly improve the performance of either the FA or LFA but did increase run-time dramatically. Failures did occur when using eight workers for the firefly variants, though at very low rate while taking approximately two-thirds the run-time of the 12 worker trials.

Table 4.7 shows the run-time and evaluation statistics for each algorithm. The firefly algorithm’s run-time sits at $\approx 300s$ which is significantly longer than the faster converging, though less reliable, PSO algorithm ($t \approx 100s$). This is in about the same ratio as the prior problems. The high standard deviation in function evaluations (and therefore run-time) for the PSO comes from the asymmetry in the function evaluations distribution. In a around 10% of trials more than 5000 function evaluations were required to satisfy the convergence criteria, further, the median function evaluations required is 2760 therefore 50% of trials sit in the range $[2760, 14000]$ (no trial required more than 14000 iterations), greatly increasing the standard deviation.
Table 4.7. Run-time statistics for the Low-thrust Mars transfer problem. The function evaluations and evaluation time correspond to the guess generation stage with the PSO and firefly algorithms while the failures correspond to the \texttt{bvp4c} evaluation.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Evals</th>
<th>Time (s)</th>
<th>Failures</th>
</tr>
</thead>
<tbody>
<tr>
<td>FA$_8$</td>
<td>$6.3\pm0.9 \times 10^3$</td>
<td>$2.6\pm0.5 \times 10^2$</td>
<td>0.4%</td>
</tr>
<tr>
<td>FA$_{12}$</td>
<td>$9\pm2 \times 10^3$</td>
<td>$3.8\pm0.9 \times 10^2$</td>
<td>0%</td>
</tr>
<tr>
<td>LFA$_8$</td>
<td>$5.1\pm0.2 \times 10^3$</td>
<td>$2.0\pm0.3 \times 10^2$</td>
<td>0.5%</td>
</tr>
<tr>
<td>LFA$_{12}$</td>
<td>$7.8\pm0.3 \times 10^3$</td>
<td>$3.3\pm0.7 \times 10^2$</td>
<td>0%</td>
</tr>
<tr>
<td>PSO$_{40}$</td>
<td>$3.0\pm1.5 \times 10^3$</td>
<td>$1.3\pm1.2 \times 10^2$</td>
<td>9.5%</td>
</tr>
</tbody>
</table>

The three problems presented in this chapter featured increasingly complicated state and co-state dynamics but were in essence tangent-steering problems with the goal of minimizing trip duration. In each case, the metaheuristics performed well and generated guesses which when used to initialize \texttt{bvp4c} typically produced the optimal solution. The LFA was well suited to these problems with consistently smaller error bars and good terminal constraint satisfaction when compared to the other metaheuristics. In the next chapter, a different type of problem will be presented in which the control variable is bounded and proceeds on a \textit{singular-arc}. In this case, derivation of the optimal control policy is significantly more involved and cannot simply be obtained through the necessary conditions.
Chapter 5  
Goddard rocket problem

This chapter discusses a problem in which the control variable, rocket thrust, is bounded and it is necessary to use Pontryagin’s minimum principle (Section 2.1.2) to derive the optimal control relationships as the necessary conditions will result in no algebraic relationship for the control. This particular problem is generally referred to as the Goddard Rocket problem, after rocketry pioneer Robert H. Goddard. As with the prior problems, metaheuristics will be used to arrive at the optimal set of initial time co-states to solve the problem. To emphasize the efficacy of using these metaheuristics in conjunction with indirect methods, a simple shooting method will be used to obtain the optimal solution to this problem instead of bvp4c. The optimal solution and metaheuristic results are given in Section 5.2. Additionally, there are several literature solutions to variations of this problem which will be discussed in Section 5.4.

5.1 The Goddard rocket

This problem deals with finding the optimal thrust controlling to attain the maximum possible altitude for a single-stage rocket in vertical flight. It is common to see this problem posed with or without the inclusion of drag forces and an altitude varying gravity term. When discussed in the absence of drag forces and under a constant gravity approximation, the optimal solution can be obtained via the minimum principle and is simply the ‘bang-bang’ control policy, whereby the rocket is thrust maximally until it runs out of fuel at which point it turns off [7,64]. The
inclusion of these terms into the system dynamics means thrusting maximally will having diminishing results as the drag forces become large and must therefore be controlled to produce the maximum altitude. As this is an altitude maximization problem, there are no integrable costs to the system so the performance index is simply

\[ \min \ J = -h(t_f), \quad (5.1) \]

where the minimum of a negative quantity yields the maximum. The state dynamics are

\[
\begin{align*}
&\begin{cases}
\dot{h} \\
\dot{v} \\
\dot{m}
\end{cases} = f = \\
&\begin{cases}
v \\
\frac{T - D(h,v)}{m} - g(h) \\
- \frac{T}{c}
\end{cases}
\end{align*}
\]

which produces the following system Hamiltonian

\[ H = \lambda_h v + \lambda_v \left( \frac{T - D(h,v)}{m} - g(h) \right) - \lambda_m \left( \frac{T}{c} \right). \quad (5.3) \]

where the drag and gravitational force are defined by

\[ D(h,v) = \frac{1}{2} D_c v^2 \text{Exp} [\beta (1 - h)], \quad D_c = \frac{\rho C_d A}{m_0 g_0} \quad \text{and} \quad g(h) = g_0 \left( \frac{h_0}{h} \right)^2. \quad (5.4) \]

The constants \( g_0 \) and \( \rho_0 \) are the gravity and air density at sea level, \( c \) is a measure of the exhaust velocity, \( \beta \) is the air density decay rate and \( C_d \) and \( A \) are the drag coefficient and cross-sectional area, respectively.

The co-state equations can be derived from the Hamiltonian normally
\[
\frac{\partial H}{\partial x} = \begin{cases} 
\dot{\lambda}_h \\
\dot{\lambda}_v \\
\dot{\lambda}_m 
\end{cases} = \begin{cases} 
\lambda_v \left( \frac{\partial g}{\partial h} + \frac{1}{m} \frac{\partial D}{\partial h} \right) \\
\lambda_v \frac{\partial D}{m \frac{\partial v}{\partial t}} - \lambda_h \\
\lambda_v \left( T - D \right) \end{cases} \quad \text{(5.5)}
\]

however, as the Hamiltonian is linear in the control variable the necessary condition \( \frac{\partial H}{\partial T} = 0 \) can no longer result in any algebraic expression for the thrust controlling:

\[
\frac{\partial H}{\partial T} = \frac{\lambda_v}{m} - \frac{\lambda_m}{c} = 0. \quad \text{(5.6)}
\]

Using the minimum principle and considering the variation in the Hamiltonian with respect to small changes in \( T \) it is clear that the Hamiltonian will be strictly increasing if

\[
\frac{\lambda_v}{m} - \frac{\lambda_m}{c} > 0, \quad \text{(5.7)}
\]

and strictly decreasing if

\[
\frac{\lambda_v}{m} - \frac{\lambda_m}{c} < 0. \quad \text{(5.8)}
\]

This implies

\[
T = \begin{cases} 
0 & \frac{\lambda_m}{m} - \frac{\lambda_m}{c} > 0 \\
T_{\text{max}} & \frac{\lambda_m}{m} - \frac{\lambda_m}{c} < 0 
\end{cases} . \quad \text{(5.9)}
\]

However, this gives no information about the singular solution where the relationship in Equation (5.6) equals zero. For the singular solution it must be required that Equation (5.6) is constant along the optimal trajectory. Taking the time derivative yields

\[
\frac{d}{dt} \left( \frac{\lambda_v}{m} - \frac{\lambda_m}{c} \right) = -\lambda_h + \frac{\lambda_v}{m} \left[ \frac{\partial D}{\partial v} + \frac{D}{c} \right] . \quad \text{(5.10)}
\]

Combining this with the Hamiltonian and Equation (5.6) results in three linear equations for the unknowns \( \lambda_h, \lambda_v \) and \( \lambda_m \) which all cannot vanish simultaneously.
Therefore, the following determinant must be zero:

\[
\begin{vmatrix}
    v & -\frac{D}{m} - g & 0 \\
    0 & c & -m \\
    -c & \frac{c}{m} (\frac{\partial D}{\partial v} + D) & 0
\end{vmatrix} = 0 \tag{5.11}
\]

which results in the singular arc condition

\[
v \left( \frac{\partial D}{\partial v} + \frac{D}{c} \right) - (D + mg) = 0 \tag{5.12}
\]

which when satisfied transfers the system onto the singular arc. While on the singular arc, the thrust magnitude can be found by taking the second time-derivative of Equation (5.6)

\[
\frac{d^2}{dt^2} \left( \frac{\lambda_v}{m} - \frac{\lambda_m}{c} \right) = -\dot{\lambda}_h m - \dot{m} \lambda_h + \dot{\lambda}_v \left( \frac{\partial D}{\partial v} - \frac{D}{c} \right) \\
+ \lambda_v \left( \ddot{v} \frac{\partial^2 D}{\partial v^2} + \ddot{h} \frac{\partial^2 D}{\partial h \partial v} + \frac{\dot{v} \partial D}{c \partial v} + \frac{\dot{h} \partial D}{c \partial h} \right), \tag{5.13}
\]

and solving for the thrust to yield

\[
T_s = D + mg + \frac{m \left[ c(c - v) \frac{\partial D}{\partial h} - g \left( D + c \frac{\partial D}{\partial v} \right) - vc^2 \frac{\partial^2 D}{\partial v \partial h} \right]}{D + 2c \frac{\partial D}{\partial v} + c^2 \frac{\partial^2 D}{\partial v^2}} \tag{5.14}
\]

This then derives the optimal control policy for the Goddard Rocket problem:

1. Initialize with thrust \( T = T_{\text{max}} \) until Equation (5.12) is satisfied.

2. Proceed on singular arc with \( T_s \) given in Equation (5.14).

3. When fuel is entirely consumed coast until maximum altitude is reached.

For the terminal conditions in this problem, only the final mass is constrained at \( m(t_f) = m_c m(t_0) \), where \( m_c \) is the rocket mass. Given that the velocity is free and there are no terminal parameters associated with velocity \( \lambda_v(t_f) = 0 \) and as altitude is free and being maximized, \( \lambda_h(t_f) = -1 \). Then, considering the \( H(t_f) = 0 \) condition (as terminal time is also free),

63
\[ H(t_f) = \lambda_h(t_f)v(t_f) + \lambda_v(t_f) \left( \frac{T(t_f) - D}{m} - g \right) - \lambda_m(t_f) \left( \frac{T(t_f)}{c} \right) = 0 \]

It is clear that this can only be satisfied if \( v(t_f) = 0 \) (as there will be no fuel left hence, \( T(t_f) = 0 \)).

The equations of motion can be rendered dimensionless by following the approach of Bryson \cite{65,66} and scaling the equations of motion and model parameters in terms of \( h(t_0), m(t_0) \) and \( g_0 \) using:

\[
T_{\text{max}} = 3.5g_0m(t_0), \quad c = \frac{1}{2}(g_0h_0)^{1/2}.
\]

With these choices, the initial conditions are then \( h(t_0) = m(t_0) = g_0 = 1 \) by making use of the following constants:

\[
\beta = 500, \quad m_c = 0.6, \quad D_c = 620.
\]

## 5.2 Metaheuristic results

There are four parameters in this problem which require initial guesses, \( \lambda(t_0) \) and the terminal time \( t_f \). Bounds of \([-6, 6] \) and \([0.11, 0.35] \) were placed on the initial co-state values and terminal time, respectively. The lower bound on time corresponds to twice the time it would require to completely burn through the fuel at maximum thrust\(^1\). The fitness function definition given in Equation (4.1) was used with weights skewed towards the \( \lambda_h(t_f), \lambda_v(t_f) \) and \( H(t_f) \) constraints. The reason for this is that the \( m(t_f) \) constraint is trivial to satisfy as the thrusting policy derived above and the goal of maximizing \( h(t_f) \) generally guarantees its satisfaction.

The averaged state, co-state and control trajectories are shown in Figures 5.2 and 5.3 along with the optimal solution obtained via a shooting method\(^2\). Because of the high variability in \( \lambda(t_0) \) solutions, graphing the mean co-state evolution for

---

\(^1\) \( T_{\text{max}} = 3.5g_0 \) therefore this is still a conservative lower bound.

\(^2\) The shooting method used to obtain the optimal solution will terminate if the final state constraint violation is within \( \epsilon = 10^{-12} \) or if 1000 function evaluations take place.
each guess is not useful, instead histograms of the initial costate guesses are shown in Figure 5.1. Tables 5.1 and 5.2 show the mean terminal state of the system and the terminal constraint violation for each of the metaheuristics. The LFA is the most capable of satisfying each of the constraints and delivers the greatest precision of each of the metaheuristics. The $H(t_f) = 0$ constraint in this problem is somewhat misleading as the value of $\lambda_h(t_f)$ can be ignored provided $v(t_f) = 0$. This is the case with the FA and PSO algorithms which, on average, come closer to having a terminal velocity of zero while having poor $\lambda_h(t_f) = -1$ satisfaction. This is reflected in the overall terminal constraint satisfaction (the final column of the table) where the PSO and FA have significantly higher constraint violations in spite of the small $H(t_f)$ violation, with very large variation compared with the LFA.

![Histograms of $\lambda(t_0)$ values obtained by the FA (top), LFA (middle) and PSO (bottom) for the Goddard rocket problem.](image)

**Figure 5.1.** Histograms of $\lambda(t_0)$ values obtained by the FA (top), LFA (middle) and PSO (bottom) for the Goddard rocket problem.

The thrust and singular arc condition plots shown in Figure 5.3 show the control profile obtained when integrating the average of the initial guesses generated from each metaheuristic. The averaged control profile is presented because of the high...
variation in the terminal time guessed by each meta heuristic. If the typical process of integrating each guess and then averaging is used the resulting profile features oscillations that are not representative of the true control profile of each initial guess. Because of how the control profile is derived, each initial guess is capable of reproducing the ‘bang-singular-bang’ control profile characteristic of the Goddard problem, with variations in the terminal state variables occurring as a result of the high variation in the terminal time. It is unclear why the FA is able to produce the most consistent \( t_f \) values in this problem while simultaneously being unable to consistently provide good guesses for the other free parameters.

**Table 5.1.** Averaged terminal state values obtained by each of the metaheuristics for the Goddard Rocket Problem.

<table>
<thead>
<tr>
<th>Alg.</th>
<th>( h(t_f) )</th>
<th>( v(t_f) )</th>
<th>( t_f )</th>
</tr>
</thead>
<tbody>
<tr>
<td>FA(_{12})</td>
<td>1.0128±0.0001</td>
<td>0.008±0.009</td>
<td>0.191±0.011</td>
</tr>
<tr>
<td>LFA(_{12})</td>
<td>1.0126±0.0003</td>
<td>0.012±0.015</td>
<td>0.186±0.014</td>
</tr>
<tr>
<td>PSO(_{40})</td>
<td>1.0124±0.0009</td>
<td>0.00±0.03</td>
<td>0.20±0.04</td>
</tr>
</tbody>
</table>

**Table 5.2.** Averaged terminal constraint values obtained by each of the metaheuristics for the Goddard Rocket Problem.

<table>
<thead>
<tr>
<th>Alg.</th>
<th>( m(t_f) )</th>
<th>( \lambda_h(t_f) )</th>
<th>( \lambda_v(t_f) )</th>
<th>( H(t_f) )</th>
<th>( \epsilon \times 10^{-1} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>FA(_{12})</td>
<td>0.6</td>
<td>-0.6±0.3</td>
<td>0.02±0.02</td>
<td>-0.03±0.02</td>
<td>4±3</td>
</tr>
<tr>
<td>LFA(_{12})</td>
<td>0.6</td>
<td>-0.90±0.12</td>
<td>0.006±0.007</td>
<td>-0.015±0.015</td>
<td>1±1</td>
</tr>
<tr>
<td>PSO(_{40})</td>
<td>0.6</td>
<td>-0.6±1.3</td>
<td>0.0±0.3</td>
<td>0.0±0.2</td>
<td>5±10</td>
</tr>
</tbody>
</table>

**Table 5.3.** Optimal solutions to the Goddard Rocket problem.

<table>
<thead>
<tr>
<th>Method</th>
<th>( \lambda^*_h(t_0) )</th>
<th>( \lambda^*_v(t_0) )</th>
<th>( \lambda^*_h(t_0) )</th>
<th>( t^*_f )</th>
<th>( h(t_f) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>FA(_{12})</td>
<td>-4.6072</td>
<td>-0.17553</td>
<td>-5.3259</td>
<td>0.19875</td>
<td>1.01283689</td>
</tr>
<tr>
<td>LFA(_{12})</td>
<td>-4.5832</td>
<td>-0.17462</td>
<td>1.4297</td>
<td>0.19882</td>
<td>1.01283689</td>
</tr>
<tr>
<td>PSO(_{40})</td>
<td>-4.5765</td>
<td>-0.17436</td>
<td>0.29510</td>
<td>0.19878</td>
<td>1.01283689</td>
</tr>
<tr>
<td>Optimal</td>
<td>-4.5782</td>
<td>-0.17433</td>
<td>-0.06228</td>
<td>0.198856</td>
<td>1.01283692</td>
</tr>
<tr>
<td>Dolan et al</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.19886</td>
<td>1.01283</td>
</tr>
<tr>
<td>Graichen et al</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.198856</td>
<td>1.01283692</td>
</tr>
</tbody>
</table>
Figure 5.2. State and Co-state time-history for the Goddard Rocket problem obtained by the FA (red), LFA (blue) and PSO (green) with the optimal solution obtained using a shooting method shown black. The plots of the costates show only the best solution obtain while the state plots show the average of all the trials.

5.3 Performance comparisons

The metaheuristics required significantly more function evaluations in this problem to arrive at good initial guesses, these statistics are shown in Table 5.4. The FA algorithm, predominantly terminated as a result of reaching the maximum permitted iterations rather than converging to a solution which could explain its higher constraint violation. The LFA and PSO exhibit similar tardiness, though neither of them ever reached the maximum permitted iterations. The PSO continues to take about half
as many function evaluations to terminate, however, this is again highly variable as is the run time of 300 ± 100 seconds. When used with a shooting method, none of the initial guesses failed to converge to within a tolerance $\epsilon = 10^{-12}$ in under the 1000 allowed function evaluations, with an average of $\approx 550$ evaluations required for guesses generated by each metaheuristic. The single-shooting method is typically far more sensitive to an initial guess than the collocation based bvp4c. That there were no failed initial guesses is likely a result of the more explicitly defined control policy as even the worst trials, in terms of constraint violation, were able to produce a ‘bang-singular-bang’ or ‘bang-bang’ control profile.

Table 5.4. Metaheuristic performance statistics for the Goddard Rocket problem.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Metaheuristic Evals.</th>
<th>Shooting Method Evals.</th>
<th>Run-time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FA$_{12}$</td>
<td>$11.9(1) \times 10^3$</td>
<td>$5.6(3) \times 10^2$</td>
<td>$6.1(4) \times 10^2$</td>
</tr>
<tr>
<td>LFA$_{12}$</td>
<td>$11.2(3) \times 10^3$</td>
<td>$5.6(3) \times 10^2$</td>
<td>$5.3(3) \times 10^2$</td>
</tr>
<tr>
<td>PSO$_{40}$</td>
<td>$6(3) \times 10^3$</td>
<td>$5.7(3) \times 10^2$</td>
<td>$3(1) \times 10^2$</td>
</tr>
</tbody>
</table>

5.4 Literature results

There are a number of solutions to this problem for different drag and gravitational models. References [67] and [66] provide solutions to models most similar to those discussed in this work. The thrust histories obtained are shown in Figure 5.4 and
their optimal $h(t_f)$ and $t_f$ values are given in Table 5.3. The terminal time and switching points for the maximum thrust burn, singular arc and coast are all the same as obtained using the metaheuristics. In both these works, the optimal co-state values are not given, however, the optimal trajectories for the state and control variables are identical to those shown in Figures 5.2 and 5.3. The periodic variations in the thrust history given by Dolan et al result from an insufficiently fine mesh and would be resolved with more mesh points [67].

**Figure 5.4.** Left: Thrust time history obtained by Graichen et al. Right: Thrust time history obtained by Dolan et al.
Chapter 6
Conclusion

This thesis has presented a procedure for solving optimal control problems via indirect methods. Metaheuristic algorithms have proven to be an effective tool in the optimization process, with solutions close to optimal being obtained in each of the presented problems prior to invoking a numerical optimization tool such as \texttt{bvp4c}. Analysis in this work was limited to the LFA, FA and PSO methods but in principle the methods presented here would work with any of the hundreds of metaheuristic algorithms that exist today.

The recently-developed Lèvy firefly algorithm has performed as well or better than the canonical particle swarm optimization on each of the problems, with significant advantages in robustness and consistency of performance. Further, the addition of Lèvy flights to the base firefly algorithm produced a meaningful improvement in algorithm performance, with the LFA also outperforming the FA in every way at no additional cost of computation time. The Lèvy flight firefly algorithm suffered the lowest number of bad-guess related errors, with only five trials in five thousand resulting in a divergent solution when used to initialize \texttt{bvp4c}. In the absence of the proximity based convergence criteria, the LFA greatly outperforms PSO in terms of run-time and function evaluations as $\approx 3$ times fewer swarm agents were required to generate non-divergent initial guesses. With a convergence criteria included the PSO had a much larger propensity to become trapped in unfavorable minima or solutions that would not converge using \texttt{bvp4c}. This highlights the issues of using a convergence criteria with algorithms like PSO, which converge quickly in a proximity sense. Premature termination will often occur and bad solutions will be
used where further improvement would have likely happened. As discussed above, faster termination with convergence enabled was generally the only advantage the PSO had over the LFA. To conclusively comment on whether one algorithm is more suited for use with indirect methods would require examining many more problems than discussed in this work. However, the results presented here indicate the LFA provides a significant advantage in consistency of performance relative to PSO, and if reliability is preferred over a loss in run-time, this method should be selected.

It is clear that when doing any optimization with a metaheuristic, the choice of fitness function and free parameter bounds is crucial to ensuring a successful result. In the case of the Zermelo problem, PSO generally failed at finding solutions to this problem. This is largely because the time bounds included an unphysical lower bound of 0 seconds on the solution which by chance resulted in a local minimum in the fitness function. For the other problems presented, the particle swarm optimization performed at least a factor of two better in terms of the number of bad guesses but it should be noted that the Lèvy firefly algorithm never converged to the bad minimum at all.

Despite the success of the course/fine optimization approach used in this work, there is a good deal of further work and refinement that should be done. As mentioned above, a better procedure for deriving fitness functions to use with the metaheuristics could greatly improve the likelihood of arriving at a good initial guess. Further, efforts should be made to reduce computation time spent on the metaheuristics by identifying when a sufficiently good initial guess has been obtained. That the metaheuristics were able to produce solutions often very close to optimal is impressive, but a large amount of run-time, particularly with the firefly based algorithms, was wasted improving on guesses that likely would have converged to an optimal solution when used to initialize an indirect method. The proximity convergence criteria was an attempt at solving this problem but it is clearly not appropriate for use with all metaheuristics.
Bibliography


