Imperial College London

Optimisation of Spacecraft Rendezvous Trajectories Using Multicomplex Derived Sensitivities

Final Year Short Project

by

Leo Bugeja

Supervised by Dr. Robert Hewson

to obtain the degree of Master of Engineering

at Imperial College London,

Department of Aeronautics

June 2021

Student Number: 01354416

Abstract

In this report the sensitivity based optimisation of rendezvous trajectories is used to assess the potential benefit of the multicomplex step method over existing methods. The International Space Station (ISS) was used as the rendezvous target of a low thrust, high specific impulse chaser spacecraft. The chaser spacecraft used a continuous burn to reach the target which was optimised for fuel usage. It was found that the multicomplex step method used in conjunction with the multicomplex Matlab class, was able to calculate more accurate sensitivities which in general meant faster convergence of the trajectory optimisation and a more fuel efficient final control trajectory. However, the multicomplex step method scaled poorly with increased problem complexity when compared to the finite difference method. Improvements to the class's scaling performance could prove the method to be advantageous in many applications including trajectory optimisation.

Contents

List of Figures

List of Tables

1 Introduction

With the dawning era of commercial space flight and the reinvigorated public interest in space, the frequency and importance of rendezvous manoeuvres are increasing. Today, they are most commonly used for resupply missions to the International Space Station (ISS) but other examples include: satellite repair missions such as the servicing of the Hubble Telescope, and the upcoming Artemis missions where the Orion capsule will rendezvous with the Lunar Gateway.

This report investigates the sensitivity based optimisation of rendezvous trajectories. The sensitives are calculated using the recently developed multicomplex step (MX) method and its efficacy compared to other sensitivity derivation methods for this type of problem. A key benefit to the multicomplex step method is that it can compute machine-level accurate higher order derivatives, which offers the potential to improve the performance of these optimisation problems. A newly created multicomplex Matlab class enables the use of multicomplex operations to be integrated with optimisation and numerical integration methods.

For the simplicity of this report, the rendezvous target represented as the ISS in 400 km circular orbit and the chaser spacecraft is in low earth orbit (LEO). Trajectory optimisation is performed on the target-centred relative equations of motion with the objective to reach the target and minimise for fuel consumption.

2 Literature Review

The key areas of literature to explore in order to achieve the projects aims are: rendezvous manoeuvres and the relevant equations of motion; multicomplex numbers and using the multicomplex Matlab class; sensitivity methods including multicomplex step method; time integration schemes for calculating the trajectory; sensitivity based optimisation methods which the multicomplex step method can be applied to.

2.1 Orbital Rendezvous

The main phases of a rendezvous mission are: launch, phasing, far range rendezvous, close range rendezvous and mating [5]. This report will focus on far range rendezvous which includes target chaser distances between tens of kilometres down to the order of hundreds of

meters. Farther than this and the relative equations of motion will be less applicable, and closer than this, non-Keplerian forces such as solar radiation and air resistance dominate the dynamics which is beyond the scope of this project. Additionally, regulations for safety reasons are involved with the closing and final approach which would complicate the trajectory optimisation process.

One simple rendezvous scenario is where the chaser has the same orbit but is either behind or ahead of the target as shown in Figure 1. If behind, the chaser decelerates to lower its orbit and shorten its orbital time period. The difference between the target's and new chaser's orbital time period is planned such that after one exact orbit, the chaser intersects with the target. Now that their orbital positions match, a second burn by the chaser cancels any relative velocity. It is also possible to save fuel with phasing by performing a smaller initial burn and then require multiple more orbits to catch up with the target rather than just one.

Another simple rendezvous scenario uses a Honmann transfer [8] where the chaser is in a different circular orbit from the target shown in Figure 2. An initial burn moves the chaser into a transfer orbit that intersects with the target in the future. At the point of intersection, a second burn raises or lowers the transfer orbit to match the target's orbit.

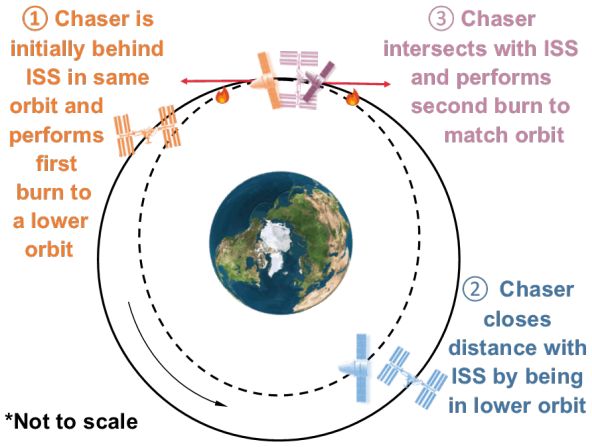

Figure 1: Phasing rendezvous

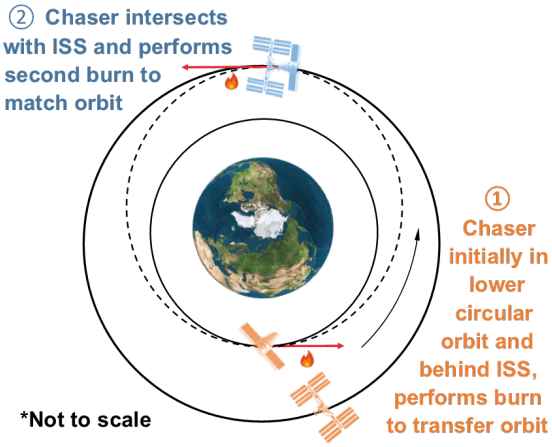

Figure 2: Honmann transfer type rendezvous

As the optimal fuel usage in these two scenarios can be calculated, they can be used to compare with the optimised trajectory results.

2.1.1 Equations of Relative Motion

Using a reference frame of the chaser's motion relative to the target's, shown in Figure 3, makes the optimisation less complicated given that the goal is always to reach a zero relative position and velocity.

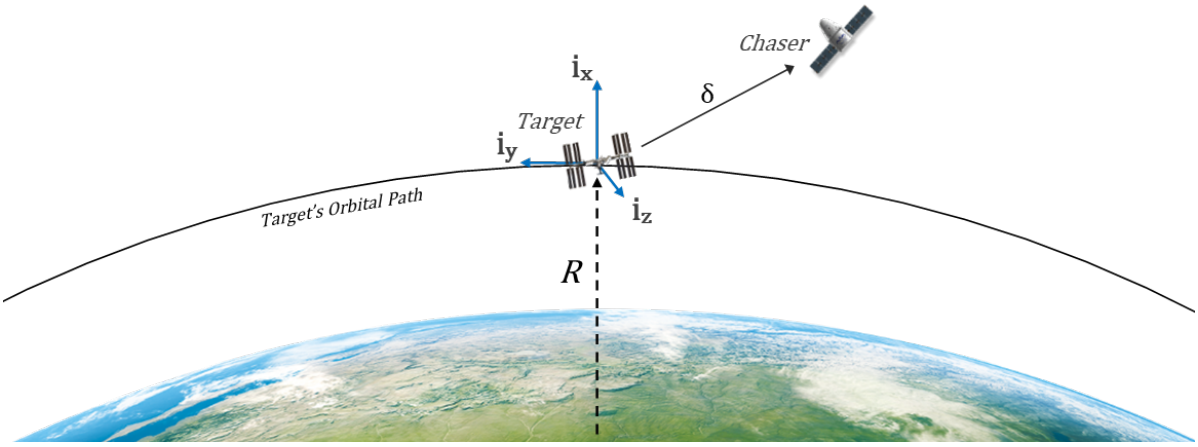

Figure 3: Figure of the relative orbital reference of the chaser with respect to the target

Given that the target's orbital radius R is constant (ie, in circular orbit), and its motion is within the reference plane (ie, $\dot{z}_{target} = 0$), then the following exact nonlinear equations of relative motion are [6]:

$$\ddot{x} = [R + x]\left[\omega + \frac{\dot{y}}{R}\right]^2 + \frac{\mu\,[R + x]}{\left([R + x]^2 + z^2\right)^{3/2}} + a_x \tag{1}$$

$$\ddot{y} = \frac{-2\dot{x}\,[R\omega + \dot{y}]}{R + x} + a_y \tag{2}$$

$$\ddot{z} = -\frac{\mu z}{\left([R + x]^2 + z^2\right)^{3/2}} + a_z \tag{3}$$

The orbital angular velocity of the target is ω and a_x, a_y, a_z are the non-Keplarian/disturbing accelerations such as the chaser's propulsion. Using the assumption that $x, z \ll R$ and $\dot{y} \ll \omega$, and performing a Taylor's expansion to 1st order, results in a linearised form which are known as the Clohessy-Wiltshire equations [6]:

$$\ddot{x} = 3\omega^2 x + 2\omega\dot{y} + a_x \tag{4}$$

$$\ddot{y} = -2\omega\dot{x} + a_y \tag{5}$$

$$\ddot{z} = -\omega^2 z + a_z \tag{6}$$

Due to linearisation, the accuracy naturally decreases with distance from the target orbit. According to [5], "In a LEO rendezvous mission, position errors will become significant at a distance of a few tens of kilometres from the origin". Hence it will be important that the initial conditions and trajectory stay within these bounds.

2.2 Multicomplex Numbers

Multicomplex numbers are an expanded set of the complex space, where there exists n imaginary commuting components which satisfy the condition $i_n^2 = -1$ [7]. The multicomplex number system $\mathbb{C}_n$, can also be recursively defined as:

$$\mathbb{C}_{n+1} = \{z = x + yi_{n+1} : x, y \in \mathbb{C}_n\} \tag{7}$$

2.2.1 Multicomplex Step Method

Multicomplex numbers are essential to the multicomplex step method which has been shown to achieve highly accurate nth order derivatives numerically [2]. It was developed out of the complex step method [14], which found that the first order derivative can be approximated by the imaginary part of the function with an imaginary step via complex Taylor expansion analysis:

$$f'(x) = \frac{\Im\left[f(x + ih)\right]}{h} + \mathcal{O}\left(h^2\right) \tag{8}$$

As opposed to finite difference methods such as forward difference,

$$f'(x) = \frac{f(x + h) - f(x)}{h} + \mathcal{O}\left(h\right), \tag{9}$$

only one function evaluation is required which means rounding errors associated with subtraction of the terms are removed. However, this is only valid for the first derivative, hence the multicomplex step method was developed [7], which eliminates the rounding error for any nth order derivative using n different multicomplex steps:

$$f^{(n)}(x) = \frac{\Im_{1\ldots n}\left[f\left(x + i_1 h + \cdots + i_n h\right)\right]}{h^n} + \mathcal{O}\left(h^2\right) \tag{10}$$

The operator $\Im_{1\ldots n}$, extracts the number containing the multicomplex parts $i_1, i_2 \ldots i_n$. Mixed derivatives of any order can also be calculated by extracting the imaginary parts which corresponds to the desired derivatives, for example:

$$\frac{\partial^2 f\left(x, y, z\right)}{\partial x \partial z^2} = \frac{\Im_{1,2,3}\left[f\left(x + i_1 h, y, z + i_2 h + i_3 h\right)\right]}{h^3} + \mathcal{O}\left(h^2\right) \tag{11}$$

2.2.2 Multicomplex Matlab Class

Multicomplex numbers can be represented as matrices with real coefficients, making the optimised vector calculations in Matlab a potential advantage. A Matlab class was created in [2],

which allows for the common mathematical operations of multicomplex numbers in addition to their use for obtaining nth-order derivatives. The performance of the class was improved [12] through implementing matrix-less operations and allow Matlab functions to run in a C++ environment by creating an interface. The Matlab class was also used to investigate halo orbits in three body problems with radiation pressure [11]. The multicomplex step method was found to have better convergence than a differential relation based method when used to calculate the state transition matrix (STM) for the differential correction method.

This class will be used in this report to extract the sensitivity information from a previous trajectory iteration, and use them as part of a sensitivity optimisation method to improve further iterations.

2.3 Sensitivity Methods

Sensitivities in the context of this report, are a measure of how a small change in the input affects a change in the output. Given the aim of the report is to assess the effectiveness of the multicomplex step method, it must therefore be compared to other equivalent methods.

Finite difference (FD) as introduced in the previous section, works similarly to the multicomplex step method but uses a real perturbation rather than an imaginary one to approximate the derivative. Finite difference is one of the most commonly used numerical methods for its ease of use and almost universal applicability.

Automatic Differentiation (AD) uses a technique which breaks down a function into composite parts of elementary operations and then uses chain rule to evaluate the derivative [10]. However, AD cannot be used in all circumstances.

Symbolic Differentiation (SD) uses algebraic manipulation to derive the components of the gradient but like AD cannot be used in all circumstances [10].

2.4 Time Integration Schemes

The Clohessy-Wiltshire equations (4 – 6) have an exact solution [3] which would allow for analytical time integration, though this only applies when the disturbing accelerations are zero which isn't helpful if the trajectory needs to be controlled. Alternatively it is possible to calculate an analytical solution for the linear equations by using Laplace transforms [3] which requires input accelerations of constant amplitudes.

However, there is currently no analytical solution to the nonlinear equations which therefore

requires a numerical integration method to solve for the trajectory. The built in ordinary differential equation solvers in Matlab such as **ode45**, cannot be used as it isn't compatible with the multicomplex class.

The Newmark-beta method [4] which is second order accurate, can directly solve the equations of motion given that they are second-order differential equations.

Alternatively the three second order differential equations (1–3), can be transformed into a set of six first order equations and solved using a method such as Runge-Kutta (RK) [4]. The fourth order accurate Runge-Kutta (RK4), is the most commonly used of the RK methods, though RK in general requires more function evaluations than Newmark-beta.

2.5 Optimisation Methods

All the optimisation methods investigated in this section are sensitivity based to demonstrate the relative benefits of using the multicomplex step method.

2.5.1 Gradient Descent and Backtracking Line Search

Gradient descent is an iterative method that minimises a given function based on the first order sensitivities (ie, the gradient). $f(\boldsymbol{x})$ is the function to be minimised which could be the distance away from a target at the end of a trajectory:

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n - t_n \boldsymbol{\nabla} f_n \tag{12}$$

Note that $f_n \equiv f(\boldsymbol{x}_n)$. Choosing a step size t that is too small results in slow convergence and a step size that is too large will not converge at all. For this reason the backtracking line search algorithm is used at each iteration to find the step size which minimises the objective to a sufficient degree with the following steps:

1. Set τ_0 to the initial step size and iteration number $j = 0$.
2. Until the Armijo condition [1], $f(\boldsymbol{x}_n) - f(\boldsymbol{x}_n - \tau_j \boldsymbol{\nabla} f_n) \leq \alpha \tau_j \boldsymbol{\nabla} f_n{}^T \boldsymbol{\nabla} f_n$, is satisfied, increment j and set $\tau_j = \beta \tau_{j-1}$.
3. Finally, use $t_n = \tau_j$ as the step size for iteration n.

The parameters are recommended to be $\alpha \in (0, 1/2)$ and $\beta \in (0, 1)$. $\alpha = 0.5$ and $\beta = 0.3$ have been chosen for general good performance with initial test runs.

The problem with gradient descent, is that at saddle points it significantly slows down due to a zig-zag pattern between the sides of the trough.

2.5.2 Newton-Raphson Method

The Newton-Rapshon method is not affected at saddle points like gradient descent because of the addition of second derivatives which adjusts the step direction to take into consideration the rate of change of the descent. Each iteration is as follows:

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n - t_n \boldsymbol{H}_n{}^{-1} \boldsymbol{\nabla} f_n \tag{13}$$

Note that $\boldsymbol{H}_n{}^{-1} \equiv \boldsymbol{H}\,[f_n]^{-1}$. The backtrack line search algorithm for gradient descent is also used here. However, given that the step direction has changed from $\boldsymbol{\nabla} f_n$ to $\boldsymbol{H}_n{}^{-1} \boldsymbol{\nabla} f_n$ the Armijo condition is now:

$$f(\boldsymbol{x}_n) - f(\boldsymbol{x}_n - \tau_j \boldsymbol{\nabla} f_n) \leq \alpha \tau_j \boldsymbol{\nabla} f_n{}^T \boldsymbol{H}_n{}^{-1} \boldsymbol{\nabla} f_n. \tag{14}$$

Directly calculating the second order derivatives can be computationally expensive. The quasi-Newton method can instead indirectly compute the second order derivatives using first order derivatives from current and previous iterations [10].

2.5.3 Constrained Minimisation

Gradient descent and Newton-Raphson are both unconstrained optimisation methods. However, in trajectory optimisation it is not necessarily sufficient to just produce a feasible solution which reaches the target. Rather it is usually necessary to minimise for other factors such as fuel and have constraints such as maximum thrust and mission duration.

A pseudo constraint minimisation method, is the penalty method with objective function $f(\boldsymbol{x})$ and constraint function $c(\boldsymbol{x})$:

$$g(\boldsymbol{x}) = f(\boldsymbol{x}) + \sigma * c(\boldsymbol{x}), \tag{15}$$

where σ is the penalty coefficient with a greater value leading to higher relative constraint minimisation. Gradient descent and Newton-Raphson can easily incorporate this method however convergence to the equivalent constrained problem is not guaranteed.

The sequential quadratic programming (SQP) method can be viewed as a generalisation of Newton's method, with nonlinear constraints involved and is considered one of the most effective methods for this type of problem [10].

Matlab has a built in nonlinear constrained problem solver **fmincon** [9]. This solver includes four optimisation algorithms to choose from: interior-point, SQP, active-set and trust-region-reflective. A necessity of the reports analysis on the efficacy of the multicomplex step method, is to be able to supply the multicomplex derived sensitivities to the optimisation method. Only

the interior-point algorithm is able to take first and second order derivatives as input along with nonlinear constraints [9]. This solver requires that the second order derivatives (ie, hessian matrix) for the objective and nonlinear constraint functions are supplied in the form of the second derivative of the Lagrangian. The Lagrangian function is of the form:

$$\mathcal{L}(\boldsymbol{x}, \lambda) = f(\boldsymbol{x}) + \sum_{i=1}^{n} \lambda_i c_i(\boldsymbol{x}), \tag{16}$$

with n number of nonlinear constraints $c_i(\boldsymbol{x})$. Following this, the second derivative of the Lagrangian function with respect to $\boldsymbol{x}$ is:

$$\boldsymbol{\nabla}^2_{xx}\mathcal{L}(\boldsymbol{x}, \lambda) = \boldsymbol{\nabla}^2 f(\boldsymbol{x}) + \sum_{i=1}^{n} \lambda_i \boldsymbol{\nabla}^2 c_i(\boldsymbol{x}) \tag{17}$$

The lagrange multipliers λ_i, are supplied by the **fmincon** solver.

3 Methodology

The methods and information obtained from the literature review are applied within this section. Firstly modifications to the Matlab class are discussed, then how the numerical integration and trajectory optimisation will be implemented.

3.1 Multicomplex Matlab Class Modifications

In order to aid the usability of the multicomplex class, some function additions were made. These three functions discussed in the section are present in Appendix B.

3.1.1 Imaginary Part Extraction

The existing utility function **CX2**, is meant to extract the two part imaginary coefficient $i_n i_m$ of a multicomplex number C, using function arguments: C, n, m. However, it does not always produce the desired result for example:

Table 1: Multicomplex array format

Multicomplex Coefficient	$real$	i_1	i_2	$i_1 i_2$	i_3	$i_1 i_3$	$i_2 i_3$	$i_1 i_2 i_3$
Array Index	1	2	3	4	5	6	7	8
Multicomplex Coefficient	i_4	$i_1 i_4$	$i_2 i_4$	$i_1 i_2 i_4$	$i_3 i_4$	$i_1 i_3 i_4$	$i_2 i_3 i_4$	$i_1 i_2 i_3 i_4$
Array Index	9	10	11	12	13	14	15	16

CX2(C, 1, 4) and **CX2(C, 4, 1)** return the coefficient in the 9th array index, which corresponds with i_4 as shown in Table 1 when in fact it should return the 10th array index. Despite this, the next coefficient along **CX2(C, 2, 4)**, correctly returns the 11th array index and not the 10th if it was the case that the previous index was correct.

Therefore to fix this issue and to increase the number of imaginary parts which can be extracted, a new utility function **CXN** was implemented in the class which can extract any N part imaginary coefficient $i_x i_y \ldots i_n$ which takes in an array of the N desired imaginary parts of the multicomplex coefficient.

The multicomplex coefficients array contain a recursive pattern where for every additional imaginary part i_{n+1} that is added to the multicomplex number, the array doubles in size with the second half of the array containing all the first half of elements but multiplied by the new imaginary part:

$$\mathbb{C}_2 \text{ array form: } [real,\ i_1,\ i_2,\ i_1 i_2]$$
$$\mathbb{C}_3 \text{ array form: } [[real,\ i_1,\ i_2,\ i_1 i_2],\ [real,\ i_1,\ i_2,\ i_1 i_2] * i_3]$$
$$= [real,\ i_1,\ i_2,\ i_1 i_2,\ i_3,\ i_1 i_3,\ i_2 i_3,\ i_1 i_2 i_3]$$

This recursive property means that the index of terms containing i_x must be larger than 2^{n-1}. Using this logic, the following equation was derived which gives the desired index position for any multi imaginary part coefficient:

$$IndexPosition\left(i_x i_y \ldots\right) = \sum_{n=[x,y\ldots]} \left(2^{n-1}\right) + 1 \tag{18}$$

This equation can be easily implemented and in fewer lines of code than the **CX2** code. Additionally because a property of multicomplex numbers is imaginary part commutativity (ie, $i_1 i_2 = i_2 i_1$) and because the Equation 18 contains pure summation, then the ordering of the imaginary part terms $x, y, \ldots$ in the functions argument, does not matter. For example, **CXN(C, [3 2 4])=CXN(C, [2 3 4])** which would return: $2^{2-1} + 2^{3-1} + 2^{4-1} + 1 = 15$th array index as expected from Table 1.

The function **real** was also added to the class, which simply returns the real part of a complex number. This adds a helpful layer of abstraction over the previous method which required using the call **C.zn(1)**.

3.1.2 Multicomplex Representation

Another difficulty when using the class was that the only representation of a multicomplex object to the programmer is the array form. This is not user friendly as the array can often be very large and it is not obvious which index position corresponds to which multicomplex coefficient. Therefore for the benefit of the programmer, the utility function **repr** was created which returns a string representation of the multicomplex number. As the array form is often sparse (ie, containing many zeros), the function will only return the non zero coefficients unless all the coefficients are zero in which case the function will return the real number 0. As an example of the output:

```
>> C = multicomplex([1, 3, 0, 0, 5, 0, 0, 1];
>> repr(C)
ans =
     ”1  +  3 * i1  +  5 * i3  +  1 * i1i2i3”
```

The function creates an array of the multicomplex coefficients recursively based on the order of the multicomplex number when it is run. To improve performance a very high order multicomplex array could be precomputed and stored so that it does not need to be recreated each function call.

Note that this representation is not stored as a class variable as this would use unnecessary space and computational time. Rather it is simply a tool to evaluate the representation when needed such as during debugging.

3.2 Numerical Integration

Fourth order Runge Kutta (RK4), will be used to numerically integrate the trajectory with respect to time. This method has been chosen because it has a high order of accuracy (fourth order) which will be important for minimising errors accumulating downrage and over large timescales which are typical in spaceflight missions.

To solve the dynamics equations with RK4, they must be in the form of the initial value problem where $\boldsymbol{y}_0$ is the initial position and velocity state:

$$\frac{d\boldsymbol{y}}{dt} = \boldsymbol{f}(t, \boldsymbol{y}), \quad \boldsymbol{y}(t_0) = \boldsymbol{y}_0 \tag{19}$$

This form is achieved by converting the three second order differential equations (1–3), into six first order equations:

$$\frac{d\boldsymbol{y}}{dt} = \frac{d}{dt}\begin{bmatrix} x \\ y \\ z \\ \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ [R+x]\,[\omega + \dot{y}/R]^2 + \frac{\mu[R+x]}{([R+x]^2+z^2)^{3/2}} \\ \frac{-2\dot{x}[R\omega+\dot{y}]}{R+x} \\ -\frac{\mu z}{([R+x]^2+z^2)^{3/2}} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ a_x \\ a_y \\ a_z \end{bmatrix} \tag{20}$$

Note that the disturbing accelerations, a_x, a_y and a_z, do not have to be constant. RK4 then evaluates the following equations at each time step:

$$\begin{aligned} k_1 &= f\,(t_n, y_n) \\ k_2 &= f\left(t_n + \frac{h}{2}, y_n + h\frac{k_1}{2}\right), \\ k_3 &= f\left(t_n + \frac{h}{2}, y_n + h\frac{k_2}{2}\right), \\ k_4 &= f\,(t_n + h, y_n + hk_3) \end{aligned} \tag{21}$$

$$y_{n+1} = y_n + \frac{1}{6}h\,(k_1 + 2k_2 + 2k_3 + k_4) \tag{22}$$

$$t_{n+1} = t_n + h \tag{23}$$

Given that the target radius is one metre, the time step should be sufficiently small such that any further decrease in h will not result in a change greater than 1 metre at the end of the trajectory.

3.3 Trajectory Optimisation

For the trajectory optimisation it is important to first confirm that the method can find a feasible solution which is defined as reaching the targets position and matching its velocity at the end of the trajectory. After this the fuel consumption can then be set as the minimisation objective while the final position and velocity are constraints. To perform the optimisation process, the disturbing acceleration parameters a_x, a_y, a_z and the integration time step dt can be varied to find the best trajectory given an initial state $\boldsymbol{y}_0$ and fixed number of integration steps n.

Changes to the time step dt, is used to control the duration of the trajectory as opposed to the number of time steps as it can only be changed in integer increments. The four parameters are assumed constant throughout the trajectory and are adjusted after each iteration by the sensitivity based optimisation method using the calculated sensitivities. The constant disturbing accelerations is representative of a low thrust, high specific impulse spacecraft using electric propulsion as an example.

3.3.1 Objectives and Constraints

One of the constraints is to achieve a distance $d(\boldsymbol{x}, \boldsymbol{y}_0)$ from the target at the end of the trajectory of less than one metre:

$$d(\boldsymbol{x}, \boldsymbol{y}_0, n) = \sqrt{x_{final}^2 + y_{final}^2 + z_{final}^2} < 1 \tag{24}$$

An additional constraint, is that the magnitude of the velocity at the end of the trajectory also needs to be close to zero. However, this constraint must be tighter as even small relative velocities can result in large changes to the trajectory overtime. Therefore, 0.1 ms^{-1}, is chosen as the maximum relative velocity as it results in less than one metre drift after one orbit:

$$v(\boldsymbol{x}, \boldsymbol{y}_0, n) = \sqrt{\dot{x}_{final}^2 + \dot{y}_{final}^2 + \dot{z}_{final}^2} < 0.1 \tag{25}$$

Once feasible solutions are known to be reachable, then methods will be used to minimise for total fuel consumption $f(\boldsymbol{x}, \boldsymbol{y}_0)$ while maintaining feasibility.

Delta-v (ie, the magnitude of the change in velocity) will be used as a measure of the fuel consumption as it is mass-normalised and is widely used in orbital mechanics. It can be expressed as the following:

$$f(\boldsymbol{x}, \boldsymbol{y}_0, n) = \Delta v = \int_{t_0}^{t_f} \frac{|F_x(t)| + |F_y(t)| + |F_z(t)|}{m(t)} dt = \sum_{t=\tau_1, \tau_2 \ldots} \Delta t\big|_\tau \left[|a_x| + |a_y| + |a_z|\right]_t \tag{26}$$

The set of disturbing acceleration phases are τ_1, τ_2 ..., and the duration of each phase is $\Delta t|_\tau$. The sum of the phase duration's must equal the trajectory duration and coasting periods can be represented with the disturbing accelerations set to zero. Initially for this report there will only be one acceleration phase. The mass of the spacecraft m is assumed to be constant because the spacecraft propulsion has a high specific impulse which will use a small amount of propellant relative to the mass of the spacecraft.

3.3.2 Trajectory Sensitivities

The first order sensitivities (ie, the gradient) of the fuel consumption with respect to the four parameters are:

$$\boldsymbol{\nabla} f = \begin{bmatrix} \frac{\partial f}{\partial a_x} \\ \frac{\partial f}{\partial a_y} \\ \frac{\partial f}{\partial a_z} \\ \frac{\partial f}{\partial dt} \end{bmatrix} \tag{27}$$

The second order sensitivities (ie, the hessian matrix) with respect to the the four parameters are:

$$\boldsymbol{\nabla}^2 f = \begin{bmatrix} \frac{\partial^2 f}{\partial {a_x}^2} & \frac{\partial^2 f}{\partial a_x \partial a_y} & \frac{\partial^2 f}{\partial a_x \partial a_z} & \frac{\partial^2 f}{\partial a_x \partial dt} \\ \frac{\partial^2 f}{\partial a_y \partial a_x} & \frac{\partial^2 f}{\partial {a_y}^2} & \frac{\partial^2 f}{\partial a_y \partial a_z} & \frac{\partial^2 f}{\partial a_y \partial dt} \\ \frac{\partial^2 f}{\partial a_z \partial a_x} & \frac{\partial^2 f}{\partial a_z \partial a_y} & \frac{\partial^2 f}{\partial {a_z}^2} & \frac{\partial^2 f}{\partial a_z \partial dt} \\ \frac{\partial^2 f}{\partial dt \partial a_x} & \frac{\partial^2 f}{\partial dt \partial a_y} & \frac{\partial^2 f}{\partial dt \partial a_z} & \frac{\partial^2 f}{\partial {dt}^2} \end{bmatrix} \tag{28}$$

Note that the hessian matrix is symmetric along it's diagonal. For the linear dynamics equations, $\frac{\partial^2 f}{\partial a_x \partial a_z}$ and $\frac{\partial^2 f}{\partial a_y \partial a_z}$ will be zero in the hessian matrix because the z motion is decoupled from x and y plane hence there are no cross acceleration terms. This means that for the multicomplex method, the imaginary step for a_z can reuse the step for a_x or a_y without the two same steps interacting together. Reducing the total order of the multicomplex number means that the array representations can be smaller and so memory usage and the number of computations required are reduced.

The sensitivities of the constraints for $d(\boldsymbol{x}, \boldsymbol{y}_0, n)$, and $v(\boldsymbol{x}, \boldsymbol{y}_0, n)$ are calculated in the same way as above but the scalar function f is replaced with d and v respectively.

Each individual sensitivity is calculated using either the finite difference method or the multicomplex step method depending on the which method is being tested. The finite difference method uses the central difference for the first order sensitivities:

$$\frac{\partial f(\boldsymbol{x}, \boldsymbol{y}_0, n)}{\partial a_x} = \frac{f\left(\begin{bmatrix} a_x + h \\ a_y \\ a_z \\ dt \end{bmatrix}, \boldsymbol{y}_0, n\right) - f\left(\begin{bmatrix} a_x - h \\ a_y \\ a_z \\ dt \end{bmatrix}, \boldsymbol{y}_0, n\right)}{2h} + \mathcal{O}\left(h^2\right) \tag{29}$$

The reason for using central difference is that to calculate the second order sensitivities with finite difference, the negative h step in the a_x direction is also required:

$$\frac{\partial^2 f(\boldsymbol{x}, \boldsymbol{y}_0, n)}{\partial {a_x}^2} = \frac{f\left(\begin{bmatrix} a_x + h \\ a_y \\ a_z \\ dt \end{bmatrix}, \boldsymbol{y}_0, n\right) - 2f\left(\begin{bmatrix} a_x \\ a_y \\ a_z \\ dt \end{bmatrix}, \boldsymbol{y}_0, n\right) + f\left(\begin{bmatrix} a_x - h \\ a_y \\ a_z \\ dt \end{bmatrix}, \boldsymbol{y}_0, n\right)}{h^2} + \mathcal{O}(h) \quad (30)$$

Therefore this negative h step can also be used in addition to the positive step to calculate the first order sensitivity with $\mathcal{O}\left(h^2\right)$ truncation error using central difference. This is compared to forward difference's $\mathcal{O}\left(h\right)$ truncation error. Additionally the multicomplex step method has the same truncation error as central difference which means the comparison can focus on the rounding errors without truncation errors affecting the analysis.

The sensitivities are calculated using the multicomplex step method as described in Equations 10 and 11 in section 2.2.1. To calculate the first order sensitivities one imaginary part is required for each parameter and to calculate the second order sensitivities two imaginary parts are required each. This means that for the nonlinear dynamics, first order sensitivity methods requires four imaginary parts and second order sensitivity methods require eight imaginary parts. For the linear dynamic equations, three and six imaginary parts are required respectively. More combinations are shown in Table 5, Appendix A.

3.3.3 Unconstrained Minimisation

Even though the gradient descent method will likely take far more iterations to converge compared to second order sensitivity methods, it only requires half the number of imaginary parts which may give it an overall run-time advantage. Gradient descent and Newton-Raphson will be compared to **fmincon** to see whether an unconstrained problem optimising only for the final position, does so in a fuel efficient way and thus determine if explicitly optimising for fuel is necessary.

3.3.4 Constrained Minimisation

Initial tests of the penalty method reveal that achieving a balance between minimising the objective and constraints is incredibly sensitive to the choice of penalty coefficient σ as well as the initial state and parameters chosen. Therefore, it will not be used. The ready to use Matlab function **fmincon**, was chosen for the constrained method versus needing to implement the SQP given the short timescale of this project. **fmincon** with the interior-point algorithm requires the hessian matrix of the objective (i.e the fuel) and constraints (ie, the final distance from target) to be supplied in the hessian of the Lagrangian form (Equation 17), and wrapped in a

hessian output function. The gradients of the objective and constraints are directly supplied to **fmincon** as outputs of the objective and constraint functions.

3.3.5 Velocity Correction Burn

The issue with the methods described so far (using gradient descent, Newton-Raphson, and **fmincon**), is that they will search for a feasible final position but not a final zero relative velocity.

An additional velocity constraint could be added to **fmincon**. However given that the control accelerations are so far planned to only have a single constant phase throughout the trajectory it is unlikely all target states will be reachable. For example Honmann transfers require a minimum of two separate burns to reach the required orbital position and velocity which a single constant acceleration in most cases could not achieve. One solution to this would be to split the constant acceleration burn into two parts and add three additional parameters for the new acceleration phase (for a total of seven parameters). However, the multicomplex method would then require an addition six imaginary parts to calculate the first and second sensitivities with the new parameters which would likely exponentially increase the run time (the impact of adding imaginary terms on the time complexity will be investigated in section 4.2 to validate this claim).

Another approach can be implemented which does not require any additional imaginary parts and guarantees a zero final relative velocity. This approach uses a pseudo impulse burn at the end of the trajectory which cancels out the remaining velocity. The additional Δv consumed for this burn is then added to the fuel objective function. This adds an incentive on the objective function to find the best balance between fuel consumed during the constant phase of the trajectory, and the final correction burn to minimise its total $Deltav$. However this approach does assume the final burn velocity change is instantaneous so that the final relative position does not drift past the target as it is decelerating. In reality the second burn would need to be started just before reaching a zero relative position so that the spacecraft decelerates as it approaches the target position. Additionally this final burn is a high thrust manoeuvre which is not possible with a low thrust spacecraft which is the context of this report. Lastly, it would be dangerous for the chaser to perform a secondary burn right before reaching the target while its relative velocity is high, as there is low room for error to not collide with the target. This is why typically the rendezvous target is positioned to trail the space stations by around one km [13] and at this point proximity operations are then performed to safely close the rest of the distance. An improved velocity correction method is discussed in section 4.6.

4 Results and Discussion

The methods I have outlined were implemented to obtain the results in this section. The discussion is included with the results as some earlier outcomes affect the choices made for later results.

4.1 Sensitivity Accuracy

In Figure 4, the relative accuracy of sensitivities derived using the finite difference versus the multicomplex step method are plotted for different step sizes. Even though this is only for one set of initial conditions and specific derivatives, the pattern of the results are representative of the general behaviour.

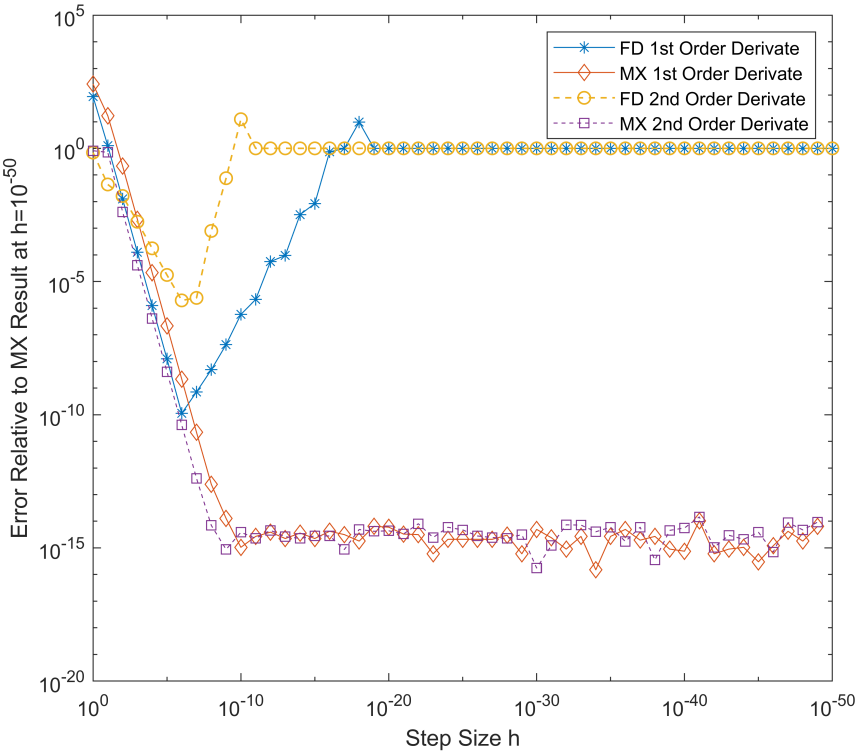

Figure 4: Sensitivity accuracy comparison for given step sizes with initial state from scenario 1 (Table 3). 1st and 2nd derivatives are represented as $\frac{\partial f}{\partial a_x}$ and $\frac{\partial^2 f}{\partial a_x \partial a_y}$.

The finite difference 1st derivatives reaches a lower minimum relative error compared to the finite difference 2nd derivatives as their truncation errors are $\mathcal{O}(h^2)$ and $\mathcal{O}(h)$ respectively

(which are from Equations 27 and 28). The multicomplex step method has a truncation error of $\mathcal{O}(h^2)$ for both derivatives, which is why it has the same gradient as the 1st order derivative. Measuring the gradient of the first downward section in the plot, will also result in an estimate of these truncation errors.

Past $h = 10^{-6}$, the accuracy of both finite difference derivatives decrease due to subtraction cancellation errors which are increasing in significance. Eventually when the step size becomes too small, the finite difference derivative approaches zero and the relative error approaches 1. On the other hand the multicomplex step method continues to increase in accuracy until it achieves the maximum arithmetic precision at h^{-10}. Thus a key benefit of using the multicomplex step method, is that the step size can be made arbitrarily small.

Given that the lowest relative error occurs at $h = 10^{-6}$ for both finite difference derivatives, this step size will be used for further finite difference calculations. The multicomplex step size is less critical as max precision is reached at $h = 10^{-10}$. Therefore $h = 10^{-15}$ is chosen as the step size to be used for this method for further calculations.

4.2 Time Complexity Analysis

Scalability for problems that require more variables or higher derivatives is an important requirement if the multicomplex step method with the Matlab class is to be commonly used. Figure 5 compares the time complexities (ie, the rate at which the run time grows for an increase in imaginary parts) for scaling of the problem. The absolute average run times of both methods are presented in Tables 5 and 6 in Appendix A. Even for zero imaginary parts the multicomplex run time is much slower than finite difference which is because the multicomplex class still has to process the real numbers through its class methods. Therefore, to ignore this overhead and instead compare how they scale relative to each other, the run times were normalised at zero imaginary parts. Note that the finite difference equivalent computation in Figure 5 means the equivalent run time to calculate the same number of sensitivities with that number of imaginary parts. For example with three imaginary parts, three first order derivatives can be calculated. More equivalencies are presented in Table 6, Appendix A.

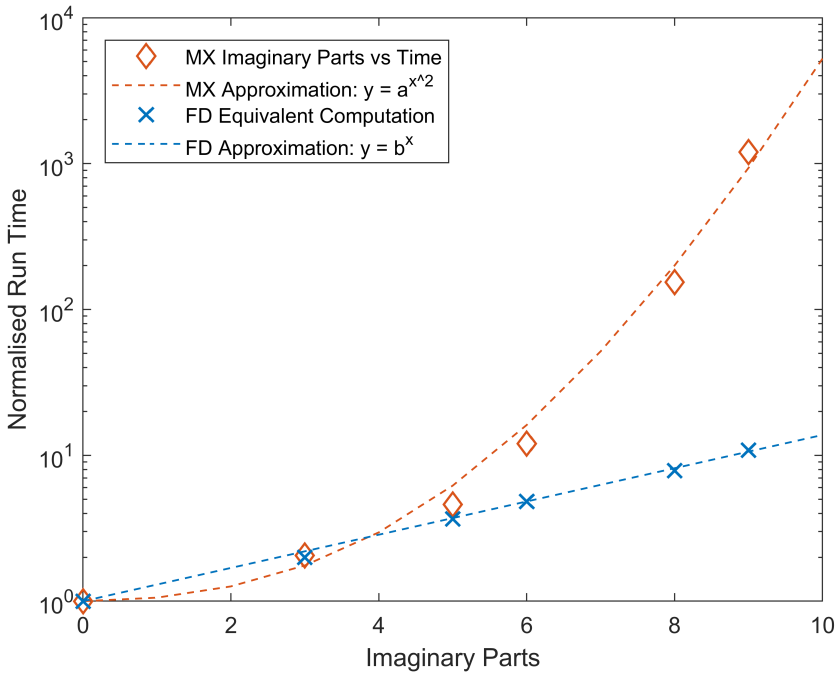

Figure 5: Time scaling of the multicomplex method versus finite difference with nonlinear dynamical equation and normalised at zero imaginary parts.

The time scaling of the multicomplex step method with the Matlab class is very poor compared to finite difference. This is because each additional imaginary part doubles the length of the multicomplex array representation. These x sized arrays are then converted to x by x sized matrices to be used for multiplication and division operations (which are most of the operations for the numerical integration). Therefore doubling the length of the multicomplex array leads to a quadrupling of the number of elements in the matrix representation and quadrupling of computations. This is analogous to a scaling of 2^{x^2} where x is the number of imaginary parts required. This scaling is confirmed by the multicomplex step line of best fit approximation in Figure 5. On the other hand finite difference only scales with 2^x, leading to a linear gradient of $log(2)$ in the logarithmic scaled chart.

Note that in Reference [12], the author attempted to change to a matrix less method for some operations to improve performance in the multicomplex Matlab class. However, multiplication and division operations were not found to be more efficient with this method.

As the nonlinear dynamics require eight imaginary parts which takes extremely long to run (data in Table 5), and given the short time scale of this project, the linearised dynamical equations which require six imaginary parts which will be used for the rest of the tests.

4.3 Optimisation Methods Comparison

This part compares the relative performance of the two unconstrained methods (gradient descent and Newton-Raphson) which are simply optimising for the final position, to **fmincon** which is minimising for the fuel (Δv) and constraining the final position. The performance criteria being assessed are: the speed of convergence towards a feasible solution, which includes the number of function calls and total run time; the amount of fuel required for the solution trajectory; the relative performance of the two sensitivity methods. The results are presented in Figure 6 and Table 2.

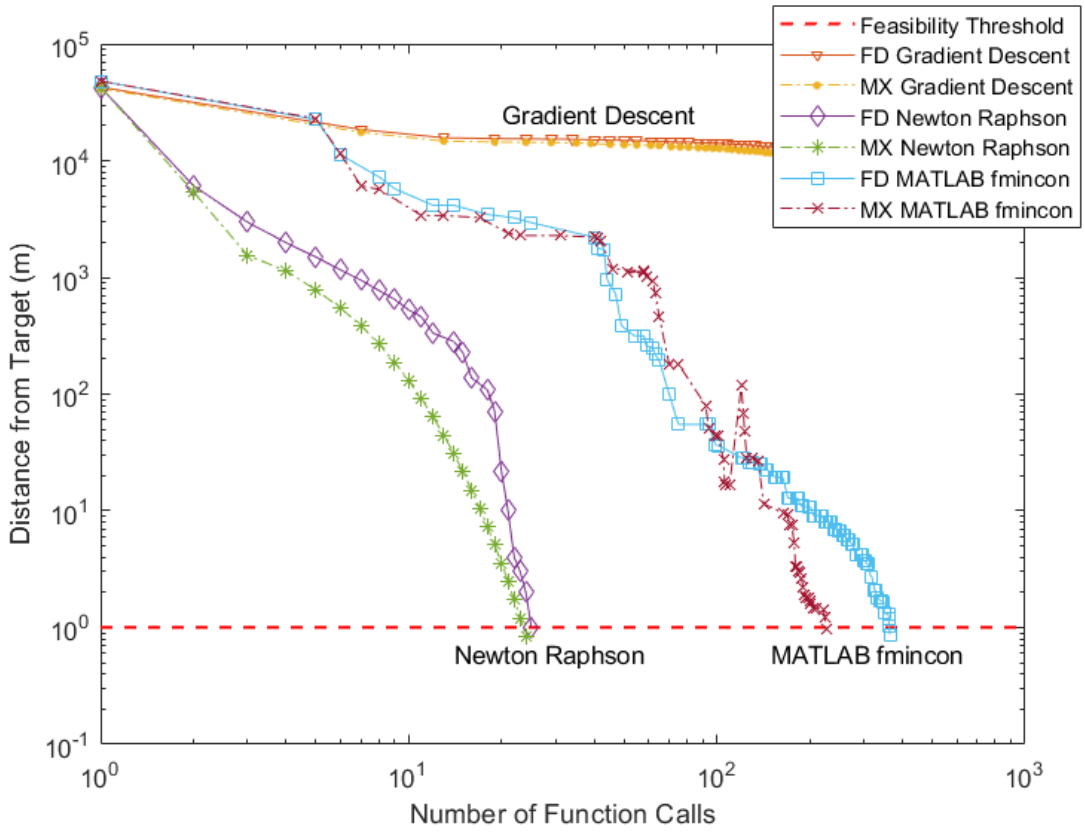

Figure 6: Convergence path of three optimisation methods and each with the comparison of multicomplex step versus finite different using scenario 1, Table 3.

The gradient descent method failed to converge to a feasible solution within a reasonable time (ie, two hours) and therefore was prematurely halted.

The multicomplex step method converged consistently faster along the iteration path compared to finite difference for gradient descent and Newton-Raphson. In contrast, **fmincon** with finite difference at some points passed the multicomplex step method though in the end required less total function calls. A reason for the non-uniformity of the **fmincon** convergence, is that this method is also balancing minimising the fuel. Therefore even though the distance

from the target might periodically increase between iterations, the fuel usage would still decrease.

Table 2: Performance and optimality of the three optimisation methods

	Function Calls	Run Time	Required (Δv)
FD Gradient Descent	$\approx 1 \times 10^8$	$\approx 2 \times 10^7 s$	$Unkown$
FD Gradient Descent	$\approx 1 \times 10^8$	$\approx 1 \times 10^9 s$	$Unkown$
FD Newton-Raphson	25	$12\,s$	$542.9\,m\,s^{-1}$
MX Newton-Rapshon	24	$526\,s$	$563.4\,m\,s^{-1}$
FD Matlab fmincon	364	$175\,s$	$23.8\,m\,s^{-1}$
MX Matlab fmincon	226	$4949\,s$	$22.7\,m\,s^{-1}$

The reduced number of function calls observed generally for the multicomplex step, is likely because the first and second order sensitivities are more accurately calculated. However in terms of run time, it was far slower as the iteration run time is much longer. This is partly because to calculate the second order derivatives for the linearised equations, six imaginary parts are required which then suffers from scaling issues as discussed in the previous section. Additionally, the time per iteration and hence run time, are partially dependent on the efficiency of the Matlab class. Conversely, the number of function calls is more indicative of the accuracy of the sensitivities provided by the methods.

The **fmincon** method requires approximately ten times the number of function calls and run time compared to Newton-Raphson despite both using second order sensitivities. This disparity is because **fmincon** is also having to balance the objective at the same time as the constraints. On the other hand, **fmincon** reduces the fuel consumption by 95% compared to Newton-Raphson. Given that any additional mass is extremely costly in space missions, reducing the fuel requirement is far more important than computation time provided that is not being used as an on-board controller that needs to be responsive in real-time.

4.4 Trajectory Visualisation

Now that the two unconstrained methods are deemed less effective than **fmincon**, this final set of tests will assess **fmincon** for the three different scenarios in Table 3. Scenario 1 starts with chaser in a 4 km lower orbit and behind the target with the y velocity set so that the orbit is circular. In scenario 2 the chaser is directly in front of the target with the same orbit and no relative z motion; In scenario 3 the chaser is in a 4 km higher circular orbit and radially inline with the target.

Table 3: Three scenarios for testing **fmincon** with multicomplex step and finite difference. *Calculated based on equivalent Honmann transfer. †Calculated based on single orbit phase transfer manoeuvre.

	Initial Position $[x, y, z]$ (km)	Initial Velocity $[\dot{x}, \dot{y}, \dot{z}]$ $(m\,s^{-1})$	Initial Parameters $[a_x, a_y, a_z]$ $(m\,s^{-2})$	Time Step (dt)	Number of Integration Time Steps (Fixed)	Theoretical Minimum Δv
Scenario 1	[-4, -10, 0]	[0, 3, 10]	[0.001, 0.001, -0.001]	2 s	2000	$16^{*}\ m\,s^{-1}$
Scenario 2	[0, 10, 0]	[0, 0, 0]	[0.001, 0.001, -0.001]	2 s	2000	$2^{\dagger}\ m\,s^{-1}$
Scenario 3	[4, 0, 0]	[0, -2, 2]	[0.001, 0.001, -0.001]	2 s	2000	$6^{*}\ m\,s^{-1}$

The results for the three different scenarios are presented in Table 4. These tests show that the multicomplex method does not always perform fewer functions calls than finite difference, which could mean that the difference in sensitivity accuracy is not significant enough to result in an improved convergence and so other factors dominate the convergence. One of these other factors could be that small differences in the initial iteration by chance steered the finite difference method down an iteration path that led to faster convergence. The scenario 3 result seems an example of this as the final disturbing accelerations and trajectory duration of the two methods are significantly different suggesting that they converged towards different local minima. On the other hand one consistent benefit of the multicomplex step over finite difference in all three scenarios, is a decrease in the required Δv. Perhaps the increased accuracy of the sensitivities with the multicomplex step are allowing convergence towards lower minima. Either way, it is hard to know without having more test scenarios and results to identify a clear pattern. Both methods with **fmincon** have a higher Δv requirement than the theoretical minimum in 3. This is most likely because a single constant phase acceleration and final correction burn is not as efficient as two separate impulse burns for a Honmann transfer. Additionally the theoretical minimum assumes that the first burn can start at any point along the orbit whereas the trajectory optimisation method forces the acceleration phase to start at the initial time position rather than coasting before reaching an optimal position to start the burn.

Table 4: Results of **fmincon** with multicomplex step and finite difference for three different scenarios

Scenario	Method	Function Calls	Run Time	Required Δv	Final Acceleration Parameters $mm\,s^{-2}$	Trajectory Duration
1	FD	201	88 s	62.4 $m\,s^{-1}$	[6.73, 1.42, -0.00514]	2773 s
	MX	184	4030 s	40.3 $m\,s^{-1}$	[6.68, 1.48, -0.307]	2726 s
2	FD	66	32 s	8.50 $m\,s^{-1}$	[1.43, -0.121, -0.000593]	4186 s
	MX	70	1533 s	7.48 $m\,s^{-1}$	[1.43, -0.122, -0.00105]	4185 s
3	FD	92	44 s	94.3 $m\,s^{-1}$	[-9.32, -0.463, -3.60]	6540 s
	MX	175	3833 s	75.0 $m\,s^{-1}$	[-7.70, -0.666, -0.938]	7628 s

Figure 7 demonstrates how a change in the acceleration parameters, would affect the shape of the trajectory. For example, an increase in a_z, would raise the whole trajectory parallel to the z axis with the larger arrows moving the trajectory by a greater amount. The optimisation methods essentially minimise for the final position by finding the combination of scalars a_x, a_y, a_z which result in a net perturbation towards the target.

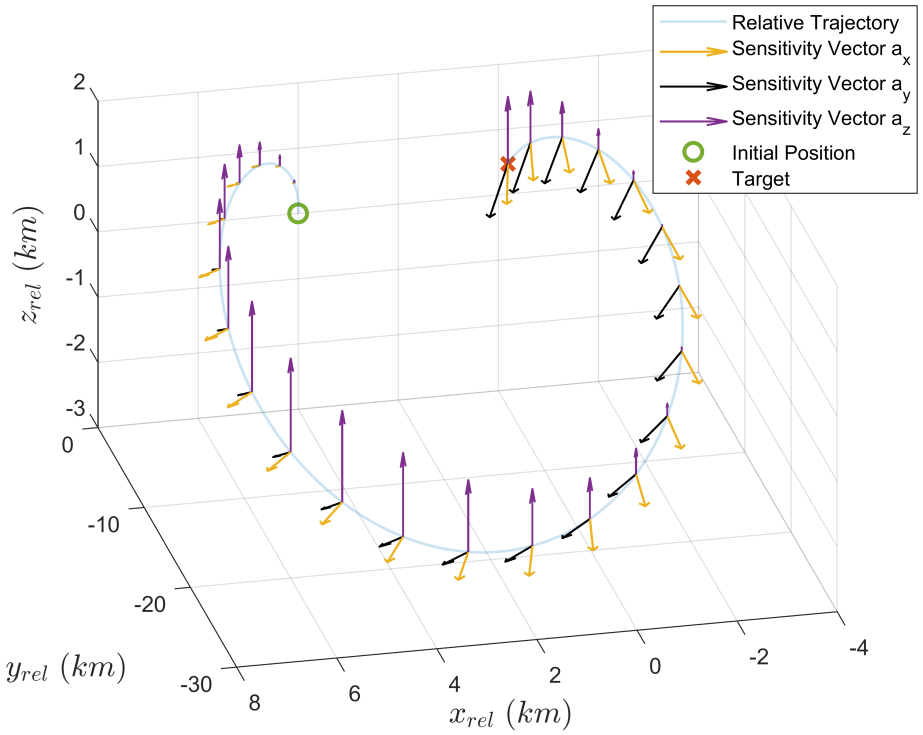

Figure 7: Sensitivity vectors along the trajectory of multicomplex step scenario 3.

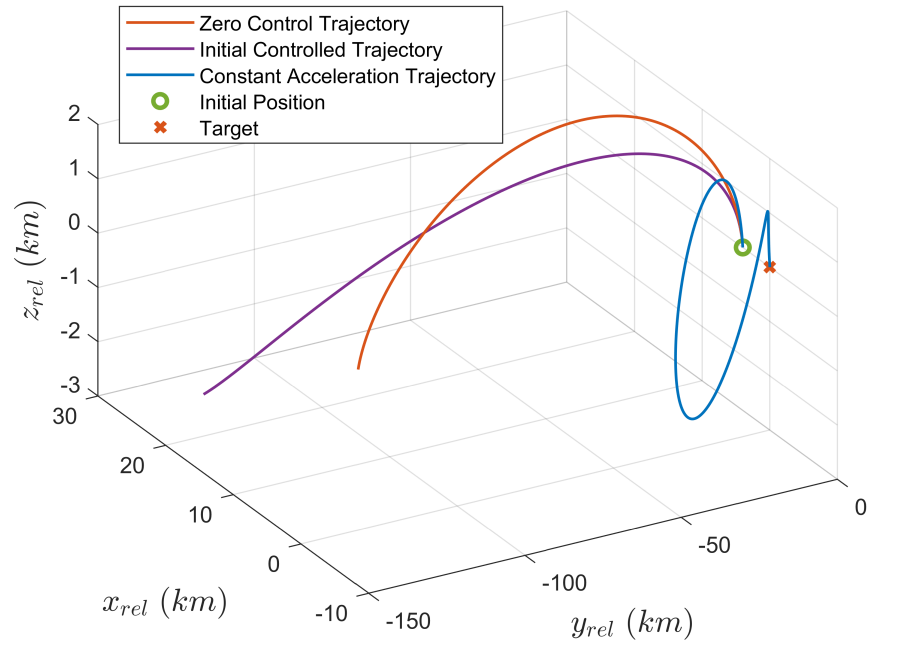

Figure 8: Relative position trajectories for multicomplex step scenario 3

Figure 8 shows that despite the initial controlled trajectory being 150 km off target the final solution was still able to converge to a feasible solution. The velocity correction burn discussed

in the methodology is shown in Figure 9, where it is cancelling the remaining relative velocity at the end of the trajectory. This means that the optimisation has chosen to use a balance of Δv between the main constant burn phase and final burn with approximately $67\ ms^{-1}$ and $8\ ms^{-1}$ respectively for this scenario. The difference in the magnitude and direction of this final burn to the the constant burn phase is demonstrated in Figure 10. Additionally even though the constant burn phase is directionally fixed in the relative target reference frame, relative to the earth the direction of the burn changes.

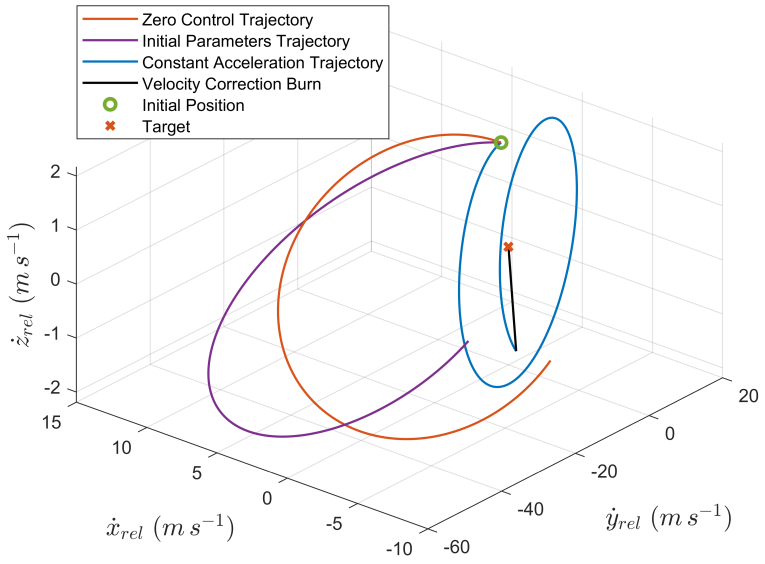

Figure 9: Relative velocity trajectories for multicomplex step scenario 3

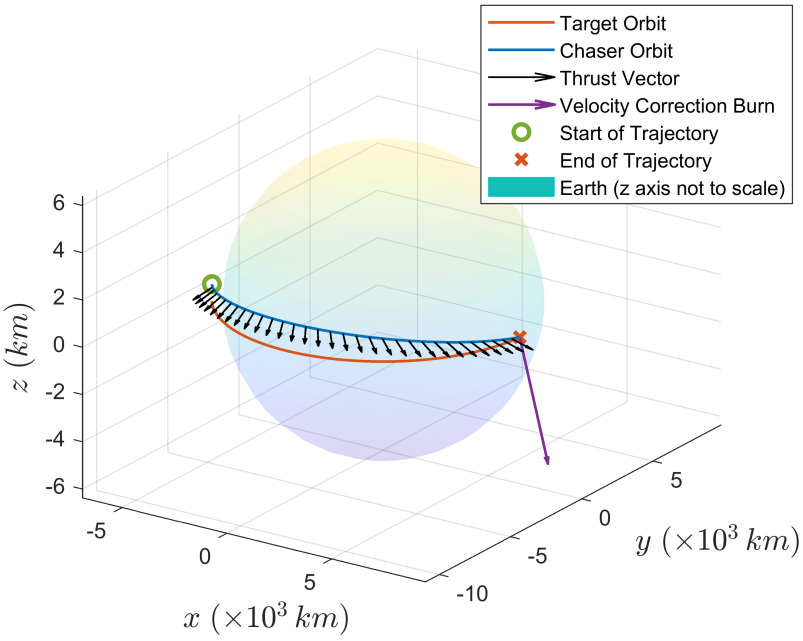

Figure 10: Target and chaser orbits relative to the earth for multicomplex step scenario 1

Lastly, the trajectory convergence of successive iterations is demonstrated in Figure 11. Only a few iterations are required to initially move within a few km of the final trajectory whereas the last few iterations change by very little as the optimisation scale changes from the order of kilometers, down to meters.

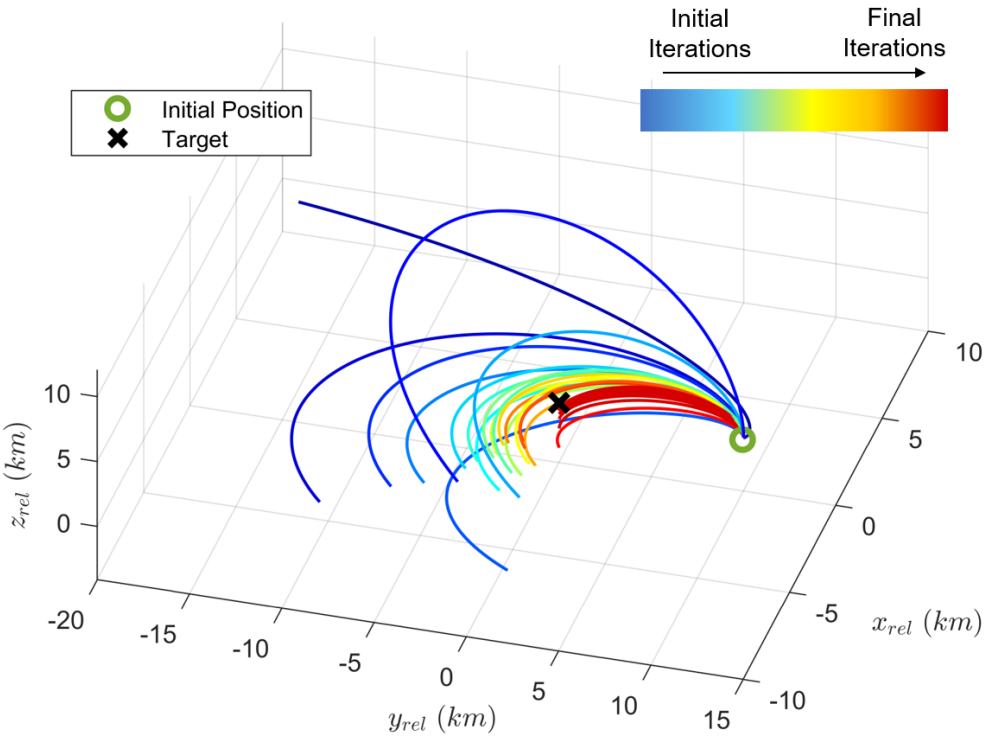

Figure 11: Successive trajectory iterations for multicomplex step scenario 2

4.5 Implementation Practicalities

An advantage of the using the multicomplex step method with the Matlab class versus finite difference, is that calculating the sensitivities is easier to implement from the programmers perspective. For the multiplex step, to calculate the first and second order derivatives of a new parameter, an imaginary part simply needs to be added to the parameter input and the derivatives are extracted using **CXn**. In contrast, finite difference requires calculating multiple additional trajectories for all the perturbation combinations. For example to add a fifth parameter, an additional six trajectory calculations are need to be added to the code whereas multicomplex only needs one trajectory calculation no matter the number of parameters. Note that a class wrapper could be implemented for the finite difference method to add these required additional trajectories though.

4.6 Methodology Improvements

Calculating the analytical solution for the linear equations using Laplace transforms [5] would enable the comparison of the absolute sensitivity errors rather than just the relative errors. Additionally, the analytical trajectories can be compared to the numerical integration method to assess its numerical accuracy and choose a time step which is sufficiently accurate. Lastly comparing the finite difference and the multicomplex step optimised trajectories with the analytical solution's, would confirm if inaccuracies in the sensitivities are leading to the sub optimal Δv or whether only having a single constant acceleration phase is the problem. In the case that the single phase is the issue, two or three phases could be added to gauge whether there is any improvement. Alternatively a continuous parameterised curve with three or four variables for each direction could be used to represent the disturbing control accelerations.

The velocity correction burn is not achievable in reality for a low thrust spacecraft. However, the approach can be suitably modified in the following way:

1. Calculate the trajectory as before with the final correction velocity.
2. Determine the amount of time t_f to achieve this change in velocity with maximum thrust.
3. Backtrack from the end of the trajectory by t_f and recompute the last section.
4. The last section will now likely land short of the target. Hence further adjustments can be made to the backtracked distance to minimise this.

The above approach still does not require adding any additional parameters or imaginary parts saving an exponential increase in run time.

5 Conclusion

The multicomplex step method was confirmed to calculate more accurate first and second order derivatives compared to the finite difference in the context of trajectory objective and constraint sensitivities. Optimising for the fuel and constraining the final trajectory position with **fmincon** is more effective than the unconstrained Newton-Raphson method in terms of required Δv. However Newton-Raphson does not converge faster to a solution given that it is not constrained. On the other hand, gradient descent's convergence is extremely slow meaning it isn't able to reach a feasible solution within reasonable time. The theoretical minimum Δv for the three scenarios are much lower than what is achieved by **fmincon** with multicomplex step and finite difference methods. Increasing the number of acceleration phases during the trajectory could close this disparity in future analysis.

The increased sensitivity accuracy of the multicomplex step method compared to finite difference, seems to lead to more optimal trajectories with a lower Δv requirement as well as possibly requiring less function calls however this is less conclusive. On the other hand, the multicomplex step method with the Matlab class suffers from poor time scaling compared to finite difference, which leads to run times that are orders of magnitude longer. In general, problems which require very accurate derivatives are going to benefit more from using the multicomplex step method and problems which are less sensitive to inaccuracies in the derivatives would benefit more from the faster run time of other methods such as finite difference.

Ultimately, improvements to the Matlab class to eliminate the multicomplex matrix operation inefficiencies would have a huge impact on the run time and more clearly justify the use of multicomplex derived sensitivities for optimisation problems.

Bibliography

[1] Larry Armijo. Minimization of functions having lipschitz continuous first partial derivatives. *Pacific Journal of Mathematics*, 16(1), 1966.

[2] Jose Casado and Rob Hewson. Algorithm 1008: Multicomplex number class for matlab, with a focus on the accurate calculation of small imaginary terms for multicomplex step sensitivity calculations. *ACM Transactions on Mathematical Software*, 46(2), 2020.

[3] Howard D. Curtis. Orbital mechanics for engineering students 3rd edition, 2014.

[4] J. Douglas Faires and Richard L. Burden. *Numerical Analysis 9th Edition*. Brooks/Cole, 2010.

[5] Wigbert Fehse. *Automated Rendezvous and Docking of Spacecraft*. Cambridge University Press, 2003.

[6] David K. Geller and T. A. Lovell. Angles-only initial relative orbit determination performance analysis using cylindrical coordinates. *The Journal of the Astronautical Sciences*, 64(1), 2017.

[7] Gregory Lantoine, Ryan P. Russell, and Thierry Dargent. Using multicomplex variables for automatic computation of high-order derivatives. *ACM Transactions on Mathematical Software*, 38(3), 2012.

[8] Wiley Larson. *Space Mission Analysis and Design 3rd Edition*. Microcosm, 1999.

[9] MathWorks. Constrained nonlinear optimization algorithms. `https://uk.mathworks.com/help/optim/ug/constrained-nonlinear-optimization-algorithms.htmllink#anchorbrnpd5f`. Accessed: 2021-06-09.

[10] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, 2nd edition, 2006.

[11] Yuri Shimane. Multicomplex-step method for space trajectory design. Master's thesis, Imperial College London, 2020.

[12] Chung Chung To. Optimisation of multicomplex step computational method and the application in orbital mechanics. Master's thesis, Imperial College London, 2021.

[13] James R. Wertz and Robert Bell. Autonomous rendezvous and docking technologies: Status and prospects. 5088, 2003.

[14] Squire William and Trapp George. Using complex variables to estimate derivatives of real functions. *Society for Industrial and Applied Mathematics*, 40(1):110–112, 1998.

A Additional Figures and Tables

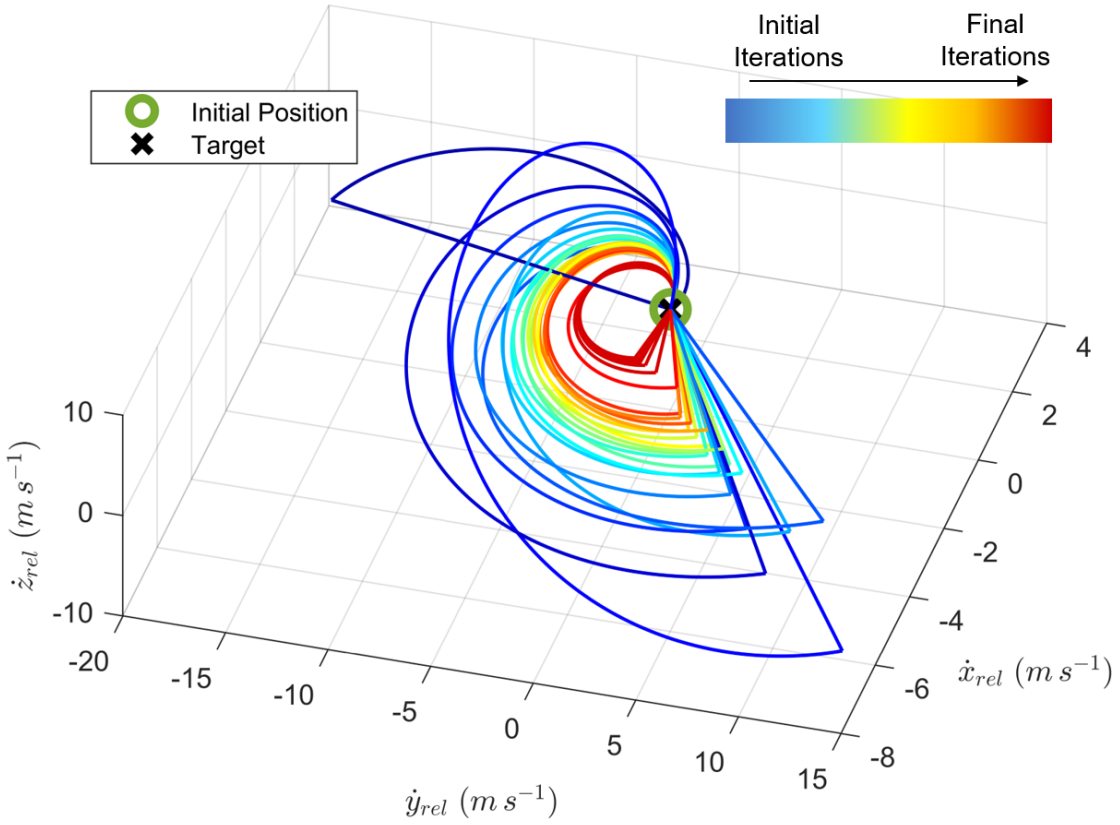

Figure 12: Velocity trajectory for multicomplex step scenario 2

Table 5: Multicomplex-step numerical integration run times

Dynamic Equations	Acceleration Sensitivities (Imaginary Parts Required)	Time Step Sensitivities (Imaginary Parts Required)	Total Imaginary Parts	Average Integration Run Time (s)
Linear	$None\ (0)$	$None\ (0)$	0	$3.2\,s$
Linear	$First\ Derivative\ (2)$	$None\ (0)$	2	$4.4\,s$
Linear	$First\ Derivative\ (2)$	$Upto\ Second\ Derivative\ (2)$	4	$6.8\,s$
Linear	$Upto\ Second\ Derivative\ (4)$	$None(0)$	4	$6.6\,s$
Linear	$Upto\ Second\ Derivative\ (4)$	$Upto\ Second\ Derivative\ (2)$	6	$21.9\,s$
Nonlinear	$None\ (0)$	$None\ (0)$	0	$5.0\,s$
Nonlinear	$First\ Derivative\ (3)$	$None\ (0)$	3	$10.3\,s$
Nonlinear	$First\ Derivative\ (3)$	$Upto\ Second\ Derivative\ (2)$	5	$21.5\,s$
Nonlinear	$Upto\ Second\ Derivative\ (6)$	$None\ (0)$	6	$60.2\,s$
Nonlinear	$Upto\ Second\ Derivative\ (6)$	$Upto\ Second\ Derivative\ (2)$	8	$768.2\,s$

Table 6: Finite difference numerical integration run times

Dynamic Equations	Acceleration Sensitivities	Time Step Sensitivities	Equivalent Imaginary Parts	Average Integration Run Time
Linear	$None$	$None$	0	$0.08\,s$
Linear	$First\ Derivative$	$None$	2	$0.14\,s$
Linear	$First\ Derivative$	$Upto\ Second\ Derivative$	4	$0.32\,s$
Linear	$Upto\ Second\ Derivative$	$None$	4	$0.34\,s$
Linear	$Upto\ Second\ Derivative$	$Upto\ Second\ Derivative$	6	$0.48\,s$
Nonlinear	$None$	$None$	0	$0.06\,s$
Nonlinear	$First\ Derivative$	$None$	3	$0.15\,s$
Nonlinear	$First\ Derivative$	$Upto\ Second\ Derivative$	5	$0.29\,s$
Nonlinear	$Upto\ Second\ Derivative$	$None$	6	$0.35\,s$
Nonlinear	$Upto\ Second\ Derivative$	$Upto\ Second\ Derivative$	8	$0.48\,s$

B New Matlab Class Functions

```
function out = CXN(C,imgn_parts)
    % This function extracts the coefficient of the user inputed
        imaginary parts contained within imgn_parts array.
    % Ex: CXn(C,[1 2 4]) will extract the i1i2i4 coefficient of
       C. As the imaginary parts are commutitative, for example:
        i1i2i4 =
    % i4i1i2, the ordering of the parts in imgn_parts array does
        not matter.

    if isnumeric(C)  max(imgn_parts) > log2(length(C.zn))  any(
       imgn_parts < 1)
        error('input not in required form or out of bounds')
    else
        index = 1;
        for n = imgn_parts
            index = index + 2^(n-1);
        end
        out = C.zn(index);
    end
end

% THE OLD CX2 FUNCTION THAT WAS REPLACED BY CXN AS ABOVE:
function out = CX2(C,im,in)

    % This function extracts the coefficient of the user inputed
        imaginary part inim. C is the multicomplex number
    % you want to extract the coefficient of. Ex: CX2(C,3,4)
       extracts the i3i4 coefficient of C.
    % It is useful for second partial derivative calculation.

    if im > in
        store = in;
        in = im;
        im = store;
    end
    if in > log2(length(C.zn))
        error('input not in required form or out of bounds')
```

```
31      end
32
33      cj=1.5;
34      val=4;
35      w=2;
36      u=1;
37      ci=0.5;
38
39      for j=2:in
40          for i=1:im
41              if j-w == 1
42                  cj=(2*cj)-1;
43                  val=val+cj;
44                  ci=0.5;
45
46              elseif i-u == 1
47                  ci=2*ci;
48                  val=val+ci;
49              end
50              w=j;
51              u=i;
52              if j-i == 1
53                  break
54              end
55          end
56      end
57      out=C.zn(val);
58  end
59
60  function out = real(C)
61      % This function extracts the real part of a multicomplex
            number
62      C.zn(1);
63  end
64
65  function out = repr(self)
66      % This function returns a string representation of the
            multicomplex number
67
68      imag_strings = strings(length(self.zn),1);
69      complex_order = log(length(self.zn))/log(2);
```

```
70
71      for n = 1:complex_order
72          imag_strings(2^(n-1)+1) = "i" + n;
73          for i = 2^(n-1)+2: 2^(n)
74              imag_strings(i) = string(imag_strings(i-2^(n-1))) +
                    string(imag_strings(2^(n-1)+1));
75          end
76      end
77
78      string_mat = strings(0);
79      if self.zn(1) ~= 0
80          string_mat(1) = string(self.zn(1));
81      end
82      for i = 2:length(self.zn)
83          if self.zn(i) ~= 0
84              string_mat(end+1) = string(self.zn(i)) + " * " +
                    string(imag_strings(i));
85          end
86      end
87
88      if isempty(string_mat)
89          out = "0";
90      else
91          out = join(string_mat, "  +  ");
92      end
93  end
```