AAS 21-715

# HYPERDUAL NUMBERS FOR ARBITRARY ORBIT TARGETING

**Christopher Rabotin***

When designing maneuvers, one must compute the variation of the orbital characteristics of the final orbit with respect to a maneuver at the start of the transfer. These variations correspond to the partial derivatives of the destination orbital elements over the transfer arc. Most mission design software, such as STK Astrogator and NASA GMAT, compute these partial derivatives by finite differencing. This method is computationally heavy and its precision is limited to half of the precision of the computer. This paper demonstrates how to use dual number theory to compute the partial derivatives of any arbitrary orbital element with respect to the initial state of the trajectory, and paper provides a table for the variation of orbital elements with respect to Cartesian state vector, a useful reference for novel maneuver design. Validation of the dual numbers method for targeting is also detailed, with test cases and performance benchmarks between GMAT and the free software Nyx, a blazing fast astrodynamics toolkit available in Rust and Python.

## NOMENCLATURE

| | |
|---|---|
| $\Phi(t_i, t_f)$ | state transition matrix from time $t_i$ to time $t_f$ |
| $\frac{\partial X_f}{\partial X_i}$ | gradient of $X_f$ with respect to $X_i$ |
| $\epsilon$ | the dual number |
| $\dot{\gamma}$ | the time derivative of the $\gamma$ variable |
| $\mathbf{\Gamma}_k$ | the objectives vector of a targeting sequence at time $k$ |
| $\mathbf{U}_k$ | the control vector of a targeting sequence at time $k$ |

## INTRODUCTION

In the field of trajectory optimization, mission designers aim to achieve an orbit whose characteristics match specific conditions required by the mission. A set of control variables, $\mathbf{U}$, is modified at each iteration of the optimization algorithm such that the trajectory converges toward the objective vector $\mathbf{\Gamma}$. To ensure convergence, the partial derivatives of the objectives with respect to the control variables must be computed. More specifically, the objectives are computed for time $t_f$ when the objectives must be met, and the control variables are modified at time $t_i$ when the mission profile allows for them to be modified, cf. Equation 1.

$$\frac{\partial \mathbf{\Gamma}_f}{\partial \mathbf{U}_i} \tag{1}$$

A common approach, used by NASA GMAT and many others, is finite differencing, where each objective is recomputed after applying a small perturbation to the control variable. The ratio of the difference in objective over the perturbation leads to the partial derivative of that objective with respect to that control variable. Although this method works well in theory, in practice these partial derivatives are only precise to half of the precision of the computer because the order of magnitude of the perturbation is small and dividing the difference of two large numbers by a very small number leads to rounding errors.

*GNC Engineer, Masten Space Systems, Mojave, CA 93501, USA

Section  introduces the mathematical notions of dual numbers and hyperdual spaces, along with their error-free auto-differentiation properties. Then, the computation of the partial derivatives of orbital elements with respect to their Cartesian representation is illustrated. Section  is summarized by a reference table detailing which orbital elements are affected by which Cartesian elements. This reference is useful for mission designers regardless of whether their toolkit is enhanced with automatic differentiation. Subsequently, section shows that one can more precisely compute Equation 1 by it splitting up into a multiplication of simpler partial derivatives. This method is also less computationally heavy and the mathematics are easier to follow. This section also re-derives a Newton-Raphson differential corrector where the state transition matrix and the orbital element partial derivatives are computed using hyperdual numbers. Finally, section  provides a brief overview of the Nyx astrodynamics toolkit, programmable from Python and Rust. Nyx is free software (AGPLv3 license) for mission design and orbit determination with an emphasis on Monte Carlo analyses; its computations are validated against GMAT. This section also shows two distinct validation examples comparing the performance of Nyx to GMAT for the same maneuver design scenarios.

## DUAL NUMBERS AND HYPERDUAL SPACE

Dual number theory enables automatic differentiation. This may be used to compute the the state transition matrices to machine precision without relying on finite differencing.[1] For the sake of clarity, the introduction to dual number theory is repeated here.

### Dual number theory

Dual numbers are a flavor of complex numbers.[2] The ubiquitous set of complex numbers, $\mathbb{C}$, may be defined as follows, where $i$ is the imaginary number:

$$\mathbb{C} = \mathbb{R}[i] = \{z = a + bi \mid (a, b) \in \mathbb{R}^2, i^2 = -1\} \tag{2}$$

Similarly, we may define the set of dual numbers as follows, where $\epsilon$ is the dual number:

$$\mathbb{D} = \mathbb{R}[\epsilon] = \{z = a + b\epsilon \mid (a, b) \in \mathbb{R}^2, \epsilon^2 = 0 \text{ and } \epsilon \neq 0\} \tag{3}$$

Moreover, for $z = a + b\epsilon$ where $z \in \mathbb{D},\ (a, b) \in \mathbb{R}$, let us define the $real$ and $dual$ parts of a dual number such as

$$\begin{cases} real(z) = a \\ dual(z) = b \end{cases} \tag{4}$$

An auto-differentiation property emerges from the addition of this nilpotent element when applying a Taylor series expansion and recalling that $\epsilon^2 = 0$ (thereby eliminating all higher order terms).[3,4] Evidently, this result is only valid for values of $a$ where the function is differentiable.

$$\begin{aligned} f : \mathbb{D} \to \mathbb{D}\ , (a, b) \in \mathbb{R}^2 & \\ f(a + b\varepsilon) &= \sum_{n=0}^{\infty} \frac{f^{(n)}(a) b^n \varepsilon^n}{n!} \\ &= f(a) + b\frac{df(a)}{da}\varepsilon \\ &\begin{cases} real(f(a + b\epsilon)) = f(a) \\ dual(f(a + b\epsilon)) = b\frac{df(a)}{da} \end{cases} \end{aligned} \tag{5}$$

By choosing $b = 1$, the first derivative comes out for free by simply evaluating the function $f$.

**Hyperdual spaces**

We can further extend the dual numbers to a hyperdual space. Let us define a hyperdual space of size 2 as follows, where $\epsilon_x$ and $\epsilon_y$ are the dual numbers for the X and Y variables respectively. Subsequently, $\epsilon_j$ corresponds to the $j$-th dual number in a hyperdual space.

$$\mathbb{D}^2 = \mathbb{R}[\epsilon_x, \epsilon_y] = \{z = a + b\epsilon_x + c\epsilon_y + d\epsilon_x\epsilon_y \mid (a,b,c,d) \in \mathbb{R}^4, \epsilon_\gamma^2 = 0,\ \epsilon_\gamma \neq 0,\ \gamma \in \{x,y\},\ \epsilon_x\epsilon_y \neq 0\} \quad (6)$$

This mathematical tool enables auto-differentiation of multi-variate functions as follows, where $dual_\gamma$ corresponds to the $\gamma$-th dual number, i.e. the number associated with $\epsilon_\gamma$.

$$\begin{aligned} &f : \mathbb{D}^2 \to \mathbb{D}^2,\ (x,y) \in \mathbb{R}^2 \\ &\qquad \begin{cases} real(f(x+\epsilon_x,\ y+\epsilon_y)) = f(x,y) \\ dual_x(f(x+\epsilon_x,\ y+\epsilon_y)) = \frac{\partial}{\partial x} f(x,y) \\ dual_y(f(x+\epsilon_x,\ y+\epsilon_y)) = \frac{\partial}{\partial y} f(x,y) \end{cases} \end{aligned} \quad (7)$$

**Example**

Let us detail a computation example of a smooth multivariate polynomial function defined over all reals.

$$\begin{aligned} &f : \mathbb{R} \to \mathbb{R},\ (x,y) \in \mathbb{R}^2 \\ &\qquad f(x,y) = 2x^3 - 0.2y^2 + x \\ &\qquad \frac{\partial}{\partial x} f(x,y) = 6x^2 + 1 \\ &\qquad \frac{\partial}{\partial y} f(x,y) = -0.4y \end{aligned} \quad (8)$$

Let us extend the definition of this function to $\mathbb{D}^2$.

$$\begin{aligned} &g : \mathbb{D} \to \mathbb{D},\ (x,y) \in \mathbb{R}^2 \\ &\qquad g(x+\epsilon_x,\ y+\epsilon_y) = 2(x+\epsilon_x)^3 - 0.2(y+\epsilon_y)^2 + (x+\epsilon_x) \end{aligned} \quad (9)$$

Recalling that $\epsilon^2 = 0$,

$$\begin{aligned} (x+\epsilon_x)^2 &= x^2 + 2x\epsilon_x \\ (x+\epsilon_x)^3 &= x^3 + 3x^2\epsilon_x \end{aligned} \quad (10)$$

Hence, $g$ may be written as follows:

$$\begin{aligned} g(x+\epsilon_x,\ y+\epsilon_y) &= 2(x+\epsilon_x)^3 - 0.2(y+\epsilon_y)^2 + (x+\epsilon_x) \\ &= 2(x^3 + 3x^2\epsilon_x) - 0.2(y^2 + 2y\epsilon_y) + (x+\epsilon_x) \\ &= (2x^3 - 0.2y^2 + x) + (6x^2+1)\epsilon_x - 0.4y\epsilon_y \end{aligned} \quad (11)$$

As expected from Equation 7, the $dual_x$ part of $g$ corresponds to the partial of $f$ with respect to $x$, the $dual_y$ part of $g$ corresponds to the partial of $f$ with respect to $y$, and the $real$ part of the $g$ corresponds to $f$.

**COMPUTING ORBITAL ELEMENT PARTIAL DERIVATIVES IN HYPERDUAL SPACE**

Cartesian coordinates are a common non-singular orbital representation: $[x, y, z, v_x, v_y, v_z]^T$ where $v_x = \dot{x}$, $v_y = \dot{y}$, and $v_z = \dot{z}$. For an arbitrary orbital element $\gamma$, we can therefore compute its partial derivative with respect to its Cartesian representation $\mathbf{X}$ as represented in Equation 12.

$$\frac{\partial \gamma}{\partial \mathbf{X}} \tag{12}$$

First, one must generate a hyperdual space where one dual number exists for each Cartesian component of position and velocity: $\epsilon_\theta$ where $\theta = \{x, y, z, v_x, v_y, v_z\}$. Then, the orbit must be expressed in that space. Finally, the computation of the orbital element is computed in that hyperdual space accounting for the dual number theory properties defined in Equation 6.

**Orbital energy example**

The orbital energy $\xi$ is computed from the gravitational parameter $\mu$, the radius vector $\mathbf{r}$ and the norm of the velocity $||\mathbf{v}||$.

$$\xi = \frac{||\mathbf{v}||^2}{2} - \frac{\mu}{\mathbf{r}} \tag{13}$$

Equation 14 shows an example of an orbit whose semi-major axis is $7864$ km, inclination is $63.5°$, eccentricity is $0.124$, RAAN is $138.9°$, argument of periapsis is $45.9°$, and true anomaly is $87.8°$. Table 1 summarizes how each Cartesian component affects specific orbital parameters for the example orbit, where ++ means a large effect, + a small effect and 0 no effect.

$$o_{\mathbb{D}} = \begin{bmatrix} -2436.45 + \epsilon_x + 0 + 0 + 0 + 0 + 0 & \text{km} \\ -2436.45 + 0 + \epsilon_y + 0 + 0 + 0 + 0 & \text{km} \\ 6891.037 + 0 + 0 + \epsilon_z + 0 + 0 + 0 & \text{km} \\ 5.0886 + 0 + 0 + 0 + \epsilon_{v_x} + 0 + 0 & \text{km/s} \\ -5.0886 + 0 + 0 + 0 + 0 + \epsilon_{v_y} + 0 & \text{km/s} \\ 1.0 + 0 + 0 + 0 + 0 + 0 + \epsilon_{v_z} & \text{km/s} \end{bmatrix} \tag{14}$$

As per Equation 13, we start by computing the magnitude of the velocity vector and then its square. By setting $\mu = 398600.4415\ km^3 \cdot s^{-2}$ and $\mathbf{r}$ to the first three rows of Equation 14, Equation 15 details the computation of the partial derivative of the orbital energy with respect to each Cartesian orbital element. Note that these computations are performed by the open-source Rust package *hyperdual*.[5]

$$\begin{aligned} |\mathbf{v}| &= 7.2655 + 0.0000\epsilon_x + 0.0000\epsilon_y + 0.0000\epsilon_z + 0.7004\epsilon_{v_x} - 0.7004\epsilon_{v_y} + 0.1376\epsilon_{v_z} \\ |\mathbf{v}|^2 &= 52.7879 + 0.0000\epsilon_x + 0.0000\epsilon_y + 0.0000\epsilon_z + 10.1772\epsilon_{v_x} - 10.1772\epsilon_{v_y} + 2.0000\epsilon_{v_z} \\ \xi &= -25.3422 - 0.0021\epsilon_x - 0.0021\epsilon_y + 0.0060\epsilon_z + 5.0886\epsilon_{v_x} - 5.0886\epsilon_{v_y} + 1.0000\epsilon_{v_z} \end{aligned} \tag{15}$$

In other words, we have computed the orbital energy and its partials with respect to each Cartesian component, summarized in Equation 16.

$$\begin{gathered} \xi = -25.3422\ \text{km}^2/\text{s}^2 \\ \frac{\partial \xi}{\partial x} = -0.0021\ \text{km/s}^2 \quad \frac{\partial \xi}{\partial y} = -0.0021\ \text{km/s}^2 \quad \frac{\partial \xi}{\partial z} = +0.0060\ \text{km/s}^2 \\ \frac{\partial \xi}{\partial v_x} = 5.0886\ \text{km/s}^3 \quad \frac{\partial \xi}{\partial v_y} = -5.0886\ \text{km/s}^3 \quad \frac{\partial \xi}{\partial v_z} = 1.0000\ \text{km/s}^3 \end{gathered} \tag{16}$$

| Parameter | $\frac{\partial \cdot}{\partial x}$ | $\frac{\partial \cdot}{\partial y}$ | $\frac{\partial \cdot}{\partial z}$ | $\frac{\partial \cdot}{\partial v_x}$ | $\frac{\partial \cdot}{\partial v_y}$ | $\frac{\partial \cdot}{\partial v_z}$ |
|---|---|---|---|---|---|---|
| Argument of Latitude | + | + | + | ++ | ++ | ++ |
| Argument of Peripase | + | + | + | ++ | ++ | + |
| Apoapsis | ++ | ++ | ++ | ++ | ++ | ++ |
| $C_3$ | + | + | ++ | ++ | ++ | ++ |
| Declination | + | + | + | 0 | 0 | 0 |
| Eccentric Anomaly | + | + | + | + | + | + |
| Eccentricity | + | + | + | ++ | ++ | ++ |
| Energy | + | + | + | ++ | ++ | ++ |
| Flight Path Angle | + | + | + | + | + | + |
| Momentum magnitude ($H$) | ++ | ++ | ++ | ++ | ++ | ++ |
| $H_X$ | 0 | ++ | ++ | 0 | ++ | ++ |
| $H_Y$ | ++ | 0 | ++ | ++ | 0 | ++ |
| $H_Z$ | ++ | ++ | 0 | ++ | ++ | 0 |
| Inclination | + | + | + | + | + | + |
| Mean Anomaly | + | + | + | + | + | + |
| Periapsis | ++ | ++ | ++ | ++ | ++ | ++ |
| Right Ascension | + | + | 0 | 0 | 0 | 0 |
| RAAN | + | + | + | ++ | ++ | ++ |
| Semi Parameter | ++ | ++ | ++ | ++ | ++ | ++ |
| Semi major axis | ++ | ++ | ++ | ++ | ++ | ++ |
| True Longitude | + | + | + | ++ | ++ | ++ |

**Table 1. Effect of each Cartesian component on select orbital parameters for example orbit**

## HYPERDUAL NUMBERS FOR ARBITRARY ORBIT TARGETING

### State transition matrix

The state transition matrix (STM, $\mathbf{\Phi}$) is a linearization procedure of a dynamical system. It is the gradient of the spacecraft dynamics at a reference point valid only for a short period of time. Equation 17 shows that the STM from time $t_i$ to $t_f$ is equal to the partial derivative of the orbit (as a state vector) at time $t_f$ with respect to the orbit at time $t_i$, where $t_f > t_i$.

$$\Phi(t_i, t_f) = \frac{\partial \mathbf{X_f}}{\partial \mathbf{X_i}} \tag{17}$$

In practice, in an ordinary differential equations integration scheme like Runge-Kutta, one will compute the gradient of the accelerations affecting a spacecraft, evaluate those at each sub-step taken by the integrator, and perform a weighted-sum of these evaluations. The Nyx astrodynamics toolkit, described in section , uses this method for all its gradient computations, achieving 64-bit precision on a 64-bit architecture. This method has been thoroughly validated and leads to a significantly better linearization than a finite differencing method.[1,6]

### Newton-Raphson differential corrector

Although differential correctors are industry standards, the literature tends to focus on the mathematical concepts more than on the practical and implementation details.

A Newton-Raphson differential corrector applies the Newton-Raphson root finding algorithm to trajectory design to satisfy some objectives ($\mathbf{\Gamma}^*$) provided some control variables ($\mathbf{U}$).[7–10] The control variables must be independent of each other while the objectives must vary with a variation in the controls. The algorithm will iteratively compute small corrections to the control vector until the objectives are met with sufficient precision, typically defined by the user.

Let $\mathbf{\Gamma}^*$ be the vector of desired objectives at time $t_f$ and $\mathbf{\Gamma}_f$ be the achieved objectives at time $t_f$, and let $\mathbf{U}_i$ be the control vector at time $t_i$, where $t_f \geq t_i$. Given some control vector $\mathbf{U}_i$ applied at time $t_i$, the trajectory is propagated until time $t_f$ where the objectives $\mathbf{\Gamma}_f$ are computed. Then, the objective error vector is computed, as in Equation 18. Note that at the first iteration, the control vector is typically provided by the user, but a vector of zeros also works. Also note that, most commonly for impulsive maneuver design, the control vector is simply a change in the velocity vector, i.e. $\mathbf{U_i} = \Delta\mathbf{v}_i$.

$$\mathbf{\Delta\Gamma} = \mathbf{\Gamma}_f - \mathbf{\Gamma}^* \tag{18}$$

This error vector, $\mathbf{\Delta\Gamma}$, must be mapped back to the controls at time $t_i$. This is done by computing the Jacobian of the controls with respect to the objectives, sometimes called the sensitivity matrix.[9] Equation 19 shows a Jacobian with $n$ control variables and $m$ objectives. An easy method to remember how to organize the Jacobian is that the component of the numerator remains fixed navigating left to right (like an arrow pointing right) and the component of the denominator remains fixed going from top to bottom (like an arrow pointing downward).

$$\mathbf{J} = \frac{\partial \mathbf{\Gamma}_f}{\partial \mathbf{U}_i} = \begin{bmatrix} \frac{\partial \Gamma_{f_0}}{\partial U_{i_0}} & \cdots & \frac{\partial \Gamma_{f_0}}{\partial U_{i_n}} \\ \vdots & \ddots & \vdots \\ \frac{\partial \Gamma_{f_m}}{\partial U_{i_0}} & \cdots & \frac{\partial \Gamma_{f_m}}{\partial U_{i_n}} \end{bmatrix} \tag{19}$$

Equation 20 shows how the error in objectives and the inverted Jacobian is used to update the controls variables for the next iteration of the Newton-Raphson algorithm.

$$\begin{aligned} \delta\mathbf{U_{i+1}} = \left(\frac{\partial \mathbf{\Gamma}_f}{\partial \mathbf{U}_i}\right)^{-1} \cdot \mathbf{\Delta\Gamma} = \frac{\partial \mathbf{U}_i}{\partial \mathbf{\Gamma}_f} \cdot (\mathbf{\Gamma}_f - \mathbf{\Gamma}^*) \\ \mathbf{U_{i+1}} = \mathbf{U_i} + \delta\mathbf{U_{i+1}} \end{aligned} \tag{20}$$

Note that if the number of control variables and the number of objectives do not match, a Moore-Penrose pseudo-inverse must be used. If there are more control variables than objectives, Equation 21 applies, otherwise use Equation 22.

$$\mathbf{J}^{-1} \simeq \mathbf{J}^T \cdot \left(\mathbf{J} \cdot \mathbf{J}^T\right)^{-1} \tag{21}$$

$$\mathbf{J}^{-1} \simeq \left(\mathbf{J}^T \cdot \mathbf{J}\right)^{-1} \cdot \mathbf{J}^T \tag{22}$$

**Applying hyperdual numbers to targeting**

As seen previously, hyperdual numbers allow machine-precision computation of the state transition matrix. Moreover, they can also be used to compute the partial derivatives of any orbital element with respect to the Cartesian orbit representation. Therefore, we can rewrite each component of the Jacobian from Equation 19 with those intermediate steps as is done in Equation 23, where $\mathbf{X}_i$ is the Cartesian state at the time $t_i$ when the control vector $\mathbf{U}_i$ is applied and $\mathbf{X}_f$ is the Cartesian state at the time $t_f$ when the objectives $\mathbf{\Gamma}_f$ are computed. It is important to note that Equation 23 is used to replace each component of the Jacobian one at a time.

$$\frac{\partial \mathbf{\Gamma}_f}{\partial \mathbf{U}_i} = \frac{\partial \mathbf{\Gamma}_f}{\partial \mathbf{X}_i} \cdot \frac{\partial \mathbf{X}_i}{\partial \mathbf{U}_i} = \frac{\partial \mathbf{\Gamma}_f}{\partial \mathbf{X}_f} \cdot \frac{\partial \mathbf{X}_f}{\partial \mathbf{X}_i} \cdot \frac{\partial \mathbf{X}_i}{\partial \mathbf{U}_i} = \frac{\partial \mathbf{\Gamma}_f}{\partial \mathbf{X}_f} \cdot \Phi(t_i, t_f) \cdot \frac{\partial \mathbf{X}_i}{\partial \mathbf{U}_i} \tag{23}$$

For example, Equation 24 shows how to change the orbital energy $\xi$ by a change in velocity in the same frame as the state transition matrix, where $V_{i_x}$ is the X component of the velocity vector at time $t_i$. Equation 25 shows how this formulation would replace the full Jacobian where the objective is a specific orbital energy

and the control vector is simply all three components of the velocity in the frame of the STM. One will note that $\frac{\partial \mathbf{X}_f}{\partial \mathbf{X}_i} \cdot \frac{\partial \mathbf{X}_i}{\partial V_{i_x}}$ is simply the fourth column of the state transition matrix when the states are represented as Cartesian vectors. Therefore, in Equation 25 the simplification to $\frac{\partial \mathbf{X}_f}{\partial V_{i_x}}$ is used.

$$\frac{\partial \xi_f}{\partial V_{i_x}} = \frac{\partial \xi_f}{\partial \mathbf{X}_f} \cdot \frac{\partial \mathbf{X}_f}{\partial \mathbf{X}_i} \cdot \frac{\partial \mathbf{X}_i}{\partial V_{i_x}} \tag{24}$$

$$\frac{\partial \xi_f}{\partial \mathbf{V}_i} = \begin{bmatrix} \frac{\partial \xi_f}{\partial \mathbf{X}_f} \cdot \frac{\partial \mathbf{X}_f}{\partial V_{i_x}} & \frac{\partial \xi_f}{\partial \mathbf{X}_f} \cdot \frac{\partial \mathbf{X}_f}{\partial V_{i_y}} & \frac{\partial \xi_f}{\partial \mathbf{X}_f} \cdot \frac{\partial \mathbf{X}_f}{\partial V_{i_z}} \end{bmatrix} \tag{25}$$

Continuing with this example, one would perform the pseudoinverse as detailed in Equations 21 and 22. Applying a Newton-Raphson differential correction, a single propagation would be performed from $t_i$ to $t_f$, time at which the energy error would be computed. The iteration is described in Equation 26.

$$\delta \mathbf{V_{i+1}} = \left( \frac{\partial \xi_f}{\partial \mathbf{V}_i} \right)^{-1} \cdot \mathbf{\Delta} \xi \qquad \mathbf{V_{i+1}} = \mathbf{V_i} + \delta \mathbf{V_{i+1}} \tag{26}$$

## VALIDATION

Nxy Space is a blazing fast toolkit for mission design and orbit determination, programmable in Rust and soon in Python.[11] A searchable mathematical specification and algorithmic description (MathSpec) is freely available online and regularly updated.[7] This MathSpec also details the validation of each algorithm as compared to NASA GMAT. Nyx is typically at least two times faster and does not suffer from common software bugs like memory leaks thanks to the Rust programming language. Moreover, Nyx is open-sourced under the AGPLv3 license and can be easily executed in a containerized environment such as Docker.[12,13] This allows Nyx to be especially well suited for Monte Carlo analyses on the cloud.[14]

Table 2 shows two validation test cases of using hyperdual numbers for orbital targeting. In these examples, an impulsive maneuver is computed and the results are compared to the same setup in GMAT. For simplicity, two-body dynamics are assumed. The initial state has a semi-major axis of $8000$ km, an eccentricity of $0.2$, an inclination of $30°$, a RAAN and argument of periapse of $60°$, and the orbit is set at periapse (i.e. the true anomaly is zero).

**Table 2. Hyperdual targeting validation**

| Objective | Finite differencing | | | Hyperdual targeting | | |
|---|---|---|---|---|---|---|
| | $\Delta v$ (m/s) | Iterations | Time (s) | $\Delta v$ (m/s) | Iterations | Time (s) |
| SMA = 8100 km | 35.504 | 3 | 0.326 | 35.505 | 3 | 0.099 |
| SMA = 8100 km and ECC = 0.4 | 3116.1 | 8 | 0.226 | 3094.0 | 8 | 0.104 |

## CONCLUSION

This paper describes a novel approach to arbitrary orbital element targeting without employing finite differencing. One of the key advantages of the hyperdual formulation is that it allows for the computation of the state transition matrix at high-fidelity, regardless of the complexity of the dynamics and without any significant impact on computational time. Another key advantage is faster convergence speeds without having to tune perturbation values. A limitation of this method is that the state transition matrix must provide a valid linearization of the orbit dynamics throughout the transfer arc. Therefore, future research should investigate using this method for the correction of the orbital elements using multiple-shooting so as to avoid any break in the linearization.

## ACKNOWLEDGMENTS

Special thanks to Sai Chikine and Tim Sullivan for the conversation which sparked this research. I would like to acknowledge Eduard Heijkoop, Eva Pettinato, and the GNC team at Masten Space Systems for their review.

## REFERENCES

[1] C. Rabotin, "Application of Dual Number Theory to Statistical Orbit Determination," *AAS Astrodynamics Specialist Conference 2019, AAS-19-716*, 2011.

[2] J. Rooney, "On the Three Types of Complex Number and Planar Transformations," *Environment and Planning B: Planning and Design*, Vol. 5, No. 1, 1978, pp. 89–99, 10.1068/b050089.

[3] F. Messelmi, "Analysis of dual functions," *Annual Review of Chaos Theory, Bifurcations and Dynamical Systems*, Vol. 4, 2013, p. 37.

[4] J. A. Fike and J. J. Alonso, "The Development of Hyper-Dual Numbers for Exact Second-Derivative Calculations," *AIAA paper 2011-886, 49th AIAA Aerospace Sciences Meeting*, 2011.

[5] A. Trent and C. Rabotin, "hyperdual - Rust package registry," 2021.

[6] C. Rabotin, "Nyx - Math Spec - State transition matrix," 2021.

[7] C. Rabotin, "Nyx - Math Spec," 2021.

[8] NASA GMAT Team, "GMAT - Differential Corrector," 2020.

[9] AGI STK Team, "STK - Technical Notes - Differential Corrector," 2020.

[10] N. L. O. Re (Parrish), "Low Thrust Trajectory Optimization in Cislunar and Translunar Space," 2018.

[11] C. Rabotin, "Nyx website," 2021.

[12] C. Rabotin, "Nyx - AGPLv3 License," 2021.

[13] C. Rabotin, "Nyx source code - Github," 2021.

[14] C. Rabotin, "Nyx Tutorial in Docker image (Gitpod)," 2021.