ARTIFICIAL NEURAL NETWORK TRAINING TO CORRECT FOR SOLAR GRAVITY POTENTIAL PERTURBATION IN CLOSE ORBITS

a project presented to
The Faculty of the Department of Aerospace Engineering
San Jose State University

in partial fulfilment of the requirements for the degree
Master of Science in Aerospace Engineering

by

Mayra Lopez-Thibodeaux

July 2021

To be Approved by

Professor Jeanine Hunter

Faculty Advisor

ABSTRACT

The research done in this project aims to lay the basis for building a grid of solar gravitational potential perturbation at every point in the phase space on a close orbit around Venus for autonomous space travel. Two different approaches have been provided to reach this goal. The most convenient one is the extraction of output weights from a Long Short-Term Memory (LSTM) Recurrent Neural Network (RNN) trained with data generated by deep space orbit determination and optimization software GMAT (General Mission Analysis Tool). The LSTM train data was generated by simulating the solar potential perturbed and unperturbed orbits with an initial state vector corresponding to that of the Venera D mission but at a 10,000 km larger radius of pericenter. This same data was used to provide a second approach for building the solar perturbation mapping by taking the difference between each of the perturbed and unperturbed position vectors of the spacecraft at every step of its orbit around Venus and towards the perturbed state. The training of the LSTM RNN was achieved with an accuracy between 89% and 92% with two LSTM layers of 50 and 10 units respectively, a 0.46-rate regularizer and a TimeDistributed wrapped Dense layer of 3 units. A burn analysis was initiated using GMAT as well to provide a technique to farther develop this investigation. The analysis was done by solar perturbation drifting of the spacecraft with different types of orbits and showed that solar perturbation could aid to reach the goals of a mission with less or not fuel once in close orbit around the planet.

Acknowledgments

The work done in this project was possible by the collaboration of a group of great computer scientists, aerodynamists and Professors. My deepest appreciation to all of them. Special thanks to my advisor for this project, Professor Jeanine Hunter for her support and always wise advice during this research, and to both Dr. Nikos Mourtos and Professor Hunter for maintaining the Aerospace Department at San Jose State where they bring in a wide viarety of projects, events, and opportunities for students to grow and thrive. I highly appreciate having the opportunity to closely having collaborated with a great, supportive colleague, Takoua Bejaoui, SJSU Graduate student from the Mechanical and Aerospace (AE) Engineering departments, who played a major role in helping with the development of the RNN LSTM presented in this paper and provided guidance for the development of some of the GMAT simulations used in this report. Thanks to Dr. Fabio Di Troia, Professor of the Computer Science (C.S) department at SJSU and Nelson Wong, SJSU Graduate students from the Computer Engineering (C.E) and C.S departments and one of the leaders of the Robotics Team, who initiated the first machine learning summer workshop based on the CS231 Stanford class' materials that thought me the foundations of my knowledge about Artificial Neural Networks. I highly appreciate their well thought advise and continuous support and guidance during this project. Thanks to Theodore Hendricks, a SJSU alumni from the AE department, for his guidance with GMAT and insight ideas about some of the orbital mechanics behind this project. Thanks to my husband Christopher Thibodeaux who has made it possible for me to reach this point in my educational career and has been a tireless supporter in all senses during my education at SJSU and much of my life. Special thanks to my beloved mother, Librada Felix de Lopez who is my most powerful motivator, example to look at and role model to follow to grow in my years of life.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS, SYMBOLS AND UNITS

Abbreviations

GMAT	General Mission Analysis Tool
NN	Artificial Neural Network
i7	Intel seven
MRS	Mean Square Error
ReLU	Rectified linear units
RNN	Recurrent Neural Network
LSTM	The Long Short-Term Memory
BPTT	Back propagation through time
API	Application Programming Interface
JSDT	Joint Science Definition Team
LLISSE	Long-Lived, In-Situ Solar System Explorer
BPTT	Back-propagation through time
KVTK	Oxygen/Hydrogen Heavy Class
Briz	Breeze-K, KM and M
UTC	Universal Time
J2000	Reference frame based on the Earth's equator and equinox on January 1, 2000 at 12:00:00 TBD
ANSI	American National Standards Institute
GUI	Graphical user interface
RK	Runge-Kutta
XOR	Exclusive or function

SGD	Stochastic gradient descent

Symbols

I_1 and I_2	States of intersection on a Poincaré map
$\boldsymbol{x}$ or $\boldsymbol{X}$	Neural network input vector
$\boldsymbol{w}$	Weight vector
$\boldsymbol{W}$	Weight matrix
b	Neuron bias
$\odot$	Element-wise multiplication
f	Activation function,
y or Y	Neuron output
$\hat{y}$	Predicted neuron output
$\boldsymbol{\theta}$	Neural network parameters
L	Loss function
U	Weight matrix between input and hidden layers
V	Weight matrix between the hidden and output transition
γ	Neural network learning rate
x, y and z	Position vector components in Cartesian coordinates
v_x, v_y, v_z	Velocity vector components in Cartesian coordinates
Δv	Change in velocity and fuel burn
a	Semi-major axis
e	Eccentricity
i	Inclination

ω	Argument of perigee
Ω	Longitude of the ascending node
ν	True anomaly
r_p	Radius at pericenter or perigee
τ	Orbital period
G	Gravitational constant equal to $6.67\text{x}10^{-20}\ \frac{\text{km}^3}{\text{kg sec}^2}$
M	Mass of Venus
E_y, E_x	XY plane on reference frame of Earth
V_y, V_x	XY plane on reference frame of Venus
Ω_{VE}	Right ascension of Venus with respect to Earth
Ω_{QV}	Right ascension of spacecraft's orbit with respect to Venus
$\mathbf{r}, \dot{\mathbf{r}}, \ddot{\mathbf{r}}$	Position vector as a function of time and its first and second derivatives in a Newtonian reference frame
x, y, z	Cartesian coordinates of 3-D space in the Newtonian reference frame
$\ddot{x}, \ddot{y}, \ddot{z}$	Second time derivatives of Cartesian coordinates of 3-D space in the Newtonian reference frame
m	Mass of the spacecraft and
$\mathbf{F}$	Force vectors acting on mass m
t	Time
h	Step in time
f^*	Feedforward neural network approximation function
ϕ	Non-linear transformation of input $\boldsymbol{x}$
$\boldsymbol{c}$	Bias vector

E	Cost function
Z	Gradient-based method input
D	Gradient-based method desired output
p	Number of inputs in the system
$J(\boldsymbol{\theta})$	MSE function evaluated over a full training set
A	A given matrix A
$\boldsymbol{h}$	Vector of hidden units
M	Function learned by the machine
η	Gradient-based method learning rate

Units

GHz	Gigahertz
km	Kilometer
km/hr	Kilometer per hour
kg	Kilograms
sec	Seconds
min	Minute
hr	Hour

Chapter 1: Introduction

1.1 Motivation

Space exploration has become crucial in the advancement of different fields of science. It provides scientists and engineers with vast amounts of data to be analyzed, and the opportunity to make new discoveries. For instance, data gathered from the Hubble Space Telescope, launched in 1990 from the John F. Kennedy Space Center in Florida, has led to the discovery of the age and size of the universe, galaxies in the early universe and their classification, new moons of Pluto, understanding seasons of other planets and advancing exoplanet science [1]. Furthermore, space exploration is important for national security communication using surveillance satellites and protection from possible asteroid impacts. It has also led to more inventions such as flexible but resilient alloys that can be folded into a rocket to be popped open from a satellite, freeze-dried food for the Space Station and plastic-coated covers with a metallic reflecting agent that reflects back 80 percent of the user's body heat to keep astronauts warm [2].

Machine learning has become essential to expand our horizons for space exploration. The *learning* of a machine refers to the self-improvement capability an algorithm obtains by finding patterns or predicting unknowns from data. Machine learning already has advanced applications in different important fields such as aviation, healthcare and banking, and is expected to enhance future space travel and exploration since it can control massive amounts of data, find dataset patterns on planetary imaging, and predict spaceship conditions. In the fields of space travel and exploration, machine learning can be mainly applied in navigation and rocket landing, analysis of visual data, and data transmission [3]. In this project, machine learning artificial neural networks is applied to spacecraft navigation in close orbit around Venus, with the purpose of making it applicable to close orbit navigation around other celestial bodies.

Presently, artificial neural networks or Neural Networks (NN) machine learning is already revolutionizing the classification of galaxies leading to a deeper understanding of the universe. Similarly, NN machine learning can improve current technology in relative spacecraft and satellite motion control. Spaceships have only an instant for action control that requires taking into account and processing geometric and kinematical location information. As space missions become more frequent and complex, requiring them to go farther in space, the demand for fast, self-adjusting navigation based in machine learning will grow [3].

Space travel and exploration require orbital transfers at various stages of a space mission. These transfers can be done via typical orbital mechanics maneuvers or perturbation-aided maneuvers. The latter provides a convenient cost-free impulse to aid orbital transfers, even though the perturbations which will change the path of the spacecraft toward the desired direction require an ideal location. In the case of capture orbits around Mars, solar gravity perturbations significantly deviate the path of a spacecraft at large apoareion [4]. This is also the case for orbits with large pericytherion around Venus given its closer proximity to the Sun. Hence, solar gravity perturbation is the focus of this paper and will be investigated via the different output parameters of an ideal type of NN that best suits the needs of the investigation at hand.

1.2 Literature Review

For a spacecraft away from the sphere of influence of a planet, two-body astrodynamics becomes inaccurate in determining its path given the gravitational perturbations due to other celestial bodies. These perturbations have been extensively investigated and used to the advantage of space travel, e.g., gravity assist maneuvers where the spacecraft is required to have close proximity to the perturbing body. Some gravity perturbation methods use a dynamical system based on stable and unstable sets of points of gravitational attraction in phase space, or manifolds, that serve as a guide on space for cheap transfers. Gravity perturbation methods have proven to be successful for the GRAIL mission in low-energy Moon-Earth transfers and for the ATEMIS mission transfer between different libration points. Other perturbation methods also use the forces present in nature to do ballistic captures to favor transfers, e.g., the Hiten mission, which was launched into a highly elliptical Earth orbit that intersected the Moon's orbit [4][5].

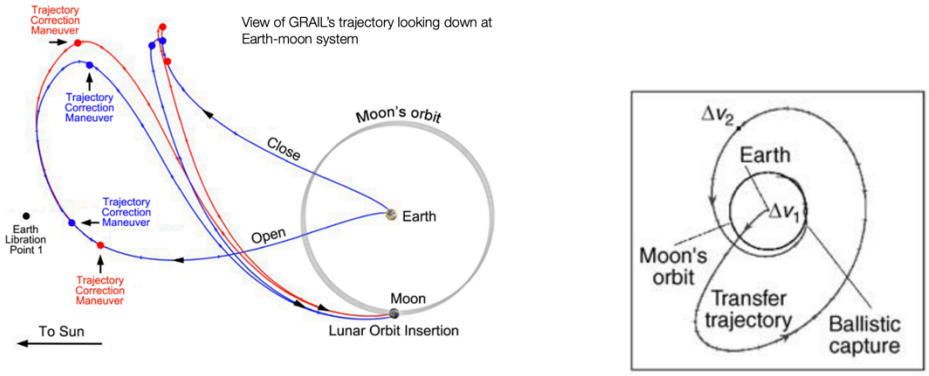


Figure 1. Perturbation methods used in favor of space travel [5], [6].

Note: The figure on the left shows GRAIL-A (red) and GRAIL-B (blue) trajectories for a launch at the open and close of the launch period. The low-energy trajectories leave Earth following a path towards the Sun, passing near the interior Sun-Earth Lagrange Point 1 (Earth Libration Point 1) before heading back towards the Earth-moon system [5]. The figure on the right shows a Hiten-like low energy trajectory in the geocentric inertial frame using ballistic capture by the Moon's orbit [6].

The Space Manifold Dynamics approach to solving astrodynamics problems makes it possible to systematically analyze the nature of a plausible mission approach by describing phase space around Lagrangian points - points where the gravitational force from the orbiting bodies cancels out. It also provides determination of Lagrangian points, interplanetary and low-energy mission determination, station-keeping strategies that keep a spacecraft in the assigned orbit, transfer determination between Lagrangian points or eclipse-avoidance strategies design [7]. In addition, ballistic captures use the gravitational pull of an orbiting body to aid spacecraft transfer when it is inserted into the orbit of a celestial body orbiting the target planet or moon at a greater velocity of that of the spacecraft. All perturbation methods mentioned earlier are restricted by the three-body astrodynamics problem. The gravitational potential influence of a secondary celestial body over a spacecraft becomes significant enough to add up to that of the primary orbiting body changing the path of the spacecraft.

Currently, there are no analytical solutions for orbital transfers involving the three-body problem. Instead, these solutions rely on numerical integration methods applied to complex astrodynamical systems, which involve strenuous computational loads [4]. Nevertheless, mission design platforms based on numerical methods for orbit propagation have become an important part of space mission design. One such platform that is widely used in the aerospace industry due to its open source and high capability nature is that developed by NASA and private industry in the last decade called General Mission Analysis Tool (GMAT). GMAT was qualified to be used in NASA Missions Operations rooms for operational planning summation in 2014. Hence, GMAT simulation calculations have been proven to be qualified for a level of reliability that will pose no undue risk on a spacecraft mission design [8]. GMAT is a platform designed for the constrained or unconstrained optimization of deep space missions of spacecraft trajectories. This system includes numerical integrators with initial and boundary value solvers for propagation as a function of time. It is also capable of synchronizing the epochs of multiple spacecrafts, plotting their trajectories and parameters, and saving them to files for comparison and processing [9]. Some of the resources in GMAT to model space missions are spacecraft, propagators, and optimizers that can be configured to fit specific models and applications that simulate the motion of a spacecraft and mission events chronologically.

Table 1 summarizes the key features of GMAT:

Table 1. GMAT key features (modified from [8]).

Basic Models	
Spacecraft model	• Orbit state • Kinematic attitude • Mass properties • Viz properties • Attached hardware
Solar system	• User-defined bodies • Libration points • Barycenters • SPK & DE ephemerides
Coordinate systems	• Inertial • Body-fixed • Relative • Rotating

Propagation	
Numerical integrators	• Runga-Kutta (several) • Prince-Dorman (several) • Adam-Bashforth-Moulton • Prince-Dorman
Force Models	• Third-body point-mass • Central-body non-spherical • Atmospheric drag • Solar radiation pressure • Earth tides • Relativistic corrections
SPK ephemeris propagator	

SPK & DE ephemerides: data banks for celestial navigation that gives the trajectory of celestial bodies and satellites
Viz: visualization

The component that simulates spacecraft motion in GMAT is called a Propagator. A GMAT Propagator is either of numerical integration type or ephemeris type (see Table 1 for description of ephemeris type). The former type of propagator offers a list of numerical integrators based on the Runge-Kutta and prediction corrector methods and requires a force model. A force model simulates the natural forces in the environment that affect the spacecraft dynamics. Once configured, the force model is added to the Propagator to solve for the astrodynamics equations numerically and may include relativistic corrections, atmospheric drag,

point mass, solar radiation, gravity and tie models [10]. Figure 2 and Table 2 below show a description of GMAT numerical integrators and an example of the setup of a Runga-Kutta89 Propagator with a force model with Venus and the Sun as the point masses influencing a spacecraft.

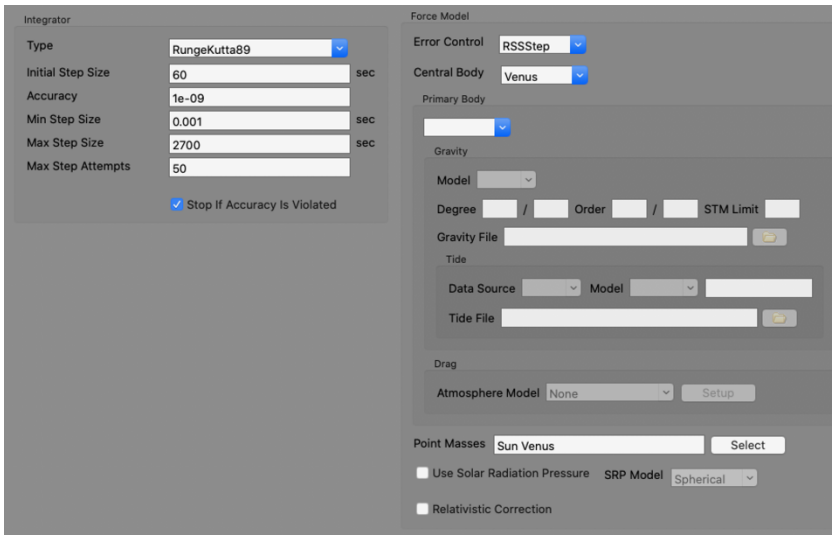


Figure 2. GMAT propagator setup.

Table 2. GMAT numerical integrators overview [10].

Option	Description
RungeKutta89	An adaptive step, ninth order Runge-Kutta integrator with eighth order error control. The coefficients were derived by J. Verner. Verner developed several sets of coefficients for an 89 integrator and we have chosen the coefficients that are the most robust but not necessarily the most efficient.
PrinceDormand78	An adaptive step, eighth order Runge-Kutta integrator with seventh order error control. The coefficients were derived by Prince and Dormand.
PrinceDormand853	An adaptive step, eighth order Runge-Kutta integrator with 5th order error control that incorporates a 3rd order correction, as described in section II.10 of "Solving Ordinary Differential Equations I: Nonstiff Problems" by Hairer, Norsett and Warner. The coefficients were derived by Prince and Dormand. This integrator performs surprisingly well at loose Accuracy settings.
PrinceDormand45	An adaptive step, fifth order Runge-Kutta integrator with fourth order error control. The coefficients were derived by Prince and Dormand.
RungeKutta68	A second order Runge-Kutta-Nystrom type integrator with coefficients developed by by Dormand, El-Mikkawy and Prince. The integrator is a 9-stage Nystrom integrator, with error control on both the dependent variables and their derivatives. This second order implementation will correctly integrate forces that are non-conservative but it is not recommended for this use. See the integrator comparisons below for numerical comparisons. You cannot use this integrator to integrate mass during a finite maneuver because the mass flow rate is a first order differential equation not supported by this integrator.
RungeKutta56	An adaptive step, sixth order Runge-Kutta integrator with fifth order error control. The coefficients were derived by E. Fehlberg.
AdamsBashforthMoulton	A fourth-order Adams-Bashford predictor / Adams-Moulton corrector as described in Fundamentals of Astrodynamics by Bate, Mueller, and White. The predictor step extrapolates the next state of the variables using the the derivative information at the current state and three previous states of the variables. The corrector uses derivative information evaluated for this state, along with the derivative information at the original state and two preceding states, to tune this state, giving the final, corrected state. The ABM integrator uses the RungeKutta89 integrator to start the integration process. The ABM is a low order integrator and should not be used for precise applications or for highly nonlinear applications such as celestial body flybys.

The GMAT Propagation tool also requires configuring the spacecraft to be simulated for propagation. To do so, initial conditions such as the epoch and classical orbital elements must be

determined. Figure 3 shows the setup of a spacecraft on a pronounced eccentric orbit around Venus with its initial position and epoch, the corresponding propagation plot in space (right) and on the surface of Venus (down left) with the propagation setup of Figure 2.

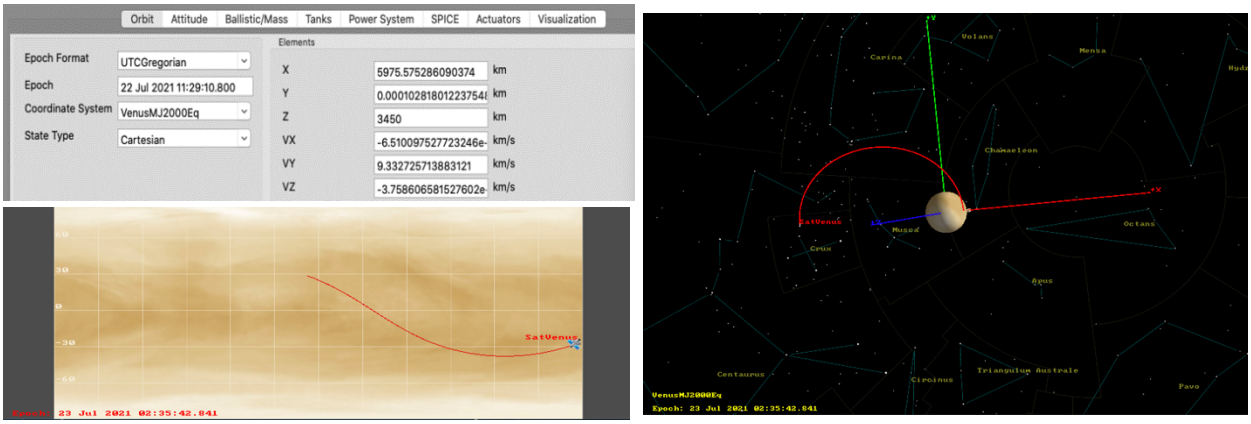


Figure 3. Spacecraft setup in GMAT and its corresponding plots.

The GMAT platform is an example of the required complexity of numerical integration applications to develop astrodynamical systems involving three-body dynamics. An alternative method that aids with the significant computational load of numerical integration simulations is the use of NN. NNs can be used to develop a database of the dynamics within a system which can be used as a roadmap to find convenient trajectories and transfers [11]. The application of NNs in the past on some fields of aerospace engineering have been proven to be successful. Such is the case of the investigation done in [12] at NASA Glenn Research Center in 2002 where the application of NNs to shorten the design cycle in the design and optimization of aircraft engine propulsion systems and monitoring the microgravity quality onboard the International Space Station were researched. In the case of NN application for design and optimization, NNs were used to provide mappings for fast analysis and design, which enabled the simulation of a highly complex model near real time within an acceptable 5% error. For system monitoring, NNs were used to develop a monitoring system tool that helped researchers to remotely assess how the space environment affected their experiments. This second investigation led to an understanding about how Back Propagation NNs can recognize new patterns and avoid the misclassification of patterns while accounting for multi-dimensional ranges of neighboring clusters [12].

De Smet et al investigate solar gravity perturbation driven transfer via the creation of a database of solutions for a spacecraft in closed orbit around Mars [11]. This methodology uses a sparse grid of numerical integrated points to train a NN in a small subset of phase space to be later developed into larger areas and other part of the solar system. The weights and biases of the NN capture the developed database, which can then be used to identify transfers for different applications. In this problem, numerical integrations of a periareion Poincaré map are used to compute the sparse grid data points, and the developed database using NNs is tested with GMAT and Monte Carlo simulations. A Poincaré map is the intersection of a periodic orbit in the state space of a periodic dynamical system with a subspace transversal to the flow of the system (refer to Figure 4 for its visual depiction). This problem is initially simplified by using the Circular Hill

System, which assumes Mars orbiting the Sun in a circular orbit at a constant angular velocity, making the problem time invariant. This simplification allows one to readily obtain a set of initial solutions to be then adjusted using the Eccentric Hill System (which assumes the same conditions as the circular Hill System except by Mars orbiting the Sun in an elliptical orbit) via a second NN. The proposed transfer under study starts at the periapse of the incoming areocentric parabola with a small maneuver to reduce the eccentricity to less than one to get it into an elliptical orbit [4][11]. The spacecraft orbit is then circularized into its final orbit by a second maneuver that aims rendezvous with Phobos (see Figure 4 below).

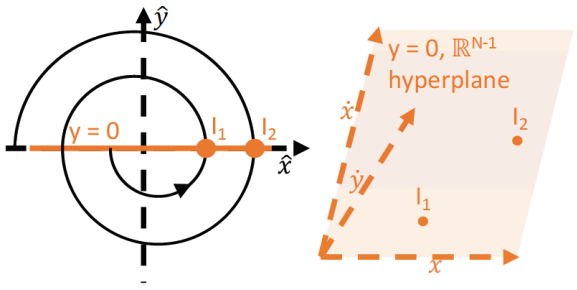


Figure 4. Depiction of a Poincaré map with states of intersection I_1 and I_2 [4].

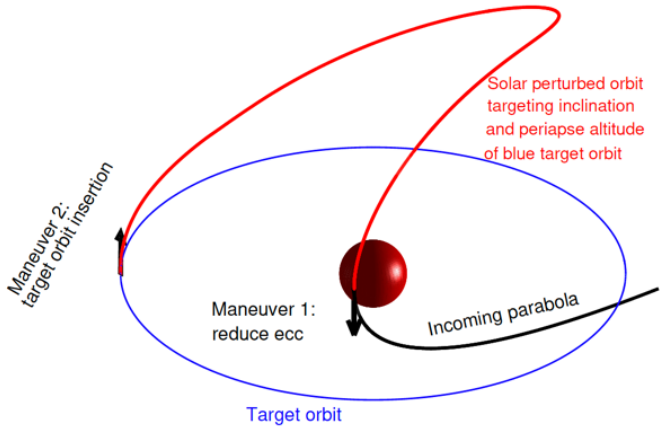


Figure 5. Propose transfer type in [11].

In order to pose an example of the capability of the use of NNs in this type of problem, the NN architecture and training results using the circular Hill System in [11] are presented in Table 3 and Figure 6:

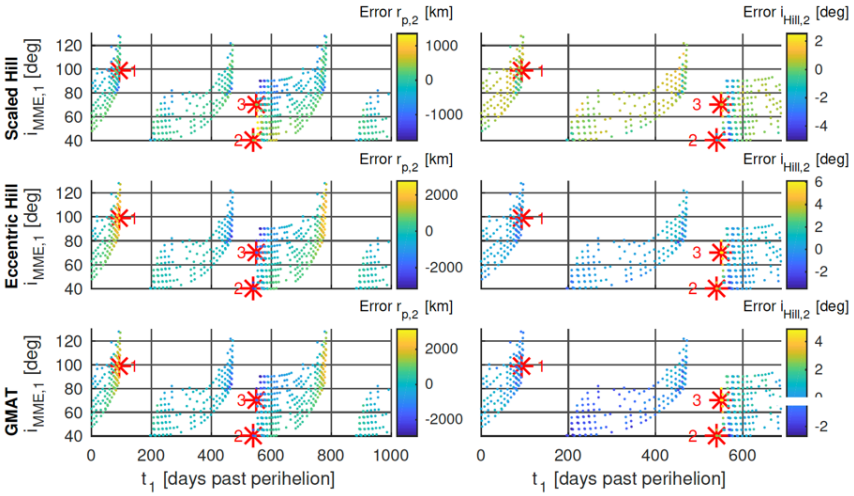


Figure 6. Error Summary in radius and inclination at periareion for the predicted transfer to arrive at Phobos [11].

Figure 6 compares the results in [11] between the scaled circular Hill system (which is an approximation to the eccentric Hill dynamics), the eccentric Hill system and numerical integration simulation with GMAT. The results show significant errors on the periapse and inclination on the trajectory predicted for arrival to Phobos in the scaled Hill model. The main sources of error are the NN predictions and the difference between the scaled circular and eccentric Hill systems. Hence, the simplification of the scaled circular Hill system, shown as "Scaled Hill" on the top in Figure 6, introduced approximate errors in the calculation of the periapse between 3000-6000 km and corresponding errors in inclination close to -3^{o} and 6^{o} This error could be fixed by adding more neurons, layers and training data to the NN. On the next row, the solutions from the eccentric Hill system using NN and numerical integration via GMAT show almost no difference. The application of NN to solve for the circular model on a small subset of phase space reduced the number of required integrated transfers from 8.4 million to 74,000 with a training time of 50 minutes on a single core of a 2.5 GHz Intel Core i7 processor, rather than 10 days using numerical integration. NN application reduced the number of integrated transfers from 3.04 billion to 492,000 for the eccentric model case with a training time of one day rather than 3570 days using numerical integration [11].

Table 3. Architecture and training results of NN using the circular Hill System [11].

Output parameter	Hidden layer neuron sizes	Final MSE training data	Final MSE validation data	Final MSE test data	Training time [min]
$r_{p,2}$	15x15	6.10e-6	6.27e-6	6.74e-6	3.03
TOF	15x15	3.96e-7	4.31e-7	4.15e-7	3.14
$i_{\text{Hill},2}$	25x25	2.38e-5	2.39e-5	3.37e-5	17.30
$\cos\Omega_{\text{Hill},2}$	25x25	2.37e-5	4.75e-5	4.96e-5	12.64
$\sin\Omega_{\text{Hill},2}$	25x25	3.65e-5	4.63e-5	5.24e-5	10.19

Table 3 shows a basic architecture and typical results of a multilayer NN using the back-propagation approach. The hidden-layer architecture of an NN as well as how the Mean Square Error (MRS) will be used are discussed in this section. Training, validation and test data corresponds to the different steps an NN follows for training, validating and testing its learning process. An NN is a machine learning model that is composed of simple computational units called neurons. Each of these neurons takes an input at its incoming edge, multiplies it by a randomly assigned weight and applies a non-linear function called the activation function to the weighted sum to produce an output. The following figure shows a visual representation of an NN where *x, w, b,* $\odot$*, f* and *y* represent input vector, weight vector, neuron bias, element-wise multiplication, activation function, and neuron output respectively [13].

$$y(x) = f(\boldsymbol{w} \odot \boldsymbol{x} + b)$$

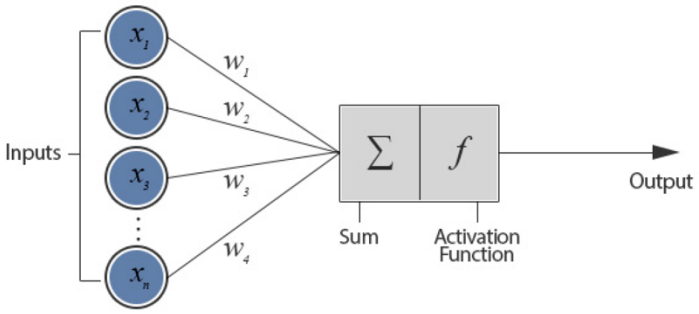


Figure 7. The structure of a NN [13].

The non-linearity of the output corresponding to the input of every neuron is introduced by the activation function. Commonly used activation functions are the tanh, the rectified linear units or ReLU, the sigmoid and the linear or identity function. Figure 8 below shows the behavior of these functions.

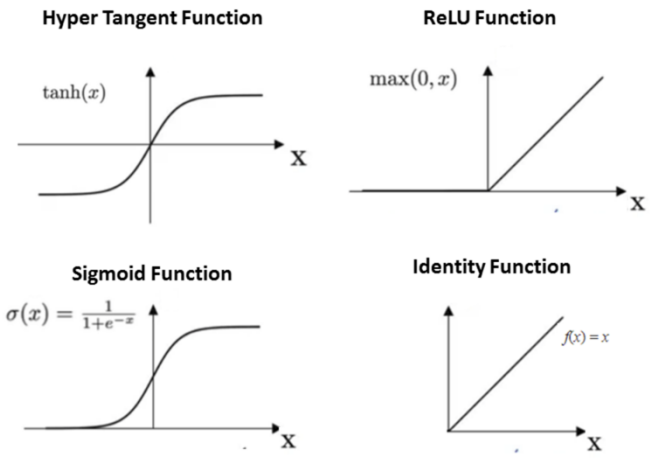


Figure 8. Typical NN activation functions [14].

The forward feed of a NN is composed of a network of layers of neurons interconnected layer by layer with assigned weights at each edge. The first layer is called the input layer because it takes the input data to train the NN. The rest of the layers of the NN are called hidden layers. The outputs are calculated per neuron starting at the input layer and forward to the last layer on the opposite side (see Figure 9). In order to facilitate learning by self-correcting, the NN uses a loss function, which measures the difference between the desired and the network outputs. A common loss function used in regression models is the mean square error or MSE given by the following equation:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \qquad (1.1)$$

where y_i represents the true value, and $\hat{y}_i$ the predicted value [13].

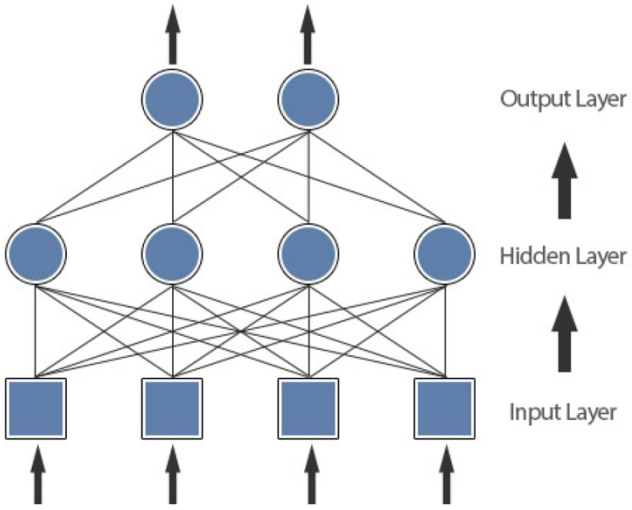


Figure 9. Representation of a forward feed of a single-hidden-layer NN [13].

An NN uses gradient descent optimization to learn. In this process, the network tunes its parameters such that it minimizes the loss function by calculating the gradients of the loss function with respect to each of these parameters. Gradients are calculated via the back-propagation method, which is based on the chain-rule of derivatives. The gradient represents the change in the loss value of each parameter. The network parameters ($\boldsymbol{\theta}$) are adjusted in the opposite direction of the gradient by updating a scalar called the learning rate (γ) via equation

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \gamma \frac{\partial L(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \qquad (1.2)$$

The process is repeated by passing iteratively over the training data, and each of these passes is called an epoch. The parameters of a NN are the weights and biases and they are learned by training. Other parameters of the NN such as learning rate, decay, batch size and dropout are called hyperparameters and are to be appropriately set by the user before training [13].

Recurrent NNs or RNNs are of special interest to solve many astrodynamical problems because they use both current input and past information while making future predictions. They are capable of retaining dependencies across time steps by learning how to retain relevant information. RNNs do so by processing the input elements one at a time while maintaining a hidden state vector that acts as the memory for past process information. Hence, RNNs are suitable for sequential data [13].

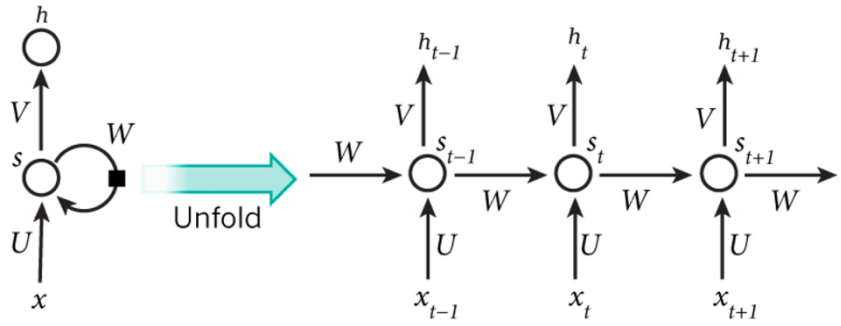


Figure 10. A standard RNN [13].

Figure 10 shows the schematic of a standard RNN with hidden state vector S which keeps a memory of the previous elements of the sequence. A current input element x_i is received at time t and state S_{t-1} from the previous time step, which is then updated S_t and the final network output h_t is calculated. U is the weight matrix between input and hidden layers and W is the weight matrix of the recurrent transition between hidden states. V is the weight matrix between the hidden and output transition. The equivalent of the back-propagation method for RNNs is the back-propagation through time (BPTT) method. This method involves multiplying the error gradient over every time step, which causes gradients to become either too large or too small over time. This issue is called the exploding or vanishing gradient problem. The Long Short-Term Memory (LSTM) RNN architecture was developed to counter this type of problem with RNN gradients [13].

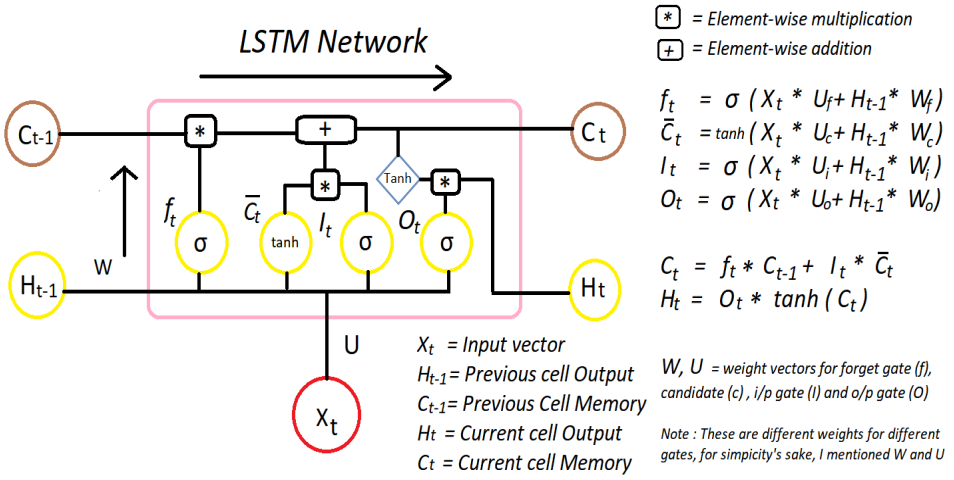


Figure 11. An LSTM cell with forget gate [15].

Figure 11 shows the architecture of an LSTM RNN with forget gate. Refer to the equations shown on this figure to better understand the function of the main components of this type of RNN. Its main components are:

1. Input: The LSTM unit takes the input X_t and output from the previous step H_{t-1} and their weighted sum is passed through the tanh activation which produces $\bar{C}_t$.
2. Input gate *I*: the input gate reads X_t and $H_{t-1,}$ computes their weighted sum and applies the sigmoid activation to calculate I_t, which is multiplied by $\bar{C}_t$ and passed into the memory cell.
3. Forget gate *f*: The forget gate serves to forget irrelevant old content. This gate also reads X_t and $H_{t-1,}$ computes their weighted sum and applies the sigmoid activation to calculate f_t, which is multiplied by S_{t-1}.
4. Memory gate: It is a central unit with unit weight recurrent connection that represents a time step of 1 feedback loop. It computes the current state S_t by forgetting irrelevant information from the previous step and keeping information from the current input.
5. Output gate *O*: It controls what information to flow out of the LSTM unit by applying the sigmoid activation to the weighted sum of X_t and Ht-1.
6. Output: It is denoted by H_t and is computed by passing the S_t cell through a tanh activation and multiplying it by O_t [15].

This project uses LSTM RNN, or LSTM for short, to solve the astrodynamics problem.

1.3 Project Proposal

This project takes advantage of the computational power of NNs to demonstrate the construction of a database which can be used as a roadmap for a spacecraft in close orbit around a planet. An NN is capable of correcting two-body astrodynamics problems for the gravitational perturbations caused by multiple celestial bodies on a spacecraft in space transit. In doing so, some of the NN output parameters could represent a mapping of these perturbations on space given that these parameters can be decoded after training. In an effort to carefully study these gravity perturbations, the proposed technique is to be developed for capture orbits with high apocytherion, at which solar gravity perturbations significantly deviate the spacecraft aerocentric orbits. Hence, the data to be analyzed will involve closed orbits at different pericytheria starting with highly eccentric orbits to end in circular orbits that approach Venus in favor of cost-free transfers. The large number of numerical integrations required by this type of analysis can be significantly reduced by the use of NN without compromising sufficient accuracy.

This project proposes an Artificial Recursive Neural Network (RNN) model for the orbital dynamics of a spacecraft in closed orbit around Venus. The RNN will be trained to accomplish path-correction in the presence of solar gravity used to its advantage for cost-free transfers. The method and position of transfer from an orbit with large apocytherion will be chosen depending on the position on the orbit where the solar gravity pull provides the minimum cost of transfer.

1.4 Methodology

The predicted development of this project proposes the following procedure:

1. Create a database using simulation methods for RNN implementation:
   The input data of the RNN consists of the parameters to be corrected when compared with its output or training data. In this problem, the input data are the position and velocity vectors of a spacecraft in Cartesian coordinates at every orbit time step with no solar gravity perturbations. The output, or training data, is composed of the position and velocity vectors of the spacecraft in Cartesian coordinates at every step on time with solar perturbation considered. The input and training data will be simulated by deep space orbit determination and optimization software GMAT (General Mission Analysis Tool) given an initial state vector (position and velocity vectors) and time of propagation. Initial position and velocity vectors corresponding to capture orbits with high apocytherion will be used to generate the first set of data at different pericytheria. Posterior data sets with less eccentric orbits with initial state vectors corresponding to the most convenient transfer orbits will be used to create a solar gravity perturbation mapping for the phase space covered by these orbits.

2. Application of the Back-Propagation approach of an RNN:
   RNNs use the Back-Propagation approach, which will be used to solve for the proposed problem. This approach trains an NN through supervised learning involving a forward pass that propagates input data forward from the input layer to the output layer of the network. This pass generates an output and its error value, which is corrected if within an acceptable range, while leaving synaptic weights (connection amplitude between nodes) intact, and the network is trained with the new set of input data. The back-propagation algorithm can be summarized as follows:
   1) The network is fed with an input vector and a corresponding desired output vector.
   2) The output of the network is calculated using the forward pass.
   3) The error in the output signal is calculated.
   4) The process moves on if the error is within the acceptable range or goes backward to update the network weights.
   5) The process repeats until all input vectors are consumed.
   Hence, the above-mentioned steps will be applied in the ideal RNN to analyze the perturbed closed orbits involved in this project.
3. RNN Implementation (LSTM):
   Long Short-Term Memory (LSTM) is a type of Recurrent Neural Network (RNN) capable of learning order dependence in sequence prediction models. This type of RNN is well suited for classifying, processing and making predictions based on time series, which very well serves the purposes of this project. Hence, LSTM will be used for this project via Keras, a deep learning Application Programming Interface (API) based in Python, running on top of the open-source machine learning platform TensorFlow. TensorFlow is based in differentiable programming, where the program's parameters can be optimized via gradient descent. Keras is a powerful, robust API well suited to serve the purpose of this project. Keras is a well-documented framework that makes its application accessible to the intermediate level programmer. However, it involves highly advanced, complex algorithms that might be time consuming to learn and understand for non-expert programmers.

4. Developing a Solar Perturbation Mapping of Phase Space:

Within each node of an RNN, there is a set of inputs, weight, and a bias value. As the input enters the node, it is multiplied by a weight value and the resulting output is either observed or passed to the next layer in the RNN. For instance, a single node might take the input data and multiply it by an assigned weight value, add a bias and pass the data to the next layer. The output layers might tune the inputs from the hidden layers to produce the desired values within a specific range. Weights and biases are learnable parameters within the RNN that are initiated randomly before the learning process begins. As training continues, both parameters are adjusted to match the desired output. In this problem, the weights represent the small changes that solar gravity perturbation makes on the input data. The weight will adjust until the unperturbed data values approach the perturbed data values. This analysis will not only allow for the mapping of the solar gravity perturbations but also to detect where these perturbations can aid desired orbit transfers. However, the resultant weight matrix of an NN has a complex relation with its outputs due to the gradient descent backward method, and it might require an autoencoder that will manipulate the data in a way that can more clearly relate to the input data. Disentangling the complexity of NN weight matrices is presently an area under research and great interest in the field of machine learning.

Chapter 2: Data Generation for RNN Training

In order to train the RNN model in this investigation, a vast data bank describing the position and velocity of a spacecraft in closed orbits around Venus is needed. This data bank involves the position and velocity vector components at every point in time for a given close orbit, which will be used not only to train the RNN but also to construct the mapping of the solar perturbation present at every step. Although one of the objectives of this investigation is to be able to detangle the weights of an RNN to represent the solar perturbations, this is a new field of study still under development in computer science. Hence, it might take more than the time allotted to the completion of this project to reach such goal. Therefore, the solar gravity perturbations are also being tracked by calculating the change in every one of the position components, x, y and z and velocity components v_x, v_y, and v_z between the solar gravity perturbed and unperturbed orbits of the spacecraft around Venus.

2.1 State Vectors for Close Orbit Determination

GMAT is being used to generate the position and velocity vectors of the spacecraft for every time point of every close orbit being mapped in this project with and without solar gravity potential perturbations. The initialization of the GMAT function simulating the motion of the spacecraft, the Propagator, requires initial state vectors for the position and velocity of the spacecraft or the initial orbital elements. The recently proposed Venera D mission aiming to do a comprehensive mission to Venus was used as a model to determine the initial conditions of the spacecraft's orbit for this project. The projected arrival date to Venus of this mission is December 5 of 2026 [16].

2.1.1 The Venera D Mission

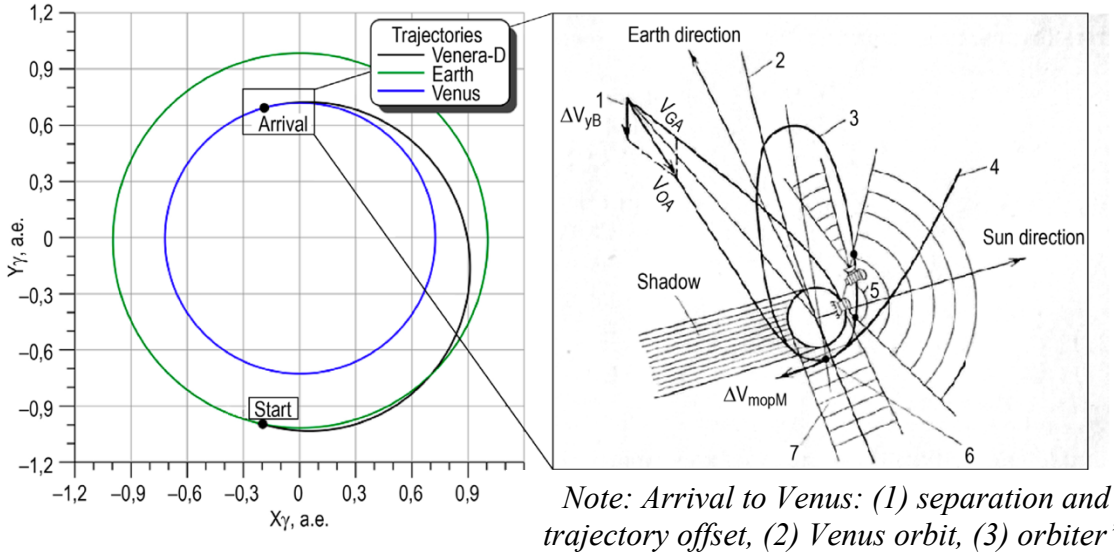
The Joint Science Definition Team (JSDT) formed by The Russian Space Agency, Space Research Institute of the Russian Academy of Sciences, and NASA, proposed the Venera-Dolgozhivuschaya (Venera-D) mission in 2014 with the goal of *"understanding Venus as a system, from the top of the atmosphere to the surface and interior"* [16, p1]. The Venera D mission will consist of a lander with an attached Long-Lived, In-Situ Solar System Explorer (LLISSE), which will sample the atmosphere and image the surface during descent before landing at high altitude on the northern hemisphere of Venus [17].

In spite of the fact that water is not, at this point present on Venus' surface, accessible geomorphology information shows a past filled with a surface molded from ongoing volcanism, blaming, and collapsing, which might be connected to the previous presence of water [18]. Life in Venus that might have evolved during its wet era, when its climate was still habitable, may remain extant today given its thick sulfuric acid clouds [19], [20]. This poses an opportunity to better understand when Venus may have been habitable in the past. Moreover, the study of the existence of water in Venus [21], [22] and its climate, including possible formation and loss or migration of microbial life forms from surface to the clouds, makes the planet an appealing destination for the study of life on other Earth-like planets [16].

According to the launch opportunities in 2026 and 2027, the trajectory analysis of this mission is as follows:

1) Launch from the Vostochny launch facility in Russia in 2026 with backup dates in 2027 and 2029 using the Angara-A5 carrier rocket;
2) Earth-Venus trajectory transition using hydrogen KVTK or Briz upper stage vehicle;
3) Flight along Earth-Venus trajectory with necessary corrections;
4) Descent module separation two days prior the arrival to Venus;
5) Transfer maneuver to place orbiter on nominal approaching orbit;
6) Descent module entry to the atmosphere;
7) Orbital module transfer to a high elliptical orbit using the rocket engine;
8) Separation of a subsatellite, if provided;
9) Nominal scientific operations assuming data transmission from the surface of Venus and the subsatellite to Earth though the orbiter [16].

For the scenario of a launch from Earth on May 30 of 2026, an initial Δv (delta-v or burn) of 3.905 km/s will take Venera D from a circular low orbit around Earth with an altitude of 200 km to a departure velocity vector at infinity (or v-infinity) of 3.905 km/s. A second Δv of 0.899 km/s will transfer it into a high elliptic orbit around Venus with pericenter (or periapsis) altitude of 500 km and orbital period of 24 hours. The right ascension of the asymptotic velocity arriving vector in J2000 coordinates is expected to be 186.73°. The required pericenter altitude can be guaranteed by and inclination of 90°±10°, and argument of pericenter $\pi/2 < \omega < \pi$ or $3\pi/2 < \omega < 2\pi$ [16]. Figure 13 below provides a visual scenario of the orbital mechanics of this mission with a launch date on June 2026.



Note: Earth-Venus transfer trajectory with start in June 2026.

Note: Arrival to Venus: (1) separation and trajectory offset, (2) Venus orbit, (3) orbiter's orbit, (4) flyby trajectory, (5) area of orbiter-lander communication, (6) braking and transferring the orbiter onto the orbit around Venus, and (7) radio shadow of Venus.

Figure 12. Venera D Orbital Mechanics [16].

2.1.2 The Astrodynamics

The initial conditions for propagation of both the unperturbed and perturbed orbits to be used in the initial stage of this project, include the orbital parameters projected in [16] mentioned above. For the initial point of the first orbit, the two-body astrodynamics equations of motion can be propagated via the Runga Kutta numerical methods in GMAT. This propagation method is necessary to solve the three-body problem in the presence of solar gravity perturbations. GMAT's propagator requires initial conditions given by either the initial state vectors (AKA position and velocity) or the six Keplerian elements, which describe an orbit. The six elements are:

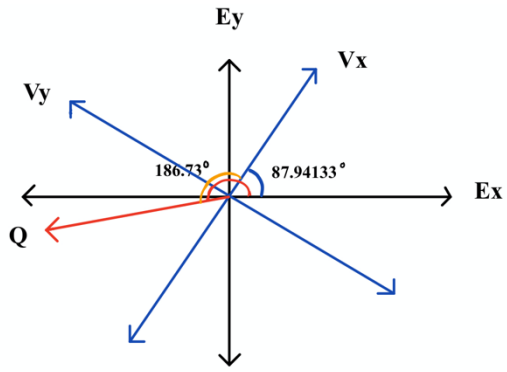
- a = Semi-major axis (size)
- e = Eccentricity (shape)
- i = inclination (tilt)
- ω = argument of perigee (twist)
- Ω = longitude of the ascending node (pin)
- ν = true anomaly (angle at a given time) [23]

Having the pericenter altitude and period, τ, of the initial high elliptical orbit after the point of transfer, its semi-major axis and eccentricity can be found via the equation for the period of a closed orbit, $\tau^2 = \frac{4\pi^2 a^3}{\mathrm{GM}}$, where G is the gravitational constant, $\mathrm{G} = 6.67\mathrm{x}10^{-20}\,\frac{\mathrm{km}^3}{\mathrm{kg\,sec}^2}$, and M the mass of Venus, $\mathrm{M} = 4.8675\ \mathrm{x}\ 10^{24}$ kg [24]. Solving for a in the previous equation, we have

$$a = \sqrt[3]{\frac{\tau^2 \mathrm{GM}}{4\pi^2}} = \sqrt[3]{\frac{(86400\mathrm{s})^2 (6.67\mathrm{x}10^{-20}\frac{\mathrm{km}^3}{\mathrm{kg\,sec}^2})(4.8676\ \mathrm{x}\ 10^{24}\ \mathrm{kg})}{4\pi^2}}.$$

This yields a semi major axis $a = 39{,}448.744$ km. The given pericenter altitude can be used to find the radius at pericenter by adding the radius of Venus [24] to it: r_p = 500 km + 6051.8 km = 6551.8 km. Relating r_p to the eccentricity of the elliptical orbit yields the equation $r_p = a(1 - e)$. Solving for e in this equation, we have $e = 1 - \frac{r_p}{a}$. This results in an eccentricity of $e = 0.8339$.

Figure 14 below shows the XY plane on the reference frames of Earth and Venus and the position of spacecraft Q with respect to both frames. The right ascension of the asymptotic velocity arriving vector in J2000 coordinates of 186.73° can be converted to Venus' reference frame, which represents the right ascension of the orbit at radius of pericenter. As the sketch shows, this can be done by subtracting the right ascension of Venus with respect to Earth, Ω_{VE} from that of the asymptotic velocity arriving vector.



Longitude of ascending node (J2000) Earth = -11.26064°
Longitude of ascending node (J2000) Venus = 76.68069° [24]
E = Earth reference frame (J2000)
V = Venus reference frame (J2000)
Right ascension of Venus with respect to Earth:
Ω_{VE} = 76.68069° + 11.26064° = 87.94133° (shown in blue)
Right ascension of spacecraft's orbit with respect to Venus:
Ω_{QV} = 186.73° - 87.94133° = 98.7887° (shown in orange)

Figure 13. Conversion of the right ascension of the asymptotic velocity arriving vector in J2000 coordinates to that of the spacecraft's orbit with respect to Venus.

The orbital elements calculated so far were tested by propagating the spacecraft for a full period in GMAT. The calculated semimajor axis of 39,448.744 km proved to yield a radius of pericenter too small to serve the purposes of this investigation, and it was increased by 10,000 km. This increment to the semimajor axis yields a period of:

$$\tau = \sqrt{\frac{4\pi^2 a^3}{GM}} = \sqrt{\frac{4\pi^2(49{,}448.744\text{km})^3}{(6.67\text{x}10^{-20}\frac{\text{km}^3}{\text{kg sec}^2})(4.8676\text{ x }10^{24}\text{ kg})}} = \approx 33.6814\text{ hr.} \approx 33\text{ hr.}, 40\text{ min. and }53\text{ sec.}$$

The test also proved 100° to be the most convenient value, within the allowable range, for the argument of perigee.

Collecting the known orbital elements to initiate the data generation for this project we have:

- a = Semi-major axis = 49,448.744 km
- e = Eccentricity = 0.8339
- i = inclination = 90°
- ω = argument of perigee = 100.0° (chosen from allowable range)
- Ω = longitude of the ascending node = 98.7887°
- ν = mean anomaly = 0° (at radius of pericenter).

The values above represent the initial condition of the first orbit of perturbation mapping of this investigation. Time permitting, several other orbits will be investigated and will be chosen based on the minimum cost of transfer provided. However, the initial conditions for the next orbits to be observed will be given by the GMAT propagator itself. An analysis of the GMAT numerical method for data generation will be presented in the next section, as well as a brief derivation of the astrodynamical differential equations leading to the need for numerical methods.

2.2 GMAT

The General Mission Analysis Tool, GMAT, is an open-source software developed by NASA in partnership with industry, private and public contributors. GMAT has a wide range of capabilities beyond supporting with the design and analysis of space missions. Its dynamics and environment modelling support the modeling of orbits, their analysis and detailed visualization, orbit perturbation study and maneuver planning, the determination of propulsion system requirements and estimation of the lifetime of a mission. It also provides a detailed visual representation of the solar system allowing the use of a rich set of coordinate systems, orbits and natural phenomena such as axial tilts and the phases of the moon, formations and constellations, harmonic gravity, drag, tides, and relativistic corrections. GMAT's propagation feature uses ephemeris files from celestial navigation data banks CCSDS, SPICE, STK, and Code 500. Propagators in GMAT naturally synchronize epochs of multiple vehicles and about fixed step integration and interpolation [10].

GMAT is implemented in ANSI standard C++ using Object Oriented methodology. It interfaces with external platforms, such as, MATLAB and Python, giving an extensible engineering for future development. It has a rich featured, interactive GUI that makes analysis quick and simple, custom scripting language that makes complex, custom analysis possible, and a command line interface for batch analysis. GMAT has enabled and upgraded missions in practically every NASA flight system including empowering new mission types, broadening the life of existing missions, and advancing new science perceptions. It has supported eight NASA missions and more than ten NASA proposal endeavors. Up to date, GMAT has benefit over 30 organization including 15 universities and 12 commercial firms with their publication of results in the open literature. GMAT is under the Apache License 2.0, and supports Windows 7+, Mac OSX 10.10+ and Linux platforms [10].

2.2.1 Propagation Method

Simulating the orbital motion of (or propagating) a spacecraft is the most fundamental capability of GMAT, which is done via the Propagator function. A GMAT Propagator is either of numerical integration type or ephemeris type. The ephemeris type uses data banks for celestial navigation that gives the trajectory of celestial bodies and satellites. The numerical integration type uses the different propagation methods mentioned on Table 2 on page 6 of this report. The data generated for the purposes of this investigation uses the Runge-Kutta 89 numerical integrator with an error control on the eighth order, and for which its coefficients were derived by J. Verner. These coefficients are chosen due to their robustness though they are not necessarily being the most efficient [10].

The analysis of spacecraft motion leads to ordinary differential equations with time as the independent variable. Many times, numerical methods make it possible or less complicated to solve for these ordinary differential equations. In this section, a brief analysis of the origin of the astrodynamical ordinary differential equations describing the motion of a spacecraft will be presented, as well as the Runge-Kutta numerical method to solve for them. Bold letters in the rest of this paper will denote vectors [25].

Newton's second law describes the particle mechanics in the Newtonian frame via the second-order differential equation

$$\ddot{\mathbf{r}} = \frac{\mathbf{F}}{m}, \tag{2.1}$$

where **r** is the position vector as a function of time, *m* is the mass of the spacecraft and **F**, the forces acting on it. Whether there might or not a closed, analytical solution to this equation, depends on how complex the force function F might be. In the most trivial case, the problem can be solved using ordinary differential equation integration methods, which yield:

$$\mathbf{r} = \frac{\mathbf{F}}{2m}t^2 + \mathrm{C}_1 t + \mathrm{C}_2, \text{ with } \mathbf{F} \text{ and } m \text{ being constant} \tag{2.2}$$

C_1 and C_2 in this equation are the vector constants of integration making six scalar constants of integration in total. Given that the position and velocity at time $t = 0$ are r_0 and $\dot{\mathrm{r}}_0$ yields an initial value problem. Applying the initial conditions to equation 2, we have that $\mathrm{C}_1 = \mathrm{r}_0$ and $\mathrm{C}_2 = \dot{\mathrm{r}}_0$, and the equation becomes

$$\mathbf{r} = \frac{\mathbf{F}}{2m}t^2 + \dot{\mathbf{r}}_0 t + \mathbf{r}_0, \text{ with F and } m \text{ being constant} \tag{2.3}$$

Similarly, if we know the position, $\mathbf{r}_0$ at $t = 0$, and velocity, $\dot{\mathbf{r}}_f$, at a later time $t = t_f$, we can apply the boundary conditions for a boundary value problem where $\mathrm{C}_1 = \dot{\mathbf{r}}_f - \frac{\mathbf{F}}{2m}t_f$ and $\mathrm{C}_2 = \mathbf{r}_0$ yielding

$$\mathbf{r} = \frac{\mathbf{F}}{2m}t^2 + (\dot{\mathbf{r}}_f - \frac{\mathbf{F}}{2m}t_f)\mathrm{t} + \mathbf{r}_0, \text{ with } \mathbf{F} \text{ and } m \text{ being constant} \tag{2.4}$$

The goal here is to solve the initial value problem [25].

Equation one contains three components:

$$\ddot{x} = \frac{\mathrm{F}_x(t,\mathbf{r},\dot{\mathbf{r}})}{m} \qquad \ddot{y} = \frac{\mathrm{F}_y(t,\mathbf{r},\dot{\mathbf{r}})}{m} \qquad \ddot{z} = \frac{\mathrm{F}_z(t,\mathbf{r},\dot{\mathbf{r}})}{m} \tag{2.5}$$

These are uncoupled second-order differential equations and will be reduced to six first-order for the purpose of numerical solution. Introducing the auxiliary variables y_1 through y_6 for reduction, we have:

$$\begin{aligned} y_1 = x \quad y_2 = y \quad y_3 = z \\ y_4 = \dot{x} \quad y_5 = \dot{y} \quad y_6 = \dot{z} \end{aligned} \tag{2.6}$$

The position and velocity in terms of these auxiliary variables in a Newtonian frame become:

$$\mathbf{r} = y_1\hat{\boldsymbol{\imath}} + y_2\hat{\boldsymbol{\jmath}} + y_3\hat{\boldsymbol{k}} \; \dot{\mathbf{r}} = y_4\hat{\boldsymbol{\imath}} + y_5\hat{\boldsymbol{\jmath}} + y_6\hat{\boldsymbol{k}}$$

Their derivatives with respect to time yield:

$$\frac{\mathrm{d}y_1}{\mathrm{d}t} = \dot{x} \quad \frac{\mathrm{d}y_2}{\mathrm{d}t} = \dot{y} \quad \frac{\mathrm{d}y_3}{\mathrm{d}t} = \dot{z}$$

$$\frac{\mathrm{d}y_4}{\mathrm{d}t} = \ddot{x} \quad \frac{\mathrm{d}y_5}{\mathrm{d}t} = \ddot{y} \quad \frac{\mathrm{d}y_6}{\mathrm{d}t} = \ddot{z}$$

Combining equations 5 and 6, we get:

$$\begin{aligned} \dot{y}_1 &= y_4 \\ \dot{y}_2 &= y_5 \end{aligned} \tag{2.7}$$

$$\dot{y}_3 = y_6$$
$$\dot{y}_4 = \mathbf{F}_x\,(t, y_1, y_2,\ y_3,\ y_4, y_5,\ y_6)$$
$$\dot{y}_5 = \mathbf{F}_y(t, y_1, y_2,\ y_3,\ y_4, y_5,\ y_6)$$
$$\dot{y}_6 = \boldsymbol{F}_z(t, y_1, y_2,\ y_3,\ y_4, y_5,\ y_6)$$

Since these equations are coupled, they can be written more compactly in the following way

$$\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y}) \tag{2.8}$$

Where

$$\mathbf{y} = \begin{Bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \end{Bmatrix} \quad \dot{\mathbf{y}} = \begin{Bmatrix} \dot{y}_1 \\ \dot{y}_2 \\ \dot{y}_3 \\ \dot{y}_4 \\ \dot{y}_5 \\ \dot{y}_6 \end{Bmatrix} 0 \quad \mathbf{f} = \begin{Bmatrix} y_4 \\ y_5 \\ y_6 \\ \frac{F_x\,(t,y)}{m} \\ \frac{F_y\,(t,y)}{m} \\ \frac{F_z\,(t,y)}{m} \end{Bmatrix} \tag{8}$$

In order to obtain a numerical solution for equation $\dot{\mathbf{y}} = \mathbf{f}(t, \mathrm{y})$ over the time interval $t_0 \le t \le t_f$, we divide or mesh the interval into *N* discrete times $t_1, t_2, t_3 \ldots t_N$, with $t_1 = t_1$ and $t_N = t_f$. The step size *h* represents the time difference between adjacent times on the mesh [25].

Let us define equation 7 as $\dot{\mathbf{y}}_i = \mathbf{f}(\,t_i,\ \mathbf{y}_i\,)$ for time $t = t_i$. The Runge-Kutta or RK methods were developed by German mathematicians Carle Runge (1856-1927) and Martin Kutta (1867-1944). For the explicit, single-step RK, $\mathbf{y}_{i+1}$ at $t_i + h$ is obtained from $\mathbf{y}_i$ at t_i by the formula

$$\mathbf{y}_{i+1} = \mathbf{y}_i + h\varphi(\,t_i,\ \mathbf{y}_i, h\,) \tag{2.9}$$

where φ is a function average of the derivative d**y**/d*t* over the interval t_i to $t_i + h$ obtained by evaluating the derivative $\mathbf{f}(t, \mathrm{y})$ at different stages within the time interval. The order of an RK method reflects the accuracy to which φ is computed, compared to a Taylor series expansion. An RK method of the *p* order is called RK*p* method,
and it is as accurate in computing $\mathbf{y}_i$ from equation 9 as a Taylor series of the *p*th order is:

$$\mathbf{y}(t_i + h) = \mathbf{y}_i + \mathrm{c}_1 h + \mathrm{c}_2 h^2 \ldots + \mathrm{c}_p h^p \tag{2.10}$$

For the RK method only the first derivative $\mathbf{f}(t, \mathrm{y})$ is required, which is available from equation 2.7 itself. The higher the RK order is, the more stages it has and the more accurate φ is. For a number of stages *s*, there are *s* times $\tilde{t}$ within time interval $t_i \le t \le t_i + h$ where the derivatives $\mathbf{f}(t, \mathrm{y})$ are evaluated. These times are given by specifying numerical values of the nodes a_m in the expression

$$\tilde{t}_{\,m} = t_i + a_m h, \text{ where } m = 1, 2, 3, \ldots, s$$

At each of these times the value of $\tilde{y}$ is obtained by providing numerical values for the coupling coefficients *bmn* in the formula

$$\tilde{\mathbf{y}}_m = \mathbf{y}_i + h \sum_{n=1}^{m-1} bmn\, \tilde{\mathbf{f}}_n, \text{ where } m = 1, 2, 3, \ldots, s \tag{2.11}$$

The vector of derivatives $\tilde{\mathbf{f}}_m$ is evaluated at stage m by substituting and $\tilde{t}_m$ and $\tilde{\mathbf{y}}_m$ into equation 7:

$$\tilde{\mathbf{f}}_m = \mathbf{f}(\tilde{t}_m, \tilde{\mathbf{y}}_m), \text{ where } m = 1, 2, 3, \ldots, s \qquad (2.12)$$

The increment function φ is a weighted sum of the derivatives $\tilde{\mathbf{f}}_m$ over the s stages within the time interval t_i to $t_i + h$,

$$\varphi = \sum_{m=1}^{s} cm\, \tilde{\mathbf{f}}_{\boldsymbol{m}} \qquad (2.13)$$

The coefficients cm are known as the weights. Substituting equation 13 into equation 9 yields

$$\mathbf{y}_{i+1} = \mathbf{y}_i + h \sum_{m=1}^{s} cm\, \tilde{\mathbf{f}}_{\boldsymbol{m}} \quad (14)$$

The numerical values of the coefficients a_m, b_m, and c_m depend on which RK method is being used. It is convenient to write these coefficients as arrays, so that

$$\{\mathbf{a}\} = \begin{Bmatrix} a_1 \\ a_2 \\ \vdots \\ a_2 \end{Bmatrix} \quad [\mathbf{b}] = \begin{bmatrix} b_{11} & & & \\ b_{21} & b_{22} & & \\ \vdots & \vdots & \cdots & \\ b_{s1} & b_{s2} & \cdots & b_{s,s-1} \end{bmatrix} \quad \{\mathbf{c}\} = \begin{Bmatrix} c_1 \\ c_2 \\ \vdots \\ c_2 \end{Bmatrix}, \text{ where } s \text{ is the number of stages (15).}$$

[**b**] is undefined when $s = 1$ and nodes {a}, coupling coefficients [b], and weights {c} for a given RK method are not necessarily unique [25].

2.2.2 Set Up

The set up in GMAT is initiated by configuring the spacecraft, which includes the initial conditions to start propagation. This configuration requires not only the six Keplerian elements above calculated above, but also the epoch (data of arrival to Venus in this case) and coordinate system. The epoch is in Coordinate Universal Time (UTC) time scale based on the current Gregorian year and is located at 0-degree latitude, the Prime Meridian. This is the usual calendar format plus Hour:Minute:Second.fraction [26].
The coordinate system used to propagate is based on the Venus mean orbital elements J2000 defined based on the Earth's equator and equinox on January 1, 2000 at 12:00:00 TBD. Figure 15 shows this configuration with the six Keplerian elements calculated above and the expected epoch for the arrival of Venera D to Venus with launch date on May 30, 2026.

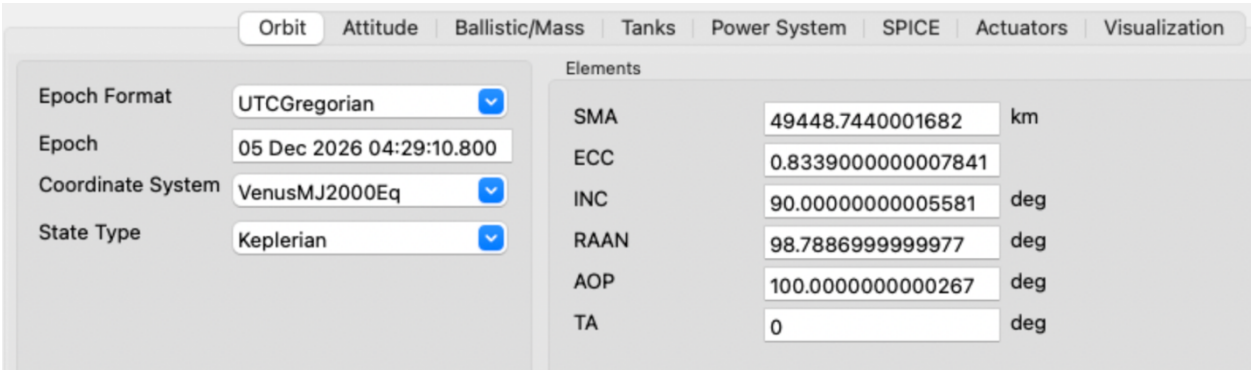


Figure 14. Configuration of the spacecraft in GMAT.

The next step of the set up includes the spacecraft propagator which defines the method of propagation which requires a force model via the FM field on the propagator object. The propagator includes the following components pertinent to this setup:

1) Type – numerical integrator Runge-Kutta 89 (method defined on Table 2 and explained in detail in the previous section).
2) Initial step size – size of the first step attempted by the integrator.
3) Accuracy – desired accuracy for an integration step. GMAT uses the method selected in the ErrorControl field on the Force Model to determine a metric of the integration accuracy.
4) FM – identifies the force model used by an integrator.
5) Min step size – minimum allowable step size.
6) Max step size - maximum allowable step size.
7) Max step attempts - The number of attempts the integrator takes to meet the tolerance defined by the Accuracy field.
8) Stop if accuracy is violated - Flag to stop propagation if integration error value defined by Accuracy is not satisfied [10].

The following figure shows the numerical integrator set up with the force model required.

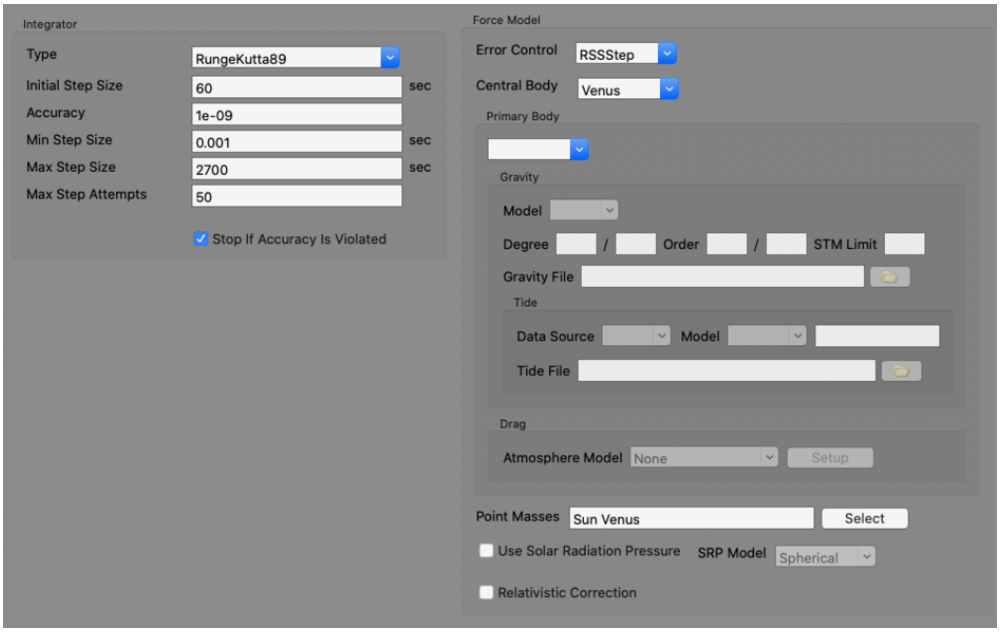


Figure 15. GMAT propagator set up with its numerical integrator (left) and force model (right).

As discussed in the previous section, the analysis of a spacecraft motion leads to ordinary differential equations and a force model is required by the numerical integrator. The setup for the model analyzed in this project has the following components:

1) Error control – controls how error in the current integration step is estimated. The error in the current step is computed by the selection of ErrorControl and compared to the value set in the Accuracy field to determine whether the step has an acceptable error or needs to

be improved. All error measurements are relative error. The RSSStep is the Root Sum Square (RSS) relative error measured with respect to the current step.

2) Central body – central body of propagation, which must be a celestial body with mass and cannot be a libration point, barycenter, spacecraft, or other special point. In this setup the central body is Venus.
3) Point masses – a list of celestial bodies to be treated as point masses in the force model. A body cannot be both, the primary body in the point masses list. For the needs of this project only the Sun and Venus are analyzed [10].

The rest of the components of the GMAT propagator feature are not of relevance for this investigation at the moment.

The orbit view plot gives a graphical view of the propagation of the satellite following the previous set up and obeys the Propagation and Spacecraft configuration under the Mission Sequence tab (or function in the code) of the GMAT GUI. This configuration includes the previously programmed propagator setup with its corresponding spacecraft (named SatVenus in this case) and the time or upper bound parameter to propagate. In this configuration, the spacecraft starts at the pericenter and ends at the periapsis of Venus as it can be seen on the figure below [10].

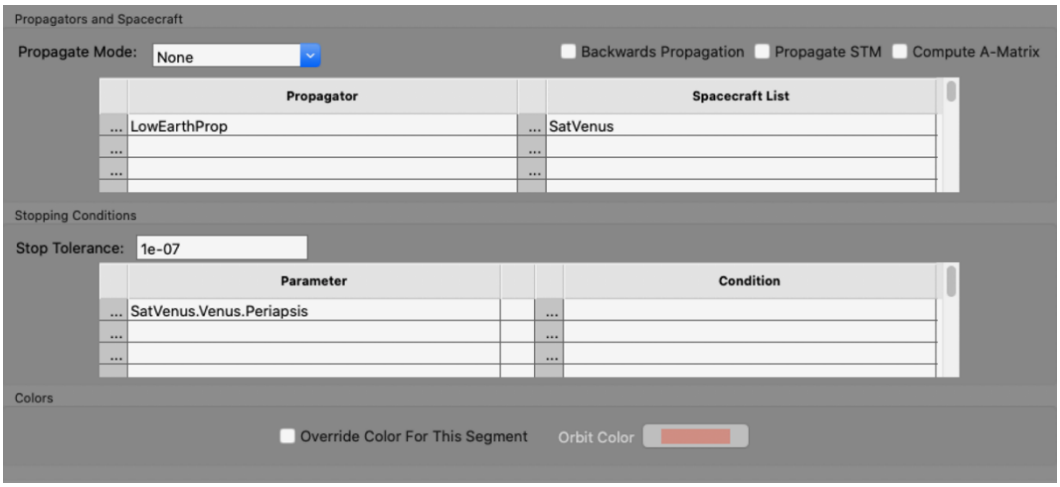


Figure 16. Mission sequence set up in GMAT.

The orbit view plot allows to configure the visuals of the orbit done by the propagator and includes the object to be plotted such as starts, constellations, labels and axes, mesh grids for the celestial body, and x-y or ecliptic planes. It also allows plotting of the Sun line to see the location of the Sun with respect to the system under observation, various spacecraft and celestial bodies with respect to the chosen coordinate system, point of reference and a defined vector use as the point of view from which the system can be observed. This setup also has the MVenusJ2000 coordinate system with Venus as the point of reference and view direction. Figures 18 and 19 show the described setup and its resultant orbit view plot respectively. The direction of the Sun with respect to Venus is shown as a yellow line on the plot. The data generated by GMAT corresponding to the setup presented in this section is presented in Table 4, and it corresponds to the orbit view plot presented in Figure 19.

The three components of the position and velocity vectors with respect to the number of seconds lapsed per propagation step are presented and will be analyzed in detail in the following chapters of this paper.

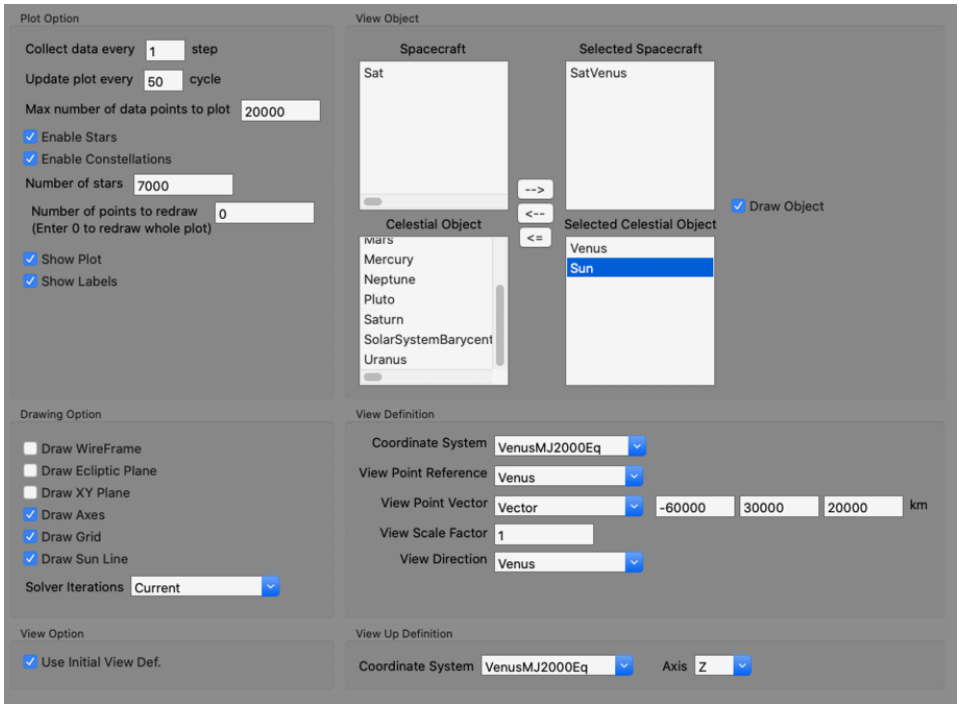


Figure 17. Orbit view plot setup in GMAT.

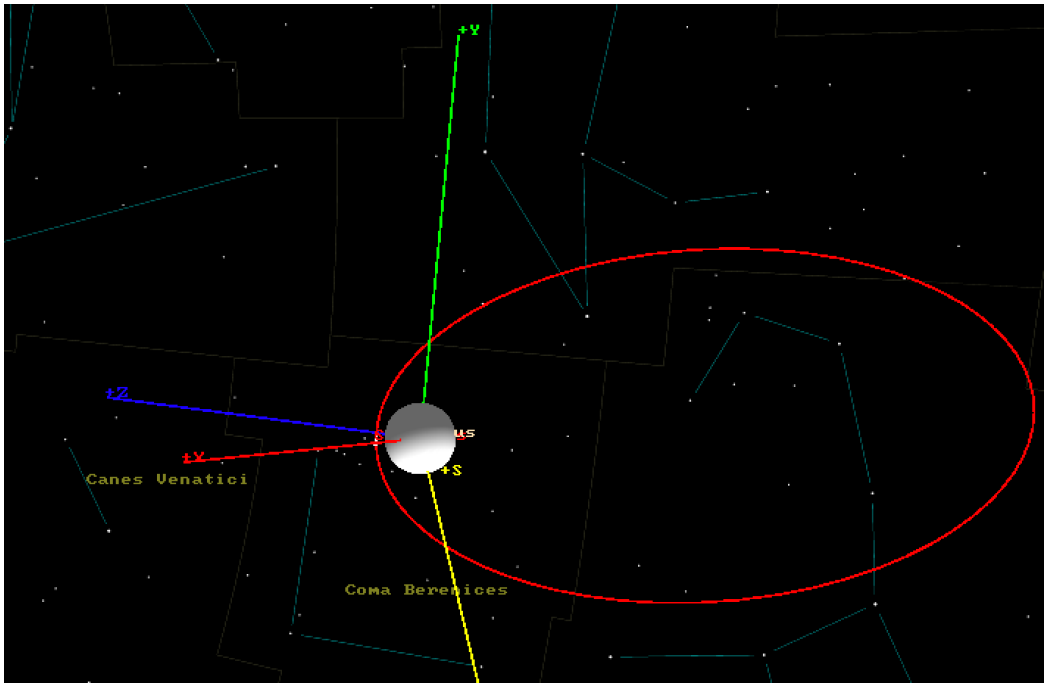


Figure 18. Orbit view plot in GMAT corresponding with the setup for the data generation of this project.

Table 4. Data file generated by GMAT corresponding with the setup presented in this chapter.

SatVenus.ElapsedSecs	SatVenus.VenusMJ2000Eq.X	SatVenus.VenusMJ2000Eq.Y	SatVenus.VenusMJ2000Eq.Z	SatVenus.VenusMJ2000Eq.VX	SatVenus.VenusMJ2000Eq.VY	SatVenus.VenusMJ2000Eq.VZ
0	217.4771076	-1406.651584	8072.298168	1.282806588	-8.297249948	-1.480411229
59.99999988	294.1875191	-1902.817931	7974.939298	1.273759101	-8.238730399	-1.764185212
255.2485593	538.7149398	-3484.432141	7543.399966	1.226778068	-7.93485499	-2.641610591
470.0055897	794.3225173	-5137.713297	6883.689673	1.150114462	-7.438991389	-3.474812439
697.7596589	1045.306412	-6761.088267	6009.707272	1.052073446	-6.804857747	-4.16576382
970.4442447	1315.344388	-8507.70588	4790.153302	0.928668973	-6.006672134	-4.735403827
1222.554714	1535.587739	-9932.249652	3552.29949	0.820013264	-5.303882188	-5.056188804
1485.03541	1737.178828	-11236.14977	2198.544636	0.718160256	-4.645092412	-5.237369624
1775.846567	1931.436273	-12492.61554	661.6026036	0.620434316	-4.012996695	-5.315564955
2109.217771	2122.135813	-13726.06863	-1111.250687	0.526662438	-3.406476026	-5.307111306
2499.306046	2309.572743	-14938.41901	-3166.947891	0.437659984	-2.830804196	-5.223132485
2956.162798	2489.952184	-16105.12124	-5520.83985	0.355406159	-2.298782804	-5.076121353
3487.655062	2658.329346	-17194.19219	-8167.482872	0.281479318	-1.8206207	-4.8813127
4110.696459	2812.221245	-18189.5718	-11136.50547	0.215614231	-1.394602376	-4.650540025
4855.102301	2949.725121	-19078.95294	-14500.17139	0.1567455	-1.01383678	-4.390236793
5762.287666	3066.749778	-19835.87352	-18350.52353	0.104041475	-0.672944749	-4.104137583
6817.246579	3151.766937	-20385.7681	-22522.95782	0.059513247	-0.384934102	-3.812955572
8046.720729	3200.575641	-20701.46461	-27026.73486	0.021910198	-0.141715772	-3.52101316
9485.111283	3207.995732	-20749.45743	-31876.49683	-0.009859609	0.063772961	-3.230296393
11172.92236	3167.492829	-20487.48162	-37078.29345	-0.036646744	0.237033663	-2.941823142
13157.81242	3071.177916	-19864.51024	-42625.62932	-0.059120583	0.382395934	-2.655936111
15492.71096	2910.056187	-18822.36373	-48486.96224	-0.077789679	0.503149186	-2.372797461
18192.71096	2678.444913	-17324.28476	-54509.98604	-0.092855157	0.600594382	-2.096373156
20892.71096	2412.626292	-15604.94815	-59842.50986	-0.103451704	0.669134526	-1.859177614
23592.71096	2122.469814	-13728.18967	-64575.35389	-0.11106909	0.718405516	-1.650769599
26292.71096	1814.730346	-11737.69962	-68776.23754	-0.116590343	0.754118801	-1.464176005
28992.71096	1494.263179	-9664.881795	-72497.11573	-0.120570813	0.779866383	-1.294527494
31692.71096	1164.702318	-7533.240696	-75778.73436	-0.123376006	0.798012361	-1.138295035
34392.71096	828.8640804	-5360.992051	-78653.58948	-0.125255065	0.810168177	-0.992838703
37092.71096	489.0000677	-3162.699036	-81147.92204	-0.126382381	0.817461779	-0.856128335
39792.71096	146.9628178	-950.3437303	-83283.10432	-0.126882347	0.820697724	-0.72656351
42492.71096	-195.6814719	1265.943782	-85076.62754	-0.126844681	0.820456287	-0.602853424
45192.71096	-537.5734905	3477.371494	-86542.81957	-0.126334218	0.817156781	-0.483934173
47892.71096	-877.5028703	5676.11069	-87693.37402	-0.125397311	0.811099012	-0.368910119
50592.71096	-1214.364435	7855.012958	-88537.74401	-0.1240661	0.802490814	-0.257011153
53292.71096	-1547.123899	10007.38833	-89083.43581	-0.12236134	0.791466406	-0.147560634
55992.71096	-1874.789833	12126.82393	-89336.22641	-0.120294263	0.778098407	-0.039950585
58692.71096	-2196.389638	14207.02859	-89300.32116	-0.117867727	0.762405259	0.066378181
61392.71096	-2510.947844	16241.69248	-88978.46256	-0.115076809	0.744355082	0.171952657
64092.71096	-2817.465385	18224.35303	-88371.99714	-0.111908938	0.723866533	0.277282192
66792.71096	-3114.898644	20148.25942	-87480.90477	-0.108343597	0.700806863	0.382871979
69492.71096	-3402.137089	22006.2279	-86303.79202	-0.104351572	0.674987068	0.489236342
72192.71096	-3677.978211	23790.47966	-84837.84891	-0.099893687	0.646153707	0.596912668
74892.71096	-3941.09822	25492.45134	-83078.76618	-0.094918899	0.613976583	0.706476914
77592.71096	-4190.016506	27102.56509	-81020.60736	-0.089361544	0.578030923	0.818561791
80292.71096	-4423.051172	28609.94098	-78655.62651	-0.083137408	0.537771952	0.933879101
82992.71096	-4638.261799	30002.02675	-75974.01788	-0.076138088	0.492498458	1.053248235
85692.71096	-4833.373813	31264.10863	-72963.57709	-0.06822283	0.441299957	1.177633781
88392.71096	-5005.675916	32378.64785	-69609.24381	-0.059206446	0.382978539	1.308196634
91092.71096	-5151.877137	33324.35599	-65892.48086	-0.048840985	0.315930333	1.446365435
93792.71096	-5267.901648	34074.86781	-61790.42138	-0.036787056	0.237959995	1.593939212
96492.71096	-5348.58424	34596.77143	-57274.67781	-0.022567225	0.145979269	1.753239035
99192.71096	-5387.200464	34846.56915	-52309.64342	-0.005486763	0.035494353	1.927338664
101892.711	-5374.707222	34765.76527	-46850.01008	0.015508828	-0.100315673	2.120425473
104592.711	-5298.443343	34272.46096	-40837.04277	0.042108228	-0.272373862	2.338378882
107292.711	-5139.739856	33245.89658	-34192.85205	0.077211366	-0.499438063	2.589700286
109426.3842	-4937.198551	31935.76692	-28427.38966	0.114406492	-0.740033669	2.819954362
111235.5371	-4694.054993	30363.0029	-23127.66085	0.156395691	-1.011638902	3.043750211
112773.7793	-4418.600055	28581.23486	-18283.11301	0.204048743	-1.319879755	3.25921351
114091.3616	-4115.496214	26620.62275	-13856.01663	0.258722544	-1.67353308	3.463838671
115235.0625	-3784.861554	24481.93115	-9786.5971	0.322679418	-2.087232398	3.653636912
116231.0859	-3427.732197	22171.8628	-6064.835817	0.398337143	-2.57661669	3.817813596
117072.1542	-3058.008306	19780.33049	-2802.210303	0.485255412	-3.138837679	3.934060911
117729.2917	-2710.709488	17533.85582	-198.8151206	0.575825	-3.724675899	3.978696753
118266.5653	-2376.675367	15373.18519	1933.536062	0.671445098	-4.343181847	3.94346975
118729.3025	-2042.90124	13214.19859	3732.068875	0.774838089	-5.011964763	3.807752595
119137.7286	-1704.433833	11024.85627	5238.458082	0.885846508	-5.73000588	3.537914858
119471.2156	-1391.884434	9003.164012	6357.808952	0.990493472	-6.406897662	3.141958294
119751.6088	-1100.969659	7121.414761	7170.925176	1.085068884	-7.018641919	2.624652442
120002.5387	-818.1083358	5291.759993	7751.721976	1.168505095	-7.558332618	1.971234285
120251.8131	-517.5559927	3347.674774	8140.729677	1.239908072	-8.020186292	1.114570679
120487.4337	-219.6328982	1420.59841	8289.543381	1.283995771	-8.305353661	0.122340391
120722.6246	84.17494663	-544.5406675	8189.486504	1.292843745	-8.362576679	-0.98463744
120825.488	216.7603057	-1402.150079	8062.769979	1.283693678	-8.303386722	-1.478505191

Chapter 3: RNN Back-Propagation Implementation

One of the most dreamed-of inventions since ancient Greek times has been a machine that can think. Some of the mythical inventor figures of those times are Hephaestus, Pygmalion and Daedalus with Pandora, Galatea and Talos as a representation of artificial life [27], [28], [29]. It took more than a hundred years for a programmable computer to be built after it was first conceived [30]. Nowadays, artificial intelligence (AI) is revolutionizing science and thriving with numerous applications and new topics of research.

In the beginning stages of AI, the field immediately proved to be readily able to solve problems that are extremely difficult for the human intellect; problems involving formal, mathematical rules. On the other hand, a true challenge for AI proved to be performing tasks that are easy for people to do, but difficult to describe such as problems solved by intuition like language, face and path recognition and the planning and scheduling of craft operations [31], [32]. A solution to this problem is to allow computers to learn from experience by presenting the world in the form of hierarchical concepts defined in their relation to simpler concepts. This learning process avoids the need for a human operator having to specify the knowledge the computer needs to learn and allows it to learn complicated concepts by building them from simpler ones. A visual representation of these concepts would be a deep, multi-layer process for which the name of AI "deep learning" has been assigned [31].

3.1 Deep Learning and Deep Neural Networks

Deep learning is a type of machine learning that involves representation-learning methods based on multiple levels of representation starting with simple but non-linear modules each of which transform the representation of one level, or raw data, to a higher, more complex level of representation. With enough such transformations, a classification task can amplify important aspects of the input and suppress irrelevant variations. For instance, the input image is an array of pixel values. The learned feature in the first layer of representation usually detects the absence or presence of edges at certain orientations and locations in the image. The second layer typically spots specific arrangement of the edges to detect patterns despite small differences between them. The third layer commonly assembles patterns into large combinations corresponding to parts of familiar objects, and posterior layers would recognize objects as combinations of these parts. The relevance of deep learning is that these layers are not human engineered but learned from data using a procedure with a general purpose [33]. The image shows visual representation of identification of an image via deep learning.

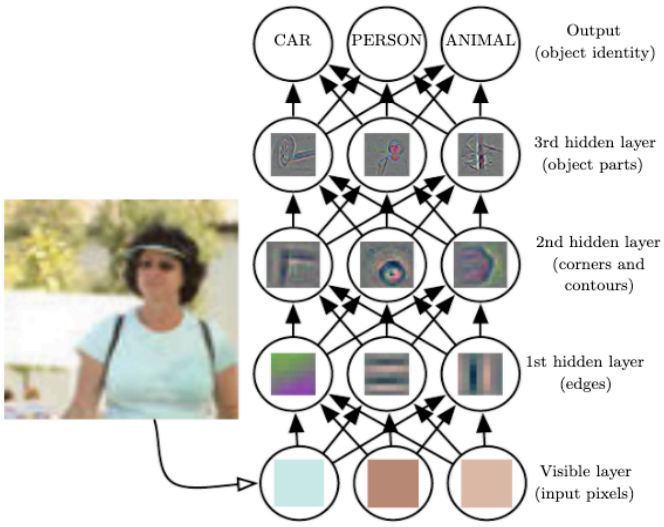


Figure 19. Illustration of a deep learning model [31].

A perfect example of a deep learning model is the feedforward deep network, or multilayer perceptron, which is just a mathematical function that maps a set of input values to output values. This function is composed of many simpler functions with each of their applications representing a new depiction of the input [31]. Feedforward deep networks are also called feedforward neural networks. The name of neural network was inspired by the structure and functioning of a biological brain, as it comprises computational units called nodes or neurons (perceptron layers). Deep neural networks are NNs with multiple hidden layers of neurons stacked together, each with a non-linear module and each of which receives the output of its previous layer. Each of the neurons in a deep NN takes an input at its incoming edge, multiplies it by a randomly assigned weight and applies a nonlinear function called the activation function to the weighted sum to produce an output. Recall Figures 7 and 8 with a visual representation of a NN and its typical activation functions where $\boldsymbol{x}$, $\boldsymbol{w}$, b, $\odot$, f and y represent input vector, weight vector, neuron bias, element-wise multiplication, activation function, and neuron output respectively. Then the output is given by $y(x) = f(\boldsymbol{w} \odot \boldsymbol{x} + b)$, and it is an approximate representation of the input vector to a level of accuracy that depends on how vast the training (or input) data and training time are [11].

3.1.1 Feedforward Neural Networks

A feedforward neural network has the goal of approximating some function f^*. In the case of a classifier $y = f^*(\boldsymbol{x})$ that maps an input to a category y, a feedforward NN defines a mapping $\boldsymbol{y} = f(\boldsymbol{x}; \boldsymbol{\theta})$ and learns the value of the parameters $\boldsymbol{\theta}$ that best approximate the function. The feedforward models are so named because information flows through the function $\boldsymbol{x}$ being evaluated, through the intermediate computations to define f, to end at the output y. There are no feedback connections where outputs are fed back into themselves, in which case the model would become a recurrent NN or RNN [31]. The RNN model is described in the next section since it is of special interest to this project.

The network part of feedforward NNs comes from the fact that it is associated with an acyclic nature of how its functions are related. For instance, having functions $f^{(1)}$, $f^{(2)}$, and $f^{(3)}$ connected in a chain as $f(\boldsymbol{x}) = f^{(3)}(f^{(2)}(f^{(1)}(\boldsymbol{x})))$, would represent a typical structure of a NN. Additionally, $f^{(1)}$ would be called the first layer, $f^{(2)}$ the second layer, and so on. The depth of the model is determined by the length of the chain, which is where the name "deep learning" originated from. The final layer of the NN is the output layer. The training of a NN involves approximating $f(\boldsymbol{x})$ to $f^*(\boldsymbol{x})$ where the training data yields approximate examples of $f^*(\boldsymbol{x})$ evaluated at different points of the training process. A label $y \approx f^*(\boldsymbol{x})$ accompanies every example $\boldsymbol{x}$, and all training examples directly determine what the output layer must do at each $\boldsymbol{x}$ in order to approximate y. The behavior of the rest of the layers is not directly determined by the training data, but the algorithm must decide how to use these layers to find the best approach to approximate f^*. Since in these layers the training data does not show the desired output, they are said to be hidden layers. The width of a NN is determined by the dimensionality of its hidden layers, each of which is usually vector valued. Each element of this vector or unit plays a role similar to a neuron because it receives input from many other units and computes its own activation value. A layer consists of many units, each representing a vector-to-scalar function [31].

Feedforward NNs use linear models and then find ways to overcome the limitations of these simple models. Linear models such as linear regression are convenient due to how easily and reliably, they can be fit, but their capacity is limited to linear functions, which limits the understanding of the interaction between any two input variables. In order to get linear models to represent non-linear functions of $\boldsymbol{x}$, the linear model could be applied to a transformed input $\phi(\boldsymbol{x})$ rather than $\boldsymbol{x}$ itself, where ϕ is a non-linear transformation. In this approach, ϕ could be described as providing a new representation of $\boldsymbol{x}$ or a set of features describing $\boldsymbol{x}$ and the model would be $y = f(\boldsymbol{x}; \boldsymbol{\theta}, \boldsymbol{w}) = \phi(\boldsymbol{x}; \boldsymbol{\theta})^{\mathrm{T}}\boldsymbol{w}$, where $\boldsymbol{\theta}$ are parameters that describe ϕ and $\boldsymbol{w}$ are parameters that map $\phi(\boldsymbol{x})$ to the desired output. Such an approach is an example of a feedforward network with a hidden layer defined by ϕ with the representation $\phi((\boldsymbol{x}; \boldsymbol{\theta})$ parametrized and an optimization logarithm to find the $\boldsymbol{\theta}$ that corresponds to a good representation [31].

The training of a feedforward network requires the same design decisions as a linear model, such as choosing the optimizer, cost function and the form of the output units. This type of network introduces the use of hidden layers, which requires the choice of an activation function responsible for computing the hidden layer values [31].

3.1.1.1 The XOR Example

A simple depiction of a feedforward network is the "exclusive or function", also named the XOR function. This function is an operation on binary values x_1 and x_2, and targets the desired function $y = f^*(\boldsymbol{x})$ to be learned. The XOR function returns 1 when x_1 or x_2 is 1, and 0 otherwise. In this model, a learning algorithm will adapt parameters $\boldsymbol{\theta}$ to approach f the closest possible to f^* given the function $y = f(\boldsymbol{x}; \boldsymbol{\theta})$. Hence, the goal is for the network to operate correctly on the points X= $\{[0,0]^{\mathrm{T}}, [0,1]^{\mathrm{T}}, [1,0]^{\mathrm{T}}$, and $[1,1]^{\mathrm{T}}\}$. The network is then trained on these four points and it is treated as a regression problem with a square mean error loss function (MSE). The MSE function evaluated over the full training set is

$$J(\boldsymbol{\theta}) = \frac{1}{4}\sum_{\boldsymbol{x}\epsilon \mathrm{X}}(f*(\boldsymbol{x}) - f(\boldsymbol{x};\ \boldsymbol{\theta}))^2 \qquad (3.1)$$

Assuming $f(\boldsymbol{x};\ \boldsymbol{\theta})$ is a linear model with $\boldsymbol{\theta}$ consisting of $\boldsymbol{w}$ and b, it is defined as

$$f(\boldsymbol{x};\ \boldsymbol{w}, b) = \boldsymbol{x}^{\mathrm{T}}\boldsymbol{w} + \mathrm{b} \qquad (3.2)$$

Solving using normal equations (ex. $A^TA\hat{\boldsymbol{x}} = A^T\boldsymbol{b}$) yields $\boldsymbol{w} = 0$ and $b = \frac{1}{2}$. The linear model outputs 0.5 everywhere indicating it cannot be represented by a linear model (as it can be seen in Figure 21), and the use of a transformation is in place. A feedforward network with one hidden layer is presented as a follow up approach. This feedforward network has a vector of hidden units $\boldsymbol{h}$ computed by function $f^{(1)}(\boldsymbol{x};\ \boldsymbol{W}, \boldsymbol{c})$, which are then used as input for the second layer. Although the output layer is still a linear regression model, it is now applied to $\boldsymbol{h}$ rather than $\boldsymbol{x}$. The different functions within the network are now chained together: $\boldsymbol{h} = f^{(1)}(\boldsymbol{x};\ \boldsymbol{W}, \boldsymbol{c})$ and $y = f^{(2)}(\boldsymbol{h};\ \boldsymbol{w}, b)$ and $f(\boldsymbol{x};\ \boldsymbol{W}, \boldsymbol{c}, \boldsymbol{w}, b) = f^{(2)}(f^{(1)}(\boldsymbol{x}))$ overall for the full model. For instance, if $f^{(1)}(\boldsymbol{x}) = \boldsymbol{W}^{\mathrm{T}}\boldsymbol{x}$ and $f^{(2)}(\boldsymbol{h}) = \boldsymbol{h}^{\mathrm{T}}\boldsymbol{w}$, then $f(\boldsymbol{x}) = \boldsymbol{x}^{\mathrm{T}}\boldsymbol{W}\boldsymbol{w}$. It is clear then that these features cannot be described by a linear function. NNs usually do so by applying a transformation determined by learning parameters followed by a fixed nonlinear function called an activation function. Following that same strategy, the vector of hidden units can be defined as $\boldsymbol{h} = g(\boldsymbol{W}^{\mathrm{T}}\boldsymbol{x} + \boldsymbol{c})$ where $\boldsymbol{W}$ contains the weights of a linear transformation and $\boldsymbol{c}$ the biases [31].

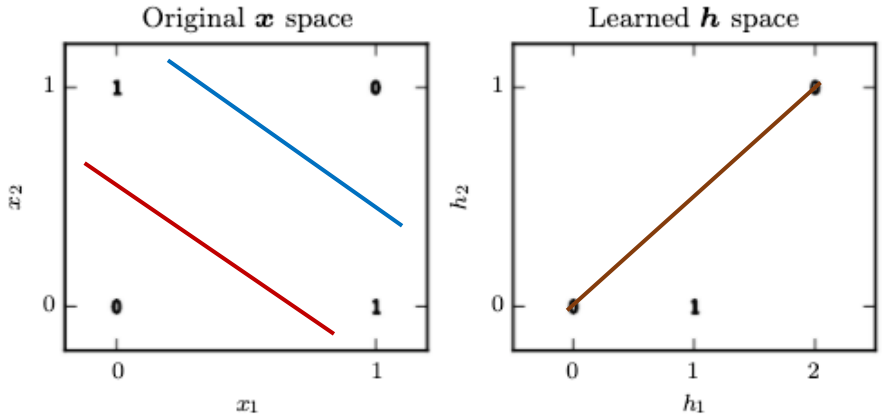


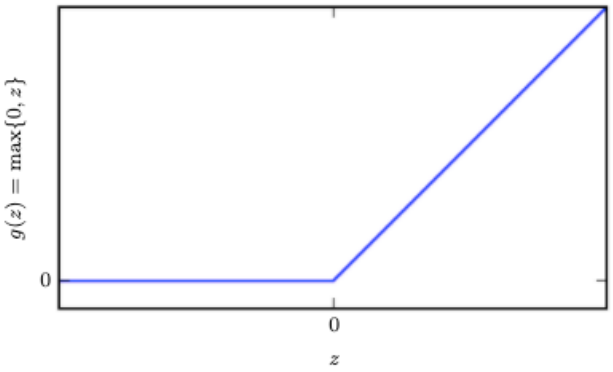
Figure 20. Solving for the XOR problem by learning a representation [modified from [31].

Note: The bold numbers on the graph indicate the value that the learned function is expected to output at each point. The plot on the left shows how the XOR function cannot be implemented by a linear model: when $x_1 = 0$, the output from the model must increase as x_2 increases, and when $x_1 = 1$, the output from the model must decrease as x_2 increases. The two lines show the need for two different regression lines to fit the three different regions of the model. The plot on the right show that a linear model can solve the problem in the transformed space represented by the features extracted by the NN. The line represents a linear regression line fit for the two regions of the model. In this example solution, the two points corresponding to an output of 1 have been collapsed into a single point in the feature space. The motivation of learning the feature space in this example is to make the capacity of the model greater to be able to fit the training set [31].

The next numerical example presents an affine transformation from a vector $\boldsymbol{x}$ to a vector $\boldsymbol{h}$, which will require a bias parameter vector. The activation function g is usually an applied element-wise function defined as $\boldsymbol{h}_i = g(\boldsymbol{x}^{\mathrm{T}} \boldsymbol{W}_{:i} \boldsymbol{x} + \boldsymbol{c}_i)$. The most recommended activation function use presently for NNs is the rectified linear unit, or ReLU [34], [35], [36] defined by the activation function $g(z) = \max\{0, z\}$ shown in the figure below.

Figure 21. The rectified linear unit activation

Note: This activation function is the default recommended one for most feedforward NNs. Its application to the output of a linear transformation results in a nonlinear transformation. Piecewise functions being very closed to linear, they preserve many of the properties that make linear models easy to optimize with gradient-based methods. It also allows to build complicated systems from minimal components [31].



The complete network can then be specified as

$$f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c}, \boldsymbol{w}, b) = \boldsymbol{w}^{\mathrm{T}} \max\{0, \boldsymbol{W}^{\mathrm{T}} \boldsymbol{x} + \boldsymbol{c}\} + b \qquad (3.3)$$

Then we can also specify the XOR problem if we let

$$\boldsymbol{W} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \qquad (3.4)$$

$$\boldsymbol{C} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \qquad (3.5)$$

$$\boldsymbol{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}, \qquad (3.6)$$

$$b = 0.$$

The next steps show how the NN processes a batch of inputs. $\boldsymbol{X}$ is the design matrix, with one sample per row and containing all four points in the binary input space:

$$\boldsymbol{X} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}. \qquad (3.7)$$

First, the NN multiplies the input by the weight matrix of the first layer obtaining:

$$\boldsymbol{XW} = \begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 2 & 2 \end{bmatrix}. \qquad (3.8)$$

Note how this operation collapsed [0 1] and [1 0] into a single point [1 1].

Next, the bias vector $\boldsymbol{c}$ is added:

$$\boldsymbol{XW} + \boldsymbol{c} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}. \qquad (3.9)$$

The examples in this space fit along a line with a slope of 1. The output along this line is expected to begin at 0, then rise to 1, and drop back to 0 again. However, such a function cannot be fit by a linear model, and it is then when the rectified linear transformation is in place. Transforming each example $\boldsymbol{x}$ into $\boldsymbol{h}$ as it is shown in Figure 21 we have:

$$\begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}. \qquad (3.10)$$

The relationship between the examples has been changed, and they no longer lie on a single line but on a space where a linear model can solve the problem as can be seen on Figure 21.

Finally, the NN obtains the correct answer for every example in the batch by multiplying (3.10) by the weight vector $\boldsymbol{w}$:

$$\begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}. \qquad (3.11) \ [31]$$

Although this example shows how a NN model obtains an answer with zero error, in real life situations there might be billions of model parameters and training examples for which a simple solution cannot be guessed. In order to overcome that challenge, a gradient-based optimization algorithm can be used to find parameters that yield very small error. The solution to the XOR problem is a global minimum of the loss function, so gradient descent could converge to this

point. In actuality, gradient descent does not find an exact, integer-valued, and easy to understand solution like the one found in this example [31].

3.1.1.2 Gradient-Descent Learning and the Cost Function

Most of the successful approaches to automatic machine learning can be categorized as gradient-based learning methods. Figure 23 shows how, when applying these learning methods, a learning machine computes a function $M(Z^p, W)$ where Z^p is the p^{th} input in the system, and W is the collection of adjustable parameters in the system. A cost function $E^p = C(D^p, M(Z^p, W))$, measures the discrepancy between desired output, D^p, for pattern Z^p and the output given by the system. The average of the errors E^p over an input-output set, called the training set, is given by the average cost function $E_{train}(W)$. The learning problem consists of finding the value of W that minimizes $E_{train}(W)$. It is of special interest here to measure the error rate of the system in the field, which is estimated by measuring the accuracy on a separate set of samples from the training set, called the test set. The Mean Square Error is the most common cost function used, and it is given by

$$E^p = \frac{1}{2}(D^p - M(Z^p, W))^2, \quad E_{train} = \frac{1}{P}\sum_{p=1} E^p \qquad (3.12)\ [37]$$

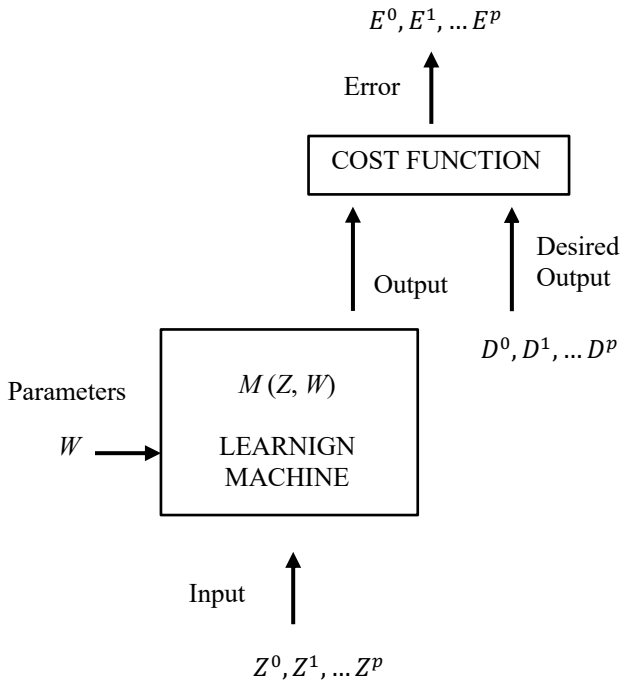


Figure 22. Gradient-based learning machine (modified from [37]).

Among gradient-based methods, the most popular and common one to optimize NNs is gradient descent. Gradient descent is a way to minimize the objective function $J(\boldsymbol{\theta})$ (parametrized by parameters $\boldsymbol{\theta}$ of the model) by updating its parameter in the opposite direction of the gradient of the objective function, $\nabla_\theta J(\boldsymbol{\theta})$, with respect to these parameters. The learning rate η determines the step size in order to reach the local minimum. For instance, the model follows the direction of the slope of the surface created by the objective function downhill until reaching a valley [38].

There are three variants of gradient descent, and they differ in how much data is used to compute the gradient of the objective function. One of these three variants is the batch gradient descent, which computes the gradient of the cost function with respect to the parameters $\boldsymbol{\theta}$ of the entire dataset:

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \eta \cdot \nabla_\theta J(\boldsymbol{\theta}) \qquad (3.13)\ [38]$$

For example, consider the following data set in the table below with $N = 6$ labeled data instances:

Table 5. Data set with 4 features (age, job, education, marital) and label y [39].

i	age	job	education	marital	...	y
1	56	management	university	married	...	0
2	32	unemployed	high school	married	...	1
3	41	technician	university	divorced	...	1
4	51	admin	high school	married	...	0
5	31	entrepreneur	university	single	...	0
6	27	housemaid	high school	single	...	1

During the training process using this data set, the NN computes a prediction that is compared to the ground truth label for each instance. Both the prediction and the label are then used to calculate the loss function for that given sample. However, the weights are not updated until all data instances of the dataset have been processed, but the gradients for each instance in the dataset are calculated and summed. This accumulated gradient is then divided by the number of data instances, which is 6 in this example. Finally, the weights are updated in the negative direction of this averaged sum. Hence, for the given dataset, the gradients for each of the six samples are to be calculated and summed. Then the sum of the gradients is divided by 6 and used to perform single gradient descent to update weights of the NN [39].

Batch gradient descent is computationally efficient, since it does not need to be updated after each sample, and it has a very stable convergence of the weights to the optimal weights. That way the highest increase of the loss function is achieved by getting a very good estimate of the true gradient, since the individual gradients over each sample in the dataset are calculated and

averaged. On the other hand, batch gradient descent calls for slower learning because only one update is performed after N number of samples have been processed. Also, the learning process can get stuck in a local minimum of the loss function to never reach the global optimum, at which the NN achieves the best results, because the calculated gradients are closed to each other. Noisy gradients could, however, overcome this issue by introducing small variations in the directional values that allow the gradient to jump from local minimum of the loss function to continue updating towards the global minimum [39].

The other gradient descent variant is stochastic gradient descent (SGD), which updates the parameters for each training example $x^{(i)}$ and label $y^{(i)}$:

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \eta \cdot \nabla_{\theta} J(\boldsymbol{\theta}; x^{(i)}; y^{(i)}) \qquad (3.14)\ [38]$$

While batch gradient descent for large data sets is redundant as it recomputes gradients for similar examples before each parameter update, SGD does away with redundancy by performing one update at a time. It is hence usually faster and performs frequent updates with a high variance that cause the objective function to fluctuate heavily enabling it to jump to new, and potentially better, local minima. However, this fluctuating behavior might complicate convergence to the exact minimum, which can be avoided by slowly decreasing the learning rate [31].

Consider the data set given in Table 5. In SGD, the prediction is made and compared to the prediction with the label to calculate the gradient of the loss function as well. However, in this case, the weights are update after each data instance (boxed in red in Table 6) has been processed by the NN. Hence, the gradients are calculated and the weights of the NN are updated six times.

***T*able 6. Updated Step with stochastic gradient descent [39].**

i	age	job	education	marital	...	y
1	56	management	university	married	...	0
2	32	unemployed	high school	married	...	1
3	41	technician	university	divorced	...	1
4	51	admin	high school	married	...	0
5	31	entrepreneur	university	single	...	0
6	27	housemaid	high school	single	...	1

Other advantages of SGD are that it provides immediate performance insights, since it is not necessary to wait until the end of the data set to see how the NN is performing, and it also makes it possible for the NN to learn faster because an update is performed after each data instance is processed. On the other hand, STG can be computationally intensive given that the

weight updates are done more often, and it also might be unable to settle on a global minimum of the loss function due to the noisiness of its gradients [39].

A third variant of gradient descent is the mini-batch gradient descent, which combines the best of the other two variants. For the mini-batch gradient descent, the training set is divided into batches of *n* size. For instance, for a dataset with 10,000 samples, a suitable size for *n* would be 8, 16, 32, 64, 128. Just as in the batch gradient descent case , an average gradient is computed across the data instance in a mini-batch. The gradient descent step is performed after each mini-batch of samples has been processed.[38]

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \eta \cdot \nabla_{\theta} J\left(\boldsymbol{\theta}; x^{(i:i+n)}\, y^{(i:i+n)}\right) \qquad (3.15)$$

For instance, if the data set in Table 5 is considered once more, the six data instances may be divided into batches of size $n = 2$, resulting in three mini-batches.

Table 7. Updated step with mini-batch gradient descent [39].

i	age	job	education	marital	...	y
1	56	management	university	married	...	0
2	32	unemployed	high school	married	...	1
3	41	technician	university	divorced	...	1
4	51	admin	high school	married	...	0
5	31	entrepreneur	university	single	...	0
6	27	housemaid	high school	single	...	1

In the given example, two gradients for the two data instances (boxed in red in the table above) in each mini-batch are calculated and divided by two to obtain the average gradient over that mini-batch. This average gradient is used to perform a gradient descent step, which is done a total of three times [39].

The computational efficiency of mini-batch gradient descent is between that of the two variants mentioned earlier, and it is more stable converging towards a global minimum, since the average gradient is calculated over *n* samples that results in less noise. This variant also allows faster learning given that the weights are updated more often than it is in the other two variants, which results in a much faster learning process. However, mini-batch gradient descent requires the introduction of the new hyperparameter *n*, which becomes the second most important parameter for the overall performance of the NN. It is important, then, to take the time to try many different batch sizes until a final batch size that works best with the other parameters, such as the learning rate, is found [39].

For this project, the batch gradient descent approach is being used for the time steps of a single full elliptical orbit at present. Once the dataset includes other orbits considered for the training of the NN, the mini-batch gradient descent will be adopted with a batch number matching the number of elliptical orbits used during training. However, STG and mini-batch gradient descent will be considered as well for training using the first orbit being analyzed to compare the results between these different approaches in an effort to find the best resolution possible.

In this section, the feedforward deep network was presented as a typical example of deep neural network methods. As mentioned in the beginning of section 3.1.1, feedforward models are named as such because information flows forward the NN. There are no feedback connections where outputs are fed back into itself, in which case the model would become a recurrent NN or RNN [31]. The RNN model would be described next since it is of special interest to this project.

3.2 Recurrent Neural Networks

A common assumption about machine learning models, including NNs, is the independence among data samples. This assumption, though, does not hold for sequential types of data, such as time series, speech, language, etc. A way to account for sequential dependency is to concatenate a fixed number of consecutive data samples together to be treated as one single data point. However, this approach has proven to be highly dependent on finding the optimal window size, since a small window size does not capture the longer dependencies, and a too large window size would add unnecessary noise. Furthermore, in the case where long-range dependencies in data ranging over hundreds of time steps are present, a window-based method would not scale. In addition, conventional NNs cannot handle variable length sequences, which is the case for many domains such as language translation and speech modeling [13].

While feedforward NNs are limited to passing the data forward from input to output, recurrent NNs (RNNs) have a feedback loop where data can be fed back into the input at some point before it is fed forward again for further processing and final output [40]. RNNs also have the ability to use their feedback connections to store representations of recent input events in the form of activations. Therefore, RNNs can be very useful when dealing with time dependency data [40], [41] as it is the case for the proposed problem in this report.

RNNs process the input sequence one element at a time while maintaining a hidden state vector acting as a memory for past information. The learning process is selective to relevant information allowing the NN to capture dependencies across several time steps, which makes it possible to use both current input and past information while making future predictions. In this model, learning happens automatically without much knowledge of the cycles or time dependencies in data. RNNs eliminate the need of fixed size time window and can handle variable length sequences.

Recall Figure 24 shown in chapter one, where an RNN and the unfolding in time of the computation involved in its forward computation is shown [33]:

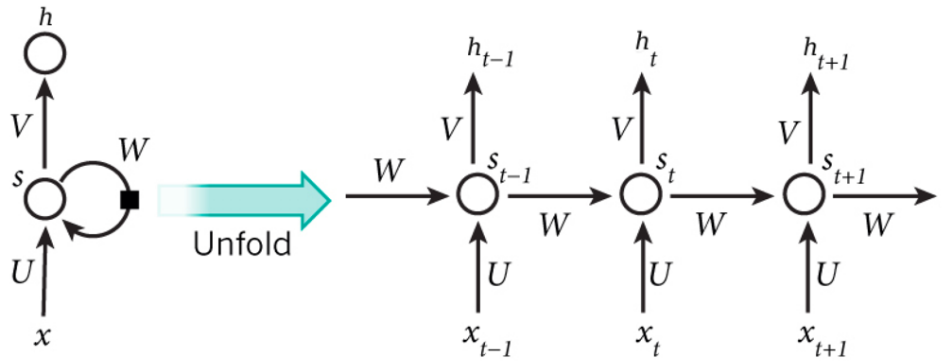


Figure 23. Recalling figure 10, a standard RNN and its unfolding in time [13].

The artificial neurons shown in the figure above as hidden units grouped under node *s* with values s_t at time *t*, get inputs from the other neurons at the previous time steps. The figure on the left represents a standard RNN with a circle depicting a time step and a black square the delay of the time step while this process takes place. This can be considered a feedback connection of the hidden neurons across time. Through this process, an RNN can map an input sequence with elements x_t into an output sequence with elements o_t depending on all the previous x'_t (for $t' \leq t$). At time *t*, the RNN receives as input the current sequence element x_t and the hidden state from the previous time step s_{t-1}. The hidden state is then updated to s_t and finally the output of the network h_t is calculated. Hence, the current output h_t depends on all the previous inputs x'_t. The sample parameters represented by matrices *U, V, W,* are utilized every time step. *U* is the weight matrix between the input and hidden layers similar to a conventional NN. *W* is the weight matrix for the recurrent transition between one hidden state to the next. *V* is the weigh matrix for the hidden to output transition [13], [33]. The following equations summarize the computations carried out at each time step:

$$s_t = \sigma(Ux_t + Ws_{t-1} + b_t)$$
$$h_t = softmax(Vs_t + b_h) \qquad (3.16)\ [13]$$

The *softmax* in 3.16 represents the softmax function, which is often used as the activation function for the output layer in a multiclass classification problem, and *b* is the bias at the corresponding level of the RNN. This function ensures that all the outputs range from 0 to 1 and their sum is 1. For a *K* class problem, the softmax equation is:

$$y_k = \frac{e^{a_k}}{\sum_{k'=1}^{K} e^{a_{k'}}} \; for\ k = 1, \ldots, K \qquad (3.17)\ [13]$$

In this equation, a is a parameter learned during training. The standard RNN shown in Figure 24 can be considered a deep network with the number of layers equivalent to the number of time steps in the input sequence. An RNN can process variable length sequences, since the same weights are used for each time step. A new input is received at each time step, and, given that the hidden state s_t is updated via equation 3.16, the information can flow in the RNN for the

arbitrary number of time steps. This allows the RNN to maintain a memory of all the past information [13].

Many other variants than those observed in Figure 24 are possible, such as one where the network generates a sequence of outputs, like words, each of which is used as an input for the next time step. The backpropagation logarithm can be directly applied to the unfolded computational graph network on the right to compute the derivative of a total error with respect to all the states s_t and all the parameters [33]. Backpropagation is the most common method used to train RNNs and it is discussed in detail in the next section.

3.2.1 RNN Training with Backpropagation

When a feedforward NN is used to accept an input $\boldsymbol{Z}$ and produce an output Y, information flows forward through the network. The initial information provided by input $\boldsymbol{Z}$ then propagates up to the hidden layers to finally produce output Y. This is call forward propagation. Forward propagation can continue onward during training until it produces a scalar cost $E(\boldsymbol{W})$. The backpropagation algorithm or backprop for short, allows the information from the cost function to then flow backward through the network in order to compute the gradient [31].

Backprop algorithm has been cornerstone in machine learning to train NNs, with a rich history of having been reinvented several times by independent researchers (Griewank, 2012; Schmidhuber, 2015). It has been one of the most studied and used training algorithms since it gained popularity mainly through the work of Fumenlhart el al. (1986) [42]. The backprop procedure to compute the gradient of an objective function with respect to the weights of a multilayer stack of modules is simply a practical application of the chain rule for derivatives. The key concept of this method is that the derivative, or gradient, of the objective with respect to the input of a module can be computed by working backwards from the gradient with respect to the output of that module (which is also the input of the subsequent module). The gradients can be propagated through all modules by applying the backpropagation equation repetitively, starting from the output at the top where the network produces its prediction, all the way to the bottom where the external input is fed. Computing the gradients with respect to the weights of each module becomes simple once these gradients have been propagated through all modules (refer to Figure 24 to understand this process) [33].

Many deep learning applications use feedforward NN work architectures, which learn to map a fixed-sized input to a fixed-size output. In order to move from one layer to another, a set of units compute a weighted sum of their inputs form the previous layer and pass the result through a non-linear function such as the rectified linear unit (ReLU), which was introduced in chapter 1 and shown again when solving for the XOR example problem in sub-section 3.1.1.1. In past decades, NNs used smoother non-linear functions like the $\tanh(z)$ or the sigmoid functions, but ReLU has proven to be faster, especially with NNs with many layers. As seen before, units that are not on the output or input layers are called hidden units. The hidden layers can be seen as distorting the input in a non-linear way in Figure 25, so that categories become linearly separable by the last layer [33].

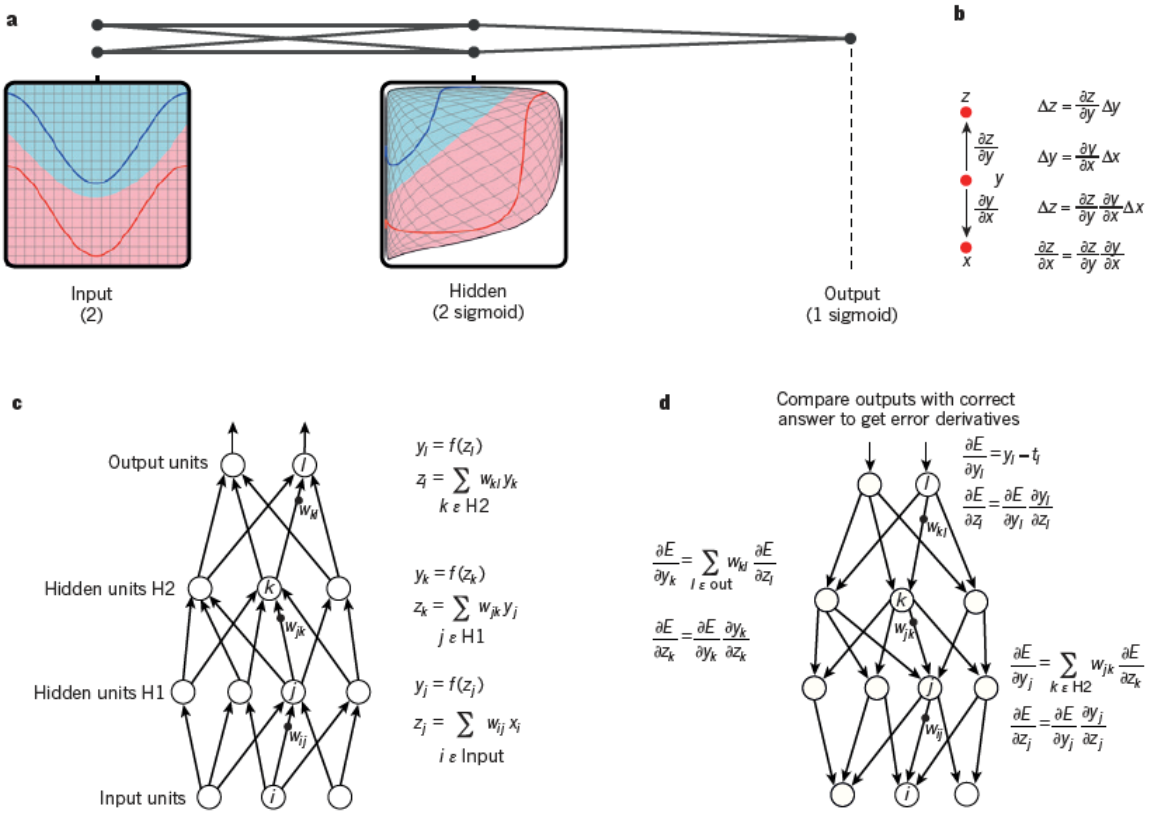


Figure 24. Multilayer NNs and backpropagation [33].

Note: Reproduced with permission from C. Olah. (http://colah.github.io/).

In Figure 25:

a. The connected dots represent a multilayer NN distorting the input space to make the classes of data (an example of which is shown by the red and blue curves) linearly separable. As it can be seen on the left, the grid of the input class is also transformed as seen on the right by the hidden units. This example illustrates the initiation of the process with only two input units and two hidden units, but in a real word application, NNs contain tens or hundreds of thousands of units.

b. The chain rule of derivatives describes how the small change of x on y, and that of y on z are composed. A small change Δx in x gets transformed first into a small change Δy in y when getting multiplied by $\partial y/\partial x$, the definition of the partial derivative. In the same way, the change Δy creates a change Δz in z. Substitution of one equation into the other gives the chain rule of derivatives, which is how Δx gets turned into Δz through multiplication by the product of $\partial y/\partial x$ and $\partial z/\partial x$. It works the same way when x, y and z are vectors, and the derivatives are Jacobian matrices.

c. The equations for computing a forward pass in a NN with two hidden layers and one output layer are shown, each constituting a module through which gradients can be backpropagated.

The total input z at each layer is first computed for each unit, which is a weighted sum of the outputs of the units in the layer below. A non-linear function $f(z)$ is then applied to z to get the unit output. Note that the bias term has been omitted in this example just for simplicity. The non-linear functions used in NNs include the ReLU, $f(z) = \max(0, z)$ the hyperbolic tangent, $f(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$, and the logistic function $f(z) = \frac{1}{1+e^{-z}}$.

d. The equations used for computing the backward pass are shown. The error derivative with respect to the output of each unit are computed at each hidden layer. This is a weighted sum of the error derivatives with respect to the total inputs to the units in the layer above. Then the error derivative with respect to the output is converted into the error derivative with respect to the input by multiplying it by the gradient of $f(z)$. The error derivative with respect to the output of a unit is computed at the output layer by differentiating the cost function. This gives $y_l - t_l$ if the cost function for unit l is $0.5(y_l - t_l)^2$ where t_l is the target value. Once $\partial E/\partial z_k$ is known, the error-derivative for the weight w_{jk} on the connection from the j unit layer below is just $y_j (\partial E/\partial z_k)$ [33].

RNN training can be achieved by unfolding the RNN and creating a copy of the model for each time step just as in the unfolded RNN part of Figure 24. Then, the RNN can be treated as a multilayer NN and be trained in a way similar to backprop. This approach is called back propagation through time (BPTT). Ideally, RNNs can be trained to learn long-range dependencies over arbitrarily long sequences using BPTT by learning to tune weights to put the right information in memory. In practice, though, training an RNN is not simple, and it can perform poorly even when the outputs and relevant inputs are separated by only 10-time steps. Using BPTT to train a RNN requires backpropagating the error gradients across several time steps. It can be seen in Figure 24, that in a standard RNN the recurrent edge has the same weight for each time step. Therefore, back-propagation of the error involves multiplying the error gradient with the same value repetitively, which causes the gradients to become either too small or too large. This problem is known as exploding and vanishing gradients respectively [13].

Modification to the training procedure and new RNN architectures were proposed to deal with the exploding and vanishing gradient problems such as LSTM RNNs. The LSTM architecture has been investigated and proven to be very useful in learning long-term dependencies as compared to standard RNNs and have become a popular variant of RNN [13]. The LSTM architecture is the approach used in this project, and it will be presented in detail in the next chapter, along with the specific setup of the LSTM RNN used to solve the problem. In summary, the artificial intelligence computational procedures presented in this chapter were implemented in an LSTM RNN via the deep learning Application Programming Interface (API), Keras, based in Python and running on top of the open-source machine learning platform, TensorFlow.

Chapter 4: LSTM RNN Implementation

As mentioned in the previous chapter, RNNs are networks with loops in them to allow information to persist. An RNN can be thought of as multiple copies of the same network, each of which passes the message to a successor just like in the unrolled RNN shown in Fig. 24. Hence, one of the appeals of RNNs is the idea that they might be able to connect previous information to the present task. This is sometimes possible, such as in the case of a language model trying to predict the next word based on the previous one in a sentence. If trying to predict the last word in "we live on the planet *Earth*," no farther context is needed, since it is obvious the next word is going to be Earth. In such cases, RNN have no problem learning to use past information because the gap between the relevant information and the place where it is needed is small. However, there are cases that need more context. Consider the case where the last word is to be predicted in the sentence "I am from Mexico…I speak fluent *Spanish*." Recent information suggests for the next word to be the name of a language, but in order to narrow down which language, the context of Mexico is needed from further back. In this case, it is highly probable that the gap between the relevant information and the point where it is needed becomes very large and RNNs become unable to connect the information. This is known as the problem of Long-Term Dependencies [43].

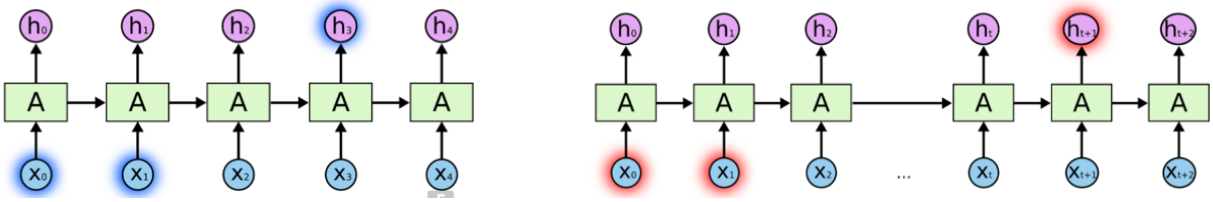


Figure 25. Long-Term Dependency Problem [43].

Left - Unrolled RNN able to learn past information. Right - Unrolled RNN unable to learn past information due to the long gap between relevant information and the point where it is needed.

Long Short Term Memory networks, or LSTMs for short, are a type of RNNs that were introduced by Hochreiter & Schmidhuber in 1997 as a means to make RNNs capable to learn long-term dependency problems and avoid the exploding or vanishing gradient problem mentioned in the previous chapter. LSTMs were improved in following work by many people and are now widely used. They are highly effective to solve a variety of problems and their default behavior is remembering information for long periods of time [43]. LSTMs are very well suited to solve the problem proposed in this project because they use current input and past information while making future predictions and retaining dependencies across short or large time steps by learning how to retain relevant information.

4.1 LSTM RNN Architecture

All RNNs have a chain of repeated modules of NN. In the case of standard RNNs, a simple structure, such as a single tanh layer, is the repeating module (as shown in Figure 26 below).

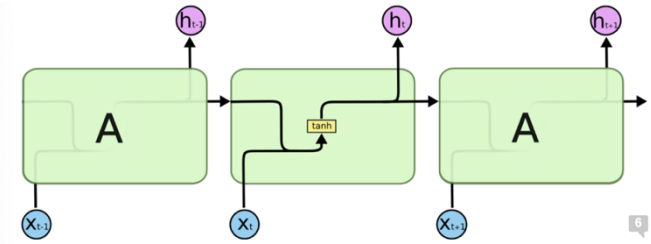


Figure 26. RNN with repeating module containing a single layer [43].

LSTMs also have a chain of repeating modules, but in this case, the module has four NNs (rather than one) interacting in a specific way (see Figure 28 below).

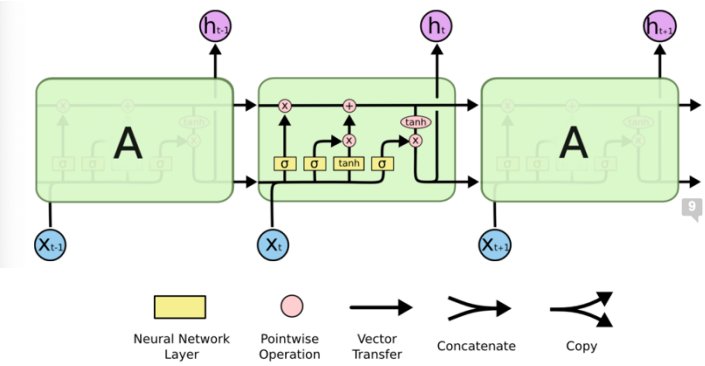


Figure 27. An LSTM has a repeating module containing four interacting layers.

In Figure 28 each line passes an entire vector starting from the output of one node to the input of other nodes. The pink circle represents pointwise operations, such as vector addition, and the yellow boxes are NN layers learned. The lines merging represent concatenation, while the forking line represents its content being copied and being sent to different locations.

The core idea behind LSTMs is the cell state represented by the horizontal line running through the top of the diagram. The cell state runs straight down the entire chain with minor linear interactions only, making it easy for information to just flow along it unchanged.

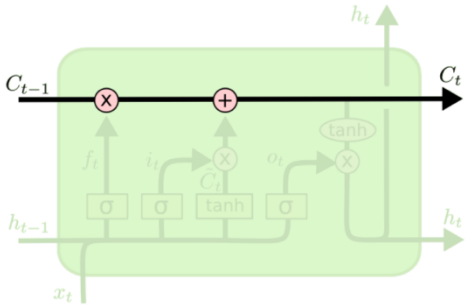


Figure 28. The cell state of an LSTM [43].

The LSTM has structures called gates that regulate the ability to remove or add information to the cell state. Gates are composed of a sigmoid NN layer and a pointwise multiplication operation and are a way to optionally let information through.

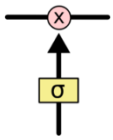
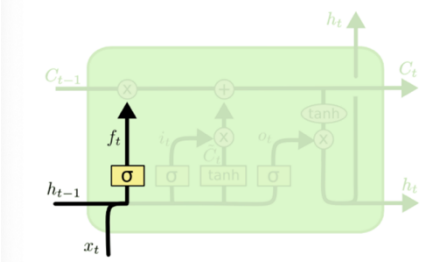


Figure 29. An LSTM gate structure [43].

4.1.1 Steps in an LSTM Walk Through

1. An LSTM network begins by deciding what information will be discarded from the cell state. This decision is made by a sigmoid (σ) layer called the *forget gate layer* (f_t), which looks at previous cell output, h_{t-1}, and input vector, x_t, and outputs a number between 0 and 1 for each number in the cell state C_{t-1}. An output of 1 corresponds to completely keeping the information at that point while an output of 0 represents to completely discarding the information at that point. For instance, in the case of the previous example of a language model trying to predict the next word based on all the previous ones, the cell state might decide to include the gender of the subject for the correct pronouns to be used. When a new subject is perceived, the gender of the previous subject is forgotten. The figure below shows this process and calculation of the forget gate layer with W_f and b_f being the vector weights and biases of the layer.



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Figure 30. The forget gate layer of an LSTM [43].

2. The next decision to be made is what new information to store in the cell state, which has two parts to it. A sigmoid layer called the *input gate layer* (i_t) decides which values will be updated first. Then, a tanh layer creates a vector of new candidate values, $\tilde{C}_t$, that might be added to the state. These two will be combined in the next step to create an updated state. Following the example of the language model, the gender of the new subject would be added to the cell state to replace the forgotten old subject's gender. The following figure shows this process and calculation of the input gate layer and candidate values with W_i and b_i being the vector weights and biases of the input layer and W_c and b_c those of the new candidate values.

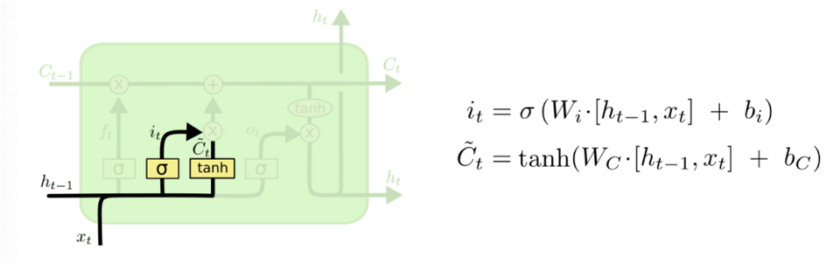


Figure 31. The input gate layer of an LSTM [43].

3. Next, the old cell state, C_{t-1}, is updated to the new cell state or candidate gate, C_t. Now that previous steps have decided what to do, the new state is calculated by multiplying the old state by f_t (to forget the things decided to be forgotten earlier) and add $i_t * \tilde{C}_t$. This now represents the new state value scaled by how much was decided to update each state value. Proceeding with the example of the language model, this is where the information about the gender of the old subject is dropped to add new information, as it was decided in the previous steps.

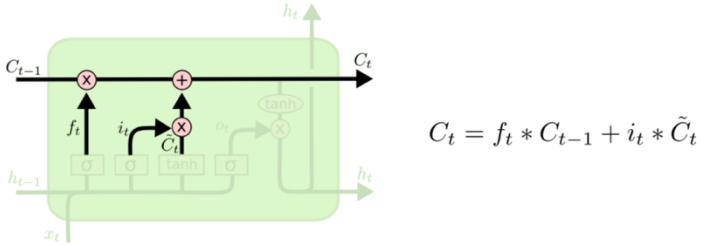
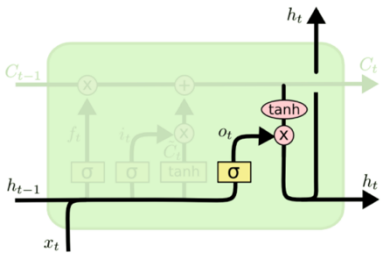


Figure 32. The new cell state of an LSTM [43].

4. In this last step, the output (o_t) is decided by filtering the cell state. First, a sigmoid function decides what parts of the cell will become part of the output. Then, the cell state goes through a tanh function to push its values between -1 and 1. The resultant values are then multiplied by the sigmoid gate output in order to output only the parts that have been decided. In the example language model, for instance, the output information relevant to a verb might be considered, since it just saw a subject. For example, the output might be whether the subject is singular or plural in order to know the form in which a verb should

be conjugated given that is what follows [43]. The following figure shows this process and calculation of the output gate layer and candidate values with W_o and b_o being the vector weights and biases of the output layer and h_t is the current cell output.



$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Figure 33. The output gate layer of an LSTM [43].

There are other variant of LSTMs involving slightly differences that might better serve the purpose of other problems under consideration. These variants, however, are not relevant to this project and won't be discussed.

4.2 LSTM RNN Application Programming Interface and Machine Learning Platform

This project uses the deep learning Application Programming Interface (API) Keras running on top of the machine learning platform TensorFlow [44]. Keras is a powerful, user friendly open-source Python library for developing and evaluating deep learning models such as neural networks [45]. Its fast prototyping and experimentation with a simple API make it highly suitable to solve the problem proposed in this project. It allows the configuration of NNs in a modular way by combining different layers, activation and loss functions, as well as optimizers, etc. Keras also contains implementation of LSTM with forget gates as described in the previous section with BPTT implemented and state maintenance [13].

4.2.1 Keras BPTT Implementation

It can be recalled from section 3.2.1 that RNN training can be achieved by unfolding the RNN and creating a copy of the model for each time step just as in the unfolded RNN part of Figure 24. Next, the RNN can be treated as a multilayer NN and be trained in a way similar the backprop (explained in section 3.2.1). This approach is what is called back propagation through time (BPTT). Keras has a modified version of BPTT implementation. Given that unfolding RNNs across an input sequence of thousands of time steps is computational inefficient, an RNN in Keras is unfolded to a maximum number of time steps. This parameter is specified when inputting the data, which is to be fed in the form of a 3-dimensional array of shape: (*batch, timesteps, feature)*. The *batch* argument is the number of data points or samples, *timesteps* specifies the past observations for a feature or steps for which the RNN is unfolded, and *features,*

in this case, are the three components of the spacecraft's position vectors in cartesian coordinates. The input data is divided into sequences that overlap and with a time interval of one. For this reason, it is important that the time steps are consistent across the whole data sample for all the bank of data training the model. Each sequence has *a timesteps* number of consecutive time steps and forms one training sample to the RNN model. BPTT is done over individual samples for *timesteps* number of time steps during training [13].

4.2.2 State Maintenance in Keras

LSTM in Keras is maintained in two different ways:

1. Default Model: The samples in a batch are assumed to be independent of each other, and state is preserved only over individual input sequences for *timesteps* number of time steps.

2. Stateful Mode: The state cell is maintained among various training batches in this mode. The final state of the i^{th} sample of one batch is used as the initial state of the i^{th} sample of the next batch with the samples within one batch staying independent. A one-to-one mapping between samples of consecutive batches is assumed in order to maintain state across batches. Therefore, shuffling of samples should be avoid in this mode.

The reason behind the independence of samples within a batch, lays back on the history of the development of LSTMs. Language modeling and recognition tasks motivated this implementation, since they were the key ideas driving LSTM development and implementation. Language models training tasks contain samples that are usually individual sentences, and a short *timesteps* value equal to the maximum length of the sentence, in words, is enough to capture the necessary sequential dependencies. Hence, different samples can be treated independently. However, this behavior could be quite restrictive for many domains of datasets [13].

For the purpose of this report, state maintenance has been set up by constructing a many-to-many model with a Time Distributed Dense layer, both of which will be explained in detailed in the next section with its implementation in the algorithm to be presented.

4.2.3 TensorFlow

TensorFlow is an easy-to-use, open-source Python library for numerical computation that makes machine learning faster and easier. It is a Google initiative with machine leaning frameworks that eases the process of acquiring data, training models, serving predictions and refining future results. It uses Python as a front-end API for building applications with the framework, while executing those application in high-performance C++.

The way TensorFlow works is by allowing developers to create dataflow graphs, which are structures that describe how data moves through a graph, or a series of processing nodes. Each node in the graph represents a mathematical operation, and each connection or edge between

nodes is a multidimensional data array, or a tensor. All this is provided to the programmer via Python language. Hence, TensorFlow applications are Python applications themselves, and the nodes and tensors are Python objects. However, the actual math operations are done via libraries of transformations available through TensorFlow written as high-performance C++ binaries. That way, Python only directs the traffic between the pieces, and provides high-level programming abstractions to hook them together [46].

TensorFlow supports most modern platforms such as a local machine, cluster in the cloud, iOS and Android devices, CPUs or GPUs. If used in Google cloud, TensorFlow can be ran on Google's custom TensorFlow Processing Unit (TPU) silicon for further acceleration [46]. Its integration with Keras API began with the release of TensorFlow 1.0 in February 2017 and enhanced to version TensorFlow 2.0 in October 2019 to make Keras its central high-level API, easier to work with and improve training performance and runtime [47].

4.3 Algorithm and LSTM RNN Set Up

The LSTM RNN (LSTM for short) is trained on "input" data with the position of a spacecraft at every minute (each of which is a time step) of its trajectory around Venus with no solar perturbations. This input data is to approach the "label" data composed of the spacecraft's position at every minute on its trajectory around Venus with solar perturbations, with the same exact initial conditions to those of the input data and at the corresponding time steps. Due to the shorter orbital period of the perturbed orbit, the label data had to be trimmed one step (corresponding to one minute) to match the number of steps of the input data. The full data set is composed of three slightly different orbits with and without solar perturbations for label data and input data respectively.

For the purpose of training the LSTM, the data set was divided into three subsets (each corresponding to every one of the three slightly different orbits): a training set with data corresponding to the orbit with similar initial conditions to those of the Venera D mission; a testing set with data corresponding to the same orbit with a shift of +15° in inclination; and a validation set with data corresponding to the same orbit with a shift of -15° in inclination.

The LSTM training is being done in Google Colab (abbreviated term for Colaboratory) with graphics processing units (GPUs) processors. Colab is a product from Google Research that allows anybody to write and execute arbitrary Python code through the browser. It is well suited for machine learning, data analysis and education. It is hosted by Jupyter notebook service that requires no setup to use, while providing free access to computing resources including GPUs. Jupyter notebooks allow anyone to use and share data with others without the need to having to download, install or run anything [48]. GPUs are specialized electronic circuits designed to manipulate and alter memory rapidly to accelerate the creation and display of images in a device. Modern GPUs are very efficient at manipulating computer graphics and are more efficient that the general-purpose central processing units (CPUs) for algorithms that process large blocks of data in parallel [49].

The libraries used in this project are presented in table below.

Table 8. Software libraries used in this project.

Library	Version
Keras	2.4.0
TensorFlow	2.4.1
sickit-learn	0.24.1
Mathplotlib	3.4.1
Pandas	1.2.4

The Keras API and TensorFlow libraries have been introduced above and are the main banks of libraries used to train the RNN. The sickit-learn library contains efficient tools for predictive data analysis, and it is built in on NumPy (library for multidimensional arrays, matrices and high-level mathematical functions), SciPy (mathematical algorithms and convenience functions built on NumPy), and matplotlib (cross-platform, data visualization and graphical plotting library). Pandas is a column-oriented data analysis API and will be introduced with more detail in the next section.

4.3.1 Data Gathering and Preparation for Training

Data for the LSTM training was gathered and prepared by mounting Google Drive to the Google Colab runtime's virtual machine using an authorization code, and the Python Data Analysis (Pandas) library.

4.3.1.1 Fetching the Data

Google Drive was mounted in runtime via, mainly, the following python commands:

```
from google.colab import drive
drive.mount('/content/drive')
```

These commands give the virtual machine access to a Google Drive to find and read the file with target data given that an authorization code, automatically generated in runtime, is provided. This process is required every time access to data in a Google Drive is needed. The Pandas library is an open-source Python package mostly used for data science, data analysis and machine learning tasks. It is built on top the NumPy package, which provides support for multi-dimensional arrays [50]. Some of the best functionalities of Pandas are data loading, reading, renaming, mapping, shaping, groupby and statistics, joining, masking and handling missing values [51]. The data set for LSTM training was gathered from three different folders in Google Drive, each one corresponding to the data of each of the three orbits used for training, testing and validation.

4.3.1.2 Data Splitting, Normalization and Reshaping

The data were split into three sets corresponding to each on the three orbits, and each of the three data sets were split once more into two data subsets: the train data set (X) and the label data set (Y).

Normalizing the data for RNN training generally speeds up the learning process and makes it more stable leading to faster convergence because:

1) The weights of a model are initialized to small random values and updated via an optimization algorithm in response to estimates of error on the training dataset. Hence, unnormalized data can lead to large or small error estimates that might lead to vanishing or exploding gradients.
2) It is much easier for the RNN to perform the necessary operations on numbers between 0 and 1 than it is with larger or smaller numbers [52].

There are other reasons why normalization favors RNN training, but those do not apply to the type of data being handled in this project.

Data normalization was done separately across the unperturbed or training data of each orbit and the perturbed or label data of each orbit using *Min-Max Normalization*. In order to do so the *Min-Max Normalization* formula was used:

$$X_{normalized} = \frac{X - X_{min}}{X_{max} - X_{min}} \qquad Y_{normalized} = \frac{Y - Y_{min}}{Y_{max} - Y_{min}}$$

In the above equations, X stands for training data and Y for label data, $X_{normalized}$ and $Y_{normalized}$ are the normalized values of each data point in both data sets, X and Y represent the unnormalized values, X_{min} and Y_{min} are the minimum points and X_{max} and Y_{max} are the maximum points in the data sets. This process resulted in the data being shifted to values between 0 and 1. The normalized data was then reshaped to feed to the LSTM.

As mentioned in section 4.2.1, data is to be fed to an LSTM in the form of a 3-dimensional array of shape: (*batch, timesteps, feature)* where the *timesteps* are the past observations for a feature. The reshaping was done using a batch size of 404, a *timesteps* number of 5 and an input dimension of 3 features (one for each component of the position vector). A batch size of 404 splits the 2020 data points into 5 even parts (one for each *timesteps*). This batch size was chosen to make the training run faster given that a many-to-many model was adopted (which takes more time to compute). The many-to-many model will be explained in detail in the next section.

4.3.4 The Model

This project uses a *Sequential Model*. NNs are defined in Keras as a sequence of layers. The *Sequential* class is the container of these layers. Once an instance of the *Sequential* class is created, layers can be created and added in the order that they should be connected to train the NN. The LSTM recurrent layer composed of memory units is called LSTM(). A fully connected layer that often follows LSTM layers used to output a prediction is called Dense(). For example, an LSTM hidden layer with 2 memory cells followed by a Dense output layer with 1 neuron can be define as:

```
model = Sequential()
model.add(LSTM(2))
model.add(Dense(1))
```

or it can be done in one step by creating an array of layers and passing it to the constructor of the Sequential class [53]:

```
layers = [LSTM(2), Dense(1)]
model = Sequential(layers)
```

Creating a *Sequential Model* via Keras with an LSTM with 50 memory cells and its hyperparameters (to be discussed next) followed by a Dense output layer of 3 cells will look like:

```
model=keras.models.Sequential([ keras.layers.LSTM(50, hyperparameters)
                                keras.layers.Dense(3)]
```

4.3.4.1 LSTM Hyperparameters

As one might recall from previous sections, the training of an NN is converted into an error minimization or optimization exercise aiming to minimize the loss function (equation 1.2 introduced in page 16) by tuning its parameters. The algorithm used to perform optimization is called "gradient", which involves calculating the gradients of the loss function with respect to the network parameters, such as weights and biases. Gradients are computed via the back-propagation method based on the chain rule of derivatives as shown in section 3.2.1. Recall that the gradient is a measure of the change in the loss value corresponding to a small change in a network parameter according to equation 1.2, $\boldsymbol{\theta} = \boldsymbol{\theta} - \gamma \frac{\partial L(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$, which depends on the learning rate scalar, γ, use to update the parameters, $\boldsymbol{\theta}$, in the opposite direction of the gradient. This process makes several passes iteratively over the training data, and every pass or *epoch* moves the parameters closer to their optimum values which minimizes the loss function [13].

For large sets of data, calculating the loss and gradient over the entire dataset can be computationally slow and infeasible. Therefore, variants of gradient descent called *optimizers* are used. Optimizers divide the data into subsets called batches, and the parameters are updated after calculating the loss function over one batch. Popular optimizers are SGD (stochastic gradient descent), RMSprop, AdaGrad and Adam [13]. The optimizer used in this project is Adam, which is a combination of AdaGrad and RMSprop, and will be explained in more detailed in the discussion about the LSTM architecture for this project.

A common problem when training NNs, is *overfitting.* Overfitting occurs when the model tries to fit the noise in training data, and it is often caused by using a more complex model than required. When overfitting, the model performs well on training data but poorly on new data. Overfitting during training can be avoided in several ways. In *early-stopping*, a small subset of data is used as a validation set, and the loss function on the training set is compared to the value on the validation set after every epoch. When the loss of the validation set starts increasing despite the loss on the training set is decreasing, overfitting is taking place, and the model can be stopped. Another common method used in deep learning to avoid overfitting is *dropout*. In

dropout, a fixed percentage of NN connections are randomly removed in each training epoch [13].

Network parameters, such as weights and biases, are learned by the training algorithm. On the other hand, parameters such as learning rate, dropout, training batch size, decay, etc. are parameters of the learning algorithm that need to be set with the appropriate values by the user, and they are called *hyperparameters* [13]. The following table presents the hyperparameters used for the LSTM model, and each of them will be explained in the model architecture section coming next.

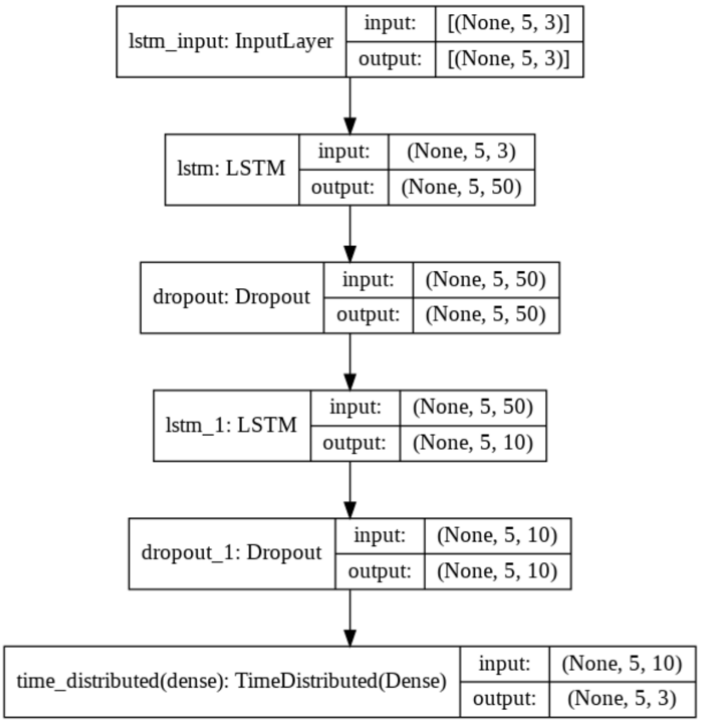
Table 9. LSTM Hyperparameters.

Hyperparameter	Type/Value	Location
activation	relu	LSTM layers
recurrent_activation	zeros	LSTM layers
bias_initializer	zeros	LSTM layers
kernel_initializer	glorot_uniform	LSTM layers
recurrent_initializer	glorot_uniform	LSTM layers
stateful	False	LSTM layers
return_sequences	False	LSTM layers
return_state	False	LSTM layers
Dropout	regularizer	2nd layer
TimeDistributed	wrapper layer	Dense layer
Adam	optimizer	After model
learning rate (lr)	0.0002	Within optimizer
loss	mse	Model compiler
metrics	accuracy	Model compiler
batch_size	101	Model training history
epochs	2000	Model training history
validation_freq	1	Model training history
shuffle	False	Model training history

4.3.4.2 LSTM RNN Model Architecture

The RNN comprised of an LSTM layer with 50 units, followed by a regularizer Dropout layer with a rate of 0.46, a second LSTM layer with 10 units, and an output TimeDistributed wrapped Dense layer with 3 units. The architecture of the model can be seen in the mapping of table 10 below.

Table 10. LSTM RNN architecture Map.



As can be seen, the different layers of the RNN take into 3-dimensional data arrays, of which the first element (*batch* or number of units) is read as "None" meaning that it is up to the RNN to determine that argument during the learning process. As mentioned before, the LSTM arguments were set to be *timesteps* = 5 and *feature* = 3 for the input layer. The 2020 input data points were divided into 5 parts, which resulted in a batch or sample size of 404 with five data points per feature each. After the input is taken, it is up to the RNN to decide the batch size that will better suits the training.

The hyperparameters in the LSTM RNN model are defined as follows:

activation – The activation function is responsible for transforming the summed weighted input from the node into the activation of the node or output for that input. The ReLu is a piecewise linear function that will output the input directly if positive, or output zero otherwise. It has become the default activation function for many types of NNs because it makes it easier to train a model and enhances its performance. This is another reason why the dataset used in this project was normalized [54].

recurrent_activation – Activation function to use for the recurrent step during training and set to the ReLu function in this case.

bias_initializer – Initializer for the bias vector of the weights, which is set up to be initialized at zero by default. Recall that the bias is analogous to the intercept in a linear equation. It is an additional parameter in the RNN which is used to adjust the output along with the weighted sum of the inputs to the neuron.

kernel_initializer – Initializer for the kernel weights matrix, used for the linear transformation of the inputs. It is set to be glorot_uniform as a default in Keras. The Glorot uniform initializer draws samples form a uniform distribution within [-limit, limit], where limit = $\frac{\sqrt{6}}{W_{in}+W_{out}}$, and W_{in} and W_{out} are the are the input and output units of the weight tensor respectively.

recurrent_initializer – Initializer for the recurrent_kernel weights matrix, used for the linear transformation of the recurrent state, which is set by default as orthogonal. In this RNN training it was set equal to glorot_uniform as the kernel weight matrix initializer.

stateful – Boolean parameter set to False by default. If True, the last state for each sample at index *i* in a batch will be used as initial state for the sample of index *i* in the following batch.

return_sequences – Boolean parameter set to False by default that determines whether to return the last state in addition to the output [55].

Dropout – The Dropout layer randomly set input units to zero with a frequency of *rate* at each step during training preventing overfitting. Inputs not set to zero are scaled up by $\frac{1}{1-rate}$ so that the sum over all inputs is unchanged [56].

TimeDistributed – A layer wrapper that allows to apply a layer to every temporal slice of an input. The input should be at least 3-dimensional, and the dimension of the index one will be considered to be the temporal dimension [57]. A TimeDistributed(Dense()) applies a same Dense (fully-connected) operation to every timestep of a 3-dimensional tensor [57].

Adam – Optimization algorithm and good default implementation of gradient descent. It automatically uses a custom learning rate for each weight in the model, combining the best properties of AdaGrad and RMSProp. Also, its implementation in Keras uses the best practice initial values for each of the configuration parameters. The AdaGrad algorithm individually adapts the learning rates of all model parameters by scaling them inversely proportional to the square root of the sum of all the historical squared values of the gradient. The parameters with the largest partial derivative of the loss function have a correspondingly rapid decrease in their learning rate, while the parameters with small partial derivative have a relatively small decrease in their learning rate. This results in greater progress in the more gently sloped directions of parameter space. The RMSProp algorithm modifies the AdaGrad to perform better in the nonconvex setting by changing the accumulation of the gradient into an exponentially weighted moving average. It is designed to converge rapidly when applied to a convex function. If applied to nonconvex functions for NN training, the learning trajectory may pass through many different structures to eventually arrive to a region that is locally a convex bowl. AdaGrad shrinks the learning rate according to the entire history of the square gradient and may have made the learning rate too small before arriving at such convex structure. RMSProp uses an exponentially decaying average to discard history from extreme past to allow convergence rapidly after finding a convex bowl, as if it were an instance of the AdaGrad algorithm initialized within that bowl. The name Adam derives from the phrase “adaptive moments” because momentum is

incorporated directly as an estimate of the first-order moment, with exponential weighting, of the gradients [31].

learning rate – It controls how much to update the wights in response to the estimated gradient at the end of each batch [53]. It determines the step size in order to reach the local minimum. For instance, the model
follows the direction of the slope of the surface created by the objective function downhill until reaching a valley [38].

loss – A loss function computes the quantity that a model should seek to minimize during training. In this case, it is a regression loss set to compute the mean of squares of errors (mse) between labels and predictions [58].

metrics – Accuracy metric that calculates how often predictions equal labels. This metric creates local variables *total* and *count* used to compute the frequency with which the prediction matches the label. Then this frequency is returned as a binary calculated by dividing *total*/*count*.

batch_size – Number of samples per gradient update set as an integer or None. Its default value is 32 if not specified.

epochs – Integer specifying the number of epochs to train the model. An epoch is an iteration over the entire X and Y data provided.

validation_freq – If an integer, it specifies how many training epochs to run before a new validation run is performed. Hence, if validation_freq = 1, validation is done every epoch.

shuffle – Boolean that determines whether to shuffle the training data before each epoch. In this case is set to False given that each input (each unperturbed data point) must approach its corresponding label (the corresponding perturbed data point) [59].

This RNN training follows a *many-to-many* model. This type of model produces multiple outputs after receiving multiple values. The internal state is accumulated with each input value before a final output value is produced. In this case multiple time steps are output [59], which is required in this case given that an output is desired for every input given. Most importantly, the *many-to-many* model helps with maintaining the state during training. See figure 5 below.

Note: In figure 35 each rectangle is a vector and arrows represent function (such as matrix multiplication, etc.). Input vectors are in red, output vectors are in blue and green vectors hold the RNN's state. The sequence input and sequence output size match (although they do not have to in this type of model). In this case, we want the predicted solar potential perturbed data for every input data point of the solar potential unperturbed data point [60].

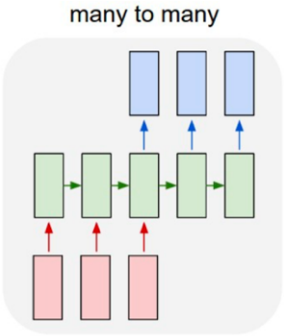


Figure 34. The many-to-many model [60].

4.3.4.3 Calculating the Weights

As shown in section 4.1.1, in every process an LSTM has four layers of the neuron, which together form a processing gate: forget gate → input gate → candidate gate → output gate (with the training sequence following the arrows). The weights in every layer of the RNN model can be extracted via the model.get_weights() function in Keras under its *Layer* class. This function outputs the kernel weights (W) and recurrent kernel weights (U) matrices with its corresponding biases (b) for every LSTM layer. The kernel weights are those that transform the inputs into some other internal values, and they have the shape [*features*, output_dim] where output_dim is the total number of kernel weights. The recurrent kernel weights are those that transform the previous hidden state into another internal value, and they have the shape [*batch*, output_dim] (recall that *batch* = number of units or cells). The biases have the shape [output_dim] [61].

For every time step, the weight and bias for every gate is updated to update the cell state of every cell in the layer. Hence, the LSTM training process outputs four times the number of units per feature for the kernel weights, four times the number of units per unit for the recurrent kernel weights and four times the number or units for the biases. The table below is a representation of the sequential model handling the RNN training with all its components:

1. First LSTM layer with 50 cells and the input layer embedded (that is [5, 3] corresponding to [*timesteps, features*]).
2. Dropout layer regularizer with a dropout rate of 0.46.
3. Second LSTM layer with 10 cells and *timesteps* = 5 preserved.
4. Dense layer with time_distributed layer wrapper with 3 cells and *timesteps* = 5 preserved.

Table 11. Sequential RNN model summary.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 5, 50)             10800
_________________________________________________________________
dropout (Dropout)            (None, 5, 50)             0
_________________________________________________________________
lstm_1 (LSTM)                (None, 5, 10)             2440
_________________________________________________________________
time_distributed (TimeDistri (None, 5, 3)              33
=================================================================
Total params: 13,273
Trainable params: 13,273
Non-trainable params: 0
_________________________________________________________________
```

Table 11 shows the total number of parameters, that is weights plus biases per layer. The first LSTM layer output weight and bias matrices are as follows:

First LSTM layer with 50 cells or units		
Matrix shape	Nature of parameters	Total number of parameters
[3, 200]	Kernel Weights [*features,* 50 units * 4]	600
[50, 200]	Recurrent Kernel Weights [*batch,* 50 units * 4]	1000
[200]	Biases [50 units * 4]	200
	Grand total number of parameters	10800

The second LSTM layer output weight and bias matrices are as follows:

Second LSTM layer with 10 cells or units		
Matrix shape	Nature of parameters	Total number of parameters
[50, 40]	Kernel Weights [input *batch,* 10 units * 4]	2000
[10, 40]	Recurrent Kernel Weights [*batch,* 10 units * 4]	400
[40]	Biases [10 units * 4]	200
	Grand total number of parameters	2440

The fourth layer is a time single ReLu gated distributed wrapped dense layer with output weight and bias matrices as follows:

Time Distributed wrapped Dense layer with 3 cells or units		
Matrix shape	Nature of parameters	Total number of parameters
[10, 3]	ReLu Weights [input *batch,* 3 units]	30
[3]	Biases [units]	3
	Grand total number of parameters	33

The output weights of a trained NN are its training signature and are meant to be implemented in the controls systems engineering of the autonomous space craft. This topic is, however, out of the scope of this paper and will be proposed as future follow up work.

4.3 LSTM Training Results

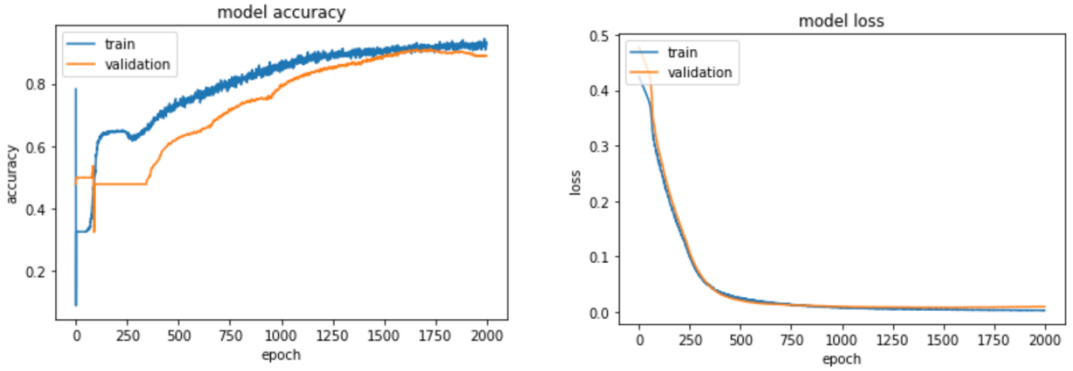
The training of the RNN was done by trial and error using many different combinations of number of LSTM hidden layers (starting with only one) all with one Dense layer for the

output. Trials also involved different variations of number of units within each hidden layer, and batch sizes using both, SGD and Adam optimizers at different initial learning rates and the various hyperparameters of the SGD optimizer. However, Adam optimizer proved to be the optimal optimization tool for this problem. The number of layers that yield the best results was two hidden LSTM layers with a Dense layer to handle the output.

Initially this model was set to be a one-to-one model, which yield poor results given that this kind of model processes from fixed-sized input to fixed-sized output without the interconnection an RNN requires. Hence, a many-to-many model was used in order to preserve state interconnection within the inner layers while yielding an output for every input during training as shown in Figure 35. In the presence of a many-to-many model, the output function for each of the many outputs must be the same function applied to each timestep. The TimeDistributedDense layer was adopted to serve that purpose. It allows for the Dense function to be applied across every output over time serving the need for the same dense function to be applied at every time step [62].

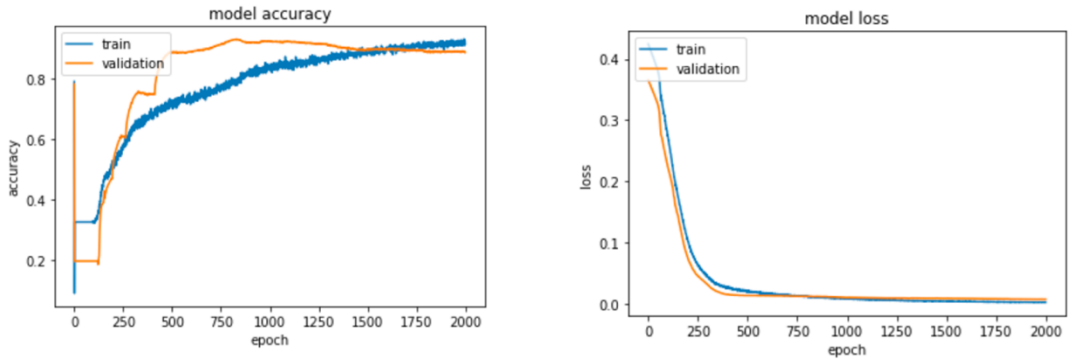
A recurrent problem during training was overfitting. To resolve this problem, a Dropout regularizer was added after the first LSTM. Regularization in machine learning reduces over-fitting by adding a penalty to the loss function to train the model not to learn interdepend sets of feature weights. It ignores (zeroes out) a random fraction, *p*, of nodes and its activations for every hidden layer, each training sample and each iteration. It uses all activations but reduce them by a factor *p* [63].

The training of the RNN was tuned by looking into the accuracy (metric that calculates how often predictions equal labels) and the loss (function computes the quantity that a model should seek to minimize during training) versus number of epochs using the architecture and summary models presented in the last two sections. The results obtained during tuning the RNN training show that they were highly dependent on the number of epochs given. The final tuning was done using numbers of epoch varying between 1000 and 4000 with the best results happening at a number of epochs equal to 2000. The results of this last tunning are presented below.




```
model.compile(Adam(lr=.0002), loss='mse', metrics=['accuracy'])
```

```
Epoch 1996/2000
4/4 - 0s - loss: 0.0030 - accuracy: 0.9302 - val_loss: 0.0095 - val_accuracy: 0.8896
Epoch 1997/2000
4/4 - 0s - loss: 0.0030 - accuracy: 0.9252 - val_loss: 0.0094 - val_accuracy: 0.8896
Epoch 1998/2000
4/4 - 0s - loss: 0.0029 - accuracy: 0.9153 - val_loss: 0.0094 - val_accuracy: 0.8896
Epoch 1999/2000
4/4 - 0s - loss: 0.0027 - accuracy: 0.9342 - val_loss: 0.0095 - val_accuracy: 0.8881
Epoch 2000/2000
4/4 - 0s - loss: 0.0030 - accuracy: 0.9272 - val_loss: 0.0095 - val_accuracy: 0.8886
```




```
model.compile(Adam(lr=.0002), loss='mse', metrics=['accuracy'])
```

```
4/4 - 0s - loss: 0.0027 - accuracy: 0.9158 - val_loss: 0.0076 - val_accuracy: 0.8901
Epoch 1996/2000
4/4 - 0s - loss: 0.0030 - accuracy: 0.9119 - val_loss: 0.0075 - val_accuracy: 0.8891
Epoch 1997/2000
4/4 - 0s - loss: 0.0028 - accuracy: 0.9188 - val_loss: 0.0075 - val_accuracy: 0.8891
Epoch 1998/2000
4/4 - 0s - loss: 0.0027 - accuracy: 0.9292 - val_loss: 0.0075 - val_accuracy: 0.8896
Epoch 1999/2000
4/4 - 0s - loss: 0.0027 - accuracy: 0.9248 - val_loss: 0.0076 - val_accuracy: 0.8906
Epoch 2000/2000
4/4 - 0s - loss: 0.0028 - accuracy: 0.9163 - val_loss: 0.0076 - val_accuracy: 0.8891
```

Figure 35. Training and validation accuracy and loss results.

It is expected to see the training accuracy curve surpass that of the validation as training goes on. This is because the RNN is expected to learn over the training data better than the validation data. The learning process is expected to behave in a way that both curves start from near zero and run close together near the end with the training curve on top. Although the latter shows to be happening in the results presented above, the former does not, since the validation accuracy curve starts at a very high value. The reason for this is due to the training data being close to equal to the label data in the first part of the spacecraft elliptical trajectory, where solar perturbations are almost null on the spacecraft since it is much closer to Venus over that section of its path.

On the other hand, the training and validation loss curves show the expected behavior with the training loss running lower than the validation loss in the final phase of the training. The results are boxed on the training history shown for both sets of plots. Notice that the results of this set of trainings do not present the same exact results given that the kernel weights of the training are randomly set every time a new training process begins. Hence, no one training process starts identical to another. Also, reproducibility problems with identical RNN training set ups were encountered using google chrome. These issues were resolved by resetting the runtime in Google Collab, clearing Google Chrome history cookies and other site data, cached images and files and refreshing the Google Colab page.

Even though a significant step forward has been made on the training of this LSTM RNN model, there is a lot more work to be done and different efficient strategies that could be applied to increase the accuracy to an ideal value of close to 98%, which is presently not known to be possible for a case that involves the nature of the data used in this problem. The next steps, time

permitting, are to feed more data to the RNN for training and another Dropout layer after the second hidden LSTM layer to prevent overfitting. The following plots are presented to show an example of overfitting and its dependency on number of epochs for training.

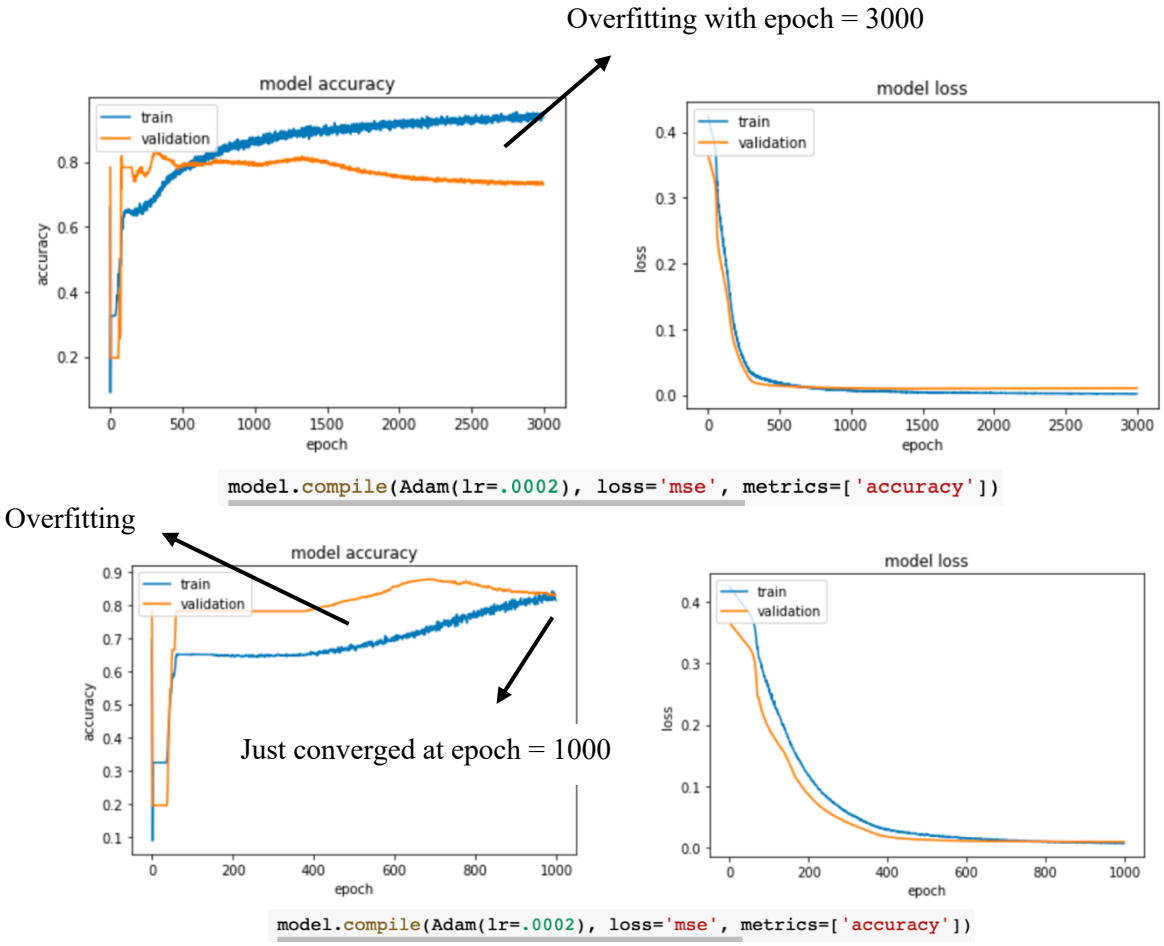


Figure 36. Examples of overfitting given different number epochs as a limit.

It is important to research about whether overfitting being dependent on the number of epochs for training compromises the validation of the training results of an LSTM RNN model or if it is an expected behavior of machine learning development. In any case, using a second Dropout layer and, perhaps, a third LSTM hidden layer with more data to train the RNN would be worth trying as a next step in this project.

Chapter 5: Solar Gravitational Potential Perturbation Analysis

In this chapter, an analysis of the solar gravity potential perturbation on the spacecraft in the closed orbit has been done. The motivations for this analysis are: 1) to learn how to take advantage of such perturbations to approach Venus in the most fuel-efficient way possible, and 2) the possibility of training the LSTM for autonomous burning at strategic points of phase space around Venus using the approach of this analysis. Also, an alternative RNN approach is provided given that the weights of and RNN LSTM cannot provide the user with a one-to-one correspondence between output weights and output samples, which would represent the solar perturbations per vector component of the position and velocity of the spacecraft in close orbit.

Depending on the time available for a mission to approach the planet for either re-entry or stationary orbiting, the approach to this problem can be done differently. For example, the spacecraft could be permitted to drift towards the Sun as it orbits Venus, or small burns could be done at strategic solar perturbation points to try to approach the planet faster or circularize its orbit while letting it drift towards the Sun to approach the planet. A recommended first step before burn analysis has been initiated where the spacecraft was let to drift towards the Sun with no burn at all with a minimum radius of approach at perigee was given as a limit. However, this approach leaves the spacecraft in a highly eccentric elliptical orbit, which requires a burn or several small burns to be circularized. Nevertheless, there is the possibility of using strategic solar perturbation points for these burns at several passes of the spacecraft around the planet to save fuel. The next section shows a plausible approach to start building a grid of potential perturbation points to aid space travel by making a burn analysis with as many orbits as possible to cover phase space around Venus. The burn analysis presented next is only done for few orbits as a proof of concept.

5.1 Venus Approach by Solar Gravitational Potential Perturbation Drift

The solar gravitational potential influence on the spacecraft is highly dependent on Venus position with respect to the Sun. Therefore, it is advisable to first study the ideal initial position of the spacecraft as it enters close orbit around Venus. In the case of the Venera D mission, the orbital parameters of its path around the planet might be favorable to take advantage of the solar perturbation to approach the planet for re-entry but it might not be as much for a low orbit around it because it keeps its high ellipticity. In this analysis the path of the spacecraft was simulated to be left drifting towards the solar potential perturbation using GMAT with a limit radius of perigee of 6400 km. This limit was calculated by adding the radius of Venus to the height of its atmosphere. For instance, the mesosphere of Venus extends beyond 100 km of altitude [64]. To ensure the safety of the spacecraft as it comes the closest possible to Venus, the height of its atmosphere was taken to be 300 km. The calculation is shown below:

$$r_{limit} = r_{venus} + h_{atmosphere\ venus} = 6051.8\ \text{km} + 300\ \text{km} = 6351.8\ \text{km} \approx 6400\ \text{km}$$

Different kinds of orbits were simulated at two different epochs to investigate two different positions at radius of perigee with respect to the Sun. The Venera D orbit that has been investigated so far in this report, was done twice by varying its eccentricity, inclination, right

ascension, and argument of perigee in close orbit with Venus to place the spacecraft at a different position with respect to the Sun. The third orbit was done at a different epoch in 2021, and its semi-major axis an ellipticity were changed as well. Other orbit variations were explored but the most relevant ones pertinent to the findings of this section will be presented.

Recall the orbital elements presented in chapter 2 based on the Venera D mission at a larger semi-major axis:

- a = Semi-major axis = 49,448.744
- e = Eccentricity = 0.8339
- i = inclination = 90°
- ω = argument of perigee = 100.0° (chosen from allowable range)
- Ω = longitude of the ascending node = 98.7887°
- ν = mean anomaly = 0° (at radius of pericenter).

The epoch for the simulation using these orbital parameters is that of the Venera D mission as well. The setup of the epoch and orbital parameter in GMAT, as well as a visualization of it, is shown in the figures below.

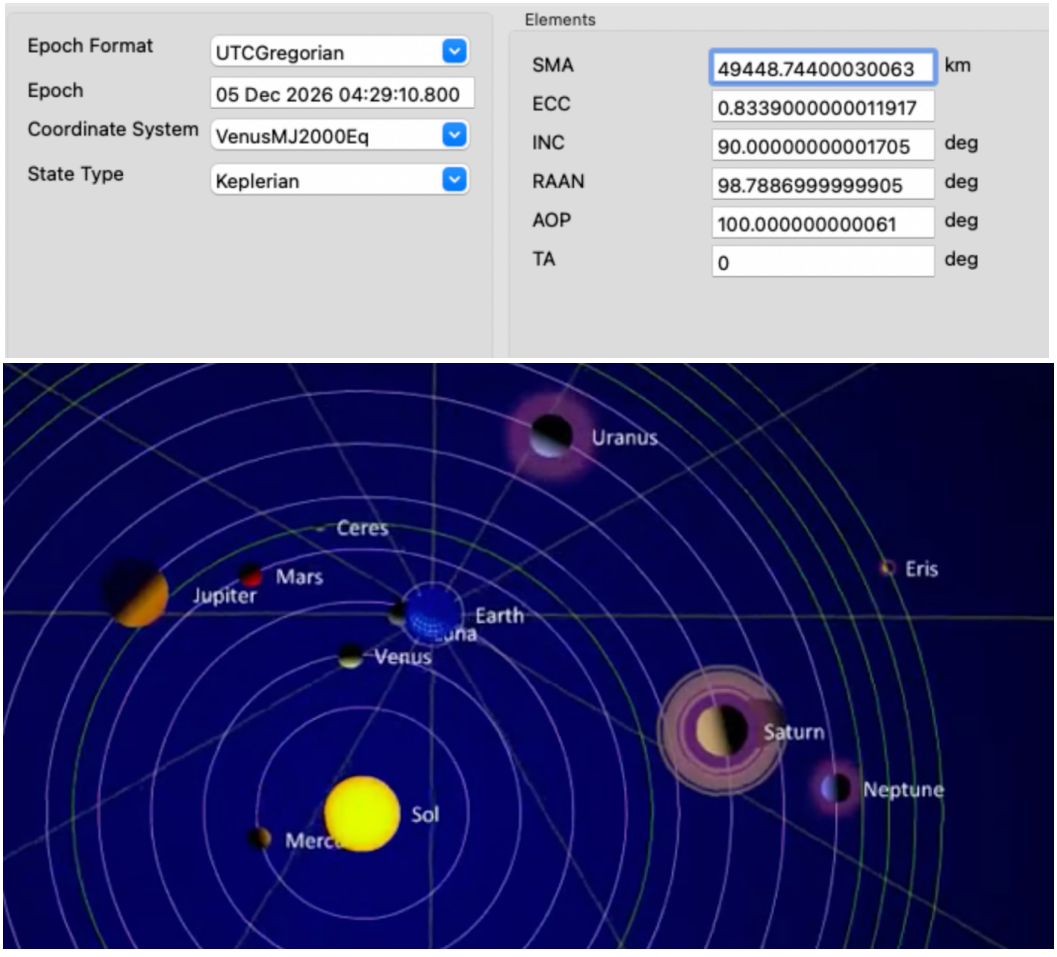


Figure 37. Orbital parameters and epoch in GMAT and position of Venus in the Solar System at the epoch of the Venera D mission [65].

Figure 37 shows the orbital parameters mentioned and the epoch setup for the simulation in GMAT, as well as the position of Venus with respect to the Sun in the solar system at that epoch using the online propagator in [65]. The latter is important to observe to understand how different epochs have an influence on the drifting of the spacecraft.

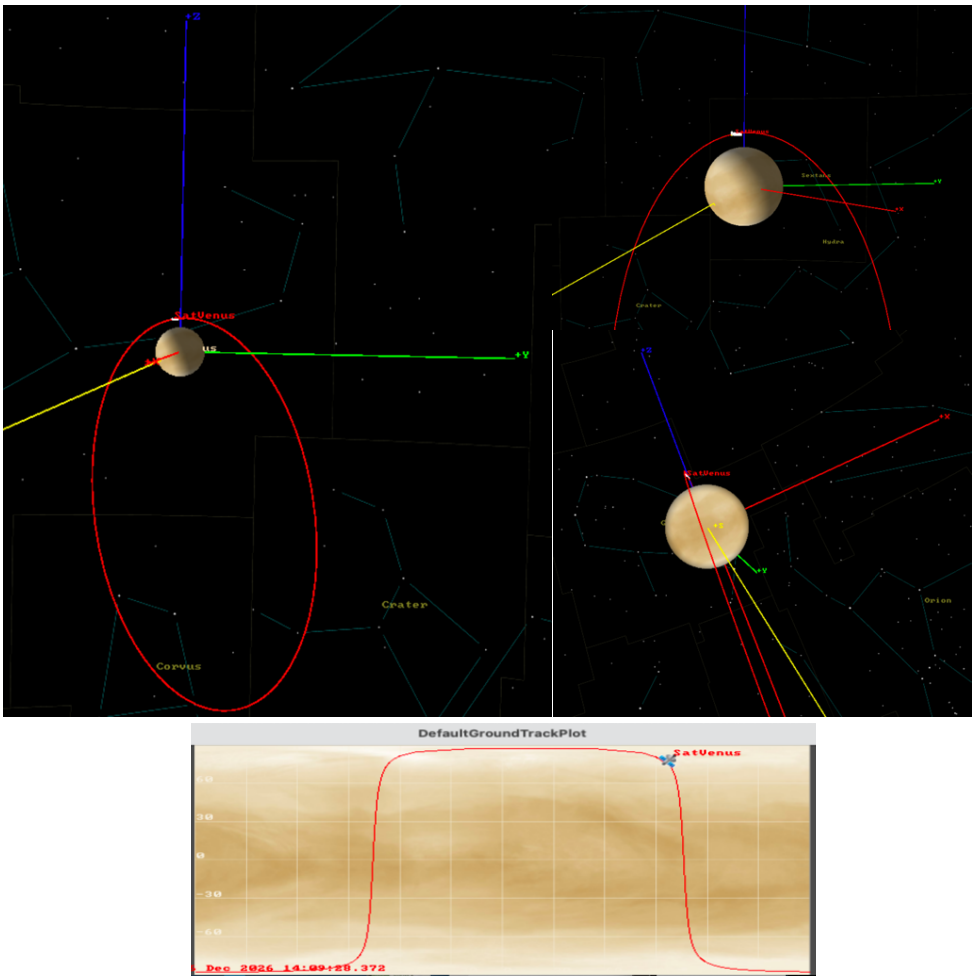


Figure 38. Position of the spacecraft with respect to the Sun in a Venera D-like closed orbit around Venus.

Note: A full view of the orbit of the spacecraft around Venus is shown on the left. On the right side a closer look on the upper right and a front view of the Sun line (shown in yellow) with respect of the spacecraft's orbit are shown. The plot below represents the path of the spacecraft as perceived on Venus' ground.

Figure 38 presents a visualization of how the spacecraft is positioned with respect to the Sun with these orbital parameters and its ground track on Venus surface.

Using the *Mission Tree* feature in GMAT, the spacecraft was simulated to be left drifting towards the perturbation solar gravitational potential until it reached the limit radius of perigee before mentioned. the *Mission Tree* feature in GMAT is an ordered, hierarchical display of the command mission sequence in the GMAT script created [9].

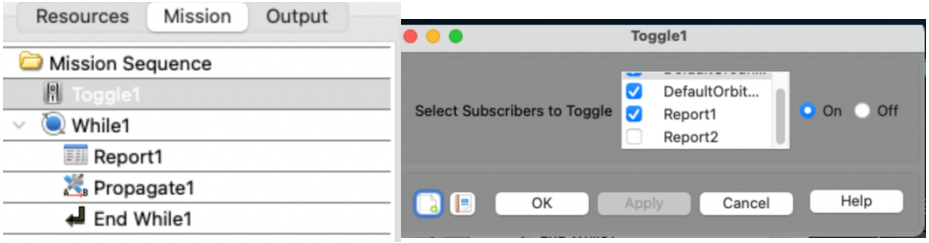


Figure 39. Mission Tree with the command mission sequence used and the Toggle function.

Figure 39 shows the mission tree created for the spacecraft solar perturbation drift simulated in the analysis of this type of orbit. The *Toggle* function allows turning on and off the collection of data output. A while loop was used to keep the spacecraft propagating on the closed orbit until the limit radius of perigee was reached. Within the loop, the data was recorded in *Report 1* every pass at radius of perigee.

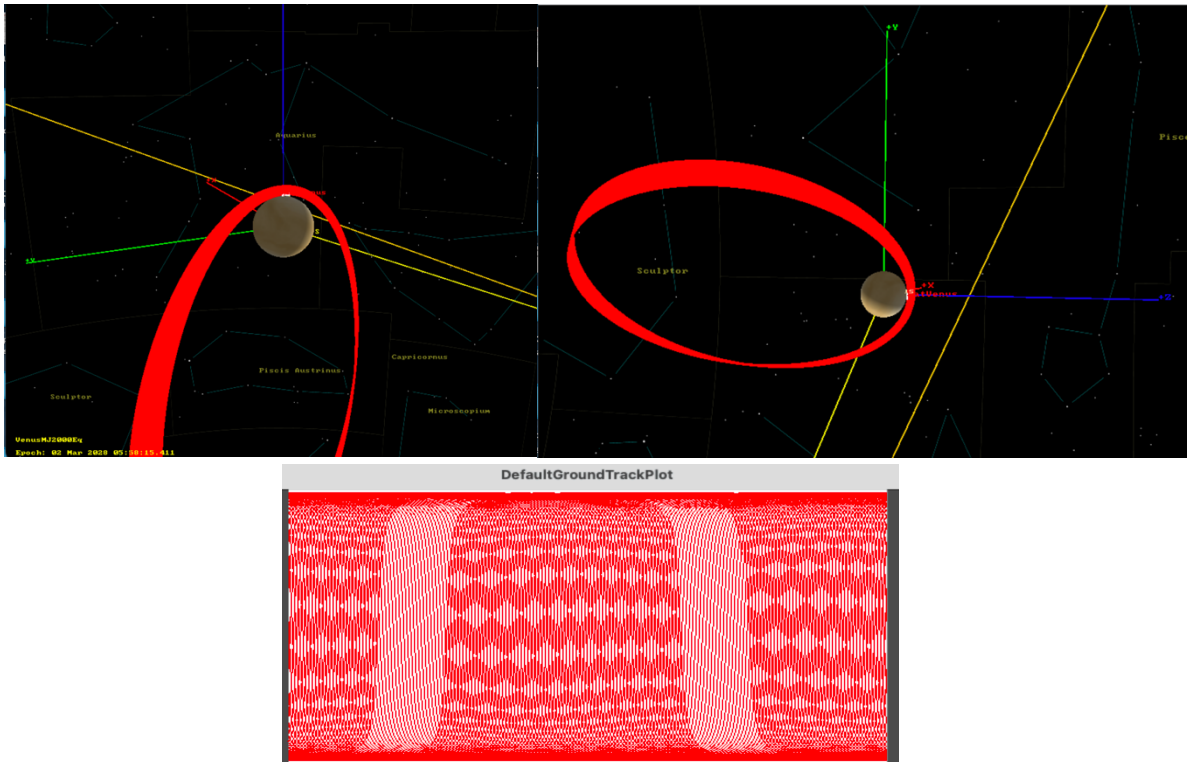


Figure 40. Spacecraft approach to Venus after drifting towards the solar potential perturbation for 451.65 days.

Note: The propagation of the spacecraft is shown from different perspectives and at the end of the 451.65- day period when the spacecraft reaches the limit radius of pericenter. The yellow line represents the path of the Sun as seen from Venus during that period. The plot below shows the path of the spacecraft on the ground of Venus.

Figure 40 shows the simulation done in GMAT starting with a radius of pericenter of about 8213.4 km and ellipticity of 0.8339. The spacecraft took 451.65 days to reach a radius of perigee of approximately 6403.3 km with a final eccentricity of about 0.87.

Although the spacecraft successfully reaches Venus at the desired distance of approach free of burns, there are two caveats to this approach depending on the goals of the mission under consideration: 1) the ellipticity of the spacecraft's closed orbit got more pronounced; 2) depending on the mission's time constraints, the time of approach might be too long. In the case where a close, circular orbit around the planet is desired or time constraints to complete the mission are present, burns for circularization and/or faster approach will be necessary.

In an effort to simulate a planet approach while circularizing the orbit free of burns, other simulations were done with the spacecraft left drifting towards the solar potential perturbation. The set up for the next orbit analysis in GMAT and a visualization of it is shown below.

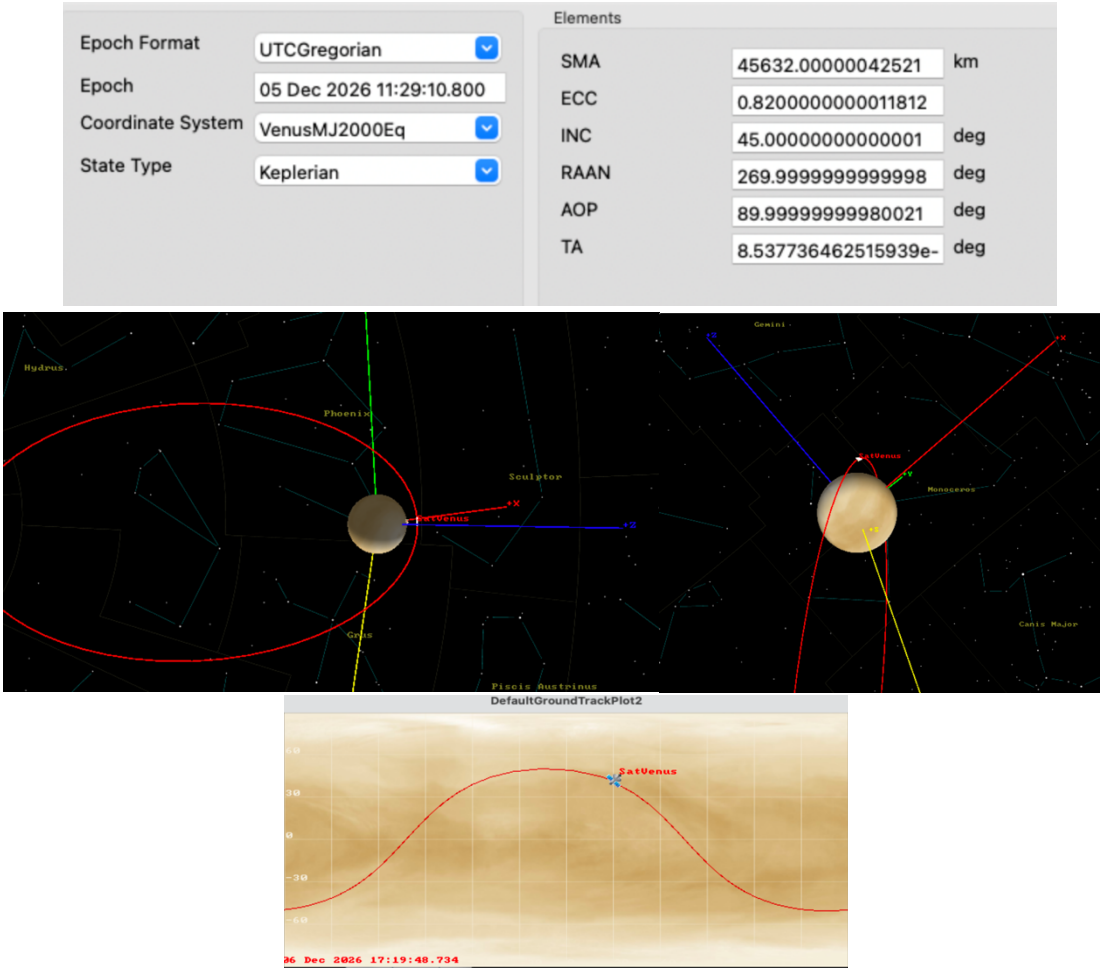


Figure 41. Orbit view of the set up for the second simulation.

Note: The simulation done for this orbit was done with the same epoch as the Venera D mission but different orbital elements. The track plot shown below represents the path of the spacecraft as seen on the ground of Venus

Figure 41 shows the simulation of a closed orbit at the same epoch but different position of the spacecraft with respect to the Sun. The orbital parameters are shown in the top section of the figure. In this simulation the semi-major axis, eccentricity and inclination were reduced (with the semi-major axis and eccentricity corresponding to those of the Venera 16 mission) and the right ascension of ascending node (RAAN) or longitude of the ascending node (denoted by Ω in this report) were increased to place the spacecraft's orbit farther from the Sun line (shown in yellow). These are the orbital parameter used in this simulation:

- a = Semi-major axis = 45,632 km
- e = Eccentricity = 0.82
- i = inclination = 45°
- ω = argument of perigee = 90.0°
- Ω = longitude of the ascending node = 270°
- ν = mean anomaly = 0° (at radius of pericenter). GMAT readjusted this parameter as closed as zero as possible with it being approximately 8.47 x 10^{-7}. This kind of readjustment of parameters happened often during the simulations done for this project. The reason for it is still uncertain but it could be that the propagator tries to set up the initial state vector such that it would accomplish the accuracy given. As it can be seen on the top part of figures 37 and 41 where the epoch, level of accuracy and initial state vector are set up for propagation, all orbital parameters are readjusted to very closed values to the ones entered (shown above). In this simulation the spacecraft was propagated for about 4.8 years starting at a radius of pericenter of 8,213.76 km and eccentricity of 0.82 to a final radius or pericenter of about 15,972 km and eccentricity of approximately 0.65.

Figure 42. Orbit view and track plot of the second simulation done after running over close to 4.8 years.

Note: The orbital parameters of this simulation match the semi-major axis and eccentricity of the Venera 16 mission. The figure shows how the spacecraft goes farther from the planet as it circularizes. The track plot shown below shows the path of the spacecraft as seen on the ground of Venus after going around the planet many times.

The results of letting this simulation run for close to 4.8 years are shown in figure 42. The eccentricity of the spacecraft's orbit around Venus is successfully decreased by the solar potential perturbation but it is taken farther away from Venus as the orbit's eccentricity decreases. After approximately 453 days, the radius of pericenter increases from 8,213.76 km to about 9,948.20 km and its eccentricity decreased from a 0.82 to about 0.78. A circular close stationary orbit around the planet will require at least one burn in this case.

Compared to the Venera D-like mission, the next simulation was done at a different epoch (that of the Venera 16 mission, July 22, 2021), as well as a larger semi-major axis, eccentricity, and longitude of the ascending node, and a smaller inclination and argument of perigee. As can be noticed, the argument of perigee and longitude of the ascending node were left the same as the second simulation. The orbital parameters are as follows:

- a = Semi-major axis = 70,000.8 km
- e = Eccentricity = 0.86
- i = inclination = 30°
- ω = argument of perigee = 90.0°

- Ω = longitude of the ascending node = 270°
- ν = mean anomaly = 0° (at radius of pericenter)

The epoch and setup of these orbital parameters and a visualization of Venus with respect to the Sun in the solar system are shown in the figure below.

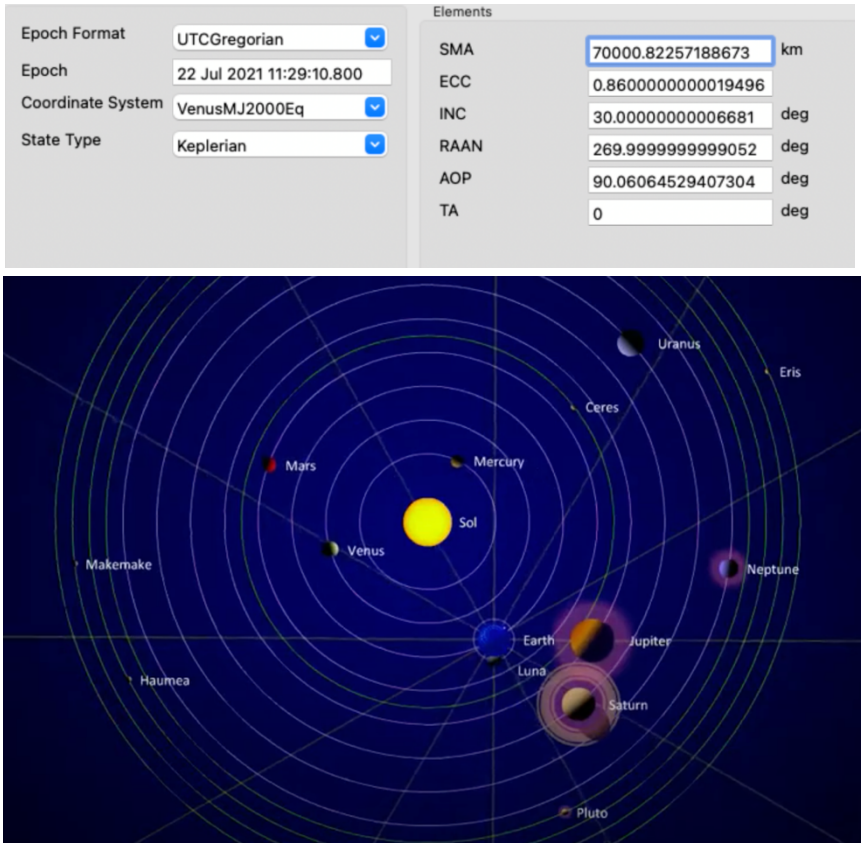


Figure 43. Orbital parameters and epoch in GMAT and position of Venus in the Solar System at the epoch of the Venera 16 mission [65].

Figure 43 shows the orbital parameters and epoch above mentioned setup in GMAT's propagator for simulation, as well as a visualization of the position of Venus with respect to the Sun in the solar system at that epoch using the online propagator in [65].

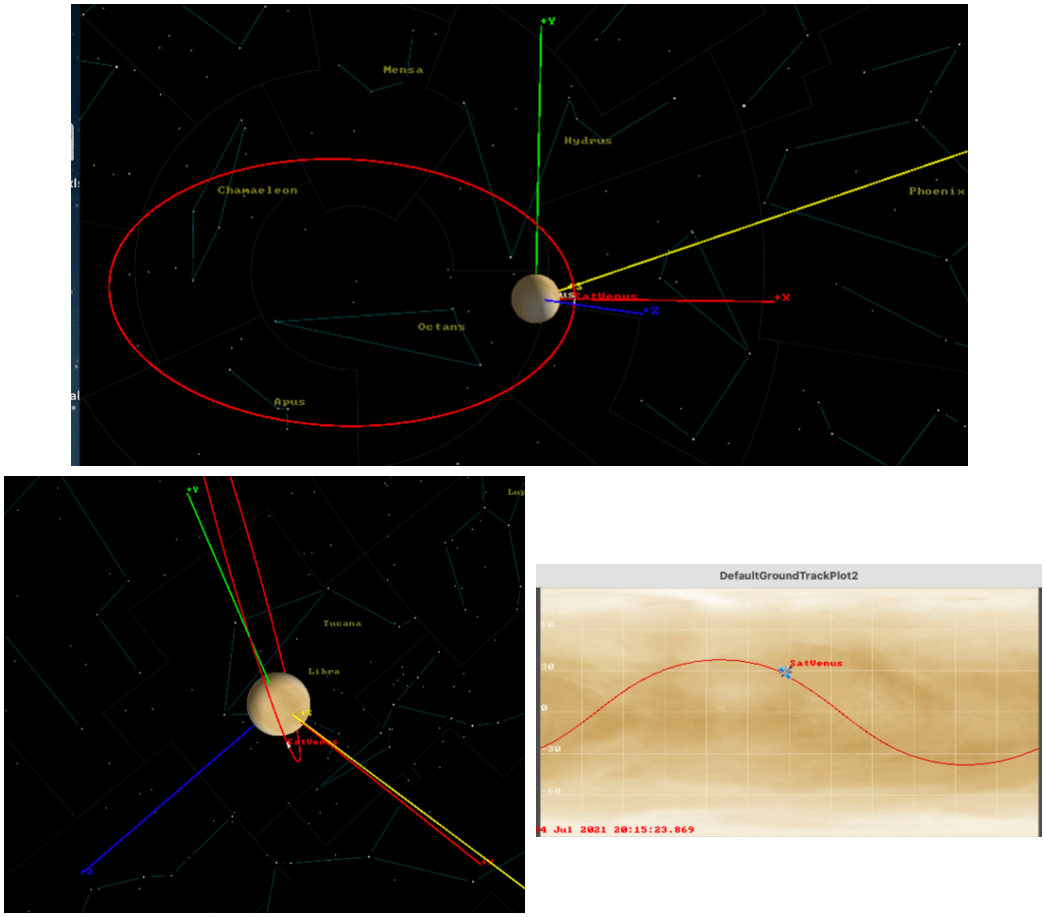


Figure 44. Orbit view of the set up for the third simulation.

Note: This orbit simulation was done with the same epoch as the Venera 16 mission but different orbital elements. The yellow line represents the location of the Sun. The track plot shown below represents the path of the spacecraft as seen on the ground of Venus.

The propagation of the spacecraft for this simulation corresponds to approximately five years starting at a radius of pericenter of 8213.4 km and eccentricity of 0.86 and ending at a radius of pericenter of approximately 13,924.8 km in and eccentricity of 0.57730723 at the end of the five-year period.

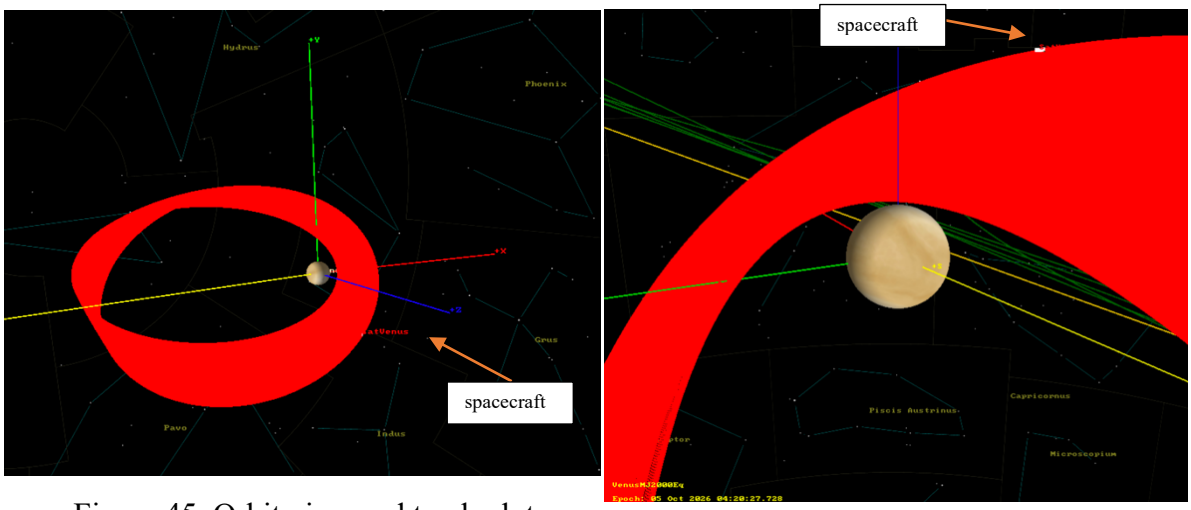


Figure 45. Orbit view and track plot of the second simulation

Note: The orbital parameters of this simulation match the epoch of the Venera 16 mission. The figure shows how the spacecraft goes farther from the planet as it circularizes. The track plot shown below shows the path of the spacecraft as seen on the ground of Venus after going around the planet many times.

The results after letting this simulation run for an approximate 5-year propagation period are shown in figure 45. Like the previous simulation, the eccentricity of the spacecraft's orbit around Venus is successfully decreased by the solar potential perturbation. However, it is taken farther away from Venus as the orbit's eccentricity decreases even more rapidly than the previous simulation. After approximately 453 days, the radius of pericenter increased from 8,213.76 km to about 13,924.8 km and its eccentricity decreased from a 0.86 to about 0.80. A circular close stationary orbit around the planet with these initial conditions will also require at least one burn.

The simulations presented in this section show the relevance of doing a deeper study on how the initial state vector and epoch of a closed orbit around Venus (and most probably around other planets close to the Sun) can better serve the purpose of taking advantage of the solar gravitational potential perturbation depending on the goals of a mission. In the case of the Venera D orbit, the solar perturbation drift carried the spacecraft to the outskirts Venus' atmosphere in over a year but at the cost of a higher eccentricity. For the other two simulations the spacecraft went farther away from Venus as its orbit got increasingly circular. For a close, circular orbit around the planet, perhaps an ideal initial state vector at the right epoch could help the spacecraft drift towards the planet as the eccentricity of its orbit decreases or, at least, increases very slowly. Those conditions could be convenient to save fuel at it is burned at strategic solar perturbation points. A basic solar perturbation study with suggestions on how to

proceed with a deeper investigation of a solar gravitational perturbation vector field on the spacecraft during a full period will be presented in the next section.

5.2 Solar Gravitational Potential Perturbation Vector Field

A vector field grid for the perturbed position and velocity of the spacecraft at every point of a close orbit for many different orbits, such that most of phase space around Venus can be covered, can be developed by either 1) extracting the output weights from the training of an RNN or 2) by doing and analysis of the solar perturbation on the spacecraft at every step of its path throughout many different orbits around Venus. An approach on how to explore these two possibilities are presented in this section.

One of the goals of this project is to analyze the weights and biases resulting from the training of an LSTM to correct the state vectors (position and velocity) at every point of a closed orbit around Venus for solar gravitational potential perturbations. It is expected for these weights to represent the solar perturbations of phase space of the spacecraft in closed orbit. In order to do so, there must be a one-to-one correspondence between weight and bias to the corrected element of every featured or output of the trained LSTM. Recall that the features of the LSTM presented in this investigation represent the three components of the spacecraft's position and velocity at every step of its path in orbit around Venus. Hence, every feature element will be *x, y* and *z* and v_y, v_x and v_z, coming into the LSTM as perturbed data (the input) to be matched to the "correct" unperturbed data (the target or label). The output of the LSTM is the input value corrected to match the label data with its weights and biases. However, as it was presented in table 11 in chapter 4, the weights and biases in an LSTM do not have a one-to-one correspondence with the output data but depend on the number of LSTM cells and gates of the LSTM cell. If we look at the chart presented below with a summarized architecture of the LSTM, we can see that there are two weights and one bias per gate, which makes eight weights and four biases per cell, since there are four gates in every LSTM cell.

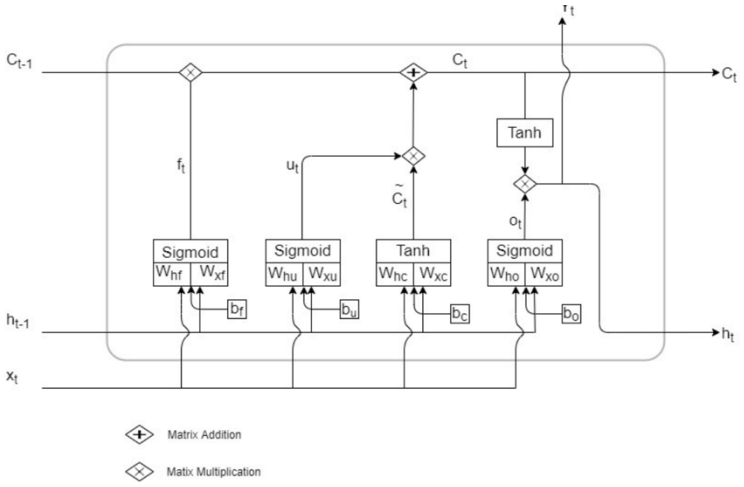


Figure 46. A summary of LSTM architecture [66].

Figure 46 shows the two weights (one from the hidden layer denoted by h and one from the input denoted by x) and bias per gate in a single LSTM unit. The LSTM used for this investigation presented in chapter 4 yields 33 weights and biases (10 weights and 1 bias per cell for 3 output cells) for the 2020 elements per feature used to train it. Detangling the weights corresponding to every element of the output is quite complex. Understanding how an NN calculates weights, and their disentanglement are presently topics under investigation. The code in Google Colab used to train the RNN LSTM presented in this paper provided in appendix A shows the output weights per layer and a visualization of them and its biases. However, they were not discussed in this report because they are not directly related to the physical implication of the solar perturbation study done in this project but rather a representation of the numerical methods use by the RNN LSTM to correct for them.

A way to detangle the weights per element output of an NN are using *autoencoders*. Autoencoders are NNs which goal is to reconstruct its input dataset, that is, to learn to copy inputs to its outputs. Autoencoders are used mainly for denoising, reduction of dimensionality, pretraining and generating data [67].

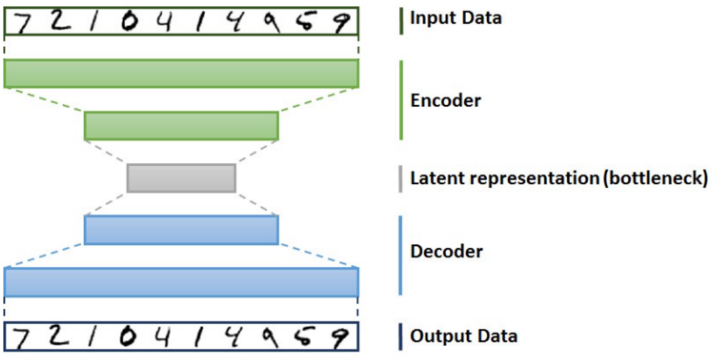


Figure 47. Composition of an autoencoder [67].

An autoencoder is composed of an encoder used to convert input data into a latent representation (a bottleneck layer), and a decoder used to convert the latent representation into outputs (reconstruction). The encoder and decoder might comprise many neuron layers while the latent representation is usually only one layer. The overall architecture is similar to the multilayer perceptron, with the particularity that the output layer size must match the one of the input layers. The latent representation layer size determines how much information an autoencoder can keep, which is usually significantly smaller than the input data. Restricting its size forces the autoencoder to find patterns in the inputs and eliminate irrelevant features [67].

Autoencoders with tied weights have decoders weights that are the transposed of the encoder weights. This is a way to reducing the number of parameters in the model while sharing them. Advantages of tying weights include the risk reduction of overfitting and increased of training speed [67].

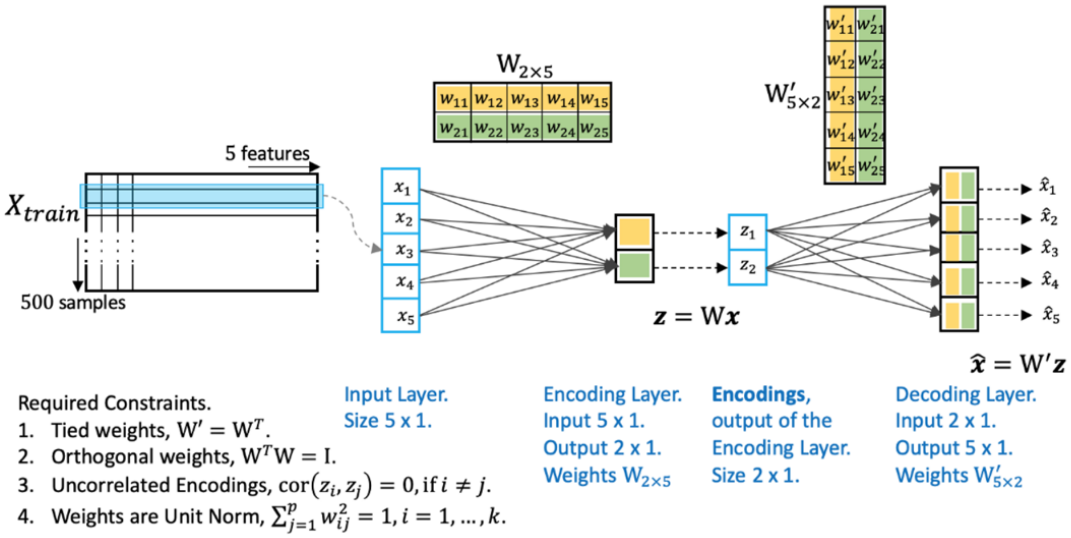
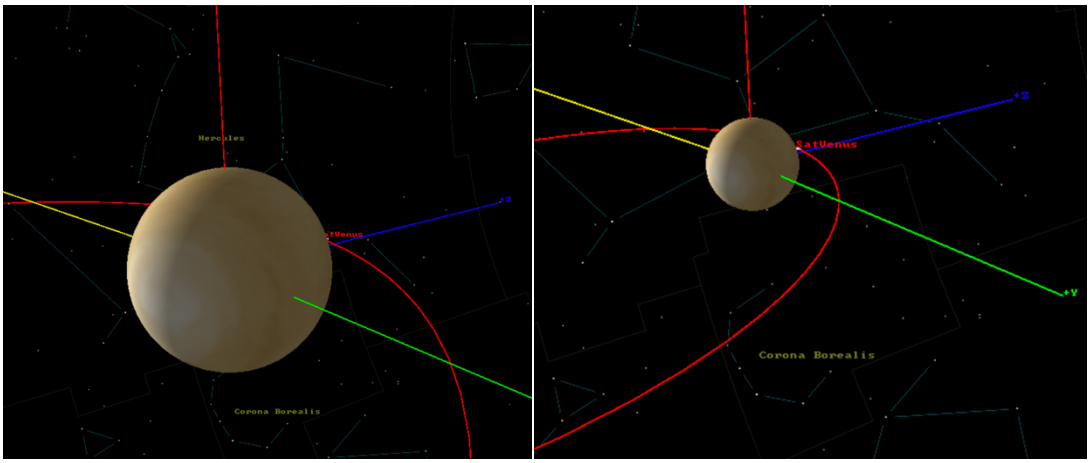


Figure 48. Example of an autoencoder with 5 features and 500 samples [68].

Figure 48 shows an example of an autoencoder with five features and 500 samples. The constraints are shown on the left bottom corner where the encoder takes the transpose of the weights to bottleneck the information and the decoder outputs a copy of the input with its weights and biases per element. This way, autoencoders pose a much more efficient way to detangle the weights and biases corresponding to every element of an RNN output rather than spending much men and computing power trying to decodified weights per element using a mathematical approach.

Another approach to analyze the solar perturbation on the spacecraft throughout its orbit around Venus is by creating a gravitational potential vector field using GMAT data from the propagation of the spacecraft with and without the solar gravitational potential. This vector field is done by taking the difference between perturbed and unperturbed data points per unit time from the spacecraft's propagation in the direction of the perturbed orbit.



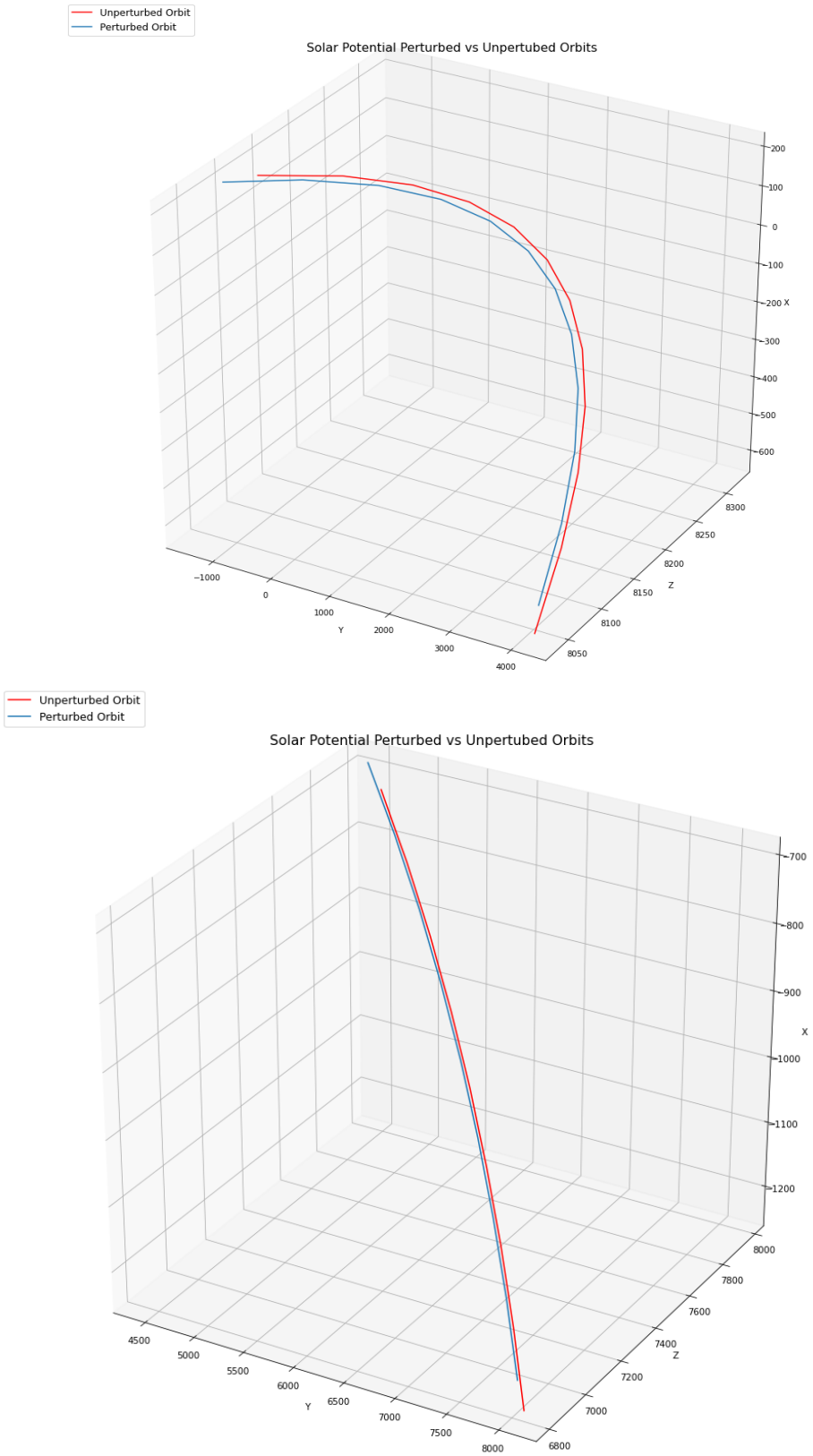


Figure 49. Perturbed and unperturbed orbits closed to radius of pericenter.

A comparison between the perturbed and unperturbed orbits is shown in figure 49. The top image is the plot generated by the GMAT for the spacecraft's propagation with solar potential perturbation. It is shown to better visualize the 3D plots of the perturbed and unperturbed orbits and where the Sun line with respect to them is. The middle plot represents the last 12 steps (with a rate of step/10 min) of the orbit's period before reaching radius of pericenter. The bottom plot is that of the last 9 steps, with the same rate, before the last 12 steps. The curbs in the bottom plot are very closed together because the solar potential perturbation is very small in comparison to the large distances covered by the spacecraft's orbit. The last steps of the orbit's period are shown for convenience, since the perturbation is larger and easier to see in that section of the orbit. These plots are consistent with the perturbation vector field calculated using GMAT propagation data and presented in the figure below.

Note: The arrows in this plot represent the displacement, in kilometers, the spacecraft drifted away from its unperturbed path. The dot on the left lower corner represents the scale of the arrows, which is 10 km pet dot. The axis of the vector plot is z versus y according to GMAT data. The solar perturbation on x is represented by the vector color scheme, which is lightest for the smallest (negative mid 30s) values and darkest for the largest (in the 70s) values.

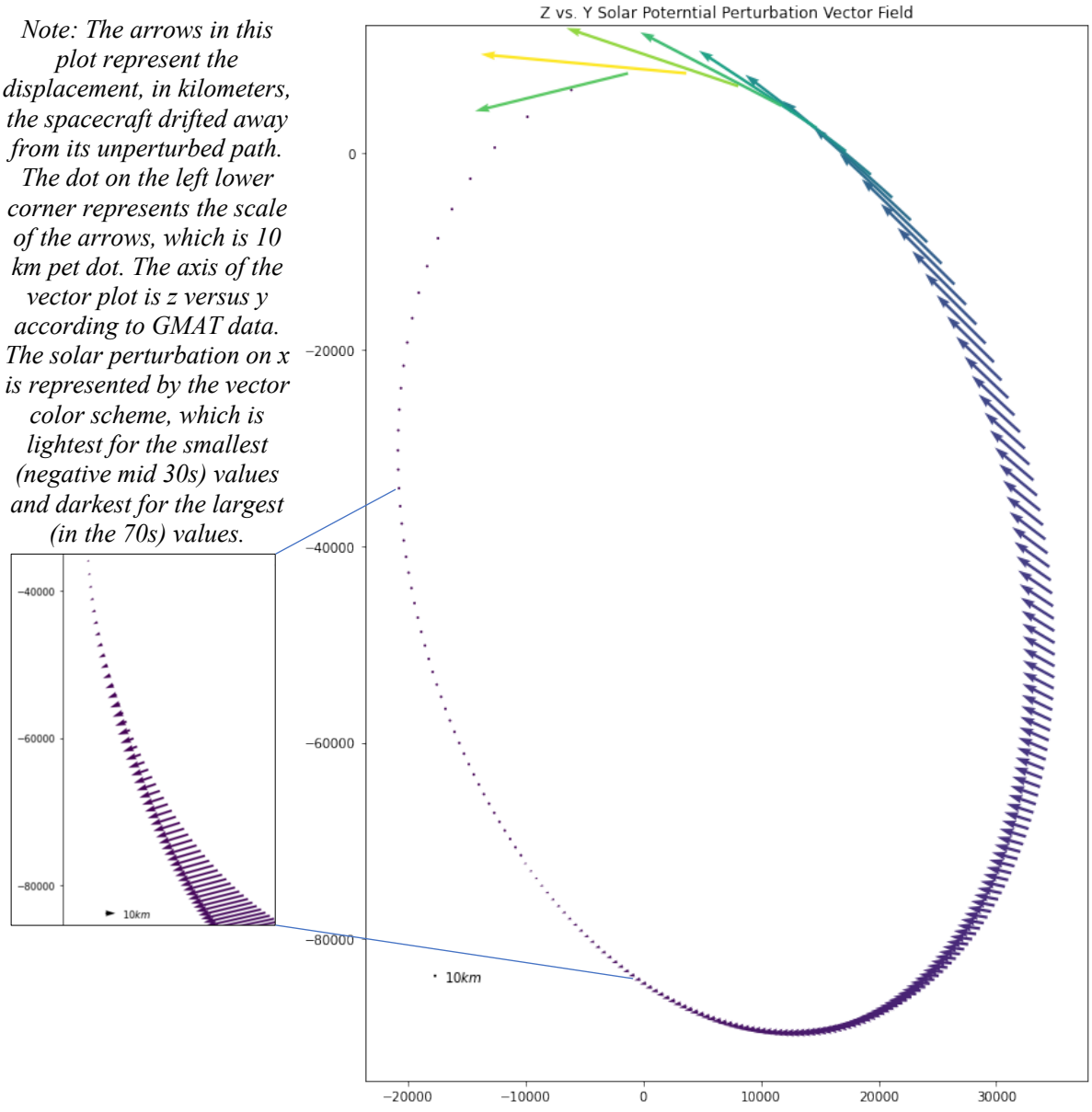


Figure 50. Solar gravitational potential vector field at a rate of one step every 10 minutes.

Figure 50 shows the results of the calculation of the solar perturbation vectors for a step every 10 minutes of the spacecraft's path. This plot is useful to visualize how the spacecraft is being acted upon by the Sun's and Venus' solar perturbation and confirm that it is indeed being pulled more towards the Sun at large distances away from the planet. A burn analysis has been done on five different regions on the vector field where the perturbations are larger including at perturbed radius of pericenter to be compared with the burn at the unperturbed radius of pericenter, which is the common technique for orbit circularization. This burn analysis is focused more on solving for circularizing the orbit, since it was already proven that the solar perturbation does aid with re-entry. The four areas of investigation other than at radius of pericenter are approximately shown in the following figure.

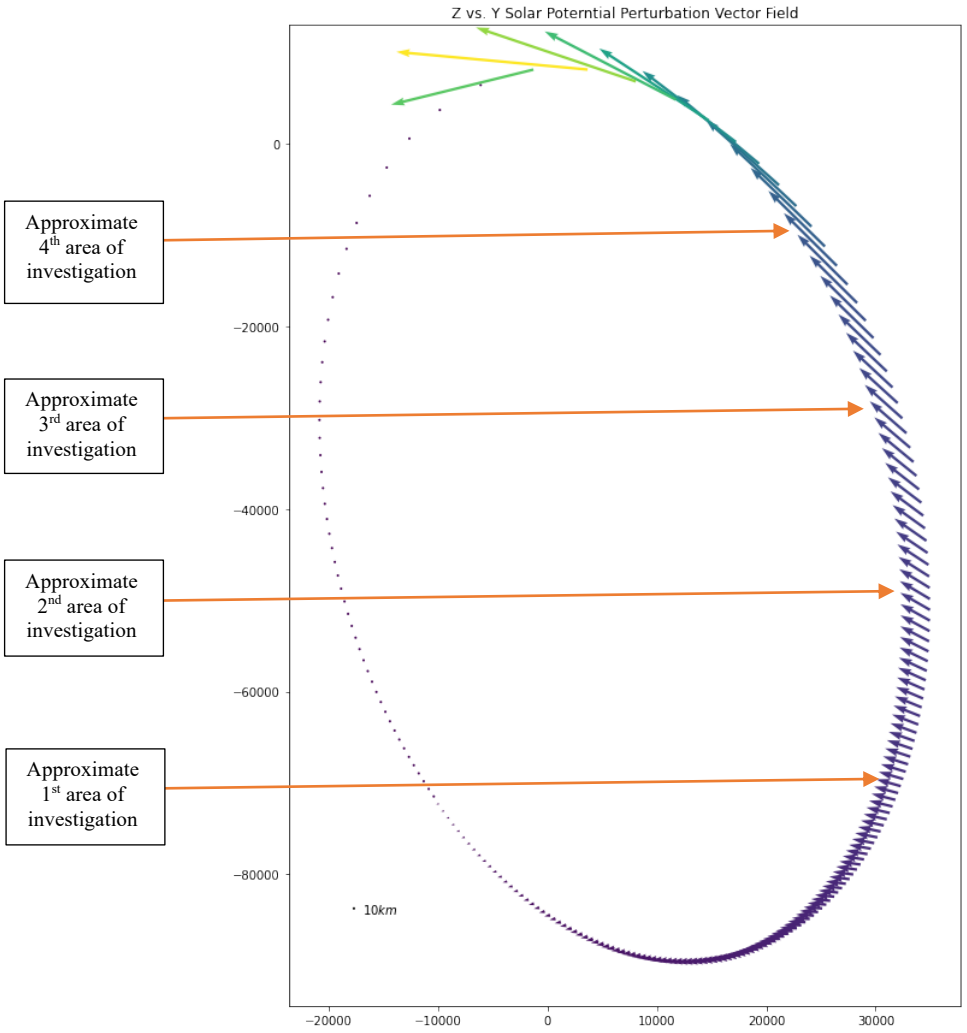


Figure 51. Approximate location of the four areas of investigation for burn analysis other than at radius of pericenter.

Figure 51 shows approximately where the areas that will be investigated for burn analysis are on the orbit's perturbed path. It is the perturbed data that is being investigated because it is to be compared to burning at the unperturbed (or the path corrected for perturbations) point at radius of pericenter, proven to be one of the best locations to burn for orbit circularization. The results for the burn analysis are presented in the next section.

5.3 Solar Gravitational Potential Perturbation Vector Field Burn Analysis Results

Three different burn analyses will be presented in this section. The first one will aim to achieve an eccentricity of zero with a tolerance of 0.1, which is the value that reach convergence to the closest value of eccentricity of zero. The second analysis will have as a goal to reach both an eccentricity of zero and the smallest possible radius of pericenter. The third analysis will be for a planet approach for re-entry having as a goal only to achieve the smallest radius of pericenter possible with the before mentioned limit of 6400 km, which is where the outskirt of Venus' atmosphere is considered to be for this research.

The simulation done for the burn analysis also uses the *Mission Tree* feature in GMAT, where a mission sequence is created.

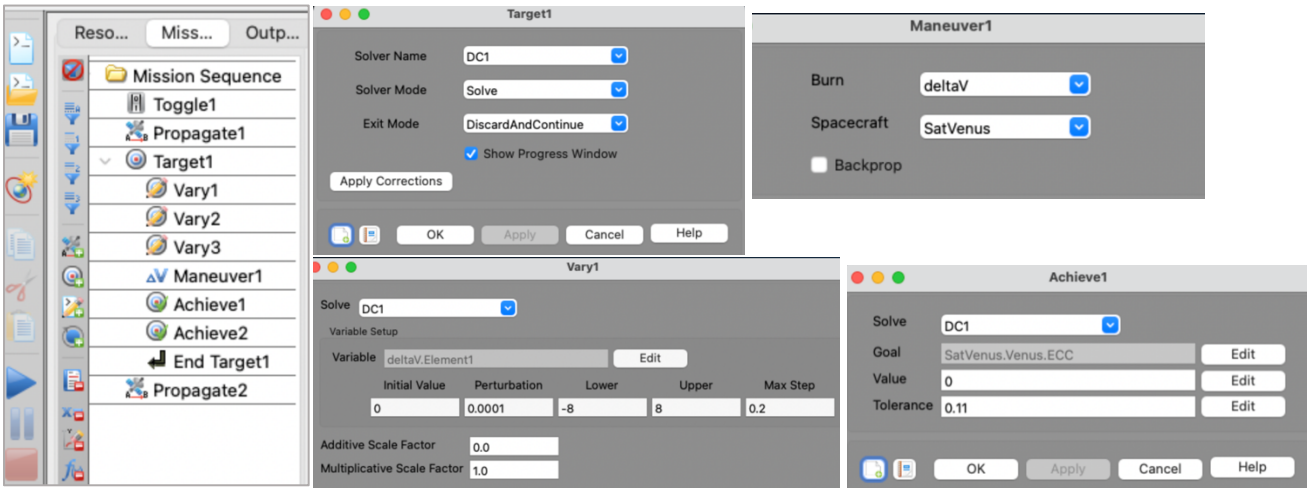


Figure 52. Mission Tree with the mission command sequence and description of its various commands.

The target sequence for the mission is determined within the *Target* and *EndTarget* commands where the differential corrector (DC) and solver mode are specified. The *Vary* command is where the variable name, initial value, limit bounds and the maximum step of the differential corrector are specified for every one of the elements of the delta-V (or burn) vector. *Very1, Very2* and *Very3* were setup to correspond to *Element1, Element2* and *Element3* respectively. *Element1, Element2* and *Element3* correspond to the velocity, normal and binormal elements of the *Velocity-Normal-Binormal* coordinate system respectively. *Velocity-Normal-Binormal* or *VNB* is a non-inertial coordinate system based upon the motion of the spacecraft

with respect to the origin sub-field, in this case Venus. For example, *Origin* was chosen as Venus, then the X-axis of this coordinate system is the along the velocity of the spacecraft with respect to the Venus, the Y-axis is along the instantaneous orbit normal (with respect to the Venus) of the spacecraft, and the Z-axis points away from the Venus as much as possible while remaining orthogonal to the other two axes, completing the right-handed set. The limit bounds in the *Vary* command were chosen as the smallest and largest acceptable values for each component of a Delta-V vector [10]. The rest of the elements under this command were taken to be the default values shown in figure 52. The *Manuver1* command specifies the burn and spacecraft programed in the Resource section and links this command to that set up.

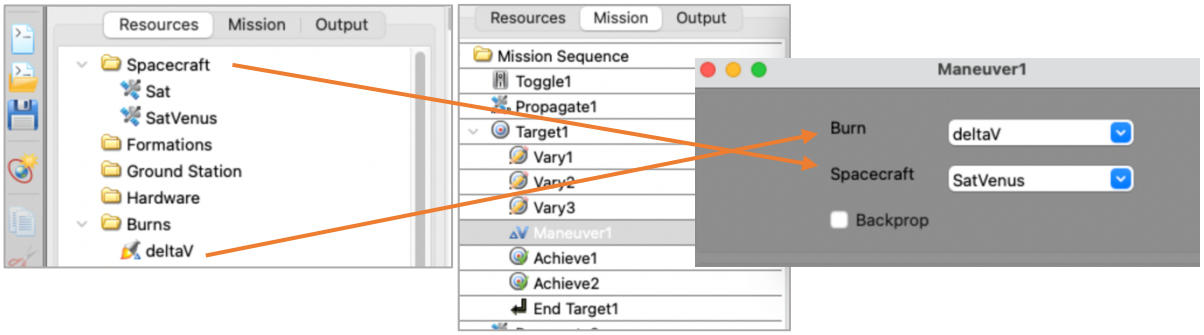
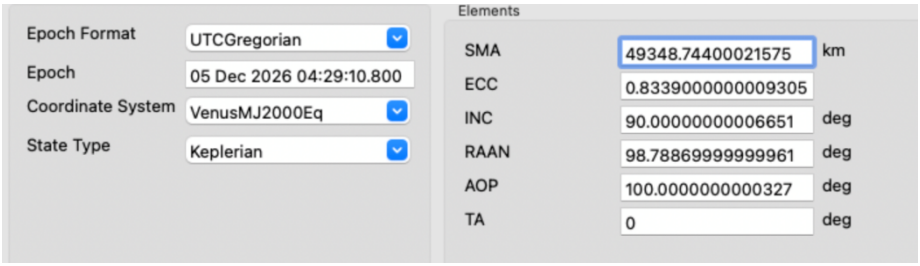
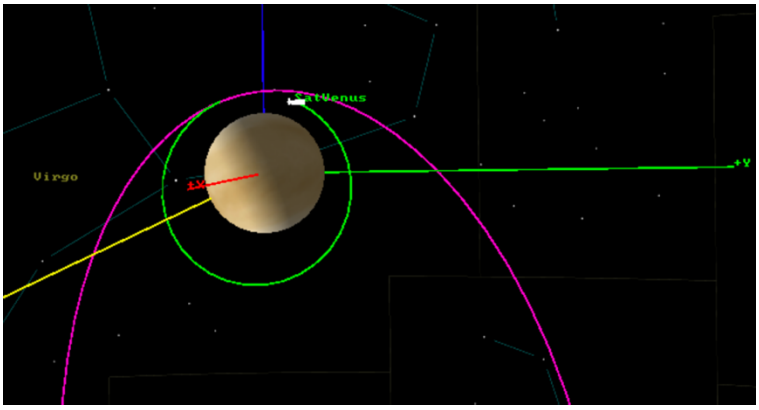


Figure 53. Setup for the *Maneuver* command in GMAT.

Figure 53 shows the setup for the *Manuver1* command and how it links *Maneuver1* in the Mission Tree (in the middle image) to those programmed in the Resources Tree (on the far-left image). The *Achieve* commands, are those that the maneuver is to achieve. In this example *Achieve1* is setup to achieve an eccentricity of zero (as shown on the far bottom right of figure 52), and *Achieve2* a radius of pericenter of 8,197 km, which is the approximately the radius of pericenter of the orbit under study.

In summary, the *Mission Sequence* in the *Mission Tree* goes as follows: *Toggle1* activates the desired graphs and data acquisition as an output, *Propagate1* propagates the spacecraft to the orbital point for burn, target1command sequence activates the burn vector components to be varied, the programed spacecraft and Delta-V from the *Resources Tree*, the target orbital elements to be achieved by the burn and the *Propagate2* command to propagate the spacecraft to the desired final orbital point after burn. A table of results is presented and analyzed next.

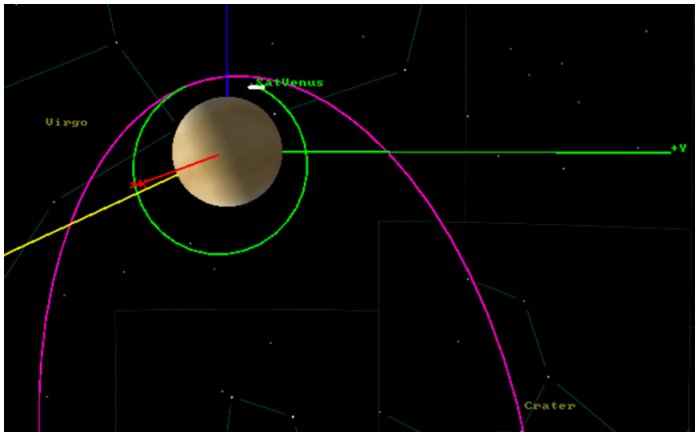
Table 12. Burn analysis results**.**

<table>
<tr>
<td>Initial State Orbital Parameters
 
</td>
<td>Recall the initial orbital parameters discussed throughout this report for comparison to the change in state of the spacecraft’s orbit after the burn in each of the analyzes presented below. The orbit’s period corresponding to these parameters is approximately 33.7 hrs. and a radius of pericenter of about 8,196.8264 km.</td>
</tr>
<tr>
<td colspan="2">Circularization with an Achieve1 target variable of eccentricity = 0 and a 0.1 tolerance</td>
</tr>
<tr>
<td>Orbital point for Burn</td>
<td>Approximate Results
Orbital elements right after burn
ΔV_{mag} = magnitude of Delta-V
RadPer = radius of pericenter</td>
</tr>
<tr>
<td>Radius of pericenter for the unperturbed orbit after a full period
 

<table>
<tr><th>Control Variable</th><th>Current Value</th><th>Last Value</th><th>Difference</th></tr>
<tr><td>deltaV.Element1</td><td>-1.8</td><td>-1.8</td><td>2.220446049250313e-16</td></tr>
<tr><td>deltaV.Element2</td><td>-1.054381468378108e-05</td><td>-1.054381468378108e-05</td><td>1.694065894508601e-21</td></tr>
<tr><td>deltaV.Element3</td><td>-0.1940114205019212</td><td>-0.1940114205019212</td><td>0</td></tr>
<tr><th>Constraints</th><th>Desired</th><th>Achieved</th><th>Difference</th></tr>
<tr><td>(==) SatVenus.Venus.ECC</td><td>0</td><td>0.1889474616298359</td><td>0.1889474616298359</td></tr>
<tr><td colspan="4">NO CONVERGENCE</td></tr>
</table></td>
<td>ΔV_{mag} = 1.81043 km/s
SMA = 9707.27 km
PerRad = 7873.103 km
ECC = 0.189
INC = 90.00°
RAAN = 98.79°
AOP = 65.59°
TA = 58.48°
Orbit Period = 2.93 hrs.

The table below is the Delta-V DC solver report, which identifies the result as nonconvergent because it could not reach the tolerance value for a zero eccentricity of 0.1. It shows the last two DC solver steps for Delta-V to achieve an eccentricity of zero.</td>
</tr>
<tr>
<td>Orbital point for Burn
The following data was generated accounting for solar perturbation</td>
<td>Approximate Results
Orbital elements right after burn
ΔV_{mag} = magnitude of Delta-V
RadPer = radius of pericenter</td>
</tr>
</table>

Radius of pericenter after a full period



Control Variable	Current Value	Last Value	Difference
deltaV.Element1	-1.8	-1.8	2.220446049250313e-16
deltaV.Element2	-1.053697634817305e-05	-1.053697634817305e-05	1.694065894508601e-21
deltaV.Element3	-0.193898854181878	-0.193898854181878	-2.775557561562891e-17

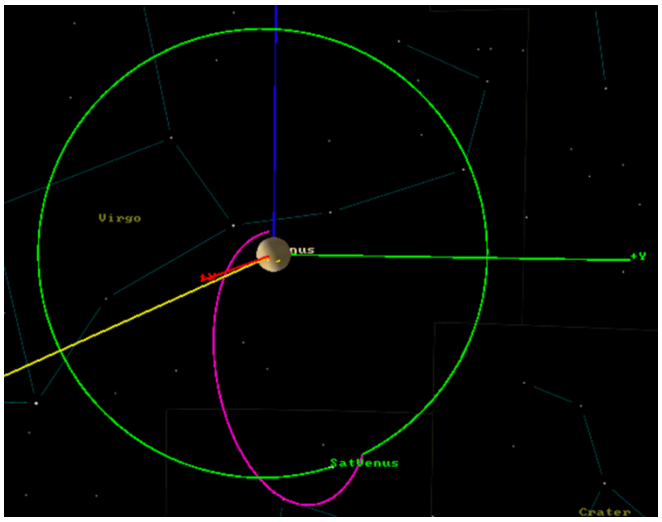
Constraints	Desired	Achieved	Difference
(==) SatVenus.Venus.ECC	0	0.189486282349685	0.189486282349685

NO CONVERGENCE

ΔV_{mag} = 1.81041 km/s
SMA = 9701.72 km
PerRad = 7863.376 km
ECC = 0.1895
INC = 90.00°
RAAN = 98.79°
AOP = 65.635°
TA = 58.44°
Orbit Period = 2.93 hrs.

The table below is the Delta-V DC solver report, which identifies the result as nonconvergent because it could not reach the tolerance value for a zero eccentricity of 0.1. It shows the last two DC solver steps for three components of Delta-V to achieve an eccentricity of zero.

1st area of investigation at approximately (-4937.322332, 31936.51537, -71381.18602) with TA = 194.372° and Time Elapse = 87382.8924 secs ≈ 24.27 hrs



Control Variable	Current Value	Last Value	Difference
deltaV.Element1	0.1501170010806216	0.1501170010806216	0
deltaV.Element2	6.304261103546826e-05	6.304261103546826e-05	0
deltaV.Element3	1.396498282719725	1.396498282719725	-4.440892098500626e-16

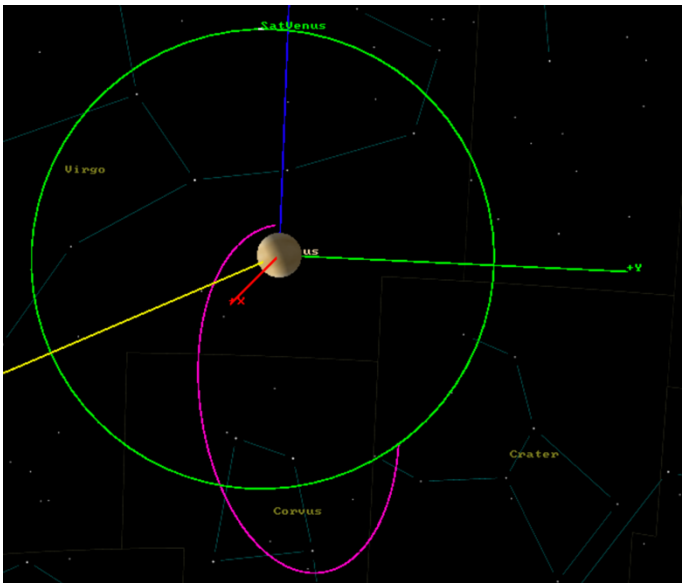
Constraints	Desired	Achieved	Difference
(==) SatVenus.Venus.ECC	0	0.06969976205749695	0.06969976205749695

CONVERGED

ΔV_{mag} = 1.4046 km/s
SMA = 77102.1 km
PerRad = 75398.3 km
ECC = 0.0697
INC = 90.00°
RAAN = 98.8°
AOP = 37.43°
TA = 257.19°
Orbit Period = 65.56 hrs.
Elapse Time = 24.3 hrs.

The table below is the Delta-V DC solver report, which identifies the result as convergent since it reaches the desired value for a zero eccentricity within the level of tolerance of 0.1. It shows the last two DC solver steps for three components of Delta-V to achieve an eccentricity of zero.

2nd area of investigation at approximately (-5397.955008, 34916.15161, -52461.21294) with a TA = 203.9755° and a Time Elapse = 99470.64052secs ≈ 27.63 hrs.



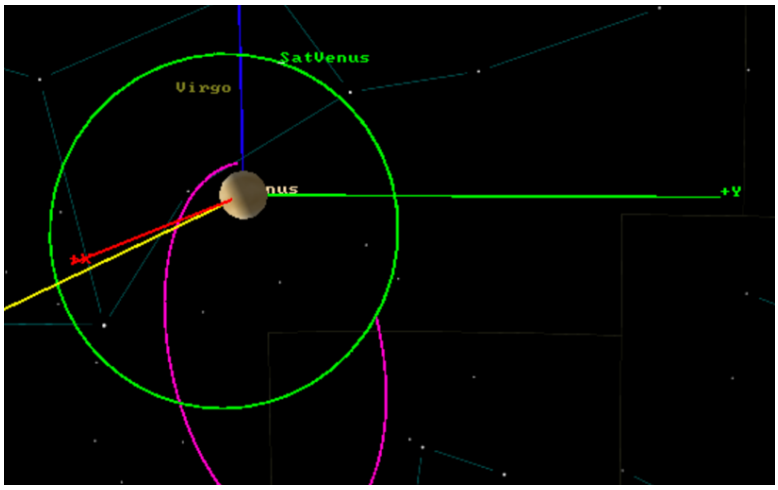
Control Variable	Current Value	Last Value	Difference
deltaV.Element1	-0.512953494616218	-0.512953494616218	0
deltaV.Element2	5.195078574244285e-05	5.195078574244285e-05	0
deltaV.Element3	1.771295067365892	1.771295067365892	4.440892098500626e-16
Constraints	**Desired**	**Achieved**	**Difference**
(==) SatVenus.Venus.ECC	0	0.07042165832737987	0.07042165832737987

CONVERGED

ΔV_{mag} = 1.844 km/s
SMA = 62720.1 km
PerRad = 58303.23 km
ECC = 0.07042
INC = 90.00°
RAAN = 98.8°
AOP = 37.3°
TA = 257.1°
Orbit Period = 65.56 hrs. Elapse Time = 427.65 hrs.

The table below is the Delta-V DC solver report, which identifies the result as convergent since it reaches the desired value for a zero eccentricity within the level of tolerance of 0.1. It shows the last two DC solver steps for three components of Delta-V to achieve an eccentricity of zero.

3rd area of investigation at approximately (-5076.737489, 32838.38482, -31961.91718) with a TA = 216.1313728° and Time Elapse = 108492.976 secs ≈ 30.14 hrs.



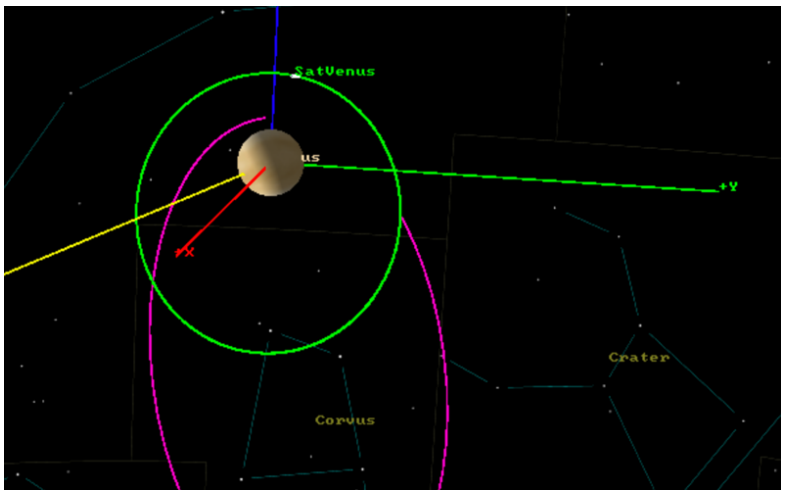
Control Variable	Current Value	Last Value	Difference
deltaV.Element1	-0.7883983515610004	-0.7883983515610004	0
deltaV.Element2	3.443561161847415e-05	3.443561161847415e-05	0
deltaV.Element3	1.8	1.8	-2.220446049250313e-16
Constraints	**Desired**	**Achieved**	**Difference**
(==) SatVenus.Venus.ECC	0	0.2488594576024455	0.2488594576024455

NO CONVERGENCE

ΔV_{mag} = 1.965 km/s
SMA = 45639.8 km
PerRad = 34281.92 km
ECC = 0.248856
INC = 90.00°
RAAN = 98.8°
AOP = 59.26°
TA = 257.68°
Orbit Period = 29.86 hrs. Elapse Time = 30.15 hrs.

The table below is the Delta-V DC solver report, which identifies the result as nonconvergent because it could not reach the tolerance value for a zero eccentricity of 0.1. It shows the last two DC solver steps for three components of Delta-V to achieve an eccentricity of zero.

4th area of investigation at approximately (-3812.527883, 24660.90067, -10034.24295) with a TA = 238.114014° and Time Elapse = 115522.907 secs ≈ 32.1 hrs.



Control Variable	Current Value	Last Value	Difference
deltaV.Element1	-1.192846102282701	-1.192846102282701	-4.440892098500626e-16
deltaV.Element2	1.100774798587414e-05	1.100774798587414e-05	-1.694065894508601e-21
deltaV.Element3	1.734157054065581	1.734157054065581	-4.440892098500626e-16
Constraints	**Desired**	**Achieved**	**Difference**
(==) SatVenus.Venus.ECC	0	0.376313781788855	0.376313781788855

NO CONVERGENCE

ΔV_{mag} = 2.105 km/s
SMA = 26040.15 km
PerRad = 16240.9 km
ECC = 0.3763
INC = 90.00°
RAAN = 98.8°
AOP = 90.244°
TA = 250.33°
Orbit Period = 12.87 hrs. Elapse Time = 32.1 hrs.

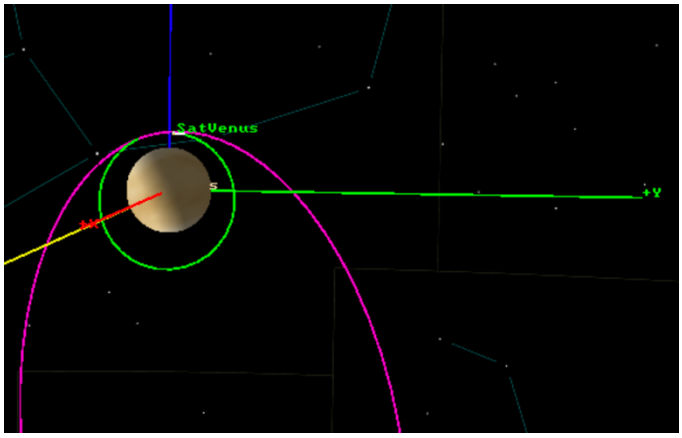
The table below is the Delta-V DC solver report, which identifies the result as nonconvergent because it could not reach the tolerance value for a zero eccentricity of 0.1. It shows the last two DC solver steps for three components of Delta-V to achieve an eccentricity or zero.

Circularization with *Achieve1* target variable of eccentricity = 0 and a 0.1 tolerance and *Achieve2* target variable of radius of pericenter = 8,197 km and a 50 km tolerance

Orbital point for Burn
The following data was generated accounting for solar perturbation unless stated otherwise

Approximate Results
Orbital elements right after burn
ΔV_{mag} = magnitude of Delta-V
RadPer = radius of pericenter

Radius of pericenter for the unperturbed orbit after a full period

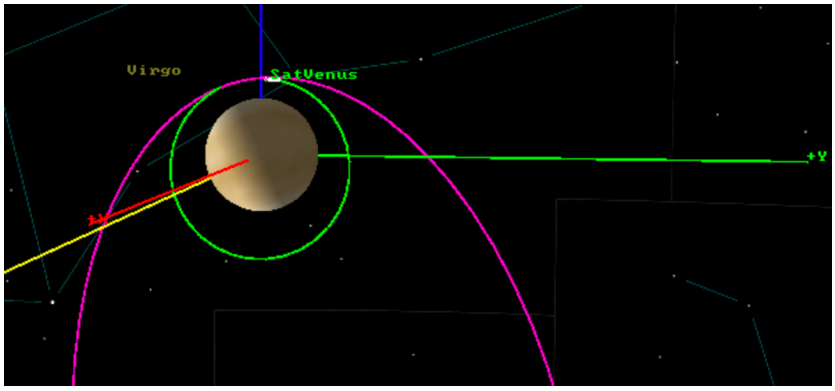


Control Variable	Current Value	Last Value	Difference
deltaV.Element1	-1.8	-1.8	2.220446049250313e-16
deltaV.Element2	-1.022200937602258e-05	-1.022200937602258e-05	-1.694065894508601e-21
deltaV.Element3	-0.5082720414573511	-0.5082720414573511	0
Constraints	**Desired**	**Achieved**	**Difference**
(==) SatVenus.Venus.ECC	0	0.1623904803493615	0.1623904803493615
(==) SatVenus.Venus.RadPer	8197	8184.876304768831	-12.12369523116922

NO CONVERGENCE

ΔV_{mag} = 1.8704 km/s
SMA = 9777.71 km
PerRad = 8184.88 km
ECC = 0.1624
INC = 90.00°
RAAN = 98.79°
AOP = 81.32°
TA = 42.77°
Orbit Period = 2.96 hrs..

Radius of pericenter after a full period

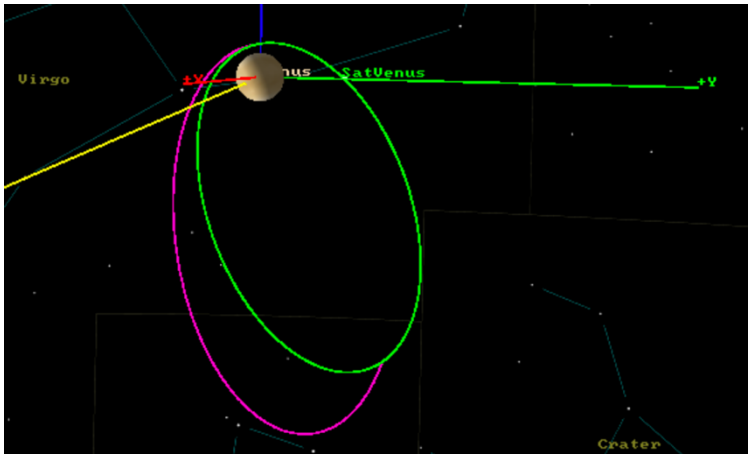


Control Variable	Current Value	Last Value	Difference
deltaV.Element1	-1.8	-1.8	2.220446049250313e-16
deltaV.Element2	-1.020387159831468e-05	-1.020387159831468e-05	-5.082197683525802e-21
deltaV.Element3	-0.5189515190220095	-0.5189515190220095	0
Constraints	**Desired**	**Achieved**	**Difference**
(==) SatVenus.Venus.ECC	0	0.1623099309056261	0.1623099309056261
(==) SatVenus.Venus.RadPer	8197	8183.663742312976	-13.33625768702404

NO CONVERGENCE

ΔV_{mag} = 1.873 km/s
SMA = 9769.32 km
PerRad = 8186.65 km
ECC = 0.1623
INC = 90.00°
RAAN = 98.79°
AOP = 81.94°
TA = 42.16°
Orbit Period = 2.96 hrs.

1st area of investigation at approximately (-4937.322332, 31936.51537, -71381.18602) with TA = 194.372° and Time Elapse = 87382.8924 secs ≈ 24.27 hrs

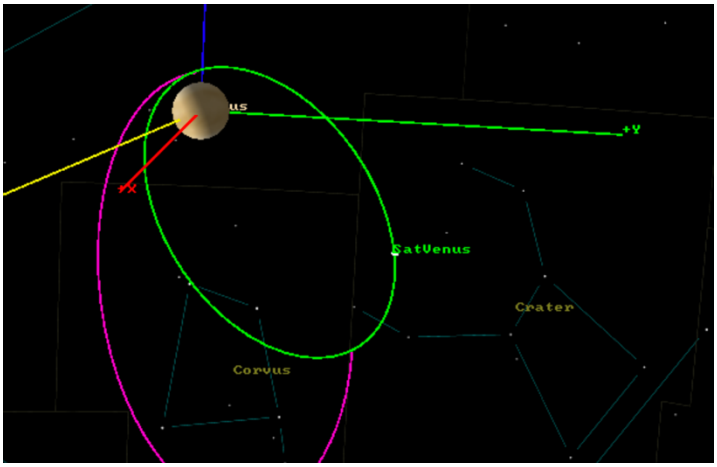


Control Variable	Current Value	Last Value	Difference
deltaV.Element1	-0.6000000000000001	-0.6000000000000001	0
deltaV.Element2	-4.091425420235686e-07	-4.091425420235686e-07	5.293955920339377e-23
deltaV.Element3	0.5384005596620141	0.5384005596620141	0
Constraints	**Desired**	**Achieved**	**Difference**
(==) SatVenus.Venus.ECC	0	0.8123476066023334	0.8123476066023334
(==) SatVenus.Venus.RadPer	8197	8104.00305800538	-92.9969419946201

NO CONVERGENCE

ΔV_{mag} = 0.806 km/s
SMA = 43186.25 km
PerRad = 8104.0 km
ECC = 0.8123
INC = 90.00°
RAAN = 98.8°
AOP = 112.00°
TA = 182.57°
Orbit Period = 27.48 hrs.
Elapse Time = 24.3 hrs.

2nd area of investigation at approximately (-5397.955008, 34916.15161, -52461.21294) with a TA = 203.9754682° and a Time Elapse = 99470.64052 secs ≈ 27.63 hrs.

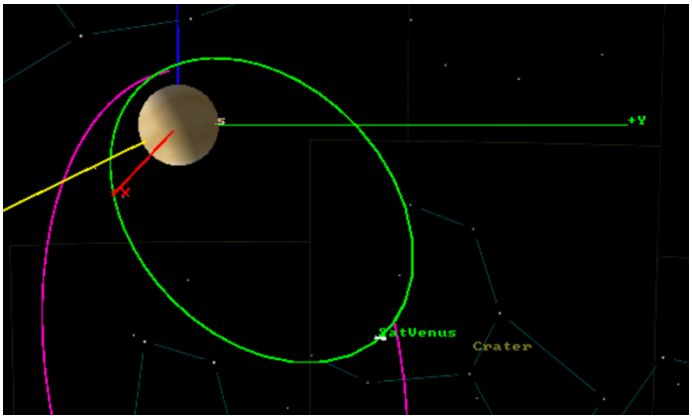


Control Variable	Current Value	Last Value	Difference
deltaV.Element1	-1.4	-1.4	0
deltaV.Element2	-1.010947119654073e-06	-1.010947119654073e-06	4.235164736271502e-22
deltaV.Element3	0.9491512753608626	0.9491512753608626	0
Constraints	**Desired**	**Achieved**	**Difference**
(==) SatVenus.Venus.ECC	0	0.7707434878635444	0.7707434878635444
(==) SatVenus.Venus.RadPer	8197	8119.661379809346	-77.33862019065418

NO CONVERGENCE

ΔV_{mag} = 1.6914 km/s
SMA = 35417.37 km
PerRad = 8119.665 km
ECC = 0.771
INC = 90.00°
RAAN = 98.8°
AOP = 125.76°
TA = 178.56°
Orbit Period = 20.4 hrs.
Elapse Time = 27.65 hrs.

3rd area of investigation at approximately (-5076.737489, 32838.38482, -31961.91718) with a TA = 216.1313728° and Time Elapse = 108492.976 secs ≈ 30.14 hrs.

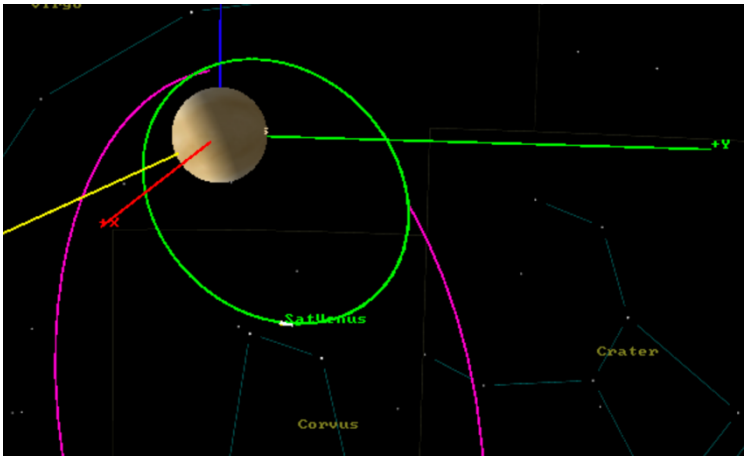


Control Variable	Current Value	Last Value	Difference
deltaV.Element1	-1.8	-1.8	2.220446049250313e-16
deltaV.Element2	-2.081073241087796e-06	-2.081073241087796e-06	0
deltaV.Element3	1.118774870559223	1.118774870559223	2.220446049250313e-16
Constraints	**Desired**	**Achieved**	**Difference**
(==) SatVenus.Venus.ECC	0	0.6966537800280322	0.6966537800280322
(==) SatVenus.Venus.RadPer	8197	8124.792718158603	-72.20728184139716

NO CONVERGENCE

ΔV_{mag} = 2.112 km/s
SMA = 26783.9 km
PerRad = 8124.8 km
ECC = 0.697
INC = 90.00°
RAAN = 98.8°
AOP = 133.3°
TA = 183.6°
Orbit Period = 29.86 hrs.
Elapse Time = 13.43 hrs.

4th area of investigation at approximately (-3812.527883, 24660.90067, -10034.24295) with a TA = 238.114014° and Time Elapse = 115522.907 secs ≈ 32.1 hrs.



Control Variable	Current Value	Last Value	Difference
deltaV.Element1	-1.8	-1.8	2.220446049250313e-16
deltaV.Element2	-4.035572076343567e-06	-4.035572076343567e-06	0
deltaV.Element3	1.17258835266406	1.17258835266406	4.440892098500626e-16
Constraints	**Desired**	**Achieved**	**Difference**
(==) SatVenus.Venus.ECC	0	0.5610451122443133	0.5610451122443133
(==) SatVenus.Venus.RadPer	8197	8127.718750938174	-69.2812490618262

NO CONVERGENCE

ΔV_{mag} = 2.148 km/s
SMA = 18516.07 km
PerRad = 8127.72 km
ECC = 0.561
INC = 90.00°
RAAN = 98.8°
AOP = 134.5°
TA = 205.965°
Orbit Period = 7.715 hrs.
Elapse Time = 32.1 hrs.

Table 12 shows the sequence of simulations done trying to understand how solar gravitational potential could aid either circularizing an orbit around Venus or approaching Venus while circularizing the spacecraft's orbit. The first part of the table shows the results of the simulations trying to circularize the very elliptical orbit under study starting with a location of burn at radius of pericenter after a full period and four other areas on the spacecraft orbit where solar perturbation becomes more prominent. The period of the modified Venera D mission presented in this report is about 33.7 hours. The areas under study happen approximately 24.27 hrs. and TA = 194.4°, 27.63 hrs. and TA = 203.98°, 30.14 hrs. and TA = 216.13°, 32.1 hrs. and TA = 238.11°, and at the end of the orbit's period at radius of pericenter.

The first part of the table where the goal is circularizing the orbit only, shows that burning at the radius of pericenter decreases both the eccentricity and radius of perigee with the Delta-V for the perturbed orbit 0.2 m/s less that the perturbed one, a smaller radius of pericenter by about 10 km and eccentricity 0.0005 larger. Perhaps these values could get more significant after several rounds of the perturbed orbit before burning at radius of pericenter, or by making smaller burns at several passes at radius of pericenter. Compared to the other four areas under study, the radius of pericenter of the perturbed orbit proved to be a more effective spot to burn given a close distance to the planet is relevant for a mission. Otherwise, circularization at a much larger radius of pericenter is much closer to an eccentricity of zero with the first area leading the way followed by areas 2, 3 and 4 in order of increasing eccentricity and with their radius of pericenter decreasing as the eccentricity increases. This can be noticed on the plots generated by GMAT in the first part of table 12.

The second part of the table aims to present the results of the simulations targeting both, a small eccentricity close to zero and a limit radius of pericenter equal to that of the Venera D-like orbit. For the unperturbed and perturbed orbits at radius of pericenter, the Delta-V for the perturbed orbit is approximately 2.6 m/s less that the perturbed one with a smaller radius of pericenter by about 1.77 km and eccentricity 0.0001 larger. Burning at radius of pericenter reduces both the eccentricity and radius of perigee with a slight difference between the burn happening at and unperturbed versus a perturbed orbit, with the perturbed orbit placing the spacecraft at a slightly closer distance of radius of perigee and smaller burn but larger eccentricity (in the 10^{-3} difference for ΔV_{mag} and e). The values are, once more shown below for reference.

Table 13. Perturbed vs unperturbed results targeting e and r_p

Unperturbed Orbit	**Perturbed Orbit**
ΔV_{mag} = 1.873 km/s	ΔV_{mag} = 1.8704 km/s
PerRad = 8186.65 km	PerRad = 8184.88 km
ECC = 0.1623	ECC = 0.1624

Circularization at a larger eccentricity but radius of pericenter closer to the desired value is observed for the other four areas of study. In this case, the propagator has a much harder time reaching an eccentricity of zero while preserving the radius of pericenter at the same distance in all four areas. Area 1 presents this behavior more pronounced, which lessens as the spacecraft moves closer to radius of perigee in areas 2, 3 and 4 in order of decreasing eccentricity and increasing radius of pericenter toward the target value. This can also be noticed on the plots generated by GMAT in the second part of table 12.

Generating a grid of data for as many points of phase space as possible around Venus at different points of approach to enter a closed orbit with respect to the Sun could be a promising mission to accomplish for different types of goals such as planet re-entry, circularization, etc. This report presents only the basis of this idea with a very simple analysis that aims to prove this concept.

Chapter 6: Summary, Conclusion and Future Work

6.1 Summary and conclusion

This report presents the analysis of a Venera D-like mission. All parameters of the orbit under study matching those of the Venera D mission except by the radius of pericenter which was increased by 10,000 km to better serve the proof-of-concept objective. One of the goals of this investigation was to train an RNN LSTM to correct for solar gravity potential perturbation on a spacecraft while in closed orbit around Venus. A database for such an orbital path was gathered via the RK89 propagation simulator in GMAT to be fed into an RNN LSTM with two LSTM layers with 50 and 10 units respectively, a 0.46-rate regularizer and a TimeDistributed wrapped Dense layer of 3 units for the output data.

The original database generated and presented in table 4, chapter 2, had to be modified to be used for the training of the LSTM later in chapter 4 because the time steps of such data set were not evenly distributed, which is required to train any type of RNN. Hence, six features of the 2020 samples were used for training where each feature corresponds to each one of the three components of the location of the spacecraft at every point of its orbit in cartesian coordinates for both, the unperturbed and perturbed orbits. The 2020 samples correspond to all steps on the spacecraft's orbit for a full period at a rate of one step per minute. The data was then split into the training data set (X) and the label data set (Y). The LSTM is to approach as close as possible the train data to the label data. It is expected for the difference between the train and label data during this training to represent the solar gravitational potential perturbation. This difference is to be represented by the output weights (considering the shift given by its biases) of the RNN LSTM. This, ideally, would create a grid of weights or solar gravitational potential perturbation per step on the spacecraft's path of the given orbit. However, it was found that obtaining the output weights per output sample is a far more complicated task since this would require a one-to-one correspondence between weight and output sample. As mentioned in chapter 5, the weights and biases in an LSTM do not have a one-to-one correspondence with the output data but depend on the number of LSTM cells and gates of the LSTM cell. Rather, there are two weights and one bias per gate in every LSTM cell, which makes eight weights and four biases per cell, since there are four gates in every LSTM cell. With the output layer of the RNN LSTM built for training in this project being a dense layer of 3 units, the training output resulted in 33 weights and biases (10 weights and 1 bias per cell for 3 dense layer output cells) for the 2020 elements per feature used to train it.

The other two sets of 2020 samples with a similar initial state as the orbit under study with two different angles of inclination ($\pm 15°$ than that of the Venera D-like orbit) were generated in GMAT for the RNN LSTM training testing and validation. These two other data sets were split the same way as the training data and were generated with the same rate of orbital step per minute. The RNN LSTM was able to approach the training data to that of the label with and accuracy of approximately between 89% and 92% with number of 2000 training epochs. There is much more work to be done regarding this part of this investigation to reach a level of accuracy of at least 98%, and next recommended steps will be provided in the next section. It is

important to note here that the data was "normalized" by using the so-called *Min-Max Normalization* in the field of Artificial Intelligence. In reality, *Min-Max Normalization* is a way to rescale data to [0,1] rather than normalizing it as it is in Statistics. In the next section, it will be discussed why it is recommended to also normalize the data to get more accurate results when training an RNN, and how this normalization and rescaling should be done per feature rather than across training and label data.

An analysis of the perturbation on the spacecraft in orbit due to solar gravitational potential was done and presented in chapter 5. This analysis was started by letting the spacecraft drift towards the solar perturbation until it either reached the outskirts of the atmosphere of Venus or it significantly decreased the eccentricity of the orbit. This was done as a means to find out how solar perturbation can aid either the re-entry to Venus or the circularization of the spacecraft's orbit around it at a low enough radius of pericenter for observation of the planet. The result of this analysis shows that the drifting to reach the planet or circularizing its orbit around it highly depends on the initial state vector of the closed orbit with respect to the Sun. Therefore, one of the simulations was done at a different epoch and initial state vector where the Sun is at a different position with respect to Venus and the initial state vector of the spacecraft's orbit.

The spacecraft drifting analysis was done in three different kinds of orbits. The first one was the Venera D-like orbit which was left drifting until it successfully reached the outskirts of Venus' atmosphere at a radius of perigee of approximately 6403.3 km and an ellipticity of about 0.87 in 451.65 days. Hence, using solar gravitational potential perturbation to aid planet re-entry might be a promising method to save fuel either by letting a spacecraft drift for some time or making a small burn to help it reach the planet in less time than it would take by drifting only. The second orbit was done at the same epoch as the Venera-D like orbit, smaller semi-major axis, eccentricity, inclination and argument of perigee but larger longitude of the ascending node. This initial state vector was such that it placed the spacecraft father from the Sun line (the line joining the center of mass of Venus and the Sun). A drifting of about 4.8 years decreased the eccentricity of this orbit from 0.82 to 0.65 at a radius of pericenter of about 15,972 km. In this case, the radius of pericenter increased as its eccentricity decreased. Therefore, this type of orbit could be useful for circularization rather than approach to the planet. However, it took a long time for the solar perturbation drifting of the spacecraft to decrease its eccentricity by 0.17. The relevance of this observation is to learn how the position of the spacecraft with respect to the Sun affects the orbital path of the spacecraft.

The third orbit was done at a different epoch, higher semi-major axis, eccentricity, and argument of perigee but smaller inclination and longitude of the ascending node. In this type of orbit the spacecraft was approximately at the same angular distance from the Sun line as that of the previous orbit but with its radius of pericenter almost in front of the Sun. This type of orbit took about 5 years to go from an eccentricity of 0.86 to about 0.58 but at a radius of pericenter about 5700 km larger than the initial one. Perhaps a combination of an initial state vector placing the spacecraft close to the Sun line and its radius of pericenter in front of the Sun could more successfully aid the circularization of its orbit without increasing its radius of pericenter as much.

A second alternative for creating a grid of solar gravitational perturbation acting on the spacecraft at different orbital points was presented in chapter 5 as well. This could be done for as many orbits as possible to create a solar perturbation vector field on phase space around Venus. As a proof of concept, a vector field for the Venera D-like orbit was done for some of the orbital points of the spacecraft's path using the data generated in GMAT. One step per every ten minutes was taken for both, perturbed and unperturbed orbits to create this solar perturbation vector field. This was done by taking the difference between every perturbed and unperturbed step toward the direction of the unperturbed orbit. This solar perturbation vector field is useful to visualize how exactly the spacecraft is being acted upon by the Sun's and Venus' solar perturbation and determined the regions where solar and Venus' perturbations on the spacecraft are more significant. A burn analysis was done on five different regions on the vector field where the perturbations are larger including at perturbed radius of pericenter to be compared with the burn at the unperturbed orbit's radius of pericenter, which is a common technique for orbit circularization. This burn analysis focused more in solving for circularizing the orbit, since it was already proven that the solar perturbation does aid with re-entry. The four areas of investigation other than at radius of pericenter are approximately shown in figure 51.

Table 12 shows the sequence of results from the burn analysis done via GMAT simulations. The first sequence of burn analysis was done with the goal of circularizing the Venera D-like orbit only. In this part of the analysis, it was found that burning at the radius of pericenter decreases both the eccentricity and radius of perigee (and this might be the reason why it is a common point of burn for circularization) with the Delta-V for the perturbed orbit 0.2 m/s less that the perturbed one, a smaller radius of pericenter by about 10 km and an eccentricity 0.0005 larger. Although the value of eccentricity reached by burning at the perturbed orbit's radius of pericenter is slightly larger, it might be convenient to have the spacecraft closer to the planet for observation with a bit less of a burn. As mentioned before, these values might get more significant with letting the spacecraft drift for several passes at radius of pericenter before burning fuel or do tiny burns at every pass. The simulation for the other four areas achieved an eccentricity very closed to zero but at the cost of a much larger radius of pericenter (within 65 to 25 thousand difference). The smallest the orbit's eccentricity got, the larger its radius of pericenter and the less of a burn it required. This behavior was more pronounced farther away from radius of pericenter, being more pronounced in area of analysis 1 followed by areas 2, 3 and 4 in preceding order. Hence, circularizing at area 1 is most effective orbit if being farther away from the planet suits the goals of the mission under consideration.

The second sequence of simulations was done on the same orbit with the goal of circularizing it at a radius of pericenter like that of the Venera D-like orbit, 8,197 km. Burning at a radius of pericenter of the perturbed orbit resulted in approximately 2.6 m/s less that the unperturbed one and with a smaller radius of pericenter by about 1.77 km and eccentricity 0.0001 larger. In this case, burning at the unperturbed orbit the simulation reached the values to be achieves slightly more closely. Once more, the perturbed orbit simulation yielded a slightly smaller radius of pericenter and burn and larger eccentricity. The other four areas presented similar behavior as the first set of simulations. Reaching a small eccentricity was unsuccessful while preserving the same radius of pericenter. This behavior was more pronounced in the area of analysis 1 and increasingly less at shorter distances from radius of pericenter in areas 2, 3 and 4.

This analysis shows that the radius of pericenter is indeed the best point to burn for circularization while preserving a small radius of pericenter. Solar perturbation slightly helps to achieve this with a closer approach to the planet with a bit let of a burn and with the possibility of getting more aid by letting the spacecraft drift for several passes through radius of pericenter. However, if circularizing is the main goal at the cost of a large radius of pericenter, burning farther away from center of perigee, around area 1of the perturbed orbit is more effective.

6.2 Future Work

Due to limited time, the training of the RNN LSTM was not advanced to try more effective techniques found later during this project to improve the training accuracy. For instance, the number of units and layers of the RNN LSTM developed in this project was done by trial and error. This should be investigated more in depth, since some reliable sources, such as [69], recommend a more systematic way to better determined the number of neurons to use in every layer that have proven to be effective to train NNs. These recommendations were tried in the beginning phases of the development of the LSTM with not much success. The reason for that is believed to be the fact that the data was not scaled in a more effective way nor normalized at all. It is recommended for the data used for training to not only be scaled to make the optimization process faster, but also to be normalized to make it more precise. Also, the scaling and normalizing of data should be done per feature rather than across a set of features as it was done in this report (normalization was done across the 3 features for the position components of the unperturbed data and separately for the other 3 features of the components of the perturbed data). Normalizing data per feature leads to the preservation of the variance for each feature in the data set. The variance is an important part of the optimization process done by the any NN, and it is the difference between validation and training error [70], which results in the accuracy of the model. In this way the variance is responsible for the differences in predictions of the same observation in different mappings from input to output in the statistical model use for training [71]. Therefore, it is important for the variance to be understood by the NN for a every feature of the data set to be accounted for when optimizing.

A larger bank of data with several orbits at different initial state vectors and different positions with respect to the Sun were proposed to better train the RNN LSTM in the beginning of this project, and it is still highly recommended. Also, this larger bank of data should include the other six features composed by the velocity vector components for the spacecraft at every orbital step for the perturbed and unperturbed orbits.

The extraction of LSTM output weights was not investigated enough to put into practice. A possible way to do this is with autoencoders, which make possible a one-to-one correspondence between RNN output weights and elements without compromising the speed of training as a one-to-one RNN model might do. In the case that developing a grid of solar perturbation vectors in phase space around Venus is to be done in a way other than extracting the weights of a trained NN, a solar perturbation vector field could be constructed by the simulation of as many orbits as possible in a similar manner as shown in chapter 5. This could help with finding the areas where the gravitational potential field of the Sun can aid the path of a spacecraft

of a given mission and train the RNN LSTM to do autonomous burning in such areas. It is important to do a detailed study of the most convenient ways to place the spacecraft in close orbit with respect to the Sun beforehand in order to take advantage from the solar perturbation in the most effective way to meet the goals of the particular mission. The initial placement of the spacecraft in close orbit would depend on the goal of the mission of interest, such as circularization, planet re-entry, etc.

Another way to advance this project even further is by implementing the grid of solar perturbation vector field (via the weights of the RNN LSTM or multiple orbit simulations) into a spacecraft control system by burning the grid of values into a type of flash memory to the microprocessor and memory chips of a computer. This type of process is described in Nelson Wong's thesis titled *"On Clustering Low-Cost SoC FPGA Devices for Deep Learning Inference Applications."* An abstract of this paper can be found in Appendix F for reference. Given that the solar perturbation vector field can be implemented in the control system of choice, it is ideal to be able to turn on and of the solar perturbation corrections at any point in order to manipulate the areas where it is convenient to let the spacecraft drift, or its path to be corrected from such perturbations.

References

[1] Kramer, M., "5 amazing discoveries made using the Hubble Telescope in the past 25 years." *Mashasble* [online article], 2015, URL https://mashable.com/archive/hubble-telescope-discoveries [retrieved 28 May, 2020].

[2] Kiger, P., "10 Reasons Why Space Exploration Matters to You." *HowStuffWorks* [online article], 2021, URL: https://science.howstuffworks.com/10-reasons-space-exploration-matters.htm [retrieved 28 May, 2020].

[3] Tian, R., Sarazen, M., "What to Expect About Space Exploration and Machine Learning After Elon Musk's SpaceX Makes History," *Synced AI Technology & Industry Review* [online article], 2018, URL: https://medium.com/syncedreview/the-new-age-of-discovery-space-exploration-and-machine-learning-64883f7dc7f9, [retrieved 27 September, 2020].

[4] De Smet, S., "On the design of solar gravity driven planetocentric transfers using artificial neural networks." Ph.D. Dissertation, Department of Aerospace Engineering Sciences, University of Colorado, Boulder, Colorado, 2018.

[5] NASA, "Gravity Recovery and Interior Laboratory (GRAIL)," *GRAIL Launch,* URL: https://www.nasa.gov/pdf/582116main_GRAIL_launch_press_kit.pdf, [retrieved 29 May, 2020].

[6] Koon, S., Lo, M., Marsden, J., Ross, S., "Low Energy Transfer to the Moon," *Celestial Mechanics and Dynamical Astronomy*, Vol. 81, Sep. 2001, pp. 63-73. https://doi.org/10.1023/A:1013359120468.

[7] Gomez, G., Barrabés, E. "Space Manifold Dynamics. Encyclopedia of Life Support Systems", *Encyclopedia of Life Support Systems Journal (EOLSS)*, URL: https://www.eolss.net/Sample-Chapters/C01/E6-119-55-06.pdf [retrieved 29 May, 2020].

[8] Parker, J., "Training Overview – GMAT Fundamentals," *NASA Goddard Space Flight Center* [tutorial video], 2014, URL: https://www.youtube.com/watch?v=9vt9iBuhYng, [retrieved 29 May, 2020].

[9] Anis, A., "General Mission Analysis Tool (GMAT)," *Goddard Space Flight Center Innovative Partnerships Program Office*, URL: https://opensource.gsfc.nasa.gov/projects/GMAT/index.php, [retrieved 29 May 2020].

[10] NASA, "General Mission Analysis Tool (GMAT) User Guide," *The GMAT Development Team*, URL: http://gmat.sourceforge.net/docs/nightly/html/index.html [retrieved 26 September 2020].

[11] De Smet, S., Parker, J., Scheeres, D., "Systematic Exploration of Solar Gravity Driven Orbital Transfers in the Martian System Using Artificial Neural Networks," *2018 AAS/AIAA Astrodynamics Specialist Conference*, AAS 18-216, Snowbird, Utah, August 31, 2018.

[12] Jules, K., Lin, P., "Artificial Neural Networks Applications: From Aircraft Design Optimization to Orbiting Spacecraft On-board Environment Monitoring." NASA Glenn Research Center, Cleveland, Ohio, August 2002.

[13] Singh, A., "Anomaly Detection for Temporal Data using Long Short-Term Memory (LSTM)," Master's Thesis at KTH Information and Communication Technology, Stockholm, Sweden, 2017.

[14] Navlani, S., "Neural Network Models in R," *Data Camp Tutorials* [online tutorial], URL: https://www.datacamp.com/community/tutorials/neural-network-models-r, [retrieved 26 September, 2020].

[15] Sanjeevi, M., "Chapter 10.1: DeepNLP — LSTM (Long Short-Term Memory) Networks with Math," *Medium Machine Learning* [online article], 2018, URL: https://medium.com/deep-math-machine-learning-ai/chapter-10-1-deepnlp-lstm-long-short-term-memory-networks-with-math-21477f8e4235, [retrieved 27 September, 2020].

[16] The Venera-D Joint Science Definition Team, "Venera-D: Expanding Our Horizon of Terrestrial Planet Climate and Geology Through the Comprehensive Exploration of Venus," Phase II Final Report*,* Jan. 31, 2019.

[17] Kremic, T.; Hunter, G.; and Rock, J., "Long-lived in-situ solar system explorer (LLISSE)," *15th Meeting of the Venus Exploration and Analysis Group (VEXAG), Laurel, MD,* URL: https://www.lpi.usra.edu/vexag/meetings/archive/vexag_15/presentations/8-Kremic-LLISSE.pdf [retrieved 28 September, 2020].

[18] Műller, N.; Helbert, J.; Hashimoto, G.L.; Tsang, C.C.C.; Erard, S.; Piccioni, G.; and Drossart, P., "Venus surface thermal emission at 1 μm in VIRTIS imaging observations: Evidence for variation of crust and mantle differentiation conditions," *J. Geophys. Res*., Vol. 113, 2008. https://doi.org/10.1029/2008JE003118.

[19] Way, M.J., Del Genio, A. D., Kiang, N. Y., Sohl, L. E., Grinspoon, D. H., Aleinov, I., Kelley, M., Clune, T., "Was Venus the first habitable world of our solar system?" *Geophys. Res.Lett*., Vol. 43, 2016, no. 16, pp. 8376–8383. https://doi.org/10.1002/2016GL069790.

[20] Limaye, S.S.; Mogul, R.; Smith, D.J.; Ansari, A.H.; Słowik, G.P.; and Vaishampayan, P., "Venera-D Phase II Final report 147 Venus' Spectral Signatures and the Potential for Life in the Clouds," *Astrobiology*, vol. 18, 2018, no. 9, pp. 1181–1198. http://doi.org/10.1089/ast.2017.1783.

[21] Moroz, V., "Stellar magnitude and albedo data of Venus," *Venus*, Hunten, D.M., Colin, L., Donahue, T.M., and Moroz, V.I. (Eds.), Univ. of Arizona Press, Tucson, AZ, 1983, pp. 27–35. https://doi.org/10.1016/0273-1177(85)90202-9.

[22] Barabash, S.; Fedorov, A.; Lundin, R.; and Sauvaud, J.A., "Martian atmospheric erosion rates," *Science,* Vol. 315, 2007, pp. 501–503.

[23] Project Calliope, "The 6 Classic Orbital Elements," *Science 2.0* [online article], URL: https://www.science20.com/satellite_diaries/6_classic_orbital_elements-79561, [retrieved 15 October, 2020].

[24] Williams, D., "Planetary Fact Sheet – Metric," *NASA Goddard Space Flight Center fact sheet*, URL: https://nssdc.gsfc.nasa.gov/planetary/factsheet/, [retrieved 29 October, 2020].

[25] Cutis, H., *Orbital Mechanics for Engineering Students*, 3rd ed., Elsevier Ltd., Oxford, 2014**,** Chaps. 1, pp. 37 - 42.

[26] Navigation and Ancillary Information Facility, "Navigation and Ancillary Information Facility," *NASA tutorial* [online tutorial], URL: https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/Tutorials/pdf/individual_docs/04_concepts.pdf [retrieved 29 October, 2020].

[27] Ovid and Martin, C. *Metamorphoses*. W.W. Norton, New York, 2004.

[28] Sparkes, B. *The Red and the Black: Studies in Greek Pottery*. Routledge, Abingdon, 1996.

[29] Tandy, D. W. *Works and Days: A Translation and Commentary for the Social Sciences*. University of California Press, Berkeley, 1997.

[30] Lovelace, A. Notes upon L. F. Menabrea's "Sketch of the Analytical Engine invented by Charles Babbage," London, 1842.

[31] Goodfellow, I., Bengio, Y., Courville, A., "Deep Learning," *an MIT Press book, Ch. 1, 6, 8, 10* [online book], URL: http://www.deeplearningbook.org, [retrieved November 6, 2020].

[32] Meß, J., Dannemann, F., Greif, F., "Techniques of Artificial Intelligence for Space Applications - A Survey," *European Workshop on On-Board Data Processing (OBDP2019) Research Gate* [online conference paper], URL: https://www.researchgate.net/publication/336073809_Techniques_of_Artificial_Intelligence_for_Space_Applications_-_A_Survey#fullTextFileContent, [retrieved November 6, 2020].

[33] LeCun,Y., Bengio, Y., Hinton, G., "Review – Deep Learning," *doi 10.1038 nature14539 Vol. 251 article review*, URL: https://www.cs.toronto.edu/~hinton/absps/NatureDeepReview.pdf, [retrieved November 8, 2020].

[34] Jarrett, K., Kavukcuoglu, K., Ranzato, M., and LeCun, Y. "What is the best multi-stage architecture for object recognition?" *12th International Conference on Computer Vision*, IEEE, Kyoto, 2009, pp. 2146-2153. doi: 10.1109/ICCV.2009.5459469.

[35] Nair, V. and Hinton, G. (2010). "Rectified linear units improve restricted Boltzmann machines," *International Conference on Machine Learning*, Haifa, 2010. URL: https://www.cs.toronto.edu/~fritz/absps/reluICML.pdf, [retrieved November 8].

[36] Glorot, X., Bordes, A., and Bengio, Y. (2011a). “Deep sparse rectifier neural networks,” *International Conference on Artificial Intelligence and Statistics*. Vol. 15, Fort Lauderdale, FL, 2011. URL: https://proceedings.mlr.press/v15/glorot11a/glorot11a.pdf, [retrieved November 8].

[37] LeCun,Y., Bottou, L. Orr, G., Müller, K., “Efficient BackProp,” *Montavon G., Orr G.B., Müller KR. (eds) Neural Networks: Tricks of the Trade. Lecture Notes in Computer Science*, vol 7700. Springer, Berlin, Heide*lberg*. https://doi.org/10.1007/978-3-642-35289-8_3.

[38] Ruder, S. “An Overview of Gradient Descent Optimization Algorithms,” *Insight Centre for Data Analytics, NUI Galway Aylien Ltd., Dublin*, URL: https://arxiv.org/pdf/1609.04747.pdf, [retrieved 21 November, 2020].

[39] Oppermann, A. “Stochastic-, Batch-, and Mini-Batch Gradient Descent Demystified – Why do we need Stochastic, Batch and Mini Batch Gradient Descent when implementing Deep Neural Networks?,” *More from Toward Data Science a medium publication sharing concepts, ideas, and codes*, URL: https://towardsdatascience.com/stochastic-batch-and-mini-batch-gradient-descent-demystified-8b28978f7f5, [retrieved 27 November, 2020].

[40] Dematos, G., Boyd, M.S., Kermanshahi, B., Kohzadi, N., Kaastra, I. “Feedforward versus recurrent neural networks for forecasting monthly japanese yen exchange rates,” *Financial Engineering and the Japanese Markets* 3, pp. 59–75. https://doi.org/10.1007/BF00868008.

[41] Kolen, J. F., Kremer, S. C., "Gradient Flow in Recurrent Nets: The Difficulty of Learning LongTerm Dependencies," *A Field Guide to Dynamical Recurrent Networks*, IEEE, 2001, pp.237-243. doi: 10.1109/9780470544037.ch14.

[42] Baydin, A. G., Pearlmutter, B. A., Radul, A. A., Siskind, J. M., “Automatic differentiation in machine learning: a survey.” *Journal of Machine Learning Research*, 18(153), pp. 1-43, 2018, arXiv:1502.05767**.**

[43] Olah, C. “Understanding LSTM Networks,” *Github Olah’s blog*, 2015, URL: http://colah.github.io/posts/2015-08-Understanding-LSTMs/, [retrieved 14 April, 2021].

[44] Keras SIG. “About Keras,” *Python open-source library,* URL: https://keras.io/about/, [retrieved 14 April, 2021].

[45] Brownlee, J. “Your First Deep Learning Project in Python with Keras Step-By-Step,” *Machine Learning Mastery book*, 2020, URL: https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/, [retrieved 17 April, 2021].

[46] Yegulalp, S. “What is TensorFlow? The Machine Learning Library Explained,” *Software Development InfoWorld*, 2019, URL: https://www.infoworld.com/article/3278008/what-is-tensorflow-the-machine-learning-library-explained.html,
[retrieved 18 April, 2021].

[47] Johnson, K. “Google Launches TensorFlow 2.0 with Tighter Keras Integration,” *VB Lab insights The Machine Making sense of AI*, URL: https://venturebeat.com/2019/09/30/google-launches-tensorflow-2-0-with-tighter-keras-integration/, [retrieved 19 April, 2021].

[48] Google Colaboratory. “Frequently Asked Questions The Basics,” *Google Research Collaboratory*, URL: https://research.google.com/colaboratory/faq.html#:~:text=Colaboratory%2C%20or%20%E2%80%9CColab%E2%80%9D%20for,learning%2C%20data%20analysis%20and%20education, [retrieved 20 April, 2021].

[49] Wikipedia. “Graphics Processing Unit,” *Wikimedia Foundation trademark* [online encyclopedia], URL: https://en.wikipedia.org/wiki/Graphics_processing_unit. [retrieved 20 April, 2021].

[50] ActiveState. “What Is Pandas In Python? Everything You Need To Know,” *ActiveState Software Inc.* [online article], URL: https://www.activestate.com/resources/quick-reads/what-is-pandas-in-python-everything-you-need-to-know/, [retrieved 21 April, 2021]

[51] Jeevan, M. “14 Best Python Pandas Features,” *Dataconomy Data Science 101* [online article], 2015, URL: https://dataconomy.com/2015/03/14-best-python-pandas-features/, [retrieved 21 April, 2021].

[52] Brownlee, J. “How to use Data Scaling Improve Deep Learning Model Stability and Performance,” *Machine Learning Mastery Pty.* [online article], URL: https://machinelearningmastery.com/how-to-improve-neural-network-stability-and-modeling-performance-with-data-scaling/, [retrieved 22 April, 2021].

[53] Brownlee, J. *Long Short-Term Memory Networks with Python - Develop Sequence Prediction Models with Deep Learning.* Machine Learning ebook, 2020.

[54] Brownlee, J. “A Gentle Introduction to the Rectified Linear Unit (ReLu),” *Machine Learning Mastery Pty.* [online article], 2020, URL: https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/, [retrieved 24 April, 2021].

[55] Keras SIG. “LSTM Layer,” *Python open-source library*, URL: https://keras.io/api/layers/recurrent_layers/lstm/, [retrieved 25 April, 2021].

[56] Keras SIG. “Dropout Layer,” *Python open-source library*, URL: https://keras.io/api/layers/regularization_layers/dropout/, [retrieved 25 April, 2021].

[57] Keras SIG. “TimeDistributed Layer,” *Python open-source library*, URL: https://keras.io/api/layers/recurrent_layers/time_distributed/, [retrieved 25 April, 2021]

[58] Brownlee, J. "How to Use the TimeDistributed Layer in Keras," *Machine Learning Mastery Pty.* [online article], 2017, URL: https://machinelearningmastery.com/timedistributed-layer-for-long-short-term-memory-networks-in-python/, [retrieved 25 April, 2021].

[58] Keras SIG. "Losses," *Python open-source,* URL: https://keras.io/api/losses/, [retrieved 25 April, 2021].

[59] Keras SIG. "Model training APIs," *Python open-source library*, URL: https://keras.io/api/models/model_training_apis/, [retrieved 25 April, 2021].

[59] Brownlee, J. "Gentle Introduction to Models for Sequence Prediction with RNNs," *Machine Learning Mastery Pty.* [online article], 2017, URL: https://machinelearningmastery.com/models-sequence-prediction-recurrent-neural-networks/, [retrieved 25 April, 2021].

[60] Karpathy, A. "The Unreasonable Effectiveness of Recurrent Neural Networks," *Andrej Karpathy blog* [online article], URL: http://karpathy.github.io/2015/05/21/rnn-effectiveness/, [retrieved 25 April, 2021].

[61] Stack overflow questions. "How to interpret weights in a LSTM layer in Keras," *Stack Exchange Inc.*, 2019, URL: https://stackoverflow.com/questions/42861460/how-to-interpret-weights-in-a-lstm-layer-in-keras, [retrieved 29 April, 2021].

[62] Keras-team. "When and How to use TimeDistributedDense #1029," *Github* [online blog], URL: https://github.com/keras-team/keras/issues/1029, [retrieved 30 April, 2021].

[63] Budhiraja, A. "Dropout in (Deep) Machine learning," *Medium digital publishing platform*, URL: https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5, [retrieved 30 April, 2021].

[64] Lumen Astronomy. "The Massive Atmosphere of Venus," *OpenStax CNX* [online article], URL: https://courses.lumenlearning.com/astronomy/chapter/the-massive-atmosphere-of-venus/, [retrieved 20 June, 2021].

[65] Hayling Graphics. "A video showing the motion of the planets of our Solar System 2026," *The Planets Today Store* [online animation], URL: https://www.theplanetstoday.com/solar_system_video_2026, [retrieved 26 June, 2021].

[66] The Nobles. "Demystifying LSTM Weights and Bias Dimensions," *Analytics Vidhya article*, 2020, URL: https://medium.com/analytics-vidhya/demystifying-lstm-weights-and-biases-dimensions-c47dbd39b30a, [retrieved 27 June, 2021].

[67] Mayrand-Provencher, L. "Building an Autoencoder with Tied Weights in Keras," *Medium* [online tutorial], 2019, URL: https://medium.com/@lmayrandprovencher/building-an-autoencoder-with-tied-weights-in-keras-c4a559c529a2, [retrieved 29 June, 2021].

[68] Ranjan C. “Build the right Autoencoder — Tune and Optimize using PCA principles. Part II,” *Towards data science* [online tutorial], 2019, URL: https://towardsdatascience.com/build-the-right-autoencoder-tune-and-optimize-using-pca-principles-part-ii-24b9cca69bd6, [retrieved 28 June, 2021].

[69] Kasperski, A. “How to decide the number of hidden layers and nodes in a hidden layer?,” *Research Gate* [online discussion], 2016. URL: https://www.researchgate.net/post/How-to-decide-the-number-of-hidden-layers-and-nodes-in-a-hidden-layer/581ce9f5eeae397d27799324/citation/download, [retrieved 11 July, 2021].

[70] Talaulikar, A. “How to measure the variance of a statistical model?,” *Towards data science* [online article], 2020, URL: https://towardsdatascience.com/measure-variance-of-statistical-model-e3b4725095b6, [retrieved 11 July, 2021].

[71] Radhakrishnan, P. “Bias and Variance in Neural Network,” *buZZrobot Artificial Intelligence for Human Intelligence* [online article], 2017, URL: https://buzzrobot.com/bias-and-variance-11d8e1fee627, [retrieved 11 July, 2021].

Appendices

Appendix A – Keras Python codes via Google Colab - Jupyter Notebook.
Code found at Google Drive link:
https://drive.google.com/drive/folders/1rC1SQRnTclREhKPQsQGh8idEVtmzKB_a?usp=sharing

Appendix B - Datasets used for RNN LSTM Model Training via Google Colab - Jupyter Notebook.
Spreadsheets found at Google Drive link:
https://drive.google.com/drive/folders/1puF8JPd2vouMrt7rn2yRb17iB8z_pNlE?usp=sharing

Appendix C - Datasets used to Generate the Solar Perturbation Vector Field via GMAT.
Spreadsheets found at Google Drive link: https://drive.google.com/drive/folders/1s-bLBsiyXQqUeH7MfXkKolTphC-b74G7?usp=sharing

Appendix D - Datasets Generated by Solar Perturbation Drifting via GMAT.
Spreadsheets found at Google Drive link: https://drive.google.com/drive/folders/1nocEHxOc-sEqY4iWTQGJVzAgWAkLSawm?usp=sharing

Appendix E - Datasets Generated by the Burn Analysis done in GMAT.
Spreadsheets found at Google Drive link:
https://drive.google.com/drive/folders/1Wjo7qE4Fe3M1zKSNn5oNWYWU2h9djyT6?usp=sharing

Appendix F - Nelson Wong's Thesis Abstract:
On Clustering Low-Cost SoC FPGA Devices for Deep Learning Inference Applications (tentative title).
The thesis investigates the efficacy of linking multiple sub-$100 system-on-chip field programmable gate array devices to perform inferencing. This exploration involves Xilinx's XC7Z020 and XC7Z010, which contain block RAM (BRAM) and DSP slices scattered across their programmable logic fabric. The DSP slices are leveraged for their multiply-accumulate to efficiently perform vector-matrix multiplication, while block RAM slices cache network parameters to achieve sub-millisecond multi-layer inference (results pending). I'm still far from the end of this thesis but the above should still hold true by the end. The SD card in the [SD card -> DDR memory -> BRAM cache] pipeline has been adjusted; attaching the cluster to network-attached storage and managing parameter loading over Ethernet made for a more flexible architecture. The XC7Z010 was especially targeted due to its popularity in previous-generation crypto currency miners; the Chinese online retail service AliExpress has been flooded with refurbished boards that use this chip and are very affordable (currently $16.50 each).