Polytech Nice

Project report

---

numerical solution of the optimal orbit transfer problem in the plane

---

15th April 2024

Gerbaud Florent

Table of Contents

List of Algorithms

List of Figures

1 Theory

1.1 Nomenclature

- r := satellite radial distance attracting body
- u := radial velocity
- v := tangential speed
- m := satellite mass
- $\frac{\partial m}{\partial t} = \dot{m} = -\frac{T}{g_0 \cdot Isp}$:= dm motor mass flow
- ϕ := angle of thrust direction with r
- μ := gravitational constant of the attracting body
- θ := the angle formed by r with r(0)

1.2 Initial and Final Conditions:

$$\boxed{\begin{array}{l} r(0) = r_0, \quad u(0) = 0, \quad v(0) = \sqrt{\dfrac{\mu}{r_0}} \\ r\left(t_f\right) = r_f, \quad u\left(t_f\right) = 0, \quad v\left(t_f\right) = \sqrt{\dfrac{\mu}{r_f}} \end{array}}$$

1.3 Hamiltonian and Adjoint State Dynamics:

$$\dot{x} = \begin{bmatrix} \dot{r} \\ \dot{u} \\ \dot{v} \end{bmatrix} = f(x, t, \phi) = \begin{bmatrix} u \\ \frac{v^2}{r} - \frac{\mu}{r^2} + \frac{T}{m_0 + \dot{m} \times t} \sin(\phi) \\ -\frac{uv}{r} + \frac{T}{m_0 + \dot{m} \times t} \cos(\phi) \end{bmatrix}$$

$$\lambda^T = \begin{bmatrix} \lambda_r & \lambda_u & \lambda_v \end{bmatrix}$$

$$H = 1 + \lambda^T f = 1 + \lambda_r u + \lambda_u \left(\frac{v^2}{r} - \frac{\mu}{r^2} + \frac{T}{m_0 + \dot{m} \times t} \sin(\phi) \right) + \lambda_v \left(-\frac{uv}{r} + \frac{T}{m_0 + \dot{m} \times t} \cos(\phi) \right)$$

$$\dot{\lambda} = \begin{bmatrix} \dot{\lambda}_r \\ \dot{\lambda}_u \\ \dot{\lambda}_v \end{bmatrix} = -\frac{\partial f^T}{\partial x} \lambda = \begin{bmatrix} -\lambda_u \left(-\frac{v^2}{r^2} + \frac{2\mu}{r^3} \right) - \lambda_v \left(\frac{uv}{r^2} \right) \\ -\lambda_r + \lambda_v \frac{v}{r} \\ -\lambda_u \frac{2v}{r} + \lambda_v \frac{u}{r} \end{bmatrix}$$

1.4 Control Equation

$$H = 1 + \lambda^T f = 1 + \lambda_r u + \lambda_u \left(\frac{v^2}{r} - \frac{\mu}{r^2} + \frac{T}{m_0 + \dot{m} \times t} \sin(\phi) \right) + \lambda_v \left(-\frac{uv}{r} + \frac{T}{m_0 + \dot{m} \times t} \cos(\phi) \right)$$

$$\frac{\partial H}{\partial \phi} = \lambda_u \left(\frac{T}{m_0 + \dot{m} \times t} \cos(\phi) \right) - \lambda_v \left(\frac{T}{m_0 + \dot{m} \times t} \sin(\phi) \right) = 0$$

$$\frac{\partial H}{\partial \phi} = 0 \Leftrightarrow \lambda_u \cos(\phi) - \lambda_v \sin(\phi) = 0 \Rightarrow \tan(\phi) = \frac{\lambda_u}{\lambda_v}$$

1.5 Optimality Condition on t_f

$$\left[H = 1 + \lambda^T f = 1 + \lambda_r u + \lambda_u \left(\frac{v^2}{r} - \frac{\mu}{r^2} + \frac{T}{m_0 + \dot{m} \times t} \sin(\phi) \right) + \lambda_v \left(-\frac{uv}{r} + \frac{T}{m_0 + \dot{m} \times t} \cos(\phi) \right) \right]_{t_j} = 0$$

This condition can be replaced by the equivalent condition at $t_0 = 0$:

$$\left[\lambda^T \times \lambda \right]_{t_0} = 1$$

1.6 Two-Point Boundary Value Problem Formulation

Unknowns at $t = t_0 = 0$: $\lambda(t_0)^T = [\lambda_r \ \lambda_u \ \lambda_v]_{t_0}$
Unknown at $t = t_f$: t_f System of equations to solve:

$$g(\lambda_{r0}, \lambda_{u0}, \lambda_{v0}, t_f) = \begin{bmatrix} r(t_f) - r_f \\ \\ u(t_f) \\ \\ v(t_f) - \sqrt{\frac{\mu}{r_f}} \\ \\ \left[1 + \lambda^T f \right]_{t_f} \end{bmatrix} = 0$$

2 Implementation

Algorithm 1 Problem data declaration

Data:

- **Initial Conditions:**
  - $AU = 149597870690$
  - $r_0 = AU$
  - $u_0 = 0$
  - $v_0 = \sqrt{\frac{\mu_{\text{body}}}{r_0}}$
  - $m_0 = 1000$
- **Final Conditions:**
  - $r_f = 1.5 \times AU$
  - $u_f = 0$
  - $v_f = \sqrt{\frac{\mu_{\text{body}}}{r_f}}$
- **Power**:
  - Set T to an array containing values from 0.1 to 0.6 with a step size of 0.1.
- **Earth gravity**
  - $g_0 = 9.80665$
- **Specific motor impulse**
  - $Isp = 3000$
- **sun's gravitational constant**
  - $\mu_{body} = 1.32712440018e + 20$

2.1 Problem Data declaration

In this code, we have to normalize the data due to the problem of orders of magnitude. This allows us to make the resolution easier and more accurate for the solver.

Algorithm 2 Problem data declaration

Normalized unit:

- **Unit:**
  - $DU = r_0$;
  - $VU = v_0$;
  - $MU = m_0$;
  - $TU = \frac{DU}{VU}$;
  - $FU = \frac{MU \times DU}{TU^2}$;

2.2 System Dynamics Resolution function

Algorithm 3 Dynamical model of a spacecraft

Definition of the struct:

param.$DU = DU$;
param.$VU = VU$;
param.$MU = MU$;
param.$TU = TU$;
param.$FU = FU$;
param.$\mu_{\text{body}} = \frac{\mu_{\text{body}}}{\text{param.DU}^3 \times \text{param.TU}^2}$;
param.$T = \frac{T(1)}{\text{param.FU}}$;
param.$g0_{\text{Isp}} = \frac{g0 \times Isp}{\text{param.VU}}$;
param.$m0 = \frac{m0}{\text{param.MU}}$;
param.$t0 = 0$;
param.$x_{t0} = \begin{bmatrix} \frac{r0}{\text{param.DU}} \\ \frac{u0}{\text{param.VU}} \\ \frac{v0}{\text{param.VU}} \end{bmatrix}$;
param.$x_{tf} = \begin{bmatrix} \frac{rf}{\text{param.DU}} \\ \frac{uf}{\text{param.VU}} \\ \frac{vf}{\text{param.VU}} \end{bmatrix}$;

Algorithm 4 Function `dynpol`

Input:

- t: A given time step
- $x = \begin{bmatrix} r \\ u \\ v \\ \lambda_r \\ \lambda_u \\ \lambda_v \end{bmatrix}$: the state of the dynamic system vector

function $dynpol(t, x)$

$\mu = \text{param}.\mu_{\text{body}}$

$T = \text{param}.T;$

$g0_{\text{Isp}} = \text{param}.g0_{\text{Isp}}$

$m0 = \text{param}.m0;$

$t0 = \text{param}.t0;$

$dm = -\frac{T}{g0_{\text{Isp}}}$

$\phi = \arctan\left(\frac{x(5)}{x(6)}\right)$

$dx = x;$

$dx(1) = x(2);$

$dx(2) = \frac{x(3)^2}{x(1)} - \frac{\mu}{x(1)^2} + \frac{T}{m0+dm\times t} \times \sin(\phi)$

$dx(3) = -\frac{x(2)\times x(3)}{x(1)} + \frac{T}{m0+dm\times t} \times \cos(\phi)$

$dx(4) = -x(5) \times \left(-\frac{x(3)^2}{x(1)^2} + 2 \times \frac{\mu}{x(1)^3}\right) - x(6) \times \frac{x(2)\times x(3)}{x(1)^2}$

$dx(5) = -x(4) + x(6) \times \frac{x(3)}{x(1)};$

$dx(6) = -x(5) \times (2 \times \frac{x(3)}{x(1)}) + x(6) \times \frac{x(2)}{x(1)};$

end function

Algorithm 5 Function `gnultmin`

Input:

- $p = \begin{bmatrix} t_f \\ \lambda_{r_0} \\ \lambda_{u_0} \\ \lambda_{v_0} \end{bmatrix}$: the state of the dynamic system vector

function GNMT(p)
  $t0 = \text{param}.t0;$
  $r0 = \text{param}.x_{t0}(1);$
  $u0 = \text{param}.x_{t0}(2);$
  $v0 = \text{param}.x_{t0}(3);$
  $rf = \text{param}.x_{tf}(1);$
  $uf = \text{param}.x_{tf}(2);$
  $vf = \text{param}.x_{tf}(3);$
  $y0 = [r0 \quad u0 \quad v0 \quad p(2) \quad p(3) \quad p(4)];$
  $tf = p(1);$
  $\text{atol} = 1e - 10;$
  $\text{rtol} = 1e - 10;$
  $y = \text{ode}(\text{"}rk\text{"}, y0, t0, tf, \text{dynpol}, \text{atol}, \text{rtol});$
  $gnmt = p;$
  $gnmt(1) = y(1) - rf;$
  $gnmt(2) = y(2) - uf;$
  $gnmt(3) = y(3) - \sqrt{\text{param}.\mu_{\text{body}}/rf};$
  $gnmt(4) = p(2)^2 + p(3)^2 + p(4)^2 - 1;$
end function

The function `gnultmin()` takes p as a parameter and returns the column vector $gnmt$.

Absolute and relative tolerances of 10^{-10} have been chosen for solving the differential equation using the function `ode()`. If the order of magnitude of the relative tolerance is smaller than 10^{-10}, Scilab displays a warning message and increases this tolerance to have an order of magnitude of 10^{-10}.

We aim to achieve better accuracy than that of the solver.

Algorithm 6 Control History Calculation

for $i = 1$ **to** 6 **do**
  $param.T \leftarrow \frac{T(i)}{param.FU}$
  $[x, v, info] \leftarrow \text{fsolve}(p0, gnultmin)$
  $\text{disp}(info)$
  $\text{disp}(v)$
  $InitialCond \leftarrow [\text{param.x\_t0(1)}; \text{param.x\_t0(2)}; \text{param.x\_t0(3)}; x(2); x(3); x(4)]$
  $t \leftarrow 0 : 0.01 : x(1)$
  $sol \leftarrow \text{ode}(InitialCond, param.t0, t, dynpol)$
  $timeInDays \leftarrow t \times \frac{param.TU}{86400}$
  $AngleInDegrees \leftarrow \text{atan}(sol(5, :), sol(6, :)) \times \frac{360}{2\pi}$
  figure
  $\text{plot}(timeInDays, AngleInDegrees)$
  $\text{xlabel}('Time(days)')$
  $\text{ylabel}('Controlindeg')$
  $powerInNewton \leftarrow param.T \times param.FU$
  xgrid
  $h \leftarrow \text{gca}()$
  $h.\text{background} \leftarrow \text{color}('white')$
  $\text{title}('Controlhistoryfor' + \text{string}(powerInNewton) +' N')$
end for

Each thrust (in Newtons) is contained in T. We proceed with increasing thrusts, starting with the lowest one (here 0.1), so the thrust duration decreases. The very first time the function $gnultmin()$ is passed as a parameter to the function $fsolve()$, i.e., to calculate the trajectory with the first thrust, we use p_0 for the initialization of $gnultmin()$. The result of the first use of $fsolve()$ called p_1 will serve for the second time when we pass $gnultmin()$ as a parameter to $fsolve()$, hence for calculating the trajectory with the second thrust. Thus, for each thrust except the first one, we use the result of $fsolve()$ from the previous thrust. With p_0, we can then obtain p_i for all $i \in [1; 6]$.

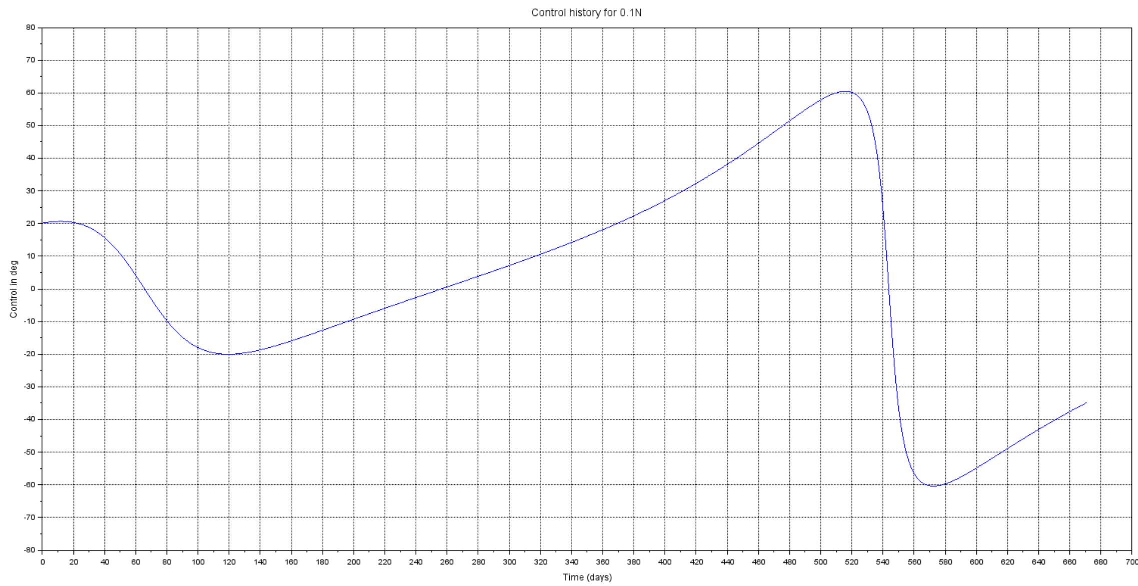

Figure 1: thrusts of 0.1N

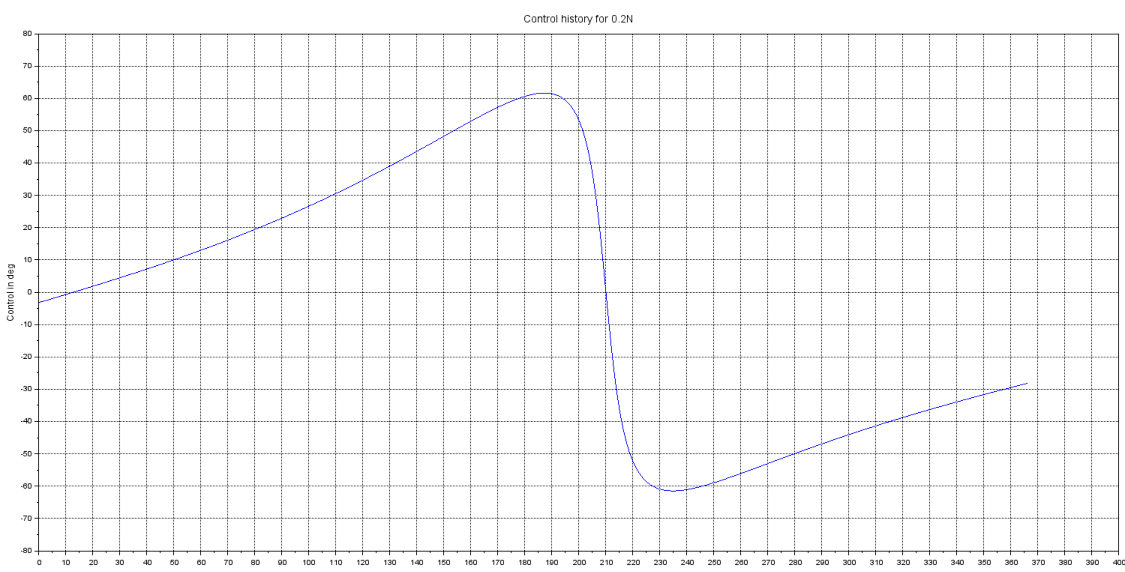

Figure 2: thrusts of 0.2N

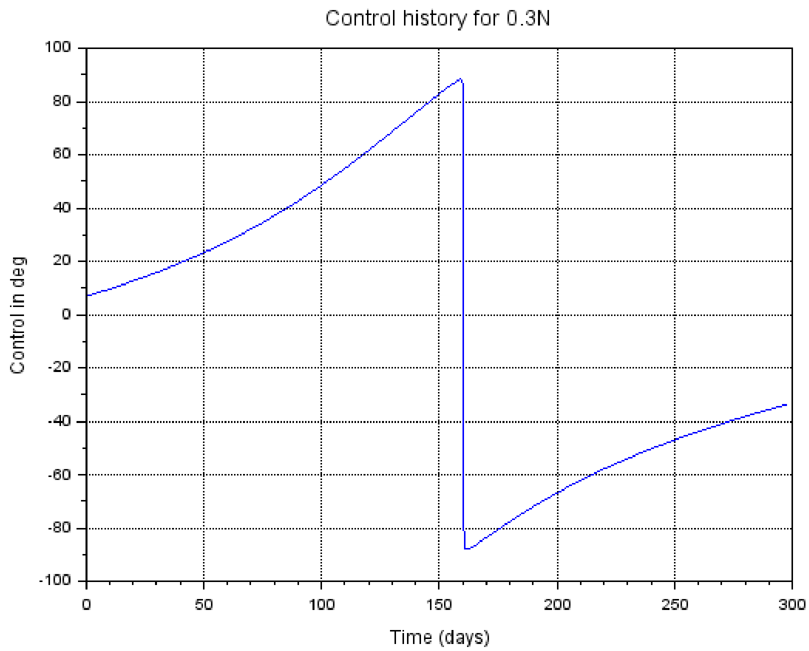

Figure 3: thrusts of 0.3N

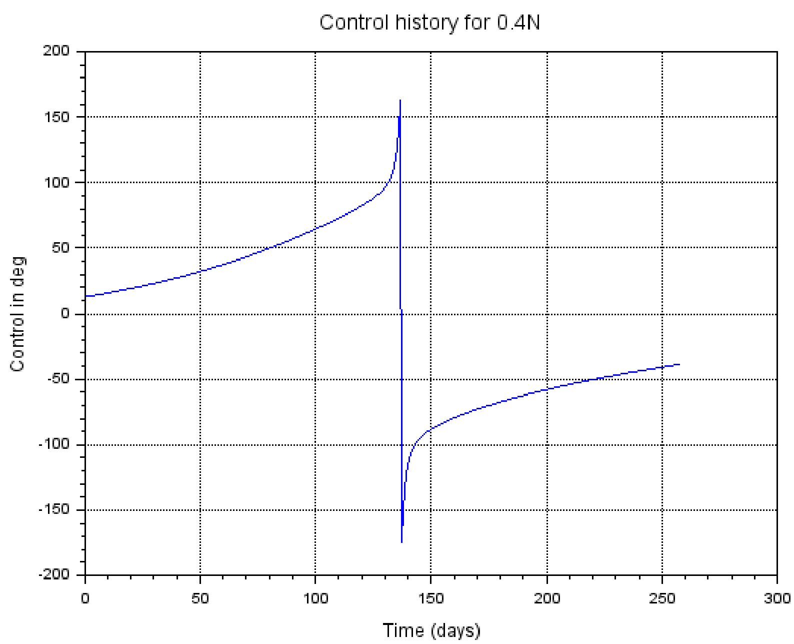

Figure 4: thrusts of 0.4N

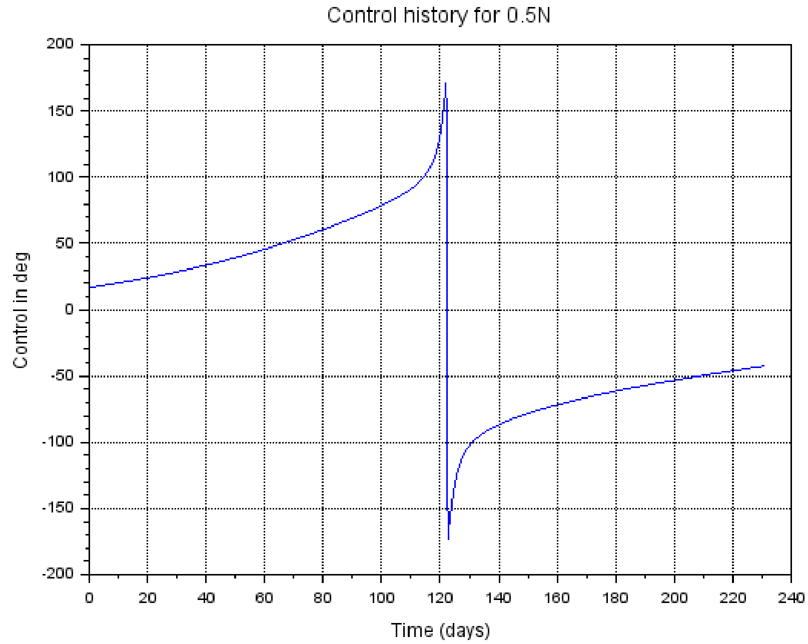

Figure 5: thrusts of 0.5N

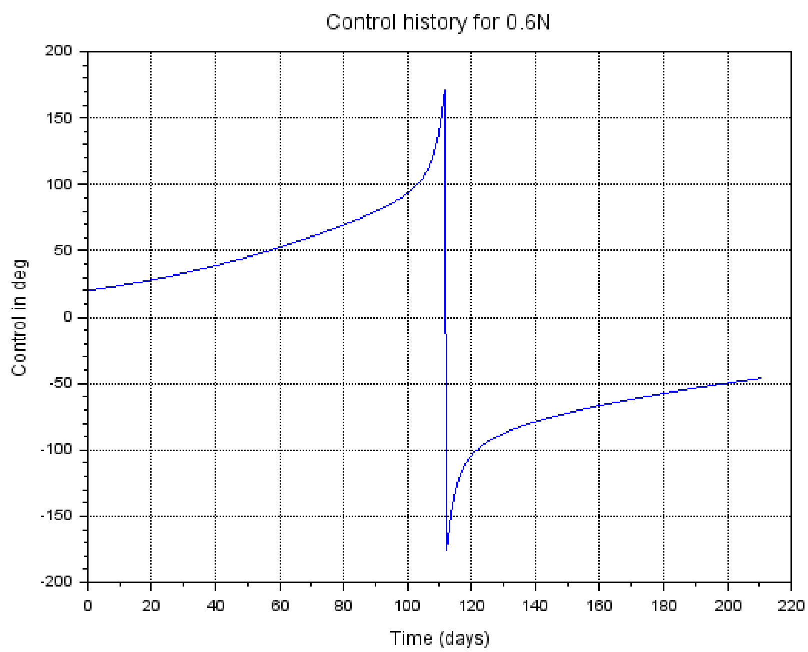

Figure 6: thrusts of 0.6N