Session 2 Shooting Method Raise Apogee (Extra Exercise 2)

Exercise: Differential Corrector/Shooting Method Raise Apogee (STM propagation). Assume a spacecraft is in an equatorial orbit with perigee altitude of 2000 km and apogee altitude of 6000 km.

Implement a differential corrector to compute the ΔV required to raise the apogee up to 12000-km altitude in half a period of the 2000x6000-km orbit. Assume the maneuver is performed at the perigee.

1. Initialize problem: define constants and info provided in statement, etc.

2. Compute parameters describing initial orbit: semi-major axis, eccentricity, mean motion, period. Compute also the velocity at perigee.

3. Define state vector (components of position and velocity) at the location of the maneuver: at perigee. These are the initial conditions of a trajectory that will be propagated along time.

```
initial_state = [0,-r_perigee,0,v_perigee,0,0]; %state at perigee
```

Note: the perigee is arbitrarily placed here along the negative y-direction.

4. Because we want to implement a differential corrector, we have to propagate along time not only the position and velocity components but also the elements of the state-transition matrix. Therefore, we also need to define the initial conditions for the state-transition matrix.

Remember that, at t_0: $\boldsymbol{\phi}(t_0, t_0)$ = identity matrix.

In MATLAB, the `eye(a)` function can be used to define an identity matrix of size `a`.

```
initial_STM = eye(6); %initial STM, 6x6 identity matrix
```

5. We will use MATLAB's ode45 function to propagate position and velocity components and the state-transition matrix along time. This function requires as input the initial condition for all of the elements to be propagated along time. In this case, we have a total of 6+6·6=42 elements that we want to propagate.

This initial condition needs to be provided as input in vector form: row vector of 42 elements.

```
initial_conditions = [initial_state, reshape(initial_STM,1,36)];
```

Notice that we are using MATLAB's function `reshape` to rearrange a 6x6 matrix into a row vector of 36 elements. `initial_conditions` is in this way a 1x42 row vector.

6. Function ode45 also requires as inputs the time of propagation, a function containing the ordinary differential equations that we want to integrate, and we can also define the tolerances for the propagation.

```
options_ode45 = odeset('AbsTol',1e-6,'RelTol',1e-9); %tolerances
time_prop = [0,TOF]; %time of propagation

[Time_out,X_out] = ode45(@(t,x)eom_2BP_with_STM(t,x,mu),
time_prop,initial_conditions,options_ode45); %call ode45 to
integrate two-body problem with STM
```

Note that the function containing the ordinary differential equations is called here `eom_2BP_with_STM(t,x,mu)`. `Time_out` and `X_out` are the outputs of the ode45 function: time vector, position and velocity components and elements of the state-transition matrix along the corresponding time vector.

7. Function `eom_2BP_with_STM` should output the time derivative of the 42 elements to be propagated, `dx_dt`. The inputs of this function are:
- the time at which we want to compute the those time derivatives, `t`,
- the 42 elements evaluated at time `t`, `x`, and
- the central body's gravitational parameter, `mu`.

```
function [dx_dt] = eom_2BP_with_STM(t,x,mu)
```

Vector `dx_dt` is therefore another vector with 42 elements, in this case, `dx_dt` should be a column vector. The first 6 elements in `dx_dt` are the time derivatives of the position and velocity components, and the remaining 36 are the time derivatives of the elements in the state-transition matrix.

In the two-body problem, we have as equations of motion:

$$\dot{\boldsymbol{r}} = \boldsymbol{v}$$

$$\dot{\boldsymbol{v}} = -\frac{\mu}{|\boldsymbol{r}|^3}\boldsymbol{r}$$

These are the first 6 elements in `dx_dt`.

The time derivative of the state-transition matrix is such that:

$$\dot{\boldsymbol{\phi}}(t, t_0) = \boldsymbol{A}\left(x_{ref}(t), (t)\right) \cdot \boldsymbol{\phi}(t, t_0)$$

where $\boldsymbol{A}(\boldsymbol{x}(t), t) = \begin{bmatrix} \boldsymbol{0}_{3x3} & \boldsymbol{I}_{3x3} \\ \boldsymbol{A}_{21} & \boldsymbol{0}_{3x3} \end{bmatrix}$ and:

$$\boldsymbol{A}_{21} = \begin{bmatrix} -\frac{\mu}{|\boldsymbol{r}|^3} + \frac{3\mu x^2}{|\boldsymbol{r}|^5} & \frac{3\mu xy}{|\boldsymbol{r}|^5} & \frac{3\mu xz}{|\boldsymbol{r}|^5} \\ \frac{3\mu xy}{|\boldsymbol{r}|^5} & -\frac{\mu}{|\boldsymbol{r}|^3} + \frac{3\mu y^2}{|\boldsymbol{r}|^5} & \frac{3\mu yz}{|\boldsymbol{r}|^5} \\ \frac{3\mu xz}{|\boldsymbol{r}|^5} & \frac{3\mu yz}{|\boldsymbol{r}|^5} & -\frac{\mu}{|\boldsymbol{r}|^3} + \frac{3\mu z^2}{|\boldsymbol{r}|^5} \end{bmatrix}$$

Once the 3x3 matrix $\boldsymbol{A}_{21}$ is computed, `A_21`, matrix $\boldsymbol{A}(\boldsymbol{x}(t), t)$ can be constructed such as:

```
A = [zeros(3), eye(3); A_21, zeros(3)];
```

The elements of the state-transition matrix at the current time, $\boldsymbol{\phi}(t, t_0)$, are contained in vector `x`: `x(7:end)`, since the first 6 elements in `x` are those corresponding to the position and velocity components.

The time derivative of the state-transition matrix, $\dot{\boldsymbol{\phi}}(t, t_0)$, can therefore be computed as:

```
STM = reshape(x(7:end),6,6); %state-transition matrix
dSTM_dt = A*STM; %time derivative of state-transition matrix
```

Note that the state-transition matrix is rearranged first into a 6x6 matrix using function `reshape`.

Finally, the 42-element column vector `dx_dt` can be constructed as:

```
dx_dt = [dr_dt_and_dv_dt; reshape(dSTM_dt,36,1)];
```

Notice that the 6x6 `dSTM_dt` matrix is rearranged into a column vector of 36 elements using function `reshape`. In this way, `dx_dt` is a column vector with 42 elements.

8. Calling function `eom_2BP_with_STM` with function ode45 (as in step 6) allows us to propagate not only the position and velocity but also the state-transition matrix along time. As aforementioned, the outputs of the ode45 function are the time vector, `Time_out`, and the components of position and velocity and elements of the state-transition matrix along time, in `X_out`.

Assuming that `Time_out` vector has n elements (from time $t_0 = 0$ to time $t_1 = TOF$), then `X_out` is a matrix of dimensions n by 42.

The last row of `X_out` thus contains the state vector at time t_1: $\boldsymbol{r}(t_1)$ and $\boldsymbol{v}(t_1)$, and the elements of the state-transition matrix $\boldsymbol{\phi}(t_1, t_0)$:

```
r_t1 = X_out(end,1:3); %position at t_1
v_t1 = X_out(end,4:6); %velocity at t_1
STM_t0_to_t1  =  reshape(X_out(end,7:end),6,6);  %STM  from  t_0=0  to
t_1=TOF
```

Note that the 6x6 matrix $\boldsymbol{\phi}(t_1, t_0)$ is constructed by rearranging the corresponding elements in matrix `X_out` through function `reshape`.

9. Once we know the final position, we can compute how far we are from the desired final position:

```
target_position = [0,target_apogee,0]; %target position
error_vector = target_position-r_t1;
```

Note that the goal in this exercise is to raise the apogee to a desired value `target_apogee`, and the apogee is placed along the positive y-direction given that the perigee was defined along the negative y-direction in step 3.

10. A differential corrector consists in computing the required change in initial conditions to achieve a desired change in final conditions. The state-transition matrix relates these quantities.

In this case, we want the change in initial velocity to achieve a certain final position.

$$\delta\boldsymbol{r}(t_1) = \boldsymbol{\phi}_{rv}(t_1, t_0) \cdot \delta\boldsymbol{v}(t_0) = \boldsymbol{K} \cdot \delta\boldsymbol{v}(t_0)$$

$$\delta\boldsymbol{v}(t_0) = \boldsymbol{K}^T(\boldsymbol{K}\boldsymbol{K}^T)^{-1}\delta\boldsymbol{r}(t_1)$$

Notice that we are only interested in the 3x3 block $\boldsymbol{K} = \boldsymbol{\phi}_{rv}(t_1, t_0)$ within the 6x6 state-transition matrix $\boldsymbol{\phi}(t_1, t_0)$, which relates initial velocity and final position.

$$\boldsymbol{\phi}(t_1, t_0) = \begin{bmatrix} \boldsymbol{\phi}_{rr} & \boldsymbol{\phi}_{rv} \\ \boldsymbol{\phi}_{vr} & \boldsymbol{\phi}_{vv} \end{bmatrix}$$

In MATLAB, we should only select the elements of interest from matrix `STM_t0_to_t1`:

```
K = STM_t0_to_t1(1:3,4:5);
```

And finally, the required change in initial velocity, $\delta\boldsymbol{v}(t_0)$, can be computed based on the desired change in final position $\delta\boldsymbol{r}(t_1)$ (here called `error_vector`):

```
dv0 = K'*inv(K*K')*error_vector'; %required change in initial
velocity
```

Notice that `error_vector` is transposed from a row vector to a column vector, and it represents the desired change in final position, $\delta\boldsymbol{r}(t_1)$.

11. Once the required change in initial velocity is computed, new initial conditions for the trajectory can be computed, `initial_state`, and the ode45 function can be employed again to propagate the trajectory and the state-transition matrix with these new initial conditions.

Repeating steps 8 through 10 with these new initial conditions, we will obtain new final conditions `r_t1` and `v_t1`, a new state transition matrix `STM_t0_to_t1`, a new vector `error_vector`, and, eventually, a new required change in the initial velocity `dv0`.

This process should be repeated until the difference between the resulting and the desired final state, $|\delta\boldsymbol{r}(t_1)|$ or `norm(error_vector)`, is below a certain threshold, or until the maximum allowed number of iterations is achieved.

```
while norm(error_vector)>tolerance && itr<allowed_number_of_itr
```

12. The overall structure of the differential corrector is, therefore (for this particular exercise):

```
%Define constants
...

%Define initial conditions
...

%Define time of propagation
...

%Call ode45 to obtain final state and state-transition matrix
...

%Compute difference between desired final state and resulting final
state
...

%While-loop to iterate on the initial conditions, until convergence
is achieved
while norm(error_vector)>tolerance && itr<allowed_number_of_itr

%Compute required change in initial conditions
...

%Define new initial conditions
...

%Define time of propagation
...
```

```
%Call ode45 to obtain final state and state-transition matrix
...

%Compute difference between desired final state and resulting final
state
...

end %end of while-loop
```

Exercise: Differential Corrector/Shooting Method Raise Apogee (F and G coefficients). Assume a spacecraft is in an equatorial orbit with perigee altitude of 2000 km and apogee altitude of 6000 km.

Implement a differential corrector to compute the ΔV required to raise the apogee up to 12000-km altitude in half a period of the 2000x6000-km orbit. Assume the maneuver is performed at the perigee.

1. Initialize problem: define constants and info provided in statement, etc.

2. Compute parameters describing initial orbit: semi-major axis, eccentricity, mean motion, period. Compute also the velocity at perigee.

3. Define position and velocity vectors at the time of maneuver: at perigee.

```
r0 = [0;-r_perigee;0]; %initial position at periapsis
v0 = [v_perigee;0;0];  %initial velocity at periapsis
```

Note: the perigee is arbitrarily placed here along the negative y-direction.

4. Define target position (apogee at a certain altitude).

```
target_pos = [0; target_apogee]; %target position, x and y at
apoapsis
```

Note that the goal in this exercise is to raise the apogee to a desired value `target_apogee`, and the apogee is placed along the positive y-direction given that the perigee was defined along the negative y-direction in step 3.

Also, notice that only the x and y components of the target position are specified, since the motion is assumed to be on the x-y plane.

5. Because we are working on the two-body problem, the final position and velocity can be analytically expressed in terms of the initial conditions through the F and G coefficients. No numerical integration of the equations of motion is required.

$$\boldsymbol{r}_f = F\boldsymbol{r}_0 + G\dot{\boldsymbol{v}}_0$$

$$\boldsymbol{v}_f = \dot{F}\boldsymbol{r}_0 + \dot{G}\dot{\boldsymbol{v}}_0$$

where

$$F = 1 - \frac{a}{r_0}(1 - \cos(\Delta E))$$

$$G = \Delta t + \sqrt{\frac{a^3}{\mu}}(\sin(\Delta E) - \Delta E)$$

$$\dot{F} = -\frac{\sqrt{\mu a}}{r_f r_0}\sin(\Delta E)$$

$$\dot{G} = 1 - \frac{a}{r_f}(1 - \cos(\Delta E))$$

$$a = \frac{\mu}{\frac{2\mu}{r_0} - v_0^2}$$

Notice that the change in eccentric anomaly, ΔE, can be obtained from Kepler's equation for a given time-of-flight (through MATLAB's fzero function or implementing a Newton-Raphson algorithm).

$$M = n\Delta t = n \cdot TOF = \Delta E - e \cdot sin(\Delta E) = \Delta E - \left(1 - \frac{r_0}{a}\right)\sin(\Delta E) - \frac{\sigma_0}{\sqrt{a}}(\cos(\Delta E) - 1)$$

where

$$n = \sqrt{\frac{\mu}{a^3}}$$

$$\sigma_0 = \frac{\boldsymbol{r}_0 \cdot \dot{\boldsymbol{r}}_0}{\sqrt{\mu}}$$

For instance, In MATLAB, we can compute ΔE using the fzero function:

```
%Transfer - Change in eccentric anomaly
sigma0 = dot(r0,v0)/sqrt(mu);
deltaM = n*TOF;
deltaE = fzero(@(deltaE) deltaM -...
    (deltaE - (1-norm(r0)/a)*sin(deltaE) -...
    sigma0/sqrt(a)*(cos(deltaE)-1)),deltaM); %change in eccentric
anomaly for transfer
```

Because we know the initial position and velocity, we can thus compute the final position through the F and G coefficients:

```
%Final position for given initial conditions
rf = F*r0 + G*v0;
```

6. Once we know the final position, we can compute how far we are from the desired final position:

```
%Difference between desired final position and resulting final
position
error_vector = target_pos-rf(1:2);
```

Notice that we are only selecting the x and y components of the resulting final position, as the motion occurs in the x-y plane.

7. A differential corrector consists in computing the required change in initial conditions to achieve a desired change in final conditions. The state-transition matrix relates these quantities. In this case, we want to know the required change in initial velocity to achieve a desired final position at time t_1.

Notice that we are only interested in the 3x3 block $\boldsymbol{\phi}_{rv}(t_1, t_0)$ within the 6x6 state-transition matrix $\boldsymbol{\phi}(t_1, t_0)$, which relates initial velocity (at time t_0) and final position (at time t_1).

$$\boldsymbol{\phi}(t_1, t_0) = \begin{bmatrix} \boldsymbol{\phi}_{rr} & \boldsymbol{\phi}_{rv} \\ \boldsymbol{\phi}_{vr} & \boldsymbol{\phi}_{vv} \end{bmatrix}$$

$$\delta \boldsymbol{r}_f = \delta \boldsymbol{r}(t_1) = \boldsymbol{\phi}_{rv}(t_1, t_0) \cdot \delta \boldsymbol{v}(t_0) = \boldsymbol{K} \cdot \delta \boldsymbol{v}(t_0)$$

$$\delta \boldsymbol{v}_0 = \delta \boldsymbol{v}(t_0) = \boldsymbol{K}^T (\boldsymbol{K}\boldsymbol{K}^T)^{-1} \delta \boldsymbol{r}(t_1)$$

In the two-body problem, the $\boldsymbol{\phi}_{rv}$ 3x3 matrix can also be computed analytically as:

$$\boldsymbol{\phi}_{rv}(t_1,t_0)=\frac{\delta \boldsymbol{r}(t_1)}{\delta \boldsymbol{v}(t_0)}=\frac{\delta \boldsymbol{r}_f}{\delta \boldsymbol{v}_0}=\frac{r_0}{\mu}(1-F)(\Delta \boldsymbol{r}\cdot \dot{\boldsymbol{r}}_0^T-\Delta \boldsymbol{v}\cdot \boldsymbol{r}_0^T)+\frac{C}{\mu}\dot{\boldsymbol{r}}_f\cdot \dot{\boldsymbol{r}}_0^T+G\cdot \boldsymbol{I}_{3x3}$$

where

$$\Delta \boldsymbol{r}=\boldsymbol{r}_f-\boldsymbol{r}_0$$

$$\Delta \boldsymbol{v}=\boldsymbol{v}_f-\boldsymbol{v}_0$$

$$C=a\sqrt{\frac{a^3}{\mu}}(3\sin(\Delta E)-(2+\cos(\Delta E))\Delta E)-a\Delta t(1-\cos(\Delta E))$$

In MATLAB:

```
deltar = rf-r0;
deltav = vf-v0;
I_3by3 = eye(3); %3x3 identity matrix

%State-transition matrix relating final position and initial
velocity
STM_rf_v0 = norm(r0)/mue*(1-F)*(deltar*v0'-deltav*r0') +...
                C/mue*vf*v0' + G*I_3by3;
```

Notice that MATLAB's eye() command is used to construct a 3x3 identity matrix. Also notice that vectors are transposed from their column form to row vectors through the command `'` where required.

And finally, the required change in initial velocity, $\delta \boldsymbol{v}(t_0)$, can be computed based on the desired change in final position $\delta \boldsymbol{r}(t_1)$ (here called `error_vector`):

```
K_xy = STM_rf_v0(1:2,1:2); %Select only components associated to x
and y coordinates

dv0_xy = K_xy'*inv(K_xy*K_xy')*error_vector; %required change in
initial velocity
```

Notice that we only select the sub-block in matrix $\boldsymbol{\phi}_{rv}(t_1,t_0)$ associated to the x and y components, since the motion occurs in the x-y plane.

8. Once the required change in initial velocity is computed, a new initial velocity for the trajectory can be computed:

```
%Update initial velocity
v0(1:2) = v0(1:2)+dv0;
```

Repeating steps 5 through 7 with these new initial conditions, we will obtain a new final position `rf`, a new vector `error_vector`, a new state transition matrix `STM_rf_v0`, and, eventually, a new required change in the initial velocity `dv0`.

This process should be repeated until the difference between the resulting and the desired final state, $|\delta \boldsymbol{r}(t_1)|$ or `norm(error_vector)`, is below a certain threshold, or until the maximum allowed number of iterations is achieved.

```
while norm(error_vector)>tolerance && itr<allowed_number_of_itr
```

9. The overall structure of the differential corrector is, therefore (for this particular exercise):

```matlab
%Define constants
...

%Define time-of-flight
...

%Define target position
...

%Define initial conditions
...

%Compute F and G coefficients
...

%Compute final position
...

%Compute difference between desired final state and resulting final
state
...

%While-loop to iterate on the initial conditions, until convergence
is achieved
while norm(error_vector)>tolerance && itr<allowed_number_of_itr

%Compute state transition matrix through F and G coefficients
...

%Compute required change in initial velocity
...

%Define new initial velocity
...

%Compute F and G coefficients
...

%Compute final position
...

%Compute difference between desired final state and resulting final
state
...

end %end of while-loop
```