HOMEWORK: EXERCISE 2 - Space mining mission

It is March 20, 2053. A space mining mission has been established in Europa, Jupiter's moon. From there, they intend to visit all other major moons in the search for valuable resources for space colonization. The first target is Ganymede, the next of the four Galilean moons.

As the mission commanders, you must use the patched-conics method and Lambert's algorithm with elliptic trajectories to determine the best departure time to Ganymede, between the present date and for the next two weeks, and best flight duration, to achieve the lowest mission ΔV .

(a) Implement the following functions, using units of km, s, and rad for the inputs and outputs:

i. `[a,e,Omega,i,omega,theta]=rv2coe(r,v,mu)` ,

ii. `[r,v] = coe2rv(a,e,Omega,i,omega,theta,mu)` ,

iii. `[E] = theta2E(theta,e)` ,

iv. `[theta] = E2theta(E,e)` ,

v. `[M] = E2m(E,e)` ,

vi. `[E] = M2E(M,e,E0,maxiter,tol)` ,

for the elliptic two body problem, as we have done in class. Note that in the last one you need to implement a solver. Use Newton-Raphson's method.

These are the equivalent functions we would have with AstrodynamicsEdu:

i. `coe = stateVector_to_COE(stateVector,mu)`

ii. `stateVector = COE_to_stateVector(coe,mu)`

iii. `E = trueAnomaly_to_eccentricAnomaly(stateVector,mu,theta)`

iv. `theta = eccentricAnomaly_to_trueAnomaly(stateVector,mu,E)`

v. `Me = eccentricAnomaly_to_meanAnomaly(stateVector,mu,E)`

vi. `E = meanAnomaly_to_eccentricAnomaly(stateVector,mu,E)`

In [ ]:
```julia
using Pkg
# Pkg.activate(".")
Pkg.activate("C:\\Users\\alici\\Desktop\\Julia\\HOMEWORK")
Pkg.instantiate()
Pkg.add(url="https://github.com/AliciaSBa/AstrodynamicsEdu.jl.git")

using Roots
using AstrodynamicsEdu
using LinearAlgebra
```

```
 Activating project at `C:\Users\alici\Desktop\Julia\HOMEWORK`
   Updating git-repo `https://github.com/AliciaSBa/AstrodynamicsEdu.jl.git`
   Updating registry at `C:\Users\alici\.julia\registries\General.toml`
  Resolving package versions...
No Changes to `C:\Users\alici\Desktop\Julia\HOMEWORK\Project.toml`
No Changes to `C:\Users\alici\Desktop\Julia\HOMEWORK\Manifest.toml`
```

(b) Use the NASA JPL Horizons ephemeris server to find the position and velocity of Europa and Ganymede with respect to Jupiter, in a jupiter-centered, ecliptic, pseudoinertial reference frame, at 00:00 March 20, 2053. This will be t = 0; the state of the moons at later times shall be computed using the 2BP functions from the previous point. You should also use Horizons to find the gravitational parameter μ of Europa and Ganymede (they are not available in the Course Formulary).

The radius and gravitational parameter of Jupiter are already in the AstrodynamicsEdu library as `mu_Jupiter` and `R_Jupiter`

In [ ]:
```
# Gravitational parameters and radius of the moons of Jupiter
    # (that do not appear in the AstroConstants module)
mu_Europa = 3202.71 # km^3/s^2
mu_Ganymede = 9887.83 # km^3/s^2
R_Europa = 1560.8 # km
R_Ganymede = 2631.2 # km

# Positions of the moons with respect to Jupiter at the specified date
    # (20 March 2053 00:00)
r0_Europa = MyVector([3.660674324064600E+05, -5.574382118933767E+05,
                                -1.130028161238341E+04]) # km
v0_Europa = MyVector([1.160430186118396E+01, 7.496163341137031E+00,
                                5.325632226511305E-01])  # km/s
state0_Europa = MyStateVector(r0_Europa,v0_Europa)
r0_Ganymede = MyVector([-1.574197237263062E+05, 1.056563893373634E+06,
                                3.441445759743074E+04]) # km
v0_Ganymede = MyVector([-1.077978381266393E+01, -1.575638346889765E+00,
                                -2.390760437391640E-01]) # km/s
state0_Ganymede = MyStateVector(r0_Ganymede,v0_Ganymede)
```

```
MyStateVector(MyVector([-157419.7237263062, 1.056563893373634e6, 34414.45759743074]), MyVector
([-10.77978381266393, -1.575638346889765, -0.239076043739164]))
```

(c) Implement a function `[v1,v2] = Lambert(r1,r2,t,s,mu,maxiter,tol)` that solves the Lambert problem for two position vectors r1, r2 in km and a time of flight t in s. In this function, s is a flag that takes the value +1 for a short arc transfer and -1 for a long arc transfer, mu is the gravitational parameter of the 2BP, maxiter is the maximum number of iterations to consider in the solver, and tol is the solver tolerance. The function shall return the velocity vectors v1 and v2 in km/s at the beginning and the end of the transfer. It shall only solve for elliptic transfers, and raise an error if no elliptic transfer is possible for the given time of flight.

The functions that we already have are:

`coe1,coe2 = Lambert_solve(r1,r2,tF,mu,k)`

`coe1,coe2 = Lambert_conic(r1,r2,eT,k)`

`err = error_function(mu, r1, r2, eT, tf, k)`

The functions that we need:

`v1,v2 = Lambert(r1,r2,tF,s,mu,maxiter,tol)`

note: s = +1(short arc) or -1(long arc)
v1: velocity vector @ beginning transfer
v2: velocity vector @ end transfer
Only solves for elliptic transfers.

We would use: `coe1,coe2 = Lambert_conic(r1,r2,eT,k)`
`stateVector1 = COE_to_stateVector(coe1,mu)`
`v1 = stateVector.v`
`stateVector2 = COE_to_stateVector(coe2,mu)`
`v2 = stateVector.v`

In [ ]:
```
function Lambert(r1::MyVector,r2::MyVector,tF::Float64,s,mu::Float64,
                 maxiter::Number,tol::Number)
    # Input:
        # r1: initial position vector in the inertial frame
        # r2: final position vector in the inertial frame
        # tF: time of flight
        # s: Boolean that activates the short way when true (1 or -1)
        # mu: gravitational parameter of the central body
        # maxiter: maximum number of iterations
        # tol: tolerance for the root-finding algorithm
    # Output:
        # v1: velocity vector at the beginning of the transfer arc
        # v2: velocity vector at the end of the transfer arc

    # Would need to modify Lambert_solve to take in the s,maxiter,tol arguments

    function error_function(mu, r1, r2, eT, tf, k)
        coe1, coe2 = Lambert_conic(r1, r2, eT, k)
        err = tf - timeOfFlight(coe1, coe2, mu)
        return err
    end

    function Lambert_solve2(r1::MyVector,r2::MyVector,tF::Float64,mu::Float64,
                 k::MyVector,tol::Number,maxiter::Number)
        # Input:
            # r1: initial position vector
            # r2: final position vector
            # tF: time of flight
            # mu: gravitational parameter
            # K: plane normal (only used when rr1 and rr2 are collinear)
        # Output:
            # coe1: Classical Orbital Elements of the connecting orbit at the initial time
            # coe2: Classical Orbital Elements of the connecting orbit at the final time

        #tol = 1.0e-8
        #maxiter = 1000
        # Calculate the fundamental eccentricity
        r1_norm = norm(r1)
        r2_norm = norm(r2)
        c = norm(r2 - r1)
        eF = -(r2_norm - r1_norm)/c
        eTP = sqrt(1.0 - eF^2)

        # Solve for the transverse eccentricity using the bisection method
        try
            f(eT) = error_function(mu,r1,r2,eT,tF,k)
            eT = find_zero(f, (-eTP*0.999,eTP*0.999), xtol=tol, maxevals=maxiter)
            coe1, coe2 = Lambert_conic(r1,r2,eT,k)
            return coe1, coe2
        catch
            #println("No elliptic transfer was found")
```

```
            coe1 = MyCOE(RAAN=NaN, inc=NaN, e=NaN, omega=NaN, theta=NaN, a=NaN)
            coe2 = MyCOE(RAAN=NaN, inc=NaN, e=NaN, omega=NaN, theta=NaN, a=NaN)
            return coe1, coe2
        end

    end

    coe1,coe2 = Lambert_solve2(r1,r2,tF,mu,k0,tol,maxiter)

    stateVector1 = COE_to_stateVector(coe1,mu)
    stateVector2 = COE_to_stateVector(coe2,mu)
    v1 = stateVector1.v
    v2 = stateVector2.v

    return v1,v2
end
```

```
Lambert (generic function with 1 method)
```

(d) Use the Lambert function to produce a porkchop plot of the characteristic energy v_1,rel^2 + v_2,rel^2 in km^2/s^2 as a function of the departure time and the time of flight in the considered window. Discuss your results and identify a preferred option for the trip.

The objective is to determine the transfer orbit with the lowest characteristic energy between Europa and Ganymede. To achieve this, a sensitivity analysis is performed by varying the departure date and travel time. For each combination, the Lambert function will be used, which solves for the velocity at the beginning and at the end of the transfer orbit. Because of the dependency on time of the positions, to compute the new state vectors in each instance of time, the relationships between the anomalies and time will be used.

In the calculation of the relative energy required for the transfer, the short arc and long arc transfers are compared, and the option with the minimum energy is selected for each combination of launch and travel times.

In [ ]:
```
# Initial true anomaly of the moons
coe0_Europa = stateVector_to_COE(state0_Europa,mu_Jupiter)
coe0_Ganymede = stateVector_to_COE(state0_Ganymede,mu_Jupiter)

# Initial mean anomaly of each moon
M0_Europa = trueAnomaly_to_meanAnomaly_E(state0_Europa, mu_Jupiter,
                coe0_Europa.theta)
M0_Ganymede = trueAnomaly_to_meanAnomaly_E(state0_Ganymede, mu_Jupiter,
                coe0_Ganymede.theta)

# Mean angular rate of each moon
n_Europa = sqrt(mu_Jupiter/coe0_Europa.a^3)
n_Ganymede = sqrt(mu_Jupiter/coe0_Ganymede.a^3)

# Set the loop that will solve Lambert's problem for each travel time
maxiter = 1000
tol = 1e-8
t_launch = range(0,stop=14,length=200)*24*3600 # s
t_travel = range(1,stop=8,length=1000)*24*3600 # s

# Initialize the vectors
pork_plot = zeros(length(t_travel),length(t_launch))
r_Europa = zeros(3,length(t_launch))
v_Europa = zeros(3,length(t_launch))
r_Ganymede = zeros(3,length(t_launch),length(t_travel))
v_Ganymede = zeros(3,length(t_launch),length(t_travel))
```

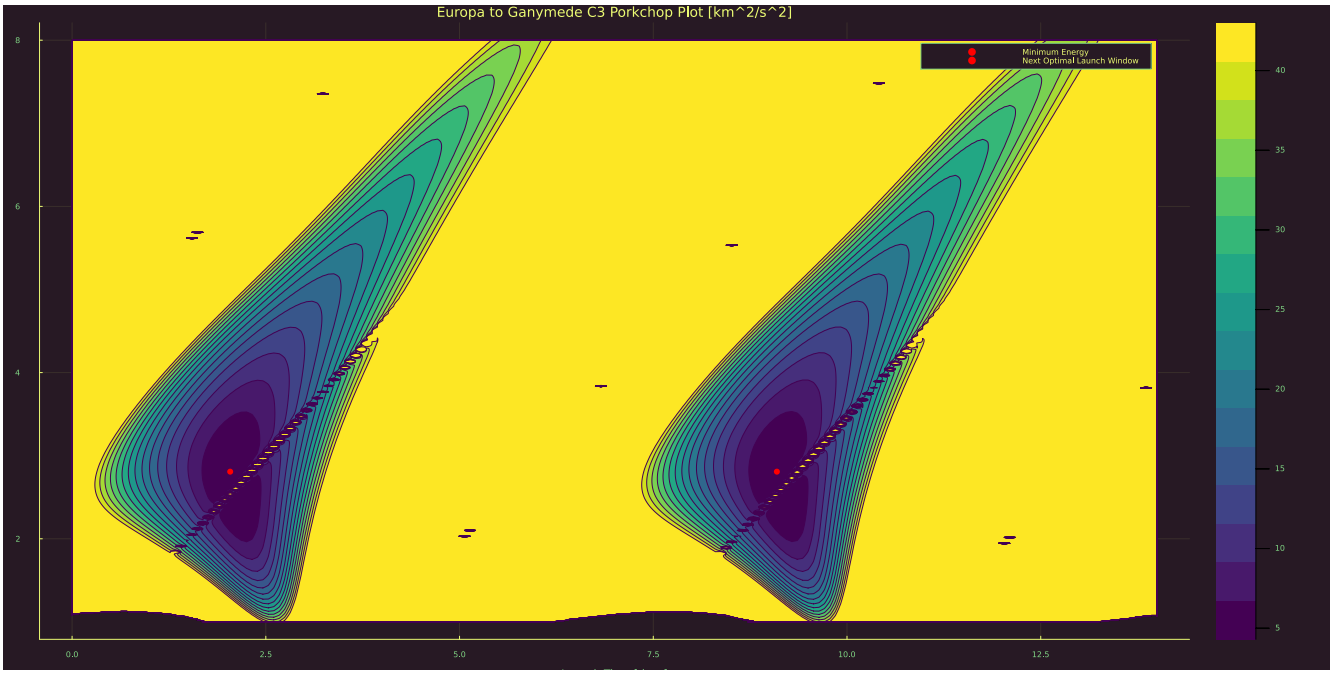
```
for i=1:length(t_launch)
    # Initial positions of Europa
    M1_Europa = M0_Europa + t_launch[i]*n_Europa
    theta1_Europa = meanAnomaly_to_trueAnomaly_E(state0_Europa,mu_Jupiter,
                            M1_Europa)
    coe1_Europa = MyCOE(RAAN=coe0_Europa.RAAN, inc=coe0_Europa.inc,
                        e=coe0_Europa.e, omega=coe0_Europa.omega,
                        theta=theta1_Europa, a=coe0_Europa.a)
    state1_Europa = COE_to_stateVector(coe1_Europa,mu_Jupiter)
    r_Europa[:,i] = state1_Europa.r.c0
    v_Europa[:,i] = state1_Europa.v.c0

    # Iterate for different travel times
    for j=1:length(t_travel)
        # Final positions of Ganymede
        M1_Ganymede = M0_Ganymede + (t_travel[j]+t_launch[i])*n_Ganymede
        theta1_Ganymede = meanAnomaly_to_trueAnomaly_E(state0_Ganymede,mu_Jupiter,
                                M1_Ganymede)
        coe1_Ganymede = MyCOE(RAAN=coe0_Ganymede.RAAN, inc=coe0_Ganymede.inc,
                                e=coe0_Ganymede.e, omega=coe0_Ganymede.omega,
                                theta=theta1_Ganymede, a=coe0_Ganymede.a)
        state1_Ganymede = COE_to_stateVector(coe1_Ganymede,mu_Jupiter)
        r_Ganymede[:,i,j] = state1_Ganymede.r.c0
        v_Ganymede[:,i,j] = state1_Ganymede.v.c0

        # Solve Lambert's problem for the short distance and long distance cases
        vs1_Europa, vs1_Ganymede = Lambert(MyVector(r_Europa[:,i]),
                MyVector(r_Ganymede[:,i,j]),t_travel[j],1,mu_Jupiter,maxiter,tol)
        vns1_Europa, vns1_Ganymede = Lambert(MyVector(r_Europa[:,i]),
                MyVector(r_Ganymede[:,i,j]),t_travel[j],-1,mu_Jupiter,maxiter,tol)

        C3s1 = norm(vs1_Europa.c0 - v_Europa[:,i])^2 +
                norm(vs1_Ganymede.c0 - v_Ganymede[:,i,j])^2
        C3ns1 = norm(vns1_Europa.c0 - v_Europa[:,i])^2 +
                norm(vns1_Ganymede.c0 - v_Ganymede[:,i,j])^2
        try
            if C3s1 < C3ns1
                pork_plot[j,i] = C3s1
            else
                pork_plot[j,i] = C3ns1
            end
        catch
            pork_plot[j,i] = NaN
        end
    end
end

# Find the minimum energy and get the launch time and travel time
minC3 = minimum(skipmissing(isnan(x) ? missing : x for x in pork_plot))
println("Minimum value:", minC3)
positions = findall(x -> x == minC3, pork_plot)
println("Positions: ", positions)

t_launch_min = t_launch[positions[1][2]]/(24*3600)
t_travel_min = t_travel[ positions[1][1]]/(24*3600) # days

# Calculate the synodic period to determine the next optimal launch window
t_synodic = 2*pi/(abs(n_Europa - n_Ganymede))/(24*3600) # days
t_launch_next = t_launch_min + t_synodic
t_travel_next = t_travel_min
```

```
Minimum value:4.3003262737889685
Positions: CartesianIndex{2}[CartesianIndex(259, 30)]
```

```
2.8078078078078077
```

The analysis results are presented graphically in a Porkchop plot, which illustrates the characteristic energy for different combinations of departure date and travel time. We can use the `plot_porkchop` function of AstrodynamicsEdu.

In [ ]:
```
# Plot the contour of the relative energy
figure = plot_porkchop(t_launch/(24*3600),t_travel/(24*3600),pork_plot,
                           t_synodic)
title!("Europa to Ganymede C3 Porkchop Plot [km^2/s^2]")
xlabel!("Launch Time [days]")
ylabel!("Travel Time [days]")
display(figure)
```



For the reminder of the problem, consider your preferred option for the trip:

(e) For the departure from Europa, assume that your launchpad is on the moon's equator at the datum. Determine the ΔV to launch into a circular parking orbit 200 km high, ignoring any atmosphere. Then, compute the semimajor axis and eccentricity of the departure hyperbola, and the ΔV for injection. You can assume that the launch time and the plane of the parking orbit allow a direct tangential burn into the departure hyperbola.

1st Maneuver: From Europa surface to a circular parking orbit we would need 2 burns. The first for going into a Hohmann transfer and the second to circularize the orbit.

$$\Delta v_1 = \sqrt{\frac{2\mu}{R_{Europa}} - \frac{\mu}{a_{tr}}}$$

$$\Delta v_2 = \sqrt{\frac{\mu}{r_{parking}}} - \sqrt{\frac{2\mu}{r_{parking}} - \frac{\mu}{a_{tr}}}$$

$$\Delta v_{parking} = \Delta v_1 + \Delta v_2$$

In [ ]:
```
# 1. Delta-v to launch from Europa surface to circular parking orbit (2 burns)
r_park = R_Europa + 200 # km
v_park = circularVelocity(r_park,mu_Europa)
a_tr = (r_park + R_Europa)/2
```

```
deltaV1_park = speed_visViva(R_Europa,a_tr,mu_Europa)
deltaV2_park = abs(speed_visViva(r_park,a_tr,mu_Europa) - v_park)
deltaV_park = deltaV1_park + deltaV2_park
println("Delta-v to launch from Europa surface to circular parking orbit: ",
            deltaV_park, " km/s")
```

```
Delta-v to launch from Europa surface to circular parking orbit: 1.5161975221417654 km/s
```

2nd Maneuver: Hyperbolic trajectory. We must use the the absolute velocity of the spacecraft at the beginning of Lambert transfer and the velocity of Europa to find the excess hyperbolic velocity. From here we can obtain the semi-major axis of the hyperbolic orbit. And using the relationship with the periapsis, obtain the eccentricity of the hyperbolic orbit.

$$v_h = v_{s/c} - v_{Europa} = \sqrt{-\frac{\mu}{a}}$$

$$r_p = a(1-e)$$

$$\Delta v_{hyper} = \sqrt{\frac{2\mu}{r_{parking}} - \frac{\mu}{a_{hyper}}} - \sqrt{\frac{\mu}{r_{parking}}}$$

Total Departure:

$$\Delta v_{departure} = \Delta v_{parking} + \Delta v_{hyper}$$

In [ ]:
```
# 2. Compute a and e of the departure hyperbola, and Delta-v for injection

# Considering preferred option for the trip:
r_opt_EU = r_Europa[:,positions[1][2]]
v_opt_EU = v_Europa[:,positions[1][2]]
r_opt_GA = r_Ganymede[:,positions[1][2],positions[1][1]]
v_opt_GA = v_Ganymede[:,positions[1][2],positions[1][1]]

vs1_Europa, vs1_Ganymede = Lambert(MyVector(r_opt_EU),MyVector(r_opt_GA),
                           t_travel_min*24*3600,1,mu_Jupiter,maxiter,tol)
vns1_Europa, vns1_Ganymede = Lambert(MyVector(r_opt_EU),MyVector(r_opt_GA),
                           t_travel_min*24*3600,-1,mu_Jupiter,maxiter,tol)
C3s1 = norm(vs1_Europa.c0 - v_Europa[:,end])^2 +
                    norm(vs1_Ganymede.c0 - v_Ganymede[:,end,end])^2
C3ns1 = norm(vns1_Europa.c0 - v_Europa[:,end])^2 +
                    norm(vns1_Ganymede.c0 - v_Ganymede[:,end,end])^2

if C3s1<C3ns1
    vopt_EU = vs1_Europa.c0
    vopt_GA = vs1_Ganymede.c0
else
    vopt_EU = vns1_Europa.c0
    vopt_GA = vns1_Ganymede.c0
end

# Compute a and e of the departure hyperbola
a_hyper = -mu_Europa/norm(vopt_EU - v_opt_EU)^2
println("a_hyper: ", a_hyper)
e_hyper = 1 - r_park/a_hyper
println("e_hyper: ", e_hyper)
v_hyper = speed_visViva(r_park,a_hyper,mu_Europa)
println("v_hyper: ", v_hyper)
deltaV_hyper = abs(v_hyper - v_park)
println("Delta-v for injection: ", deltaV_hyper, " km/s")

println("")
```

```
deltaV_departure = deltaV_park + deltaV_hyper
println("Total Delta-v for departure from Europa: ", deltaV_departure, " km/s")
```

```
a_hyper: -1306.8836643189525
e_hyper: 2.3473272702644072
v_hyper: 2.4674756806346116
Delta-v for injection: 1.1188115932740414 km/s

Total Delta-v for departure from Europa: 2.635009115415807 km/s
```

(f) For the arrival at Ganymede, determine the impact parameter for an arrival hyperbola with pericenter at 300 km from the moon's datum. Determine the ΔV needed to acquire a circular orbit at that altitude.

Final Maneuver: Arrival at Ganymede.

$$v_h = v_{s/c} - v_{Ganymede} = \sqrt{-\frac{\mu}{a}}$$

$$r_p = a(1-e)$$

Impact parameter: $B = -a\sqrt{e^2-1}\Delta v_{arrival} = \Delta v_{parking,arrival} + \Delta v_{hyper,arrival}$$

In [ ]:
```
# 1. Impact parameter for arrival hyperbola
rp_GA = R_Ganymede + 300 # km
a_hyper_arrival = -mu_Ganymede/norm(vopt_GA - v_opt_GA)^2
e_hyper_arrival = 1 - rp_GA/a_hyper_arrival
B_arrival = -a_hyper_arrival*sqrt(e_hyper_arrival^2 - 1)
println("Impact parameter for arrival hyperbola: ", B_arrival, " km")
println("Eccentricity of arrival hyperbola: ", e_hyper_arrival)
println("Semi-major axis of arrival hyperbola: ", a_hyper_arrival, " km")
println("")

# 2. Compute Delta-v to acquire circular orbit at that altitude
v_hyper_arrival = speed_visViva(rp_GA,a_hyper_arrival,mu_Ganymede)
v_park_arrival = circularVelocity(rp_GA,mu_Ganymede)
deltaV_arrival = abs(v_hyper_arrival - v_park_arrival)
println("Delta-v to acquire circular orbit at Ganymede: ", deltaV_arrival,
                                                          " km/s")
println("")
```

```
Impact parameter for arrival hyperbola: 6319.062957954673 km
Eccentricity of arrival hyperbola: 1.5483287110525419
Semi-major axis of arrival hyperbola: -5345.6985580299615 km

Delta-v to acquire circular orbit at Ganymede: 1.0952870206913197 km/s
```

Although we are not explicitly asked to compute, it is interesting to find the total Δv that the mission would require, as this would most important result in the preliminary evaluation of the mission. Depending of this value, we will need to choose one propulsion system or another.

In [ ]:
```
# Total Delta-v of the mission
deltaV_tot = deltaV_departure + deltaV_arrival
println("Total Delta-v of the mission: ", deltaV_tot, " km/s")
```

```
Total Delta-v of the mission: 3.7302961361071265 km/s
```

This result is relatively low, which is to be expected as we are we are performing the mission inside the the time window necessary to obtain the minimum Δv.