1.

a) $V_{circ} = \sqrt{\frac{\mu}{r_0}} \Rightarrow \boxed{\begin{bmatrix} 0 \\ 7.059 \\ 0 \end{bmatrix} \text{km/s}}$

b) $a = \begin{bmatrix} a_r \\ a_t \end{bmatrix} = \boxed{\begin{bmatrix} -\frac{\mu}{r^2}\hat{r} \\ T_{sp}\hat{v} \end{bmatrix}}$

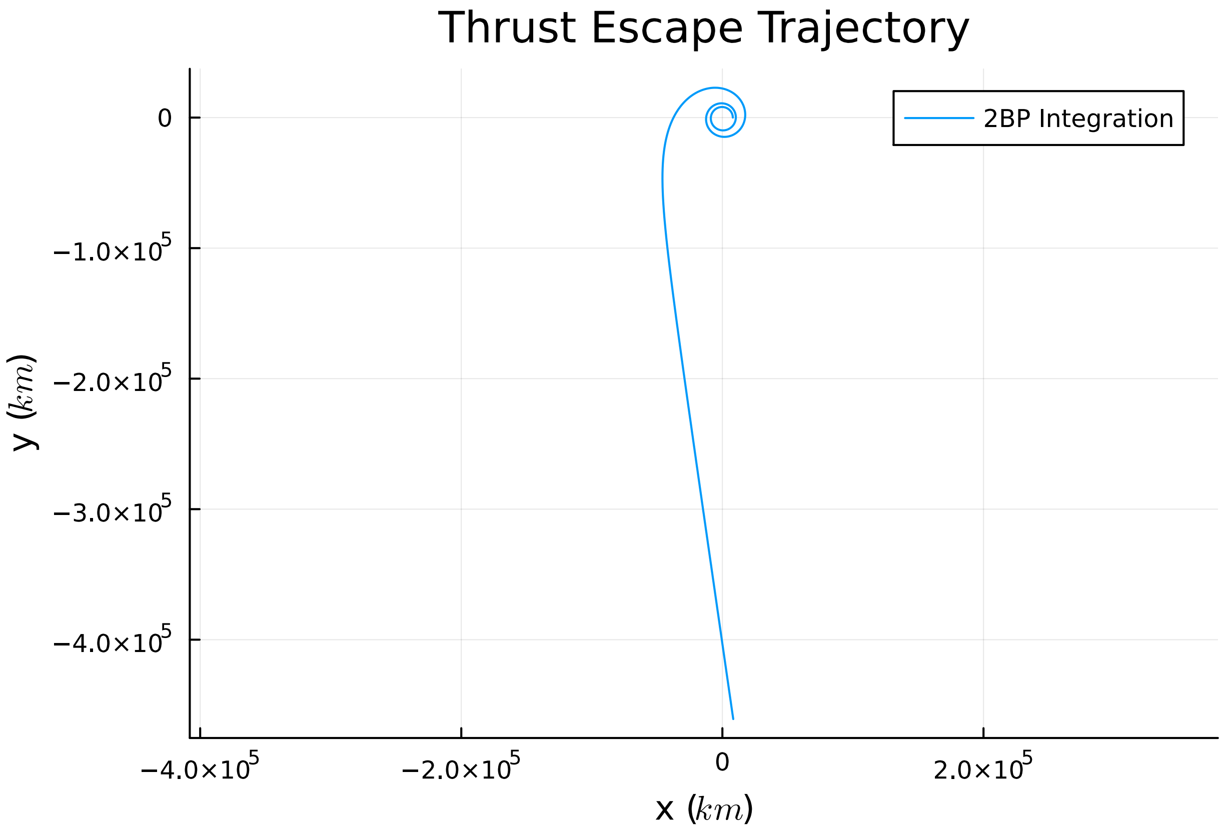
c) $t_{esc} = \frac{v_0}{a_t}\left[1 - \left(\frac{20 a_t^2 r_0^2}{v_0^4}\right)^{1/8}\right]$

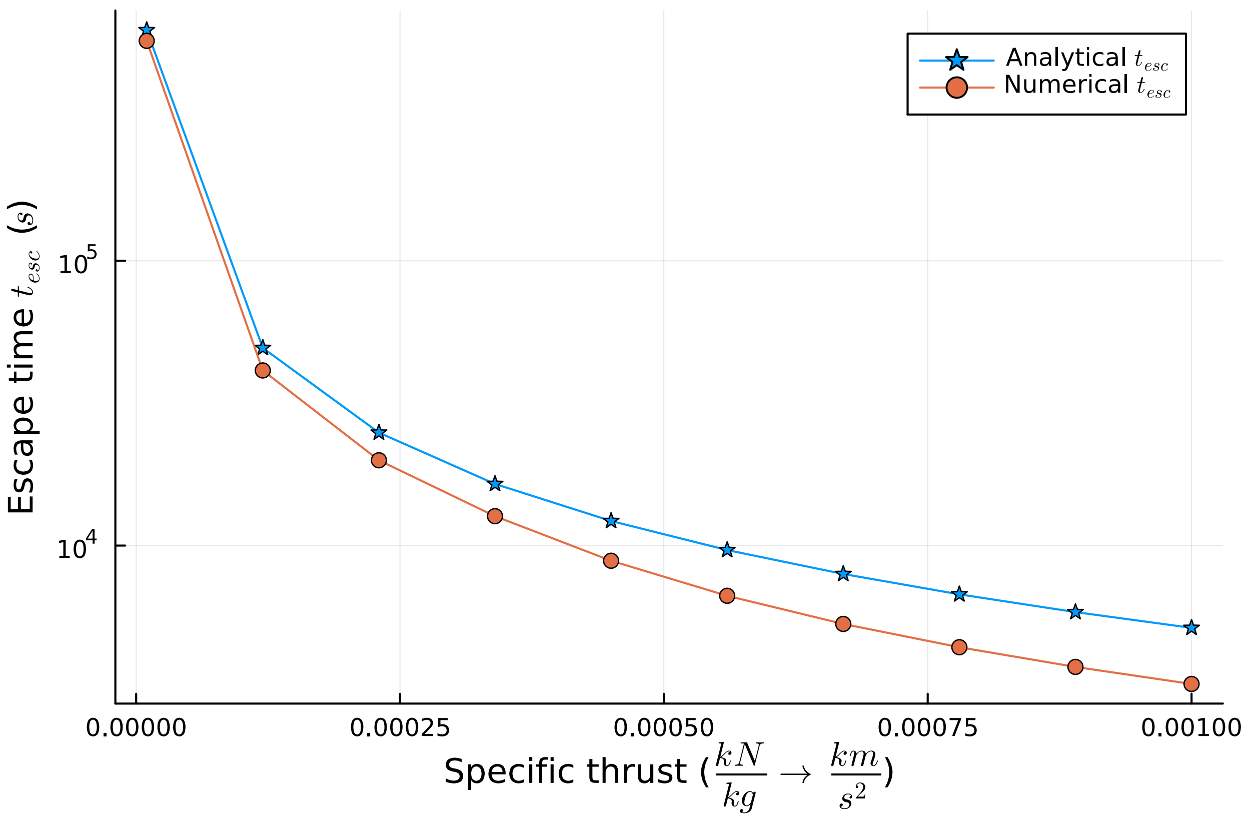
$= \boxed{59815 \text{ sec}}$

e) The analytic approximation neglects how gravity weakens as you spiral outward, so it overestimates t_{esc}

```julia
# enae404 hw07
include("../../../code/sfd.jl")
using .SpaceFlightDynamics
using LinearAlgebra
using Plots
using LaTeXStrings

# problem 01
# givens
T_s = 1e-4
r_o = 8000.0
r_1 = r_o * [1.0, 0.0, 0.0]

# part a
v_o = sqrt(μ_Earth / r_o)
v_1 = v_o * [0.0, 1.0, 0.0]
@show v_1

# part b
a_r = -1 * μ_Earth / r_o^2
a_t = T_s
a = [a_r, a_t]
@show a

# part c
t_e = v_o / a_t * (1 - (20 * a_t^2 * r_o^2 / v_o^9)^(1 / 8))
@show t_e

# part d
sv = solve_2BP_thrust(StateVectors(r_1, v_1), (0.0, 2 * t_e), μ=μ_Earth, T_spec=T_s,
int_pts=500)
v_e = sv[end].v
@show v_e

xs = [sv.r[1] for sv in sv]
ys = [sv.r[2] for sv in sv]
plt = plot(
    xs, ys, label="2BP Integration",
    title="Thrust Escape Trajectory",
    xlabel=L"x ($km$)",
    ylabel=L"y ($km$)",
    aspect_ratio=:equal,
    grid=true)
display(plt)

# part e
time_step = 2 * t_e / length(sv)
t_e_num = 0
for i ∈ eachindex(sv)
    ε = 0.5 * norm(sv[i].v)^2 - μ_Earth / norm(sv[i].r)
    if ε > 0
        global t_e_num = i * time_step
        break
    end
end
@show t_e_num

# part f
analytic_tesc(a_t) = v_o / a_t * (1 - (20 * a_t^2 * r_o^2 / v_o^9)^(1 / 8))
```

```
function numeric_tesc(a_t; int_pts=2000)
    t_e = analytic_tesc(a_t)
    t_end = 10 * t_e
    sv = solve_2BP_thrust(
        StateVectors(r_1, v_1),
        (0.0, t_end),
        μ=μ_Earth,
        T_spec=a_t,
        int_pts=int_pts
    )
    N = length(sv)
    ts = range(0, t_end, length=N)
    ε = [0.5 * norm(sv[i].v)^2 - μ_Earth / norm(sv[i].r) for i in 1:N]
    idx = findfirst(ε .>= 0)
    if idx === nothing
        return NaN
    elseif idx == 1
        return ts[1]
    else
        t1, t2 = ts[idx-1], ts[idx]
        e1, e2 = ε[idx-1], ε[idx]
        return t1 - e1 * (t2 - t1) / (e2 - e1)
    end
end

T_specs = range(1e-5, 1e-3, length=10)
t_anal = [analytic_tesc(T) for T in T_specs]
t_num = [numeric_tesc(T) for T in T_specs]

plot(
        T_specs, t_anal,
    label=L"Analytical $t_{esc}$",
    xlabel=L"Specific thrust ($\frac{kN}{kg}\to\frac{km}{s^2}$)",
    ylabel=L"Escape time $t_{esc}$ ($s$)",
    yscale=:log10,
    marker=:star5,
    legend=:topright,
    grid=true
)
plot!(
    T_specs, t_num,
    label=L"Numerical $t_{esc}$",
    marker=:circle
)

v_1 = [0.0, 7.0586865084801715, 0.0]
a = [-0.006228131903125, 0.0001]
t_e = 59814.507545356515
v_e = [1.3415898646080981, -9.296827531119956, 0.0]
t_e_num = 50483.4443682809
```


Thrust Escape Trajectory
2BP Integration
0
-1.0×10^5
-2.0×10^5
-3.0×10^5
-4.0×10^5
y (km)
-4.0×10^5
-2.0×10^5
0
2.0×10^5
x (km)


Analytical t_{esc}
Numerical t_{esc}
10^5
10^4
Escape time t_{esc} (s)
0.00000
0.00025
0.00050
0.00075
0.00100
Specific thrust ($\frac{kN}{kg} \rightarrow \frac{km}{s^2}$)

```julia
function two_body!(du, u, μ, t)
        # u = [ x, y, z, vx, vy, vz ]
        # du[1:3] = v
        # du[4:6] = acceleration
        @views du[1:3] .= u[4:6]
        r = @view u[1:3]
        r_norm = norm(r)
        @views du[4:6] .= -μ .* r ./ (r_norm^3)
end

function two_body_thrust!(du, u, params, t)
        μ, T_spec_kN_per_kg = params

        # unpack position & velocity views
        @views du[1:3] .= u[4:6]
        r = @view u[1:3]
        v = @view u[4:6]

        # gravity
        r_norm = norm(r)
        grav_acc = -μ .* r ./ (r_norm^3)

        a_thrust_mag = T_spec_kN_per_kg
        v_norm = norm(v)
        thrust_acc = v_norm > 0 ? a_thrust_mag .* (v ./ v_norm) : zero(v)

        @views du[4:6] .= grav_acc .+ thrust_acc
end

function solve_2BP(initial::StateVectors,
                   tspan::Tuple{Float64, Float64};
                   μ::Float64 = μ_Earth,
                   reltol::Float64 = 1e-9,
                   abstol::Float64 = 1e-9,
                   int_pts::Int64 = 2)

        # pack initial state
        u0 = vcat(initial.r, initial.v)

        # setup and solve ODE problem
        prob = ODEProblem(two_body!, u0, tspan, μ)
        sol  = solve(prob, Tsit5(), reltol=reltol, abstol=abstol,
saveat=range(start=tspan[1], stop=tspan[2], length=int_pts))

        # unpack back into StateVectors
        return [StateVectors(u[1:3], u[4:6]) for u in sol.u]
end

function solve_2BP_thrust(initial::StateVectors,
                          tspan::Tuple{Float64, Float64};
                          μ::Float64 = μ_Earth,
                          T_spec::Float64 = 1e-4,
                          reltol::Float64 = 1e-9,
                          abstol::Float64 = 1e-9,
                          int_pts::Int64 = 2)

        # pack initial state
        u0 = vcat(initial.r, initial.v)
```

```
        # setup and solve ODE problem
        prob = ODEProblem(two_body_thrust!, u0, tspan, (μ, T_spec))
        sol  = solve(prob, Tsit5();
              reltol = reltol,
              abstol = abstol,
              saveat = range(tspan[1], tspan[2], length=int_pts))

        # unpack back into StateVectors
        return [ StateVectors(u[1:3], u[4:6]) for u in sol.u ]
end

export solve_2BP, solve_2BP_thrust
```

```
Error: UndefVarError: `StateVectors` not defined in `Main.var"##WeaveSandBo
x#233"`
Suggestion: check for spelling errors or missing imports.
```

2.

$R_M = 3389.5$ km

$r_p = 4389.5$ km

$e = 0.25$

$a = 5825.7$ km

$v_p = \sqrt{\mu \frac{1+e}{r_p}}$

$v_{circ} = \sqrt{\frac{\mu}{r_p}}$

$\mu_M = 4.2828E4\ km^4/s^2$

$v_p = 3.49$ km/s

$v_{circ} = 3.12$ km/s

$\Delta V = v_e \log\left(\frac{m_0}{m_1}\right)$

$V_e = I_{sp} g_0 = 2452$ m/s

$m_0 = 1500$ kg $\Rightarrow \boxed{m_{prop} = 210\ kg}$

$\epsilon = \frac{m_{struct}}{m_{prop}} = 0.15 \rightarrow m_{struct} = 31.5$ kg

$m_0 - (m_{struct} + m_{prop}) = m_{payload} = 1258.5$ kg

$\frac{m_{payload}}{m_0} = 0.84 \Rightarrow \boxed{84\%}$

enae404 hw07

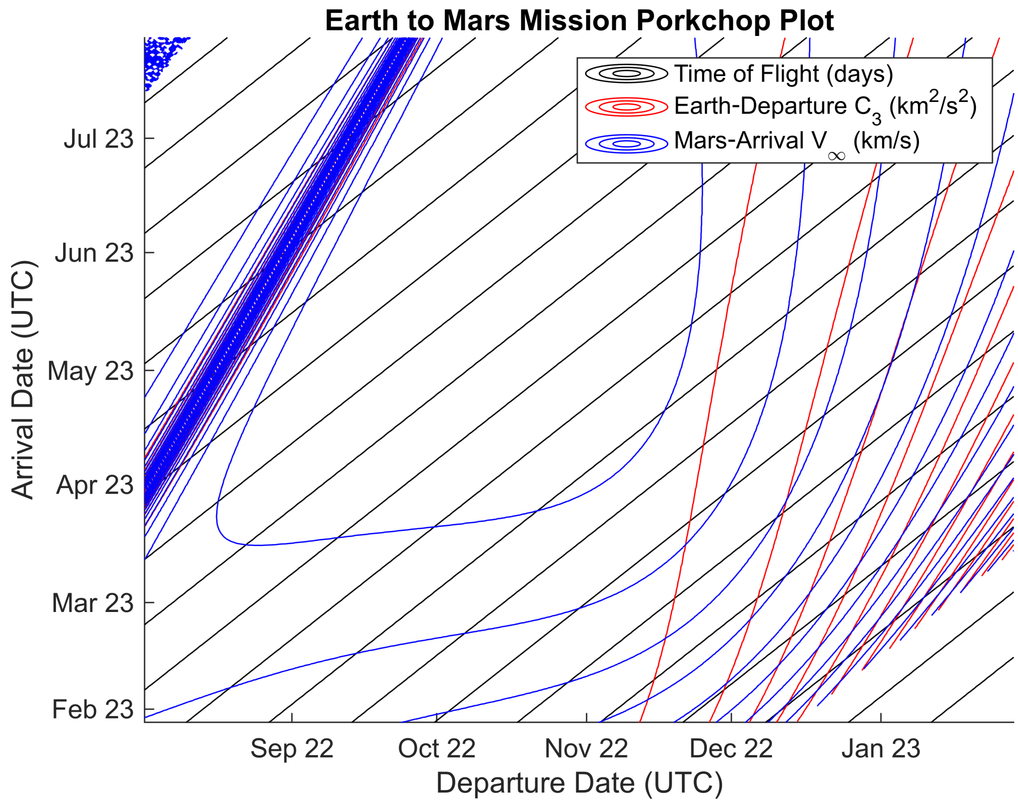
vai srivastava

```
addpath("./.") % path to provided code
addpath("../../../code") % path to custom ENAE404 lib
```

problem 3

```
dep0 = [2022,8,1,12,0,0];
arr0 = [2023,1,28,12,0,0];
nDep = 400;
nArr = 400;
minTOF = 45;
maxTOF = 500;

depStartDN = datenum(dep0);
arrStartDN = datenum(arr0);
depEndDN = arrStartDN;
depDates = linspace(depStartDN, depEndDN, nDep);
arrEndDN = arrStartDN + (depEndDN - depStartDN);
arrDates = linspace(arrStartDN, arrEndDN, nArr);

[DEP, ARR] = meshgrid(depDates, arrDates);
TOF = (ARR - DEP);

valid = (TOF >= minTOF) & (TOF <= maxTOF);

C3 = nan(size(TOF));
VinfMars = nan(size(TOF));

for i = 1:numel(DEP)
    if ~valid(i)
        continue
    end

    dv_dep = datevec(DEP(i));
    dv_arr = datevec(ARR(i));
    jd_dep = Provided.ymdhms2jd(dv_dep(1), dv_dep(2), dv_dep(3), dv_dep(4),
dv_dep(5), dv_dep(6));
    jd_arr = Provided.ymdhms2jd(dv_arr(1), dv_arr(2), dv_arr(3), dv_arr(4),
dv_arr(5), dv_arr(6));
    tof_sec = (jd_arr - jd_dep)*86400;

    [RE, VE] = Provided.findEarth(jd_dep);
    [RM, VM] = Provided.findMars(jd_arr);

    [V1.short, V2.short, ~, ~] = SFD.solve_lambert(RE, RM, tof_sec, SFD.mu_Sun,
false);
    [V1.long, V2.long, ~, ~] = SFD.solve_lambert(RE, RM, tof_sec, SFD.mu_Sun, true);
```

```
    Vinf_dep.short = V1.short - VE;
    Vinf_dep.long = V1.long - VE;
    Vinf_arr.short = V2.short - VM;
    Vinf_arr.long = V2.long - VM;

    short = norm(Vinf_dep.short)+norm(Vinf_arr.short);
    long = norm(Vinf_dep.long)+norm(Vinf_arr.long);

    if short < long
        C3(i) = norm(Vinf_dep.short)^2;
        VinfMars(i) = norm(Vinf_arr.short);
    else
        C3(i) = norm(Vinf_dep.long)^2;
        VinfMars(i) = norm(Vinf_arr.long);
    end
end

depDatesNum = depDates;
arrDatesNum = arrDates;

figure; hold on;
contour(depDatesNum, arrDatesNum, TOF, 20, 'k');
contour(depDatesNum, arrDatesNum, C3, 20, 'r');
contour(depDatesNum, arrDatesNum, VinfMars, 20, 'b');
legend('Time of Flight (days)', 'Earth-Departure C_3 (km^2/s^2)', 'Mars-Arrival
V_{\infty} (km/s)');
title("Earth to Mars Mission Porkchop Plot");
xlabel('Departure Date (UTC)');
ylabel('Arrival Date (UTC)');
datetick('x','mmm yy','keeplimits');
datetick('y','mmm yy','keeplimits');
```


Earth to Mars Mission Porkchop Plot
Time of Flight (days)
Earth-Departure C_3 (km^2/s^2)
Mars-Arrival V_∞ (km/s)
Jul 23
Jun 23
May 23
Apr 23
Mar 23
Feb 23
Arrival Date (UTC)
Sep 22
Oct 22
Nov 22
Dec 22
Jan 23
Departure Date (UTC)

```
function [v1, v2, e, rp] = solve_lambert(r1, r2, TOF, mu, long_way)
%SOLVE_LAMBERT Solve Lambert's problem
% Solver for Lambert's problem using non-rigorous stumpff method
    arguments
        r1 double
        r2 double
        TOF double
        mu = SFD.mu_Earth
        long_way = false
    end
    r1_norm = norm(r1);
    r2_norm = norm(r2);
    cos_dth = dot(r1, r2)/(r1_norm*r2_norm);
    dth = acos(min(max(cos_dth, -1),1));
    if long_way
        if dth < pi
            dth = 2*pi - dth;
        end
    else
        if dth > pi
            dth = 2*pi - dth;
        end
    end
    A = sin(dth)*sqrt(r1_norm*r2_norm/(1 - cos(dth)));
    if A == 0
        error('Cannot compute Lambert solution: A = 0');
    end
    F = @(z) (( (r1_norm + r2_norm + A*(z.*SFD.stumpff_C3(z) -
1)./sqrt(SFD.stumpff_C2(z)))./SFD.stumpff_C2(z) ).^(3/2).*SFD.stumpff_C3(z)
+ A*sqrt(r1_norm + r2_norm + A*(z.*SFD.stumpff_C3(z) - 1)./
sqrt(SFD.stumpff_C2(z))) ) / sqrt(mu) - TOF;
    z = 0;
    for iter = 1:200
        Fz = F(z);
        if abs(Fz) < 1e-8
            break;
        end
        delta = 1e-6;
        dF = (F(z + delta) - F(z - delta))/(2*delta);
        z = z - Fz/dF;
    end
    C3 = SFD.stumpff_C3(z);
    C2 = SFD.stumpff_C2(z);
    y = r1_norm + r2_norm + A*(z*C3 - 1)/sqrt(C2);
    f = 1 - y/r1_norm;
    g = A*sqrt(y/mu);
    gdot = 1 - y/r2_norm;
    v1 = (r2 - f*r1)/g;
    v2 = (gdot*r2 - r1)/g;
    h_vec = cross(r1, v1);
    e_vec = (1/mu)*((norm(v1)^2 - mu/r1_norm)*r1 - dot(r1,v1)*v1);
    e = norm(e_vec);
```

```
    energy = norm(v1)^2/2 - mu/r1_norm;
    a = -mu/(2*energy);
    rp = a*(1 - e);
end
```

Published with MATLAB® R2024b