Bilaga i: Huvudprogram

Denna bilaga innehåller huvudkoden som använts i uppgiften.

```
% Projektarbete "Rymdskeppet Futten illa ute"
% Andreas Fröderberg & Henrik Hvitfeldt
close all; clear all; clc;
%% Declare variables
check = 0;    % Enables accuracy control of quadinterpol
global alpha

pm = char(177);    % Plus minus sign
h = 0.01;  % Step length used in RK4
bisec_err = 0.0001;

%% Part 1: Test H-values and calculate passing r, t and phi for alpha = 90
% Runge Kuttas method while not crash and while not pass
disp('Part 1')
alpha = 90;
t_list = [];                          % Empty vector of passing t:s
r_list = [];                          % Empty vector of passing r:s
phi_list = [];                        % Empty vector of passing phi:s
H_list = [];                          % Empty vector of passing phi:s
crash_list = [];                      % Empty vector of passing phi:s
starting_heights = [10, 8, 6, 2];

% Test and plot trajectories for different H
figure ()
plotStyle = {'b','g','r','m'};                              % Set colors of trajectory plots
for i = 1:length(starting_heights)                          % Test H:s
    H = starting_heights(i);
     trajectory = RKeval(h, H);                             % Evaluate trajecory with RK4
     trajectory.H=H;
     trajectories(i) = trajectory;
    % Plot trajectory
     polar(trajectory.phi, trajectory.r, plotStyle{i})                % Plot tracectory
```

```matlab
    view([90 -90])                          % Flip plot to 0 deg up
    grid on; hold all;
    legendInfo{i} = ['Starting height ' num2str(H) ' earth radii'];       % Add legend info for use in legend command
end
title ('Trajectories for different starting altitudes, \alpha =90')
legend(legendInfo)      % Set legends accoring to legend info

%find passing values for all trajectories
for i = 1:length(starting_heights)
    trajectories_temp(i)=futten_pass(trajectories(i));
end
trajectories=trajectories_temp;


% Display results part 1
make_table(trajectories, 'pass.xls')

%% Part 2 Find H*, H when Futten just passes earth
% Bisection method to determine critical starting height, H_star, when
% trajectory just passes earth.
start1=3;        %first start value for bisection method
start2=6;        %second startvalue for bisection method
start=[start1 start2];

H_star = bisection_meth(@futt_extra, start, h);
% Evaluate trajectory for H_star
trajectory_star = RKeval(h, H_star);
trajectory_star = futten_pass(trajectory_star);
fprintf('Futten passerar precis jordytan vid %0.3f%c%f jordradier\n', H_star, pm, bisec_err)
fprintf('vilket ger passeringsradie %0.3f%c%f jordradier\n', trajectory_star.r_pass, pm, trajectory_star.r_err)
fprintf('Passeringshastigheten är då %0.3f%c%f jordradier/h\n', trajectory_star.v_pass, pm, trajectory_star.v_err)
%%
% Calculate trajectory length
[X.star,Y.star]=euklid(trajectory_star.r,trajectory_star.phi); %turns polar to kartesis kordi.
```

```
traj_length = arclength(X.star, 0, Y.star); %distance traveled
traj_length_2h = arclength((X.star(1:2:end)), 0, Y.star(1:2:end));
fprintf('och Futten har åkt avståndet %0.3f%c%f jordradier\n', traj_length, pm, abs(traj_length - traj_length_2h))

%% Plot trajectory at H_star
figure()
polar(trajectory_star.phi, trajectory_star.r,'r')
title('Trajectory at critical H, \alpha =90')
view([90 -90])
hold on;
phi_earth = 0:360/(length(trajectory_star.r)):360;
r_earth = ones(1,length(trajectory_star.r)+1);
polar(phi_earth,r_earth,'b')
legend('Trajectory','Earth')
pass_speed = trajectory_star.v_pass;
%% least square and length
figure()

plot (X.star, Y.star)
hold on
grad=2;
[C]=least_square(X.star,Y.star,grad); %C=poly coeffs, dC=derivate
legendtext = sprintf('Interpolerad med polynom %0.2f%c^2%0.2f%c+%0.2f=r', C(1), 966, C(2), 966, C(3));
title('Kurva från RK4 mot interpolad till grad 2')
leg = {'RK4' legendtext};
% Plot interpolated
plot (X.star,polyval(C,X.star))
legend(leg)
title('Interporad kurva för Futten mot kurva frpn RK4')
xlabel('x [jordradier]')
ylabel('y [jordradier]')
%(analytiskt beräknad banlängd till 4.00492 från minsta kvadrat)
fprintf('Banlängden vid interpolering är %f jordradier\r\n', 4.00492)

%% Part 3 Find H* for different alphas

H_star_alpha = [];
alpha_list = [];
```

```
pass_speed_list = [];
figure()
% Loop over alphas and plot trajectory
i = 1;
alpha_legend = {};
alpha_pass = [];
alphas = [150:-10:50];
for alpha = alphas
    H_alpha = bisection_meth(@futt_extra, start, h);                %optimal height for different alpha
    H_star_alpha = [H_star_alpha; H_alpha];                                %list of optimal heights
    alpha_list = [alpha_list alpha];                                                %alpha list
    trajectory_star_alpha=RKeval(h,H_alpha);                                %calculates the passing curve
    trajectory_pass_alpha=futten_pass(trajectory_star_alpha);            %calculates the passing paramaters for above curve
    pass_speed_list = [pass_speed_list trajectory_pass_alpha.v_pass];   %corresponding speed of angle 70:10:130
    polar(trajectory_pass_alpha.phi, trajectory_pass_alpha.r)
    view([90 -90])                              % Flip plot to 0 deg up
    hold on
    legend_text = sprintf('%c=%d', 945, alpha);
    v_err(i) = trajectory_pass_alpha.v_err;
    H_err(i) = trajectory_pass_alpha.r_err;
    alpha_legend{i} = legend_text;
    pass_speed(i) = trajectory_pass_alpha.v_pass;
    i = i + 1;
end

% Flip to fit table
alphas = alphas';
pass_speed = pass_speed';
v_err = v_err';
H_err = H_err';

% Calculate errors

```

```
title('Trajectories at different starting angles')
legend(alpha_legend)

alpha_table = table(alphas, H_star_alpha, H_err, pass_speed, v_err);
disp('Passing parameters for different alphas')
disp(alpha_table)
writetable(alpha_table, 'alphas.xls')

%% Convergence check for RK4
disp('Check convergence for RK4')
alpha = 90;
error_vektor=[];
no_of_tests = 3;

% Loop for different step length
for a=0:no_of_tests-1
    h_test = h*2^a;
    % Run convergence tests
    trajectory_error=RKevalerror(h_test, H_star,2);
    trajectories_err(a+1) = trajectory_error;
    % Save all values
    end_value=[ trajectory_error.r(end),...
                trajectory_error.rdot(end),...
                trajectory_error.phi(end),...
                trajectory_error.phidot(end)];
   error_vektor=[error_vektor; end_value];
end

r = getTableData(error_vektor(:, 1));
rdot = getTableData(error_vektor(:, 2));
phi = getTableData(error_vektor(:, 3));
phidot = getTableData(error_vektor(:, 4));

tab = table(r, rdot, phi, phidot);
writetable(tab, 'RKerrors.xls');
rows = {'end(h)' 'end(2h)' 'end(4h)' 'diff(h; 2h)e6' 'diff(2h; 4h)e6' 'kvot' 'rel error'};
tab.Properties.RowNames = (rows);
disp(tab)

% Error of Hermite
Hermitefel(H_star);
```

Bilaga ii: Runge-Kutta 4

Denna bilaga innehåller koden till Runge-Kutta 4 som använts, samt dess hjälpfunktioner.

RKeval itererar RK4 tills jorden passerats, alltså när $\dot{r} = 0$.

```
function [trajectory] = RKeval (h, H)
% RKEVAL (h, H) Evaluates system until earth is passed or crashed into for
% starting height H and step lengt h.
% Used RK4step for evaluation.
% Returns struct with trajectory parameters

u = [H 0 0 0];                              % Initial conditions
t = 0;                                              % Starting t
trajectory = struct(    't',         t,...
                        'r',         u(1),...
                        'rdot',      u(2),...
                        'phi',       u(3),...
                        'phidot',    u(4));


% Evaluates while rdot is negative
while trajectory.rdot <= 0
    [t, u] = RK4step(t, u, h);        % RK step evaluation
    trajectory.t = [trajectory.t; t];
    trajectory.r = [trajectory.r; u(1)];
    trajectory.rdot = [trajectory.rdot; u(2)];
    trajectory.phi = [trajectory.phi; u(3)];
    trajectory.phidot = [trajectory.phidot; u(4)];
end
```

RKstep utför ett steg av RK4.

```
function [tout, uout] = RK4step (t, u, h)
% RK4STEP (t, u, h) calculates a step with RK4 method with step length h.

% RK4 factors
k1 = h*F(t, u);
k2 = h*F(t+0.5*h, u+0.5*k1);
k3 = h*F(t+0.5*h, u+0.5*k2);
k4 = h*F(t+h, u+k3);

```

```
uout = u + (k1+2*k2+2*k3+k4)/6;
tout = t + h;
```

F innehåller differentialekvationerna.

```
function [u_ut] = F(t, u)
% F (t, u) returns derivatives of physical system.
% Output is [r rdot phi phidot]

global alpha % Is global so no input is needed (for use in ode45)

% Define parameters
g = 20;                    % Gravity constant[earth radii/hour]
R = 1;                     % Radius of earth
G = Grav(u(1), g, R);      % Engine force

x1 = u(2);
x2 = u(1)*u(4)^2+G*cosd(alpha)-g*(R/u(1))^2;
x3 = u(4);
x4 = (G*sind(alpha)-2*u(2)*u(4))/u(1);

u_ut = [x1 x2 x3 x4];
```

Bilaga iii: Passering

Denna bilaga innehåller funktionen som använts för att beräkna Futtens passeringsvärden, samt Hermite- och linjär interpolering.

---

futten_pass beräknar värdena för futtens passering.

```
function [trajectory] = futten_pass(trajectory)
% FUTTEN_PASS (trajectory) calculates the different passing parameters
%r,phi,t,v using linear interpolation and hermit for when rdot equals zero
%i.e spaceship is heading to space again.

% Predefined errors
RK_r_err = 1.002e-6;
RK_phi_err = 9.1e-07;
RK_phidot_err = 8.3e-07;
herm_relerr = 0.302e-6;

rdot_pass=0;

% Calculate passing values
[t_pass, t_err] = linpol(         trajectory.rdot(end-2:end),...
                                  trajectory.t(end-2:end),...
                                  rdot_pass);
r_pass=herm(     t_pass,...
                 trajectory.t(end-1:end),...
                 trajectory.r(end-1:end),...
                 trajectory.rdot(end-1:end));
phi_pass=herm(   t_pass,...
                 trajectory.t(end-1:end),...
                 trajectory.phi(end-1:end),...
                 trajectory.phidot(end-1:end));
[phidot_pass, phidot_err] =linpol(   trajectory.rdot(end-2:end),...
                                     trajectory.phidot(end-2:end),...
                                     rdot_pass);

v_pass=phidot_pass*r_pass;
v_pass_err = phidot_err + phidot_pass*RK_phidot_err;

%add to struct
```

```
trajectory.t_pass = t_pass;
trajectory.t_err = t_err;
trajectory.r_pass = r_pass;
trajectory.r_err = r_pass*(herm_relerr + RK_r_err);
trajectory.phi_pass = phi_pass;
trajectory.phi_err = phi_pass*(herm_relerr + RK_phi_err);
trajectory.v_pass = v_pass;
trajectory.v_err = v_pass_err;
```

---

linpol är en funktion för linjärinterpolering.

```
function [y_out, err]=linpol (x,y,xq)
% LINPOL (x,y,xq) returns y(xq) with a linear
    interpolation of the points
% x=[x1 x2] and y=[y1 y2]. Also returns error of
    estimation.

% Calculate interpolated point
y_out = y(2)+(y(3)-y(2))/(x(3)-x(2))*(xq-x(2));

% Calculate interpolation at double step size
y_out_2h = y(1)+(y(2)-y(1))/(x(2)-x(1))*(xq-x(1));

% Calculate error
err = abs(y_out - y_out_2h);
```

---

herm är en funtion för Hermite-interpolering.

```
function yut = herm(xq,x,y,k)
% HERM (xq,x,y,k)
x1=x(1);
x2=x(2);
y1=y(1);
y2=y(2);
k1=k(1);
k2=k(2);

h = x2-x1;

c1 = y1 ;
c2 = (y2-y1)/h;
c3 = (k2-c2)/h^2;
c4 = (k1-c2)/h^2;

yut = c1+c2*(xq-x1)+c3*(xq-x1)^2*(xq-x2)+c4*(xq-x1)*(xq-
    x2)^2;
```

```
end
```

bisection_meth innehåller sekantmetoden.

```
function [root] = bisection_meth (fhandle, start, h)
% BISECTION_METH (fhandle,f, start, it_allowed, delta, h, c1)
% fhandle is the function evaluated with bisection method to find roots, f is the
% variable searched for if the function outputs a struct, else it should be set to 0=zero.
%start is the start values, It_allowed is the number of iterations allowed. delta is
% the allowed h. h is not right and neither is c1.
% Finds root for function 'fhandle' using bisection method. Precision is
% size of h (dx) allowed.

delta = 0.0001;
it_allowed = 100;
x0 = start(1);
x1 = start(2);
it = 1;        % Starting values, it = iteration counter
[out] = feval(fhandle, h, x0); % Evaluate function at starting value 1
f0 = out;
deltax = 1;

% Bisection method
while abs(deltax) > delta
    % Convergence ceiling
    if it > it_allowed
        disp('Bisection iteration timeout')
        break
    end
    [out] = feval(fhandle, h, x1);             % Evaluate function at x1
    f1=out;
    deltax = (x1 - x0)/(f1 - f0)*f1;          % Calculate dx
    x0 = x1;
    f0 = f1;                        % New best guess
    x1 = x1 - deltax;                          % New best x = old best x - h
```

```
    it = it + 1;                                    % Iteration
        counter step
end

root = x1;
```

Bilaga iv: Övriga funktioner

Denna bilaga innehåller funktioner som inte lika lätt kan presenteras i sammanhang av uppgiftslösningen.

---

least_square är en funtion för minstakvadratanpassning.

```
function[c]=least_square(X,Y,grad)
% LEAST_SQUARE (X,Y,grad)X is the independent vector, Y is the vector of the result, grad is the grad of
% the desired polynomial.
for a=1:grad+1                    %constructing the matrix A of F(X)=Y
    A(:,a)=X.^(grad+1-a);
end
  c=A\Y;
```

---

cartesian gör om polära koordinater till kartesiska.

```
function [x, y] = cartesian(R, phi)
% CARTESIAN (R, phi) transforms polar coordinates to cartesian space

y=R.*sin(phi);
x=R.*cos(phi);
```

---

arclength beräknar banlängden för en kurva med givna punkter i kartesiska koordinater.

```
function [ L ] = arclength( x, C,y)
%  this funciton determin the length of a vector function. further is uses the
%  formula sum sqprt (1+f'^2) h=>0.
dL=[];

for a=1:length(x)-1;
     deltax=x(a+1)-x(a);
     % If a polynomial, get intermediate points
     if y==0
     deltay=polyval(C,x(a+1))-polyval(C,x(a));
     % If a set of descrete points, use linear sumation
     elseif C==0
          deltay=y(a+1)-y(a);
     end
dS=sqrt(deltax^2+deltay^2);
dL=[dL dS];
```

```
end
L=sum(dL);
```

Hermitefel räknar ut felskattningen för Hermiteinterpolering.

```
function [] = Hermitefel(H)
% HERMITFEL

% Define parameters
h = 0.01;
u = [H 0 0 0];
%bana_big = RKeval(h*2, H);
bana_small = RKeval(h, H);
bana_big = struct(  't',     bana_small.t(1:2:end),...
                    'r',     bana_small.r(1:2:end),...
                    'rdot',  bana_small.rdot(1:2:end),...
                    'phi',   bana_small.phi(1:2:end),...
                    'phidot',    bana_small.phidot(1:2:end
                        ));

x1 = [];
y1 = [];
x2 = [];
y2 = [];
phi1=[];
phi2=[];

k = 2;
% Get interpolation for 2*h
for n = 1:length(bana_big.t) - 1
    [x, y, phi] = herm_step(bana_big, n, k);
    x1 = [x1 x];
    y1 = [y1 y];
    phi1=[phi1 phi];
    hold on
end

k = 1;
% Get interpolation for h
for n = 1:length(bana_small.t) - 1
    [x, y, phi] = herm_step(bana_small, n, k);
    x2 = [x2 x];
    y2 = [y2 y];
    phi2=[phi2 phi];
    hold on
end
```

```

err_abs = max(abs(y1 - y2));
ind = find(max(abs(y1 - y2)) == err_abs);
err_rel = err_abs/y1(ind);


fprintf('Hermitefel %fe-6\n', err_rel*1e6)
```

---

herm_step Hermiteinterpolerar mellan två punkter och returnerar interpolerade värden mellan kurvorna (används for att plotta hel interpolerad kurva).

```
function [xpos, v, phi] = herm_step(bana, n, k)

points = 100*k;
xpos = [];
v = [];
phi=[];
x1 = bana.t(n);
x2 = bana.t(n + 1);


for i = 1:points
    x = x1 + (x2-x1)*i/points;
    p = herm(    x,...
                 [x1, x2],...
                 [bana.r(n), bana.r(n + 1)],...
                 [bana.rdot(n), bana.rdot(n + 1)]);
    p2= herm(    x,...
                 [x1, x2],...
                 [bana.phi(n), bana.phi(n + 1)],...
                 [bana.phidot(n), bana.phidot(n + 1)]);

    v = [v p];
    phi=[phi,p2];
    xpos = [xpos x];
end
```