MATH 211.3 Winter Term 2024 Assignment

Assignment #09

Name: Kanwar Pannu

Student Number: 11336732

NSID: eza147

**Problem 1**

clear;

clc;

inter = [0, 100];

ic1 = [2, 0.2, 2, -0.2];

ic2 = [0, -0.01, 0, 0.01];

n = 10000;

p = 5;

orbit\_two\_body(inter, ic1, ic2, n, p);

function orbit\_two\_body(inter, ic1, ic2, n, p)

h = (inter(2) - inter(1)) / n;

y = zeros(n+1, 8);

t = linspace(inter(1), inter(2), n+1);

y(1, :) = [ic1 ic2];

for i = 1:n

y(i+1, :) = eulerstep\_two\_body(t(i), y(i, :), h);

end

figure;

plot(y(:, 1), y(:, 3), 'r-', y(:, 5), y(:, 7), 'b-'); % Body 1 in red, Body 2 in blue

xlabel('X position');

ylabel('Y position');

title('Two-Body Problem Trajectories');

axis equal;

set(gca, 'XLim', [-5 5], 'YLim', [-5 5]);

end

function y=eulerstep\_two\_body(t, x, h)

y = x + h \* ydot\_two\_body(t, x);

end

function z=ydot\_two\_body(t, x)

m1 = 0.3;

m2 = 0.03;

G = 1;

x1 = x(1); vx1 = x(2); y1 = x(3); vy1 = x(4);

x2 = x(5); vx2 = x(6); y2 = x(7); vy2 = x(8);

r = sqrt((x2 - x1)^2 + (y2 - y1)^2);

F = G \* m1 \* m2 / r^2;

ax1 = F \* (x2 - x1) / (m1 \* r);

ay1 = F \* (y2 - y1) / (m1 \* r);

ax2 = -F \* (x2 - x1) / (m2 \* r);

ay2 = -F \* (y2 - y1) / (m2 \* r);

z = [vx1, ax1, vy1, ay1, vx2, ax2, vy2, ay2];

end

**A graph of a problem

Description automatically generated**

**10**

clear;

clc;

inter = [0, 100];

n = 10000;

ic1\_a = [2, 0.2, 2, -0.2];

ic2\_a = [0, 0, 0, 0];

ic3\_a = [-2, -0.2, -2, 0.2];

orbit\_three\_body(inter, ic1\_a, ic2\_a, ic3\_a, n, 5);

ic1\_b = [2, 0.20001, 2, -0.2];

orbit\_three\_body(inter, ic1\_b, ic2\_a, ic3\_a, n, 5);

function orbit\_three\_body(inter, ic1, ic2, ic3, n, p)

h = (inter(2) - inter(1)) / n;

y = zeros(n+1, 12);

t = linspace(inter(1), inter(2), n+1);

y(1, :) = [ic1, ic2, ic3];

for i = 1:n

y(i+1, :) = eulerstep\_three\_body(t(i), y(i, :), h);

end

% Plotting

figure;

plot(y(:, 1), y(:, 3), 'r-', y(:, 5), y(:, 7), 'b-', y(:, 9), y(:, 11), 'g-');

xlabel('X position');

ylabel('Y position');

title('Three-Body Problem Trajectories');

legend('Body 1', 'Body 2', 'Body 3');

axis equal;

end

function y = eulerstep\_three\_body(t, x, h)

% Euler step for three-body dynamics

y = x + h \* ydot\_three\_body(t, x);

end

function z = ydot\_three\_body(t, x)

% Derivative function for the three-body problem

% Masses

m1 = 0.3; m2 = 0.03; m3 = 0.03;

G = 1; % Gravitational constant

% Extract positions and velocities

x1 = x(1); y1 = x(3);

vx1 = x(2); vy1 = x(4);

x2 = x(5); y2 = x(7);

vx2 = x(6); vy2 = x(8);

x3 = x(9); y3 = x(11);

vx3 = x(10); vy3 = x(12);

% Distances

r12 = sqrt((x2 - x1)^2 + (y2 - y1)^2);

r13 = sqrt((x3 - x1)^2 + (y3 - y1)^2);

r23 = sqrt((x3 - x2)^2 + (y3 - y2)^2);

% Accelerations due to gravitational attraction

ax1 = G \* m2 \* (x2 - x1) / r12^3 + G \* m3 \* (x3 - x1) / r13^3;

ay1 = G \* m2 \* (y2 - y1) / r12^3 + G \* m3 \* (y3 - y1) / r13^3;

ax2 = G \* m1 \* (x1 - x2) / r12^3 + G \* m3 \* (x3 - x2) / r23^3;

ay2 = G \* m1 \* (y1 - y2) / r12^3 + G \* m3 \* (y3 - y2) / r23^3;

ax3 = G \* m1 \* (x1 - x3) / r13^3 + G \* m2 \* (x2 - x3) / r23^3;

ay3 = G \* m1 \* (y1 - y3) / r13^3 + G \* m2 \* (y2 - y3) / r23^3;

% Assembling the derivative vector

z = [vx1, ax1, vy1, ay1, vx2, ax2, vy2, ay2, vx3, ax3, vy3, ay3];

end

**A graph of a problem

Description automatically generated**

**Problem 2**

**3**

clear;

clc;

differential\_equations\_solver

function print\_table(t, y)

fprintf('%5s %12s\n', 't', 'Approximation');

for i = 1:length(t)

fprintf('%5.1f %12.6f\n', t(i), y(i));

end

end

function differential\_equations\_solver

h = 0.1;

tspan = [0, 1];

y0 = 1;

[t\_a, y\_a] = rk4(@(t, y) diff\_eqn\_a(t), tspan, y0, h);

[t\_b, y\_b] = rk4(@(t, y) diff\_eqn\_b(t, y), tspan, y0, h);

[t\_c, y\_c] = rk4(@(t, y) diff\_eqn\_c(t, y), tspan, y0, h);

[t\_d, y\_d] = rk4(@(t, y) diff\_eqn\_d(t, y), tspan, y0, h);

[t\_e, y\_e] = rk4(@(t, y) diff\_eqn\_e(t, y), tspan, y0, h);

[t\_f, y\_f] = rk4(@(t, y) diff\_eqn\_f(t, y), tspan, y0, h);

fprintf('Differential Equation (a):\n');

print\_table(t\_a, y\_a);

fprintf('\nDifferential Equation (b):\n');

print\_table(t\_b, y\_b);

fprintf('\nDifferential Equation (c):\n');

print\_table(t\_c, y\_c);

fprintf('\nDifferential Equation (d):\n');

print\_table(t\_d, y\_d);

fprintf('\nDifferential Equation (e):\n');

print\_table(t\_e, y\_e);

fprintf('\nDifferential Equation (f):\n');

print\_table(t\_f, y\_f);

end

function dydt = diff\_eqn\_a(t)

dydt = t;

end

function dydt = diff\_eqn\_b(t, y)

dydt = t^2 \* y;

end

function dydt = diff\_eqn\_c(t, y)

dydt = 2 \* (t + 1) \* y;

end

function dydt = diff\_eqn\_d(t, y)

dydt = 5 \* t^4 \* y;

end

function dydt = diff\_eqn\_e(t, y)

dydt = 1 / y^2;

end

function dydt = diff\_eqn\_f(t, y)

dydt = t^3 / y^2;

end

% Fourth-order Runge-Kutta method

function [t, y] = rk4(dydt, tspan, y0, h)

t = tspan(1):h:tspan(2);

y = zeros(size(t));

y(1) = y0;

for i = 1:length(t) - 1

k1 = h \* feval(dydt, t(i), y(i));

k2 = h \* feval(dydt, t(i) + h/2, y(i) + k1/2);

k3 = h \* feval(dydt, t(i) + h/2, y(i) + k2/2);

k4 = h \* feval(dydt, t(i) + h, y(i) + k3);

y(i+1) = y(i) + (k1 + 2\*k2 + 2\*k3 + k4)/6;

end

end

**Differential Equation (a):**

**t Approximation**

**0.0 1.000000**

**0.1 1.005000**

**0.2 1.020000**

**0.3 1.045000**

**0.4 1.080000**

**0.5 1.125000**

**0.6 1.180000**

**0.7 1.245000**

**0.8 1.320000**

**0.9 1.405000**

**1.0 1.500000**

**Differential Equation (b):**

**t Approximation**

**0.0 1.000000**

**0.1 1.000333**

**0.2 1.002670**

**0.3 1.009041**

**0.4 1.021563**

**0.5 1.042547**

**0.6 1.074655**

**0.7 1.121126**

**0.8 1.186095**

**0.9 1.275069**

**1.0 1.395612**

**Differential Equation (c):**

**t Approximation**

**0.0 1.000000**

**0.1 1.233674**

**0.2 1.552695**

**0.3 1.993687**

**0.4 2.611633**

**0.5 3.490211**

**0.6 4.758552**

**0.7 6.618827**

**0.8 9.392252**

**0.9 13.596905**

**1.0 20.081267**

**Differential Equation (d):**

**t Approximation**

**0.0 1.000000**

**0.1 1.000010**

**0.2 1.000321**

**0.3 1.002434**

**0.4 1.010294**

**0.5 1.031745**

**0.6 1.080865**

**0.7 1.183021**

**0.8 1.387744**

**0.9 1.804843**

**1.0 2.717976**

**Differential Equation (e):**

**t Approximation**

**0.0 1.000000**

**0.1 1.091394**

**0.2 1.169608**

**0.3 1.238564**

**0.4 1.300593**

**0.5 1.357210**

**0.6 1.409461**

**0.7 1.458101**

**0.8 1.503696**

**0.9 1.546681**

**1.0 1.587402**

**Differential Equation (f):**

**t Approximation**

**0.0 1.000000**

**0.1 1.000025**

**0.2 1.000400**

**0.3 1.002021**

**0.4 1.006359**

**0.5 1.015387**

**0.6 1.031404**

**0.7 1.056744**

**0.8 1.093405**

**0.9 1.142696**

**1.0 1.205073**

**>>**

**7**

clear;

clc;

dydt = @(t, y) sin(y);

tspan = [0, 4];

y0s = [0, 100];

h\_exact = 0.1 \* 2^-5;

[t\_exact, y\_exact] = rk4(dydt, tspan, y0s(1), h\_exact);

y\_exact\_ref = interp1(t\_exact, y\_exact, tspan(1):h\_exact:tspan(2), 'spline');

errors = zeros(6, 2);

for j = 1:length(y0s)

y0 = y0s(j);

figure;

hold on;

for k = 0:5

h = 0.1 \* 2^(-k);

[t, y] = rk4(dydt, tspan, y0, h);

y\_exact\_current = interp1(t\_exact, y\_exact, t, 'spline');

errors(k+1, j) = max(abs(y - y\_exact\_current));

if k == 0 || k == 5

plot(t, y, 'DisplayName', sprintf('h = %.4f', h));

end

end

title(sprintf('Runge-Kutta Approximations with y(0) = %d', y0));

xlabel('t');

ylabel('y(t)');

legend show;

plot(t\_exact, y\_exact, 'k', 'LineWidth', 1.5, 'DisplayName', 'Reference Solution');

hold off;

end

% Log-log plot of the error using the reference "exact" solution

figure;

loglog(0.1 \* 2.^(-[0:5]), errors, '-o');

title('Log-log plot of the error');

xlabel('Step size (h)');

ylabel('Error');

legend({'y0 = 0', 'y0 = 100'}, 'Location', 'southwest');

function [t, y] = rk4(f, tspan, y0, h)

t = tspan(1):h:tspan(2);

if t(end) < tspan(2)

t(end+1) = tspan(2);

end

y = zeros(size(t));

y(1) = y0;

for i = 1:(length(t) - 1)

k1 = h \* f(t(i), y(i));

k2 = h \* f(t(i) + 0.5 \* h, y(i) + 0.5 \* k1);

k3 = h \* f(t(i) + 0.5 \* h, y(i) + 0.5 \* k2);

k4 = h \* f(t(i) + h, y(i) + k3);

y(i + 1) = y(i) + (k1 + 2 \* k2 + 2 \* k3 + k4) / 6;

end

end

**A screenshot of a computer

Description automatically generated**

**A graph with numbers and lines

Description automatically generated**

**A graph on a white background

Description automatically generated**

**11**

clear;

clc;

function lorenz\_rk4

s = 10;

r = 28;

b = 8/3;

x0 = 5;

y0 = 5;

z0 = 5;

tspan = [0, 50];

h = 0.001;

lorenz\_sys = @(t, Y) [

s \* (Y(2) - Y(1));

r \* Y(1) - Y(2) - Y(1) \* Y(3);

Y(1) \* Y(2) - b \* Y(3)

];

N = ceil((tspan(2) - tspan(1)) / h);

T = (0:N-1) \* h;

Y = zeros(N, 3);

Y(1, :) = [x0, y0, z0];

for i = 1:N-1

k1 = h \* lorenz\_sys(T(i), Y(i, :));

k2 = h \* lorenz\_sys(T(i) + h/2, Y(i, :) + k1/2);

k3 = h \* lorenz\_sys(T(i) + h/2, Y(i, :) + k2/2);

k4 = h \* lorenz\_sys(T(i) + h, Y(i, :) + k3);

Y(i+1, :) = Y(i, :) + (k1 + 2\*k2 + 2\*k3 + k4) / 6;

end

figure;

hold on;

axis([-20 20 -30 30 0 50]);

view(3);

title('Lorenz Attractor');

xlabel('X');

ylabel('Y');

zlabel('Z');

for i = 1:N

plot3(Y(1:i, 1), Y(1:i, 2), Y(1:i, 3), 'b');

drawnow limitrate;

if i > 1

plot3(Y(i-1:i, 1), Y(i-1:i, 2), Y(i-1:i, 3), 'r', 'LineWidth', 2);

end

if i < N

pause(h);

end

end

end

**13**

clear;

clc;

lorenz\_trajectory\_symbols

function lorenz\_trajectory\_symbols

s = 10; r = 28; b = 8/3;

h = 0.001;

t\_final = 20;

delta = 1e-5;

Y0 = [5, 5, 5];

Y0\_perturbed = Y0 + [delta, 0, 0];

[T, Y] = lorenz\_rk4(s, r, b, Y0, t\_final, h);

[~, Y\_perturbed] = lorenz\_rk4(s, r, b, Y0\_perturbed, t\_final, h);

symbols = Y(:,1) > 0;

symbols\_perturbed = Y\_perturbed(:,1) > 0;

agreement = find(symbols == symbols\_perturbed);

agreement\_time\_units = numel(agreement) \* h;

fprintf('The symbol sequences of the two trajectories agree for %.4f time units.\n', agreement\_time\_units);

end

function [T, Y] = lorenz\_rk4(s, r, b, Y0, t\_final, h)

lorenz\_eqns = @(t, Y) [-s\*(Y(1)-Y(2)); r\*Y(1)-Y(2)-Y(1)\*Y(3); Y(1)\*Y(2)-b\*Y(3)];

steps = ceil(t\_final / h);

T = 0:h:t\_final;

Y = zeros(length(T), 3);

Y(1,:) = Y0;

% Perform the RK4 integration

for i = 1:(length(T) - 1)

k1 = lorenz\_eqns(T(i), Y(i,:));

k2 = lorenz\_eqns(T(i) + 0.5 \* h, Y(i,:) + 0.5 \* h \* k1');

k3 = lorenz\_eqns(T(i) + 0.5 \* h, Y(i,:) + 0.5 \* h \* k2');

k4 = lorenz\_eqns(T(i) + h, Y(i,:) + h \* k3');

Y(i+1,:) = Y(i,:) + (h/6) \* (k1 + 2\*k2 + 2\*k3 + k4)';

end

end

**>>The symbol sequences of the two trajectories agree for 19.5980 time units.**

**Problem 3**

**1**

clear;

clc;

minimal\_standard\_rng\_volume\_approximation;

function minimal\_standard\_rng\_volume\_approximation

a = 16807;

m = 2^31 - 1;

x = 1;

N = 1e6;

points = zeros(N, 3);

for i = 1:N

x = mod(a \* x, m);

points(i, 1) = x / m;

x = mod(a \* x, m);

points(i, 2) = x / m;

x = mod(a \* x, m);

points(i, 3) = x / m;

end

radius = 0.04;

center = [1/3, 1/3, 1/2];

count\_inside = sum(sum((points - center).^2, 2) <= radius^2);

volume\_approx = count\_inside / N;

actual\_volume = (4/3) \* pi \* radius^3;

fprintf('Actual volume of the sphere: %f\n', actual\_volume);

fprintf('Monte Carlo approximation: %f\n', volume\_approx);

fprintf('Error: %f\n', abs(volume\_approx - actual\_volume));

end

**>>** **Actual volume of the sphere: 0.000268**

**Monte Carlo approximation: 0.000273**

**Error: 0.000005**