PHYS 410 Homework 2

Gavin Pringle, 56401938

October 27, 2024

Introduction

In this homework assignment, the fourth-order Runge-Kutta method for computing numerical solutions of ODEs is explored. This is done in stages, culminating in creating a MATLAB function that numerically integrates an ODE using an algorithm that automatically varies the step size of the integrator in order to achieve a relative error tolerance.

First, a function `rk4step` is written that computes a single fourth-order Runge-Kutta step for a system of coupled first-order ODEs, returning the approximate values of the dependent variables after a defined time step. The function `rk4step` is then used in the function `rk4` which computes the solution of an initial value problem over a range of values for the independent variable, done by taking multiple fourth-order Runge-Kutta steps in a loop. Lastly, the function `rk4ad` is written which finds the numerical solution of an initial value problem by comparing the results of fourth-order Runge-Kutta steps of different sizes and then varying the step size as until the error in the approximation is below a specified relative tolerance.

Review of Theory

Casting systems of ODEs in first-order form

In order to solve complicated ODEs numerically, it is useful to first cast them in a canonical form that is easier for a computer program to understand. Any ODE defining the function $y(x)$ that is of the form

$$f(x, y, y', y'', y^{(3)}, \ldots, y^{(N)}) = 0$$

can be rewritten as a system of N coupled first-order ODEs for the functions $y_i(x), \quad i = 1, 2, 3, \ldots, N$:

$$y_i'(x) \equiv \frac{dy_i}{dx}(x) = f_i(x, y_1, y_2, y_3, \ldots, y_N)$$

where f_i are known functions of x and y_i. This is equivalent to

$$\mathbf{y}' = \mathbf{f}(x, \mathbf{y}) \quad \text{where} \quad \mathbf{y} \equiv (y_1, y_2, y_3, \ldots, y_N)$$

For example, the function $y^{(4)}(x) = f(x)$ can be written as

$$y_3' = f, \quad y' = y_1, \quad y_1' = y2, \quad y_2' = y_3$$

The fourth-order Runge-Kutta step

The fourth-order Runge-Kutta step is defined as:

$$y_i(x_0 + h) = y_i(x_0) + \frac{h}{6}(f_{0,i} + 2f_{1,i} + 2f_{2,i} + f_{3,i})$$

1

where

$$f_0 = f(x_0, y_0)$$

$$f_1 = f\left(x_0 + \frac{h}{2}, y_0 + \frac{h}{2}f_0\right)$$

$$f_2 = f\left(x_0 + \frac{h}{2}, y_0 + \frac{h}{2}f_1\right)$$

$$f_3 = f\left(x_0 + h, y_0 + hf_2\right)$$

ADD i's to above eqn!!!!!

This can be understood as

Numerical Approach

Consecutive fourth-order Runge-Kutta steps

Adaptive step sizing

Implementation

ODE test functions

Results

rk4step.m Output

rk4.m Output

rk4ad.m Output

Conclusions

Appendix A - rk4step.m Code

```matlab
%% Problem 1 - Single Fourth Order Runge-Kutta Step

% Function that computes a single fourth order Runge-Kutta Step.
%
% Inputs
%        fcn:     Function handle for right hand sides of ODEs (returns
%                 length-n column vector).
%        t0:      Initial value of independent variable.
%        dt:      Time step.
%        y0:      Initial values (length-n column vector).
%
% Output
%        yout:    Final values (length-n column vector)
function yout = rk4step(fcn, t0, dt, y0)
    % Compute terms in RK step
    f0 = fcn(t0, y0);
    f1 = fcn(t0 + dt/2, y0 + (dt/2)*f0);
    f2 = fcn(t0 + dt/2, y0 + (dt/2)*f1);
    f3 = fcn(t0 + dt, y0 + dt*f2);
    % Add terms to compute full RK step
    yout = y0 + (dt/6)*(f0 + 2*f1 + 2*f2 + f3);
end
```

Appendix B - trk4step.m Code

```matlab
%% Problem 1 - Test Script

close all; clear; clc;

format long;

% Function that computes right hand sides of ODEs for simple harmonic
% oscillator with unit angular frequency. For use in rk4step, rk4, and
%     rk4ad.
%
% Governing DE:                                    x" = -x
% Canonical first order dependent variables:    x1 = x, x2 = x'
% System of Equations:                          x1' = x2, x2' = -x1
%
% Inputs
%       t:        Independent variable at current time-step
%       x:        Dependent variables at current time-step (length-n column
%                 vector).
%
% Outputs
%         dxdt:  Computes the derivatives of x1 and x2 at the current
%                time-step (length-n column vector).
function dxdt = fcn_sho(t, x)
    dxdt = zeros(2,1);
    dxdt(1) = x(2);
    dxdt(2) = -x(1);
end

% Function parameters for exact solution of x(t) = sin(t)
x0 = [0; 1];     % Initial conditions
t0 = 0;          % Initial time
% Vector of linearly increasing time-step lengths
dt = linspace(0.01, 0.3, 1000);

% Run Runge-Kutta step at various time steps
xout = zeros(2, length(dt));
for i = 1:length(dt)
    xout(:,i) = rk4step(@fcn_sho, t0, dt(i), x0);
end

% Calculate the error at each time step length using the known exact
%     solution
errors = abs(xout(1,:) - sin(dt));

% Plot error as a function of dt and compare to C*t^5
hold on;
plot(dt, errors, "Color", 'r', "LineWidth", 3);
C = 8.3e-3;
plot(dt, C*dt.^5, "--", "Color", 'b', "LineWidth", 3);
title("Magnitude of error vs. time step length dt shown to scale as dt^5");
xlabel("Time step length dt");
ylabel("Magnitude of error");
legend(["Error", "C * t^5"], 'location', 'best');
ax = gca;
ax.FontSize = 12;
```

Appendix C - rk4.m Code

```matlab
%% Problem 2 - Runge-Kutta System of ODEs Integrator

% Function that numerically computes the solution to a system of ODEs
% over a given period of time using a fourth-order Runge-Kutta method.
%
% Inputs
%       fcn:     Function handle for right hand sides of ODEs (returns
%                length-n column vector)
%       tspan:   Vector of output times (length nout).
%       y0:      Initial values (length-n column vector).
%
% Outputs
%       tout:    Vector of output times.
%       yout:    Output values (nout x n array. The ith column of yout
%                contains the nout values of the ith dependent variable).
function [tout yout] = rk4(fcn, tspan, y0)
    % Number of equations in ODE system
    n = max(size(y0));
    % Number of time-steps
    nout = max(size(tspan));

    % Initialize array for output values
    yout = zeros(nout, n);
    yout(1,:) = y0.';

    % Integrate ODE
    for i = 2:nout
        % Step size for the current step
        dt = tspan(i) - tspan(i-1);
        % Compute the values of the dependent variables at the next step
        yout(i,:) = rk4step(fcn, tspan(i-1), dt, yout(i-1,:).').';
    end

    % Generate array of output values
    tout = tspan;
end
```

Appendix D - trk4_sho.m Code

```matlab
%% Problem 2 – Test Script – Simple Harmonic Oscillator

close all; clear; clc;

format long;

% Function that computes right hand sides of ODEs for simple harmonic
% oscillator with unit angular frequency. For use in rk4step, rk4, and
%    rk4ad.
%
% Governing DE:                                    x" = -x
% Canonical first order dependent variables:     x1 = x, x2 = x'
% System of Equations:                           x1' = x2, x2' = -x1
%
% Inputs
%       t:       Independent variable at current time-step
%       x:       Dependent variables at current time-step (length-n column
%                vector).
%
% Outputs
%       dxdt:  Computes the derivatives of x1 and x2 at the current
%                time-step (length-n column vector).
function dxdt = fcn_sho(t, x)
    dxdt = zeros(2,1);
    dxdt(1) = x(2);
    dxdt(2) = -x(1);
end

% Function parameters for exact solution of x(t) = sin(t)
x0 = [0; 1];              % Initial conditions
t0 = 0; tf = 3*pi;        % Start and end times

% Vector of output times for each discretization level
tspan6 = linspace(t0, tf, 2^6 + 1);
tspan7 = linspace(t0, tf, 2^7 + 1);
tspan8 = linspace(t0, tf, 2^8 + 1);

% Compute ODE numerical solution at each discretization level
[tout6 xout6] = rk4(@fcn_sho, tspan6, x0);
[tout7 xout7] = rk4(@fcn_sho, tspan7, x0);
[tout8 xout8] = rk4(@fcn_sho, tspan8, x0);

% Plot the solutions at each discretization level
fig1 = figure(1);
hold on
plot(tout6, xout6(:,1), "LineWidth", 2);
plot(tout7, xout7(:,1), "LineWidth", 2);
plot(tout8, xout8(:,1), "LineWidth", 2);
title("Numerical solutions to SHO ODE at various discretization levels");
xlabel("Independent Variable t");
ylabel("Dependent Variable x");
legend(["l = 6", "l = 7", "l = 8"], 'location', 'best');
ax = gca;
ax.FontSize = 12;

% Compute the errors at each time step for each discretization level
```

```matlab
56  errors6 = xout6(:,1) − sin(tout6).';
57  errors7 = xout7(:,1) − sin(tout7).';
58  errors8 = xout8(:,1) − sin(tout8).';
59
60  % Plot the scaled errors for each discretization level
61  fig2 = figure(2);
62  hold on
63  plot(tout6, errors6, "LineWidth", 2);
64  plot(tout7, 2^4*errors7, "LineWidth", 2);
65  plot(tout8, 4^4*errors8, "LineWidth", 2);
66  grid on
67  title({"Scaled errors of numerical solutions to SHO ODE at ", ...
68          "various discretization levels"});
69  xlabel("Independent Variable t");
70  ylabel("Scaled error");
71  legend(["error @ l=6", "2^4 * error @ l=7", "4^4 * error @ l=8"], ...
72          'location', 'best');
73  ax = gca;
74  ax.FontSize = 12;
```

Appendix E - trk4_vdp.m Code

```matlab
%% Problem 2 - Test Script - Van der Pol oscillator

close all; clear; clc;

format long;

% Function that computes right hand sides of ODEs for Van der Pol
% Oscillator. Following Tsatsos: https://arxiv.org/pdf/0803.1658
%
% Governing DE: x" = -x - a(x^2 - 1)x'
% Canonical first order dependent variables: x1 = x, x2 = x'
% System of Equations:
%          x1' = x2
%          x2' = -x1 - a(x1^2 - 1)*x2
%
% Inputs
%        t:       Independent variable at current time-step
%        x:       Dependent variables at current time-step (length-n column
%                 vector).
%
% Outputs
%        dxdt:   Computes the derivatives of x1 and x2 at the current
%                time-step (length-n column vector).
function dxdt = fcn_vdp(t, x)
    global a;
    dxdt = ones(2,1);
    dxdt(1) = x(2);
    dxdt(2) = -x(1) - a*(x(1)^2 - 1)*x(2);
end

% Function parameters
x0 = [0; -6];         % Initial conditions
t0 = 0; tf = 100;     % Start and end times
global a; a = 5;      % Adjustable parameter

% Discretization level
level = 12;
tspan = linspace(t0, tf, 2^level + 1);

% Compute ODE numerical solution
[tout xout] = rk4(@fcn_vdp, tspan, x0);

% Plot position vs time
fig1 = figure(1);
plot(tout, xout(:,1), "LineWidth", 2)
title("Numerical solution of Van der Pol oscillator ODE - Position x vs.
    Time t");
xlabel("Independent Variable - Time t");
ylabel("Dependent Variable - Position x");
ax = gca;
ax.FontSize = 12;

% Plot phase space evolution
fig2 = figure(2);
plot(xout(:,1), xout(:,2), "LineWidth", 2)
title({"Phase space evolution of Van der Pol oscillator ODE", ...
```

```
56            "Velocity dx/dt vs. Position x"});
57  xlabel("Position x");
58  ylabel("Velocity dx/dt");
59  ax = gca;
60  ax.FontSize = 12;
```

Appendix F - rk4ad.m Code

```matlab
%% Problem 3 - Adaptive Fourth Order Runge-Kutta System of ODEs Integrator

% Function that numerically computes the solution to a system of ODEs
% over a given period of time using a fourth-order Runge-Kutta method
% with adaptive steps sizes to ensure a relative tolerance is reached.
%
% Inputs
%        fcn:     Function handle for right hand sides of ODEs (returns
%                 length-n column vector)
%        tspan:   Vector of output times (length nout vector).
%        reltol:  Relative tolerance parameter.
%        y0:      Initial values (length-n column vector).
%
% Outputs
%        tout:    Output times (length-nout column vector, elements
%                 identical to tspan).
%        yout:    Output values (nout x n array. The ith column of yout
%                 contains the nout values of the ith dependent variable).
function [tout yout] = rk4ad(fcn, tspan, reltol, y0)
    % Number of equations in ODE system
    n = max(size(y0));
    % Number of time-steps
    nout = max(size(tspan));
    % Lower bound on step size
    floor = 1.0e-4;

    % Initialize array for output values
    yout = zeros(nout, n);
    yout(1,:) = y0.';

    % Integrate ODE
    for i = 2:nout
        % Compute coarse rk4step arguments
        tprev = tspan(i-1);
        yprev = yout(i-1,:).';
        dt = tspan(i) - tspan(i-1);

        % Compute fine and coarse approximations for y(tprev + dt)
        yc = rk4step(fcn, tprev, dt, yprev);
        if dt/2 < floor
            % If fine step is lower than floor, cannot narrow down any
                further
            yout(i,:) = yc.';
            continue;
        end
        yhalf = rk4step(fcn, tprev, dt/2, yprev);
        yf = rk4step(fcn, tprev + dt/2, dt/2, yhalf);

        % Check if error meets relative tolerance parameter
        if abs((yc - yf)/yf) < reltol
            yout(i,:) = yf.';
            continue;
        else
            % Iteratively compute yf at repeatedly halved dt sizes
            % until reltol is met or floor is reached
            j = 2;
```

```matlab
56                  while dt/(2^j) > floor % Decrease step size by half each
                        iteration
57                      yc = yf;
58                      yf = yprev;
59                      for k = 0:2^j - 1 % Number of steps to get to tprev + dt
60                          yf = rk4step(fcn, tprev + k*dt/(2^j), dt/(2^j), yf);
61                      end
62
63                      if abs((yc - yf)/yf) < reltol
64                          break;
65                      end
66
67                      j = j + 1;
68                  end
69                  yout(i,:) = yf.';
70              end
71      end
72
73      % Generate array of output values
74      tout = tspan;
75  end
```

Appendix G - trk4ad_sho.m Code

```matlab
1  %% Problem 3 − Test Script − Simple Harmonic Oscillator
2
3  close all; clear; clc;
4
5  format long;
6
7  % Function that computes right hand sides of ODEs for simple harmonic
8  % oscillator with unit angular frequency. For use in rk4step, rk4, and
       rk4ad.
9  %
10 % Governing DE:                                x" = −x
11 % Canonical first order dependent variables:   x1 = x, x2 = x'
12 % System of Equations:                         x1' = x2, x2' = −x1
13 %
14 % Inputs
15 %       t:       Independent variable at current time−step
16 %       x:       Dependent variables at current time−step (length−n column
17 %                vector).
18 %
19 % Outputs
20 %       dxdt:    Computes the derivatives of x1 and x2 at the current
21 %                time−step (length−n column vector).
22 function dxdt = fcn_sho(t, x)
23     dxdt = zeros(2,1);
24     dxdt(1) = x(2);
25     dxdt(2) = −x(1);
26 end
27
28 % Function parameters for exact solution of x(t) = sin(t)
29 x0 = [0; 1];                              % Initial conditions
30 tspan = linspace(0.0, 3.0 * pi, 65);     % Vector of output times
31
32 % Compute ODE numerical solution at each relative tolerance
33 [tout5 xout5] = rk4ad(@fcn_sho, tspan, 1.0e−5, x0);
34 [tout7 xout7] = rk4ad(@fcn_sho, tspan, 1.0e−7, x0);
35 [tout9 xout9] = rk4ad(@fcn_sho, tspan, 1.0e−9, x0);
36 [tout11 xout11] = rk4ad(@fcn_sho, tspan, 1.0e−11, x0);
37
38 % Plot the solutions at each relative tolerance
39 fig1 = figure(1);
40 hold on
41 plot(tout5, xout5(:,1), "LineWidth", 2);
42 plot(tout7, xout7(:,1), "LineWidth", 2);
43 plot(tout9, xout9(:,1), "LineWidth", 2);
44 plot(tout11, xout11(:,1), "LineWidth", 2);
45 title("Numerical solutions to SHO ODE from rk4ad at various relative
       tolerances");
46 xlabel("Independent Variable t");
47 ylabel("Dependent Variable x");
48 legend(["reltol = 1.0e−5", "reltol = 1.0e−7", "reltol = 1.0e−9", ...
49         "reltol = 1.0e−11"], 'location', 'best');
50 ax = gca;
51 ax.FontSize = 12;
52
53 % Compute the errors at each time step for each discretization level
54 errors5 = xout5(:,1) − sin(tout5).';
```

```matlab
55  errors7 = xout7 (:,1) - sin(tout7).';
56  errors9 = xout9 (:,1) - sin(tout9).';
57  errors11 = xout11 (:,1) - sin(tout11).';
58
59  % Plot the errors for each relative tolerance
60  fig2 = figure (2);
61  hold on
62  plot(tout5 , errors5 , "LineWidth", 2);
63  plot(tout7 , errors7 , "LineWidth", 2);
64  plot(tout9 , errors9 , "LineWidth", 2);
65  plot(tout11 , errors11 , "--", "LineWidth", 2);
66  grid on
67  title ({"Errors of numerical solutions to SHO ODE at ", ...
68          "various rk4ad relative tolerances"});
69  xlabel("Independent Variable t");
70  ylabel("Error");
71  legend (["reltol = 1.0e-5", "reltol = 1.0e-7", "reltol = 1.0e-9", ...
72          "reltol = 1.0e-11"], 'location', 'best');
73  ax = gca;
74  ax.FontSize = 12;
```

Appendix H - trk4ad_vdp.m Code

```matlab
1  %% Problem 3 - Test Script - Van der Pol oscillator
2
3  close all; clear; clc;
4
5  format long;
6
7  % Function that computes right hand sides of ODEs for Van der Pol
8  % Oscillator. Following Tsatsos: https://arxiv.org/pdf/0803.1658
9  %
10 % Governing DE: x" = -x - a(x^2 - 1)x'
11 % Canonical first order dependent variables: x1 = x, x2 = x'
12 % System of Equations:
13 %        x1' = x2
14 %        x2' = -x1 - a(x1^2 - 1)*x2
15 %
16 % Inputs
17 %     t:       Independent variable at current time-step
18 %     x:       Dependent variables at current time-step (length-n column
19 %              vector).
20 %
21 % Outputs
22 %        dxdt:  Computes the derivatives of x1 and x2 at the current
23 %               time-step (length-n column vector).
24 function dxdt = fcn_vdp(t, x)
25     global a;
26     dxdt = ones(2,1);
27     dxdt(1) = x(2);
28     dxdt(2) = -x(1) - a*(x(1)^2 - 1)*x(2);
29 end
30
31 % Function parameters
32 x0 = [0; -6];                         % Initial conditions
33 tspan = linspace(0.0, 100, 4097);     % Vector of output times
34 global a; a = 5;                      % Adjustable parameter
35 reltol = 1.0e-10;                     % Relative tolerance
36
37 % Compute ODE numerical solution
38 [tout xout] = rk4ad(@fcn_vdp, tspan, reltol, x0);
39
40 % Plot position vs time
41 fig1 = figure(1);
42 plot(tout, xout(:,1), "LineWidth", 2, "Color", 'm')
43 title({"Numerical solution of Van der Pol oscillator ODE using rk4ad", ...
44       "Position x vs. Time t, Relative tolerance = 1.0e-10"});
45 xlabel("Independent Variable - Time t");
46 ylabel("Dependent Variable - Position x");
47 ax = gca;
48 ax.FontSize = 12;
49
50 % Plot phase space evolution
51 fig2 = figure(2);
52 plot(xout(:,1), xout(:,2), "LineWidth", 2, "Color", 'm')
53 title({"Phase space evolution of Van der Pol oscillator ODE using rk4ad",
      ...
54       "Velocity dx/dt vs. Position x, Relative tolerance = 1.0e-10"});
55 xlabel("Position x");
```

```matlab
56  ylabel(" Velocity dx/dt");
57  ax = gca;
58  ax.FontSize = 12;
```