

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

The fourth-order Runge-Kutta step

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

```
1 %% Problem 1 — Single Fourth Order Runge–Kutta Step
2
3 % Function that computes a single fourth order Runge–Kutta Step.
4 %
5 % Inputs
6 %     fcn:      Function handle for right hand sides of ODEs (returns
7 %              length–n column vector).
8 %     t0:      Initial value of independent variable.
9 %     dt:      Time step.
10 %     y0:      Initial values (length–n column vector).
11 %
12 % Output
13 %     yout:     Final values (length–n column vector)
14 function yout = rk4step(fcn, t0, dt, y0)
15     % Compute terms in RK step
16     f0 = fcn(t0, y0);
17     f1 = fcn(t0 + dt/2, y0 + (dt/2)*f0);
18     f2 = fcn(t0 + dt/2, y0 + (dt/2)*f1);
19     f3 = fcn(t0 + dt, y0 + dt*f2);
20     % Add terms to compute full RK step
21     yout = y0 + (dt/6)*(f0 + 2*f1 + 2*f2 + f3);
22 end
```

Appendix B - trk4step.m Code

```

1 %% Problem 1 – Test Script
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
9 % rk4ad.
10 % Governing DE:  $x'' = -x$ 
11 % Canonical first order dependent variables:  $x_1 = x, x_2 = x'$ 
12 % System of Equations:  $x_1' = x_2, x_2' = -x_1$ 
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 t0 = 0; % Initial time
31 % Vector of linearly increasing time-step lengths
32 dt = linspace(0.01, 0.3, 1000);
33
34 % Run Runge-Kutta step at various time steps
35 xout = zeros(2, length(dt));
36 for i = 1:length(dt)
37     xout(:,i) = rk4step(@fcn_sho, t0, dt(i), x0);
38 end
39
40 % Calculate the error at each time step length using the known exact
41 % solution
42 errors = abs(xout(1,:) - sin(dt));
43
44 % Plot error as a function of dt and compare to  $C \cdot t^5$ 
45 hold on;
46 plot(dt, errors, "Color", 'r', "LineWidth", 3);
47 C = 8.3e-3;
48 plot(dt, C*dt.^5, "--", "Color", 'b', "LineWidth", 3);
49 title("Magnitude of error vs. time step length dt shown to scale as dt^5");
50 xlabel("Time step length dt");
51 ylabel("Magnitude of error");
52 legend(["Error", "C * t^5"], 'location', 'best');
53 ax = gca;
54 ax.FontSize = 12;

```

Appendix C - rk4.m Code

```
1 %% Problem 2 – Runge–Kutta System of ODEs Integrator
2
3 % Function that numerically computes the solution to a system of ODEs
4 % over a given period of time using a fourth–order Runge–Kutta method.
5 %
6 % Inputs
7 %     fcn:      Function handle for right hand sides of ODEs (returns
8 %              length–n column vector)
9 %     tspan:    Vector of output times (length nout).
10 %     y0:       Initial values (length–n column vector).
11 %
12 % Outputs
13 %     tout:     Vector of output times.
14 %     yout:     Output values (nout x n array. The ith column of yout
15 %              contains the nout values of the ith dependent variable).
16 function [tout yout] = rk4(fcn, tspan, y0)
17     % Number of equations in ODE system
18     n = max(size(y0));
19     % Number of time–steps
20     nout = max(size(tspan));
21
22     % Initialize array for output values
23     yout = zeros(nout, n);
24     yout(1,:) = y0.';
25
26     % Integrate ODE
27     for i = 2:nout
28         % Step size for the current step
29         dt = tspan(i) – tspan(i–1);
30         % Compute the values of the dependent variables at the next step
31         yout(i,:) = rk4step(fcn, tspan(i–1), dt, yout(i–1,:)).';
32     end
33
34     % Generate array of output values
35     tout = tspan;
36 end
```

Appendix D - trk4_sho.m Code

```

1 %% Problem 2 – 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
9 % rk4ad.
10 % Governing DE:  $x'' = -x$ 
11 % Canonical first order dependent variables:  $x_1 = x, x_2 = x'$ 
12 % System of Equations:  $x_1' = x_2, x_2' = -x_1$ 
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 t0 = 0; tf = 3*pi; % Start and end times
31
32 % Vector of output times for each discretization level
33 tspan6 = linspace(t0, tf, 2^6 + 1);
34 tspan7 = linspace(t0, tf, 2^7 + 1);
35 tspan8 = linspace(t0, tf, 2^8 + 1);
36
37 % Compute ODE numerical solution at each discretization level
38 [tout6 xout6] = rk4(@fcn_sho, tspan6, x0);
39 [tout7 xout7] = rk4(@fcn_sho, tspan7, x0);
40 [tout8 xout8] = rk4(@fcn_sho, tspan8, x0);
41
42 % Plot the solutions at each discretization level
43 fig1 = figure(1);
44 hold on
45 plot(tout6, xout6(:,1), "LineWidth", 2);
46 plot(tout7, xout7(:,1), "LineWidth", 2);
47 plot(tout8, xout8(:,1), "LineWidth", 2);
48 title("Numerical solutions to SHO ODE at various discretization levels");
49 xlabel("Independent Variable t");
50 ylabel("Dependent Variable x");
51 legend(["l = 6", "l = 7", "l = 8"], 'location', 'best');
52 ax = gca;
53 ax.FontSize = 12;
54
55 % Compute the errors at each time step for each discretization level

```

```

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

```

1 %% Problem 2 – 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:  $x_1 = x$ ,  $x_2 = x'$ 
12 % System of Equations:
13 %       $x_1' = x_2$ 
14 %       $x_2' = -x_1 - a(x_1^2 - 1)x_2$ 
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 t0 = 0; tf = 100;  % Start and end times
34 global a; a = 5;    % Adjustable parameter
35
36 % Discretization level
37 level = 12;
38 tspan = linspace(t0, tf, 2^level + 1);
39
40 % Compute ODE numerical solution
41 [tout xout] = rk4(@fcn_vdp, tspan, x0);
42
43 % Plot position vs time
44 fig1 = figure(1);
45 plot(tout, xout(:,1), "LineWidth", 2)
46 title("Numerical solution of Van der Pol oscillator ODE – Position x vs.
47       Time t");
48 xlabel("Independent Variable – Time t");
49 ylabel("Dependent Variable – Position x");
50 ax = gca;
51 ax.FontSize = 12;
52
53 % Plot phase space evolution
54 fig2 = figure(2);
55 plot(xout(:,1), xout(:,2), "LineWidth", 2)
56 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

```

1 %% Problem 3 – Adaptive Fourth Order Runge–Kutta System of ODEs Integrator
2
3 % Function that numerically computes the solution to a system of ODEs
4 % over a given period of time using a fourth–order Runge–Kutta method
5 % with adaptive steps sizes to ensure a relative tolerance is reached.
6 %
7 % Inputs
8 %     fcn:      Function handle for right hand sides of ODEs (returns
9 %              length–n column vector)
10 %     tspan:    Vector of output times (length nout vector).
11 %     reltol:   Relative tolerance parameter.
12 %     y0:       Initial values (length–n column vector).
13 %
14 % Outputs
15 %     tout:     Output times (length–nout column vector, elements
16 %              identical to tspan).
17 %     yout:     Output values (nout x n array. The ith column of yout
18 %              contains the nout values of the ith dependent variable).
19 function [tout yout] = rk4ad(fcn, tspan, reltol, y0)
20     % Number of equations in ODE system
21     n = max(size(y0));
22     % Number of time–steps
23     nout = max(size(tspan));
24     % Lower bound on step size
25     floor = 1.0e–4;
26
27     % Initialize array for output values
28     yout = zeros(nout, n);
29     yout(1,:) = y0.';
30
31     % Integrate ODE
32     for i = 2:nout
33         % Compute coarse rk4step arguments
34         tprev = tspan(i–1);
35         yprev = yout(i–1,:).';
36         dt = tspan(i) – tspan(i–1);
37
38         % Compute fine and coarse approximations for y(tprev + dt)
39         yc = rk4step(fcn, tprev, dt, yprev);
40         if dt/2 < floor
41             % If fine step is lower than floor, cannot narrow down any
42             % further
43             yout(i,:) = yc.';
44             continue;
45         end
46         yhalf = rk4step(fcn, tprev, dt/2, yprev);
47         yf = rk4step(fcn, tprev + dt/2, dt/2, yhalf);
48
49         % Check if error meets relative tolerance parameter
50         if abs((yc – yf)/yf) < reltol
51             yout(i,:) = yf.';
52             continue;
53         else
54             % Iteratively compute yf at repeatedly halved dt sizes
55             % until reltol is met or floor is reached
56             j = 2;

```

```

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

```

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
9 % rk4ad.
10 % Governing DE:  $x'' = -x$ 
11 % Canonical first order dependent variables:  $x_1 = x, x_2 = x'$ 
12 % System of Equations:  $x_1' = x_2, x_2' = -x_1$ 
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
46       tolerances");
47 xlabel("Independent Variable t");
48 ylabel("Dependent Variable x");
49 legend(["reltol = 1.0e-5", "reltol = 1.0e-7", "reltol = 1.0e-9", ...
50        "reltol = 1.0e-11"], 'location', 'best');
51 ax = gca;
52 ax.FontSize = 12;
53
54 % Compute the errors at each time step for each discretization level
55 errors5 = xout5(:,1) - sin(tout5).';

```

```

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(["reitol = 1.0e-5", "reitol = 1.0e-7", "reitol = 1.0e-9", ...
72        "reitol = 1.0e-11"], 'location', 'best');
73 ax = gca;
74 ax.FontSize = 12;

```

Appendix H - trk4ad_vdp.m Code

```

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:  $x_1 = x$ ,  $x_2 = x'$ 
12 % System of Equations:
13 %       $x_1' = x_2$ 
14 %       $x_2' = -x_1 - a(x_1^2 - 1)x_2$ 
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        "...
55        "Velocity dx/dt vs. Position x, Relative tolerance = 1.0e-10"});
56 xlabel("Position x");

```

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