Computational Engineering - Engr 8103
Problem Set #6

Allen Spain
avs81684@uga.edu

University of Georgia — 22 October 2019

Problem 1

Show that the following is another valid second order Runge-Kutta method:
From the 2nd order Taylor series method:
Where:

$$\begin{aligned} x &\to x(t) \\ f &\to f(t,x) \\ f_t &\to \frac{\partial f}{\partial t} \\ f_x &\to \frac{\partial f}{\partial x} \end{aligned}$$

$$\boxed{x(t+h) = x + hf + \frac{h^2}{2}(f_t + ff_x)}$$

From the 2nd order Runge-Kutta method:

$$\begin{aligned} K_1 &= hf \\ K_2 = hf(t + \frac{2}{3}h, x + \frac{2}{3}K_1) &= h(f + \frac{2}{3}h(f_t + ff_x)) \\ x(t+h) &= x(t) + (K_1 + 3K_2)/4 \\ &= x(t) + \frac{hf(t,x)}{4} + 3/4(h(f + \frac{2}{3}h(f_t + ff_x))) \\ &= x(t) + \frac{hf}{4} + \frac{3hf}{4} + \frac{h^2}{2}(f_t + ff_x) \\ &\boxed{x(t+h) = x(t) + hf + \frac{h^2}{2}(f_t + ff_x)} \end{aligned}$$

Problem 2

Show that when the fourth-order Runge-Kutta method is applied to the problem $x' = 2x$, the formula for advancing this solution will be

$$x(t+h) = \left[1 + 2h + 2h^2 + \frac{4}{3}h^3 + \frac{2}{3}h^4\right]x(t)$$

From the 4th order Runge-Kutta method:

$$\begin{aligned}
K_1 &= hf = 2xh \\
K_2 &= hf(t+\frac{h}{2}, x+\frac{K_1}{2}) = 2xh + 2h^2x \\
K_3 &= hf(t+\frac{h}{2}, x+\frac{K_2}{2}) = 2xh + 2xh^2 + 2xh^3 \\
K_4 &= hf(t+\frac{h}{2}, x+\frac{K_3}{2}) = 2xh + 4xh^2 + 4xh^3 + 4xh^4 \\
x(t+h) &= x(t) + (K_1 + 2K_2 + 2K_3 + K_4)/6 \\
&= x(t) + (2xh + 2(2xh + 2xh^2) \\
+2(2xh + 2xh^2 + 2xh^3) &+ (2xh + 4xh^2 + 4xh^3 + 4xh^4))/6 \\
x(t+h) &= x(t)[1 + 2h + 2h^2 + \frac{4}{3}h^3 + \frac{2}{3}h^4]
\end{aligned}$$

Problem 3

Consider the following ODE:

$$x' = -y \quad x(0) = 1 y' = x \quad y(0) = 0$$

(a) Write a Matlab code that solves this equation system on the interval [0, 10] using - Second order Taylor series method - Second order Runge-Kutta method (does not matter which one you use) - Fourth order Runge-Kutta method

```
%
% Computational Engr 8103
% Allen Spain
%
function ODEComparison

fx_x = 0;
fx_y = -1;
fy_x = 1;
fy_y = 0;

f1 = @(y) -y; % function
ff1 = @(y) fx_y + fy_x*(f1(y)); % function 2

f2 = @(x) x;
ff2 = @(x) fy_x + fx_y*f1(x);

% initial conditions
range = 10;
h = 0.5;
t(1) = 0;

x(1) = 1;
y(1) = 0;

for i=1:range/h

  %  2nd order Taylor Series method
  x(i+1) = x(i) + h*f1(y(i)) + h^2*ff1(y(i))/2;
  y(i+1) = y(i) + h*f2(x(i)) + h^2*ff2(x(i))/2;

  % 2nd order Runge-Kutta
```

```
  % x' = -y
  K1 = h*f1(y(i));
  K2 = h*(f1(y(i)) + (h*0.5*fx_y) + (0.5*K1*fx_x));
  Rx(i+1) = x(i) + ( K1 + K2 )/2;

  % y' = x
  K1 = h*f2(x(i));
  K2 = h*(f2(x(i)) + (h*0.5*fy_x) + (0.5*K1*fy_y));
  Ry(i+1) = y(i) + ( K1 + K2 )/2;

  % 4th order Runge-Kutta
  K1 = h*f1(y(i));
  K2 = h*(f1(y(i)) + (h*0.5*fx_y) + (0.5*K1*fx_x));
  K3 = h * h*(f1(y(i)) + (h*0.5*fx_y) + (0.5*K2*fx_x));
  K4 = h * h*(f1(y(i)) + (h*0.5*fx_y) + (0.5*K3*fx_x));
  Rx4(i+1) = x(i) + ( K1 + 2*K2 + 2*K3+ K4 )/6;

  K1 = h*f2(x(i));
  K2 = h*(f2(x(i)) + (h*0.5*fy_x) + (0.5*K1*fy_y));
  K3 = h * h*(f2(x(i)) + (h*0.5*fy_x) + (0.5*K2*fy_y));
  K4 = h * h*(f2(x(i)) + (h*0.5*fy_x) + (0.5*K3*fy_y));
  Ry4(i+1) = y(i) + ( K1 + 2*K2 + 2*K3 + K4 )/6;

  t(i+1) = t(i) + h; % increment the counter

end
plot(x,y,'r.-','DisplayName','2nd Order Taylor Series')
hold on
plot(Rx,Ry,'m.-','DisplayName','2nd Order Runge-Kutta')
plot(Rx4,Ry4,'b.-','DisplayName','4th Order Runge-Kutta')
hold off

lgd = legend;
lgd.NumColumns = 1;
```

(b) Run your code for h = 0.1, 0.25, 0.5 and 1. Include a hard copy of all four graphs along with your HW solutions.

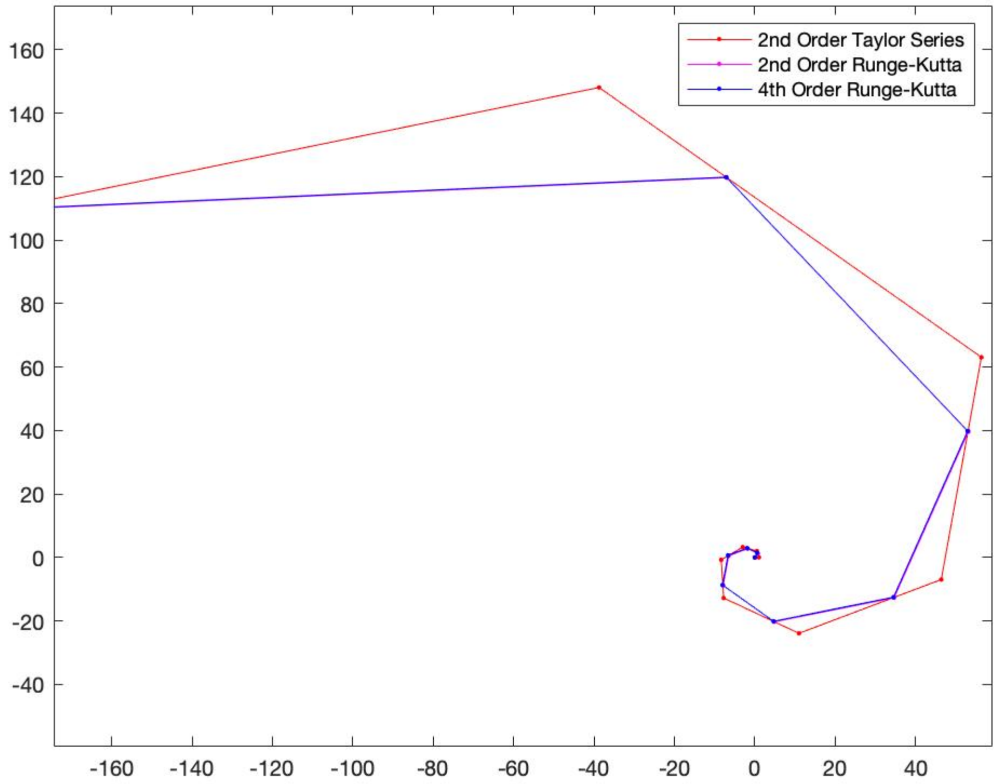
2nd Order Taylor Series
2nd Order Runge-Kutta
4th Order Runge-Kutta
160
140
120
100
80
60
40
20
0
-20
-40
-160
-140
-120
-100
-80
-60
-40
-20
0
20
40

Figure 1: h = 1

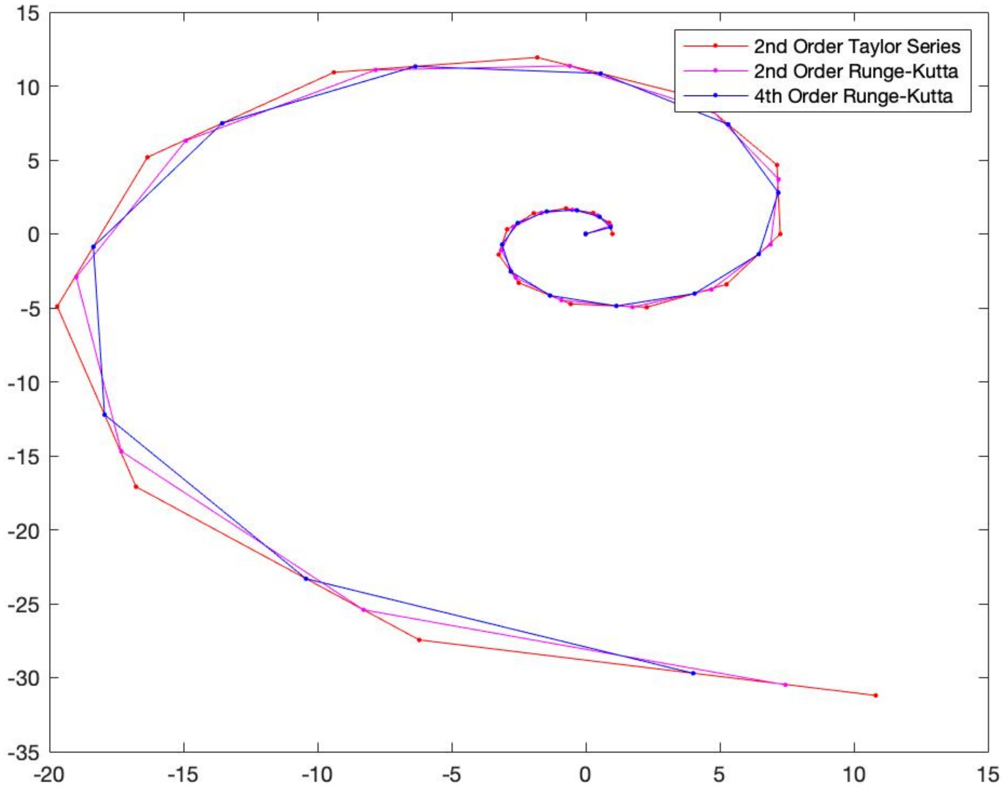
2nd Order Taylor Series
2nd Order Runge-Kutta
4th Order Runge-Kutta
15
10
5
0
-5
-10
-15
-20
-25
-30
-35
-20
-15
-10
-5
0
5
10
15

Figure 2: h = 0.5

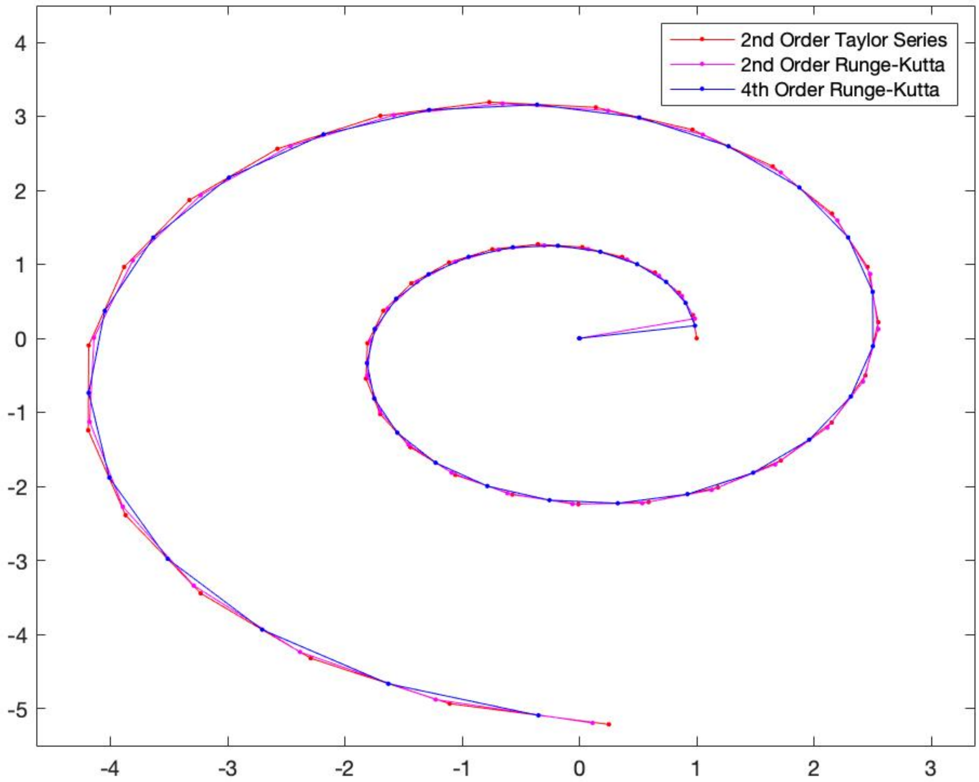

Figure 3: h = 0.25

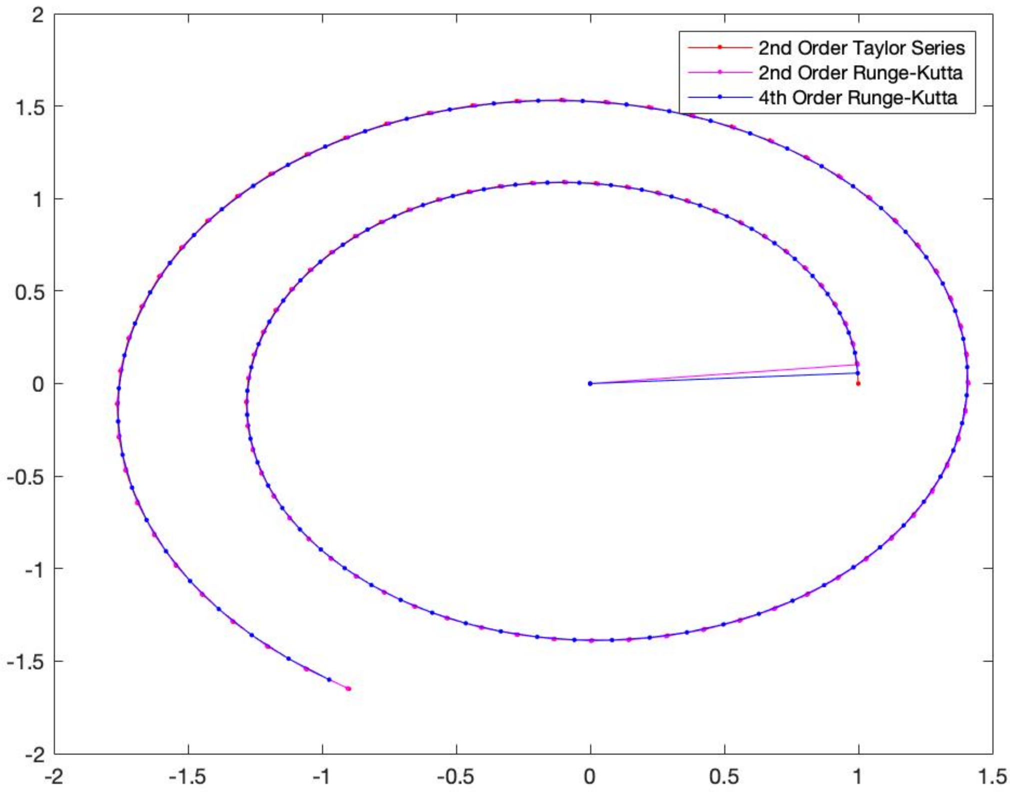

Figure 4: h = 0.1

Problem 4

Consider the following IVP

$$x' = x(1-x)$$
$$x(0) = 0.01$$

with the solution function $x(t) = 1 - 1/(1 + e^t/99)$
(a) Modify adaptive.m available on the course website to implement the Runge-Kutta-Fehlberg algorithm, provided below, to solve this IVP for $0 \leq t \leq 16$ The code should plot the analytical solution and the numerical solution on the same figure.

```
function rkf

f = @(t,x) x*(1-x);
time = 16;
tolerance = 0.01;
t(1) = 0;
x(1) = 0.01;

h = 1;
i = 1;

% The adaptive step size is the same no matter what method you use
```

```matlab
% Stiff equation stiff ode is an ode which is difficult to solve
    numerically
% Stiff solvers, finds the derivative in future time
while t(i) < time
  K1 = h*f(t(i),x(i)); % 6 k values
  K2 = h*f(t(i)+h/4,x(i)+K1/4);
  K3  = h*f(t(i)+3*h/8,x(i) + 3*K1/32 + 9*K2/32);
  K4 = h*f(t(i) + 12*h/13, x(i) + 1932*K1/2197 - 7200*K2/2197 + 7296*K3
     /2197);
  K5 = h*f(t(i) + h, x(i) + 439*K1/216 - 8*K2 + 3680*K3/513 - 845*K4
     /4104);
  K6 = h*f(t(i) + h/2,x(i) - 8*K1/27 + 2*K2 - 3544*K3/2565 + 1859*K4
     /4104 - 11*K5/40);

  % Two approximations for x(t + h) fourth order and 5th order
  x_RKF4 = x(i) + 25*K1/216 + 1408*K3/2565 + 2197*K4/4104 - K5/5;
  x_RKF5 = x(i) + 16*K1/135 + 6656*K3/12825 + 28561*K4/56430 - 9*K5/50 +
      2*K6/55;



  if abs(x_RKF5-x_RKF4) < tolerance
    x(i+1) = x_RKF5;
    t(i+1) = t(i) + h;
    h = 2*h;
    i = i+1;
  else
    h = h/2;
  end
end

for t_theory = 1:1:16
    x_theory(t_theory) = 1 - 1/(1 + exp(t_theory)/99);
end

t_theory = 1:1:16
plot(t_theory,x_theory,'--b','DisplayName','2nd Order Runge-Kutta')
hold on
plot(t,x,'--r','DisplayName','2nd Order Runge-Kutta')
hold off
grid

lgd = legend;
lgd.NumColumns = 1;
```

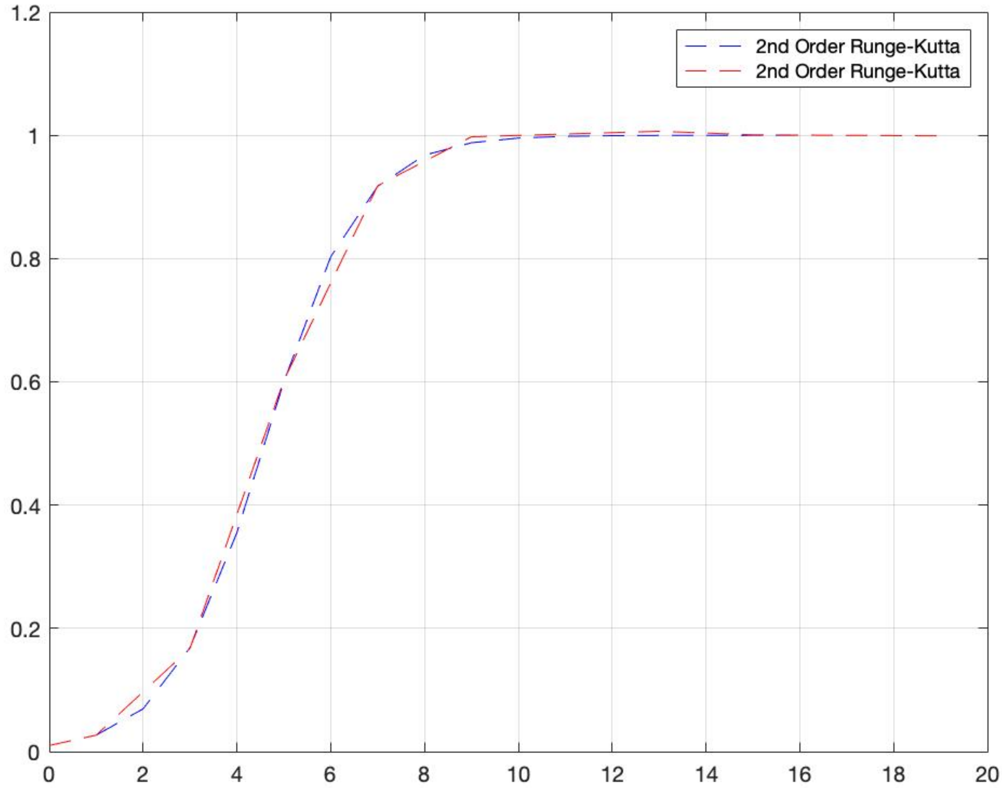

Figure 5: tolerance = 0.01

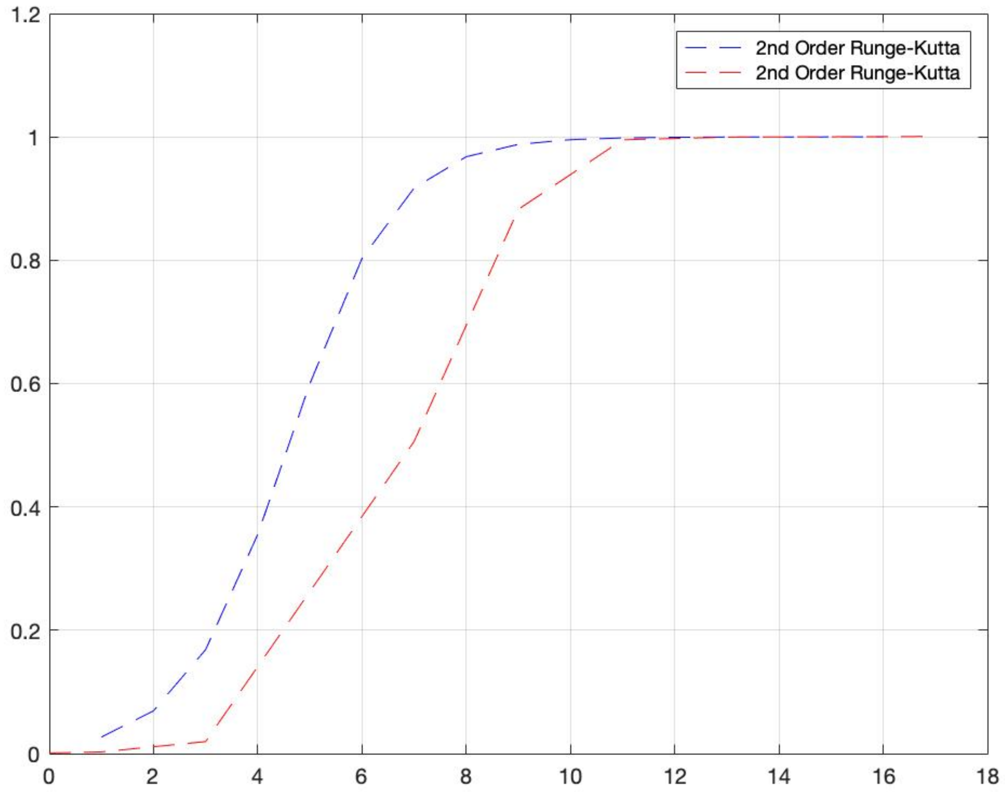

Figure 6: tolerance = 0.001