Math128aHw11

Trustin Nguyen

November 20, 2024

Exercise Set 5.9

Exercise 2: Use the Runge-Kutta method for systems to approximate the solutions of the following systems of first-order differential equations and compare the results to the actual solutions.

b .

$$
\begin{aligned}
u_1' &= \frac{1}{9}u_1 - \frac{2}{3}u_2 - \frac{1}{9}t^2 + \frac{2}{3} \\
u_2' &= u_2 + 3t - 4 \\
u_1(0) &= -3 \\
u_2(0) &= 5
\end{aligned}
$$

for $0 \leqslant t \leqslant 2$, $h = 0.2$.

Answer. Here is the comparison:

```
Command Window

approx =

   -3.0000    5.0000
   -3.6242    5.2856
   -4.3155    5.7673
   -5.1063    6.4884
   -6.0366    7.5021
   -7.1548    8.8730
   -8.5202   10.6803
  -10.2054   13.0205
  -12.2988   16.0118
  -14.9086   19.7981
  -18.1667   24.5556


actual =

   -3.0000    5.0000
   -3.6242    5.2856
   -4.3155    5.7673
   -5.1064    6.4885
   -6.0366    7.5022
   -7.1548    8.8731
   -8.5204   10.6805
  -10.2056   13.0208
  -12.2991   16.0121
  -14.9089   19.7986
  -18.1672   24.5562

fx
```

and here is my code:

```matlab
function approx = RKsystem(funcs, init, a, b, N)
m = size(funcs, 2);
h = (b - a) / N;
t = a;
approx = zeros(N + 1, m);
% rows of approx represent the different approximations at time t
% for each function

for j = 1:m
    approx(1, j) = init(j);
end

k = zeros(4, m);

for i = 1:N
    for j = 1:m
        currf = funcs{j};
        k(1, j) = h * currf(t, approx(i, :));
    end

    for j = 1:m
        currf = funcs{j};
        k(2, j) = h * currf(t + h / 2, approx(i, :) + k(1, :) / 2);
    end

    for j = 1:m
        currf = funcs{j};
        k(3, j) = h * currf(t + h / 2, approx(i, :) + k(2, :) / 2);
    end

    for j = 1:m
        currf = funcs{j};
        k(4, j) = h * currf(t + h, approx(i, :) + k(3, :));
    end

    for j = 1:m
        approx(i + 1, j) = approx(i, j) + (k(1, j) + 2 * k(2, j) + 2 * k(3, j) + k(4, j
    end

    t = a + i * h;
end

end

function y = q1f1(t, funcs)
% funcs is the function evaluations of the other functions
y = funcs(1) / 9 - 2 * funcs(2) / 3 - t^2 / 9 + 2 / 3;
end

function y = q1f2(t, funcs)
y = funcs(2) + 3 * t - 4;
end

function y = q1actf1(t)
y = -3 * exp(t) + t^2;
end
```

```
function y = q1actf2(t)
y = 4 * exp(t) - 3 * t + 1;
end

function actual = evalAct(funcs, a, b, N)
m = size(funcs, 2);
h = (b - a) / N;
actual = zeros(N + 1, m);

for j = 1:m
    t = a;
    currf = funcs{j};
    for i = 1:N + 1
        actual(i, j) = currf(t);
        t = a + i * h;
    end
end

funcs = {@q1f1, @q1f2};
init = [-3, 5];

approx = RKsystem(funcs, init, 0, 2, 10)
```

Exercise 4: Use the Runge-Kutta for Systems Algorithm to approximate the solutions of the following higher-order differential equations and compare the results to the actual solutions.

b $t^2y'' + ty' - 4y = -3t, 1 \leqslant t \leqslant 3, y(1) = 4, y'(1) = 3$, with $h = 0.2$; actual solution $y(t) = 2t^2 + t + t^{-2}$.

Answer. Let

$$u_1 = y, u_2 = y'$$

Then we get:

$$u_1' = u_2$$

and

$$t^2u_2' + tu_2 - 4u_1 = -3t$$

so

$$u_2' = \frac{1}{t^2}(-tu_2 + 4u_1 - 3t)$$

Now we apply RK with

$$\begin{aligned} u_1' &= u_2 \\ u_2' &= \frac{4}{t^2}u_1 - \frac{1}{t}u_2 - \frac{3}{t} \\ u_1(1) &= 4 \\ u_2(1) &= 3 \end{aligned}$$

Here is my comparison:

```
Command Window
approx =

    4.0000    3.0000
    4.7749    4.6425
    5.8308    5.8712
    7.1113    6.9119
    8.5893    7.8574
   10.2508    8.7504
   12.0875    9.6127
   14.0946   10.4559
   16.2691   11.2869
   18.6088   12.1096
   21.1126   12.9268


actual =

    4.0000
    4.7744
    5.8302
    7.1106
    8.5886
   10.2500
   12.0866
   14.0936
   16.2679
   18.6076
   21.1111

fx >>
```

and my code:

```
function approx = RKsystem(funcs, init, a, b, N)
m = size(funcs, 2);
h = (b - a) / N;
t = a;
approx = zeros(N + 1, m);
% rows of approx represent the different approximations at time t
% for each function

for j = 1:m
    approx(1, j) = init(j);
end

k = zeros(4, m);

for i = 1:N
    for j = 1:m
        currf = funcs{j};
        k(1, j) = h * currf(t, approx(i, :));
    end

    for j = 1:m
        currf = funcs{j};
        k(2, j) = h * currf(t + h / 2, approx(i, :) + k(1, :) / 2);
    end

    for j = 1:m
        currf = funcs{j};
        k(3, j) = h * currf(t + h / 2, approx(i, :) + k(2, :) / 2);
    end
```

```
    for j = 1:m
        currf = funcs{j};
        k(4, j) = h * currf(t + h, approx(i, :) + k(3, :));
    end

    for j = 1:m
        approx(i + 1, j) = approx(i, j) + (k(1, j) + 2 * k(2, j) + 2 * k(3, j) + k(4, j
    end

    t = a + i * h;
end

end

function y = q2f1(t, funcs)
y = funcs(2);
end

function y = q2f2(t, funcs)
y = 4 * funcs(1) / t^2 - funcs(2) / t - 3 / t;
end

function y = q2actf1(t)
y = 2 * t^2 + t + 1 / t^2;
end

function actual = evalAct(funcs, a, b, N)
m = size(funcs, 2);
h = (b - a) / N;
actual = zeros(N + 1, m);

for j = 1:m
    t = a;
    currf = funcs{j};
    for i = 1:N + 1
        actual(i, j) = currf(t);
        t = a + i * h;
    end
end

funcs = {@q2f1, @q2f2};
init = [4, 3];

approx = RKsystem(funcs, init, 1, 3, 10)

actFuncs = {@q2actf1};
actual = evalAct(actFuncs, 1, 3, 10)
```

Exercise 5: The study of mathematical models for predicting the population dynamics of competing species has its origin in independent works published in the early part of the 20th century by A. J. Lotka and V. Volterra (see, for example, |Lol |, rLo21, and |Vo|.).

Consider the problem of predicting the population of two species, one of which is a predator, whose population at time t is $x_2(t)$, feeding on the other, which is the prey, whose population is $x(t)$. We will assume that the prey always has an adequate food supply and that its birthrate at any time is proportional to the number of prey alive at that time; that is, birthrate (prey) is $k_1x_1(t)$. The death rate of the prey depends on both the number of prey and predators alive at that time. For simplicity, we assume death rate (prey) = $k_2x_1(t)x_2(t)$.

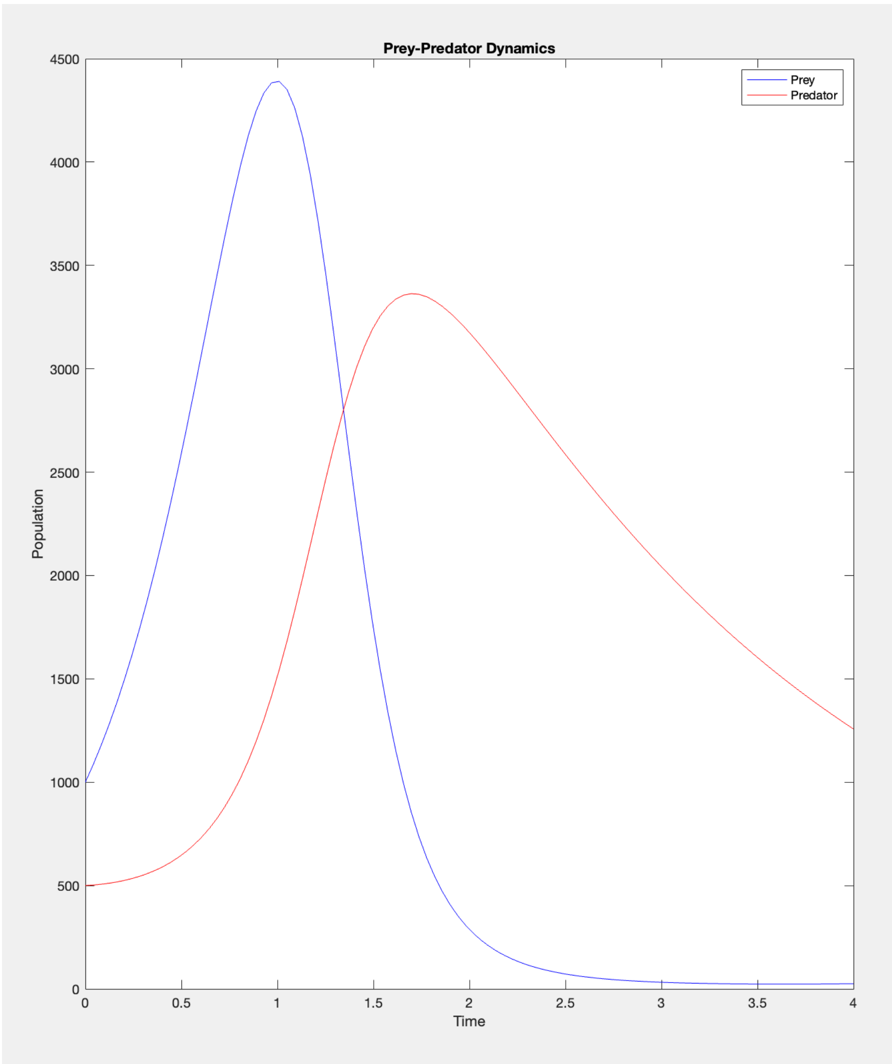
The birthrate of the predator, on the other hand, depends on its food supply, $x(t)$ as well as on the number of predators available for reproduction purposes. For this reason, we assume that the birthrate (predator) is $k_3x(t)x_2(t)$. The death rate of the predator will be taken as simply proportional to the number of predators alive at the time; that is, death rate (predator) = $k_4x_2(t)$.

Since $x(t)$ and $x_2(t)$ represent the change in the prey and predator populations, respectively, with respect to time, the problem is expressed by the system of nonlinear differential equations

$$x_1'(t) = k_1x_1(t) - k_2x_1(t)x_2(t) \qquad \text{and} \qquad x_2'(t) = k_3x_1(t)x_2(t) - k_4x_2(t)$$

Solve this system for $0 < t < 4$, assuming that the initial population of the prey is 1000 and of the predators is 500 and that the constants are $k_1 = 3, k_2 = 0.002, k_3 = 0.0006$, and $k_4 = 0.5$. Sketch a graph of the solutions to this problem, plotting both populations with time, and describe the physical phenomena represented. Is there a stable solution to this population model? If so, for what values x_1 and x_2 is the solution stable?

Answer. Here is the graph:



and here is my code:

```
function approx = RKsystem(funcs, init, a, b, N)
m = size(funcs, 2);
h = (b - a) / N;
t = a;
approx = zeros(N + 1, m);
% rows of approx represent the different approximations at time t
% for each function

for j = 1:m
    approx(1, j) = init(j);
end

k = zeros(4, m);

for i = 1:N
    for j = 1:m
        currf = funcs{j};
        k(1, j) = h * currf(t, approx(i, :));
    end

    for j = 1:m
        currf = funcs{j};
        k(2, j) = h * currf(t + h / 2, approx(i, :) + k(1, :) / 2);
    end

    for j = 1:m
        currf = funcs{j};
        k(3, j) = h * currf(t + h / 2, approx(i, :) + k(2, :) / 2);
    end

    for j = 1:m
        currf = funcs{j};
        k(4, j) = h * currf(t + h, approx(i, :) + k(3, :));
    end

    for j = 1:m
        approx(i + 1, j) = approx(i, j) + (k(1, j) + 2 * k(2, j) + 2 * k(3, j) + k(4, j
    end

    t = a + i * h;
end

end

function y = q3f1(t, funcs)
y = 3 * funcs(1) - 0.002 * funcs(1) * funcs(2);
end

function y = q3f2(t, funcs)
y = 0.0006 * funcs(1) * funcs(2) - 0.5 * funcs(2);
end

funcs = {@q3f1, @q3f2};
init = [1000, 500];

approx = RKsystem(funcs, init, 0, 4, 99);
```

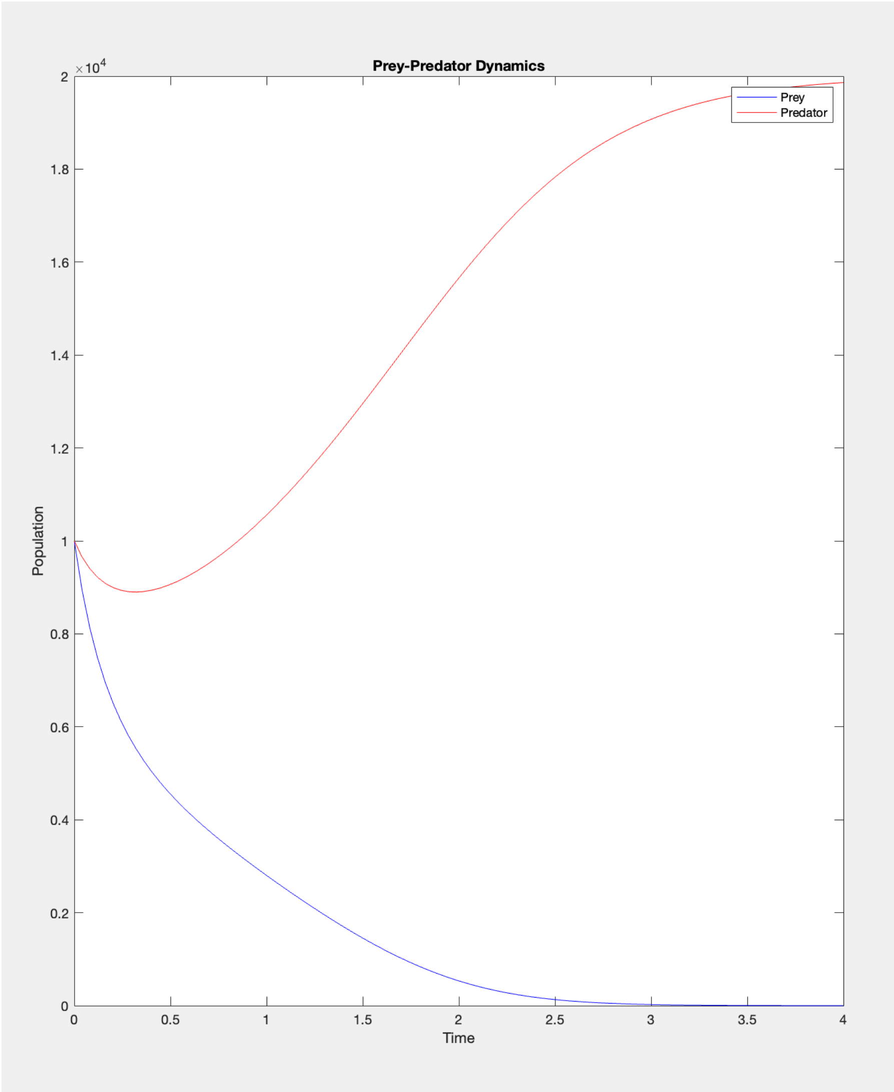
```
t = linspace(0, 4, 100);

plot(t, approx(:, 1), 'b-', t, approx(:, 2), 'r-'); % Optional: Add line styles/colors
legend('Prey', 'Predator');
xlabel('Time');
ylabel('Population');
title('Prey-Predator Dynamics');
```

Exercise 6: In Exercise 5, we considered the problem of predicting the population in a predator-prey model. Another problem of this type is concerned with two species competing for the same food supply. If the numbers of species alive at time t are denoted by $x_1(t)$ and $x_2(t)$, it is often assumed that, although the birthrate of each of the species is simply proportional to the number of species alive at that time, the death rate of each species depends on the population of both species. We will assume that the population of a particular pair of species is described by the equations

$$\frac{dx_1(t)}{dt} = x_1(t)[4 - 0.0003x_1(t) - 0.0004x_2(t)]$$
$$\frac{dx_2(t)}{dt} = x_2(t)[2 - 0.0002x_1(t) - 0.0001x_2(t)]$$

If it is known that the initial population of each species is $10,000$, find the solution to this system for $0 < t < 4$. Is there a stable solution to this population model? If so, for what values of x_1 and x_2 is the solution stable?

Answer. Here is my graph:



and the code:

```
function approx = RKsystem(funcs, init, a, b, N)
m = size(funcs, 2);
h = (b - a) / N;
t = a;
approx = zeros(N + 1, m);
% rows of approx represent the different approximations at time t
% for each function

for j = 1:m
    approx(1, j) = init(j);
end

k = zeros(4, m);

for i = 1:N
    for j = 1:m
        currf = funcs{j};
        k(1, j) = h * currf(t, approx(i, :));
    end
```

```
    for j = 1:m
        currf = funcs{j};
        k(2, j) = h * currf(t + h / 2, approx(i, :) + k(1, :) / 2);
    end

    for j = 1:m
        currf = funcs{j};
        k(3, j) = h * currf(t + h / 2, approx(i, :) + k(2, :) / 2);
    end

    for j = 1:m
        currf = funcs{j};
        k(4, j) = h * currf(t + h, approx(i, :) + k(3, :));
    end

    for j = 1:m
        approx(i + 1, j) = approx(i, j) + (k(1, j) + 2 * k(2, j) + 2 * k(3, j) + k(4, j
    end

    t = a + i * h;
end

end

function y = q4f1(t, funcs)
y = funcs(1) * (4 - 0.0003 * funcs(1) - 0.0004 * funcs(2));
end

function y = q4f2(t, funcs)
y = funcs(2) * (2 - 0.0002 * funcs(1) - 0.0001 * funcs(2));
end

funcs = {@q4f1, @q4f2};
init = [10000, 10000];

approx = RKsystem(funcs, init, 0, 4, 99);
t = linspace(0, 4, 100);

plot(t, approx(:, 1), 'b-', t, approx(:, 2), 'r-'); % Optional: Add line styles/colors
legend('Prey', 'Predator');
xlabel('Time');
ylabel('Population');
title('Prey-Predator Dynamics');
```

Exercise Set 5.11

Exercise 2: Solve the following stiff initial-value problems using Euler's method and compare the results with the actual solution.

b $y' = -10y + 10t + 1, 0 \leqslant t \leqslant 1, y(0) = e$, with $h = 0.1$; actual solution $y(t) = e^{-10t+1} + t$.

Answer. Here are the approximations vs the actual result:

```
Command Window
approx =

    2.7183
    0.1000
    0.2000
    0.3000
    0.4000
    0.5000
    0.6000
    0.7000
    0.8000
    0.9000
    1.0000


actual =

    2.7183
    1.1000
    0.5679
    0.4353
    0.4498
    0.5183
    0.6067
    0.7025
    0.8009
    0.9003
    1.0001

fx >>
```

And here is my code:

```
function approx = Euler(f, init, a, b, N)
h = (b - a) / N;
t = a;
w = init;
approx = zeros(N + 1, 1);
approx(1) = w;

for i = 1:N
    approx(i + 1) = approx(i) + h * f(t, approx(i));
    t = a + i * h;
end

end

function y = q5f(t, y)
y = -10 * y + 10 * t + 1;
end

function y = q5actf(t)
y = exp(-10 * t + 1) + t;
end
```

```
function actual = evalActStiff(f, a, b, N)
h = (b - a) / N;
t = a;
actual = zeros(N + 1, 1);
for i = 1:N + 1
    actual(i) = f(t);
    t = a + i * h;
end

end

approx = Euler(@q5f, exp(1), 0, 1, 10)
actual = evalActStiff(@q5actf, 0, 1, 10)
```

Exercise 4: Repeat Exercise 2 using the Runge-Kutta fourth-order method.

b $y' = -10y + 10t + 1, 0 \leqslant t \leqslant 1, y(0) = e$, with $h = 0.1$; actual solution $y(t) = e^{-10t+1} + t$.

Answer. Here are the approximations vs the actual result:

```
Command Window
approx =

    2.7183
    1.1194
    0.5823
    0.4433
    0.4538
    0.5202
    0.6076
    0.7028
    0.8011
    0.9004
    1.0001


actual =

    2.7183
    1.1000
    0.5679
    0.4353
    0.4498
    0.5183
    0.6067
    0.7025
    0.8009
    0.9003
    1.0001

fx >>
```

and here is my code:

```
function approx = RKfourth(f, init, a, b, N)
h = (b - a) / N;
t = a;
w = init;
approx = zeros(N + 1, 1);
approx(1) = w;

for i = 1:N

    k1 = h * f(t, approx(i));
```

```
    k2 = h * f(t + h / 2, approx(i) + k1 / 2);
    k3 = h * f(t + h / 2, approx(i) + k2 / 2);
    k4 = h * f(t + h, approx(i) + k3);

    approx(i + 1) = approx(i) + (k1 + 2 * k2 + 2 * k3 + k4) / 6;
    t = a + i * h;
end

end

function y = q5f(t, y)
y = -10 * y + 10 * t + 1;
end

function y = q5actf(t)
y = exp(-10 * t + 1) + t;
end

function actual = evalActStiff(f, a, b, N)
h = (b - a) / N;
t = a;
actual = zeros(N + 1, 1);
for i = 1:N + 1
    actual(i) = f(t);
    t = a + i * h;
end

end

approx = RKfourth(@q5f, exp(1), 0, 1, 10)
actual = evalActStiff(@q5actf, 0, 1, 10)
```

Exercise 8: Repeat Exercise 2 using the Trapezoidal Algorithm with TOL $= 10^{-5}$.

b $y' = -10y + 10t + 1, 0 \leqslant t \leqslant 1, y(0) = e$, with $h = 0.1$; actual solution $y(t) = e^{-10t+1} + t$.

Answer. Here are the approximations vs the actual result:

```
Command Window
approx =

    2.7183
    1.0061
    0.5020
    0.4007
    0.4336
    0.5112
    0.6037
    0.7012
    0.8004
    0.9001
    1.0000


actual =

    2.7183
    1.1000
    0.5679
    0.4353
    0.4498
    0.5183
    0.6067
    0.7025
    0.8009
    0.9003
    1.0001

fx >>
```

and here is my code:

```
function approx = TrapezoidNewton(f, dfy, init, a, b, N, tol, M)
h = (b - a) / N;
t = a;
w = init;
approx = zeros(N + 1, 1);
approx(1) = w;

for i = 1:N
    k1 = approx(i) + h * f(t, approx(i)) / 2;
    w0 = k1;
    j = 1;
    flag = 0;
    while flag == 0
        num = (w0 - h * f(t + h, w0) / 2 - k1);
        dem = (1 - h * dfy(t + h, w0) / 2);
        approx(i + 1) = w0 - num / dem;

        if abs(approx(i + 1) - w0) < tol
            flag = 1;
        else
            j = j + 1;
            w0 = approx(i + 1);
            if j > M
                break
            end
        end
    end
    t = a + i * h;
end
```

```
end

function y = q5f(t, y)
y = -10 * y + 10 * t + 1;
end

function y = q5fdy(t, y)
y = -10;
end

function y = q5actf(t)
y = exp(-10 * t + 1) + t;
end

function actual = evalActStiff(f, a, b, N)
h = (b - a) / N;
t = a;
actual = zeros(N + 1, 1);
for i = 1:N + 1
    actual(i) = f(t);
    t = a + i * h;
end

end

approx = TrapezoidNewton(@q5f, @q5fdy, exp(1), 0, 1, 10, 10e-5, 100)
actual = evalActStiff(@q5actf, 0, 1, 10)
```

Exercise 10: Show that the fourth-order Runge-Kutta method

$$\begin{aligned} k_1 &= hf(t_i, w_i) \\ k_2 &= hf(t_i + h/2, w_i + k_1/2) \\ k_3 &= hf(t_i + h/2, w_i + k_2/2) \\ k_4 &= hf(t_i + h, w_i + k_3) \\ w_{i+1} &= w_i + (k_1 + 2k_2 + 2k_3 + k_4)/6 \end{aligned}$$

when applied to the differential equation $y' = \lambda y$, can be written in the form

$$w_{i+1} = \left(1 + h\lambda + \frac{1}{2}(h\lambda)^2 + \frac{1}{6}(h\lambda)^3 + \frac{1}{2}(h\lambda)^4\right)$$

Answer. We evaluate:

$$\begin{aligned} k_1 &= hw_i\lambda \\ k_2 &= h(w_i + k_1/2)\lambda \\ k_3 &= h(w_i + k_2/2)\lambda \\ k_4 &= h(w_i + k_3)\lambda \end{aligned}$$

then

$$k_2 = hw_i(1 + h\lambda/2)\lambda = w_i(h\lambda + (h\lambda)^2/2)$$

and

$$k_3 = hw_i(1 + h(1 + h\lambda/2)\lambda/2)\lambda = hw_i(1 + h\lambda/2 + h^2\lambda^2/4)\lambda = w_i(h\lambda + (h\lambda)^2/2 + (h\lambda)^3/4)$$

and

$$k_4 = hw_i(1 + h\lambda/2 + (h\lambda)^2/4 + (h\lambda)^3/8)\lambda = w_i(h\lambda + (h\lambda)^2/2 + (h\lambda)^3/4 + (h\lambda)^4/8)$$

now

$$2k_2 = w_i(2h\lambda + (h\lambda)^2)$$

and

$$2k_3 = w_i(2h\lambda + (h\lambda)^2 + (h\lambda)^3/2)$$

now add then together:

$$\begin{aligned} k_1 &= w_i h\lambda \\ 2k_2 &= w_i(2h\lambda + (h\lambda)^2) \\ 2k_3 &= w_i(2h\lambda + (h\lambda)^2 + (h\lambda)^3/2) \\ k_4 &= w_i(h\lambda + (h\lambda)^2/2 + (h\lambda)^3/4 + (h\lambda)^4/8) \end{aligned}$$

we get

$$w_{i+1} = \left(1 + h\lambda + \frac{1}{2}(h\lambda)^2 + \frac{1}{6}(h\lambda)^3 + \frac{1}{24}(h\lambda)^4\right)$$