TMA4135 Matematikk 4D
Exercise 10

Odd André Owren

November 2019

1a. First, we begin by extracting the system of equations to make the iterations possible.

$$Ax = b \Rightarrow \begin{bmatrix} 3 & 1 & 1 \\ 1 & 3 & -1 \\ 3 & 1 & -5 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 5 \\ 3 \\ -1 \end{bmatrix} \Rightarrow \begin{matrix} x_0 = \frac{5-x_1-x_2}{3} \\ x_1 = \frac{3-x_0+x_2}{3} \\ x_2 = \frac{-1-3x_0-x_1}{-5} \end{matrix}$$

By using the fixed-point iteration, the above-mentioned equations uses the previous result for next iteration, i.e.:

$$x_0^{(n+1)} = \frac{5-x_1^{(n)}-x_2^{(n)}}{3}$$
$$x_1^{(n+1)} = \frac{3-x_0^{(n)}+x_2^{(n)}}{3}$$
$$x_2^{(n+1)} = \frac{-1-3x_0^{(n)}-x_1^{(n)}}{-5}$$

Doing two iterations with $x^{(0)} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$ then gives:

$$\begin{matrix} x_0^{(1)} = \frac{5-0-0}{3} = \frac{5}{3} & x_0^{(2)} = \frac{5-1-\frac{1}{5}}{3} = \frac{19}{15} \\ x_1^{(1)} = \frac{3-0+0}{3} = 1 & x_1^{(2)} = \frac{3-\frac{5}{3}+\frac{1}{5}}{3} = \frac{17}{45} \\ x_2^{(1)} = \frac{-1-3\cdot 0-0}{-5} = \frac{1}{5} & x_2^{(2)} = \frac{-1-3\cdot\frac{5}{3}-1}{-5} = \frac{7}{5} \end{matrix} \Rightarrow x^{(2)} = \begin{bmatrix} \frac{19}{15} \\ \\ \frac{17}{45} \\ \\ \frac{7}{5} \end{bmatrix}$$

The other iteration method is the Gauss-Seidel iterations. This differs from fixed-point by using the previous results before the next full iteration, i.e.:

$$x_0^{(n+1)} = \frac{5-x_1^{(n)}-x_2^{(n)}}{3}$$
$$x_1^{(n+1)} = \frac{3-x_0^{(n+1)}+x_2^{(n)}}{3}$$
$$x_2^{(n+1)} = \frac{-1-3x_0^{(n+1)}-x_1^{(n+1)}}{-5}$$

Doing two iterations with $x^{(0)} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$ then gives:

$$\begin{aligned} x_0^{(1)} &= \tfrac{5-0-0}{3} = \tfrac{5}{3} & x_0^{(2)} &= \tfrac{5-\frac{4}{9}-\frac{58}{45}}{3} = \tfrac{49}{45} \\ x_1^{(1)} &= \tfrac{3-\frac{5}{3}+0}{3} = \tfrac{4}{9} & x_1^{(2)} &= \tfrac{3-\frac{49}{45}+\frac{58}{45}}{3} = \tfrac{16}{15} \\ x_2^{(1)} &= \tfrac{-1-3\cdot\frac{5}{3}-\frac{4}{9}}{-5} = \tfrac{58}{45} & x_2^{(2)} &= \tfrac{-1-3\cdot\frac{49}{45}-\frac{16}{15}}{-5} = \tfrac{16}{15} \end{aligned} \quad \Rightarrow x^{(2)} = \begin{bmatrix} \frac{49}{45} \\ \frac{16}{15} \\ \frac{16}{15} \end{bmatrix}$$

1b. A matrix is strictly diagonally dominant if $|a_{ii}| > \Sigma_{j=1}^{i-1}|a_{ij}| + \Sigma_{j=i+1}^{n}|a_{ij}|$. For A in a), we have:

$$\begin{aligned} |a_{11}| &= 3 > 1+1 = |a_{12}| + |a_{13}| \\ |a_{22}| &= 3 > 1+1 = |a_{21}| + |a_{23}| \\ |a_{33}| &= 5 > 3+1 = |a_{31}| + |a_{32}| \end{aligned}$$

Thus, A is strictly diagonally dominant.

1c. I gave up on this one :)

2a. $y' - xy^2 = 0$, $y(0) = 1 \Rightarrow \frac{dy}{dx} = xy^2 \Rightarrow \frac{1}{y^2}dy = xdx \Rightarrow -\frac{1}{y} = \frac{x^2}{2} + c$

Applying the initial condition:

$-\frac{1}{1} = \frac{0^2}{2} + c \Rightarrow c = -1$

Solution is then:

$\underline{y = \frac{2}{2-x^2}}$

2b. Euler's method is defined by $y_{n+1} = y_n + hf(x_n, y_n)$ where $f(x,y) = y'$. In this case, $y' = xy^2 \Rightarrow f(x,y) = xy^2$. With the initial condition $y(0) = 1$, we have that $f(x_0, y_0) = 0 \cdot 1^2 = 0$. With 4 iterations we get:

$$\begin{aligned} &n = 0: && y_1 = 1 + 0.1 \cdot 0 = 1 && x_1 = 0.1 \\ &n = 1: && y_2 = 1 + 0.1 \cdot 0.1 \cdot 1^2 = 1.01 && x_2 = 0.2 \\ &n = 2: && y_3 = 1.01 + 0.1 \cdot 0.2 \cdot 1.01^2 = 1.03040 && x_3 = 0.3 \\ &n = 3: && y_4 = 1.0304 + 0.1 \cdot 0.3 \cdot 1.0304^2 = 1.06225 && x_4 = 0.4 \end{aligned}$$

The error is then $|y(0.4) - y_4| = |\frac{2}{2-0.4^2} - 1.06225| = |1.08696 - 1.06225| = 0.02471$.

2c. With Heun's method we need to calculate an intermediate value that is used when calculating the next point. The functions used in Heun's method is:

$\tilde{y}_{i+1} = y_i + hf(x_i, y_i)$

$y_{i+1} = y_i + \frac{h}{2}(f(x_i, y_i) + f(x_{i+1}, \tilde{y}_{i+1}))$

where $\tilde{y}$ is the intermediate value and $x_{i+1} = x_i + h$. Applying this to the initial value problem then gives us:

$$\begin{array}{lll} \tilde{y}_1 = 1 + 0 = 1 & y_1 = 1 + \frac{0.2}{2}(0 + 0.2) = 1.02 & x_1 = 0.2 \\ \tilde{y}_2 = 1.02 + 0.2 \cdot 0.2 \cdot 1.02^2 = 1.06162 & y_2 = 1.02 + \frac{0.2}{2}(1.02^2 \cdot 0.2 + 1.06162^2 \cdot 0.4) = 1.08589 & x_1 = 0.4 \end{array}$$

The error is then $|y(0.4) - y_2| = |\frac{2}{2-0.4^2} - 1.08589| = |1.08696 - 1.08589| = 0.00107$.

2d. The 4th order Runge-Kutta method is done by calculating 4 intermediate values k_1, k_2, k_3, k_4, and then using this to find an approximation y_{n+1}. In this case, we have:

$$\begin{array}{l} k_1 = f(x_n, y_n) = 0 \\ k_2 = f(x_n + \frac{h}{2}, y_n + \frac{h}{2}k_1) = 0.2 \cdot 1^2 = 0.2 \\ k_3 = f(x_n + \frac{h}{2}, y_n + \frac{h}{2}k_2) = 0.2 \cdot (1 + 0.2 \cdot 0.2)^2 = 0.21632 \\ k_4 = f(x_n + h, y_n + hk_3) = 0.4 \cdot (1 + 0.4 \cdot 0.21632)^2 = 0.47222 \\ y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \Rightarrow y_1 = 1 + \frac{0.4}{6}(0 + 2 \cdot 0.2 + 2 \cdot 0.21632 + 0.47222) = 1.08699 \end{array}$$

The error is then $|y(0.4) - y_1| = |\frac{2}{2-0.4^2} - 1.08699| = |1.08696 - 1.08699| = 3 \cdot 10^{-5}$.

Out of the 3 methods used it is clear that the 4th order Runge-Kutta method is the one that minimizes the error, and is the better of the methods.

3a. By defining
$y_1(x) = u_1(x)$ $y_2(x) = u_1'(x)$ $y_3(x) = u_2(x)$ $y_4(x) = u_2'(x)$
then we have that

$$\begin{array}{ll} y_2(x) = y_1'(x) & y_2'(x) = -\frac{1}{(y_1(x)-y_3(x))^2} \\ y_4(x) = y_3'(x) & y_4'(x) = \frac{1}{(y_1(x)-y_3(x))^2} \\ y_1(0) = 0 \quad y_2(0) = 1 & y_3(0) = 1 \quad y_4(0) = 0 \end{array}$$

which is a system of first-order differential equations.

3b. We start by defining the vectors for y' and y_0:

$$y'(x) = \begin{bmatrix} y_2(x) \\ -\frac{1}{(y_1(x)-y_3(x))^2} \\ y_4(x) \\ \frac{1}{(y_1(x)-y_3(x))^2} \end{bmatrix} \quad y_0(x) = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Applying Heun's method then gives us:

$$k = f(x_0, y_0) = y'(y_0) = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 1 \end{bmatrix}$$

$$\tilde{y}_1 = y_0 + hk = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} + 0.1 \cdot \begin{bmatrix} 1 \\ -1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0.1 \\ 0.9 \\ 1 \\ 0.1 \end{bmatrix}$$

$$y_1 = y_0 + \tfrac{h}{2}(k + f(x_1, \tilde{y}_1)) = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} + \tfrac{0.1}{2}\left(\begin{bmatrix} 1 \\ -1 \\ 0 \\ 1 \end{bmatrix} + \begin{bmatrix} 0.9 \\ -1.23457 \\ 0.1 \\ 1.23457 \end{bmatrix} \right) = \begin{bmatrix} 0.095 \\ 0.88828 \\ 1.005 \\ 0.11173 \end{bmatrix}$$

3c. See figure 1. It is clear from the first 2 rows that the solution for 1 step in the previous task is correct.

```
In [4]:
def heun(f, x, y, h):
    # One step of Heun's method
    k1 = f(x, y)
    k2 = f(x+h, y+h*k1)
    y_next = y + 0.5*h*(k1+k2)
    x_next = x + h
    return x_next, y_next
```

executed in 8ms, finished 14:36:47 2019-10-30

```
In [7]:
def f(x,y):
    return array(
        [
            y[1],
            -1 / (y[0]-y[2]) ** 2,
            y[3],
            1 / (y[0]-y[2]) ** 2,
        ]
    )

x0, xend = 0, 1
y0 = array([0, 1, 1, 0])

# Solve the equation
x_lv, y_lv = ode_solver(f, x0, xend, y0, h=0.1, method=heun)


print(y_lv)
```

executed in 20ms, finished 14:39:35 2019-10-30

```
[[ 0.          1.          1.          0.        ]
 [ 0.095       0.8882716   1.005       0.1117284 ]
 [ 0.17778924  0.75572145  1.02221076  0.24427855]
 [ 0.24634923  0.60614514  1.05365077  0.39385486]
 [ 0.29929193  0.44850899  1.10070807  0.55149101]
 [ 0.33635791  0.29477352  1.16364209  0.70522648]
 [ 0.35852959  0.15540339  1.24147041  0.84459661]
 [ 0.36765625  0.03608132  1.33234375  0.96391868]
 [ 0.36589163 -0.06235909  1.43410837  1.06235909]
 [ 0.35527394 -0.14204431  1.54472606  1.14204431]
 [ 0.33753543 -0.20617437  1.66246457  1.20617437]]
```

Figure 1: Solving the ODE with Heun in the interval [0,1]

4a. See figure 2. One can see that when h is halved, the error is reduced by $\frac{1}{16}$, thus we have an order of 4.

```
In [9]:  def rk4(f, x, y, h):
             k1 = f(x, y)
             k2 = f(x + h / 2, y + h / 2 * k1)
             k3 = f(x + h / 2, y + h / 2 * k2)
             k4 = f(x + h, y + h * k3)

             y_next = y + h*(k1 + 2*k2 + 2*k3 + k4)/6
             x_next = x + h
             return x_next, y_next
executed in 8ms, finished 15:27:22 2019-10-30
```

```
In [10]: # Test the order of a method, given a test equation with exact solution
         def f1(x, y):
             return -2*x*y

         def y_eksakt(x):
             return exp(-x**2)

         h = 0.1
         x0, xend= 0, 1
         y0 = 1

         print('h           error          order \n---------------------------------')

         for n in range(10):
             x_num, y_num = ode_solver(f1, x0, xend, y0, h, method = rk4)
             error = norm(y_eksakt(xend)-y_num[-1])       # Error in the end point
             if n is 0:
                 order = NaN          # Nothing to compare
             else:
                 order = log2(error_old/error)     # Calculate  the order p
             print(format('{:.3e}   {:.3e}   {:7.2f}'.format( h, error, order)))
             h = 0.5*h                 # Reduce the stepsize
             error_old = error
executed in 518ms, finished 15:27:51 2019-10-30

h           error          order
---------------------------------
1.000e-01   1.625e-06       nan
5.000e-02   1.025e-07      3.99
2.500e-02   6.407e-09      4.00
1.250e-02   3.999e-10      4.00
6.250e-03   2.497e-11      4.00
3.125e-03   1.559e-12      4.00
1.563e-03   9.520e-14      4.03
7.813e-04   8.549e-15      3.48
3.906e-04   1.110e-14     -0.38
1.953e-04   1.033e-14      0.10
```

Figure 2: RK4 used to solve tests as described in oving10_python_files.ipynb

4b. See figure 3 for solution of Lotka-Volterra-equation with RK4.

```
In [14]: f = lotka_volterra
         x0, xend = 0, 10
         y0 = array([2, 0.5])


         x_lv, y_lv = ode_solver(f, x0, xend, y0, 0.5, rk4)
         plot(x_lv,y_lv, 'o');
         xlabel('x')
         legend(['y1','y2'],loc=1);


         x_lv, y_lv = ode_solver(f, x0, xend, y0, 0.1, rk4)
         plot(x_lv,y_lv);
         xlabel('x')
         legend(['y1','y2'],loc=1);
executed in 871ms, finished 15:32:06 2019-10-30
```

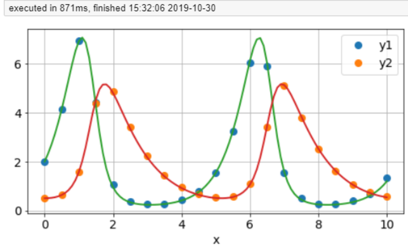

Figure 3: Solution of Lotka-Volterra with RK4, h=0.5 and h=0.1

For use of Heun's method, one can see that with h=0.5 the solver crashes with an overflow, and it is not able to solve the Lotka-Volterra-equation. See figure 4. For Heun with h=0.1 we get the solution shown in figure 5.

In [16]:
```
f = lotka_volterra
x0, xend = 0, 10
y0 = array([2, 0.5])
h = 0.5

x_lv, y_lv = ode_solver(f, x0, xend, y0, h, heun)
plot(x_lv, y_lv);
xlabel('x')
legend(['y1','y2'], loc=1);
```
executed in 766ms, finished 15:33:26 2019-10-30

```
/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:5: RuntimeWarning: overflow encountered in double_scalars
  """
/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:6: RuntimeWarning: overflow encountered in double_scalars

/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:6: RuntimeWarning: invalid value encountered in double_scalars
```

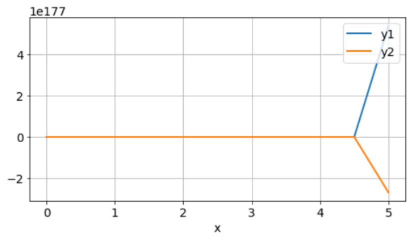

Figure 4: Heun with h=0.5

In [18]:
```
h = 0.1

x_lv, y_lv = ode_solver(f, x0, xend, y0, h, heun)
plot(x_lv, y_lv);
xlabel('x')
legend(['y1','y2'], loc=1);
```
executed in 758ms, finished 15:33:45 2019-10-30

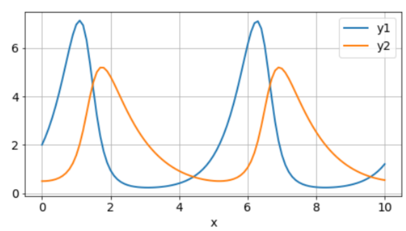

Figure 5: Heun with h=0.1