2️⃣

Math 449: Assignment 2

Question 1:

Consider the polynomial: $\pi_n(t) = \Pi_{i=0}^{n}(t-i), 0 \le t \le n$

A)

Show that: $|\pi_n(t)| \le \frac{n!}{4}, 0 \le t \le n$

1. $|\pi_n(t)| = |(t-0)(t-1)\ldots(t-n+1)(t-n)|$
2. Following the hint, lets show that for $0 \le i_0 \le n-1$ and $i_0 \le t \le i_0+1$ we have $(t-i_0)(i_0+1-t) \le \frac{1}{4}$
   a. We have that $(i_0+1-t) = 1-(t-i_0)$.
   b. Lets rename some variables:
      i. $a = (t-i_0)$
      ii. $1-a = (i_0+1-t)$
   c. And we know that $0 \le a \le 1$
   d. So we have $a(1-a) = a-a^2$ on the interval $[0,1]$.
   e. If we take the derivative we get $1-2a$ on $[0,1]$
   f. This is 0 when $a = \frac{1}{2}$, and $f(\frac{1}{2}) = \frac{1}{4}$.
   g. Checking the boundaries, $f(0) = 0-0^2-0$, $f(1) = 1-1^2 = 0$
   h. So we have that $\frac{1}{4}$ is the maximum and that $(t-i_0)(i_0+1-t) \le \frac{1}{4}$ given our conditions, $0 \le i_0 \le n-1$ and $i_0 \le t \le i_0+1$.
3. Given that $0 \le t \le n$, we know that there is one i_j such that $i_0 \le t \le i_0+1$, as the i's are the integers between $[0,n]$.
   a. This i_0 is simply $\lfloor t \rfloor$ and $i_0+1 = \lceil t \rceil$.
4. From (2), (3), we know that somewhere in our product we have $(t-\lfloor t \rfloor)(t-\lceil t \rceil) \le |(t-\lfloor t \rfloor)(t-\lceil t \rceil)| = |(t-\lfloor t \rfloor)(\lceil t \rceil - t)| \le \frac{1}{4}$
5. And thus we have:
   a. $|\pi_n(t)| = |\Pi_{i=0}^{n}(t-i)| = |(t-\lfloor t \rfloor)(t-\lceil t \rceil)\Pi_{i=0,i\neq\lceil t \rceil,\lfloor t \rfloor}^{n}(t-i) \le |\frac{1}{4}\Pi_{i=0,i\neq\lceil t \rceil,\lfloor t \rfloor}^{n}(t-i)|$
6. And now we just need to show that $|\Pi_{i=0,i\neq\lceil t \rceil,\lfloor t \rfloor}^{n}(t-i)| \le n!$
   a. $|\Pi_{i=0,i\neq\lceil t \rceil,\lfloor t \rfloor}^{n}(t-i)| = \Pi_{i=0,i\neq\lceil t \rceil,\lfloor t \rfloor}^{n}|(t-i)| \le \Pi_{i=0,i\neq\lceil t \rceil,\lfloor t \rfloor}^{n}\lceil|(t-i)|\rceil = \lceil|(t-0)|\rceil\lceil|(t-1)|\rceil\ldots\lceil|(t-n)|\rceil \le n!$
7. Therefore we have that $|\pi_n(t)| \le \frac{n!}{4}, 0 \le t \le n$

B)

Deduce that for equally spaced interpolation points, $x_i = a+ih, i = 0,\ldots,n$ **the interpolation error for any given smooth function** f **satisfies:**

$$||f-p_n|| \leq \frac{h^{n+1}}{n+1}M_{n+1}$$

Where $M_{n+1} = max_{a\leq x\leq b}|f^{(n+1)}(x)|$

1. = Let $R(x) \equiv f(x) - p_n(x)$
   a. We can think of $R(x)$ as the interpolation error at some point x
2. Consider a polynomial q_{n+1} that interpolates $R(x)$ at the points $x_i = a + ih$, as well as $x_{n+1} = x$ (here x is wherever we're evaluating the error with $R(x)$)
   a. Now x is fixed, so we will use z as our variable term
3. We have that $q_{n+1}(z) = R(x)L_{n+1}(z) = (f(x) - p_n(x))\Pi_{i=0}^{n}\frac{z-x_i}{x-x_i}$
   a. $L_{n+1}(z)$ is the Lagrange polynomial corresponding to x_{n+1} here
4. Now consider the interpolation error of q_{n+1} on $R(x)$, $R_q(z) \equiv R(z) - q_{n+1}(z)$.
   a. R_q must, by the nature of it being the interpolation error, have $n+2$ roots in the interval $[a,b]$, specifically these are $z = x_0, x_1, ..., x_n, x$.
   b. Now we can use Rolle's Theorem to see that $R_q^{(n+1)}$ must have at least one root in $[a,b]$.
   c. $\exists\zeta \in [a,b]$ such that $R_q^{(n+1)}(\zeta) \equiv R^{(n+1)}(\zeta) - \frac{d^{(n+1)}}{dz^{(n+1)}}q_{n+1}(\zeta) = 0$
   d. And we know that $p_n^{(n+1)}(z) = 0$ because it's the $n+1$th derivative of a degree n polynomial, so $R^{(n+1)}(\zeta) = f^{(n+1)}(\zeta)$
   e. We also have, because q_{n+1} has degree $n+1$, that $q_{n+1}^{(n+1)}(\zeta) = (f(x) - p_n(x))\frac{(n+1)!}{\Pi_{i=0}^{n}(x-x_i)}$.
5. We can combine 4e and 4c to get:
   a. $f^{(n+1)}(\zeta) = (f(x) - p_n(x))\frac{(n+1)!}{\Pi_{i=0}^{n}(x-x_i)}$
   b. Multiplying both sides by $\frac{\Pi_{i=0}^{n}(x-x_i)}{(n+1)!}$ we get:
   c. $\frac{\Pi_{i=0}^{n}(x-x_i)}{(n+1)!}f^{(n+1)}(\zeta) = f(x) - p_n(x)$
6. Now we can use a change of variables in 5c, $x \to t : x = th + a$, and we get:
   a. $f(x) - p_n(x) = f^{(n+1)}(\zeta)\frac{1}{(n+1)!}\Pi_{i=0}^{n}(x - x_i) = f^{(n+1)}(\zeta)\frac{1}{(n+1)!}\Pi_{i=0}^{n}(th + a - ih - a) = \frac{f^{(n+1)}(\zeta)}{(n+1)!}\Pi_{i=0}^{n}h(t - i) = \frac{f^{(n+1)}(\zeta)}{(n+1)!}h^{n+1}\Pi_{i=0}^{n}(t-i)$
      i. Remember that $\pi_n(t) = \Pi_{i=0}^{n}(t-i) \leq \frac{n!}{4}$ from Part A)
   b. We have: $\frac{f^{(n+1)}(\zeta)}{(n+1)!}h^{n+1}\Pi_{i=0}^{n}(t-i) \leq \frac{f^{(n+1)}(\zeta)}{(n+1)!}h^{n+1}\frac{n!}{4} \leq \frac{f^{(n+1)}(\zeta)}{4(n+1)}h^{n+1}$
      i. We also have, from the definition of M_{n+1}, that $f^{(n+1)}(\zeta) \leq M_{n+1}$
   c. So we have $\frac{f^{(n+1)}(\zeta)}{4(n+1)}h^{n+1} \leq \frac{h^{n+1}}{4(n+1)}M_{n+1} \leq \frac{h^{n+1}}{(n+1)}M_{n+1}$
7. And using 5c we can substitute: $||f - p_n|| \leq \frac{h^{n+1}}{(n+1)}M_{n+1}$

Question 2

Derive the 3rd order Adams-Moulton method, and deduce that it's truncation error τ_h is $O(h^3)$.

1. Consider the differential equation: $\dot{x} = f(x,t)$ with $t \in [0,T]$. Now let $h = \frac{T}{n}, x_i = ih$ be a discretization of $[0,T]$ on n points. Let $x_0, x_1, ...x_i, i \leq k$ be a sequence of (approximately) known points.

    a. Then we have: $x_{i+1} = x_i + \int_{t_i}^{t_i+1} \dot{x}(t)dt = x_i + \int_{t_i}^{t_i+1} f(x(t),t)dt$

2. Let P_{k+1} be the interpolation polynomial of $\dot{x}$ using the data points: $(t_{i-k}, \dot{x}_{i-k}), (t_{i-k+1}, \dot{x}_{i-k+1}), ...(t_i, \dot{x}_i), (t_{i+1}, \dot{x}_{i+1})$ where $\dot{x}_j \approx f_j := f(x_j, t_j)$
3. We can use P_{k+1} and the Newton-Cotes quadrature formula to get:
    a. $x_{i+1} = x_i + \int_{t_i}^{t_i+1} P_{k+1}(t)dt$
4. The 3rd order Adams-Moulton method corresponds to setting $k = 1$.
5. Thus we have $x_{i+1} - x_i = \int_{t_i}^{t_i+1} P_2(t)dt$
    a. With:
    b. $P_2(t) = \Sigma_{j=i-1}^{i+1} L_j(t)f_j = \frac{t-t_i}{t_{i-1}-t_i}\frac{t-t_{i+1}}{t_{i-1}-t_{i+1}}f_{i-1} + \frac{t-t_{i-1}}{t_i-t_{i-1}}\frac{t-t_{i+1}}{t_i-t_{i+1}}f_i + \frac{t-t_{i-1}}{t_{i+1}-t_{i-1}}\frac{t-t_i}{t_{i+1}-t_i}f_{i+1}$
6. So we have:
7. $x_{i+1} - x_i = f_{i-1}\int_{t_i}^{t_{i+1}} \frac{t-t_i}{t_{i-1}-t_i}\frac{t-t_{i+1}}{t_{i-1}-t_{i+1}}dt + f_i\int_{t_i}^{t_{i+1}} \frac{t-t_{i-1}}{t_i-t_{i-1}}\frac{t-t_{i+1}}{t_i-t_{i+1}}dt + f_{i+1}\int_{t_i}^{t_{i+1}} \frac{t-t_{i-1}}{t_{i+1}-t_{i-1}}\frac{t-t_i}{t_{i+1}-t_i}dt$
    a. $f_{i-1}\int_{t_i}^{t_{i+1}} \frac{t-t_i}{t_{i-1}-t_i}\frac{t-t_{i+1}}{t_{i-1}-t_{i+1}}dt$
        i. $= f_{i-1}\int^{t_{i+1}} t_i \frac{t-ti}{-h}\frac{t-t_{i+1}}{-2h}dt = \frac{f_{i-1}}{2h^2}\int^{t_{i+1}} t_i t^2 - t(ti+1) - tt_i + t_i(t_{i+1})dt = \frac{f_{i-1}}{2h^2}\int^{t_{i+1}} t_i t^2 - t(ti + 1 + t_i) + t_i(t_{i+1})dt$
        ii. $= \frac{f_{i-1}}{2h^2}[\frac{t^3}{3} - \frac{t^2}{2}(t_i + t_{i+1}) + t(t_i)(t_{i+1}h)]|_{t_i}^{t_{i+1}} = \frac{f_{i-1}}{2h^2}[(\frac{t_{i+1}^3}{3} - \frac{t_{i+1}^2}{2}(t_i + t_{i+1}) + t_{i+1}^2 t_i) - (\frac{t_i^3}{3} - \frac{t_i^2}{2}(t_i + t_{i+1}) + t_i^2 t_{i+1})] = \frac{f_{i-1}}{2h^2}\frac{-t_{i+1}^3+3t_{i+1}^2t_i}{6} - \frac{-t_i^3+3t_i^2t_{i+1}}{6} = \frac{f_{i-1}}{2h^2}\frac{-t_{i+1}^3+t_i^3+3t_{i+1}^2t_i-3t_i^2t_{i+1}}{6}$
        iii. $= \frac{f_{i-1}}{2h^2}\frac{-t_{i+1}^3+t_i^3+3t_it_{i+1}(t_{i+1}-t_i)}{6} = \frac{f_{i-1}}{2h^2}\frac{t_i^3-t_{i+1}^3+t_it_{i+1}h}{6} = \frac{f_{i-1}}{2h^2}\frac{t_i^3-(t_i+h)^3+t_i(t_i+h)h}{6}$
        iv. $= \frac{f_{i-1}}{2h^2}\frac{-h^3-3h^2t_i-3ht_i^2+t_i^2h+t_ih^2}{6} = \frac{f_{i-1}}{2h^2}\frac{-h^3}{6} = -\frac{hf_{i-1}}{12}$
    b. $f_i\int_{t_i}^{t_{i+1}} \frac{t-t_{i-1}}{t_i-t_{i-1}}\frac{t-t_{i+1}}{t_i-t_{i+1}}dt$
        i. = $\frac{f_i}{-h^2}\int_{t_i}^{t_{i+1}}(t-t_{i-1})(t-t_{i+1})dt = \frac{f_i}{-h^2}\int_{t_i}^{t_{i+1}} t^2 - t(t_i + h + t_i - h) + (t_i - h)(t_i + h)dt = \frac{f_i}{-h^2}\int_{t_i}^{t_i+h} t^2 - 2tt_i + t_i^2 - h^2dt$
        ii. $= \frac{f_i}{-h^2}\int_{t_i}^{t_i+h}(t-t_i)^2 - h^2dt = \frac{f_i}{-h^2}[\frac{(t-t_i)^3}{3} - th^2]|_{t_i}^{t_i+h}$
        iii. $= \frac{2h}{3}f_i$
    c. $f_{i+1}\int_{t_i}^{t_{i+1}} \frac{t-t_i}{t_{i+1}-t_i}\frac{t-t_{i-1}}{t_{i+1}-t_{i-1}}dt$
        i. $= \frac{f_{i+1}}{2h^2}\int_{t_i}^{t_i+h}(t-t_i)(t-t_i+h)dt$
        ii. $= \frac{f_{i+1}}{2h^2}\int_{t_i}^{t_i+h} t^2 - tt_i + th - tt_i + t_i^2 - t_ihdt$
        iii. $= \frac{f_{i+1}}{2h^2}\int_{t_i}^{t_i+h}(t-t_i)^2 + th - t_ihdt$
        iv. $= \frac{f_{i+1}}{2h^2}[\frac{(t-t_i)^3}{3} + \frac{t^2h}{2} - tt_ih]|_{t_i}^{t_i+h}$
        v. $= \frac{f_{i+1}}{2h^2}[(\frac{h^3}{3} + \frac{(t_i+h)^2h}{2} - (t_i+h)t_ih) - (\frac{t_i^2h}{2} - t_i^2h)]$
        vi. $= \frac{f_{i+1}}{2h^2}[\frac{h^3}{3} + \frac{t_i^2h+2t_ih^2+h^3}{2} - t_i^2h - t_ih^2 - \frac{t_i^2h}{2} + t_i^2h]$
        vii. $= \frac{f_{i+1}}{2h^2}[\frac{5h^3}{6}] = \frac{5hf_{i+1}}{12}$
    d. We have via a,b,c that:
        i. $x_{i+1} - x_i = h[\frac{-1}{12}f_{i-1} + \frac{2}{3}f_i + \frac{5}{12}f_{i+1}]$
        ii. This is the Adams-Moulton 3rd order formula!
8. The truncation error can be obtained by integrating the interpolation error:
    a. $\tau_h = \frac{1}{h}[x(t_{i+1}) - \hat{x}(t_{i+1})] = \frac{1}{h}\int_{t_i}^{t_{i+1}} f(x(t),t) - P_{k+1}(t)dt$
9. In the case of the 3rd order Adams-Moulton:

$$\tau_h = \frac{1}{h[(3)!]}\frac{d^3[f(x(\zeta),\zeta)]}{dt^3}\int_{t_i}^{t_{i+1}}(t-t_{i-1})(t-t_i)(t-t_{i+1})dt$$

$$= \frac{1}{6h}\frac{d^3[f(x(\zeta),\zeta)]}{dt^3}\int_{t_i}^{t_{i+1}}(t-(t_i-h))(t-t_i)(t-(t_i+h))dt$$

$$= \frac{1}{6h}\frac{d^3[f(x(\zeta),\zeta)]}{dt^3}\int_{t_i}^{t_{i+1}}h^2t_i-h^2t-t_i^3+3t_i^2t-3t_it^2+t^3dt$$

$$= \frac{1}{6h}\frac{d^3[f(x(\zeta),\zeta)]}{dt^3}[h^2t_it-\frac{h^2t^2}{2}-t_i^3t+\frac{3t_i^2t^2}{2}-t_it^3+\frac{t^4}{4}]|_{t_i}^{t_i+h}$$

$$= \frac{1}{6h}\frac{d^3[f(x(\zeta),\zeta)]}{dt^3}\frac{-h^4}{4} = \frac{-h^3}{24}\frac{d^3[f(x(\zeta),\zeta)]}{dt^3} = O(h^3)$$

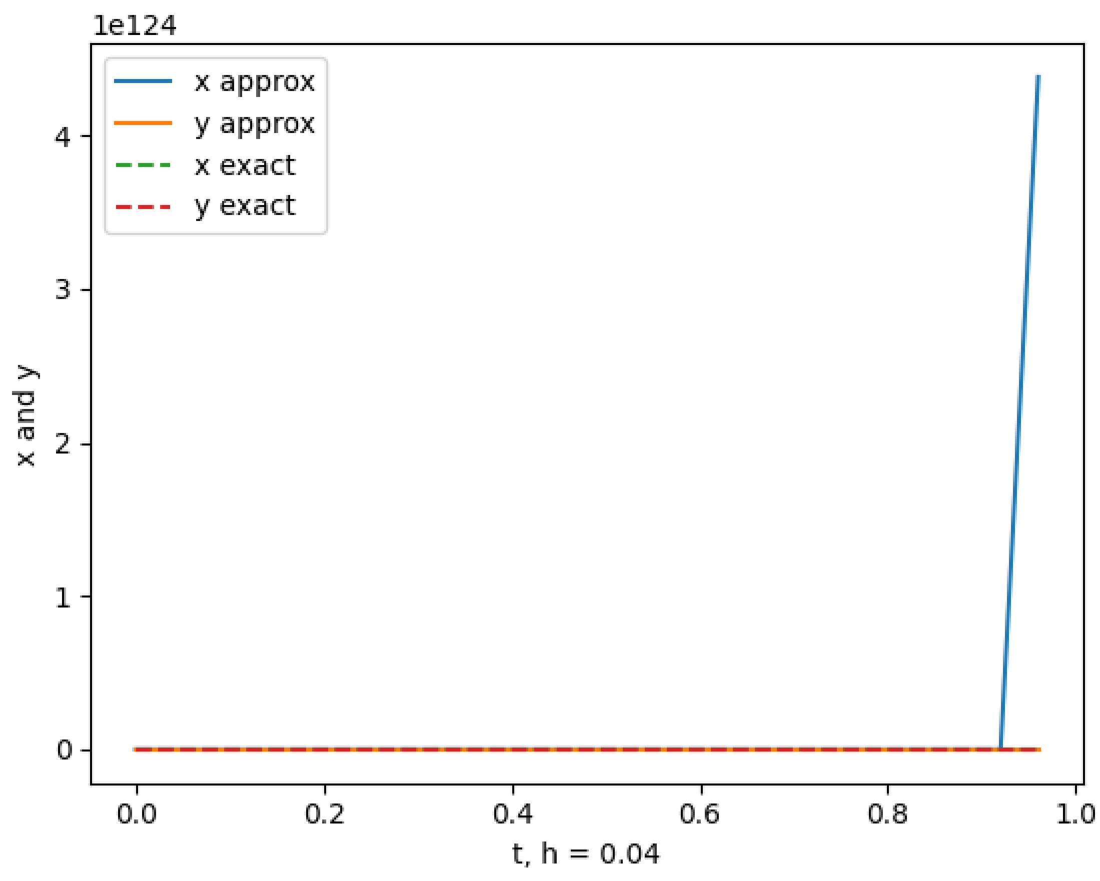
Question 3

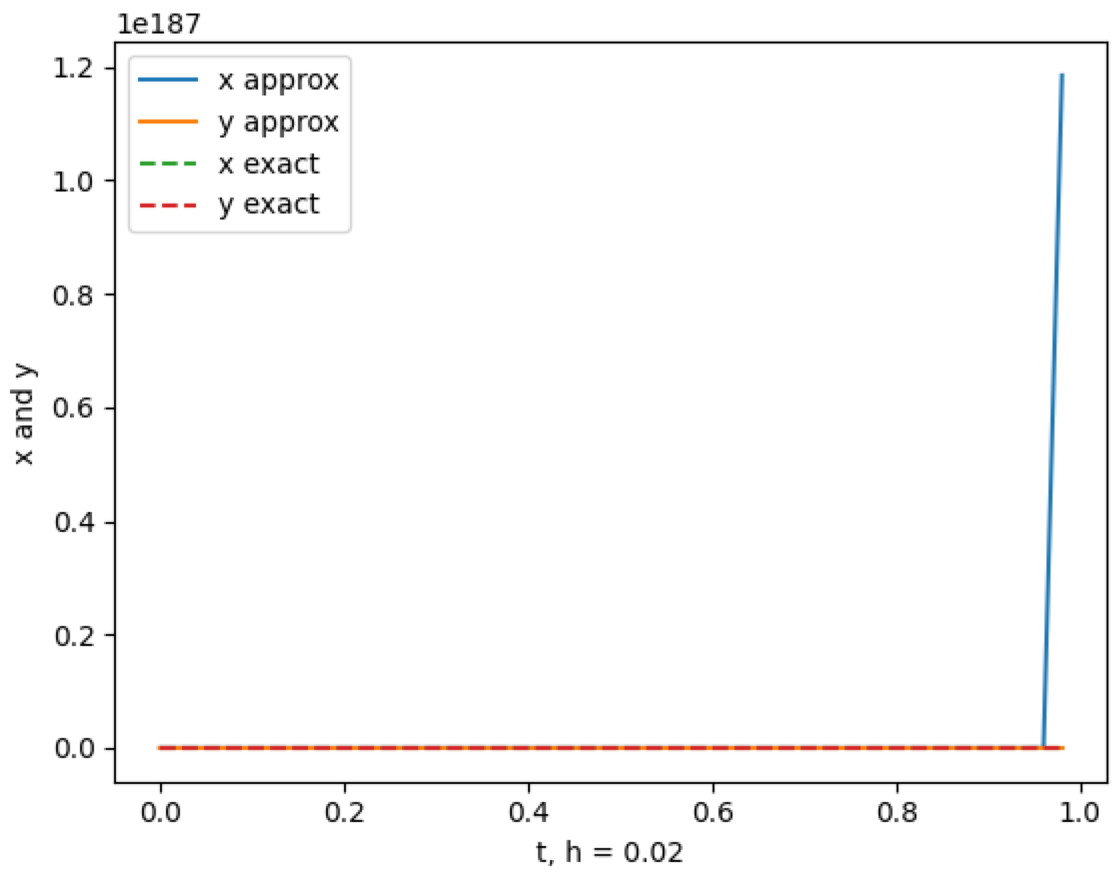
Part A)

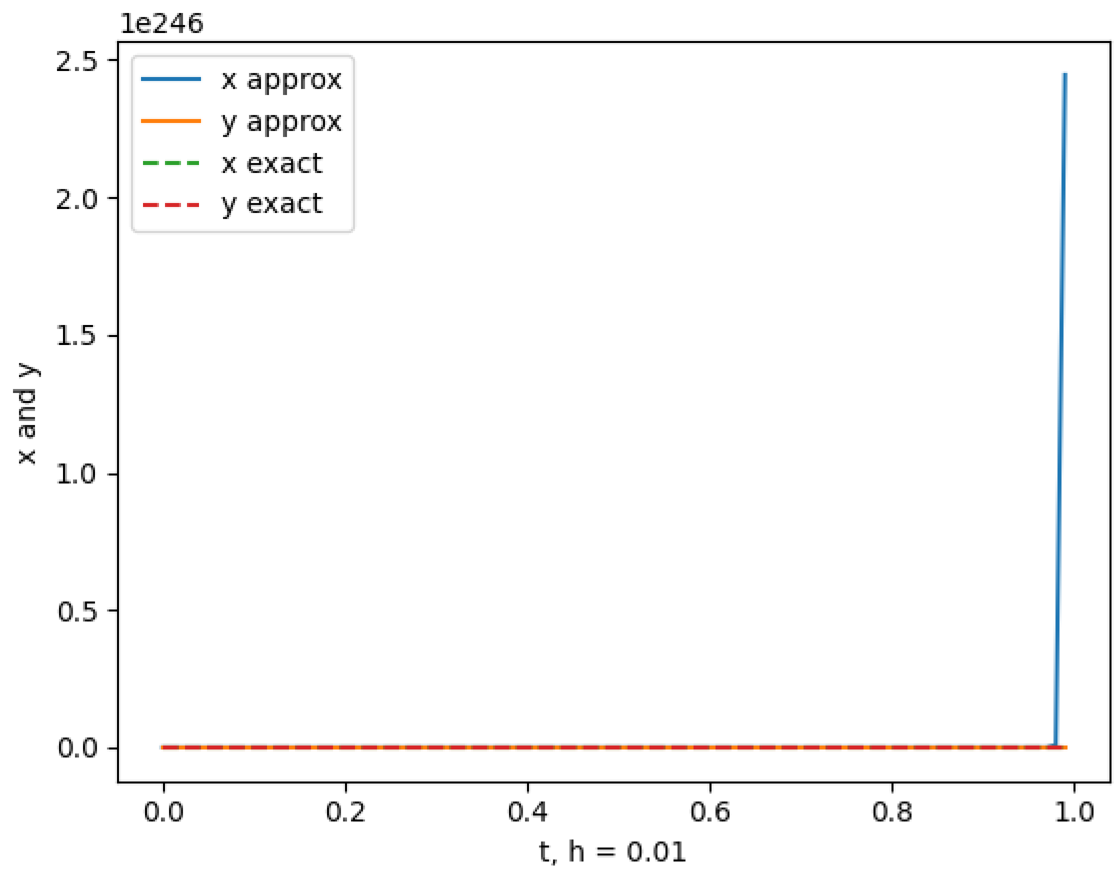
Code:

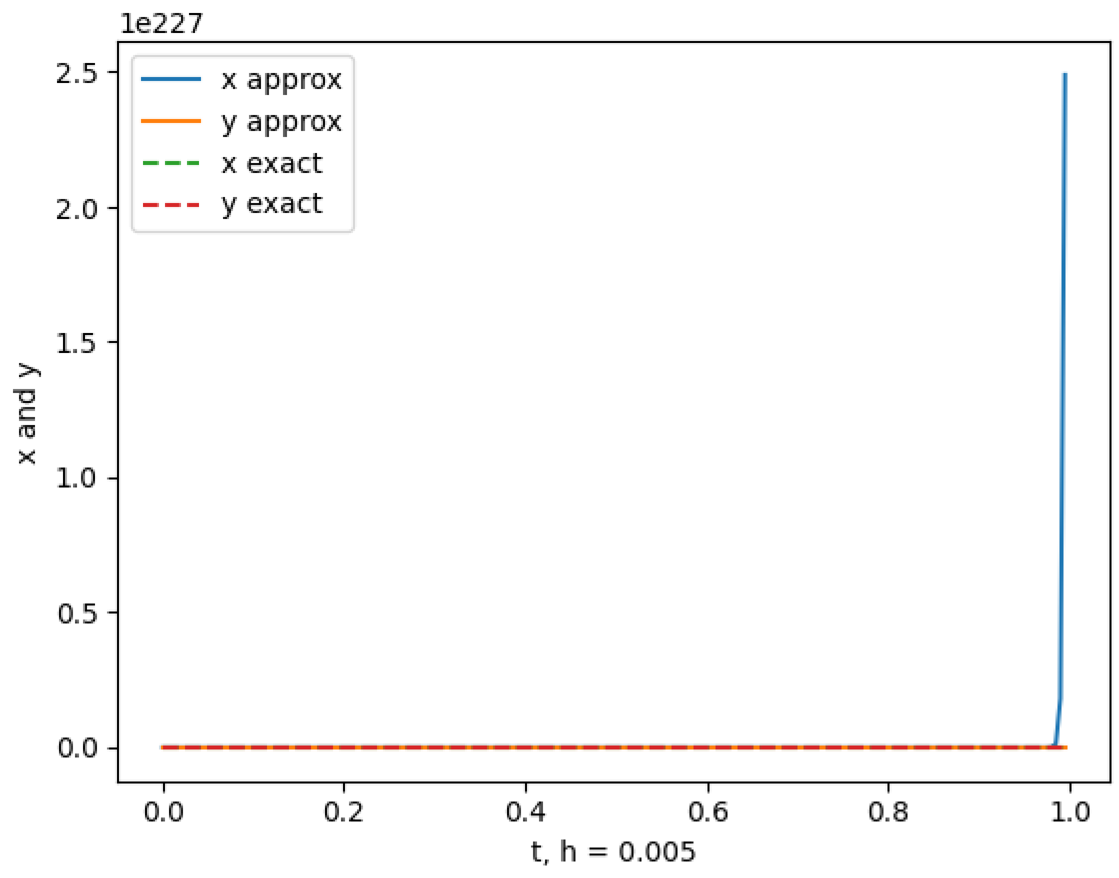
```
def solve_and_plot_with_runge_kutta_4(h):
    def exact_solution(t):
        '''This function returns the exact solution of the differential equation'''
        x = np.exp(-1000*t) * (9979/9999) + np.exp(-t/10) * (20/9999)
        y = 2 * np.exp(-t/10)
        return x, y
    #use our runge_kutta_4th_order function to solve the differential equation
    def ddtxy(t, XY):
        '''This function returns the derivative of x and y at time t'''
        A = np.array([[-1000, 1],
                      [0, -1/10]])
        dXYdt = A @ XY
        return dXYdt
    def runge_kutta_4th_order(t, XY, h, f):
        '''This function returns the approximate value of the function at t + h'''
        #intermediate slope estimations
        k1 = f(t, XY)
        k2 = f(t + h/2, XY + h/2 * k1)
        k3 = f(t + h/2, XY + h/2 * k2)
        k4 = f(t + h, XY + h * k3)
        #update the value of the function
        XY_next = XY + ((h/6) * (k1 + 2*k2 + 2*k3 + k4))
        return XY_next
    t = 0
    t_values = [t]
    x0 = 1
    y0 = 2
    xy0 = np.array([x0, y0])
    XY_values = [xy0]
    while t < 1:
        '''This loop solves the differential equation using the runge_kutta_4th_order function'
        t_values.append(t)
        xy0 = runge_kutta_4th_order(t, xy0, h, ddtxy)
        # print(xy0)
        XY_values.append(xy0)
        t += h
    #plot the results
    XY_values = np.array(XY_values)
    plt.plot(t_values, XY_values[:, 0], label='x approx')
    plt.plot(t_values, XY_values[:, 1], label='y approx')
    #plot the exact solution
    plt.plot(t_values, [exact_solution(t)[0] for t in t_values], label='x exact', linestyle='--
    plt.plot(t_values, [exact_solution(t)[1] for t in t_values], label='y exact', linestyle='--
    plt.xlabel('t, h = {}'.format(h))
    plt.ylabel('x and y')
    # plt.yscale('log')

    plt.legend()
    plt.show()
```

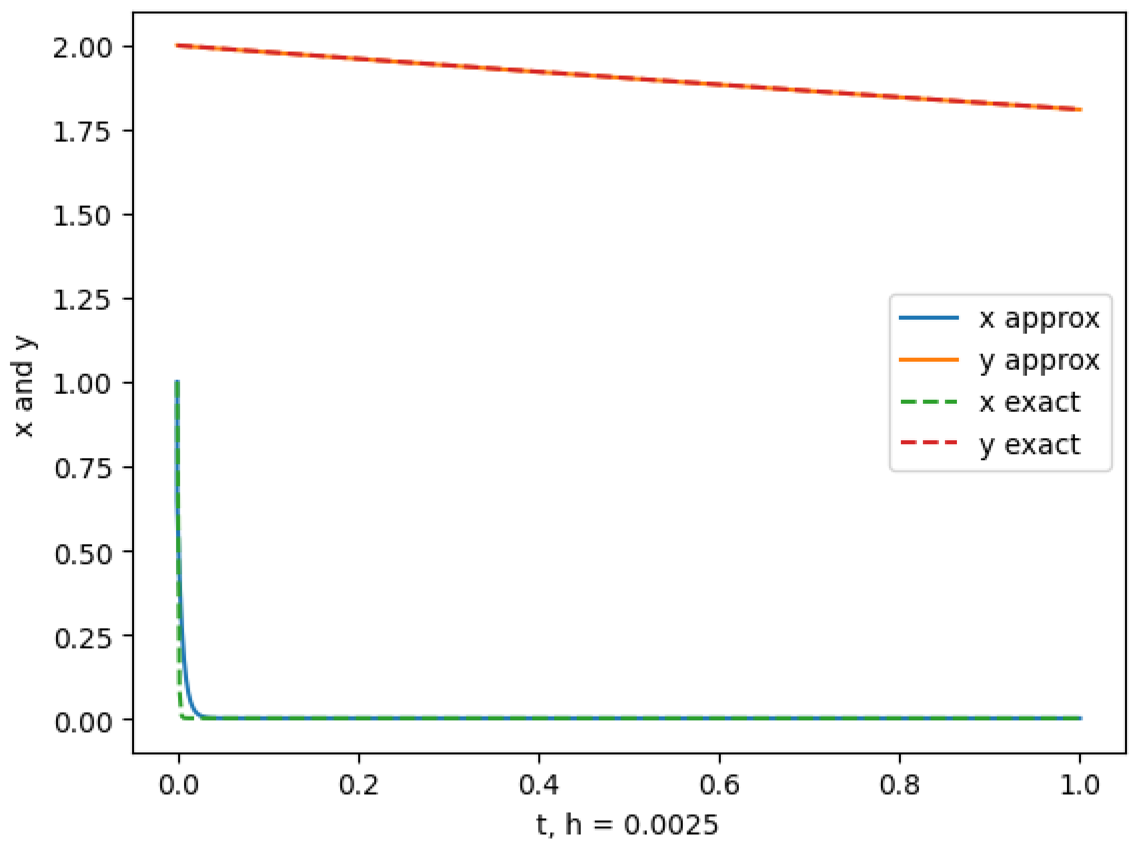
Plots:

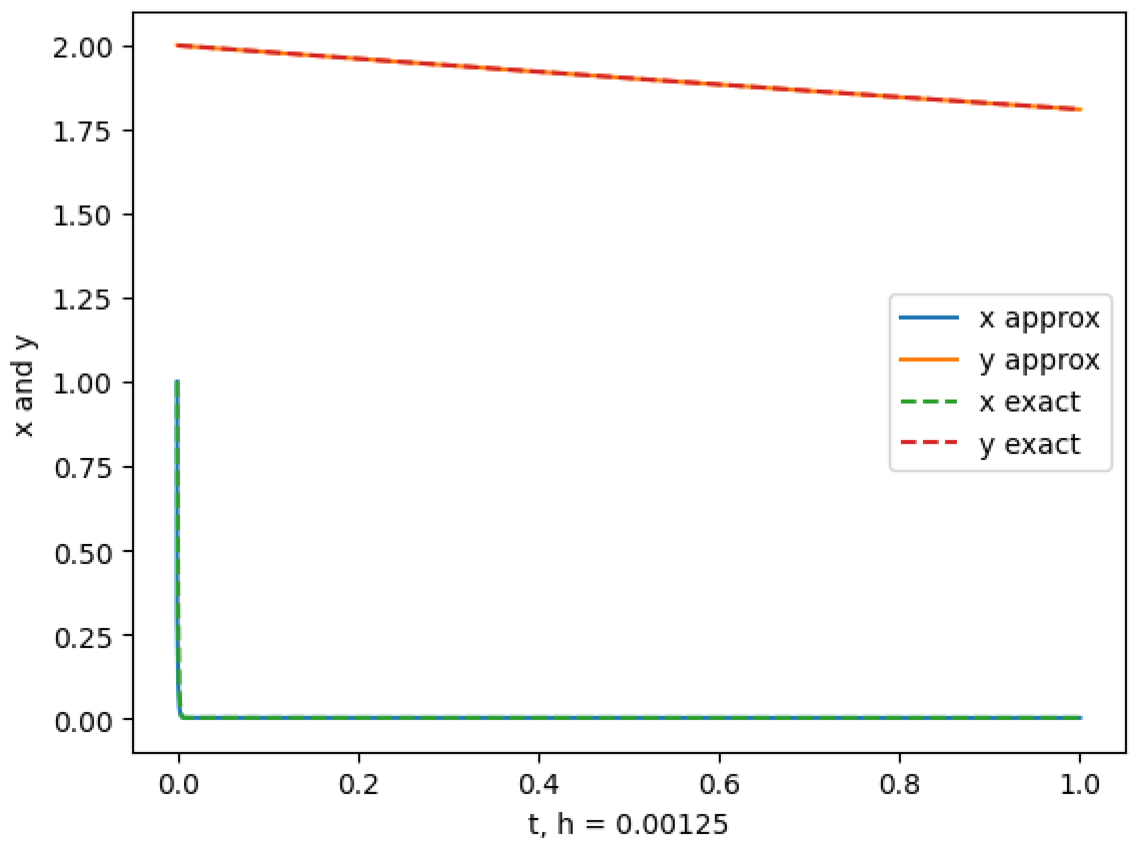





1e246
2.5
2.0
1.5
1.0
0.5
0.0
x and y
x approx
y approx
x exact
y exact
0.0
0.2
0.4
0.6
0.8
1.0
t, h = 0.01


1e227
2.5
2.0
1.5
1.0
0.5
0.0
x and y
x approx
y approx
x exact
y exact
0.0
0.2
0.4
0.6
0.8
1.0
t, h = 0.005


2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
x and y
x approx
y approx
x exact
y exact
0.0
0.2
0.4
0.6
0.8
1.0
t, h = 0.0025



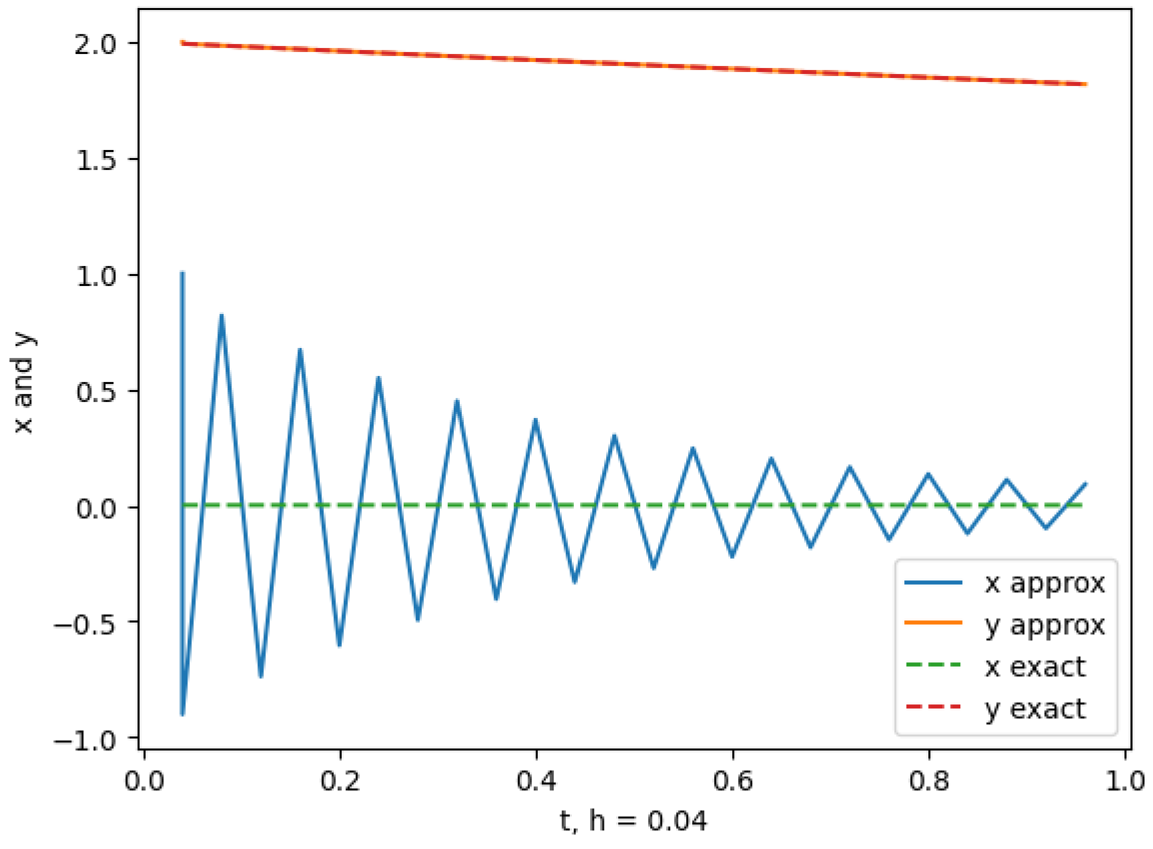
Part B)

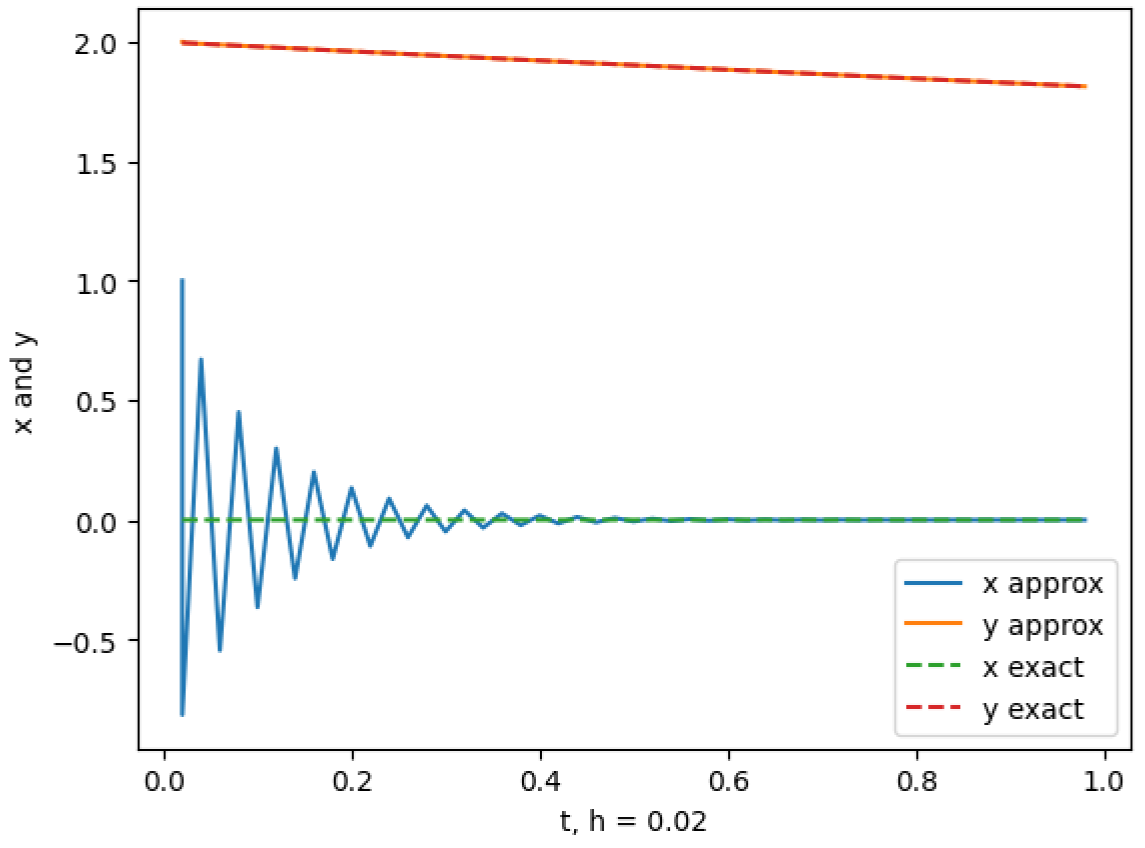
i)

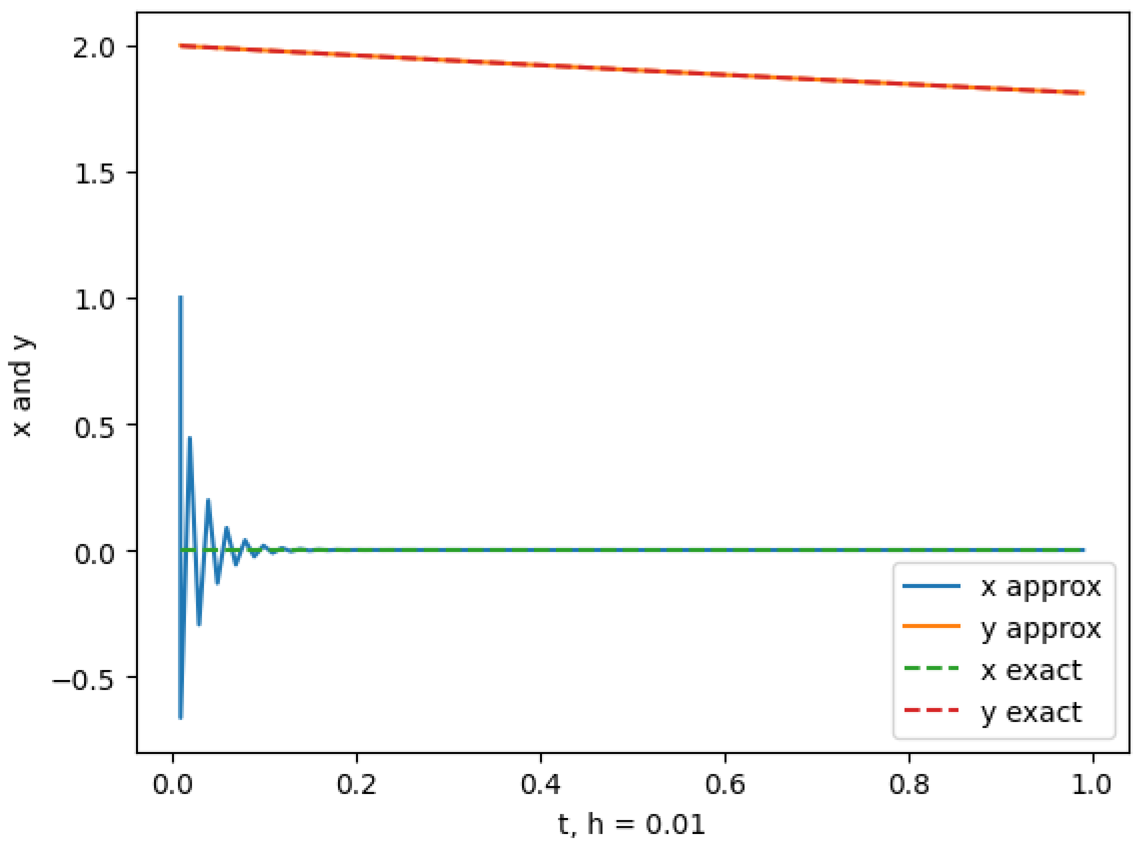
Code:

```
def solve_adef solve_and_plot_with_adam_moulton_2(h):
    def exact_solution(t):
        '''This function returns the exact solution of the differential equation'''
        x = np.exp(-1000*t) * (9979/9999) + np.exp(-t/10) * (20/9999)
        y = 2 * np.exp(-t/10)
        return x, y
    #use our runge_kutta_4th_order function to solve the differential equation
    def ddtxy(xy):
        '''This function returns the derivative of x and y at time t'''
        A = np.array([[-1000, 1],
                      [0, -1/10]])
        dXYdt = A @ xy
        return dXYdt

    def adam_moulton_2(t, xy, h, f):
        '''This function returns the value of the function at t + h'''
        #solved analytically for next values of x and y
        x, y = xy
        x_next = 1/(500*h + 1) * (x - 500*h*x + h*y)
        y_next = 1/(1 + h/20)*(y - (h/20)*y)
        xy_next = np.array([x_next, y_next])
        f_next_approx = f(xy_next)
        xy_next = xy + (h/2) * (f_next_approx + f(xy))
        return xy_next
    t = h
    t_values = [t]
    x0 = 1
    y0 = 2
    xy0 = np.array([x0, y0])
    XY_values = [xy0]
    while t < 1:
        '''This loop solves the differential equation using the adam_moulton_2 function'''
        t_values.append(t)
        xy0 = adam_moulton_2(t, xy0, h, ddtxy)
        XY_values.append(xy0)
        t += h
    #plot the results
    XY_values = np.array(XY_values)
    plt.plot(t_values, XY_values[:, 0], label='x approx')
    plt.plot(t_values, XY_values[:, 1], label='y approx')
    #plot the exact solution
    plt.plot(t_values, [exact_solution(t)[0] for t in t_values], label='x exact', linestyle='--
    plt.plot(t_values, [exact_solution(t)[1] for t in t_values], label='y exact', linestyle='--
    plt.xlabel('t, h = {}'.format(h))
    plt.ylabel('x and y')
    # plt.yscale('log')
    plt.legend()
    plt.show()nd_plot_with_adam_moulton_2(h):
    def exact_solution(t):
        '''This function returns the exact solution of the differential equation'''
        x = np.exp(-1000*t) * (9979/9999) + np.exp(-t/10) * (20/9999)
        y = 2 * np.exp(-t/10)
        return x, y
    #use our runge_kutta_4th_order function to solve the differential equation
    def ddtxy(xy):
        '''This function returns the derivative of x and y at time t'''
        A = np.array([[-1000, 1],
                      [0, -1/10]])
        dXYdt = A @ xy
        return dXYdt

    def adam_moulton_2(t, xy, h, f):
        '''This function returns the value of the function at t + h'''
        #solved analytically for next values of x and y
        x, y = xy
        x_next = 1/(500*h + 1) * (x - 500*h*x + h*y)
        y_next = 1/(1 + h/20)*(y - (h/20)*y)
        xy_next = np.array([x_next, y_next])
        f_next_approx = f(xy_next)
        xy_next = xy + (h/2) * (f_next_approx + f(xy))
        return xy_next
    t = h
    t_values = [t]
    x0 = 1
    y0 = 2
    xy0 = np.array([x0, y0])
    XY_values = [xy0]
    while t < 1:
        '''This loop solves the differential equation using the adam_moulton_2 function'''
        t_values.append(t)
        xy0 = adam_moulton_2(t, xy0, h, ddtxy)
        XY_values.append(xy0)
        t += h
    #plot the results
    XY_values = np.array(XY_values)
    plt.plot(t_values, XY_values[:, 0], label='x approx')
    plt.plot(t_values, XY_values[:, 1], label='y approx')
    #plot the exact solution
    plt.plot(t_values, [exact_solution(t)[0] for t in t_values], label='x exact', linestyle='--
    plt.plot(t_values, [exact_solution(t)[1] for t in t_values], label='y exact', linestyle='--
    plt.xlabel('t, h = {}'.format(h))
    plt.ylabel('x and y')
    # plt.yscale('log')
    plt.legend()
    plt.show()
```

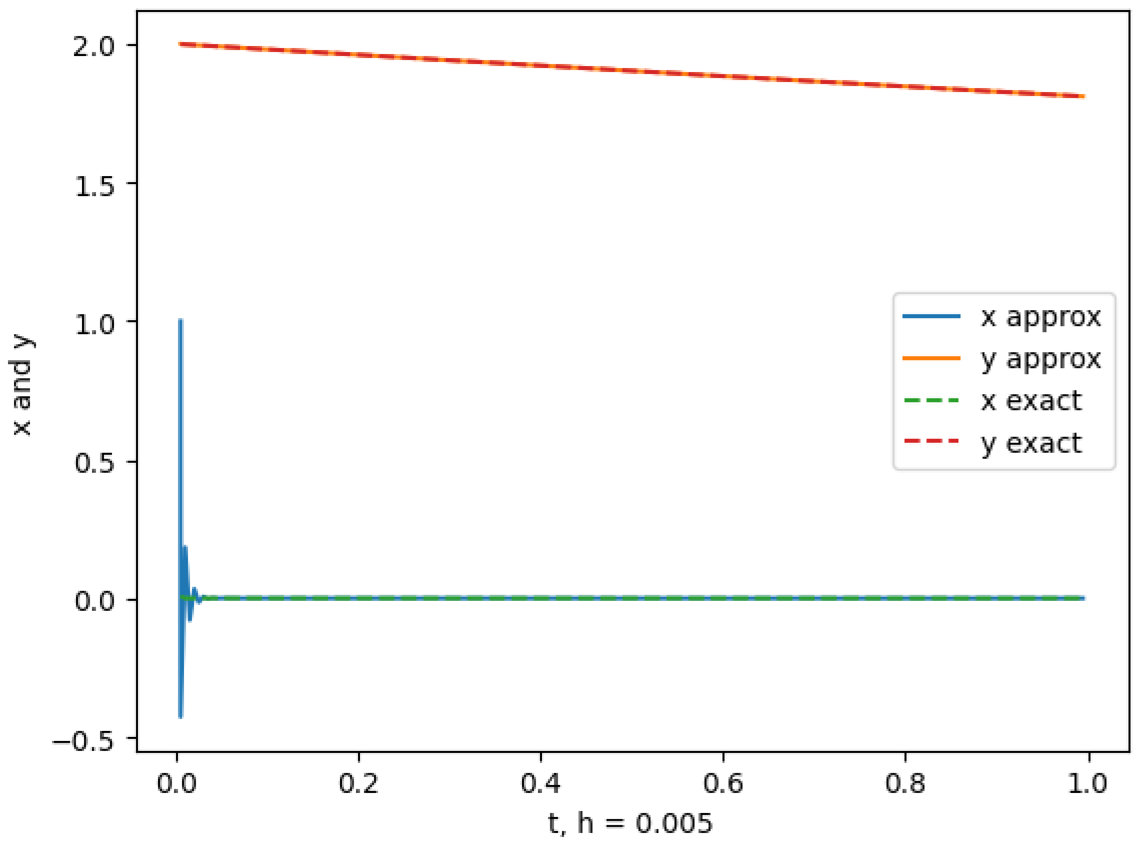
Plots:



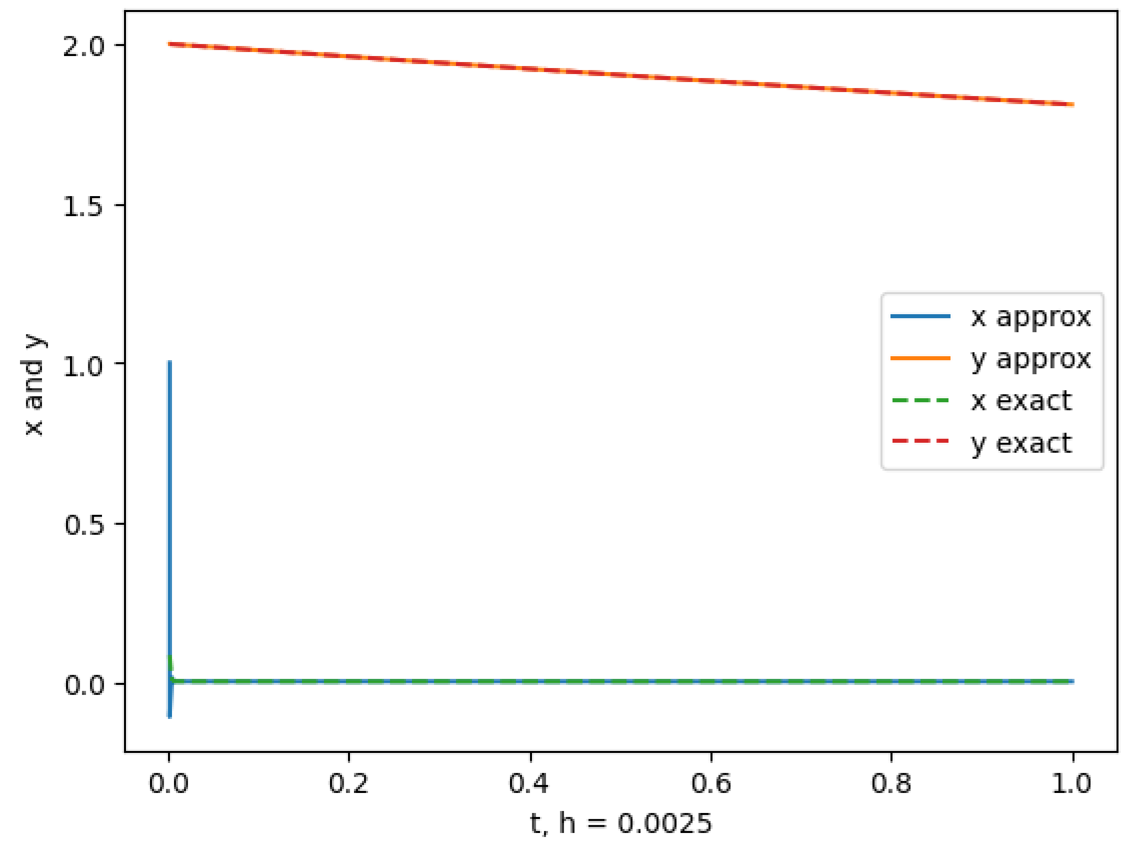




x and y
t, h = 0.01
x approx
y approx
x exact
y exact



x and y
t, h = 0.005
x approx
y approx
x exact
y exact



x and y
t, h = 0.0025
x approx
y approx
x exact
y exact

We can get a reasonable solution (well, depending on your definition of reasonable I suppose), with a time step of 0.0025, which is twice as large as what it took with RK4.

ii)

While RK4 converges with order $O(h^4)$, it is an explicit method.The problem we are solving is stiff, so RK4 requires a much smaller timestep in order to exhibit good behavior. Why is our problem stiff? Because of the -1000 term d/dt

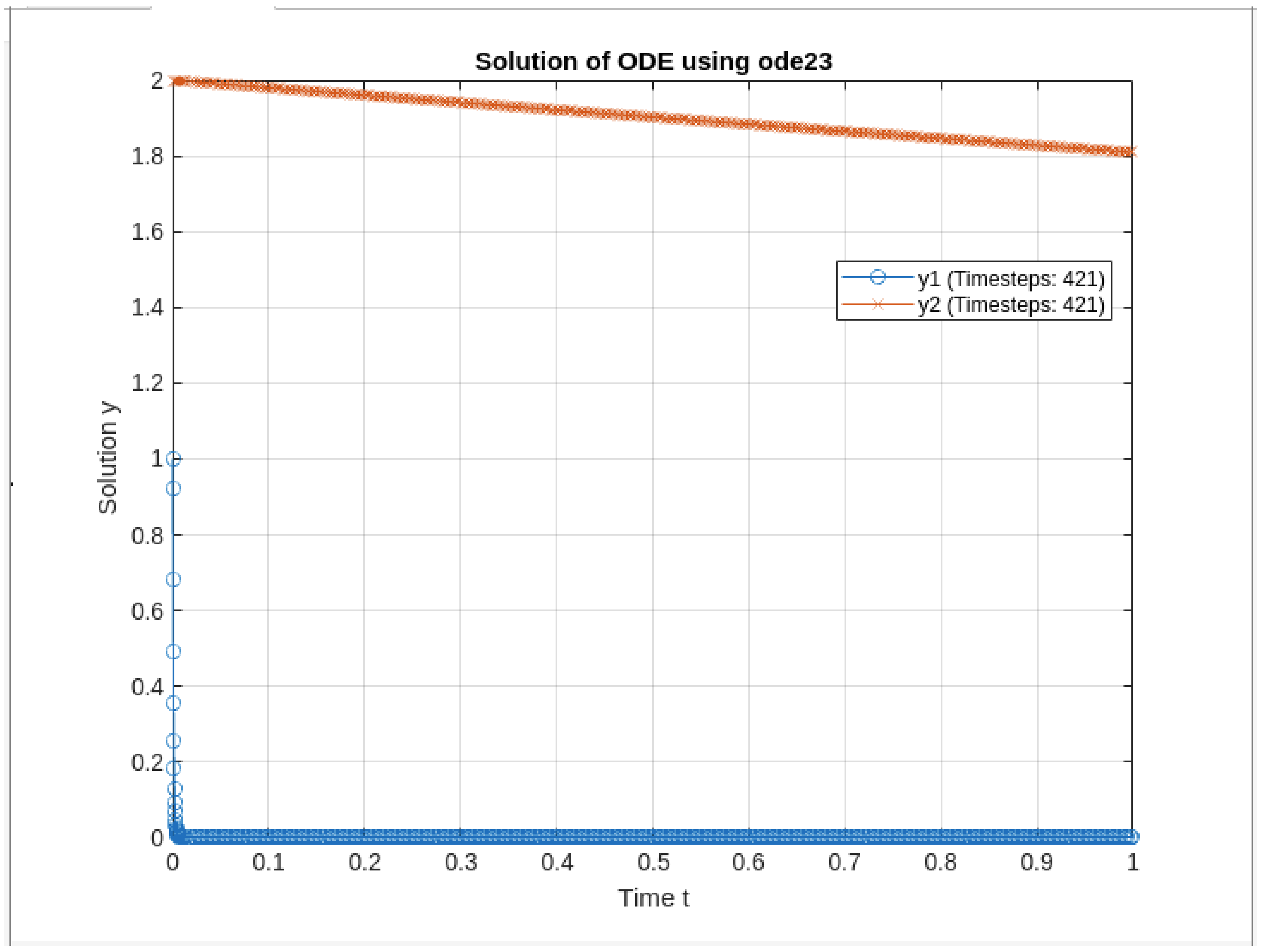
Part C)

I did the prior parts in python as that is my prefered programming language. Here I use the matplot built in functions **ode23** and **ode45**. I use the provided ode function:

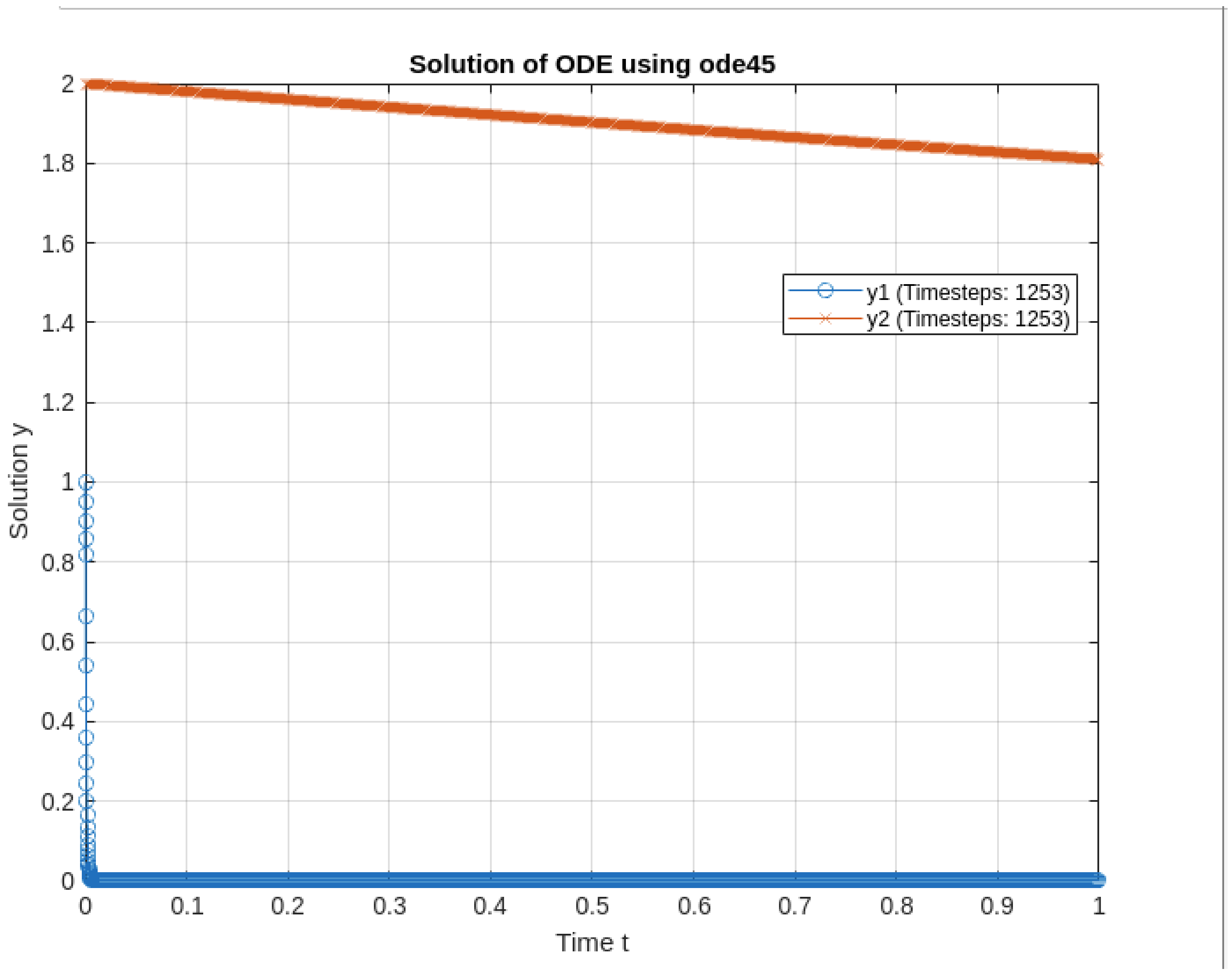
```
function yp=myodefunction(t,y)
y1 = -1000*y(1) + y(2);
y2 = -y(2)/10;
yp=[y1; y2];
end
```

And call it with:

```
[t, y] = ode23(@myodefunction, [0 1], [1 2]); %ode45(@myodefunction, [0,1], [1,2]);

numTimesteps = length(t);

figure;

plot(t, y(:,1), '-o', t, y(:,2), '-x');

legendInfo = {sprintf('y1 (Timesteps: %d)', numTimesteps), sprintf('y2 (Timesteps: %d)', numTime
legend(legendInfo, 'Location', 'Best');

xlabel('Time t');
ylabel('Solution y');
title('Solution of ODE using ode23');

grid on;
```



And with **ode45**:



As we can see in the legend, **ode23** used 421 timesteps, and **ode45** used 1253. They solve it with the same accuracy as the good solutions from RK4 and AM2. Like the python implementation of RK4, **ode45** needs $h \approx \frac{1}{1000}$. Surprisingly, **ode23** does not need as many timesteps! It seems like because the **ode45** method uses more terms than the **ode23** method, there is more of a chance for system to “blow up”, so h must be smaller.

What I mean by this; Observe RK3:

$$
\begin{aligned}
k_1 &= f(t_n, y_n),\\
k_2 &= f\left(t_n + \frac{1}{2}h, y_n + \frac{1}{2}hk_1\right),\\
k_3 &= f(t_n + h, y_n - hk_1 + 2hk_2),\\
y_{n+1} &= y_n + \frac{h}{6}(k_1 + 4k_2 + k_3),
\end{aligned}
$$

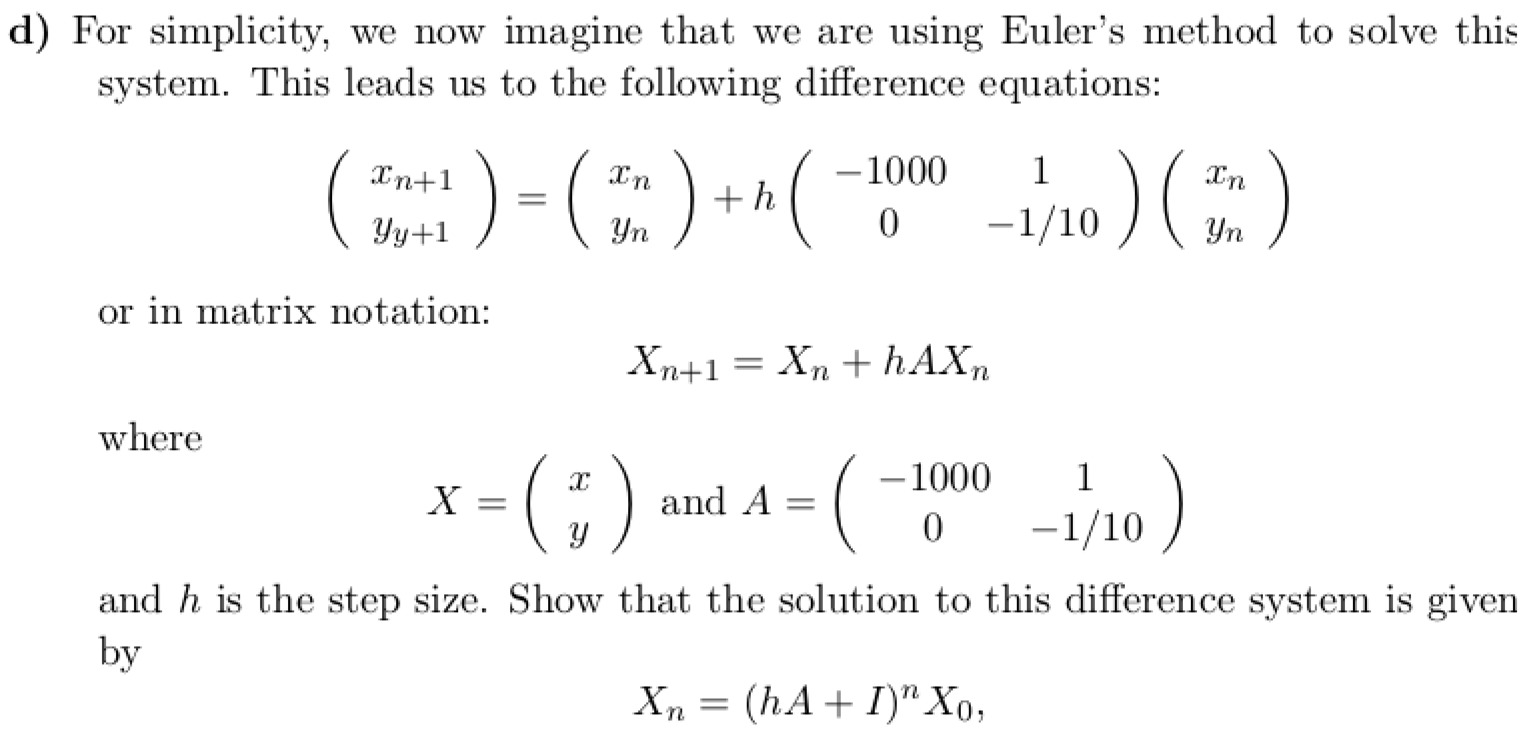
And RK5:

$$
\begin{aligned}
k_1 &= f(t_n, y_n),\\
k_2 &= f\left(t_n + \frac{1}{4}h, y_n + \frac{1}{4}hk_1\right),\\
k_3 &= f\left(t_n + \frac{1}{4}h, y_n + \frac{1}{8}hk_1 + \frac{1}{8}hk_2\right),\\
k_4 &= f\left(t_n + \frac{1}{2}h, y_n - \frac{1}{2}hk_2 + hk_3\right),\\
k_5 &= f\left(t_n + \frac{3}{4}h, y_n + \frac{3}{16}hk_1 + \frac{9}{16}hk_4\right),\\
k_6 &= f\left(t_n + h, y_n - \frac{3}{7}hk_1 + \frac{2}{7}hk_2 + \frac{12}{7}hk_3 - \frac{12}{7}hk_4 + \frac{8}{7}hk_5\right),\\
y_{n+1} &= y_n + \frac{h}{90}(7k_1 + 32k_3 + 12k_4 + 32k_5 + 7k_6),
\end{aligned}
$$

h is used many more times in RK5, so it must be smaller to avoid blowing up.

Part D)

i)



d) For simplicity, we now imagine that we are using Euler's method to solve this system. This leads us to the following difference equations:

$$\begin{pmatrix} x_{n+1} \\ y_{y+1} \end{pmatrix} = \begin{pmatrix} x_n \\ y_n \end{pmatrix} + h \begin{pmatrix} -1000 & 1 \\ 0 & -1/10 \end{pmatrix} \begin{pmatrix} x_n \\ y_n \end{pmatrix}$$

or in matrix notation:

$$X_{n+1} = X_n + hAX_n$$

where

$$X = \begin{pmatrix} x \\ y \end{pmatrix} \text{ and } A = \begin{pmatrix} -1000 & 1 \\ 0 & -1/10 \end{pmatrix}$$

and h is the step size. Show that the solution to this difference system is given by

$$X_n = (hA + I)^n X_0,$$

Lets take as our base case that

$$X_{n+1} = X_n + hAX_n = (I + hA)X_n$$

And then we can show

$$X_{n+2} = (I + hA)X_{n+1} = (I + hA)(I + hA)X_n = (I + hA)^2 X_n$$

So we have $X_n = (hA + I)^n X_0$ by induction

ii)

Deduce the value of h for which the above solution will give meaningful results (e.g. a solution that does not blow up for large n). Compare with your prediction in part I.

We are worried about this matrix blowing up:

$$B^n = (hA + I)^n = \begin{pmatrix} -1000h + 1 & h \\ 0 & 1 - \frac{h}{10} \end{pmatrix}^n$$

How can we analyze it's behavior as n increases? Let's use the hint and write it in the form $P^{-1}DP$ by diagonalizing B.

1. First me must find the eigenvalues of B
   a. need λ such that $det(B - I\lambda) = 0$
   b. we get $\lambda_1 = -1000h + 1, \lambda_2 = 1 - \frac{h}{10}$
   c. These will be the diagonals of D
2. Then we need the eigenvectors to form the columns of P
   a. $(B - I\lambda_1)x = 0$
      i. $x = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$
   b. $(B - I\lambda_2)y = \begin{pmatrix} -1000h + \frac{h}{10} & h \\ 0 & 0 \end{pmatrix} y = 0$
      i. $y = \begin{pmatrix} \frac{10}{9999} \\ 1 \end{pmatrix}$
3. $D = \begin{pmatrix} -1000h + 1 & 0 \\ 0 & 1 - \frac{h}{10} \end{pmatrix}$, $P = \begin{pmatrix} 1 & \frac{10}{9999} \\ 0 & 1 \end{pmatrix}$, $P^{-1} = \begin{pmatrix} 1 & \frac{-10}{9999} \\ 0 & 1 \end{pmatrix}^n$
4. And we now have:
5. $B^n = PD^nP^{-1}$
6. So we can see that to avoid D blowing up we need that $|1 - 1000h| \leq 1$
   a. $-1 \leq 1 - 1000h \leq 1$
   b. $-2 \leq -1000h \leq 2$
   c. $2 \geq 1000h \geq -2$
   d. $\frac{1}{500} \geq h \geq \frac{-1}{500}$

And since h is our timestep (and therefore must be positive), we have that $h \leq \frac{1}{500}$, which is a similar result to what we see with the **ode23** call (subdividing into 423 points).

Question 4

4. **Bonus question** Prove that the Vandermonde determinant is given by

$$\det \begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{bmatrix} = \prod_{i>j}(x_i - x_j).$$

Let V_n refer to the $n \times n$ Vandermonde matrix. We can use the fact that if we add a scalar multiple of a column to another column that the determinant doesn't change.

$$V_n = \begin{pmatrix} 1, x_0, x_0^2 \cdots x_0^n \\ 1, x_1, x_1^2, \cdots, x_1^n \\ \vdots \ \vdots \ \vdots \ \vdots \\ 1, x_n, x_n^2, \cdots x_n^n \end{pmatrix}$$

We can subtract from each column the previous column multipled by x_0 and get;

$$V_n = \begin{pmatrix} 1, 0, 0 \cdots 0 \\ 1, x_1 - x_0, x_1(x_1 - x_0), \cdots, x_1^{n-1}(x_1 - x_0) \\ \vdots \ \vdots \ \vdots \ \vdots \\ 1, x_n - x_0, x_n(x_n - x_0), \cdots x_n^{n-1}(x_n - x_0) \end{pmatrix}$$

We can then use the Laplace expansion to transform the matrix while preserving the determinant.

$$R = \begin{pmatrix} x_1 - x_0, x_1(x_1 - x_0), \cdots, x_1^{n-1}(x_1 - x_0) \\ \vdots \ \vdots \ \vdots \ \vdots \\ x_n - x_0, x_n(x_n - x_0), \cdots x_n^{n-1}(x_n - x_0) \end{pmatrix}$$

With $det(R) = det(V_n)$. All the entries of R in the ith row have are a multiple of $x_i - x_0$, so we can them out and get:

$$det(R) = (x_1 - x_0)(x_2 - x_0) \cdots (x_n - x_0) det(\begin{pmatrix} 1, x_1, x_1^2 \cdots x_1^{n-1} \\ 1, x_2, x_2^2, \cdots x_2^{n-1} \\ \vdots \quad \vdots \\ 1, x_n, x_n^2, \cdots, x_n^{n-1} \end{pmatrix}) = det(V_n)$$

Note that the matrix above is a Vandermonde matrix dimensions $n - 1 \times n - 1$. We can do the above column elimination and Laplace expansion repeatedly until we have;

$$det(V_n) = \Pi_{i>j}(x_i - x_j)$$

Deduce that the interpolation polynomial only exists if all the interpolation points are distinct.

This is actually easy to see as a consequence of the result we just showed with the Vandermonde matrix. The interpolation polynomial is equivalent to a solution to this set of equations:

$$a_0 + a_1 x_i + a_2 x_2^2 \cdots a_n x_i^n = f(x_i), i = 0, 1, \ldots n$$

For the $n+1$ unknown a_i's. This can be written in matrix form as

$$V_n \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} f_0 \\ f_1 \\ \vdots \\ f_n \end{pmatrix}$$

And we know from above that if the points aren't distinct, then we will have $det(V_n) = (x_1 - x_0) \cdots 0 \cdots$, i.e V_n is singular and has no inverse, therefore the system above has no solution. Since the solutions to the above are equivalent to the interpolating polynomial, and we have no solutions, then we have no interpolating polynomial.