Øving 2 Emil Tveiten TMA4130

1) a) $f(x) = -2x^4 + 2x^2 - 3x + 2$, $x_0 = -1$

Løsning: I) Regner ut deriverte

$f(x) = -2x^4 + 2x^2 - 3x + 2$ $\quad f(-1) = -2 + 2 + 3 + 2 = \underline{5}$

$f'(x) = -8x^3 + 4x - 3$ $\quad f'(-1) = 8 - 4 - 3 = \underline{1}$

$f^{(2)}(x) = -24x^2 + 4x$ $\quad f^{(2)}(-1) = -24 - 4 = \underline{-28}$

$f^{(3)}(x) = -48x + 4$ $\quad f^{(3)}(-1) = 48 + 4 = \underline{52}$

$f^{(4)}(x) = -48$ $\quad f^{(4)}(-1) = \underline{-48}$

For alle $f^{(k)}(x)$, $k > 4 \Rightarrow f^{(k)}(x) = 0$

II) Regner ut Taylor polynomer:

$$T_n(x) = \sum_{k=0}^{n} \frac{f^{(k)}(x_0)}{k!} \cdot (x - x_0)^k$$

$T_0(x) = f(-1) = \underline{5}$

$T_1(x) = T_0(x) + \frac{f'(x_0)}{1!}(x - x_0)^1 = 5 + \frac{1}{1} \cdot (x+1) = \underline{6 + x}$

$T_2(x) = T_1(x) + \frac{f^{(2)}(x_0)}{2!}(x - x_0)^2 = 6 + x - \frac{28}{2}(x+1)^2 = \underline{-14x^2 - 27x - 8}$

$T_3(x) = T_2(x) + \frac{f^{(3)}(x_0)}{3!}(x - x_0)^3 = -14x^2 - 27x - 8 + \frac{52}{6}(x+1)^3 = \underline{\frac{26}{3}x^3 + 12x^2 - x + \frac{2}{3}}$

$T_4(x) = T_3(x) + \frac{f^{(4)}(x_0)}{4!}(x - x_0)^4 = \frac{26}{3}x^3 + 12x^2 - x + \frac{2}{3} - \frac{48}{24}(x+1)^4 = \underline{-2x^4 + \frac{2x^3}{3} - 9x - \frac{4}{3}}$

4) Taylor rekke av $g(x) = e^{1-2x}$, $x_0 = 0$

Formel: $g(x) = \sum_{k=0}^{\infty} \frac{g^{(k)}(x_0)}{k!}(x-x_0)^k \overset{x_0=0}{\Longrightarrow} \sum_{k=0}^{\infty} \frac{g^{(k)}(0)}{k!} \cdot x^k$

Løsning:

$g(x) = e^{1-2x}$	$g(0) = e$
$g'(x) = -2e^{1-2x}$	$g'(0) = -2e$
$g^{(2)}(x) = 4e^{1-2x}$	$g^{(2)}(0) = 4e$
$g^{(3)}(x) = -8e^{1-2x}$	$g^{(3)}(0) = -8e$
$g^{(4)}(x) = 16e^{1-2x}$	$g^{(4)}(0) = 16e$

→ Se at $g^{(k)}(x) = (-1)^k \cdot (2)^k \cdot e^{1-2x}$

$= (-2)^k \cdot e^{1-2x}$ og $g^{(k)}(0) = (-2)^k \cdot e$

Taylorrekka av $g(x)$ blir da:

$$e^{1-2x} = \sum_{k=0}^{\infty} \frac{g^{(k)}(0)}{k!} \cdot x^k$$

$$= \sum_{k=0}^{\infty} \frac{(-2)^k \cdot e}{k!} \cdot x^k$$

2a) Regn ut $l_n(x)$ for:

$x_0 = -1,\ x_1 = 0,\ x_2 = 1,\ x_3 = 2$

$$l_0(x) = \left(\frac{x-x_1}{x_0-x_1}\right)\cdot\left(\frac{x-x_2}{x_0-x_2}\right)\cdot\left(\frac{x-x_3}{x_0-x_3}\right) = \frac{x}{-1}\cdot\frac{x-1}{-2}\cdot\frac{x-2}{-3} = \frac{x\cdot(x-1)\cdot(x-2)}{-6}$$

$$l_1(x) = \left(\frac{x-x_0}{x_1-x_0}\right)\cdot\left(\frac{x-x_2}{x_1-x_2}\right)\cdot\left(\frac{x-x_3}{x_1-x_3}\right) = \frac{x+1}{1}\cdot\frac{x-1}{-1}\cdot\frac{x-2}{-2} = \frac{(x+1)(x-1)(x-2)}{2}$$

$$l_2(x) = \left(\frac{x-x_0}{x_2-x_0}\right)\cdot\left(\frac{x-x_1}{x_2-x_1}\right)\cdot\left(\frac{x-x_3}{x_2-x_3}\right) = \frac{x+1}{2}\cdot\frac{x}{1}\cdot\frac{x-2}{-1} = \frac{(x+1)\cdot x\cdot(x-2)}{-2}$$

$$l_3(x) = \left(\frac{x-x_0}{x_3-x_0}\right)\cdot\left(\frac{x-x_1}{x_3-x_1}\right)\cdot\left(\frac{x-x_2}{x_3-x_2}\right) = \frac{x+1}{3}\cdot\frac{x}{2}\cdot\frac{x-1}{1} = \frac{(x+1)\cdot x\cdot(x-1)}{6}$$

b) Interpoler $f(x) = 2^{x^2-4-x}$

x_i	-1	0	1	2
y_i	$\frac{1}{4}$	$\frac{1}{16}$	$\frac{1}{16}$	$\frac{1}{4}$

y_i: $f(x_i)$

Polynomet blir: $P(x) = L_3(x) = y_0 l_0(x) + y_1 l_1(x) + y_2 l_2(x) + y_3 l_3(x)$

$$L_3(x) = \frac{1}{4}\cdot\frac{x\cdot(x-1)\cdot(x-2)}{-6} + \frac{1}{16}\cdot\frac{(x+1)(x-1)(x-2)}{2} + \frac{1}{16}\cdot\frac{(x+1)\cdot x\cdot(x-2)}{-2} + \frac{1}{4}\cdot\frac{(x+1)\cdot x\cdot(x-1)}{6}$$

$$= \text{..... wolfram .....}$$

$$= \frac{1}{32}\left(3x^2 - 3x + 2\right)$$

Sjekk: $L_3(-1) = \frac{1}{4}$, $L_3(0) = \frac{1}{16}$, $L_3(1) = \frac{1}{16}$, $L_3(2) = \frac{1}{4}$ OK!

4a) $P(x) = a_0 + a_1 \cos(x) + a_2 \cos(2x)$

i	0	1	2
x_i	0	$\pi/6$	$\pi/2$
y_i	1	$\frac{3}{4}$	0

$f(x) = \cos^2(x)$

Lösung system

1. $P(x_0) = a_0 + a_1 \cdot \cos(x_0) + a_2 \cdot \cos(2x_0) = a_0 + a_1 \cdot \cos(0) + a_2 \cdot \cos(0)$
2. $P(x_1) = a_0 + a_1 \cdot \cos(x_1) + a_2 \cdot \cos(2x_1) = a_0 + a_1 \cdot \cos\left(\frac{\pi}{6}\right) + a_2 \cdot \cos\left(\frac{\pi}{6} \cdot 2\right)$
3. $P(x_2) = a_0 + a_1 \cdot \cos(x_2) + a_2 \cdot \cos(2x_2) = a_0 + a_1 \cdot \cos\left(\frac{\pi}{2}\right) + a_2 \cdot \cos\left(\frac{\pi}{2} \cdot 2\right)$

→

$P(x_0) = a_0 + a_1 + a_2 = 1$

$P(x_1) = a_0 + a_1 \cdot \frac{\sqrt{3}}{2} + a_2 \cdot \frac{1}{2} = \frac{3}{4}$

$P(x_2) = a_0 + \cancel{a_1 \cdot 0} - a_2 = 0$

Tre ligninger, 3 ukjente: Wolfram :

$a_0 = \frac{1}{2}$

$a_1 = 0$

$a_2 = \frac{1}{2}$

$a_1 = 0$

Sjekk: $a_0 + a_1 \cdot \cos(x) + a_2 \cdot \cos(2x)$

$\rightarrow x = 0 \rightarrow = \frac{1}{2} + \frac{1}{2} \cdot \cos(0) = 1$

$x = \frac{\pi}{6} \rightarrow = \frac{1}{2} + \frac{1}{2} \cdot \cos\left(\frac{\pi}{3}\right) = \frac{3}{4}$

$x = \frac{\pi}{2} \rightarrow = \frac{1}{2} + \frac{1}{2} \cdot \cos(\pi) = 0$

→ $P(x) = \frac{1}{2} + \frac{1}{2} \cdot \cos(2x)$

b) Vis at $P(x) = f(x)$

Hint: $\cos(2x) = \cos^2(x) - \sin^2(x)$

$\rightarrow \sin^2(x) = \cos^2(x) - \cos(2x)$

$\rightarrow \cos^2(x) = 1 - \sin^2(x)$ Regel ①

$\cos^2(x) = 1 - \cos^2(x) + \cos(2x)$ Hintet

$2\cos^2(x) = 1 + \cos(2x)$

$\cos^2(x) = \frac{1}{2} + \frac{1}{2}\cos(2x)$ □

$f(x) = P(x)$

Polynomial Interpolation – Exercise sheet 2

TMA4130/TMA4135 Høst 2023

Lagrange interpolation

Remember how we can implement the cardinal functions and do the Lagrange interpolation:

In [2]:
```
import numpy as np
import matplotlib.pyplot as plt
from numpy import pi
```

Oppgave 2:

c) - Python algoritme som interpolerer valgfri funksjon og punkter.

Skal bare returnere en array med

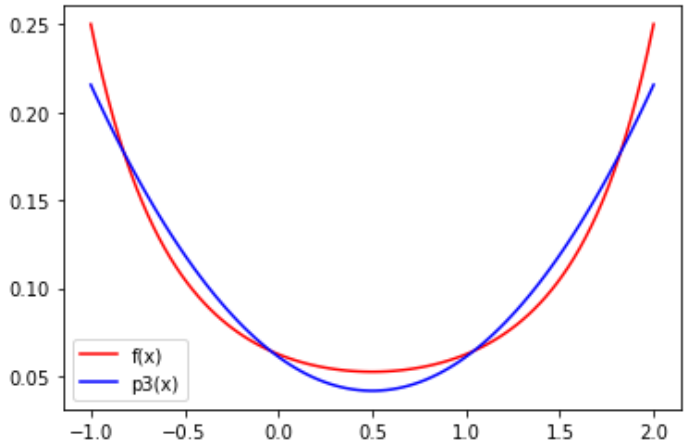
In [80]:
```
# Eksempel funksjon (samme som tidligere i oppgaven):
x0 = np.array([-1,0,1,2])
f = lambda x: 2.0**((x**2)-4-x)
```

In [104]:
```
# Interpoleringen:
def Interpol(x, f, degree):
    return np.polyfit(x,f(x),degree) #Interpolerer av valgfri grad (her: 3, siden 2b var 3.grad)

p = Interpol(x0, f, 3)
print(np.polyval(p,x0))
```

```
[0.25   0.0625 0.0625 0.25  ]
```

In [105]:
```
# Finner feilen e(x) = |f(x) - p(x)|:
x1 = np.linspace(-1,2, 100) # 1. intevall [-1,2] med 100 steg presisjon.
x2 = np.linspace(-5,5, 100) # 2. intevall [-5,5] med 100 steg.

e_x1 = np.abs(np.polyval(Interpol(x1,f,3), x1) - f(x1)) # Kan skrives finere..
e_x2 = np.abs(np.polyval(Interpol(x2,f,3), x2) - f(x2))

max_error_x1 = np.max(e_x1)
max_error_x2 = np.max(e_x2)

print("Max error on [-1,2] =", max_error_x1.round(3))
print("Max error on [-5,5] =", max_error_x2.round(3))

# Plotter f(x) og p(x) på intervallet [-1,2]
plt.plot(x1, f(x1), label='f(x)', color = 'r')
plt.plot(x1, np.polyval(Interpol(x1,f,3), x1), label='p3(x)', color = 'b') # Her har jeg valgt 3. grad
# Høyere grad => mer nøyaktig
plt.legend()
plt.show()
```

```
Max error on [-1,2] = 0.034
Max error on [-5,5] = 49447999.961
```



d) Find the Chebyshev nodes on the interval [−1, 2] for n = 3.

In [114]:
```
def chebFinder(a, b, n):
    k = np.arange(n)

    z_k = np.cos((np.pi / 2)*((2*k+1)/n))

    return ((a+b)/2) + ((b-a)/2)*z_k

print("The Chebyshev nodes on the interval [-1, 2] for n = 3 are:\n", chebFinder(-1,2,3))
```

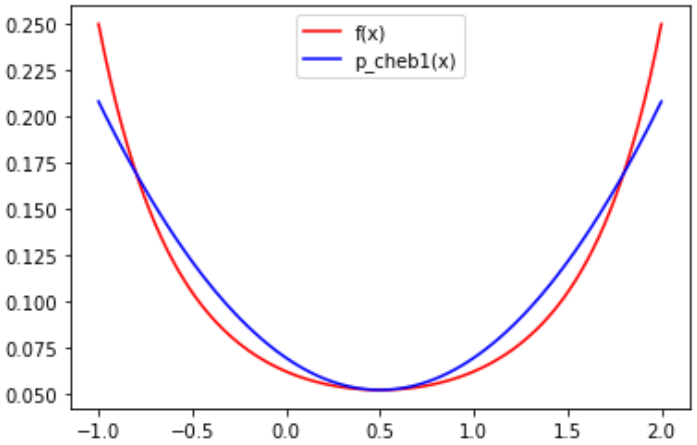
```
The Chebyshev nodes on the interval [-1, 2] for n = 3 are:
 [ 1.79903811  0.5        -0.79903811]
```

e)

Interpolate $f\ (x)$ numerically in the Chebyshev nodes for n = 3. What is the maximal error on the intervals [−1, 2] and [−5, 5]? Plot $f\ (x)$ and the interpolating polynomial in the same plot, on the interval [−1, 2].

In [172]:
```
n = 3 # antall noder
a, b, c, d = -1, 2, -5, 5

# Intevall 1, [-1,2]
x_cheb1 = chebFinder(a,b,n)
p_cheb1 = Interpol(x_cheb1, f, n-1)

# Intevall 2, [-5,5]
x_cheb2 = chebFinder(c,d,n)
p_cheb2 = Interpol(x_cheb2, f, n-1)

error1 = np.abs(np.polyval(p_cheb1, x1) - f(x1))
error2 = np.abs(np.polyval(p_cheb2, x2) - f(x2))

print("Max error on [-1,2] =", np.max(error1).round(3))
print("Max error on [-5,5] =", np.max(error2).round(3)) # Veldig høyt siden f(x),  x-> -5 divergerer voldsomt



# Plotter f(x) og p(x) på intervallet [-1,2]
plt.plot(x1, f(x1), label='f(x)', color = 'r')
plt.plot(x1, np.polyval(p_cheb1, x1), label='p_cheb1(x)', color = 'b') # Her har jeg valgt 3. grad
# Høyere grad => mer nøyaktig
plt.legend()
plt.show()
```

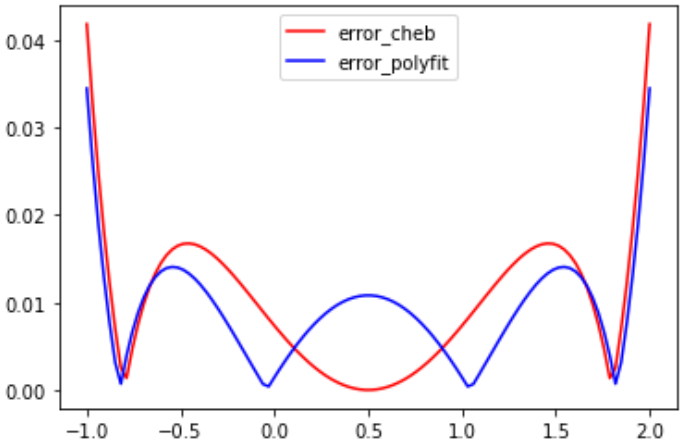
```
Max error on [-1,2] = 0.042
Max error on [-5,5] = 66419269.327
```



f)

Plot the error as a function in x, for both the interpolation in a) and the interpolation in c), on the interval [−1, 2]. Plot both errors in the same plot.

In [175]:
```
plt.plot(x1, error1, label='error_cheb', color = 'r')
plt.plot(x1, e_x1, label='error_polyfit', color = 'b')
plt.legend()
plt.show()
```



Ser nå at det kanskje var meningen vi skulle implementere lagrange interpolering i python for 2c)...

Står aldri spesifikt noe sted da.

Oppgave 3:

In [ ]:
```python
# THIS FUNCTION HAS TO BE COMPLETED CREATING THE CARDINAL FUNCTIONS
def cardinal(xdata, x):
    """
    cardinal(xdata, x):
    In: xdata, array with the nodes x_i.
        x, array or a scalar of values in which the cardinal functions are evaluated.
    Return: l: a list of arrays of the cardinal functions evaluated in x.
    """
    n = len(xdata)              # Number of evaluation points x
    l = []
    for i in range(n):          # Loop over the cardinal functions
        # ADD CODE HERE
        for j in range(n):      # Loop to make the product for l_i
            if i is not j:
                # ADD CODE HERE
        l.append(li)            # Append the array to the list
    return l
```

In [ ]:
```python
def lagrange(ydata, l):
    """
    lagrange(ydata, l):
    In: ydata, array of the y-values of the interpolation points.
         l, a list of the cardinal functions, given by `cardinal(xdata, x)`
    Return: An array with the interpolation polynomial (evaluated at `x`).
    """
    poly = 0
    for i in range(len(ydata)):
        poly = poly + ydata[i]*l[i]
    return poly
```

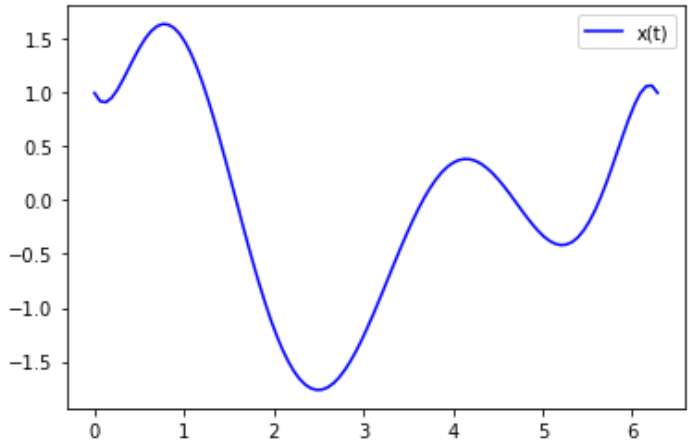
a) Interpolating $x(t)$

Hadde vært MYE lettere å bare hardkode alle 7 li - polynomene og regne ut..

-> Tok noen timer med stanging i veggen

Fra b) sin oppgavetekst kan det godt hende vi skal hardkode, vet ikke.

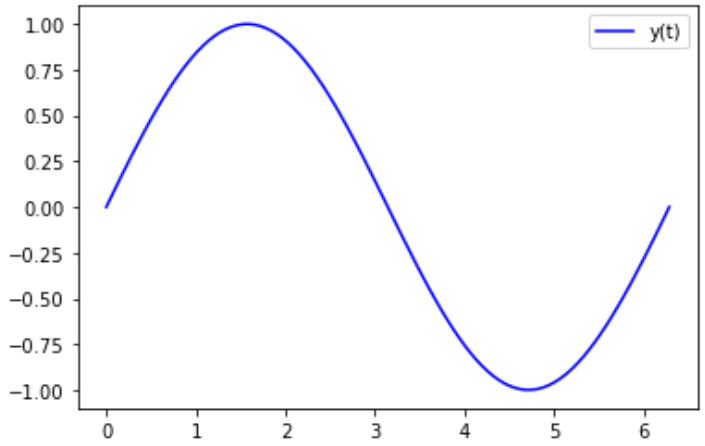
In [228]:
```
# Interpolation data
tdata = np.array([0, 0.8976, 1.7952, 2.6928, 3.5904, 4.4880, 5.3856, 6.2832])
xdata = np.array([1, 1.5984, -0.6564, -1.6828, -0.1191, 0.2114, -0.3514, 1])

# Grid points for plotting
t = np.linspace(0,tdata[-1],100)




def lagrange(data1, data2, x_koord):

    y_interpolert = np.array([])

    for i in x_koord:
        yp = 0
        for xi,yi in zip(data1,data2):
            yp += yi*np.prod((i-data1[data1 != xi])/(xi-data1[data1 != xi]))
        y_interpolert = np.append(y_interpolert, yp)

    return y_interpolert

px = lagrange(tdata, xdata,t)


# and plot it here ...
plt.plot(t, px, label='x(t)', color = 'b')
plt.legend()
plt.show()
```



b) Interpolating $y(t)$

Denne ble veldig grei :)

In [230]:
```
# Now do the same for the y coordinate...
ydata = np.array([0, 0.7818, 0.9750, 0.4339, -0.4339, -0.975, -0.7818, 0])

# the final polynomial should be called
py = lagrange(tdata, ydata,t)

# and plot it here ...

plt.plot(t, py, label='y(t)', color = 'b')
plt.legend()
plt.show()

#Hint: you can re-use the "li" already computed, since the cardinal functions depend only on tdata, not on xdata!
```

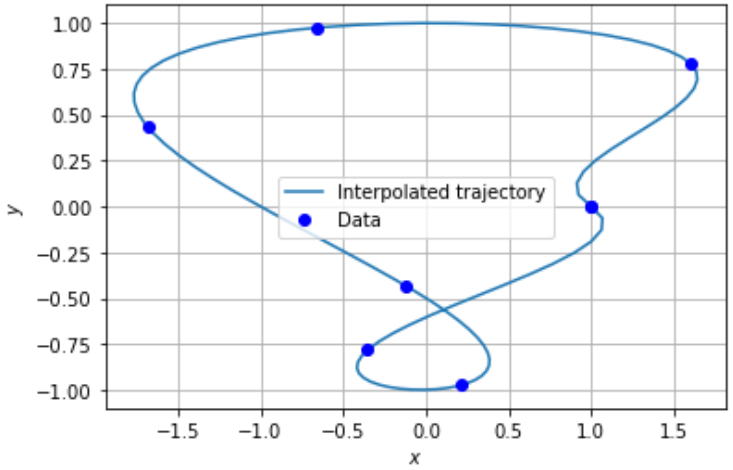


c) Trajectory

The trajectory of a particle moving in $\mathbb{R}^2$ is the locus of all (x, y) points traveled by the particle. Therefore, all we have to do is plot the interpolation of x against the interpolation of y.

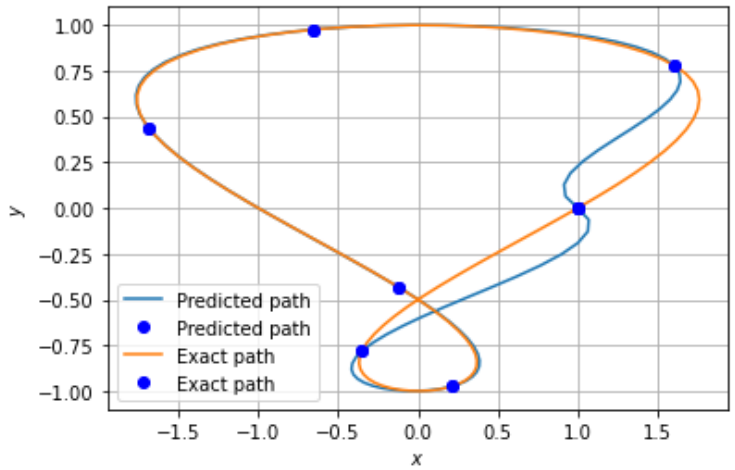
In [231]:

```
# If your polynomial in x is called px, Create your plot here
plt.plot(px, py, xdata, ydata,'ob')
plt.legend(['Interpolated trajectory', 'Data'])
plt.xlabel('$x$')
plt.ylabel('$y$')
plt.grid(True)

# Eneste jeg gjorde var ctrl + enter:)
```



d) Find the exact values and compare

In [240]:
```
# Compute the exact values

xdata_exact = np.cos(t) + np.sin(2*t)
ydata_exact = np.sin(t)

# Plot here both trajectories

plt.plot(px, py, xdata, ydata,'ob', label='Predicted path')
plt.plot(xdata_exact, ydata_exact, xdata, ydata,'ob', label='Exact path')
plt.xlabel('$x$')
plt.ylabel('$y$')
plt.grid(True)
plt.legend()
plt.show()
```



Ikke sikker hva jeg skal si, predicted ser ut til å divergere litt fra eksakt når x = [-0.4, 1.7] og y = [-0.75, 0.75]