Eksamensgjennomgang 28 November 2022

Notebooken inneholder kode for oppgavene som ble løst i timen.

På grunn av en teknisk feil ble det ikke gjort opptak av første halvdel. Opptak for andre halvdel er lagt ut, samt et ekstra opptak av gjennomgang av oppgave 14 og 15, som vi ikke rakk i timen.

Hvis du ønsker å kjøre koden som en notebook trenger du følgende filer for at den skal virke: start.txt, dna.txt, odesolver.py. Disse kan lastes ned fra samme sted som notebooken, og må lagres i samme katalog.

Oppgave 1

In [24]:
```
n = 0
k = 10

while n < k:
    n = n + 2
    k = k + 1

print(n)
```

```
20
```

Oppgave 2

In [25]:
```
x = [[-1],[1], 1, 2, 0]

print(x[-1])
print(x[x[-1]])
print(x[x[2]-1])
print(x[x[2]:x[3]])
print(x[2:3])
print(x[1:2])
print(x[x[0][0]])
```

```
0
[-1]
[-1]
[[1]]
[1]
[[1]]
0
```

Oppgave 3

In [26]:
```
def diff2nd(f, x, h=1E-6):
    r = (f(x-h) - 2*f(x) + f(x+h))/(h*h)
    return r

for k in range(1,8):
    h = 10**(-k)
    d2g = diff2nd(lambda x: 2*x**3-3*x, 1.5, h) #Missing line goes here
    print(f'h={h}: {d2g}', 12*1.5)
```

```
h=0.1: 18.000000000000057 18.0
h=0.01: 17.999999999993577 18.0
h=0.001: 17.999999999851468 18.0
h=0.0001: 17.999999979423364 18.0
h=1e-05: 18.000001489326674 18.0
h=1e-06: 17.999823853642738 18.0
h=1e-07: 17.941204077942533 18.0
```

Oppgave 4

In [27]:
```
class F:
    def __init__(self, alpha, beta):
        self.alpha = alpha
        self.beta = beta

    def __call__(self, x):
        return self.alpha + self.beta * x

    def __add__(self, obj):
        return F(self.alpha+obj.alpha, self.beta+obj.beta)

a = F(0,1)
b = F(2,3)
c = (a + b)(2)

print(c)
```

```
10
```

Oppgave 5

In [28]:
```
print(True or False and True)
print(True and (not False or True))
print(not (False and True))
print([e for e in [False,True] if not e][-1])
print(not ((2==3) and not(2==4)))
```

```
True
True
True
False
True
```

Oppgave 6

In [29]:
```
print('Q6:')
x = [-5, -2, -6, -3, -8, -3, -5, -16]

# Alternative A
x = [-5, -2, -6, -3, -8, -3, -5, -16]
y = -1
for e in x:
    if e > y:
        y = e
print(y,max(x))

# Alternative B
x = [-5, -2, -6, -3, -8, -3, -5, -16]
y = x[-1]
for i in range(len(x)):
   y = max(y,x[i])
```

```
print(y,max(x))

# Alternative C
x = [-5, -2, -6, -3, -8, -3, -5, -16]
for i in range(len(x)-1):
   if x[i+1] < x[i]:
      x[i+1] = x[i]
y = x[-1]

print(y,max(x))

# Alternative D
x = [-5, -2, -6, -3, -8, -3, -5, -16]
v = [x[i] for i in range(1, len(x)) if x[i] > x[i-1]]
y = v[-1]
print(y,max(x))

# Alternative E
x = [-5, -2, -6, -3, -8, -3, -5, -16]
import numpy as np
v = np.asarray(x)
y = -min(-v)
print(y,max(x))

# Alternative F
x = [-5, -2, -6, -3, -8, -3, -5, -16]
v = [x[0]]
for i in range(len(x)):
   if v[-1] <= x[i]:
      v.append(x[i])
y = v[-1]
print(y,max(x))

# Alternative G
x = [-5, -2, -6, -3, -8, -3, -5, -16]
s = f"y=max({x})"
exec(s)
print(y,max(x))
```

```
Q6:
-1 -2
-2 -2
-2 -2
-3 -2
-2 -2
-2 -2
-2 -2
```

Oppgave 7

In [30]:

```
import sys
sys.argv = ['check.py', 4, 0]

try:
    #One line is missing here
    #j = sys.argv[1]
    #j, k = sys.argv[2], sys.argv[3]
    #j = float(sys.argv[0])
    #h = 1/int(sys.argv[2])
    k = sys.argv[:3]
except IndexError:
    print("Feil A")
    sys.exit()
```

```
except ValueError:
    print("Feil B")
    sys.exit()
except ZeroDivisionError:
    print("Feil C")
    sys.exit()
```

Oppgave 8

In [31]:
```
def midpoint(f, x, dx=1e-4):
    return (f(x+dx)-f(x-dx))/(2*dx)

import math
y = midpoint(math.sin, math.pi/4)
print(y)
```

```
0.7071067800074049
```

Oppgave 9:

In [32]:
```
from math import factorial, asin

print('a:')
def arcsin_approx(x,N):
    s = 0
    for n in range(N+1):
        s += factorial(2*n)/(2**(2*n)*factorial(n)**2) * x**(2*n+1)/(2*n+1)
    return s

print('b:')
x = 0.5
exact = asin(x)
for N in range(1,21):
    approx = arcsin_approx(x,N)
    print(f'{N} {approx} {abs(approx-exact)}')
```

```
a:
b:
1 0.5208333333333334 0.0027654422649654453
2 0.5231770833333333 0.0004216922649654675
3 0.523525855654762 7.291994353686437e-05
4 0.523585195390005 1.3580208293784146e-05
5 0.5235961192958112 2.65630248763582e-06
6 0.523598237553187 5.380451117975582e-07
7 0.5235986637263972 1.1187190163131078e-07
8 0.5235987518596438 2.3738655041682932e-08
9 0.5235987704784364 5.119862422908739e-09
10 0.5235987744792603 1.1190385285075877e-09
11 0.5235987753509813 2.4731749981299345e-10
12 0.5235987755431232 5.5175641833216105e-11
13 0.5235987755858897 1.2409073768537837e-11
14 0.5235987755954885 2.8103075422336588e-12
15 0.5235987755976586 6.402656183013278e-13
16 0.5235987755981523 1.4654943925052066e-13
17 0.5235987755982652 3.3639757646142243e-14
18 0.5235987755982912 7.66053886991358e-15
19 0.5235987755982972 1.6653345369377348e-15
20 0.5235987755982986 2.220446049250313e-16
```

Oppgave 10

In [33]:
```
class RandomGenerator:

    def __init__(self,seed,m,a,b):
        self.seed,self.m,self.a,self.b = seed,m,a,b

    def __call__(self, n):
        res = []
        x = self.seed
        for i in range(n):
            x = (self.a * x + self.b) % self.m
            res.append(x/(self.m-1))
        return res

    def __str__(self):
        return f"seed={self.seed}, m={self.m}, a={self.a}, b={self.b}"



rand = RandomGenerator(seed=0.01, m=2**16+1, a=75, b=74)
x = rand(50)   # Skal generere en liste med 50 tilfeldige tall
print(x)
print(rand)
```

```
[0.001140594482421875, 0.08667373657226562, 0.5015678405761719, 0.6181526184
082031, 0.3618736267089844, 0.14123916625976562, 0.5939140319824219, 0.54401
01623535156, 0.8012809753417969, 0.09628677368164062, 0.22253036499023438,
0.6906623840332031, 0.8000297546386719, 0.002445220947265625, 0.184520721435
54688, 0.8399848937988281, 0.9990501403808594, 0.9287605285644531, 0.6571159
362792969, 0.2840766906738281, 0.3065605163574219, 0.9928321838378906, 0.462
4137878417969, 0.6816444396972656, 0.12368392944335938, 0.2772865295410156,
0.7973136901855469, 0.7987556457519531, 0.9069023132324219, 0.01776504516601
5625, 0.3334922790527344, 0.012668609619140625, 0.9512748718261719, 0.345661
1633300781, 0.9253349304199219, 0.4001960754394531, 0.015377044677734375, 0.
15439224243164062, 0.5803794860839844, 0.5289344787597656, 0.670619964599609
4, 0.2968635559082031, 0.2655601501464844, 0.9178504943847656, 0.83887863159
17969, 0.9160804748535156, 0.7061271667480469, 0.9598731994628906, 0.9905357
360839844, 0.2901802062988281]
seed=0.01, m=65537, a=75, b=74
```

Oppgave 11 a)

In [34]:
```
def read_DNA(filename):
    dna = ''
    with open(filename) as infile:
        for line in infile:
            words = line.split()
            dna = dna+words[-1]
    return dna


dna0 = read_DNA('dna.txt')
print(dna0)
```

```
CGTACGCCGACGTACGCCGACGCCGACGTACGCGCCGACGGTACGCGCCGACGGTACGCG
```

Oppgave 11 b)

In [35]:
```
def find_CG(dna):
    cg = []
    for i in range(len(dna)-1):
        if dna[i:i+2] == 'CG':
            cg.append(i)
```

```
    return cg

print(find_CG(dna0))
```

```
[0, 4, 7, 10, 14, 17, 20, 23, 26, 30, 32, 35, 38, 43, 45, 48, 51, 56, 58]
```

Oppgave 11 c)

In [36]:
```
def test_find_CG():
    dna = "TATACG"
    expected = [4]
    computed = find_CG(dna)
    assert expected == computed
    dna = "CGCGCG"
    expected = [0,2,4]
    computed = find_CG(dna)
    assert expected == computed

test_find_CG()

"""
Below is a more compact version of the test function. Both
of these would be given full score on the exam.
"""

"""
def test_find_CG():
    dna0 = "TATACG"
    dna1 = "CGCGCG"
    assert find_CG(dna0) == [4] and find_CG(dna1) == [0,2,4]
"""

test_find_CG()
```

Oppgave 11 d)

In [37]:
```
def longest_CG_repetition(dna):
    pos = find_CG(dna)
    res = count = 1
    for k in range(1,len(pos)):
        if pos[k] == pos[k-1] + 2:
            count += 1
            res = max(count,res)
        else:
            count = 1
    return res


#Call the function on some simple strings to see if it works:
print(longest_CG_repetition("CGCGCG"))
print(longest_CG_repetition("CGATCGCGCGCGGGGG"))
print(longest_CG_repetition("CGAACGCGCG"))
```

```
3
4
3
```

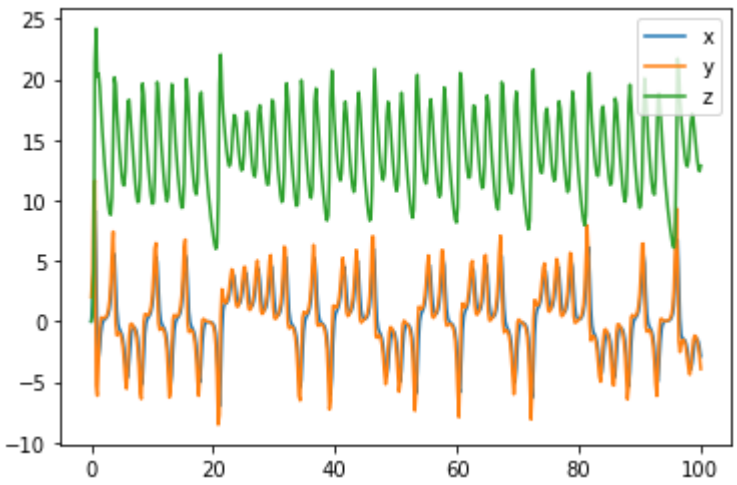
Oppgave 13 a)

In [38]:
```
import numpy as np
import matplotlib.pyplot as plt
```

```
from odesolver import *

def Lorenz(x0, y0, z0, a, b, c, T, N):
    def rhs(u,t):
        x, y, z = u
        dx = a*(y-x)
        dy = b*x-y-x*z
        dz = x*y-c*z
        return [dx,dy,dz]

    solver = RungeKutta4(rhs)
    solver.set_initial_condition([x0,y0,z0])
    t = np.linspace(0,T,N)
    u, time = solver.solve(t)
    return t, u[:,0], u[:,1], u[:,2]
```

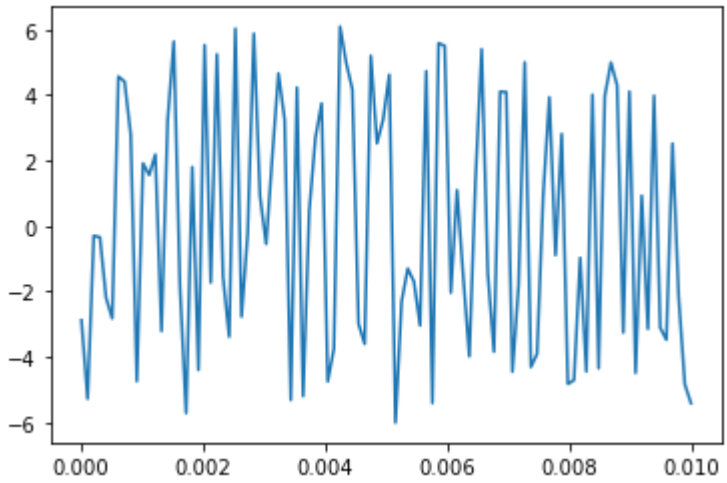
Oppgave 13 b)

In [39]:
```
t,x,y,z = Lorenz(0,2,0,a=4,b=16,c=0.5,T=100, N=500)

plt.plot(t,x,label='x')
plt.plot(t,y,label='y')
plt.plot(t,z,label='z')
plt.legend()
plt.show()
```



Oppgave 13 c)

In [40]:
```
x_init = np.linspace(0,0.01,100)
x_end = []

for x0 in x_init:
    t,x,y,z = Lorenz(x0,2,0,a=4,b=16,c=0.5,T=100,N=500)
    x_end.append(x[-1])

plt.plot(x_init,x_end)
plt.show()
```



Oppgave 14

In [52]:

```
class Game:
    def __init__(self):
        with open('start.txt') as infile:
            N = int(infile.readline())
            self.board = np.zeros((N,N), 'int')
            for line in infile:
                words = line.split(',')
                a,b,c = int(words[0]),int(words[1]),int(words[2])
                self.board[b,c] = a


    def move1(self,row0,col0,row1,col1):
        #her var det en bug i koden i opptaket (feil parenteser):
        crit1 = self.board[row0,col0] == 1 and self.board[row1,col1] == 0
        crit2 = abs(row1-row0) + abs(col1-col0) == 1
        crit3 = 0 <= row1 < N and 0 <= col1 < N
        if crit1 and crit2 and crit3:
            self.board[row0,col0] = 0 #bug i opptaket, fikset her
            self.board[row1,col1] = 1


    def move2(self,row0,col0,row1,col1):
        #her var det en bug i koden i opptaket (feil parenteser):
        crit1 = self.board[row0,col0] == 2 and self.board[row1,col1] == 0
        crit2a = abs(row1-row0) == 1 and abs(col1-col0) == 2
        crit2b = abs(row1-row0) == 2 and abs(col1-col0) == 1
        crit3 = 0 <= row1 < N and 0 <= col1 < N
        if crit1 and (crit2a or crit2b) and crit3:
            self.board[row0,col0] = 0 #bug i opptaket, fikset her
            self.board[row1,col1] = 2


game = Game()
print(game.board)
game.move1(1,2,1,1)
print(game.board)
game.move2(0,1,1,1) #illegal move, nothing changes
print(game.board)
game.move2(0,1,2,2)
print(game.board)
```

```
[[0 2 0]
 [0 0 1]
 [0 1 0]]
[[0 2 0]
 [0 1 0]
 [0 1 0]]
[[0 2 0]
 [0 1 0]
 [0 1 0]]
[[0 0 0]
 [0 1 0]
 [0 1 2]]
```

Oppgave 15 a)

In [53]:
```python
def Bspline(t,p,x):
    if t < x[p] or t> x[p+1]:
        return 0
    elif x[p] < t < x[p+1]:
        return 1/(x[p+1]-x[p])
    else:
        return 0.5/(x[p+1]-x[p])
```

Oppgave 15 b)

In [54]:
```python
"""
Here are some alternative solutions to question 15. The one
based on recursion (also listed above) is the simplest and most elegant,
but recursion was not part of the IN1900 curriculum in 2021. The
question can also be solved with a loop, but this is more complicated
and makes this one of the most difficult questions on the exam.

In the exam setting, with limited time and no possibility to run the
code, it is very difficult to get all the indices correct in the loop
version. A good score is given to all solutions that get the overall loop
structure correct, even with some minor errors in indices etc.
"""

def Bspline(t,k,p,x):
    """
    For å beregne 1 Bspline-funksjon av grad k trenger vi 2 Bsplines av grad
    k-1, 3 av grad k-2, ..., k+1 av grad 0. Vi lagrer disse funksjonsverdien
    liste av dicts. Se tidligere eksamensoppgaver på emnesiden for andre
    alternative løsninger.
    """

    if k == 0:
        return Bspline0(t,p,x)

    bsplines = []
    bsplines.append({p+k_: Bspline0(t,p+k_,x) for k_ in range(k+1)})
    for i in range(1,k+1):
        prev = bsplines[i-1]
        new = {}
        for j in range(k+1-i):
            a = (t-x[p+j])*prev[p+j]+(x[p+i+1+j]-t)*prev[p+j+1]
            b = x[p+i+1+j]-x[p+j]
            new[p+j] = a/b

    return bsplines[k][p]
```

```
def Bspline(t,k,p,x):
    """Enkleste løsning: rekursjon (ikke pensum IN1900)"""

    if k == 0:
        return Bspline0(t,p,x)

    elif k > 0:
        a = (t-x[p])*Bspline(t,k-1,p,x) + (x[p+k+1]-t)*Bspline(t,k-1,p+1,x)
        b = x[p+k+1]-x[p]
    return a/b
```

In [ ]: