Forside

UNIVERSITY OF OSLO
Faculty of mathematics and natural sciences
Exam in: IN1900 og INF1100
Exam date: 9. december 2019
Time for exam: 4 timer.
Attachments: ODESolver.pdf in problem 3.3
Permitted aids: None.
A calculator is available in Inspera.

Read the entire exam set before you start answering the questions. The exam contains multiple choice questions, and text questions where you shall write short programs or read programs and write the output from the program. If you are missing information you can make your own reasonable assumptions, as long as they are in line with the "nature" of the question. In text questions you should then specify the assumptions you made, for instance in comments to the code.

All code in the question texts is written in Python 3. You can write your answers in either Python 2 or Python 3, but you should avoid using a mix.

Most of the questions lead to short code with little need for comments, unless you do something complicated or non-standard. (which is not recommended; but in this case the comments shall explain the ideas behind the program to make it easier to evaluate the code).

A question may ask you to write a function. A main program which calls the function is in this case not needed, unless it is specifically asked for in the question text.

1.1 Hva skrives ut?

What is printed in the terminal when the following code is run?

```
u = 0
for i in range(1,5,2):
    u += i
print(u)
```

Select one alternative:

- 4 ✓
- 2
- 9
- An error message

Maximum marks: 1

1.2 Hva skrives ut?

What is printed in the terminal when the following code is run?

```
a = []
for i in range(4):
    a.append([i,i+1])
print(a[-1])
```

Select one alternative:

- ○ 4
- ○ An error message
- ○ 5
- ○ [3,4] ✓
- ○ [4,5]

Maximum marks: 1

1.3 Hva skrives ut?

What is printed when the following code is run?

```
for i in range(1,4):
    print(i,end=' ')
    for j in range(i):
        print(j,end=' ')
```

Select one alternative:

- ○ 1 0 2 0 1 3 0 1 2 ✓
- ○ 0 1 1 0 2 0 1
- ○ 0 1 0 1 2 0 1 2 3
- ○ An error message

The argument "**end = ' '** " to the print function makes each write to the screen end with a space (' ') instead of a linebreak.

Maximum marks: 1

1.4 Hva skrives ut?

What is printed when the following code is run?

```
def absolute(x):
    if x < 0:
        return -x
    return x

for x_ in [-3,4,0]:
    print(absolute(x_), end=' ')
```

Select one alternative:

- 3 -4 0
- An error message
- 3 4 0 ✓
- 3 -3 4 0

Maximum marks: 1

1.5 Hva skrives ut?

What is printed when the following code is run?

```
def freq_lists(dna_list):
   n = len(dna_list[0])
   A = [0]*n
   T = [0]*n
   G = [0]*n
   C = [0]*n
   for dna in dna_list:
      index = 0
      for base in dna:
         if base == 'A':
            A[index] += 1
         elif base == 'C':
            C[index] += 1
         elif base == 'G':
            G[index] += 1
         elif base == 'T':
            T[index] += 1
         index += 1
   return A, C, G, T
dna_list = ['TCGCT', 'GGACT', 'GCTGC']
A, C, G, T = freq_lists(dna_list)
print(G)
```

Select one alternative:

- [2, 1, 1, 1, 0] ✓
- ['G', 'G', 'G', 'G', 0]
- [0, 1, 2, 1, 0]
- 5
- ['T', 'C','G','T']

Maximum marks: 2

1.6 Hvilken påstand er riktig?

One of the following statements is correct. Which one?

Select one alternative:

- A test function returns 0 if the test passes
- A test function must always include a return statement
- A test function should always take at least one input argument
- A test function can have multiple assert statements ✓

Maximum marks: 1

1.7 Hva skrives ut?

What is printed in the terminal when the following code is run?

```
def count(dna, base):
  i = 0
  for j in range(len(dna)):
    if dna[j] == base:
      i += 1
  return i

def test_count():
  dna = 'ATTTGCGGTCCAAA'
  success = count(dna, 'A') == 4
  msg = 'count returns the wrong number'
  assert success, msg

test_count()
```

Select one alternative:

- count returns the wrong number
- Nothing is printed ✓
- AssertionError: count returns the wrong number
- Success

Maximum marks: 1

1.8 Hvilket funksjonskall?

The function **euler(rhs,u0,T,n)** applies Euler's method to solve an ordinary differential equation (ODE) with right hand side defined by the function rhs and initial condition u0, for the time interval 0 ti T, with n time steps:

```
import numpy as np

def euler(rhs,u0,T,n=100):
   t = np.linspace(0,T,n+1)
   dt = T/n
   u = np.zeros_like(t)
   u[0] = u0
   for i in range(1,n+1):
      u[i] = u[i-1]+dt*rhs(u[i-1],t[i-1])
   return u,t
```

We want to use the function to solve the differential equation y' = -0.5 y, y(0) = 1, for t in the interval 0 til 5. Which function call is correct?

Select one alternative:

- u, t = euler(f = -0.5*y, 1.0, 5)
- u, t = euler(lambda y: -0.5*y, 1.0, 5)
- u, t = euler(lambda y, t: -0.5*y, 1.0, 5) ✓
- u, t = euler(-0.5*y, 1.0, 5)

Recall that a lambda function is a compact way to define a function. For instance, the following line will define a function that returns x^2+y^2:

```
func = lambda x,y: x**2 + y**2
```

This line is equivalent to the following code:

```
def func(x,y):
   return x**2 + y**2
```

Maximum marks: 2

1.9 Hvilket funksjonskall?

The following function implements a bisection method for finding solutions of the equation f(x)=0 in the interval [a,b].

```
def bisection(f, a, b, eps=1e-5):
  fa = f(a)
  if fa*f(b) > 0:
    print(f'No unique root in [{a},{b}]')
    return None
  while b-a > eps:
    m = (a + b)/2.0
    fm = f(m)
    if fa*fm <= 0:
      b = m  # root is in left half of [a,b]
    else:
      a = m  # root is in right half of [a,b]
      fa = fm
  return m
```

We want to use the function to find a solution of the equation

$$x^3 + 2x - 1 = 0$$

in the interval [-10,10]. Which function call is correct?

Select one alternative:

- x = bisection(lambda x: x**3+2*x-1,-10,10) ✓
- x = bisection(f(x) = x**3+2*x-1,-10,10, eps=1e-5)
- x = bisection(eval('x**3+2*x-1'),-10,10)
- x = bisection(x**3+2*x-1,-10,10, eps=1e-5)

Recall that a lambda function is a compact way to define a function. For instance, the following line will define a function that returns x^2+y^2:

```
func = lambda x,y: x**2 + y**2
```

This line is equivalent to the following code:

```
def func(x,y):
  return x**2 + y**2
```

Maximum marks: 2

1.10 Hvor feiler koden?

In which line will this code stop and write an error message?

```
a = [[4,5],[0,1],2,[2,0.5],4,3,[5,6,7]]
b =[]
for e in a:
  s = 0
  for number in e:
    s += number
  b.append(s)
print(b)
```

```
print(b)
```

Select one alternative:

- for e in a:
- a = [[4,5],[0,1],2,[2,0.5],4,3,[5,6,7]]
- b.append(s)
- s += number
- for number in e: ✓

Maximum marks: 2

1.11 Hva skrives ut?

The file formula_cml.py contains the following code:

```
import sys
from math import *
try:
   formula = sys.argv[1]
   x = [float(x_) for x_ in sys.argv[2:]]
except IndexError:
   print('Missing command line argument')
   exit()

code = f"""
def f(x):
   return {formula}
"""

try:
   exec(code)
except:
   print('Something wrong in formula')
   exit()

for x_ in x:
   print(f(x_),end=' ')
```

What is printed when the program is run in the following way?

Terminal> python formula_cml.py 2*x2-3 1 3**

Fill in your answer here

Format B I U x₂ x² Tx

Words: 0

Maximum marks: 2

1.12 Hva skrives ut?

The file stars.txt contains the following:

```
Name              distance    brightness   luminosity
Alpha_Centauri_A    4.3         0.26         1.56
Alpha_Centauri_B    4.3        0.077        0.45
Alpha_Centauri_C    4.2        0.00001      0.00006
Sirius_A            8.6         1.00         23.6
```

There are no blank lines in the file.

What is printed when the following code is run?

```
stars_data = {}

with open('stars.txt') as infile:
   infile.readline()

   for line in infile:
      w = line.split()
      data = {'dist':    w[1], 'bright': w[2],'lum': w[3]}
      stars_data[w[0]] = data

print(len(stars_data),stars_data['Sirius_A']['bright'])
```

Fill in your answer here

Format

Words: 0

Maximum marks: 2

2.1 Funksjon av to variable

Write a python function **f(x,y)** that returns the value of the mathematical expression
$f(x,y) = 4x^3y - 2xy$
Fill in your answer here

```
 
```

Maximum marks: 3

2.2 Numerisk derivasjon

The derivative of a mathematical function f(x) can be approximated with the midpoint formula

$$f'(x) \approx \frac{f(x+h)-f(x-h)}{2h}$$

for a small number h.

Write a Python function **midpoint(f,x,h),** which uses this formula to estimate the derivative of a function f in the point x. The function shall return the function value f(x) and the estimated derivative. The argument f can be an arbitrary mathematical function implemented as a Python function, which takes one input argument and returns one value.

Include a line where you call the function to estimate the derivative of cos(x) in the point x=0, for h=0.001.

Fill in your answer here

Maximum marks: 5

2.3 Implementasjon av en sum

Write a Python function **cos_approx(x,n)** which computes the sum
$f(x) = \sum_{k=0}^{n} (-1)^k \frac{x^{2k}}{2k!}$
and returns the value.

x can be a decimal number (float) or a Numpy-array of decimal numbers, while n is a positive integer. Recall that k! is the factorial of k. Include necessary imports.

Fill in your answer here

Maximum marks: 5

2.4 Lesing av fil

The file constants.txt has the following contents:

```
name of constant          value               dimension
--------------------------------------------------------
light speed               299792458.0         m/s
gravitational constant    6.67259e-11         m**3/kg/s**2
Planck constant           6.6260755e-34       J*s
elementary charge         1.60217733e-19      C
Avogadro number           6.0221367e23        1/mol
Boltzmann constant        1.380658e-23        J/K
electron mass             9.1093897e-31       kg
proton mass               1.6726231e-27       kg
```

The file contains no blank lines.

Write a Python program that reads this file and stores its contents in a dictionary. The keys of the dictionary shall be the name of the constant (from the column "name of constant"), and the value shall be a list or tuple of length two that contains the numeric value of the constant (column "value") and its physical units (column "dimension").

Hint: The Python method **join** is used for joining a list of strings into a single string. As an example, the code

string_list = ['Hello','world']

hello = ' '.join(string_list)

will result in a string hello having the value "Hello world".

Fill in your answer here

Maximum marks: 6

2.5 Dictionary og sum

A polynomial p(x) can be represented as a dictionary, so that the keys of the polynomial are the exponents and the values are the coefficients in front of each term. For instance, the polynomial $p(x) = 1 - 2x^2 + 3x^4 + x^5$ can be represented as the dictionary
p = {0:1,2:-2,4:3,5:1}

Write a Python function **poly_eval(p,x)**, which evalueates such a polynomial p for a given x, and returns its value.

Fill in your answer here

```
 
```

Maximum marks: 5

2.6 Derivasjon av polynom

A general polynomial can be written on the form
$p(x) = \sum_{j=0}^{n} c_j x^j$
where c_j are constant coefficients.

The derivative of such a polynomial is given by
$\frac{dp}{dx} = \sum_{j=1}^{n} j c_j x^{j-1}$

Write a Python function **poly_diff(p)** that calculates the derivative of a general polynomial p. The argument p is a dictionary representation of a polynomial, as defined in the previous question. The function shall return a dictionary representing the derivative of p.

Fill in your answer here

Maximum marks: 5

3.1 Klasse for en funksjon

Write a Python class **F** which implements the function
$f(x; a, b, c) = ax^2 + bx + c$
The parameters a, b, and c shall be attributes, and the class shall be possible to use in the following way

f = F(a=1.0, b=2.0, c=0.0)
x = 2.0
print(f(x))   # prints the function value (8.0)

Fill in your answer here

Maximum marks: 5

3.2 ODE-løser, funksjon

Write a Python function **heun3(f, U0, T, n)**, which uses Heun's 3. order method to solve an ordinary differential equation (ODE) given by:

$$u' = f(u), \qquad u(0) = u_0.$$

Heun's 3. order method is given by

$$u_{k+1} = u_k + \tfrac{1}{4}K_1 + \tfrac{3}{4}K_3$$
$$K_1 = \Delta t f(u_k, t_k)$$
$$K_2 = \Delta t f(u_k + \tfrac{1}{3}K_1, t_k + \tfrac{1}{3}\Delta t)$$
$$K_3 = \Delta t f(u_k + \tfrac{2}{3}K_2, t_k + \tfrac{2}{3}\Delta t)$$

The arguments to the function shall be a callable function f, which defines the right hand side of the ODE, the initial condition U0, the end-time T, and the number of time steps n. The function shall return two numpy-arrays u and t, where u contains the solution and t contains the time points where we have approximated the solution. In this question you can assume that we solve a scalar ODE, where the solution has only one component. The solution array u should therefore be a one-dimensional array. Include necessary imports.

Fill in your answer here

Maximum marks: 5

3.3 ODESolver, arv

Implement the method from the previous question (Heun's 3. order method) as a sub class **Heun3(ODESolver)**. The base class ODESolver is defined in the attached file. Use inheritance to reuse as much code as possible from the base class. The class **Heun3** must support the following use:

```
from numpy import *
from ODESolver import *
rhs = lambda u,t: -0.5*u
solver = Heun3(rhs)
solver.set_initial_condition(1.0)
time = numpy.linspace(0,10,101)
u, t = solver.solve(time)
```

Fill in your answer here

Maximum marks: 5

3.4 Logistisk vekst

The model for logistic growth is defined by the following ordinary differential equation (ODE):

$$\frac{du}{dt} = au(1 - \frac{u}{R})$$

The model describes growth of a population in an environment with limited resources, where u is the size of the population, t is time, and *a* and *R* are constant model parameters. Write a Python function **rhs(u,t)** that defines the right hand side of the equation. The parameters can be local variables in the function, with values *a* = 1 and *R* = 50.

Write code for solving the equation with the Heun3-klass from the previous question. The initial condition shall be u0 = 0.1, the time interval from t=0 to t=20, and you shall use 201 time steps including the end points (dt = 0.1).

Fill in your answer here

Maximum marks: 5

3.5 SEIS-modell

This question presents a so-called SEIS-model for modeling infectious diseases. The model is a modification of the classical SIR-model, where the population is divided in three groups: those who can be infected (S), those who are infected but have not yet developed the disease, and cannot infect others (E), and those that are sick and can infect others (I). There is no immunity in the model, so those that recover from the disease return to the S-category. Let S(t), E(t), and I(t) be the number of people in each category (measured in millions). The following system of differential equations describes how S(t), E(t), and I(t) evolve over a time interval [0,T]:

$$S'(t) = -p(t)S(t)I(t) + rI(t),$$
$$E'(t) = p(t)S(t)I(t) - qE(t),$$
$$I'(t) = qE(t) - rI(t),$$

At time t=0 we have the intitial conditions S(0) = S0, E(0) = E0, I(0) = 0. The function p(t) and the constants q and r are assumed to be known. All constants and functions are >0.

Write a Python-function **SEIS(S0,E0,p,q,r,T)**, which takes initial values S0, E0, the function p(t), parameters q, r, and the end-time T as input arguments. The function shall solve the equations of the SEIS-model and return the solution. Use the Heun3 class from the previous question to solve the differential equations. Let the time be given in days. Use ten time steps per day, so that the total number of time steps for a simulation over the interval [0,T] is 10T+1. The function SEIS shall return 4 arrays:

t, which contains the time points tk where the numerical solution is calculated,

S, which contains S(0),S(t1), ... S(tn),

E, which contains E(0),E(t1), ... E(tn),

I, which contains I(0),I(t1), ... I(tn).

We want to solve the model with the following parameters; S0 = 4.0, E0 = 0.2, q = r = 0.1, and p(t) = 0.0233 (constant).

Write the code to call the function with the given parameters and T=100. Also include code to plot S(t), E(t) and I(t) in the same window, with a legend for each curve.

Fill in your answer here

Maximum marks: 8

Question 21
Attached

```python
import numpy as np

class ODESolver:
    """
    Superclass for numerical methods solving scalar and vector ODEs
      du/dt = f(u, t)

    Attributes:
    t: array of time values
    u: array of solution values (at time points t)
    k: step number of the most recently computed solution
    f: callable object implementing f(u, t)
    """
    def __init__(self, f):
        if not callable(f):
            raise TypeError('f is %s, not a function' % type(f))
        self.f = lambda u, t: np.asarray(f(u, t), float)

    def set_initial_condition(self, U0):
        if isinstance(U0, (float,int)):  # scalar ODE
            self.neq = 1
            U0 = float(U0)
        else:                            # system of ODEs
            U0 = np.asarray(U0)          # (assume U0 is sequence)
            self.neq = U0.size
        self.U0 = U0

    def advance(self):
        """Advance solution one time step."""
        raise NotImplementedError

    def solve(self, time_points):
        """
        Compute solution u for t values in the list/array
        """
        self.t = np.asarray(time_points)
        n = self.t.size
        if self.neq == 1:  # scalar ODEs
            self.u = np.zeros(n)
        else:              # systems of ODEs
            self.u = np.zeros((n,self.neq))

        # Assume that self.t[0] corresponds to self.U0
        self.u[0] = self.U0

        # Time loop
        for k in range(n-1):
            self.k = k
            self.u[k+1] = self.advance()

        return self.u, self.t


class ForwardEuler(ODESolver):
    def advance(self):
        u, f, k, t = self.u, self.f, self.k, self.t
        dt = t[k+1] - t[k]
        u_new = u[k] + dt*f(u[k], t[k])
        return u_new
```

IN1900 - solutions final exam fall 2019

Question 1.11

Correct answer:

```
-1 15
```

Question 1.12

Correct answer:

```
4 1.00
```

Question 2.1

Suggested solution:

```
def f(x,y):
    return 4*x**3*y-2*x*y
```

Question 2.2

Suggested solution:

```
from math import cos

def midpoint(f,x,h):
    return f(x), (f(x+h)-f(x-h))/(2*h)

d = midpoint(cos,x=0,h=0.001)
```

Question 2.3

Suggested solution:

```
def cos_approx(x,n):
    s = 0
    for i in range(n+1):
        s += (-1)**i * x**(2*i)/(2*factorial(i))
    return s
```

The above code is a correct implementation of the formula given in the question. However, the formula contained a typo so it was not the correct Taylor series for the cosine function, which is indicated by the name of the function. An implementation of the correct formula reads

```
def cos_approx(x,n):
    s = 0
    for i in range(n+1):
        s += (-1)**i * x**(2*i)/factorial(2*i)
    return s
```

Both answers were given full score on the exam.

Question 2.4

Suggested solution:

```
constants = {}
with open('constants.txt') as infile:
    infile.readline()
    infile.readline()
    for line in infile:
        w = line.split()
        name = ' '.join(w[:2])
        constants[name] = (float(w[2]),w[3])
```

Question 2.5

Suggested solution:

```
def poly_eval(p,x):
    s = 0
    for i in p:
        s += p[i]*x**i
    return s
```

Question 2.6

Suggested solution:

```
def poly_diff(p):
    dp = {}
    for i in p:
        if i != 0:
            dp[i-1] = i*p[i]
    return dp
```

Question 3.1

Suggested solution:

```
class F:
    def __init__(self,a,b,c):
        self.a = a
        self.b = b
        self.c = c

    def __call__(self,x):
        return self.a*x**2+self.b*x+self.c
```

Question 3.2

Suggested solution:

```
def heun3(f,U0,T,n):
    t = np.linspace(0,T,n+1)
    u = np.zeros_like(t)
    u[0] = U0
    dt = T/n
    for i in range(n):
        k1 = dt*f(u[i],t[i])
        k2 = dt*f(u[i]+1/3*k1,t[i]+1/3*dt)
        k3 = dt*f([i]+2/3*k2,t[i]+2/3*dt)
        u[i+1] = u[i]+1/4*k1+3/4*k2

    return u, t
```

Question 3.3

Suggested solution:

```
    class Heun3(ODESolver):
        def advance(self):
            u,f,t,k = self.u, self.f, self.t, self.k
            dt = t[k+1]-t[k]
            k1 = dt*f(u[k],t[k])
            k2 = dt*f(u[k]+1/3*k1,t[k]+1/3*dt)
            k3 = dt*f([k]+2/3*k2,t[k]+2/3*dt)
            return u[k]+1/4*k1+3/4*k2
```

Question 3.4

Suggested solution:

```
    def rhs(u,t):
        a = 1.0; R = 50.0
        return a*u*(1-u/R)

    from ODESolver import *

    solver = Heun3(rhs)
    solver.set_initial_condition(0.1)
    time = np.linspace(0,20,201)
    u,t = solver.solve(time)
```

Question 3.5

Suggested solution:

```
    def SEIS(S0,E0,p,q,r,T):
        def rhs(u,t):
            S,E,I = u
            dS = -p(t)*S*I+r*I
            dE = p(t)*S*I-q*E
            dI = q*E-r*I

            return [dS,dE,dI]

        U0 = [S0,E0,0]

        solver = ForwardEuler(rhs)
        solver.set_initial_condition([U0])
        time = np.linspace(0,100,501)
        solver.solve(time)
        return solver.t, solver.u[:,0], solver.u[:,1], solver.u[:,2]

    def p(t):
        return 0.0233
```

```
t, S, E, I = SEIS(4.0,0.2,p, 0.1, 0.1,100)

plt.plot(t,S,t,E,t,I)
plt.legend(['S','E','I'])
plt.show()
```