

## i Forside

UNIVERSITY OF OSLO  
Faculty of mathematics and natural sciences  
Exam in: IN1900 og INF1100  
Exam date: 28. november 2018  
Time for exam: 4 timer.  
Attachments: 1 (ForwardEuler.pdf)  
Permitted aids: None.

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?

```
s = 0
for k in range(3,6,2):
    s += k

print(s)
```

Select one alternative:

- 8
- 0
- 6
- 3



---

Maximum marks: 1

### 1.2 Hva skrives ut?

What is printed when the following code is run?

```
a = [8,9,10,11]
b = a + [12,13]
```

```
for i in b[2:-1]:  
    print(i,end=' ')
```

Select one alternative:

- 9 10 11 12 13
- An error message
- 10 11 12 13
- 10 11 12
- 9 10 11 12



---

Maximum marks: 1

### 1.3 Hva skrives ut?

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

```
for i in range(2,5):  
    print(i,end=' ')  
    for j in range(i-1):  
        print(j, end=' ')
```

Select one alternative:

- 2 0 1 3 0 1 2 4 0 1 2 3
- 1 0 2 0 1 3 0 1 2
- 2 0 3 0 1 4 0 1 2
- 0 0 1 0 2 0 1 3 0 1 2 4 0 1 2 3



---

Maximum marks: 2

### 1.4 Hva skrives ut?

What is printed when the following code is run?

```
n = 5  
dt = 1/n  
time = []  
for k in range(n+1):  
    time.append(dt*k)
```

print(time)

Select one alternative:

- An error message
- [0.0, 0.2, 0.4, 0.6, 0.8, 1.0]
- [0.2 0.4 0.6 0.8 1.0]
- [0.0 0.2 0.4 0.6 0.8]



---

Maximum marks: 2

### 1.5 Hva skrives ut?

What is printed when the following code is run?

```
def heaviside(x):  
    if x < 0:  
        return 0  
    else:  
        return 1  
  
for x in [2, 5, 10]:  
    print(heaviside(x-5), end= ' ')
```

Select one alternative:

- 1 1 1
- An error message
- 0 1 1
- 0 0 0
- 0 0 1



---

Maximum marks: 2

### 1.6 Hva skrives ut?

The file "summer\_rain.txt" has the following content (no blank lines):

Average 19.1

June 21.0

July 15.5

August 17.5

What is printed in the terminal when the following code is run? If you believe that nothing is printed you should write this explicitly in your answer, for instance "Nothing is printed".

```
infile = open('summer_rain.txt', 'r')  
infile.readline()  
rain = 0  
months = []  
for line in infile:  
    months.append(line.split()[0])  
    rain += float(line.split()[-1])  
print('The total rainfall from %s to %s was %.2f' %(months[0],months[-1],rain))
```

Fill in your answer here

Format - | **B** *I* U  $\times_2$   $\times^2$  |  $I_x$  | | | |  $\Omega$  |  $\Sigma$  |

Words: 0

Maximum marks: 3

### 1.7 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:
        for index, base in enumerate(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
    return A, C, G, T
dna_list = ['GCTCT', 'GGTAC', 'GATGC']
A, C, G, T = freq_lists(dna_list)
print(C)
```

Select one alternative:

- [0, 2, 2, 0, 1]
- [0, 1, 0, 1, 2]
- 4
- [1,2,1]
- [2,1,1]



Maximum marks: 3

## 1.8 Hva skrives ut?

What is printed when the following code is run? If you believe that nothing is printed you should write this explicitly in your answer.

```
import sys
try:
    x = float(sys.argv[1])
    coeff = [float(s) for s in sys.argv[2:]]
except IndexError:
    print('You need to provide at least two command-line arguments.')
    sys.exit(1)
except ValueError:
    print('Cannot convert argument to float.')
    sys.exit(1)

poly_val = 0
for i in range(len(coeff)):
    poly_val += coeff[i]*x**i

print('The value of the polynomial is %g' % poly_val)
```

The code is in a file poly.py, and is run in the following way:

```
Terminal> python poly.py 1.0 3.5 4
```

Fill in your answer here

Format
B
I
U
x<sub>2</sub>
x<sup>2</sup>
I<sub>x</sub>
↵
↶
↷
↺
↻
☰
☷
Ω
☒
Σ
ABC
✖

Words: 0

Maximum marks: 3

## 1.9 Hva skrives ut?

What is printed when the following code is run? If you believe that nothing gets printed you should write this explicitly in your answer.

```
data = [
('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)]
```

```

stars = {}
for star in data:
    dist, bright, lumin = star[1:]
    star_dict = {'distance':dist, 'brightness':bright, 'luminosity':lumin}
    stars[star[0]] = star_dict
print(stars['Alpha Centauri A']['distance'], end = ' ')
print(stars['Sirius A']['luminosity'])

```

Fill in your answer here

Format
B
I
U
x<sub>2</sub>
x<sup>2</sup>
I<sub>x</sub>
📄
📄
↶
↷
🔄
☰
☷
Ω
📊
✎
Σ
ABC
✖

Words: 0

Maximum marks: 3

### 1.10 Hva skrives ut?

What is printed when the following code is run? If you believe that nothing gets printed you should write this explicitly in your answer.

```

def parabola(x,a,b,c):
    return a*x**2+b*x+c

def test_parabola():
    a, b, c = 1, 1, 1
    x = [0,1,2]
    tol = 1e-6
    expected = [1,3,7]
    msg = 'Something wrong'
    for exp_, x_ in zip(expected,x):
        assert abs(exp_ - parabola(x_ , a, b, c)) < tol, msg

test_parabola()

```

Fill in your answer here

Format
-
B
I
U
 $x_2$ 
 $x^2$ 
 $I_x$ 
↶
↷
↺
↻
☰
⋮
Ω
📊
✎
Σ
ABC
✖

Words: 0

Maximum marks: 3

### 1.11 Hvilket funksjonskall?

The function `trapezoidal(f,a,b,n)` below uses numerical integration to compute the integral of a function  $f(x)$  in the interval  $x=a$  to  $x=b$ . The parameter  $n$  is the number of sub-intervals used in the approximation.

```
import numpy as np
```

```
def trapezoidal(f,a,b,n=1000):
    x = np.linspace(a,b,n+1)
    dx = (b-a)/n
    return dx*(0.5*f(x[0]) + np.sum(f(x[1:-1])) + 0.5*f(x[-1]))
```

We want to use the function to compute the integral of  $\sin(x)$  for  $x=0$  to  $x = \pi$ . Which function call is correct?

Select one alternative:

- `trapezoidal(np.sin,0,np.pi)`
- `trapezoidal(np.sin(0),np.sin(np.pi),n=1000)`
- `trapezoidal(np.sin(x),0,np.pi)`
- `trapezoidal(np.sin(x),0,np.pi,1000)`

Maximum marks: 2

### 1.12 Hvilket funksjonskall?

The function `forward_euler(rhs,u0,T,n)` applies Eulers 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 to `T`, with `n` time steps:

```
import numpy as np
```

```
from math import *
```

```
def forward_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 equation  $y' = \sin(y)$ ,  $y(0) = 1$ , for  $t$  in the interval 0 to 5. Which line is correct?

Select one alternative:

- `f = sin(u); u,t = forward_euler(f,1.0,5)`
- `u,t = forward_euler(sin,1.0,5)`
- `f = lambda u,t: sin(u); u,t = forward_euler(f,1.0,5)` ✓
- `f = lambda u: sin(u); u,t = forward_euler(f,1.0,5)`

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

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

This line is equivalent with the following code:

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

---

Maximum marks: 2

## 2.1 Stykkvis konstant funksjon

Write a Python-function `piecewise(x)` which returns the mathematical function:

$$f(x) = \begin{cases} -1.0, & x < 0 \\ 1.0, & x \geq 0 \end{cases}$$

You can assume that  $x$  is a scalar value (not an array).



1	
---	--

Maximum marks: 3

## 2.2 Testfunksjon

Write a test function for the function you wrote in the previous exercise. The function shall be tested by choosing  $x = -0.5$  and  $x = 0.5$ , and comparing the return values from the function with the expected values.

Fill in your answer here

1	
---	--

Maximum marks: 3

## 2.3 Numerisk derivasjon

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

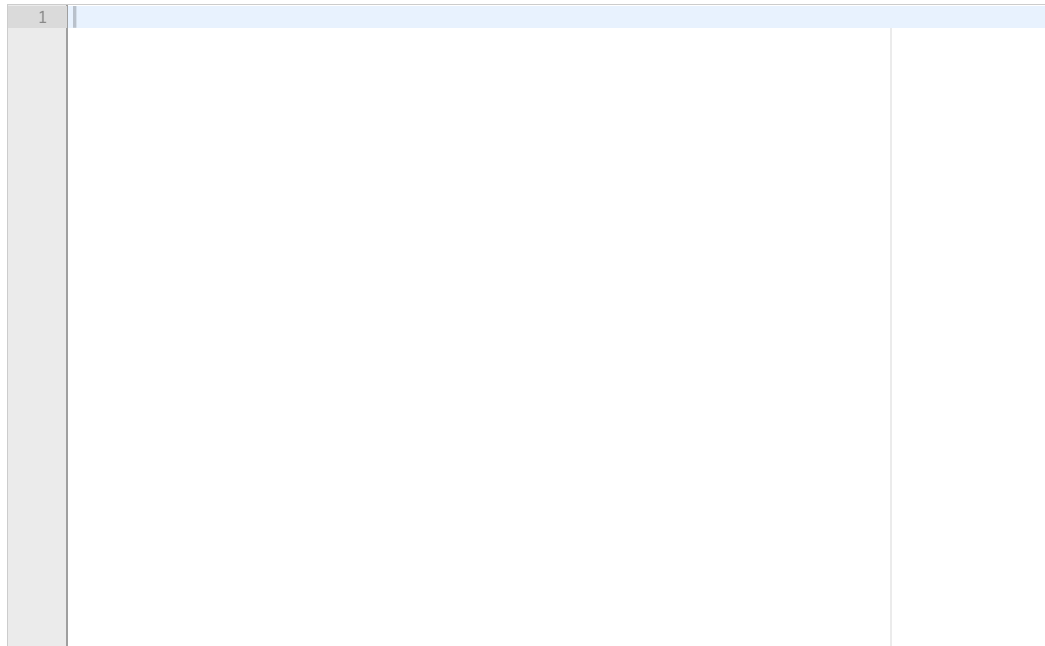
$$f'(x) \approx \frac{f(x+h) - f(x)}{h}$$

for some small number  $h$ .

Write a Python function `diff(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  $f'(x)$ . The argument  $f$  can be any mathematical function implemented in Python, which takes one input argument and returns one value.

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

**Fill in your answer here**



Maximum marks: 3

## 2.4 Implementasjon av en sum

Write a Python function `exp_approx(x,n)` which implements the formula

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

$x$  can be a floating point number (scalar) or an array of floats, while  $n$  is a positive integer. Recall that  $k!$  is the factorial of  $n$ . Include necessary imports.

1	
---	--

Maximum marks: 4

## 2.5 Plotting

The sum in the previous question approximates the exponential function. Write a function **plot\_approx(n\_values)**, which takes a list integers as argument. The function shall plot the result of **exp\_approx(x,n)** as a function of  $x$ , for all values of  $n$  in the list `n_values`. The curves shall be plotted in the same window, and for  $x$  in the interval 0 to 5.0. Also plot the exponential function  $\exp(x)$  in the same window, for the same interval of  $x$  values. Include necessary import.

(If you did not solve question 2.4 you may still assume that the function exists and works as defined in the question text.)

Fill in your answer here

1	
---	--

Maximum marks: 4

## 2.6 Lesing fra fil

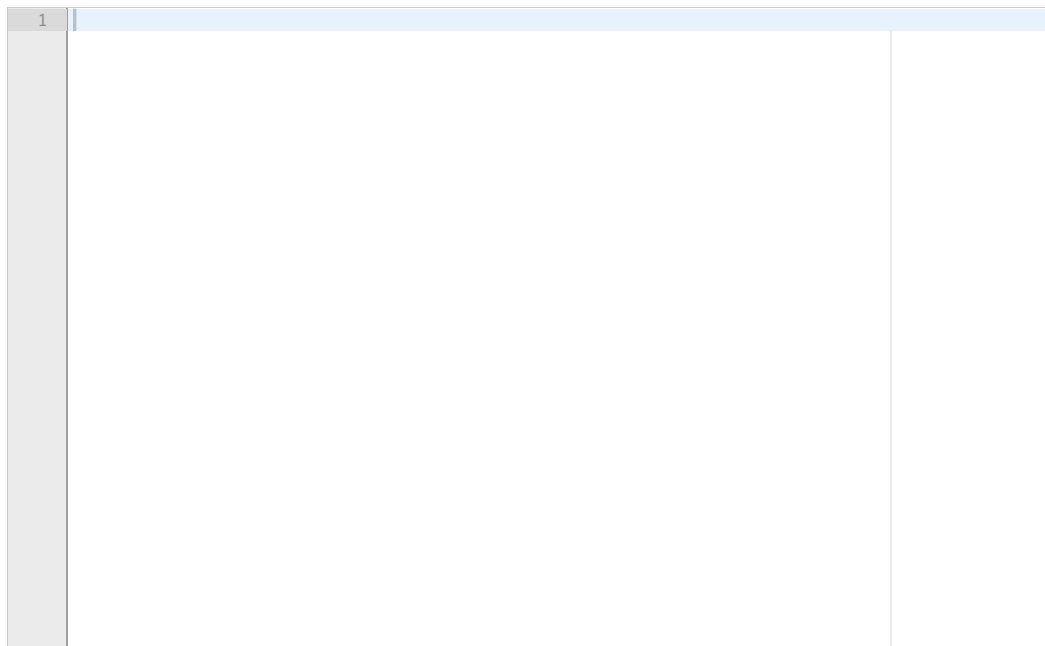
Write a function `read_file(filename)`, which reads a text file with file name given as an argument to the function, puts the content of the text file in a dictionary, and returns the dictionary.

You shall assume that each line in the file consists of two or more words separated by blanks. Each line in the file shall be read and put in the dictionary with the first word as key and the rest of the line (starting from the next non-blank character) as value. Each line then becomes a key-value pair in the dictionary. There are no blank lines in the file. Here is an example of a file that should be read by the function:

```
Norge      Oslo
USA        Washington, D.C.
Frankrike  Paris
Brunei     Bandar Seri Begawan
Malaysia   Kuala Lumpur
Chile      Santiago
Sri_Lanka  Colombo
```

Note: If words in the file are separated by more than one blank, these can be replaced by a single blank in the content of the dictionary. For instance if the file contains `Kuala Lumpur`, the corresponding value in the dictionary can be `Kuala Lumpur` or `Kuala Lumpur`, whatever you find easiest.

**Fill in your answer here**



Maximum marks: 5

## 3.1 Python-klasser

A class for a parabola is defined by the following code:

```
class Parabola:
    def __init__(self, c0, c1, c2):
        self.c0 = c0
        self.c1 = c1
        self.c2 = c2
```

```

def __call__(self, x):
    return self.c2*x**2 + self.c1*x + self.c0

def table(self, L, R, n):
    """Return a table with n points for L <= x <= R."""
    s = ""
    import numpy as np
    for x in np.linspace(L, R, n):
        y = self(x)
        s += '%12g %12g\n' % (x, y)
    return s

```

Write a class with name `Line`, for a linear function  $y = c_0 + c_1 x$ , as a subclass of `Parabola`. Use inheritance to reuse as much code as possible from `Parabola`.

The class `Line` shall be possible to use in the following way (here shown as an interactive Python session):

```

>>> from Line import Line
>>> l = Line(1.0,2.5) #create a line y=1.0+2.5*x
>>> print(l(0.5))
2.25
>>> print(l.table(0,1,3))
    0     1
0.5  2.25
1     3.5

```

Fill in your answer here

1

Maximum marks: 3

### 3.2 RK2 funksjon

Write a Python function `RungeKutta2(f, U0, T, n)`, which uses the Runge-Kutta 2 (RK2) method to solve an ordinary differential equation (ODE) given by:

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

The RK2 method is given by:

$$u_{k+1} = u_k + K_2$$

$$K1 = \Delta t f(u_k, t_k)$$

$$K2 = \Delta t f(u_k + \frac{1}{2}K1, t_k + \frac{1}{2}\Delta t)$$

The arguments to the function shall be a callable function  $f$ , which defines the right hand side of the ODE, the initial condition  $U_0$ , 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 the solution is approximated. For this question you can assume that we are solving a scalar ODE, with only one solution component. The solution array  $u$  can therefore be a one-dimensional array. Include necessary imports.

Fill in your answer here

1	
---	--

Maximum marks: 5

### 3.3 RK2 klasse

The attached file contains a class implementation of the forward Euler method. Write a class `RungeKutta2` that implements the RK2 method from the previous question. Let the class be a subclass of `ForwardEuler`, and use inheritance so that as much as possible is inherited from the base class, and only the necessary parts are written in `RungeKutta2`.

1	
---	--

Maximum marks: 3

### 3.4 SEIR-modellen

This question presents a so-called SEIR model for modeling the spreading of diseases. The model is an extension of the SIR-model, where the population is divided in four groups: those that can be infected (S), those that are infected but have not yet developed the disease, and can not yet infect others (E), those that are sick and can infect others (I), og those that have recovered and are immune (R). Let  $S(t)$ ,  $E(t)$ ,  $I(t)$  og  $R(t)$  be the number of people in each category (measured in millions). The following system of differential equations describes how  $S(t)$ ,  $E(t)$ ,  $I(t)$  og  $R(t)$  evolve over a time interval  $[0, T]$ :

$$S'(t) = -p(t)S(t)I(t),$$

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

$$I'(t) = qE(t) - rI(t),$$

$$R'(t) = rI(t).$$

At time  $t=0$  we have the initial conditions  $S(0) = S_0$ ,  $E(0) = E_0$ ,  $I(0) = 0$ ,  $R(0) = 0$ . The function  $p(t)$  and the constants  $q$  and  $r$  are assumed known. All constants and functions are  $>0$ .

Write a Python function **SEIR(S0,E0,p,q,r,T)**, which takes initial values  $S_0$ ,  $E_0$ , the function  $p(t)$ , parameters  $q$ ,  $r$ , and the final time  $T$  as input arguments. Use the ForwardEuler 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 points for a simulation over the interval  $[0, T]$  is  $10T+1$ . The function SEIR shall return 5 arrays:

$t$ , which contains the time points  $t_k$  where the numerical solution is computed,

$S$ , which contains  $S(0), S(t_1), \dots, S(t_n)$ ,

$E$ , which contains  $E(0), E(t_1), \dots, E(t_n)$ ,

$I$ , which contains  $I(0), I(t_1), \dots, I(t_n)$ ,

$R$ , which contains  $R(0), R(t_1), \dots, R(t_n)$ .

We want to solve the model with the following parameters;  $S_0 = 4.0$ ,  $E_0 = 0.2$ ,  $p = 0.0233$ ,  $q = r = 0.1$ .

Write the code for calling the function SEIR with the given parameters and  $T=100$ . Also include code for plotting  $S(t)$ ,  $E(t)$ ,  $I(t)$  and  $R(t)$  in the same window, with a legend for each curve.

1		
---	--	--

Maximum marks: 10

### 3.5 Klasse for ODE-modeller

Write a class **SEIR\_model** that represents the ODE model in the previous question. The class shall contain the following methods:

- A constructor that takes  $p(t)$ ,  $q$ ,  $r$ ,  $S_0$ ,  $I_0$  and  $T$  as arguments. The final time  $T$  and the parameters  $p$ ,  $q$  and  $r$  shall be stored as attributes (instance attributes). The initial conditions  $S_0$  and  $I_0$  are to be collected in a list  $\text{self.U0} = [S_0, I_0, 0, 0]$ .
- A `__call__`-method that defines the right hand side of the ODE system.

The class shall be possible to use in the following way:

```
def p(t):
```

```
    return 0.0223
```

```
problem = SEIR_model(p, 0.1, 0.1, 4.0, 0.2, 100)
```

```
solver = ForwardEuler(problem, problem.U0, problem.T, n=1001)
```

```
solver.solve()
```



1	
---	--

---

Maximum marks: 5

**Question 3.3**  
Attached



```

import numpy as np
class ForwardEuler:
    """
    Class 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)
    U0: initial condition (scalar or array)
    """

    def __init__(self, f,U0, T, n):
        if not callable(f):
            raise TypeError('f is not a function')
        self.f = lambda u, t: np.asarray(f(u, t))
        self.t = np.linspace(0,T,n+1)
        self.k = 0
        if isinstance(U0, (float,int)): # scalar ODE
            self.neq = 1
            self.u = np.zeros(n+1)
        else: # system of ODEs
            U0 = np.asarray(U0)
            self.neq = U0.size
            self.u = np.zeros((n+1,self.neq))
        self.U0 = U0

    def solve(self):
        """Solve the ODE from time 0 to T.
        Store solution in self.u.
        Return self.u and self.t.
        """

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

```

# UNIVERSITETET I OSLO

## Det matematisk-naturvitenskapelige fakultet

Examination in: IN1900 — Introduction to  
programming with scientific  
applications

Day of examination: Wednesday, November 28, 2018

Examination hours: 14.30 – 18.30

This examination set consists of 8 pages.

Appendices: None

Permitted aids: None

Make sure that your copy of the examination set is complete before you start solving the problems.

- This part contains suggested solutions for all questions except the multiple-choice questions.
- Note that for all the programming questions there are several possible solutions. The solutions presented here are usually among the simplest options, but not necessarily the simplest or the best. There are usually many alternative solutions that would also be given full score on the exam.

*(Continued on page 2.)*

## Question 1.6

*Answer:*

The total rainfall from June to August was 54.00

## Question 1.8

*Answer:*

The value of the polynomial is 7.5

## Question 1.9

*Answer:*

4.3 23.6

## Question 1.10

*Answer:* Nothing gets printed because the test passes.

## Question 2.1

*Answer:*

*(Continued on page 3.)*

```
def piecewise(x):  
    if x < 0:  
        return -1.0  
    else:  
        return 1.0
```

## Question 2.2

*Answer:*

```
def test_piecewise():  
    x = (-0.5,0.5)  
    expect = (-1,1)  
    tol = 1e-6  
    for x_, e_ in zip(x,expect):  
        assert abs(piecewise(x_)-e_) < tol
```

## Question 2.3

*Answer:*

```
from math import sin  
  
def diff(f,x,h):  
    return (f(x+h)-f(x))/h  
  
print(diff(sin,0.0,0.001))
```

*(Continued on page 4.)*

## Question 2.4

*Answer:*

```
from math import factorial

def exp_approx(x,n):
    s = 0
    for k in range(n+1):
        s += x**k/factorial(k)
    return s
```

## Question 2.5

*Solution:*

```
from math import exp
import matplotlib.pyplot as plt

def plot_approx(n_values):
    x = np.linspace(0,5,100)
    for n in n_values:
        y = exp_approx(x,n)
        plt.plot(x,y)
    plt.plot(x,np.exp(x))
    n_values.append('exact')
    plt.legend(n_values)
    plt.show()
```

## Question 2.6

*Solution:*

```
def read_file(filename):
```

*(Continued on page 5.)*

```
infile = open(filename, 'r')

countries = {}

for line in infile:
    words = line.split()
    countries[words[0]] = ' '.join(words[1:])

return countries
```

### Question 3.1

*Solution:*

```
class Line(Parabola):
    def __init__(self, c0, c1):
        Parabola.__init__(self, c0, c1, 0)
```

### Question 3.2

*Solution:*

```
def RungeKutta2(f, U0, T, n):
    time = np.linspace(0, T, n+1)
    u = np.zeros_like(time)
    dt = T/n

    u[0] = U0
    for i in range(1, n+1):
        K1 = dt*rhs(u[i-1], t[i-1])
        K2 = dt*rhs(u[i-1]+0.5*K1, t[i-1]+0.5*dt)
        u[i] = u[i-1]+K2
```

*(Continued on page 6.)*



```
return u,time
```

Note that the text for Question 3.2 contained a typo, as the ODE was defined as  $u' = f(u)$ , not the usual  $u' = f(u, t)$ . Answers to this question are given the same score regardless of whether they treated the right hand side as a function of one or two variables.

### Question 3.3

*Solution:*

```
from ForwardEuler import ForwardEuler

class RungeKutta2(ForwardEuler):
    def advance(self):
        f, t, u, k = self.f, self.t, self.u, self.k
        dt = t[k+1]-t[k]
        K1 = dt*rhs(u[k],t[k])
        K2 = dt*rhs(u[k]+0.5*K1,t[k]+0.5*dt)
        return u[k] + K2
```

On this question the attached pdf was incomplete. The idea was that the ForwardEuler class was a complete and usable solver class, so one only needed to reimplement the advance-function.

### Question 3.4

*Solution:*

```
def SEIR(S0,E0,p,q,r,T):
    def rhs(u,t):
        S,E,I,R = u
        dS = -p(t)*S*I
```

*(Continued on page 7.)*

```

    dE = p(t)*S*I-q*E
    dI = q*E-r*I
    dR = r*I
    return [dS,dE,dI,dR]

U0 = [S0,E0,0,0]

solver = ForwardEuler(rhs,U0,T,10*T+1)
solver.solve()
return solver.t, solver.u[:,0], solver.u[:,1], solver.u[:,2], solver.u[:,3]

def p(t):
    return 0.0233

t, S, E, I, R = SEIR(4.0,0.2,p, 0.1, 0.1,100)

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

```

## Question 3.5

*Solution:*

```

class SEIR_model:
    def __init__(self, p, q, r, S0, I0,T):
        self.p = p
        self.q = q
        self.r = r
        self.U0 = [S0, I0, 0,0]
        self.T = T

    def __call__(self, u,t):
        S,E,I,R = u

```

*(Continued on page 8.)*

```
dS = -self.p(t)*S*I
dE = self.p(t)*S*I-self.q*E
dI = self.q*E-self.r*I
dR = self.r*I
return [dS,dE,dI,dR]
```

```
problem = SEIR_model(p, 0.1, 0.1, 4.0, 0.2,100)
s = ForwardEuler(problem, problem.U0, problem.T, n=1001)
s.solve()
t,S,E,I,R = s.t, s.u[:,0], s.u[:,1], s.u[:,2], s.u[:,3]
```

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

This question also contained a typo, as E0 had been replaced by I0. This opened up for different interpretations, and different solutions will be accepted in the grading.

END