Christian Vollmann *Winter Term 2021/2022*

Elements of Mathematics

Exercise Sheet 1

Submission due date: **02.11.2021, 10:15h**

THEORY

---

1 Set Operations

Let $M = \{\alpha, 3, \gamma\}$, $N = \{\square, \gamma, \star, \circ\}$ and $K = \{\alpha, \{\alpha\}, M\}$. Please indicate the sets

1. $M \cap N$,
2. $M \cap K$,
3. $\mathcal{P}(M) \cap K$,
4. $M \setminus K$ and
5. $N \cup K$.

(10 Points)

Solution:

1. $M \cap N = \{\gamma\}$
2. $M \cap K = \{\alpha\}$
3. $\mathcal{P} \cap K = \{\{\alpha\}, M\}$
4. $M \setminus K = \{3, \gamma\}$
5. $N \cup K = \{\square, \gamma, \star, \circ, \alpha, \{\alpha\}, M\}$

2 Symmetric Difference

Let X be a set and $A, B \subset X$ be two subsets of X. The symmetric difference of A and B is then defined by

$$A \Delta B := (A \cap B^c) \cup (B \cap A^c).$$

Please show that

$$A \Delta B = (A \cup B) \cap (A \cap B)^c.$$

Hint: Use the De Morgan's rule and exploit the following distributive laws:

$$A \cap (B \cup C) = (A \cap B) \cup (A \cap C) \quad \text{and} \quad A \cup (B \cap C) = (A \cup B) \cap (A \cup C).$$

(8 Points)

Solution:

$$
\begin{aligned}
&(A \cap B^c) \cup (B \cap A^c) \\
=&((A \cap B^c) \cup B) \cap ((A \cap B^c) \cup A^c) \\
=&\underbrace{((A \cup B) \cap (B^c \cup B))}_{A \cup B} \cap \underbrace{((A \cup A^c) \cap (B^c \cup A^c))}_{B^c \cup A^c} \\
=&(A \cup B) \cap (B^c \cup A^c) \\
=&(A \cup B) \cap (A \cap B)^c
\end{aligned}
$$

3 Complex Numbers

Let i be the imaginary unit, i.e., $i^2 = -1$. Please cast the following complex numbers into the format $z = x + iy$.

1. $(4+5i)(4-5i)$
2. $\frac{2+3i}{4+5i}$
3. $\sqrt{16b} + \sqrt{3a-12a}$

(6 Points)

Solution:

1. $(4+5i)(4-5i) = 4^2 - (5i)^2 = 16 + 25 = 41$
2. $\frac{2+3i}{4+5i} = \frac{(2+3i)(4-5i)}{41} = \frac{8-10i+12i-15i^2}{41} = \frac{23+2i}{41}$
3. $\sqrt{16b} + \sqrt{3a-12a} = 4\sqrt{b} + \sqrt{-9a} = 4\sqrt{b} + i3\sqrt{a}$

4 Result for Injective Funtions

Let $f : X \to Y$ be an injective function and $A, B \subset X$. Please show that

$$f(A) \cap f(B) = f(A \cap B).$$

Hint: Split up the equality "$=$" into the parts "$\subset$" and "$\supset$". One direction is straightforward the other one requires, that f is injective. *(8 Points)*

Solution:

- $f(A \cap B) \subset f(A) \cap f(B)$:
  We have
  $$f(A \cap B) = f(A \cap B) \cap f(B \cap A) \subset f(A) \cap f(B).$$
- $f(A) \cap f(B) \subset f(A \cap B)$:
  Let $y \in f(A) \cap f(B)$,
  $\Rightarrow \quad \exists\, x_a \in A, x_b \in B : \; f(x_a) = y = f(x_b)$.

  Since f is injective, we find $x_a = x_b$
  $\Rightarrow \quad x_a, x_b \in A \cap B$
  $\Rightarrow \quad y \in f(A \cap B)$.

5 Heron's algorithm

For a nonnegative number $a \geq 0$ compute the square root $\sqrt{a}$ up to an error of 10^{-10}. For this purpose, use the following iteration:

$$x_{n+1} = \frac{1}{2}\left(\frac{x_n^2 + a}{x_n}\right).$$

Choose different initial values x_0 and observe the convergence behavior by printing the error $|x_n - \sqrt{a}|$ in each iteration step.
Hint: Use a while-loop for the iteration (`while`) and the built-in function abs as well as the Python value `a ** 0.5` to compute the error $|x_n - \sqrt{a}|$ in each iteration step. *(6 Points)*

Solution:

```
def heron(a, x0=0.1, tol=10e-10, maxiter=1000):
    """
    applies the iteration rule according to the so-called "Heron method"
    """
    counter = 0
    # while the error is above our tolerance we do the iteration
    while abs(x0 - a**0.5) > tol:
        # print the current error
        print("Error =", abs(x0 - a**0.5))
        # perfom one step of the heron method:
        x0 = 0.5 * ((x0 ** 2 + a) / x0)
        # update the counter
        counter += 1
        if counter > maxiter:
            # stop the while loop if too many iterations
            break
    print("\n Result: sqrt(a) =", x0, "\n")
    return x0


if __name__ == "__main__":
    a = 2.

    # Choose different initial values
    x0 = [0.1, 1., 100000, 1.3]
    for x in x0:
        print("\n x0 =", x, "\n---------------\n")
        heron(a, x0=x)

    help(heron)
```

6 Partial Sum

Implement a program which outputs the n-th partial sum $\sum_{k=1}^{n} a_k$ of a sequence $(a_k)_{k\in\mathbb{N}}$.

Hint: Define the sequence as a Python function (`def`) and use a for-loop (`for`) to compute the sum.
(6 Points)

Solution:

```
def partial_sum(a, n):
    """
        a: python function
        n: positive integer
```

```
    """
    summe = 0
    for k in range(1, n+1):
        summe += a(k)
    return summe


if __name__ == "__main__":

    # Our example: we sum up the numbers from 1 to n
    def a(k):
        return k  # k**2

    n = 50

    # we compare to the "kleiner Gauss" (Gauss summation formula)
    print("Our result:             ", partial_sum(a, n))
    print("Gauss summation formula:", n * (n+1)//2)
```

SCIENTIFIC COMPUTING WITH PYTHON

Scientific computing in Python is done using the **SciPy Stack** (https://scipy.org/). This term is used for a collection of packages developed for scientific computing. From this collection we will mainly use the following three packages during this lecture:

- **Numpy** (the basis)
  – provides the data type 'numpy.ndarray' (e.g., for matrices and vectors)
  – contains a huge amount of tools to perform all sorts of array manipulation

- **SciPy** (the core)
  – builds upon numpy
  – contains a huge amount of numerical methods (solving linear systems, optimization, integration, interpolation,...)

- **Matplotlib** (the visualizer)
  – allows to visualize results with high quality grafics (plots in 2d and 3d, images and videos,...)

Remarks:

- **Tutorial:** the official one can be found here: https://scipy-lectures.org/. The first chapter in this tutorial contains already enough to do the programming parts in this lecture.
- **Installation:** typically not needed! They are usually contained in the system-side Python installation or come along with the respective distribution (e.g., anaconda)
- **Import convention:** a program often starts with

```
import numpy as np
import scipy as sp
import matplotlib as mpl
import matplotlib.pyplot as plt
```

Solution:

7 NumPy and Matplotlib (not mandatory)

1. Have a look at the first chapter of the scipy-lectures: `https://scipy-lectures.org/` (namely, '1. Getting started with Python for science')
2. Play around with the numpy functions: `arange`, `linspace`, `ones`, `zeros`, `eye`, `diag`, `tile`, `repeat` and `reshape`.
3. Generate the following matrices:
   (a) Identity matrix
   (b) Zero matrix
   (c) A diagonal matrix ($D \in \mathbb{F}^{m \times n}$ is called diagonal, if $d_{ij} = 0 \; \forall i \neq j$)
   (d) Example matrix
   $$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 2 \\ 1 & 6 & 1 & 1 \end{pmatrix}$$
   (e) Example matrix
   $$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 0 & 6 \end{pmatrix}$$
   (f) Example matrix
   $$\begin{pmatrix} 4 & 3 & 4 & 3 & 4 & 3 \\ 2 & 1 & 2 & 1 & 2 & 1 \\ 4 & 3 & 4 & 3 & 4 & 3 \\ 2 & 1 & 2 & 1 & 2 & 1 \end{pmatrix}$$
   (g) Example matrix
   $$\begin{pmatrix} 1 & 6 & 11 \\ 2 & 7 & 12 \\ 3 & 8 & 13 \\ 4 & 9 & 14 \\ 5 & 10 & 15 \end{pmatrix}$$
4. Apply the functions `len()` and `numpy.shape()` to these arrays. How do the values relate? What does the attribute `ndim` tell us?
5. Use `matplotlib.pyplot.imshow()` to visualize the matrices.

Solution:

```
import numpy as np
import matplotlib.pyplot as plt

""" 4.1. """

""" 4.2."""

m, n = 10, 5

# help(np.arange)
```

```
print(np.arange(1, n, 2))
print(np.arange(n))

# help(np.linspace)
print(np.linspace(0, 1, n, endpoint=False))

# help(np.ones)
print(np.ones(n))
print(np.ones((n, n)))

# help(np.zeros)
print(np.zeros(n))
print(np.zeros((n, n)))

# help(np.eye)
print(np.eye(n))
print(np.eye(m, n))
print(np.eye(m, n, 3))

# help(np.diag)
print(np.diag(np.ones(n)))
print(np.diag(np.ones(n), 3))


# help(np.tile)
print(np.tile(np.arange(3), 2))
print(np.tile(np.arange(3), (2, 3)))
print(np.tile(np.ones((2, 2)), 3))

# help(np.repeat)
# Repeat elements of an array
print(np.repeat(np.arange(3), 2))
# print(np.repeat(np.arange(3), (2, 3)))
print(np.repeat(np.ones((2, 2)), 3))
print(np.repeat(np.ones((2, 2)), 3, axis=1))

# help(np.reshape)
x = np.eye(3)
print(x)
print(np.reshape(x, 9))

""" 4.3."""
# a) identity matrix

# b) zero matrix

# c) diagonal matrix
d = np.arange(n)
A = np.diag(d)
print(np.diag(d))
print(len(A), np.shape(A), A.ndim)
plt.figure()
plt.imshow(A)
plt.show()

# d)
A = np.ones((4, 4))
A[3, 1] = 6
A[2, 3] = 2
print(A)
print(len(A), np.shape(A), A.ndim)
plt.figure()
```

```
plt.imshow(A)
plt.show()

# e)
A = np.diag(np.arange(2, 7, 1), -1)
print(A)
print(len(A), np.shape(A), A.ndim)
plt.figure()
plt.imshow(A)
plt.show()

# f)
x = np.arange(4, 0, -1).reshape(2, 2)
A = np.tile(x, (2, 3))
print(A)
print(len(A), np.shape(A), A.ndim)
plt.figure()
plt.imshow(A)
plt.show()

# g)
x = np.arange(1, 16)
A = np.reshape(x, (5, 3), order='F')
print(A)
print(len(A), np.shape(A), A.ndim)
plt.figure()
plt.imshow(A)
plt.show()
```

Total Number of Points = 44 (T:32, P:12)