Dr. Christian Vollmann

Winter Term 2022

[November 4, 2022]

Elements of Mathematics

Sheet 01

Due date: **XXX**

Name:

Matriculation number:

Task:	1	2	3	4	Total	Grade
Points:						
Total:	6	6	12	10	34	–

A matrix as a Linear Function
Let $A \in \mathbb{F}^{m\times n}$ be a matrix. Then consider the mapping $f_A\colon \mathbb{F}^n \to \mathbb{F}^m, x \mapsto Ax$.

1. Show that
$$f_A(\lambda x + y) = \lambda f_A(x) + f_A(y),$$
for all $x, y \in \mathbb{F}^n$ and $\lambda \in \mathbb{F}$.

   Hint: A vector is a matrix with just one column, so you can make use of the computation rules for matrices given in the lecture notes.

   Remark: Functions satisfying this property are called **linear**.

2. Use this fact to show the following equivalence:
$$\ker(A) := \{x \in \mathbb{F}^n \colon Ax = 0\} = \{0\} \quad \Leftrightarrow \quad f_A(x) = f_A(y) \text{ implies } x = y,$$
$$\text{(i.e., } f_A \text{ is an injective mapping).}$$

   Hint: Split up the equality $\Leftrightarrow$ into $\Rightarrow$ and $\Leftarrow$ and prove each of them separately.

Solution:

1. Let $x, y \in \mathbb{F}^n$ and $\lambda \in \mathbb{F}$. Then
$$f_A(\lambda x + y) = A(\lambda x + y) = A(\lambda x) + Ay = \lambda Ax + Ay = \lambda f_A(x) + f_A(y).$$

2. "$\Rightarrow$" Let $\ker(A) = \{0\}$
   (To show: f_A is an injective mapping, i.e., $f_A(x) = f_A(y)$ implies $x = y$.)
   Let $x, y \in \mathbb{F}^n$ with $f_A(x) = f_A(y)$, which implies by definition $Ax = Ay$ and thus by linearity $A(x - y) = 0$.
   Thus, since $\ker(A) = \{0\}$, we conclude $x - y = 0$.

   "$\Leftarrow$" Let f_A be an injective mapping, i.e., $f_A(x) = f_A(y)$ implies $x = y$.
   (To show:: $Ax = 0 \Leftrightarrow x = 0$ (here "$\Leftarrow$" is obvious).)
   Let $Ax = 0$, then we find
$$f_A(0) = A0 = 0 = Ax = f_A(x).$$
   Thus, since f_A is assumed to be injective, $x = 0$ (take "$y = 0$").

The Subspaces Kernel and Image

1. Let $A \in \mathbb{F}^{m\times n}$. Show that $\ker(A)$ and $\mathrm{Im}(A)$ are subspaces of $\mathbb{F}^n$ and $\mathbb{F}^m$, respectively.
2. Construct two example matrices and consider their kernel and image.

Solution:

1. a) To show: $\ker(A) \subset \mathbb{F}^n$ subspace

      Proof:

      i. $A \cdot 0 = 0$, so that $0 \in \ker(A)$, thus $\ker(A) \neq \emptyset$.

      ii. For $i = 1, 2$ let $\lambda_i \in \mathbb{F},\ v_i \in \ker(A),$ then by linearity $A(\lambda_1 v_1 + \lambda_2 v_2) = \lambda_1 \underbrace{Av_1}_{=0} + \lambda_2 \underbrace{Av_2}_{=0} = 0$
      $$\Rightarrow \quad \lambda_1 v_1 + \lambda_2 v_2 \in \ker(A)$$

   b) To show: $\mathrm{Im}(A) \subset \mathbb{F}^m$ subspace

      Proof:

      i. $A \cdot 0 = 0 \in \mathrm{Im}(A)$, thus nonempty.

ii. For $i = 1, 2$ let $\lambda_i \in \mathbb{F},\ w_i \in \mathsf{Im}(A)$, then

$$\begin{aligned} & \exists v_1, v_2 \in \mathbb{F}^n : \ w_1 = Av_1,\ w_2 = Av_2 \\ \Rightarrow\quad & \lambda_1 w_1 + \lambda_2 w_2 = \lambda_1 A v_1 + \lambda_2 A v_2 = A(\lambda_1 v_1 + \lambda_2 v_2) \\ \Rightarrow\quad & \lambda_1 w_1 + \lambda_2 w_2 \in \mathsf{Im}(A) \end{aligned}$$

2. Examples:
   a) Consider the matrix composed of ones from the previous exercise.
   b) Let $A = \begin{pmatrix} 1 & 1 \\ 0 & 2 \end{pmatrix}$. Then $Ax = 0$ implies $x = 0$, so that $\ker(A) = \{0\}$. In particular, the columns are linearly independent, so that they form a basis of $\mathbb{R}^2$, with other words: $\mathbb{R}^2 = \mathsf{span}(a_1, a_2) = \mathsf{Im}(A)$.

Rank/Image and Nullity/Kernel
Consider the matrix

$$A = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix},$$

the column vector $\mathbf{1} := \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$ (i.e., a (3×1) matrix) and the row vector $\tilde{\mathbf{1}} := (1\ 1\ 1)$ (i.e., a (1×3) matrix).

1. Show that $A = \mathbf{1}\tilde{\mathbf{1}}$.
2. Find two distinct nonzero vectors x and y, so that $Ax = 0$ and $Ay = 0$.
3. How does the image $\mathsf{Im}(A)$ look like? First draw a picture. Then find a basis of $\mathsf{Im}(A)$ to determine the rank of the matrix.
4. How does the kernel $\ker(A)$ look like? First draw a picture. Then find a basis of $\ker(A)$ to determine the nullity of the matrix.

   Remark: You have to prove that your vectors are a basis.

Solution:

1. By applying the matrix-matrix product definition we multiply the matrix $\mathbf{1}$ with each column in $\tilde{\mathbf{1}}$ (here, a column is just the number 1). We obtain

$$\mathbf{1}\tilde{\mathbf{1}} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \cdot (1\ 1\ 1) = \left(1 \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} 1 \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} 1 \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \right) = A.$$

2. Since $a_1 = a_2 = a_3 = \mathbf{1}$, we have

$$0 = Ax = a_1 x_1 + a_2 x_2 + a_3 x_3 = a_1(x_1 + x_2 + x_3) \Leftrightarrow x_1 + x_2 + x_3 = 0.$$

   Choose, e.g., $x = \begin{pmatrix} -1 \\ 1 \\ 0 \end{pmatrix}$ and $y = \begin{pmatrix} -1 \\ 0 \\ 1 \end{pmatrix}$.

3. By definition of the image we have

$$\begin{aligned} \mathsf{Im}(A) &= \mathsf{span}(a_1, a_2, a_3) \\ &= \{\lambda_1 \mathbf{1} + \lambda_2 \mathbf{1} + \lambda_3 \mathbf{1} \colon \lambda_i \in \mathbb{R}\} \\ &= \{\lambda \mathbf{1} \colon \lambda \in \mathbb{R}\} \\ &= \mathsf{span}(\mathbf{1}). \end{aligned}$$

   Since $\mathbf{1} \neq 0$, we have that $\{\mathbf{1}\}$ is a basis of length 1 for $\mathsf{Im}(A)$. In particular we find

$$\mathsf{rank}(A) := \dim \mathsf{Im}(A) = 1.$$

   (Note that two equal vectors $x = y$ are linearly dependent and that a single nonzero vector $x \neq 0$ is linearly independent.)

4. From 2. we already know

$$
\begin{aligned}
\ker(A) := \{x \in \mathbb{R}^3 \colon Ax = 0\} &= \{x \in \mathbb{R}^3 \colon x_1 + x_2 + x_3 = 0\} \\
&= \{x \in \mathbb{R}^3 \colon x_1 = -(x_2 + x_3)\} \\
&= \left\{ \begin{pmatrix} -x_2 - x_3 \\ x_2 \\ x_3 \end{pmatrix} \colon x_2, x_3 \in \mathbb{R} \right\} \\
&= \left\{ x_2 \begin{pmatrix} -1 \\ 1 \\ 0 \end{pmatrix} + x_3 \begin{pmatrix} -1 \\ 0 \\ 1 \end{pmatrix} \colon x_2, x_3 \in \mathbb{R} \right\} \\
&= \operatorname{span} \left\{ \begin{pmatrix} -1 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} -1 \\ 0 \\ 1 \end{pmatrix} \right\}.
\end{aligned}
$$

Since $b_1 := \begin{pmatrix} -1 \\ 1 \\ 0 \end{pmatrix}$ and $b_2 := \begin{pmatrix} -1 \\ 0 \\ 1 \end{pmatrix}$ are linearly independent (in fact, one can show $[b_1, b_2]x = 0$ implies $x = 0$), they form a basis of $\ker(A)$ and thus we have $\dim(\ker(A)) = 2$.

Ex 4 Linear Algebra, Python 10

The Matrix-Vector Product

Implement a function that takes as input a matrix $A \in \mathbb{R}^{m \times n}$ and a vector $x \in \mathbb{R}^n$ and then returns the matrix-vector product Ax.

1. Implement the following four ways:
   a) **Dense:** Input expected as `numpy.ndarray`:
      Assume that the matrix and the vector are delivered to your function as `numpy.ndarray`.
      i. Implement the matrix-vector product "by hand" using for loops, i.e., *without* using numpy functions/methods.
      ii. Implement the matrix-vector product using `A.dot(x)`, `A@x`, `numpy.matmul(A,x)` or `numpy.dot(A,x)`.
   b) **Sparse:** Matrix expected in CSR format:
      Assume that the matrix is delivered to your function as `scipy.sparse.csr_matrix` object. The vector x can either be expected as `numpy.ndarray` or simply as a Python `list`.
      i. Access the three CSR lists via `A.data, A.indptr, A.indices` and implement the matrix-vector product "by hand" using for loops.
      ii. Implement the matrix-vector product using `A.dot(x)` or `A@x` .

2. **Test** your four different routines from above on the following matrix $A \in \mathbb{R}^{n \times n}$ with constant diagonals given by

$$
A = \begin{pmatrix} 2 & -1 & 0 & \cdots & 0 \\ -1 & 2 & -1 & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & -1 & 2 & -1 \\ 0 & \cdots & 0 & -1 & 2 \end{pmatrix} \in \mathbb{R}^{n \times n}
$$

   and the input vector

$$
x = (1, \cdots, 1)^\top \in \mathbb{R}^n \quad \text{(you can use: } \texttt{x = numpy.ones(n)}\text{)}.
$$

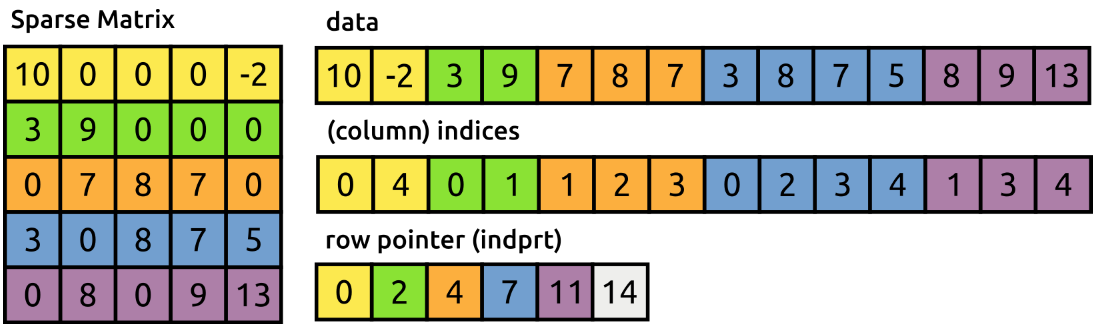
   a) Determine how $b := A \cdot x \in \mathbb{R}^n$ looks like in this example in order to facilitate a test.
   b) Test whether your four routines compute the matrix–vector product correctly by checking $A \cdot x = b$.
   c) Use different values for the dimension n (especially large $n \geq 10^5$ – note that you may exceed your hardware capacities for the dense computations).

   Remark: The matrix has "2" on the main diagonal and "-1" on the first off-diagonals.

3. For all cases:

a) **Memory:** What is the number of Gbytes needed to store an $m \times n$ array of `floats`? Print the number of Gbytes which are needed to store the matrix in all cases.
Hint: A number implemented as `float` in Python implements double precision and therefore needs 64 Bits of storage. For a `numpy.ndarray` you can type `A.nbytes` and for the `scipy.sparse.csr_matrix` you can type `A.data.nbytes + A.indptr.nbytes + A.indices.nbytes`.

b) **Computation times:** Measure the time which is needed in each case to compute the matrix-vector product for a random input vector `x = numpy.random.rand(n)`.
Hint: In the IPython shell you can simply use the *magic function* `%timeit` to measure the time for a certain operation. For example, you can type `%timeit pythonfunction(x)`. Alternatively you can use the package `timeit`.



Example of a Matrix in CSR Format

Solution:

```
import numpy as np
import scipy.sparse as scs
import timeit


def matvec_dense(A, x, byhand=0):
    """
    computes the matrix vector product based on numpy.ndarray

    Parameters
    ----------
    A : (m,n) numpy.ndarray
       matrix
    x : (n, ) numpy.ndarray
       vector
    byhand : int
       switch between for loop and @ operator

    Returns
    -------
        A*x: matrix-vector product
    """
    if byhand:
        # read the dimensions of the input objects
        m, n = np.shape(A)
        nx = len(x)

        # raise an error if the dimensions do not match
        if n != nx:
            raise Exception('dimension of A and x must match. The dimension \
                            for A and x were: {}'.format(str(np.shape(A))
                                                         + " " + str(len(x))))

        # if dimensions match, start computing the matrix-vector product:
        else:
            # initialize the output vector
            b = np.zeros(m)
            # a loop over row indices to compute each entry of b
            for i in range(m):
                # a loop over column indices to compute the inner product
```

```python
                for j in range(n):
                    b[i] += A[i, j] * x[j]
    else:
        b = A.dot(x)  # np.dot(A,x), A@x
    return b


# we could implement our own csr-class in python:
# class csr_matrix:
#     def __init__(self, data, indices, indptr):
#         self.data = data
#         self.indices = indices
#         self.indptr = indptr

def matvec_sparse(A, x, byhand=0):
    """computes the matrix vector product based on csr matrix

    Parameters
    ----------
    A: (m,n) matrix  stored in CSR, i.e., in terms of three lists; here:
       class with attributes data, indices, indptr
    x: (n, ) numpy.ndarray or list of length n (= number of cols) numbers
       vector
    byhand : int
       switch between for loop and @ operator

    Returns
    -------
        A*x: matrix-vector product
    """
    if byhand:
        # dimension check?
        # can we get the column dimension from sparse csr class? > depends
        b = [0] * (len(A.indptr) - 1)
        for i, pair in enumerate(zip(A.indptr[0:-1], A.indptr[1:])):
            for a_ij, j in zip(A.data[pair[0]:pair[1]],
                               A.indices[pair[0]:pair[1]]):
                b[i] += a_ij * x[j]
    else:
        # make sure A and x have the correct format for the dot method
        A = scs.csr_matrix(A)
        x = np.array(x)
        # compute matrix-vector product
        b = A.dot(x)
    return np.array(b)


print("\nIn order to get the docstring of our function we can type  \
      \n\n     help(functionName)\n\nFor example: ")
print(help(matvec_dense))

if __name__ == "__main__":
    # Note: the following part is only executed if the current script is
    #       run directly, but not if it is imported into another script
    # ---------------------------------------------------------------#
    #    EXPERIMENT
    # ---------------------------------------------------------------#
    # the experiment
    n = int(1e3)  # matrix column dimension
    m = n  # matrix row dimension
    runs = 50  # how many runs for time measurement
    x = np.random.rand(n)  # random vector x

    # test arrays for which we know the result
    xtest = np.ones(n)  # test input x
    btest = np.zeros(m)  # known test output b
    btest[[0, -1]] = 1
```

```python
# just some strings for printing commands
expstr = ["Time dot:      ", "Time hand:    "]
teststr = ["Test dot:      ", "Test by hand: "]

# ---------------------------------------------------------------#
#    NUMPY DENSE
# ---------------------------------------------------------------#
print("\n---- Numpy Dense ----")
A = 2 * np.eye(n) - np.eye(n, k=1) - np.eye(n, k=-1)
print("Memory:", np.round(A.nbytes * 10 ** -9, decimals=4), "Gbytes\n")
for byhand in [0, 1]:
    print(teststr[byhand], np.allclose(btest,
                                           matvec_dense(A, xtest, byhand=byhand)))


    def dense():
        return matvec_dense(A, x, byhand=byhand)


    print(expstr[byhand], timeit.timeit("dense()",
                                            setup="from __main__ import dense", number=runs),
"\n")

# ---------------------------------------------------------------#
#   SCIPY SPARSE
# ---------------------------------------------------------------#
print("\n---- Scipy Sparse ----")
A = 2 * scs.eye(n) - scs.eye(n, k=1) - scs.eye(n, k=-1)
print("Memory:", np.round((A.data.nbytes + A.indptr.nbytes +
                           A.indices.nbytes) * 10 ** -9, decimals=4),
      "Gbytes\n")
for byhand in [0, 1]:
    print(teststr[byhand],
          np.allclose(btest, matvec_sparse(A, xtest, byhand=byhand)))


    def sparse():
        return matvec_sparse(A, x, byhand=byhand)


    print(expstr[byhand], timeit.timeit("sparse()",
                                            setup="from __main__ import sparse", number=runs),
 "\n")
```