Christian Vollmann

Elements of Mathematics

Exercise Sheet 2

Submission due date: **09.11.2021, 10:15h**

THEORY

Computation Rules for Matrices and Vectors

Below you find a collection of computation rules that are helpful when dealing with matrices and vectors. Feel free to prove them based on the definitions given in the lecture.

Compatibility properties of summing and scaling matrices
Let $A, B \in \mathbb{F}^{m\times n}$ and $r, s \in \mathbb{F}$. Then

$$\begin{array}{ll} i) & (r\cdot s)\cdot A = r\cdot(s\cdot A) \\ ii) & (r+s)\cdot A = r\cdot A + s\cdot A \\ & r\cdot(A+B) = r\cdot A + r\cdot B \\ iii) & 1\cdot A = A \end{array}$$

From which we can derive:

$$\begin{array}{ll} i) & 0\cdot A = 0 \\ ii) & r\cdot 0 = 0 \\ iii) & r\cdot A = 0 \;\Rightarrow r = 0 \,\vee\, A = 0 \\ iv) & (-1)\cdot A = -A \end{array}$$

Compatibility properties of matrix sum and product
Let $A, \tilde{A} \in \mathbb{F}^{m\times n}$, $B, \tilde{B} \in \mathbb{F}^{n\times l}$, $C \in \mathbb{F}^{l\times t}$, $r \in \mathbb{F}$. Then

$$\begin{array}{rl} i) & (A\cdot B)\cdot C = A\cdot(B\cdot C) \\ ii) & (A+\tilde{A})B = AB + \tilde{A}B \\ iii) & A(B+\tilde{B}) = AB + A\tilde{B} \\ iv) & I_m A = A I_n = A \\ v) & (r\cdot A)\cdot B = r(A\cdot B) = A(r\cdot B) \\ vi) & 0A = A0 = 0 \end{array}$$

Group property of invertible matrices
For two invertible matrices $A, B \in \mathbb{F}^{n\times n}$ we find

$$\begin{array}{rl} i) & (AB)^{-1} = B^{-1}A^{-1} \\ ii) & \left(A^{-1}\right)^{-1} = A \end{array}$$

Transpose matrices
Let $A \in \mathbb{R}^{m\times r}$ and $B \in \mathbb{R}^{r\times m}$. Then

i) $(A^\top)^\top = A$,

ii) $(AB)^\top = B^\top A^\top$,

iii) $(A+B)^\top = A^\top + B^\top$,

iv) for $A \in GL(n, \mathbb{R})$ we have $(A^\top)^{-1} = (A^{-1})^\top$.

Solution:

1 Linear Dependence

Give an example where a nontrivial combination of three nonzero vectors a_i in $\mathbb{R}^4$ is the zero vector (nontrivial means that not all scaling coefficients are zero). Write your example in the form $Ax = 0$. *(4 Points)*

Solution:

Take

$$A := [a_1, a_2, a_3] := \begin{pmatrix} 1 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix} \quad \text{and} \quad x := \begin{pmatrix} 1 \\ 1 \\ -1 \end{pmatrix}$$

Construction recipe: We take two random vectors x, y and form a simple linear combination $x + y$. We know these vectors are linear dependent, so that we can take those as columns:

$$x + y - (x + y) = 0 \quad \text{gives} \quad [x, y, (x + y)] \begin{pmatrix} 1 \\ 1 \\ -1 \end{pmatrix} = 0$$

2 A matrix as a Linear Mapping

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 given above.

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

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

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

(6 Points)

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 "$y = 0$").

3 The Subspaces Kernel and Image

Let $A \in \mathbb{F}^{m \times n}$. Show that $\ker(A)$ and $\operatorname{Im}(A)$ are subspaces of $\mathbb{F}^n$ and $\mathbb{F}^m$, respectively. *(6 Points)*

Solution:

To show:

1. $\ker(A) \subset \mathbb{F}^n$ subspace
2. $\operatorname{Im}(A) \subset \mathbb{F}^m$ subspace

Proof:

1. (a) $A \cdot 0 = 0 \in \ker(A)$, thus nonempty

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

   $$\Rightarrow \quad \lambda_1 v_1 + \lambda_2 v_2 \in \ker(A)$$

2. (a) $A \cdot 0 = 0 \in \operatorname{Im}(A)$, thus nonempty

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

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

4 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 nonzero vectors x and y, so that $Ax = 0$ and $Ay = 0$.
3. How does the image $\operatorname{Im}(A)$ look like? Characterize the set mathematically and also draw a picture. Find a basis of $\operatorname{Im}(A)$ and determine the rank of the matrix, i.e., $\operatorname{rank}(A)$.
4. How does the kernel $\ker(A)$ look like? Characterize the set mathematically and also draw a picture. Find a basis of $\ker(A)$ and determine its dimension.

(10 Points)

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

$$0 = Ax = a_1x_1 + a_2x_2 + a_3x_3 = a_1(x_1 + x_2 + x_3) \Leftrightarrow x_1 + x_2 + x_3 = 0.$$

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

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

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 $\dim(\ker(A)) = 2$.

5 Matrix Product as Sum of rank-1 Matrices

Let $A \in \mathbb{R}^{m \times k}$ and $B \in \mathbb{R}^{k \times n}$. Show that

$$A \cdot B = \sum_{i=1}^{k} a_i b_i^\top = \sum_{i=1}^{k} \begin{pmatrix} a_{1i} \\ \vdots \\ a_{mi} \end{pmatrix} \cdot \begin{pmatrix} b_{i1} & \dots & b_{in} \end{pmatrix},$$

where $a_i \in \mathbb{R}^{m \times 1}$ denotes the i-th *column* of A and $b_i^\top \in \mathbb{R}^{1 \times n}$ denotes the i-th *row* of B.
Remark: Also see p. 11 in Gilbert Strang's "Linear Algebra and Learning from Data". *(4 Points)*

Solution:

Note that by definition of the matrix product we have that the entry at (μ, ν) of AB is given by

$$(AB)_{\mu\nu} = \sum_{i=1}^{k} a_{\mu i} b_{i\nu}.$$

Again, by definition of the matrix product, for the i-th column $a_i = (a_{1i}, \ldots, a_{mi})^\top \in \mathbb{R}^{m\times 1}$ and i-th row $b_i^\top = (b_{i1}, \ldots, b_{in}) \in \mathbb{R}^{1\times n}$, we find

$$(a_i b_i^\top)_{\mu\nu} = \sum_{j=1}^{1} (a_i)_{\mu j} (b_i^\top)_{j\nu} = (a_i)_{\mu 1} (b_i^\top)_{1\nu} = a_{\mu i} b_{i\nu}.$$

Thus

$$\left(\sum_{i=1}^{k} a_i b_i^\top\right)_{\mu\nu} = \sum_{i=1}^{k} \left(a_i b_i^\top\right)_{\mu\nu} = \sum_{i=1}^{k} a_{\mu i} b_{i\nu} = (AB)_{\mu\nu}.$$

PROGRAMMING

6 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 returns the matrix-vector product Ax.

Implement the following ways of doing this:

1. **Dense:** Input expected as `numpy.ndarray`:
   Assume that the matrix and the vector are delivered to your function as `numpy.ndarray`.
   (a) Implement the matrix-vector product "by hand" using for loops, i.e., *without* using `numpy.dot(A,x)` (or `numpy.matmul(A,x)` or `A@x`).
   (b) Implement the matrix-vector product using `A.dot(x)`, `A@x`, `numpy.matmul(A,x)` or `numpy.dot(A,x)`.
2. **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`.
   (a) Access the three CSR lists via `A.data`, `A.indptr`, `A.indices` and implement the matrix-vector product "by hand" using for loops.
   (b) Implement the matrix-vector product using `A.dot(x)` or `A@x` .

Test your different routines on the matrix $A \in \mathbb{R}^{n\times n}$ given by

$$A = \begin{pmatrix} 2 & -1 & 0 & \cdots & 0 \\ -1 & 2 & -1 & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & -1 & 2 & -1 \\ 0 & \cdots & 0 & -1 & 2 \end{pmatrix}$$

and a random input vector `x = numpy.random.rand(n)`. Play around with the dimension n (especially large $n \geq 10^5$ – note that you may exceed your hardware capacities for the dense computations).

For all cases:

- **Memory:** A number implemented as `float` in Python implements double precision and therefore needs 64 Bits of storage. 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. 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`.

- **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)`. 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`.

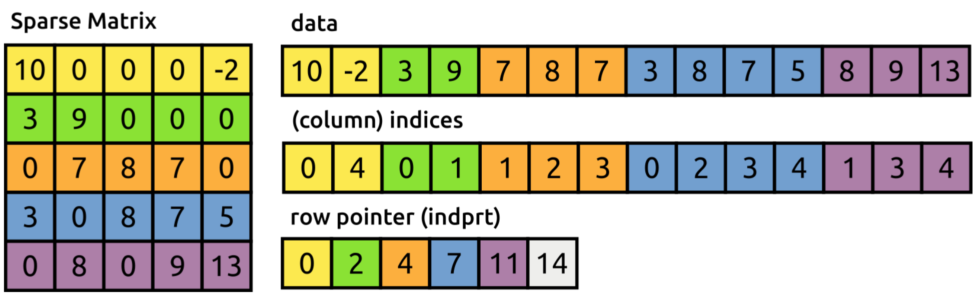

Figure 1: Example of a Matrix in CSR Format

(8 Points)

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

    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:
```

```python
        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
                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 numpy.ndarray

    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

    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
    # ---------------------------------------------------------------#
```

```python
# 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

# 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")
```

Total Number of Points = 38 (T:30, P:8)