

## Problem Sheet 1

You should try to your best to do the core problems. If time permits, try and do the rest as well.

### Problem 1. Arrow matrix-vector multiplication (core problem)

Consider the multiplication of the two “arrow matrices”  $A$  with a vector  $x$ , implemented as a function `arrowmatvec(d, a, x)` in the following MATLAB script

Listing 1: multiplying a vector with the product of two “arrow matrices”

```
1 function y = arrowmatvec(d, a, x)
2 % Multiplying a vector with the product of two ``arrow
   matrices''
3 % Arrow matrix is specified by passing two column
   vectors a and d
4 if (length(d) ~= length(a)), error('size mismatch'); end
5 % Build arrow matrix using the MATLAB function diag()
6 A = [diag(d(1:end-1)), a(1:end-1); (a(1:end-1))', d(end)];
7 y = A*A*x;
```

(1a) ☐ For general vectors  $d = (d_1, \dots, d_n)^\top$  and  $a = (a_1, \dots, a_n)^\top$ , sketch the matrix  $A$  created in line 6 of Listing 1.

HINT: This MATLAB script is provided as file `arrowmatvec.m`.

(1b) ☐ The `tic-toc` timing results for `arrowmatvec.m` are available in Figure 1. Give a detailed explanation of the results.

HINT: This MATLAB created figure is provided as file `arrowmatvectiming.{eps, jpg}`.

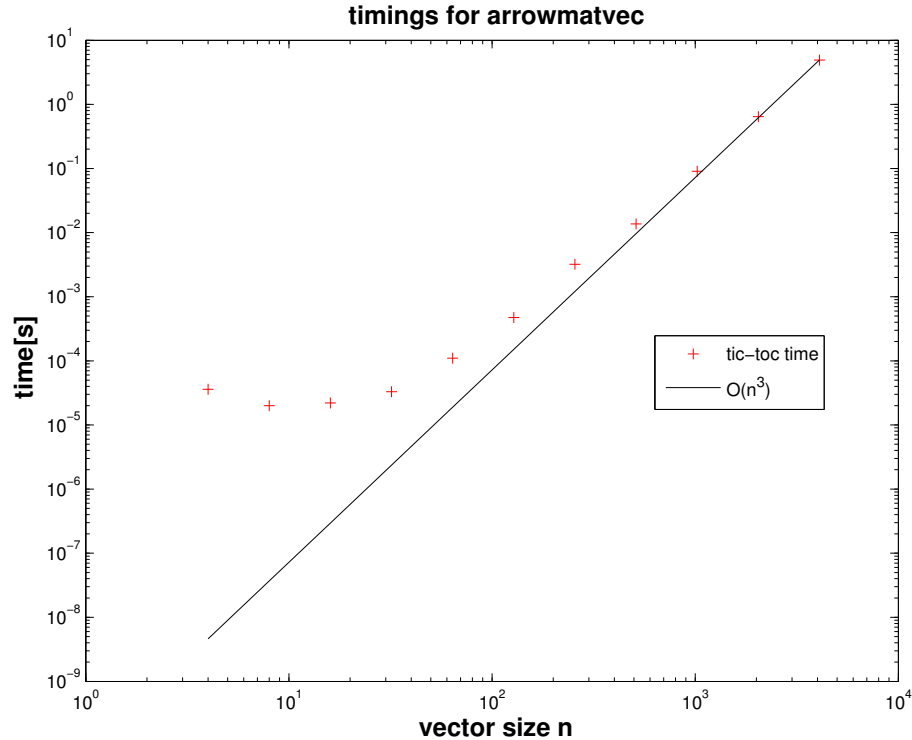


Figure 1: timings for `arrowmatvec(d, a, x)`

(1c) ☐ Write an *efficient* MATLAB function

```
function y = arrowmatvec2(d, a, x)
```

that computes the same multiplication as in code 1 but with optimal asymptotic complexity with respect to  $n$ . Here  $\mathbf{d}$  passes the vector  $(d_1, \dots, d_n)^T$  and  $\mathbf{a}$  passes the vector  $(a_1, \dots, a_n)^T$ .

(1d) ☐ What is the complexity of your algorithm from sub-problem (1c) (with respect to problem size  $n$ )?

(1e) ☐ Compare the runtime of your implementation and the implementation given in code 1 for  $n = 2^5, 6, \dots, 12$ . Use the routines `tic` and `toc` as explained in example [1, Ex. 1.4.10] of the Lecture Slides.

(1f) ☐ Write the EIGEN codes corresponding to the functions `arrowmatvec` and `arrowmatvec2`.

## Problem 2. Avoiding cancellation (core problem)

In [1, Section 1.5.4] we saw that the so-called *cancellation phenomenon* is a major cause of numerical instability, cf. [1, § 1.5.38]. Cancellation is the massive amplification of *relative errors* when subtracting two real numbers of about the same value.

Fortunately, expressions vulnerable to cancellation can often be recast in a mathematically equivalent form that is no longer affected by cancellation, see [1, § 1.5.43]. There we studied several examples, and this problem gives some more,

(2a) ◻ We consider the function

$$f_1(x_0, h) := \sin(x_0 + h) - \sin(x_0) . \quad (1)$$

It can be transformed into another form,  $f_2(x_0, h)$ , using the trigonometric identity

$$\sin(\varphi) - \sin(\psi) = 2 \cos\left(\frac{\varphi + \psi}{2}\right) \sin\left(\frac{\varphi - \psi}{2}\right) .$$

Thus,  $f_1$  and  $f_2$  give the same values, in exact arithmetic, for any given argument values  $x_0$  and  $h$ .

1. Derive  $f_2(x_0, h)$ , which does no longer involve the difference of return values of trigonometric functions.
2. Suggest a formula that avoids cancellation errors for computing the approximation  $(f(x_0 + h) - f(x_0))/h$  of the derivative of  $f(x) := \sin(x)$  at  $x = x_0$ . Write a MATLAB program that implements your formula and computes an approximation of  $f'(1.2)$ , for  $h = 1 \cdot 10^{-20}, 1 \cdot 10^{-19}, \dots, 1$ .  
HINT: For background information refer to [1, Ex. 1.5.40].
3. Plot the error (in doubly logarithmic scale using MATLAB's `loglog` plotting function) of the derivative computed with the suggested formula and with the naive implementation using  $f_1$ .
4. Explain the observed behaviour of the error.

(2b) ◻ Using a trick applied in [1, Ex. 1.5.48] show that

$$\ln(x - \sqrt{x^2 - 1}) = -\ln(x + \sqrt{x^2 - 1}) .$$

Which of the two formulas is more suitable for numerical computation? Explain why, and provide a numerical example in which the difference in accuracy is evident.

**(2c)** ☐ For the following expressions, state the numerical difficulties that may occur, and rewrite the formulas in a way that is more suitable for numerical computation.

1.  $\sqrt{x + \frac{1}{x}} - \sqrt{x - \frac{1}{x}}$ , where  $x \gg 1$ .

2.  $\sqrt{\frac{1}{a^2} + \frac{1}{b^2}}$ , where  $a \approx 0, b \approx 1$ .

### Problem 3. Kronecker product

In [1, Def. 1.4.16] we learned about the so-called Kronecker product, available in MATLAB through the command `kron`. In this problem we revisit the discussion of [1, Ex. 1.4.17]. Please refresh yourself on this example and study [1, Code 1.4.18] again.

As in [1, Ex. 1.4.17], the starting point is the line of MATLAB code

$$\mathbf{y} = \text{kron}(\mathbf{A}, \mathbf{B}) * \mathbf{x}, \quad (2)$$

where the arguments are  $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{n,n}, \mathbf{x} \in \mathbb{R}^{n \cdot n}$ .

**(3a)** ☐ Obtain further information about the `kron` command from MATLAB help issuing `doc kron` in the MATLAB command window.

**(3b)** ☐ Explicitly write Eq. (2) in the form  $\mathbf{y} = \mathbf{M}\mathbf{x}$  (i.e. write down  $\mathbf{M}$ ), for  $\mathbf{A} = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$  and  $\mathbf{B} = \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}$ .

**(3c)** ☐ What is the asymptotic complexity ( $\rightarrow$  [1, Def. 1.4.3]) of the MATLAB code (2)? Use the Landau symbol from [1, Def. 1.4.4] to state your answer.

**(3d)** ☐ Measure the runtime of (2) for  $n = 2^{3,4,5,6}$  and random matrices. Use the MATLAB functions `tic` and `toc` as explained in example [1, Ex. 1.4.10] of the Lecture Slides.

**(3e)** ☐ Explain in detail, why (2) can be replaced with the single line of MATLAB code

$$\mathbf{y} = \text{reshape}(\mathbf{B} * \text{reshape}(\mathbf{x}, n, n) * \mathbf{A}', n * n, 1); \quad (3)$$

and compare the execution times of (2) and (3) for random matrices of size  $n = 2^{3,4,5,6}$ .

**(3f)** ☐ Based on the EIGEN numerical library ( $\rightarrow$  [1, Section 1.2.2]) implement a C++ function

```

template <class Matrix>
void kron(const Matrix & A, const Matrix & B, Matrix &
    C) {
    // Your code here
}

```

returns the Kronecker product of the argument matrices A and B in the matrix C.

HINT: Feel free (but not forced) to use the partial codes provided in `kron.cpp` as well as the CMake file `CMakeLists.txt` (including `cmake-modules`) and the timing header file `timer.h`.

**(3g)** ☐ Devise an implementation of the MATLAB code (2) in C++ according to the function definition

```

template <class Matrix, class Vector>
void kron_mv(const Matrix & A, const Matrix & B, const
    Vector & x, Vector & y);

```

The meaning of the arguments should be self-explanatory.

**(3h)** ☐ Now, using a function definition similar to that of the previous sub-problem, implement the C++ equivalent of (3) in the function `kron_mv_fast`.

HINT: Study [1, Rem. 1.2.18] about “reshaping” matrices in EIGEN.

**(3i)** ☐ Compare the runtimes of your two implementations as you did for the MATLAB implementations in sub-problem (3e).

## Problem 4. Structured matrix–vector product

In [1, Ex. 1.4.14] we saw how the particular structure of a matrix can be exploited to compute a matrix-vector product with substantially reduced computational effort. This problem presents a similar case.

Consider the real  $n \times n$  matrix  $\mathbf{A}$  defined by  $(\mathbf{A})_{i,j} = a_{i,j} = \min\{i, j\}$ , for  $i, j = 1, \dots, n$ . The matrix-vector product  $\mathbf{y} = \mathbf{A}\mathbf{x}$  can be implemented in MATLAB as

$$\mathbf{y} = \min(\text{ones}(n, 1) * (1:n), (1:n)' * \text{ones}(1, n)) * \mathbf{x}; \quad (4)$$


**(4a)** ☐ What is the asymptotic complexity (for  $n \rightarrow \infty$ ) of the evaluation of the MATLAB command displayed above, with respect to the problem size parameter  $n$ ?


(4b)  Write an *efficient* MATLAB function


```
function y = multAmin(x)
```

that computes the same multiplication as (4) but with a better asymptotic complexity with respect to  $n$ .

HINT: you can test your implementation by comparing the returned values with the ones obtained with code (4).


(4c)  What is the asymptotic complexity (in terms of problem size parameter  $n$ ) of your function `multAmin`?

(4d)  Compare the runtime of your implementation and the implementation given in (4) for  $n = 2^5, 6, \dots, 12$ . Use the routines `tic` and `toc` as explained in example [1, Ex. 1.4.10] of the Lecture Slides.

(4e)  Can you solve task (4b) without using any `for`- or `while`-loop? Implement it in the function

```
function y = multAmin2(x)
```

HINT: you may use the MATLAB built-in command `cumsum`.


(4f)  Consider the following MATLAB script `multAB.m`:

Listing 2: MATLAB script calling `multAmin`


```
1 n = 10;
2 B = diag(-ones(n-1,1),-1)+diag([2*ones(n-1,1);1],0) ...
3     + diag(-ones(n-1,1),1);
4 x = rand(n,1);
5 fprintf('|x-y| = %d\n',norm(multAmin(B*x)-x));
```

Sketch the matrix **B** created in line 3 of `multAB.m`.

HINT: this MATLAB script is provided as file `multAB.m`.

(4g)  Run the code of Listing 2 several times and conjecture a relationship between the matrices **A** and **B** from the output. Prove your conjecture.


HINT: You must take into account that computers inevitably commit round-off errors, see [1, Section 1.5].

(4h)  Implement a C++ function with declaration

```
1 template <class Vector>
2 void minmatmv(const Vector &x, Vector &y);
```

that realizes the efficient version of the MATLAB line of code (4). Test your function by comparing with output from the equivalent MATLAB functions.

## Problem 5. Matrix powers


(5a)  Implement a MATLAB function

Pow(A, k)

that, using only basic linear algebra operations (including matrix-vector or matrix-matrix multiplications), computes efficiently the  $k^{\text{th}}$  power of the  $n \times n$  matrix **A**.

HINT: use the MATLAB operator `^` to test your implementation on random matrices **A**.

HINT: use the MATLAB functions `de2bi` to extract the “binary digits” of an integer.


(5b)  Find the asymptotic complexity in  $k$  (and  $n$ ) taking into account that in MATLAB a matrix-matrix multiplication requires a  $O(n^3)$  effort.

(5c)  Plot the runtime of the built-in MATLAB power (`^`) function and find out the complexity. Compare it with the function `Pow` from (5a).

Use the matrix

$$A_{j,k} = \frac{1}{\sqrt{n}} \exp\left(\frac{2\pi i j k}{n}\right)$$

to test the two functions.

(5d)  Using EIGEN, devise a C++ function with the calling sequence

```
1 template <class Matrix>
2 void matPow(const Matrix &A, unsigned int k);
```

that computes the  $k^{\text{th}}$  power of the square matrix **A** (passed in the argument **A**). Of course, your implementation should be as efficient as the MATLAB version from sub-problem (5a).

HINT: matrix multiplication suffers no aliasing issues (you can safely write  $A = A \star A$ ).

HINT: feel free to use the provided `matPow.cpp`.

HINT: you may want to use `log` and `ceil`.

HINT: EIGEN implementation of power (`A.pow(k)`) can be found in:

```
#include <unsupported/Eigen/MatrixFunctions>
```

## Problem 6. Complexity of a MATLAB function

In this problem we recall a concept from linear algebra, the diagonalization of a square matrix. Unless you can still define what this means, please look up the chapter on “eigenvalues” in your linear algebra lecture notes. This problem also has a subtle relationship with Problem 5.

We consider the MATLAB function defined in `getit.m` (cf. Listing 3)

Listing 3: MATLAB implementation of `getit` for Problem 6..

```
1 function y = getit(A, x, k)
2     [S,D] = eig(A);
3     y = S*diag(diag(D).^k)*(S\'x);
4 end
```

HINT: Give the command `doc eig` in MATLAB to understand what `eig` does.

HINT: You may use that `eig` applied to an  $n \times n$ -matrix requires an asymptotic computational effort of  $O(n^3)$  for  $n \rightarrow \infty$ .

HINT: in MATLAB, the function `diag(x)` for  $x \in \mathbb{R}^n$ , builds a diagonal,  $n \times n$  matrix with  $x$  as diagonal. If  $M$  is a  $n \times n$  matrix, `diag(M)` returns (extracts) the diagonal of  $M$  as a vector in  $\mathbb{R}^n$ .

HINT: the operator  $v.^k$  for  $v \in \mathbb{R}^n$  and  $k \in \mathbb{N} \setminus \{0\}$  returns the vector with components  $v_i^k$  (i.e. component-wise exponent)

(6a) ☐ What is the output of `getit`, when  $A$  is a diagonalizable  $n \times n$  matrix,  $x \in \mathbb{R}^n$  and  $k \in \mathbb{N}$ ?

(6b) ☐ Fix  $k \in \mathbb{N}$ . Discuss (in detail) the asymptotic complexity of `getit`  $n \rightarrow \infty$ .



Issue date: 17.09.2015

Hand-in: 24.09.2015 (in the boxes in front of HG G 53/54).

Version compiled on: September 18, 2015 (v. 1.1).

## References

- [1] R. Hiptmair. *Lecture slides for course "Numerical Methods for CSE"*.  
<http://www.sam.math.ethz.ch/~hiptmair/tmp/NumCSE/NumCSE15.pdf>. 2015.