V. Schulz
M. Klar

SS 2022
Submission: 17.05.21, until 12:15

Numerical Optimization - Sheet 6

If you are a student in mathematics please solve the exercises with no tag and the ones with the tag **Mathematics**. If you are a data science student please solve the problems with no tag and those with the tag **Data Science**. Submissions with tags other than your subject count as bonus points. The tag **Programming** marks programming exercises.

Ex 1 (6 Points)

(i) Let $Q \in \mathbb{R}^{n\times n}$ be symmetric and positive definite. Assume you want to solve

$$\min_{x\in\mathbb{R}} x^\top Qx + x^\top b \tag{1}$$

using a Jacobi preconditioned conjugate gradient method. Is $diag(Q)^{-1}$ always well defined? Give a short justification.

(ii) Let $D \in \mathbb{R}^{n\times n}$ be invertible. Assume you are given the solution of the problem

$$\min_{z\in\mathbb{R}} z^\top D^\top QDz + z^\top D^\top b. \tag{2}$$

Derive the solution x of (1).

(iii) Let $Q := diag(\{1,\dots,n\}) \in \mathbb{R}^{n\times n}$ for some $n \in \mathbb{N}$. Compute the condition numbers of Q and $D^\top QD$ for $D := \sqrt{diag(Q)^{-1}}$.

Solution 1:

(i) Yes. As Q is positive definite we obtain that $q_{ii} = e_i^\top Qe_i > 0$ for all unit vectors e_i, $i = 1,\dots,n$.

(ii) The solution is obtained from $x := zD$. Note that, while the above is equivalent to a Jacobi preconditioned CG this is not the way preconditioned CG is implemented. PCG can be achieved more efficiently by making some changes in the algorithm - which saves evaluations of the preconditioner.

(iii) As Q is symmetric and positve definite, the norm of Q is equal to its largest Eigenvalue which is n. We obtain that the norm of $Q^{-1} = diag(\frac{1}{q_{ii}})$ is given by 1 for the same reason. The condition number of Q is therefore

$$cond(Q) = n.$$

The matrix $D^\top QD = id_n$ has condition number 1. Of course the solution of a linear system is trivial now. The important effect here is the clustering of Eigenvalues though, which also appears in more general cases, e.g. in a discretization of a differential equation on an irregular grid.

Ex 2 **Data Science** (4 Points)

For the application of many iterative algorithms like the CG-iteration there is no need to store the linear system itself, but one only requires to evaluate matrix vector products. This can be an advantage not only in speed but even in terms of feasibility.
Let $Q \in \mathbb{R}^{n\times n}$ and $x \in \mathbb{R}^n$. Solve tasks a) and b) for each of the matrix classes given below in i), ii).

a) Find an algorithm which performs the matrix-vector multiplication Qx and exploits the matrix structures given below. Describe your algorithm with the help of a pseudo code.

b) Name the number of floating point operations (summation and multiplication) you have to perform and the number of reals you need to store.

Your are given the following matrices.

i) The matrix Q attains

$$Q = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 2 & 0 & \dots & 0 \\ 0 & 0 & 3 & \dots & 0 \\ 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & \dots & n \end{bmatrix}.$$

ii) The matrix Q is of the form

$$q_{ij} = \begin{cases} \frac{2}{(n-1)^2} & \text{if } i = j \\ -\frac{1}{(n-1)^2} & \text{if } |i-j| = 1 \\ 0 & \text{else.} \end{cases}$$

Solution 2:

(i) **Algorithm**

```
allocate double x[n]

for i in 1 to n:
  x[i] = x[i]*i
```

Answer
The algorithm needs $n = O(n)$ multiplications and stores $n = O(n)$ doubles. The input is overwritten here.

(ii) **Algorithm**

```
allocate double x[n]
allocate double b[n]

h = 1/(n-1)**2
# 4 Multiplications
# 2 Summations
b[1] = h(2x[1] - x[2])
b[n] = h(-x[n-1] + 2x[n])

for i in 2 to n-1:
  # 2x(n-2) Multiplication
  # 2x(n-2) Summations
  b[i] = h(2x[i] - x[i-1] - x[i+1])
```

Answer

The algorithm needs $6 + 4 * (n-2) = O(n)$ floating point operations and needs to store $2n = O(n)$ doubles.

Ex 3 **Programming** (12 Points)

Download the module `cg_test` and `numopt_python_scripts_full.ipynb`.

(i) Test the (preconditioned) conjugate gradient implementation in `numopt_python_scripts_full.ipynb` on problem (1) and (2) of **Ex 1** for Q defined as in (i) and (ii) of **Ex 2**, $D := \sqrt{diag(Q)^{-1}}$, $b = \mathbf{1} \in \mathbb{R}^n$, and $n = 10, 50, 250, 1250$.

(ii) Use the CG-algorithm to solve problem (1) where

$$Q = \begin{pmatrix} 1 & -1 & 0 & 0 & 0 & \dots \\ -1 & 2 & -1 & 0 & 0 & \dots \\ 0 & -1 & 2 & -1 & 0 & \dots \\ & & \ddots & \ddots & \ddots & \\ & \dots & 0 & -1 & 2 & -1 \\ & \dots & 0 & 0 & -1 & 1 \end{pmatrix} \in \mathbb{R}^{50.000 \times 50.000}$$

and $b = (1, -1, 1, -1, \dots) \in \mathbb{R}^{50.000}$. Which special feature does the matrix have? Why does the algorithm still converge?

(iii) Implement a function `nlcg(f, x, tol=1e-9, max_it = 1000)` which minimizes `f` by applying a version of the nonlinear CG-Algorithm. This happens in the following steps.

- Replace the stepsize α by a line search. The stepsize has to be chosen such that

$$\nabla f(x + \alpha p)^\top p = 0.$$

  You can use the function `root_scalar()` of the module `scipy.optimize` to solve this equation. Furthermore the function `grad` is provided in `cg_test`.
- Replace the residual r by the gradient $\nabla f(x)$.
- Replace β by $\max\{\beta^{PR}, 0\}$ where β^{PR} is given by the Polak-Ribiere formula.

Test the function with the help of `test(nlcg)` of the module `cg_test`.

Hints:

- The preconditioning of problem (1) by D should show no effect on the CG-algorithm for Q as in **Ex 2** (ii), and a very big effect for Q as in **Ex 2** (i).
- It is not a good idea to set up a dense array of the size 50.000×50.000. You'd better implement the matrix vector product $Q(x)$ and alter the CG-algorithm accordingly.
- It might help you to read the section about nonlinear CG in the document "An Introduction to the Conjugate Gradient Method Without the Agonizing Pain".