CDS: Numerical Methods - Assignment week 3

Solutions to the exercise have to be handed in via Brightspace in form of one or several executable Python scripts (*.py) which run without any errors. The deadline for the submission is **Monday Feb. 24, 13:30**. Feel free to use the Science Gitlab repository to submit your solutions.

1 Linear Equation Systems

In the following you will implement the Jacobi, Steepest Decent, and the Conjugate Gradient algorithm to solve linear equation systems of the form

$$A\mathbf{x} = \mathbf{b}$$

with A being a $n \times n$ matrix.

(a) First, you need to implement a Python function `d = diff(a,b)` which calculates and returns the difference `d` between two n-dimensional vectors `a` and `b` according to

$$d = ||\mathbf{a} - \mathbf{b}||_\infty = \max_{i=1,2,\ldots,n} |a_i - b_i|.$$

(b) Implement the Jacobi iteration scheme

$$\mathbf{x}^{(k)} = D^{-1}(L+U)\mathbf{x}^{(k-1)} + D^{-1}\mathbf{b}$$

in a Python function `x, k = jacobi(A, b, eps)`, where $A = D - L - U$ has been separated into its diagonal (D), lower triangular (L) and its upper triangular (U) form, `A` represents the $n \times n$ A matrix, `b` represents the n-dimensional solution vector $\mathbf{b}$, and `eps` is a scalar ε defining the accuracy up to which the iteration is performed. k is the iteration index. Initialize your iteration with $\mathbf{x}^{(0)} = \mathbf{0}$ (or with $\mathbf{x}^{(1)} = D^{-1}\mathbf{b}$;-)) and increase k until $||\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}||_\infty < \varepsilon$. Your function should return both, the solution vector $\mathbf{x}^{(k)}$ (from the last iteration step) and the last iteration index k. Hint: Use `numpy.dot()` for all needed matrix/vector products.

(c) Verify your implementation by comparing to the exact result for $\mathbf{x}^*$ for the system

$$\begin{pmatrix} 10 & -1 & 2 & 0 \\ -1 & 11 & -1 & 3 \\ 2 & -1 & 10 & -1 \\ 0 & 3 & -1 & 8 \end{pmatrix} \mathbf{x}^* = \begin{pmatrix} 6 \\ 25 \\ -11 \\ 15 \end{pmatrix}$$

obtained with `numpy.linalg.solve()` with your approximate result $\tilde{\mathbf{x}}$. Calculate $||\mathbf{x}^* - \tilde{\mathbf{x}}||_\infty$ for different accuracies $\varepsilon = 0.1, 0.01, 0.001, 0.0001$.

(d) Next, implement the Steepest Decent algorithm in a similar Python function `x, k = SD(A, b, eps)` which calculates

$$\begin{aligned} \mathbf{v}^{(k)} &= \mathbf{b} - A\mathbf{x}^{(k-1)} \\ t_k &= \frac{\langle \mathbf{v}^{(k)}, \mathbf{v}^{(k)} \rangle}{\langle \mathbf{v}^{(k)}, A\mathbf{v}^{(k)} \rangle} \\ \mathbf{x}^{(k)} &= \mathbf{x}^{(k-1)} + t_k \mathbf{v}^{(k)}. \end{aligned}$$

Again, initialize your iteration with $\mathbf{x}^{(0)} = \mathbf{0}$ and increase k until $||\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}||_\infty < \varepsilon$. Return the solution vector $\mathbf{x}^{(k)}$ (from the last iteration step) and the last iteration index k. Use `numpy.dot()` for all needed vector/vector and matrix/vector products. Verify your implementation again by comparing to the exact solution for the linear equation system given in (c).

(e) Finally, based on your steepest decent implementation from (d) implement the Conjugate Gradient algorithm in a Python function `x, k = CG(A, b, eps)` in the following way:

Initialize your procedure with:

$$
\begin{aligned}
\mathbf{x}^{(0)} &= \mathbf{0} \\
\mathbf{r}^{(0)} &= \mathbf{b} - A\mathbf{x}^{(0)} \\
\mathbf{v}^{(0)} &= \mathbf{r}^{(0)}
\end{aligned}
$$

Than repeat by increasing $k = 0, 1, \dots$:

$$
\begin{aligned}
t_k &= \frac{\langle \mathbf{r}^{(k)}, \mathbf{r}^{(k)} \rangle}{\langle \mathbf{v}^{(k)}, A\mathbf{v}^{(k)} \rangle} \\
\mathbf{x}^{(k+1)} &= \mathbf{x}^{(k)} + t_k \mathbf{v}^{(k)} \\
\mathbf{r}^{(k+1)} &= \mathbf{r}^{(k)} - t_k A\mathbf{v}^{(k)} \\
s_k &= \frac{\langle \mathbf{r}^{(k+1)}, \mathbf{r}^{(k+1)} \rangle}{\langle \mathbf{r}^{(k)}, \mathbf{r}^{(k)} \rangle} \\
\mathbf{v}^{(k+1)} &= \mathbf{r}^{(k+1)} - s_k \mathbf{v}^{(k)}
\end{aligned}
$$

until $||\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}||_\infty < \varepsilon$. Return the solution vector $\mathbf{x}^{(k)}$ (from the last iteration step) and the last iteration index k. Use `numpy.dot()` for all needed vector/vector and matrix/vector products. Verify your implementation again by comparing to the exact solution for the linear equation system given in (c). What do expect / see regarding the accuracy of your conjugate gradient function and the number of needed iteration steps?

(f) Apply all three methods to the following system

$$
\begin{pmatrix}
0.2 & 0.1 & 1.0 & 1.0 & 0.0 \\
0.1 & 4.0 & -1.0 & 1.0 & -1.0 \\
1.0 & -1.0 & 60.0 & 0.0 & -2.0 \\
1.0 & 1.0 & 0.0 & 8.0 & 4.0 \\
0.0 & -1.0 & -2.0 & 4.0 & 700.0
\end{pmatrix}
\mathbf{x}^* =
\begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{pmatrix}.
$$

Plot the number of needed iterations for each method as a function of ε using $\varepsilon = 0.1, 0.01, 0.001, 0.0001,$ $0.00001, 0.000001, 0.0000001, 0.00000001$. Explain the behavior you observe with the help of the condition number, which you can calculate with `numpy.linalg.cond()`.

(g) (optional) Try to get a better convergence behavior by pre-conditioning your matrix A using

$$\tilde{A} = CAC$$

instead, where $C = \sqrt{D^{-1}}$. If you do so, you will need to replace $\mathbf{b}$ by

$$\tilde{\mathbf{b}} = C\mathbf{b}$$

and your result will not be $\mathbf{x}$ but

$$\tilde{\mathbf{x}} = C\mathbf{x}.$$

What is the effect of C to the condition number and why is that so?