Solution 7 – MATH-250 Advanced Numerical Analysis I

Problem 4. ($\star$)

Let $A \in \mathbb{R}^{n\times n}$ be an invertible matrix, and $u, v \in \mathbb{R}^n$ be two vectors.

(a) Show that the *Sherman-Morrison formula*

$$\left(A + uv^\top\right)^{-1} = A^{-1} - \frac{1}{1 + v^\top A^{-1} u} A^{-1} u v^\top A^{-1}$$

holds true.

(b) Given that $v^\top A^{-1} u = -1$ holds, argue whether or not $M = A + uv^\top$ can be invertible. If you find that M can be invertible, give an example for A, u, and v. Otherwise, prove that M cannot be invertible.

(c) Let A be the tridiagonal matrix with diagonals $(a_1, a_2, \ldots, a_n)$ and $(b_1, b_2, \ldots, b_{n-1})$

$$A = \begin{pmatrix} a_1 & b_1 & & & & \\ b_1 & a_2 & b_2 & & & \\ & b_2 & a_3 & b_3 & & \\ & & \ddots & \ddots & \ddots & \\ & & & b_{n-2} & a_{n-1} & b_{n-1} \\ & & & & b_{n-1} & a_n \end{pmatrix},$$

and suppose that A possesses a unique LU factorisation $A = LU$. From Algorithm 4.9 one can verify that L and U are given by

$$L = \begin{pmatrix} 1 & & & & & \\ \ell_1 & 1 & & & & \\ & \ell_2 & 1 & & & \\ & & \ddots & \ddots & & \\ & & & \ell_{n-2} & 1 & \\ & & & & \ell_{n-1} & 1 \end{pmatrix}, \quad U = \begin{pmatrix} u_1 & b_1 & & & & \\ & u_2 & b_2 & & & \\ & & u_3 & b_3 & & \\ & & & \ddots & \ddots & \\ & & & & u_{n-1} & b_{n-1} \\ & & & & & u_n \end{pmatrix}$$

with the individual entries $u_1 = a_1,\ \ell_k = \frac{b_k}{u_k},\ u_k = a_k - \ell_{k-1} b_{k-1},\ k > 1$. Hence, computing the LU factorisation of A only requires $\mathcal{O}(n)$ arithmetic operations. Furthermore, forwards and backwards substitution in Algorithms 4.4 and 4.3, respectively, also require $\mathcal{O}(n)$ operations each. In total, we can therefore solve the system $Ax = b$ in $\mathcal{O}(n)$ arithmetic operations.

Using the facts above, give an algorithm which solves the system $Cx = b$ in $\mathcal{O}(n)$ operations for a vector $b \in \mathbb{R}^n$ and the matrix $C \in \mathbb{R}^{n\times n}$ given by

$$C = \begin{pmatrix} \alpha_1 & \beta_1 & & & & \beta_n \\ \beta_1 & \alpha_2 & \beta_2 & & & \\ & \beta_2 & \alpha_3 & \beta_3 & & \\ & & \ddots & \ddots & \ddots & \\ & & & \beta_{n-2} & \alpha_{n-1} & \beta_{n-1} \\ \beta_n & & & & \beta_{n-1} & \alpha_n \end{pmatrix}.$$

(d) Let's assume that $b = [1, 1, \ldots, 1]^\top$ is given and that the values of C satisfy

$$\alpha_1 = \alpha_n = -2, \quad \alpha_2 = \alpha_3 = \cdots = \alpha_{n-1} = -4, \quad \beta_1 = \beta_2 = \cdots = \beta_n = 2.$$

Implement a Python function `solve(n)` for the algorithm you developped in task (c) that returns the solution x of the linear system and the relative error $\frac{\|Cx-b\|_2}{\|b\|_2}$. For $n = 10$, print the output x of your algorithm and the relative error $\frac{\|Cx-b\|_2}{\|b\|_2}$. For $n \in$ `np.logspace(2, 7, num=6, dtype=int)`, plot the elapsed computational times of your algorithm and the relative errors depending on n in a doubly logarithmic plot with `plt.loglog` (or `matplotlib.pyplot.loglog` if you are not using the Jupyter notebook provided on Moodle).

Hints:

(i) This code will be excessively slow if you use NumPy to construct the matrix. Instead, use SciPy's `sparse` submodule. We recommend using `sps.diags_array` and `sps.coo_array` (or instead use, respectively, `scipy.sparse.diags_array` and `scipy.sparse.coo_array` if you are not using the Jupyter notebook we provided on Moodle).

(ii) SciPy's `sparse` submodule has a few ways of solving sparse linear systems. Use SciPy `sparse`'s `linalg.splu` to compute the LU decomposition and solve the linear system, or `scipy.sparse.linalg.splu` if you are not using the Jupyter notebook provided on Moodle. This function returns an object `lu = sps.linalg.splu(A)` with a `solve` function such that `x = lu.solve(b)` returns the solution of the linear system.

(iii) When assembling the $uv^\top$ matrix, make sure you use a sparse matrix and not a dense NumPy array.

Solution.

(a) Let us first verify that $A^{-1} - \frac{1}{1+v^\top A^{-1}u} A^{-1}uv^\top A^{-1}$ is the right inverse of $A + uv^\top$. We find that

$$(A + uv^\top)A^{-1} = \mathrm{id} + uv^\top A^{-1}, \tag{1}$$

and

$$\begin{aligned}(A + uv^\top)\frac{1}{1+v^\top A^{-1}u} A^{-1}uv^\top A^{-1} &= \frac{uv^\top A^{-1} + uv^\top A^{-1}uv^\top A^{-1}}{1+v^\top A^{-1}u} \\ &= \frac{u(1+v^\top A^{-1}u)v^\top A^{-1}}{1+v^\top A^{-1}u} \\ &= uv^\top A^{-1}.\end{aligned} \tag{2}$$

Finally, subtraction of (2) from (1) yields the desired result.

In an anaologous manner it can be proven that $A^{-1} - \frac{1}{1+v^\top A^{-1}u} A^{-1}uv^\top A^{-1}$ is also the left inverse of $A + uv^\top$.

(b) We are going to prove that if $v^\top A^{-1}u = -1$, then $A + uv^\top$ is not invertible.

Suppose that $v^\top A^{-1}u = -1$. This means, in particular, $u \neq 0$. Moreover, define $x = A^{-1}u$, and therefore $v^\top x = -1$. Hence we can see that $(A + uv^\top)x = Ax - u =$

$u - u = 0$. This shows that x is an element of the kernel of $A + uv^\top$. By assumption, $v^\top x = -1$ wherefore $x \neq 0$, resulting in the fact that the kernel of $A + uv^\top$ is nontrivial, and the matrix itself thus not invertible.

(c) Observe that we can write C as $A + uv^\top$ with $u = v = (1, 0, \ldots, 0, \beta_n)^\top$ and

$$A = \begin{pmatrix} \alpha_1 - 1 & \beta_1 & & & \\ \beta_1 & \alpha_2 & \beta_2 & & \\ & \ddots & \ddots & \ddots & \\ & & \beta_{n-1} & \alpha_{n-1} & \beta_{n-1} \\ & & & \beta_{n-1} & \alpha_n - \beta_n^2 \end{pmatrix}.$$

This means we can use the Sherman-Morrison formula to compute $x = C^{-1}b$. We propose Algorithm 1 to solve the system. Alternatively, you can explicitly compute the L and U factors of B and perform forward and backward substitution.

```
Algorithm 1: Sherman-Morrison Linear System Solution
1 Compute the LU factorisation of $A$ — this requires $\mathcal{O}(n)$ operations;
2 Solve $y = A^{-1}b$ and $z = A^{-1}u$ using the LU decomposition — forward and backward
    substitution require $\mathcal{O}(n)$ each;
3 Set $c = 1 + v^\top z$ — this requires $\mathcal{O}(n)$ operations;
4 Calculate $r = v^\top y$ and $\tilde{z} = z \cdot r$ — we can directly compute $\tilde{z} = (y_1 + \beta_n y_n)z$ which
    requires $\mathcal{O}(n)$ operations;
5 Finally, $x = y - \frac{\tilde{z}}{c}$ is the solution — this requires $\mathcal{O}(n)$ operations;
```

(d) Available in the Jupyter notebook `homework07-sol.ipynb` on Moodle.