MA398 Matrix Analysis and Algorithms: Exercise Sheet 9

1. (SD example) We consider the steepest decent method applied to the following data: For a real $a \gg 1$

$$A = \begin{pmatrix} 1 & 0 \\ 0 & a \end{pmatrix}, \quad b = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \quad x_0 = \begin{pmatrix} a \\ 1 \end{pmatrix}$$

   (a) Show that the even iterates are

$$x^{(2m)} = \Big(\frac{a-1}{a+1}\Big)^{2m} x_0, \quad m \in \mathbb{N},$$

   and find a formula for the odd iterates.
   Compute the $\alpha^{(k)}$ and the residuals $r^{(k)}$, $k \in \mathbb{N}$.

   (b) Show that subsequent search directions are 'almost' parallel in the metric defined by A: Use the formula

$$\cos(\phi) = \frac{\langle x, y\rangle_A}{\|x\|_A \|y\|_A}$$

   to measure the angle ϕ between to subsequent search directions and check that ϕ tends to π as $a \to \infty$.

   Remark: In contrast, the search directions in the CG method are orthogonal with respect to $\langle \cdot, \cdot \rangle_A$ which results in far better convergence properties.

2. Implement the Conjugate Gradient Method for solving linear systems. Given a positive definite matrix A and a vector b, the task is to find a vector x such that $Ax = b$. Use the pseudo code provided in the lecture notes as the base for your implementation.

   The function conjugate_gradient(A, b, x0, eps) takes as input a positive definite matrix A, a vector b, an initial guess for the solution x0, and a tolerance eps. It returns the solution to the linear system $Ax = b$ found by the Conjugate Gradient Method.

   Note: Be sure to implement checks for whether A is symmetric and positive definite. In practice, the Conjugate Gradient Method is usually applied to large, sparse systems, and more efficient implementations would take advantage of this.

3. Implement a Python function that computes the Krylov subspace $\mathcal{K}_k(r^{(0)}, A)$ for a given matrix A and vector $r^{(0)}$, where k is the number of iterations. The function should return a list of vectors spanning the Krylov subspace. You can use the numpy library for computations involving vectors and matrices. Here is a template to start with:

```
import numpy as np

def compute_krylov_subspace(r0, A, k):
    # Your code here
    pass
```

   Test your function with a random symmetric positive-definite matrix A and a random vector $r^{(0)}$, and for a few different values of k. Verify that the dimension of the Krylov subspace is as expected. You can use the numpy.linalg.matrix_rank function to compute the dimension of the subspace spanned by a list of vectors.