1 Appetizer: Gradient, Steepest Descent and Conjugate Gradient Method

Background

Let $A \in \mathbb{R}^{n\times n}$ be symmetric and positive definite (spd). Then A is in particular invertible, so that the linear system $Ax = b$ has a unique solution $x^* \in \mathbb{R}^n$ for all $b \in \mathbb{R}^n$. Let us relate this linear system to an optimization problem. For this purpose we define for a fixed spd matrix A and fixed right-hand side b the function

$$f := f_{A,b} : \mathbb{R}^n \to \mathbb{R}, x \to \tfrac{1}{2}x^T Ax - b^T x.$$

Then one can show the equivalence

$$Ax^* = b \quad \iff \quad x^* = \arg\min_{x\in\mathbb{R}^n} f(x).$$

In words, x^* solves the linear system on the left-hand side if and only if x^* is the unique minimizer of the functional f. In fact, you will learn in the next semester that the condition $Ax^* = b$ is the necessary first-order optimality condition:

$$0 = \nabla f(x) = Ax - b.$$

Due to the convexity of f this condition is also sufficient. Consequently, solving linear systems which involve spd matrices is equivalent to solving the associated optimization problem above, i.e., minimizing the function $f(x) = \frac{1}{2}x^T Ax - b^T x$. Thus, in this context iterative methods for linear systems, such as the Richardson iteration, can also be interpreted as optimization algorithms. Let us consider the (relaxed) Richardson iteration for $Ax = b$, i.e., $x_{k+1} = (I - \theta A)x_k + \theta b$. After some minor manipulations and making use of $\nabla f(x_k) = Ax_k - b$ we arrive at the equivalent formulation

$$x_{k+1} = x_k - \theta\nabla f(x_k).$$

The latter is what is called a gradient method. A step from x_k into (an appropriately scaled) direction of the gradient $\nabla f(x_k)$ yields a decrease in the objective function f , i.e., $f(x_{k+1}) \leq f(x_k)$. Along the Richardson/ Gradient method the scaling (also called step size) θ is fixed. However, one could also choose a different θ_k in each iteration step. This gives the more general version

$$x_{k+1} = x_k - \theta_k\nabla f(x_k). \tag{1}$$

The well known method of steepest descent is given by choosing

$$\theta_k = \frac{r_k^\top r_k}{r_k^\top A r_k}, \tag{2}$$

where $r_k := Ax_k - b$ is the k-th residual. This choice can be shown to be optimal in terms of convergence speed. Even general, one can think of using a different preconditioner N_k in each iteration step – this will later correspond to Newton-type optimization algorithms.

Task

Consider the following setting:

$$A = \begin{bmatrix} 2 & 0 \\ 0 & 10 \end{bmatrix}, \quad b = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad x_0 = \begin{bmatrix} 4 \\ 1.4 \end{bmatrix}.$$

Convince yourself that A is spd. Determine the minimal and maximal eigenvalue of A, i.e., λ_{min} and λ_{max}, respectively. What is the solution to $Ax = b$? Now extend your code (in particular `iter_solve()`) from previous sheets:

1. Implement the **steepest descent method** (1) by choosing the stepsize θ_k from (2) in each iteration step.
2. Find a way to apply the **conjugate gradient method** to solve a system $Ax = b$, where A is spd.
   Hint: You can either implement it on your own or find a SciPy routine (for the latter: you can collect all iterates x_k by using the callback interface).
3. **Test:** Solve the above problem with the following routines:
   a) Richardson method with $\theta = \frac{2}{\lambda_{\text{max}}}$

b) Richardson method with $\theta = 0.9 \cdot \frac{2}{\lambda_{\max}}$

c) Richardson method with optimal $\theta = \frac{2}{\lambda_{\min}+\lambda_{\max}}$

d) Steepest descent method

e) conjugate gradient method

Generate the following two plots:

1. Plot the iterates x_k for all the runs into the same 2d plot (use different colors).
2. Plot the function values $f(x_k) = \frac{1}{2}x_k^T A x_k - b^T x_k$ for each iterate and all runs into a second plot (use different colors).

Solution:

```python
# -*- coding: utf-8 -*-
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.patches import Ellipse
import importlib

# import iterative solvers from previous sheets
iterSolver = importlib.import_module("prog-LinearIterations_JacRich_solution")


if __name__ == "__main__":
    # ---------------
    # PROBLEM SETUP
    # ---------------
    eigmin, eigmax = 2, 10
    A = np.array([[eigmin, 0], [0, eigmax]])
    b = np.array([0, 0])
    x = np.array([0, 0])
    x0 = np.array([4, 1.4])

    theta_max = 2./eigmax
    theta_bad = 0.99 * theta_max
    theta_opt = 2 / (eigmin+eigmax)

    # ---------------
    # METHOD SETUP
    # ---------------
    maxiter = 200
    legend = ["Richardson max", "Richardson bad", "Richardson opt",
              "Steepest Descent", "Conjugate Gradient"]
    methods = ["Richardson", "Richardson", "Richardson",
               "steepestDescent", "CG"]
    theta = [theta_max, theta_bad,  theta_opt,
             -1, -1]
    colors = ["y", "c", "b",
              "g", "r"]
    numberRuns = len(methods)

    # ---------------
    # SOLVE
    # ---------------
    X = []
    F = []
    for i, method in enumerate(methods):
```

```
        _X = iterSolver.main(A, b, x0, maxiter,
                             {method: theta[i]}, plot=0, verbose=0)[0]
        X += [np.array(_X)]
        F += [[0.5 * np.dot(x, A.dot(x)) for x in _X]]

    # ------------- #
    # plot iterates
    # ------------- #
    fig, ax = plt.subplots()
    for i, _X in enumerate(X):
        plt.plot(_X[:, 0], _X[:, 1], colors[i]+"o-")
    # add level sets
    m = 200
    for k in range(m):
        e = Ellipse(xy=np.zeros(2), width=eigmax*0.8**k,
                    height=eigmin*0.8**k, angle=0, fill=False)
        ax.add_artist(e)
    plt.legend(legend)
    plt.axis('equal')
    plt.show()
    # -------------- #
    # plot objective
    # -------------- #
    plt.figure("objective-function")
    for i, f in enumerate(F):
        plt.plot(f, colors[i]+"x-")
    plt.legend(legend)
    plt.show()
```