Assignment 5

1. (8) Consider minimization of the quadratic form

$$f(x) = \frac{1}{2}x^T A x - b^T x + c, \tag{1}$$

where A is symmetric positive definite, by the steepest descent method.

- Show that optimal x_* is delivered by solution to equation $Ax_* = b$.
- Show that the steepest descent iteration has the form of

$$x_{i+1} = x_i + \alpha_i r_i, \tag{2}$$

where r_i is the residual at i-th iteration, $r_i = b - Ax_i$, and derive expression for the step size α_i in terms of r_i and A.

- Generate a 2D plot, illustrating the convergence of the steepest descent method for the matrix $A = \begin{bmatrix} 3 & 2 \\ 2 & 6 \end{bmatrix}$, right–hand–side $b = \begin{bmatrix} 2 \\ -8 \end{bmatrix}$ and random starting point. On the plot, show the contours (lines of the constant value) of the quadratic form, the location of the minimum x_* and locations of successive x_i.

2. (3) Consider the Richardson method to compute solution to the linear system $Ax = b$:

$$x_{k+1} = x_k - \tau(Ax_k - b) \tag{3}$$

for symmetric positive definite matrix A. Check the result, quoted on the Lecture for the error wrt to the true solution x_*, $\epsilon_k = x_k - x_*$:

$$\epsilon_{k+1} = \left(\hat{1} - \tau A\right) \epsilon_k \tag{4}$$

and show that the optimal step size reads, in terms eigenvalues of A: $\tau_{\text{opt}} = \frac{2}{\lambda_{\min} + \lambda_{\max}}$.

3. (4) The method of choice to solve linear equations with symmetric positive-definite left–hand side is the conjugate gradient method, with iterations of the form:

$$x_{i+1} = x_i + \alpha_i d_i, \quad \alpha_i = \frac{r_i^T r_i}{d_i^T A d_i} \tag{5}$$

where the step direction d_i is not the residual, as in Eq. (2), but itself is determined by another recursion:

$$d_i = r_i + \beta_i d_{i-1}, \quad \beta_i = \frac{r_i^T r_i}{r_{i-1}^T r_{i-1}}, \quad d_0 \equiv r_0. \tag{6}$$

- Implement this method to solve 2×2 system from the Exercise 5.1 and illustrate the convergence (in the manner, similar to how it was done in 5.1).
- Compute the matrix $D_{ij} = d_i^T A d_j$ for the step directions d_i.

4. (10) In this exercise, you will have to solve $N \times N$ linear system $Ax = b$ with left–hand–side matrix of a special form:

$$A = \text{diag}(d) + \sum_{i=1}^{r} s_i v_i v_i^T, \tag{7}$$

which is low–rank perturbed diagonal matrix.

- The pickled dict `data.pkl` containing the vectors d, v_i and s. Whats is the rank of matrix A?
- Would you be able to form the dense matrix A and solve the corresponding system on your laptop?

- Implement `scipy.sparse.linalg.LinearOperator` interface for an efficient multiplication of an arbitrary vector x by matrix A.
- Choosing an appropriate iterative algorithm from those implemented in `scipy.sparse.linalg`, solve the system $Ax = b$. Estimate the solution error as $\|Ax_* - b\|/\|x_*\|$.

5. (6) Low-rank-perturbed diagonal matrices are very convenient because many operations on them can be performed efficiently (like matrix-vector multiplication in the previous exercise). Here we consider another example. Consider Woodbury matrix identity, which is valid for matrices of consistent sizes:

$$(A + UCV)^{-1} = A^{-1} - A^{-1}U\left(C^{-1} + VA^{-1}U\right)^{-1} VA^{-1}. \tag{8}$$

In the particular case of a diagonal $(p \times p)$ matrix A and unit $(k \times k)$ matrix C, the matrix $A+UCV$ is low-rank perturbed diagonal (whats the rank of the perturbation?). Compose a function `woodbury(A, U, V)` to compute $(A + UV)^{-1}$ using the Eq. (8). Check your implementation, comparing the result with that of a straightforward computation. Time these two ways to compute: which one is faster and why? (consider random matrices at $p = 5000, k = 100$).

6. (15) The file `y.npy` contains timeseries of $t = nT$ observations y_i, where $i = 0, 1, ..., t-1$ [with $T = 5000$, $n = 10$]. The vector y_i stores noisy measurements of underlying periodic signal Y_i, where $i = 0, 1, ..., T-1$, so that observation y cover n periods of the signal Y.

- Solve the minimization problem

$$\min_Y \left[\sum_{i=0}^{t-1} \|y_i - Y_{i \bmod T}\|^2\right], \tag{9}$$

analytically to recover the signal Y from y and plot it.
- Imagine you know that the underlying signal is smooth. In this case, a reasonable optimization problem

$$\min_Y \left[\sum_{i=0}^{t-1} \|y_i - Y_{i \bmod T}\|^2 + \gamma \sum_{i=0}^{T-1} (Y_i - Y_{i+1})^2\right], \tag{10}$$

where we defined $Y_T = Y_0$. In this equation, γ is to be explored; for $\gamma = 0$ no smoothing is applied and the previous solution is recovered. The solution to this equation can be achieved by reducing it to a linear system $AY = b$, where A is a certain sparse matrix. Construct A, b and find Y choosing an appropriate iterative algorithm from those implemented in `scipy.sparse.linalg`. Pick several values of γ and plot the resulting Y.

7. (20) One of the famous linear problems is related to ranking web-pages relying in the structure of mutual links between them. `PageRank` is determined recursively: importance p_i of i–th page is defined as an average value of importances of all pages, which reference to the page i. This can be written formally as follows:

$$p_i = \sum_j \frac{p_j}{L(j)} l_{ij}, \tag{11}$$

where $l_{ij} = 1$ if j–th page refers to the i–th page ($l_{ij} = 0$ in the opposite case), while $L(j)$ – number of links outgoing from the page j. This system can also be written as:

$$p = Gp, \quad G_{ij} = \frac{l_{ij}}{L(j)}. \tag{12}$$

Additional convention is that for 'hanging' nodes in the graph (all entries in a certain column vanish), the corresponding column in G is filled by a number $1/n$. Finally, for regularization purpose, the parameter $0 < \beta < 1$ is introduced and the matrix G is replaced by

$$G \to \beta G + \frac{1-\beta}{n} ee^T, \tag{13}$$

where e – vector of ones. In the end, we arrive to the following system (assuming normalization $\sum_i p_i = 1$):

$$\left(\hat{1} - \beta G\right) p = \frac{1-\beta}{n} e \tag{14}$$

and we consider $\beta = 0.8$ in what follows.

- Propose a small (~ 10 nodes) connectivity graph, construct corresponding (`np.array`) matrices l and G and compute numerically the solution to Eq. (14), using `numpy.linalg.solve`. Explore the resulting values of `PageRank`.
- Download the Gnutella dataset, describing an oriented graph via the edge-list. Unpack the archive and load it to sparse `csr` matrix. Compute the `PageRank` for this data, using i) dense description of the system and ii) sparse description of the system together with an appropriate `scipy.sparse.linalg` solver. Compare the results and the runtime.
- Consider web-Stanford dataset. For this problem dense approach would be unfeasible. Using iterative method, compute the `PageRank` for this data. Which node has the largest `PageRank`?