MAA209 Numerical Optimization

École Polytechnique

Practical Session #5

Instructions: The codes should be written in Python. The use of notebooks is encouraged and you may start from the given examples. **Remove irrelevant cells from the Notebooks** if you start from a code given on Moodle. **Write comments** regarding your work and the experiments you did with the code. **Remove plotting** if the dimension is larger than two.

Upload your work on Moodle (one or multiple notebooks). **Solve one of the exercises** for a full activity point. Students who, in addition, solve correctly parts of more than one exercises will get bonus points.

Solving some **supplementary** or **Challenge** questions will get you additional bonus points. Recall however, that the **Challenge** and **supplementary** exercises are not mandatory.

Exercise 1 Quasi-Newton methods - SciPy

1. Observe the documentation of the command `scipy.optimize.minimize` available online: `https://docs.scipy.org/doc/scipy/reference/optimize.html`. Learn how to use the `minimize` command for different methods like BFGS, LBFGS, CG, etc. You may use the functions below as a starting point:

   (a) The Rosenbrock function
   $$f(x,y) = 100(y-x^2)^2 + (1-x)^2.$$
   Recall that the global minimum is found at $x^* = (1,1)$.

   (b) The N-dimensional Rosenbrock function:
   $$f(x) = \sum_{i=1}^{N-1} [100(x_{i+1}-x_i^2)^2 + (1-x_i)^2],$$
   where $x = (x_1, ..., x_N) \in \mathbb{R}^N$.

   (c) $f(x) = |x|^4$ (here you may choose to work in dimension higher than two).

   (d) The Beale function defined on $[-4.5, 4.5]^2$:
   $$f(x,y) = (1.5 - x + xy)^2 + (2.25 - x + xy^2)^2 + (2.625 - x + xy^3)^2$$
   Note that this function has a global minimum $f(3, 0.5) = 0$.

   (e) **(Challenge)** You can use the callback function and store the function values in a vector. Do this for each method used and plot the function values for all methods in logarithmic scale on the y coordinate (like in the course) to be able to visually observe the speed of convergence of every algorithm. For this you should use `matplotlib.pyplot.semilogy`.

2. Consider $\mathcal{S}$ to be a finite set of vectors of the form (n_1, n_2, n_3) with $n_i \in \{-1, 0, 1\}$. Once such a set $\mathcal{S}$ is defined consider the function $\chi(x,y,z) : (0,\infty)^3 \to \mathbb{R}$ defined by
   $$\chi(x,y,z) = \sum_{(i,j,k)\in\mathcal{S}} x^i y^j z^k.$$

   (a) **(Optional)** Prove that if $\mathcal{S}$ is not included in a half-space then the associated function χ admits a global minimum.
   Hint: If the set is not contained in a half-space then 0 is in its convex hull. It is then possible to use the generalized AM-GM inequality recalled in a previous course to find a lower bound for χ when $x, y, z \in (0,\infty)$.

   (b) Write a generic program which can compute the values of χ and the gradient $\nabla\chi$ (and eventually the Hessian matrix $D^2\chi$) when given a matrix M whose columns are the vectors in $\mathcal{S}$. Use the `scipy.optimize.minimize` command to find the minimizers in the following cases. In each case, the set $\mathcal{S}$ is made of the columns of the matrix given.

   - $\begin{pmatrix} -1 & -1 & -1 & 0 & 1 \\ -1 & -1 & 1 & 0 & 0 \\ -1 & 0 & -1 & 1 & 0 \end{pmatrix}$

- $\begin{pmatrix} -1 & -1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & -1 & 0 & 1 & 1 & -1 \\ 0 & 1 & 0 & 1 & -1 & 0 & 0 \end{pmatrix}$
- $\begin{pmatrix} -1 & -1 & -1 & 1 & 1 \\ -1 & 0 & 1 & -1 & 1 \\ 0 & -1 & 1 & 0 & 1 \end{pmatrix}$

[Bibliography: If you want to know the source of this problem and its connections to probability theory, take a look at the following webpage and the references therein: `http://www.cmap.polytechnique.fr/~beniamin.bogosel/RandomWalks.html`.]

Answer of exercise 1

Please observe the code given on moodle. In order to see the methods available in `scipy.minimize.optimize` you should check the online documentation.

Observe the structure of the callback function. This function is called at every iteration. It can be used to store variables or print information during the optimization algorithm.

1. No particular correction needs to be given here, since the mathematical aspects are quite standard.

2. Note that the partial derivative w.r.t. x is just

$$\sum_{(i,j,k)\in\mathcal{S}} ix^{i-1}y^j z^k.$$

   You could use a loop to construct the objective function and the gradient, iterating on the columns of the given matrix.

Exercise 2 The Conjugate Gradient algorithm

In all the following A is a symmetric, positive-definite matrix and f is the usual quadratic function $f(x) = \frac{1}{2}x^TAx - b^Tx$. Recall that the solution of this problem is the solution of the system $Ax = b$.

1. Implement the Gram-Schmidt algorithm which can produce a basis of A-orthogonal vectors starting from a basis of $\mathbb{R}^n$. Using this basis of A-orthogonal vectors compute the inverse of A, as shown in the course.
2. Implement the Conjugated Gradient (CG) algorithm (see Course #5) which computes the next conjugate direction at each iteration. Test numerically the following claims:
   - The CG algorithm converges in n iterations.
   - If A has only $k < n$ distinct eigenvalues then the CG algorithm converges in k iterations.

   Note: In order to generate interesting test-cases recall that any orthogonal matrix U and diagonal matrix D with positive entries generates a symmetric positive definite matrix

$$A = UDU^T.$$

   You may define an arbitrary diagonal matrix D starting from a vector with positive entries using the command `numpy.diag`. You can find general orthonormal matrices U starting from an arbitrary basis of $\mathbb{R}^n$ with the command `scipy.linalg.orth`. For example, you may consider the following code:

```
N = 100
M = np.random.rand(N,N)
U = scipy.linalg.orth(M)
D = numpy.diag(numpy.random.rand(N))
A = U@D@U.T
```

   which will generate a symmetric positive matrix A of size 100×100. Moreover, in order to test the convergence of CG in terms of the number of distinct eigenvalues you may manually set some of the values in D to be equal to the same common value in order to obtain eigenvalues with the multiplicity that you want. **Do not choose N too big. Recall that a 10000×10000 matrix already occupies 800 Mb of RAM memory and a 40000×40000 matrix will occupy almost 13 Gb of RAM.**

3. For a moderate size problem (e.g. $n = 1000$) compute the condition number Q of the matrix A and compute the following quantities

    - the error with respect to the objective function
    $$E(x_i) = \frac{1}{2}(x_i - x^*)^T A(x_i - x^*)$$
    - the error estimate
    $$4\left(\frac{\sqrt{Q}-1}{\sqrt{Q}+1}\right)^{2i}$$

    and store them in two vectors. Run the full n iterations of the CG algorithm.
    Plot the two quantities at the end of the optimization algorithm in order to have a comparison between the actual error and the error estimate. Decide where you could have stopped the optimization process with acceptable results.

Exercise 3 Challenge: Solve Laplace's equation in 1D

Given a function $f : [0,1] \to \mathbb{R}$ we want to solve the problem

$$\begin{cases} -u'' = f & \text{in } (0,1) \\ u(0) = u(1) = 0. \end{cases} \tag{1}$$

1. In some cases the explicit solution can be found. Find $u(x)$ when f is constant and equal to 1.

   In order to approximate the solution u for a general source function f, it is necessary to discretize the problem. In order to do this, consider a uniform discretization of the interval $(0,1)$

   $$0 = x_0 < x_1 < ... < x_{N+1} = 1,\ x_{i+1} - x_i = \frac{1}{N+1}$$

   and denote by u_i the approximations of $f(x_i)$. Based on the Taylor expansion it is possible to approximate the first and second derivatives using **finite differences** in the following way:

   $$u'(x) \approx \frac{u(x+h) - u(x)}{h} \qquad u''(x) \approx \frac{u(x+h) + u(x-h) - 2u(x)}{h^2}.$$

   If f_i is the value of f at x_i: $f_i = f(x_i)$ then the equality $u''(x) = 1$ written in discrete form is given by:

   $$\frac{2u_i - u_{i+1} - u_{i-1}}{h^2} = f_i, i = 1, ..., N,$$

   where $h = \frac{1}{N+1}$ is the uniform step defined by the discretization.
2. **(Optional)** Show that boundary conditions of the type $u(0) = a, u(1) = b$ can be introduced by changing the function f at nodes 1 and N in the following way:

   $$f_1 = f(x_1) + \frac{a}{h^2},\quad f_N = f(x_N) + \frac{b}{h^2}.$$

3. After discretization, equation (1) becomes a linear system

   $$AU = F,$$

   equivalent to the minimization problem

   $$\min_{U \in \mathbb{R}^N} \frac{1}{2} U^T A U - F^T U.$$

   where $U = (u_1, ..., u_N)$ and $F = (f_1, ..., f_N)$. Moreover, the matrix A has the form

   $$A = \frac{1}{h^2} \begin{pmatrix} 2 & -1 & 0 & \cdots & 0 \\ -1 & 2 & -1 & \cdots & 0 \\ 0 & -1 & 2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 2 \end{pmatrix}$$

   Construct the matrix A using the command `scipy.sparse.diags` which exploits the fact that the matrix is constant along some of its diagonals. In addition, with this definition the matrix A is sparse and the memory space occupied is $O(N)$, where N is the size of the discretization.

   Solve the previously defined linear system for different values of N, noting that A is symmetric, positive definite, so the Conjugate Gradient algorithm can be applied. Plot the solution and compare it with the analytical case when $f \equiv 1$. Try different other choices for the function f, noting that the discrete algorithm does not change (only the vector F does).
4. Give a conjecture on the rate of growth of the condition number of A as N increases. You may use the command `numpy.linalg.cond`.
5. **(Challenge)** Given $\alpha \in \mathbb{R}$ consider problem (1) for $f \equiv -1$ with the additional constraint $u(0.5) \geq \alpha$. Choose N odd, such that 0.5 is a point of the discretization and use a projected gradient descent algorithm (with variable step) in order to find the solution of the minimization problem under the constraint $u_{(N+1)/2} \geq \alpha$. Observe the behavior of the solution with respect to α.