MAA251 Numerical Optimization

École Polytechnique

Practical Session #4

Instructions: The codes should be written in Python. The use of notebooks is encouraged and you may start from the given examples. The objective is to solve as many questions as possible from at least one of the exercises.

Exercise 1. Gauss-Newton Method

1. Observe the notebook concerning the Gauss-Newton Method shown in the course and test it for the non-linear least square functions proposed below. In each case test the behavior of the algorithm for different initializations.

- **Rosenbrock** (already implemented)

$$f(x, y) = 100(y - x^2)^2 + (1 - x)^2$$

  with a global minimum at $f(1, 1) = 0$.
- **Beale** function defined on $[-4.5, 4.5]^2$ by

$$f(x, y) = (1.5 - x + xy)^2 + (2.25 - x + xy^2)^2 + (2.625 - x + xy^3)^2$$

  with a global minimum at $f(3, 0.5) = 0$.

For all the subsequent questions use the Gauss-Newton implementation to solve the non linear least-squares problem involved.

2. Implement the triangulation problem shown in the course. As inputs you are given:

- the coordinates (x_i, y_i), $i = 1, 2, 3$ of three antennas
- the distances d_i, $i = 1, 2, 3$ of a cellphone to the corresponding antenna

Test your algorithm in various cases and compare the convergence speed with the one given by the gradient descent algorithm.

3. **(Challenge: Population model)** Take a look at the data regarding the population given in the .csv file on Moodle (source `worldometers.info`). Consider the following models:

- **exponential model:** $P(t) = c\exp(dt)$, assuming that the growth rate of the population is proportional with the population size.
- **logistic model:** $P(t) = \dfrac{K}{1 + ce^{-dt}}$, assuming that the growth ratio is influenced not only by the population size, but also by the fact that the Earth is finite in space and ressources, so infinite growth cannot occur.

Try to find in each case the parameters which best fit your data using the Gauss-Newton algorithm. For example, in the first case write

$$F(c, d) = \sum_{i=1}^{N} [P_i - c\exp(dt_i)]^2,$$

where P_i is the population observed at the moment t_i. The non-linear least squares functions r_i are given by

$$r_i(t) = P_i - c\exp(dt_i).$$

Use these to find the Jacobian matrix (derivatives w.r.t. c and d) and to find parameters which minimize F. For the exponential model consider the initialization $(c, d) = (3, 2)$. For

the logistic model you have a function depending on K, c and d and consider the initialization $(K, c, d) = (12, 3, 3)$.

Once you found the optimal parameters plot the functions obtained against the data points to verify your results. What is the predicted population of the Earth in 2050 and 2100 for the two models? Which of the two models is more realistic? Which of the two models best fits the data? What is the interpretation of K in the logistic model?

4. **(Temperature model)** Consider a temperature model of the form $T(t) = A\sin(wt + \phi) + C$ where t is the time in months. The site `climate-data.org` provides the following averages of monthly maximal temperatures.

Jan.	Feb.	Mar.	Apr.	May	Jun.	Jul.	Aug.	Sep.	Oct.	Nov.	Dec.
6	7.3	12.2	15.9	19.8	22.9	24.8	24.3	21.3	15.7	10.1	6.7

Find the parameters A, w, ϕ and C which best fit your data. You may suppose that the data points are given at instances $t_i = i,\ i = 1, ..., 12$. Plot the function obtained against the data points to test your result. Predict the maximum average temperature on April 1st and April 30th using your model.

Hints:

1. For the Rosenbrock function you have $r_1 = 10(y - x^2)$ and $r_2 = 1 - x$. The functions r_i are grouped in the vector r. The Jacobian matrix J contains the partial derivatives of r_i on line i. The Gauss Newton direction is

$$-(J^T J)^{-1} J^T r$$

as noted in the course.

For the second example you have

$$r_1 = 1.5 - x + xy,\ r_2 = 2.25 - x + xy^2,\ r_3 = 2.625 - x + xy^3.$$

Once you compute the Jacobian matrix the code remains the same.

2. For the triangulation problem you have

$$r_i(x, y) = d_i - \sqrt{(x - x_i)^2 + (y - y_i)^2}.$$

The partial derivative w.r.t. x is (check it)

$$\frac{\partial r_i}{\partial x} = -\frac{x - x_i}{\sqrt{(x - x_i)^2 + (y - y_i)^2}}.$$

In order to find admissible test cases do the following:

- Choose three different coordinates (x_i, y_i) of the antennas
- Choose coordinates for the desired solution (x_0, y_0) (preferably in the interior of the triangle determined by the three points, but you may experiment if you like)
- Compute d_i as the distance between (x_i, y_i) and (x_0, y_0).
- Then test if the solution given by the algorithm is close to (x_0, y_0).

3. **Important:** If you encounter difficulties here, you can skip this part. If you don't take care you'll encounter overflows in the exponential function, especially if you evaluate it for very large values. Please follow the indications given below.

You can quickly import the data from the csv file into a Numpy array in the following way:

```
import csv
data = np.genfromtxt('WorldPopulation.csv', delimiter=',', names=True)
T   = data['Year']             # for the time series
Pop = data['Population']/1e9   # for the population in billions
```

For the population model note that

$$r_i = P_i - c\exp(dt).$$

Before going further, it is possible to reduce the ill-conditioning of the problem and avoid overflows in the exponential function by doing the following:

- consider that the population is given in billions (i.e. divide the numbers by 10^9).
- rescale the times t_i to be in $[0, 1]$: one way of doing this is $(t_i - m)/(M - m)$ where m and M are the smallest and largest time values. Note that such a rescaling can be included in c and d.

It is straight-forward to find the derivatives of the above function w.r.t. c and d. Here it would be particularly useful to use a loop to define r and the Jacobian J. You may use the following syntax:

```
def r(x):
    return np.array([
        P[i]-c*np.exp(d*t[i]) for i in range(0,N)
    ])
```

where N is the size of the data vectors. You will need to provide also the observed population vector P of size N and the corresponding times in the vector t. For the Jacobian do the same:

```
def Jac(x):
   return np.array([
      np.array([dr_i/dc, dr_i/dd]) for i in range(0,N)
   ])
```

For the logistic model the configuration is the same, but you have three variables.

4. For the temperature model do the same as for the population model. Note that you have four parameters now. The functions r_i are given by

$$r_i(A, w, \phi, C) = T_i - (A\sin(\omega t_i + \phi) + C)$$

and their derivatives w.r.t. the corresponding variables are straight-forward.

5. For the distance model just write the r vector and the jacobian using a loop over the indices $[i, j]$ with $i < j$. The rest of the code stays the same.

Exercise 2. Newton's Method

1. Implement Newton's method presented in the course and test it for various 2D functions. Look at the following cases:

(a) Recall why for quadratic functions convergence happens in one iteration. Implement one quadratic function of your choice and test this claim.

(b) Test the method for the Rosenbrock function

$$f(x,y) = 100(y-x^2)^2 + (1-x)^2.$$

with the initialization $x_0 = (2, 1.1)$. Recall that the global minimum is found at $x^* = (1,1)$.

(c) Observe the rate of convergence rate for $f(x) = |x|^4$ and explain the observed behavior.

2. Add one line-search procedure of your choice and observe the behavior of the algorithm.

3. Use the Newton-Rhapson method to solve the following non-linear systems, searching for a root near an initial choice x_0. You may vary the initialization and observe the behavior of the algorithm.

1. $\begin{cases} x_1 - x_2 + 1 &= 0 \\ x_1^2 + x_2^2 - 4 &= 0 \end{cases}$, with $x_0 = (0.8, 1.8)$. Compare with the analytical solution.

2. $\begin{cases} 3x_1 - \cos(x_2 x_3) - 3/2 &= 0 \\ 4x_1^2 - 625x_2^2 + 2x_3 - 1 &= 0 \\ 20x_3 + e^{-x_1 x_2} + 9 &= 0 \end{cases}$, with $x_0 = (1,1,1)$

3. $\begin{cases} x_1^2 - 2x_1 + x_2^2 - x_3 + 1 &= 0 \\ x_1 x_2^2 - x_1 - 3x_2 + x_2 x_3 + 2 &= 0 \\ x_1 x_3^2 - 3x_3 + x_2 x_3^2 + x_1 x_2 &= 0 \end{cases}$, with $x_0 = (1,2,3)$ and $x_0 = (0,0,0)$

4. **(Challenge)** For more examples of functions for which you can test your algorithm take a look at the following page: `https://en.wikipedia.org/wiki/Test_functions_for_optimization`. This page contains many test cases and in each case you are given the formula, the search domain and the analytic minimum. Try to validate your algorithm for as many of these function as you can.

Hints:

1. You can start from the Notebook given for the Gauss-Newton Method. Build a function called `Hess` which computes the Hessian matrix. Then replace the function `GNdir` with `Newtondir` and compute the Newton direction by solving the linear system $D^2 f(x_i)d = -\nabla f(x_i)$

Recall that the Hessian matrix contains all combinations of second derivatives. For e.g. in dimension two:

$$D^2 f(x) = \begin{pmatrix} \frac{\partial f^2}{\partial x^2} & \frac{\partial f^2}{\partial x \partial y} \\ \frac{\partial f^2}{\partial y \partial x} & \frac{\partial f^2}{\partial y^2} \end{pmatrix}$$

If you want to cheat and avoid computing the Hessian by hand you can do it in Wolfram Alpha: just type

```
hessian matrix of 100(y - x^2 )^2 + (1 - x)^2
```

2. This is already done in the Gauss-Newton Notebook.

3. The Newton-Rhapson method is given by

$$y_{i+1} = y_i - Dg(y_i)^{-1} g(y_i)$$

where $g(y) = (g_1(y), ..., g_n(y))$ is a non-linear function and $Dg(y) = (\frac{\partial g_i}{\partial x_j})$ is the Jacobian matrix. In each case you should write the function g and compute its Jacobian. For the first example you have

$$g_1 = x_1 - x_2 + 1,\ g_2 = x_1^2 + x_2^2 - 4.$$

The Jacobian matrix is

$$Dg = \begin{pmatrix} 1 & -1 \\ 2x_1 & 2x_2 \end{pmatrix}.$$

Note that in this case the solution may also be found analytically (intersection between a line and a circle). In each case you should check that the solution given by the numerical algorithm verifies the non-linear system with the desired precision.