45010: Optimization I **2019W**

Project 4: CG, BFGS and SR1

Project member(s): *Date:*

Note: *Here is a guideline for your project.*

- *The project is designed to help understand the topic, so please do not copy and paste.*
- *You are welcome to work in groups (no more than 3 in one group), but you should write it up for your submission/exercises on your own.*
- *The standard submission of project contains the following documents:*
  - *The code files*
  - *A* `README` *file*
  - *A brief writeup document answering the questions*

In your `README` *file, there are several things you need to declare:*

- *What language are you using to write the program;*
- *How to run the code(s);*
- *Contributor(s).*

You can add additional information such as possible issues, future plan in the `README` *file as well. For the writeup document, you will be answering the questions in the project.*

If you are only submitting the `ipynb` file, you can include everything in it.

4.1 Project descriptions

Please read carefully, in this project you are going to implement the conjugate gradient method and the quasi-Newton methods (BFGS and SR1).

For the conjugate gradient method, we will perform experiments on a few special matrices to test its performances and compare with other built-in functions.

For the quasi-Newton methods, we will test your algorithm on Rosenbrock function and a practical problem to show the performance, we will also compare the quasi-Newton with the standard Newton's method (step length equals 1).

4.1.1 Conjugate gradient method

The algorithm is included in Figure (4.1).

Algorithm 5.2 (CG).
Given x_0;
Set $r_0 \leftarrow Ax_0 - b$, $p_0 \leftarrow -r_0$, $k \leftarrow 0$;
while $r_k \neq 0$

$$\alpha_k \leftarrow \frac{r_k^T r_k}{p_k^T A p_k}; \tag{5.24a}$$
$$x_{k+1} \leftarrow x_k + \alpha_k p_k; \tag{5.24b}$$
$$r_{k+1} \leftarrow r_k + \alpha_k A p_k; \tag{5.24c}$$
$$\beta_{k+1} \leftarrow \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}; \tag{5.24d}$$
$$p_{k+1} \leftarrow -r_{k+1} + \beta_{k+1} p_k; \tag{5.24e}$$
$$k \leftarrow k + 1; \tag{5.24f}$$

end (while)

Figure 4.1: Conjugate gradient method from the book.

This task requires you implement the above CG algorithm, the signature of your function is `CG(A, b)` or `CG(A, b, x0)` with an input for starting point. The outputs are final solution and iteration number.

• If you do not know how to choose x_0, simply take all zeros or generated from random numbers.

• Be careful, for your implementation, there are some places to optimize such as (5.24a) ad (5.24c), you can precompute Ap_k and reuse it in these two formulas. Also see (5.24a) and (5.24d), you can precompute $r_k^T r_k$ first and reuse it. This will make your implementation slightly faster.

• The next thing about the implementation is loop stopping criteria. In the above algorithm, the loop condition is $r_k \neq 0$ only, in our implementation, we choose $\|r_k\| > 10^{-6}$ and $k < n$ instead.

The experiments are to use your CG algorithm to find the solution to $Ax = b$. A and b are given in the following cases.

1. A is Hilbert matrix of size $n \times n$, the (i, j) entry of A is

$$A_{ij} = \frac{1}{i + j - 1} \tag{4.1}$$

b is the vector of all ones of length n. Perform your CG algorithm for $n = 2, 4, 8, 16, 32$ and print out the final error $\|Ax - b\|$ (It is OK if you find the result weird). Also compare your result against the error results obtained from the builtin functions (In Python, use `np.linalg.solve`. In MATLAB, use the backslash).

2. A is the matrix of size $n \times n$ and has the tridiagonal shape.

$$A = \begin{pmatrix} 2 & -1 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & -1 & 2 & -1 & 0 & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & \dots & 2 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \dots & -1 & 2 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & \dots & 0 & -1 & 2 \end{pmatrix} \tag{4.2}$$

and b is all ones vector of size n. Perform your CG for $n = 2, 4, 8, 16, 32$ and print out the final error $\|Ax - b\|$. Also compare your result against the error results obtained from the builtin functions (In Python, use `np.linalg.solve`. In MATLAB, use the backslash).

4.1.2 BFGS and SR1

In this part, we will implement the famous BFGS and SR1 algorithms. The algorithms are under the same framework just with different update formulas.

1. Test your algorithms with Rosenbrock function. What are the iteration numbers for BFGS and SR1. Compare them with your previous Newton's method. Starting points are $(1.2, 1.2)$ and $(-1.2, 1)$.
2. Test your algorithms with the Powell's quartic function

$$f(x) = (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10(x_1 - x_4)^4 \tag{4.3}$$

with starting point as $x_0 = [3, -1, 0, 1]^T$, optimal solution $x^* = [0, 0, 0, 0]^T$. What are the iteration numbers for BFGS and SR1.

4.1.3 Optional task (extra 1pt)

Fill the table. The methods are

1. Exact line search steepest descent method
2. Fixed step length steepest descent method with $\alpha = 10^{-3}$.
3. Newton's method with fixed step length as 1.
4. Backtracking steepest descent method with $c_1 = 10^{-4}$, $\rho = 0.1$.
5. Backtracking Newton's method with $c_1 = 10^{-4}$, $\rho = 0.9$.
6. Heavyball method with $\alpha = 10^{-3}$ and $\beta = 0.95$.
7. Trust region method with Cauchy point, parameters are set as in project 3.
8. Trust region method with dogleg method, parameters are set as in project 3.

9. BFGS method with exact line search.
10. BFGS method with backtracking line search.
11. SR1 method with exact line search.
12. SR1 method with backtracking line search.

The stopping criteria is set as gradient's norm less than 10^{-9}.

And perform the experiments on the following problems:

P1. Quadratic function as in Project 1. Starting at $(5, 5)$.

P2. Rosenbrock function as in Project 1. Starting at $(-1.2, 1)$.

P3. Powell's quartic function in this project.Starting at $(3, -1, 0, 1)$.

Methods	Problem	iteration number
1	P1	
2	P1	
3	P1	
4	P1	
5	P1	
6	P1	
7	P1	
8	P1	
9	P1	
10	P1	
11	P1	
12	P1	

Methods	Problem	iteration number
1	P2	
2	P2	
3	P2	
4	P2	
5	P2	
6	P2	
7	P2	
8	P2	
9	P2	
10	P2	
11	P2	
12	P2	

Methods	Problem	iteration number
1	P3	
2	P3	
3	P3	
4	P3	
5	P3	
6	P3	
7	P3	
8	P3	
9	P3	
10	P3	
11	P3	
12	P3	

4.2 Starter code kit

The starter code kit is in Python (MATLAB users should have no difficulty to understand.) It is up to you to following the starter code kit's style or not. As a reminder, the kit only fills out framework, there are some details you will have to write on your own.

4.3 Submission

If you prefer to use the GitHub or Bitbucket, etc. to store your code, then it is OK to simply submit the repository's link on a text file (if your repository is public), otherwise, please submit all files in a zip file through Canvas.