

CS726, Fall 2008
Homework 4
(due Friday 10/3/08)

Submit your Matlab files electronically using the instructions on the course web page. Once you have set up the path in your user directory, you need to put your files for submission in a single directory and run the command

```
handin -c cs726-1 -a hwk4 -d <directory name>
```

You should hand in files with the names `BBtests.m`, `SDQ.m`, `BBQ.m`, `BBaltQ.m`, `BBcyclicQ.m`, and `hw4comments.txt`.

In this assignment you will compare the performance of several line search methods - nonmonotone methods for the most part - on a random convex quadratic function. You'll perform 10 runs of each method, each from a different random starting point, and compute the average number of iterations for each methods, the number of runs on which the method failed to converge (in a specified maximum number of iterations), and the maximum increase of f from one iteration to the next, over all the runs of that method.

Efficiency of your code is not important.

The function f is defined by $f(x) = (1/2)x^T Qx$, where Q is an $n \times n$ diagonal matrix with diagonal elements selected randomly in the range $[0, 10^4]$. You can use the following code at the start of your main program `BBtests.m` to generate Q and print its condition number:

```
n=10;  
Q = 4*rand(n,1); Q = diag(10.0.^Q);  
condQ=cond(Q);  
fprintf(1,' condition number of Q is %6.2e\n', condQ);
```

Each of the line search methods that you code up should have the same list of input and output arguments. I give the specification for the steepest descent code; the others will be similar. The calling sequence for `SDQ` should be as follows:

```
function [inform, iters] = SDQ(Q,x,params)
```

where

- Q is the Hessian, generated as above;
- x contains the starting value of x on input. This should be generated randomly (and differently for each run, of course) by choosing each element independently from a uniform distribution on $[0, 1]$.
- the structure `params` is defined as follows: `params=struct('tol',1.e-5,'maxits',100000)`, where the algorithm terminates when $\|\nabla f(x)\|_2 \leq \text{tol}$, and `maxits` is the maximum number of iterations to take before declaring failure.

- `inform.status` is set to 0 on output if failure was detected, and 1 otherwise, while `inform.largestchange` contains the largest value of $f(x_{k+1})/f(x_k)$ detected during the run.
- `iters` is the number of iterations required.

For each of the codes called by `BBtests.m` you should report:

- the number of failures during the 10 runs;
- the average number of iterations;
- the largest ratio $f(x_{k+1})/f(x_k)$ detected during all the runs.

Program up the following methods:

- Steepest descent with exact line search: `SDQ.m`
- Barzilai-Borwein: `BBQ.m`
- Barzilai-Borwein with the alternate formula: `BBaltQ.m`
- cyclic Barzilai-Borwein: `BBcyclicQ.m`. The calling sequence here contains an additional parameter to indicate cycle length, that is, `[inform, iters] = BBcyclicQ(Q,x,cycleLength,params)`. Test this routine with different values of `cycleLength` in a reasonable range.

Use $n = 10$ in your submitted version of `BBtests.m` and list the output for this n in the `hw4comments.txt`. The results for each method should occupy only a single line of output; `BBtests.m` should generate no more than 10 lines of output overall.

In `hw4comments.txt` you should also report on such issues as

- The overall comparative performance of the different methods.
- The maximum increase ratios $\max_k f(x_{k+1})/f(x_k)$.
- How do the results change if you increase the condition number of the Q (by broadening the range of diagonal values)?
- How do the results change as n is increased?
- Is there a rule-of-thumb value of `cycleLength` that works well under most circumstances?
- By experimenting, can you find some other scheme in the same spirit as the schemes considered above that has consistently better performance?