CS726, Fall 2016

Homework 3 (due Friday 3/2/18 at 6:00pm)

All exercise numbers refer to the course text, *Numerical Optimization* (second edition, 2006).

1. Exercise 3.2 from the textbook: Show by drawing a picture that if $0 < c_2 < c_1 < 1$ in the weak Wolfe conditions, there may be no α that satisfies these conditions.

2. Exercise 3.6 from the textbook.

3. Write a Matlab code to implement the extrapolation-bisection line search (EBLS) procedure discussed in class (and specified below). Use parameters $c_1 = 10^{-3}$ and $c_2 = 0.5$ in EBLS, and rather than iterating "forever," allow a maximum of 25 adjustments of α. The header line of your routine should be

   ```
   function [x,alpha] = EBLS(fun, x, d, alpha_start)
   ```

   Where the output `alpha` is the value of α identified by the procedure, and the input `alpha_start` is the initial guess $\bar{\alpha}$. The other parameters are as follows:

   fun - a pointer to a function (such as obja, objb, rosenbrock)

   x - a strucure with three fields x.p, x.f, and x.g in which x.p contains the point x, while x.f and x.g contain the function and gradient values corresponding to x. On input, x.p is set to the starting point values (for example, `x = struct('p', [-1.2, 1.0]);`) while x.f and x.g are set to the corresponding function and gradient values. On input, it is assumed that x.f and x.g are set to the current function and gradient values at x.p. On output, it is assumed that x.p is set to $x + \alpha d$ (where α is the step length identified by the EBLS procedure), and x.f and x.g are set to the corresponding values of function and gradient.

   d - a vector containing the search direction

The routine should call on `fun` to evaluate the objective function and gradient as computed points, as follows

```
x.f = feval(fun,x.p,1);
```

and

```
x.g = feval(fun,x.p,2);
```

respectively. (The last argument "1" and "2" indicates to feval to return the function and gradient, respectively.)

Given $0 < c_1 < c_2 < 1$, set $L \leftarrow 0$, $U \leftarrow +\infty$, $\alpha \leftarrow \bar{\alpha}$;
repeat
  if $f(x + \alpha d) > f(x) + c_1 \alpha \nabla f(x)^T d$ **then**
    Set $U \leftarrow \alpha$ and $\alpha \leftarrow (U + L)/2$;
  else if $\nabla f(x + \alpha d)^T d < c_2 \nabla f(x)^T d$ **then**
    Set $L \leftarrow \alpha$;
    if $U = +\infty$ **then**
      Set $\alpha \leftarrow 2L$;
    else
      Set $\alpha = (L + U)/2$;
    end if
  else
    Stop (Success!);
  end if
until Forever

4. Test your routine by writing a Matlab program `SteepDescentLS.m` to implement the steepest descent method, with $d_k = -\nabla f(x_k)$. Terminate when either $\|\nabla f(x_k)\|_2 \leq 10^{-4}$ or 100000 function evaluations have been taken, whichever comes first. The first line of the function should be

   ```
   function [inform,x] = SteepDescentLS(fun,x,sdparams)
   ```

   The inputs `fun` and `x` are as described above, while `sdparams` is the following structure:

   ```
   sdparams = struct('maxit',100000,'toler',1.0e-4,'eta',0.0);
   ```

Here, `maxit` is the maximum number of (outer) iterations of the descent method, `toler` is the convergence threshold for $\|\nabla f\|$, and `eta` is explained below.

The output `inform` is a structure containing two fields: `inform.status` is 1 if the gradient tolerance is achieved and 0 if not, while `inform.iter` is the number of steps taken. The output `x` is the solution structure, with point, function, and gradient values at the final value of x_k.

Test `SteepDescentLS` on the three functions below. (Matlab codes that implement these functions can be downloaded under the names `obja.m`, `objb.m`, and `rosenbrock.m`.) Your program will be tested using the code `descentLS.m`, which can also be downloaded.

(a) $f(x) = x_1^2 + 5x_2^2 + x_1 - 5x_2$

(b) $f(x) = x_1^2 + 5x_1x_2 + 100x_2^2 - x_1 + 4x_2$

(c) $f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$ (Rosenbrock's banana function).

Note that the global variables `numf` and `numg` are reported by the program `descentLS` and incremented by the function evaluation routines.

Do not print out the value of x at each iteration!

For the initial guess of steplength $\bar{\alpha}$ at each invocation of EBLS within SteepDescentLS, try a number of possibilities:

- $\bar{\alpha} = 1$;
- $\bar{\alpha} = \eta \times$ (final successful value of α from previous call to EBLS), where η is a chosen parameter greater than 1.

These are indicated in your code by the setting of `sdparams.eta`. If this variable is set of 0, use $\bar{\alpha} = 1$. Otherwise, use $\eta =$`sdparams.eta`.

Experiment with different values of η, tabulating the dependence of the total number of iterations and the total number of function evaluations on this quantity. Do you see any pattern?

5. Write a routine to perform steepest descent with backtracking line search. At iteration k, you should start the backtracking with some value $\bar{\alpha}_k$, and choose α_k to be the first scalar in the sequence $\bar{\alpha}_k, \beta\bar{\alpha}_k, \beta^2\bar{\alpha}_k, \ldots$ for which the sufficient decrease condition

$$f(x_k + \alpha d_k) \le f(x_k) + c_1 \alpha \nabla f(x_k)^T d_k,$$

is satisfied. Use the values $c_1 = .001$ and $\beta = 0.5$ in your code. The first line of your code should be

```
function [inform,x] = SteepDescentBacktrack(fun,x,sdparams)
```

where the arguments have the same meaning and settings as in `SteepDescentLS`. Your program will be tested using the code `descentBacktrack.m`, which can also be downloaded.

Experiment with the same ways of choosing the initial guess $\bar{\alpha}_k$ of steplength at iteration k as for `SteepDescentLS`. What value of η are most effective? Is the choice of η more critical in the case of `SteepDescentBacktrack` than for `SteepDescentLS`?

Submit your codes for `SteepDescentBacktrack.m`, `SteepDescentLS.m`, `EBLS.m`, and the output from `descentBacktrack.m` and `descentLS.m`. Also hand in your written responses to the first questions and to the questions about the performance of your code.