CS726 - Nonlinear Optimization I - Homework III

September 26, 2012

This assignment is due at the beginning of class on October 3.

This is our first coding assignment. As I mentioned in class, you can either use Matlab or Numerical Python to complete the assignment. This assignment should be submitted electronically using learn@uw. The assignment name is hwk3.

- If you are using Matlab, you should hand in exactly 3 files with the following names: GradientDescent.m, NewtonMethod.m, StepSize.m.
- If you are using Numerical Python, you should hand in one file with three defined functions GradientDescent, NewtonMethod, and StepSize.
- For those choosing to use Numerical Python, you may find it useful to import some useful numpy routines

  ```
  from numpy import dot
  from numpy.linalg import lstsq, norm
  ```

  Note that `dot(A,B)` is matrix multiply for numpy.array's (Do not convert to numpy.matrix). `A\b` in MATLAB is `lstsq(A,b)`.
- You will be graded on how well you minimize code length, number of function calls, run time, and optimization accuracy.

1. Write a subroutine (StepSize.m) to implement backtracking line search. If using Matlab, the header line of the subroutine should be:

   ```
   function [t_step,x_step] = StepSize(fun, x, d, t, params)
   ```

   If using numpy, the definition should be

   ```
   def StepSize(fun, x, d, t, params):
   ```

   The input parameters are

   - **fun** a pointer to a function (such as obja, objb, objc).
   - **x** the starting point
   - **d** a vector containing the search direction
   - **t** the initial value of the step length
   - **params**: In Matlab, this is a structure containing parameter values for the routine

```
    params = struct('alpha', 0.4, 'beta', 0.8, 'maxit', 100);
```

In numpy, this should be a python dictionary with the same values.

The subroutine should call on fun to evaluate the objective function, gradient, and Hessian at computed points, using

```
f_at_x = fun(x,'func');
```

and

```
grad_f_at_x = fun(x,'grad');
```

and

```
hess_f_at_x = fun(x,'hess');
```

respectively. It should compute a step size t_k that satisfies the step criteria. The output parameters are

- **t_step** the value t_k that satisfies the test criteria
- **x_final** the final vector $x_{k+1} = x_k + t_k d_k$.

2. Write a Matlab program (GradientDescent.m) to implement the method of gradient descent. Use backtracking line search, calling the script you wrote in Part 1. Stop the method when either the gradient of the objective function has norm less than 10^{-4} or $1,000$ steps have been made. The header of the function should be

```
function [inform,x] = GradientDescent(fun,x,gdparams)
```

in Matlab and

```
def  GradientDescent(fun,x,gdparams):
```

in Python. The inputs fun and x are as above, and gdparams is the following structure/dictionary

```
gdparams = struct('maxit',5000,'tol',1.0e-4);
```

The output inform is a structure/dictionary containing two fields, **inform.status** is 1 if the gradient tolerance was achieved and 0 if not, **inform.iter** is the number of steps taken and **x** is the solution point.

3. Write a Matlab program (NewtonMethod.m) to implement Newton's method. Use backtracking line search, calling the script you wrote in Part 1. Stop the method when either the gradient of the objective function has norm less than 10^{-9} or $1,000$ steps have been made. The header of the function should be

```
function [inform,x] = NewtonMethod(fun,x,nmparams)
```

The inputs fun and x are as above, and nmparams is the following structure

```
nmparams = struct('maxit',100,'tol',1.0e-9);
```

The output inform is a structure containing two fields, **inform.status** is 1 if the gradient tolerance was achieved and 0 if not, **inform.iter** is the number of steps taken and **x** is the solution point.

4. Matlab and Python functions that implement the functions in the following list

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

(b) $f_b(x) = x_1^2 + 5x_1x_2 + 100x_2^2 - x1 + 4x_2$

(c) $f_c(x) = e^{x_1+3x_2-0.1} + e^{x_1-3x_2-0.1} + e^{-x_1-0.1}$

(d) $f_d(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$

can be found (as obja.m, objb.m, objc.m,objd.m) on the web-site for this course. The program will be tested using the scripts hwk3.m and hwk3.py that are available on the web site. For each of the above four functions, the algorithm will be run using the starting point $(-1.2, 1)$, and with an initial estimate of the step size at each call to your linesearch routine being 1.

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