TEMPO Summer School on Numerical Optimal Control and Embedded Optimization
University of Freiburg, July 27 - August 7, 2015

Exercise 3: Newton's method for optimization

Rien Quirynen   Dimitris Kouzoupis   Moritz Diehl

http://syscop.de/teaching/numerical-optimal-control/

Let us look into the minimization of Rosenbrock's function:

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

Newton's method for root-finding We will first formulate and solve the Rosenbrock problem:

$$\underset{x,y}{\text{minimize}} \quad f(x,y) \tag{1}$$

by implementing our own version of an unconstrained Newton-type optimization algorithm.

3.1 Write a MATLAB function `f_eval(z)` to evaluate the Rosenbrock function $f(z)$ where $z = (x, y)$.

3.2 To solve the Rosenbrock problem, we are interested in the solution(s) to the following first order necessary condition:

$$\nabla f(x^\star, y^\star) = 0. \tag{2}$$

Write down the gradient on paper and make a MATLAB function `Df_eval(z)` to evaluate it:

```
function [f,Df] = Df_eval(z)

end
```

3.3 In order to implement an exact Newton scheme, we will additionally need to evaluate the Hessian matrix $\nabla^2 f(\cdot)$. Again, write this first down analytically on paper and then create a MATLAB function `D2f_eval(z)` to evaluate $f(\cdot)$, $\nabla f(\cdot)$ and $\nabla^2 f(\cdot)$ like this:

```
function [f,Df,D2f] = D2f_eval(z)

end
```

3.4 Now, let us implement an exact Newton scheme to solve the nonlinear system $\nabla f(z^\star) = 0$:

$$z^{[k+1]} = z^{[k]} - \left(\nabla^2 f(z^{[k]})\right)^{-1} \nabla f(z^{[k]}) \tag{3}$$

where $z^{[k]} = (x^{[k]}, y^{[k]})$ and k is the current iteration number. You can use $(x^{[0]}, y^{[0]}) = (0, 0)$ as a starting point, and a possible stopping criterion for this scheme could be $\left\|\nabla f(z^{[k]})\right\| < 10^{-10}$.

Solving the NLP using `fmincon`

3.5 Let us first try to solve the unconstrained minimization problem above, but using `fminunc` instead:

```
options = optimoptions(@fminunc,'Display','iter', ...
      'Algorithm','quasi-newton');
x_sol = fminunc(@f_eval,[0 0],options)
```

How do the iterations compare to our self-written Newton scheme? Let us provide `fminunc` with the necessary derivative information as follows:

```
options = optimoptions(@fminunc,'Display','iter', ...
      'Algorithm','trust-region','GradObj','on','Hessian','on');
x_sol2 = fminunc(@D2f_eval,[0 0],options)
```

Did the performance in number of iterations improve? It is important to note that `fminunc` performs a step size selection, unlike our self-written scheme. In addition, the actual iterations of a Quasi-Newton method are generally cheaper than for an exact Newton scheme.

3.6 Now formulate and solve the following constrained optimization problem:

$$\begin{aligned} \underset{x,y}{\text{minimize}} \quad & f(x,y) \\ \text{subject to} \quad & x^2 + y^2 \leq 1 \end{aligned} \tag{4}$$

You will need to write a MATLAB function `Dc_eval(z)`

```
function [cineq, ceq, Dcineq, Dceq] = Dc_eval(z)
    cineq = ...;
    ceq = [ ];
    Dcineq = ...;
    Dceq = [];
end
```

to evaluate the nonlinear (inequality and equality) constraints, which you then pass to the `fmincon` solver:

```
options = optimoptions(@fmincon,'Display','iter', ...
      'Algorithm','interior-point','GradObj','on', ...
      'GradConstr','on','Hessian','bfgs');
x_sol = fmincon(@Df_eval,[0 0],[],[],[],[],[],[],@Dc_eval,options)
```

3.7 **Extra**: Let us call `fmincon` with exact Hessian information by writing a MATLAB function to evaluate the Hessian of the Lagrangian:

```
function [ H ] = hessian_fun( z,lambda )
    [¬,¬,D2f] = D2f_eval(z);
    D2c = ...;  % TODO: Hessian inequality constraint
    H = D2f + lambda.ineqnonlin*D2c;
end
```

Finally, the updated options to pass for exact Hessians are:

```
options = optimoptions(@fmincon,'Display','iter','Algorithm', ...
      'interior-point','GradObj','on','GradConstr','on', ...
      'Hessian','user-supplied','HessFcn',@hessian_fun);
```