Multidimensional Newton's Method

In multidimensional Newton's Method, we'll assume that we have a function $y = f(x_1, \ldots, x_n)$ for which we're trying to determine the roots of the gradient,

$$\nabla f(\mathbf{x}) = \vec{0}$$

In that case, the function file (Matlab or Python) for f should actually output three items:

- $f(\mathbf{x})$ (which is a real number)
- $\nabla f(\mathbf{x})$ (which is a vector)
- $Hf(\mathbf{x})$, or the Hessian of f, which is a matrix of all of the second derivatives of f:

$$(Hf(\mathbf{x})_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j}(\mathbf{x})$$

With that, recall that multidimensional Newton's method for the gradient of f is given by:

$$\mathbf{x}_{i+1} = \mathbf{x}_i - Hf^{-1}(\mathbf{x}_i)(\nabla f(\mathbf{x}))^T$$

(where we're assuming the gradient is a row, so the transpose as shown above is a column).

Computations

For numerical stability, it is often recommended that we don't actually compute the inverse matrix. Rather, consider the following: If we define a new vector δ to be our update term:

$$\vec{\delta} = Hf^{-1}(\mathbf{x}_i)\nabla f(\mathbf{x}_i)^T$$

Then the vector solves the system of equations:

$$Hf(\mathbf{x}_i)\vec{\delta} = \nabla f(\mathbf{x}_i)^T$$

(again, the right hand side of the equation is a column vector). So in the code, you'll see us solving this equation.

Here's the code for that

```
function out=MultiNewton(F,x0,numits,tol)

for k=1:numits
   [g,gradg,hessg]=F(x0);
   if cond(hessg)>1000000
       error('The Hessian Matrix is not invertible');
   end
```

```
   delta=hessg\gradg;  % Assumes gradg is a column vector

   xnew=x0-delta;

   d=norm(gradg);
   if d<tol
      out=xnew;
      break
   end
   x0=xnew;
end
fprintf('Newton used %d iterations\n',k);
out=xnew;
```

Here's a quick example. First, the function file:

```
function [y,dy,hy]=testfunc(x)
%  A test function for Newton's Method:
%  The input is the VECTOR x (with elements x,y below)
%
%     y = (1/4)x^4-(1/2)x^2+(1/2)y^2
%    dy = Gradient = [x^3-x; y]  (The gradient will output as a COLUMN)
%    hy = Hessian  = [3x^2-1, 0;0,1]

y=(1/4)*x(1)^4-(1/2)*x(1)^(2)+(1/2)*x(2)^2;
dy=[x(1)^3-x(1); x(2)];
hy=[3*x(1)^2-1, 0;0,1];
```

Now the function call would be something like

```
yout=MultiNewton(@testfunc,[-3;2],100,1e-6);
```

And the output will be: "Newton used 8 iterations", and `yout` would be $(-1, 0)$.

Example in Python

```
import numpy as np

def f(z):
    x, y = z

    w=(1/4)*x**4-(1/2)*x**2+(1/2)*y**2
    dw=np.array([x**3-x, y])
```

```
    hw=np.array([
         [3*x**2-1,0],
         [0,1]
         ])

    return w, dw, hw

def newton_method(f, grad_f, hess_f, x0, tol=1e-6, max_iter=20):
    x = np.array(x0, dtype=float)
    for i in range(max_iter):
        g, gradg, hessg = f(x)

        if np.linalg.cond(hessg)>1000000
            print(f"Error- Hessian is not invertible")
            return x

        if np.linalg.norm(grad) < tol:
            print(f"Converged in {i} iterations")
            return x
        delta = np.linalg.solve(hess, grad)
        x = x - delta
        print(f"Iter {i+1}: x = {x}, f(x) = {f(x)}")
    print("Did not converge")
    return x

# Initial guess
x0 = [2.0, 2.0]
minimum = newton_method(f, grad_f, hess_f, x0)
print(f"Found minimum at {minimum}, f(x) = {f(minimum)}")
```