Nonlinear equations

Introduction

Non-linear equations or *root-finding* is a problem of finding a set of n variables $\{x_1, \ldots, x_n\}$ which satisfy n equations

$$f_i(x_1, ..., x_n) = 0 \ , \ i = 1, \ldots, n \ , \tag{1}$$

where the functions f_i are generally non-linear.

Newton's method

Newton's method (also reffered to as Newton-Raphson method, after Isaac Newton and Joseph Raphson) is a root-finding algorithm that uses the first term of the Taylor series of the functions f_i to linearise the system (1) in the vicinity of a suspected root. It is one of the oldest and best known methods and is a basis of a number of more refined methods.

Suppose that the point $\mathbf{x} \equiv \{x_1, \ldots, x_n\}$ is close to the root. The Newton's algorithm tries to find the step $\Delta\mathbf{x}$ which would move the point towards the root, such that

$$f_i(\mathbf{x} + \Delta\mathbf{x}) = 0 \ , \ i = 1, \ldots, n \, . \tag{2}$$

The first order Taylor expansion of (2) gives a system of linear equations,

$$f_i(\mathbf{x}) + \sum_{k=1}^{n} \frac{\partial f_i}{\partial x_k} \Delta x_k = 0 \ , \ i = 1, \ldots, n \ , \tag{3}$$

or, in the matrix form,

$$J\Delta\mathbf{x} = -\mathbf{f}(\mathbf{x}), \tag{4}$$

where $\mathbf{f}(\mathbf{x}) \equiv \{f_1(\mathbf{x}), \ldots, f_n(\mathbf{x})\}$ and J is the matrix of partial derivatives[1],

$$J_{ik} \equiv \frac{\partial f_i}{\partial x_k} \ , \tag{5}$$

called the *Jacobian matrix*.

The solution $\Delta\mathbf{x}$ to the linear system (4)—called the Newton's step—gives the approximate direction and the step-size towards the solution.

The Newton's method converges quadratically if sufficiently close to the solution. Otherwise the full Newton's step $\Delta\mathbf{x}$ might actually diverge from the solution. Therefore in practice a more conservative step $\lambda\Delta\mathbf{x}$ with $\lambda < 1$ is usually taken. The strategy of finding the optimal λ is referred to as *line search*.

It is typically not worth the effort to find λ which minimizes $\|\mathbf{f}(\mathbf{x} + \lambda\Delta\mathbf{x})\|$ exactly, since $\Delta\mathbf{x}$ is only an approximate direction towards the root. Instead an inexact but quick minimization strategy is usually used, like the *backtracking line search* where one first attempts the full step, $\lambda = 1$, and then backtracks, $\lambda \leftarrow \lambda/2$, until the condition

$$\|\mathbf{f}(\mathbf{x} + \lambda\Delta\mathbf{x})\| < \left(1 - \frac{\lambda}{2}\right) \|\mathbf{f}(\mathbf{x})\| \tag{6}$$

is satisfied. If the condition is not satisfied for sufficiently small $\lambda_{\min}$ the step is taken with $\lambda_{\min}$ simply to step away from this diffictul place and try again. A typical algrorithm of the Newton's method with backtracking line search and condition (6) is shown in Table 1.

A somewhat more refined backtracking linesearch is based on an approximate minimization of the function

$$g(\lambda) \doteq \frac{1}{2}\|\mathbf{f}(\mathbf{x} + \lambda\Delta\mathbf{x})\|^2 \tag{7}$$

[1] in practice if derivatives are not available analytically one uses finite differences

$$\frac{\partial f_i}{\partial x_k} \approx \frac{f_i(x_1, \ldots, x_{k-1}, x_k + \delta x, x_{k+1}, \ldots, x_n) - f_i(x_1, \ldots, x_k, \ldots, x_n)}{\delta x}$$

with $\delta x \ll s$ where s is the typical scale of the problem at hand.

Table 1: Newton's algorithm with simple backtracking line search.

repeat

  solve $J\Delta\mathbf{x} = -\mathbf{f}(\mathbf{x})$ for $\Delta\mathbf{x}$

  $\lambda = 1$

  while $\|\mathbf{f}(\mathbf{x}+\lambda\Delta\mathbf{x})\| > \left(1 - \frac{\lambda}{2}\right)\|\mathbf{f}(\mathbf{x})\|$ and $\lambda > \frac{1}{64}$ do

    $\lambda = \lambda/2$

  $\mathbf{x} = \mathbf{x} + \lambda\Delta\mathbf{x}$

until converged (e.g. $\|\mathbf{f}(\mathbf{x})\| <$ tolerance)

using interpolation. The values $g(0) = \frac{1}{2}\|\mathbf{f}(\mathbf{x})\|^2$ and $g'(0) = -\|\mathbf{f}(\mathbf{x})\|^2$ are already known (check this). If the previous step with certain λ_{trial} was rejected, we also have $g(\lambda_{\text{trial}})$. These three quantities allow to build a quadratic approximation,

$$g(\lambda) \approx g(0) + g'(0)\lambda + c\lambda^2 \,, \tag{8}$$

where

$$c = \frac{g(\lambda_{\text{trial}}) - g(0) - g'(0)\lambda_{\text{trial}}}{\lambda_{\text{trial}}^2} \,. \tag{9}$$

The minimum of this approximation (determined by the condition $g'(\lambda) = 0$),

$$\lambda_{\text{next}} = -\frac{g'(0)}{2c} \,, \tag{10}$$

becomes the next trial step-size.

The procedure is repeated recursively until either condition (6) is satisfied or the step becomes too small (in which case it is taken unconditionally in order to simply get away from the difficult place).

Broyden's quasi-Newton method

The Newton's method requires calculation of the Jacobian at every iteration. This is generally an expensive operation. Quasi-Newton methods avoid calculation of the Jacobian matrix at the new point $\mathbf{x} + \delta\mathbf{x}$, instead trying to use certain approximations, typically rank-1 updates.

Broyden algorithm estimates the Jacobian $J + \delta J$ at the point $\mathbf{x} + \delta\mathbf{x}$ using the finite-difference approximation,

$$(J + \delta J)\delta\mathbf{x} = \delta\mathbf{f} \,, \tag{11}$$

where $\delta\mathbf{f} \equiv \mathbf{f}(\mathbf{x} + \delta\mathbf{x}) - \mathbf{f}(\mathbf{x})$ and J is the Jacobian at the point $\mathbf{x}$.

The matrix equation (11) is under-determined in more than one dimension as it contains only n equations to determine n^2 matrix elements of δJ. Broyden suggested to choose δJ as a rank-1 update, linear in $\delta\mathbf{x}$,

$$\delta J = \mathbf{c}\,\delta\mathbf{x}^T \,, \tag{12}$$

where the unknown vector $\mathbf{c}$ can be found by substituting (12) into (11), which gives

$$\delta J = \frac{\delta\mathbf{f} - J\delta\mathbf{x}}{\|\delta\mathbf{x}\|^2}\delta\mathbf{x}^T \,. \tag{13}$$

In practice if one wonders too far from the point where J was first calculated the accuracy of the updates may decrease significantly. In such case one might need to recalculate J anew. For example, two successive steps with $\lambda_{\min}$ might be interpreted as a sign of accuracy loss in J and subsequently trigger its recalculation.

Table 2: Javascript implementation of Newton's method

```
load('../linear/qrdec.js'); load('../linear/qrback.js');

function newton(fs,x,acc,dx){//Newton's root-finding method
   var norm2=function(v)v.reduce(function(s,e)s+e*e,0)
   var norm =function(v)Math.sqrt(v.reduce(function(s,e)s+e*e,0))
   if(acc==undefined)acc=1e-6; if(dx==undefined)dx=1e-3
   var J = [[0 for(i in x)] for(j in x)]
   var minusfx=[-fs[i](x) for (i in x)]
   do{
      for(i in x) for(k in x){// calculate Jacobian
         x[k]+=dx
         J[k][i]=(fs[i](x)+minusfx[i])/dx
         x[k]-=dx }
      var [Q,R]=qrdec(J), Dx=qrback(Q,R,minusfx)// Newton's step

/*
      var s=2
      do{ // simple backtracking linesearch
         s=s/2;
         var z=[x[i]+s*Dx[i] for(i in x)]
         var minusfz=[-fs[i](z) for(i in x)]
      }while(norm(minusfz)>(1-s/2)*norm(minusfx) && s>1./128)
*/

   var g0 = .5*norm2(minusfx);
   var gp0 = -2*g0;
   var snew = 1
do{
        var s=snew
   var z =[x[i]+s*Dx[i] for(i in x)]
   var minusfz=[-fs[i](z) for(i in x)]
   var gs = .5*norm2(minusfz);
   var c = (gs-g0-gp0*s)/s/s
   snew = -0.5*gp0/c;
}while(gs > Math.pow(1-s/2,2)*g0 && s>1./64);

      minusfx=minusfz; x=z; // step done
   }while(norm(minusfx)>acc)
   return x; }//end newton
```