1 Nonlinear equations

Nonlinear 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.

1.1 Newton's method

The 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 it is a basis of a number of more refined methods.

Suppose that the vector $\mathbf{x} \equiv \{x_1, \ldots, x_n\}$ is close to the root. Let us try to find the step $\Delta\mathbf{x}$ which would bring us to the solution,

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

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

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

or, in the matrix form,

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

where J is the matrix of partial derivatives[1],

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

called *Jacobian matrix*, and

$$\mathbf{f}(\mathbf{x}) \equiv \{f_1(\mathbf{x}), \ldots, f_n(\mathbf{x})\}\,. \tag{6}$$

The solution $\Delta\mathbf{x}$ to the linear system (4) 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 $\lambda\Delta\mathbf{x}$ is usually taken where $\lambda < 1$ is chosen (in a process called *linesearch*) to satisfy certain conditions. The simplest condition is

$$\|\mathbf{f}(\mathbf{x} + \lambda\Delta\mathbf{x})\| \le \|\mathbf{f}(\mathbf{x})\|\,. \tag{7}$$

An algrorithm of the Newton's method with backtracking linesearch and condition (7) is shown in Table 1.

Table 1: Newton's algorithm with simple backtracking linesearch.

repeat
  solve $J\Delta\mathbf{x} = -\mathbf{f}(\mathbf{x})$ for $\Delta\mathbf{x}$
  $\lambda = 1$
  while $\|\mathbf{f}(\mathbf{x} + \lambda\Delta\mathbf{x})\| > \|\mathbf{f}(\mathbf{x})\|$ and $\lambda > \frac{1}{128}$ do
    $\lambda = \lambda/2$
  $\mathbf{x} = \mathbf{x} + \lambda\Delta\mathbf{x}$
until converged (e.g. $\|\mathbf{f}(\mathbf{x})\| <$ tolerance)

1.2 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 suggested to estimate the Jacobian $J + \delta J$ at the point $\mathbf{x} + \delta\mathbf{x}$ using the finite-difference approximation

$$(J + \delta J)\delta\mathbf{x} = \delta\mathbf{f} \tag{8}$$

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

The matrix equation (8) 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 $\delta\mathbf{x}$,

$$\delta J = \mathbf{c}\,\delta\mathbf{x}^T\,, \tag{9}$$

where the unknown vector $\mathbf{c}$ can be easily found from (8), giving

$$\delta J = \frac{\delta\mathbf{f} - J\delta\mathbf{x}}{\|\delta\mathbf{x}\|^2}\delta\mathbf{x}^T\,. \tag{10}$$

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

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

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

2 Optimization

Optimization is a problem of finding minimum (or maximum) of a given real (non-linear) function $f(\mathbf{p})$ of an n-dimensional argument $\mathbf{p} = \{x_1, \ldots, x_n\}$.

2.1 Downhill simplex method

The *downhill simplex method* (also called Nelder-Mead method) is a commonnly used non-linear optimization algorithm implemented e.g. in GNU Scientific Library. The minimum of a function in an n-dimensional space is found by transforming a simplex (a polytope of $n+1$ vertexes) according to the function values at the vertexes, moving it downhill until it converges towards the minimum.

To discuss the algorithm we need the following definitions:

Simplex: a figure (polytope) represented by $n+1$ points, called vertexes, $\{\mathbf{p}_1, \ldots, \mathbf{p}_{n+1}\}$ (where each point $\mathbf{p}_k$ is an n-dimensional vector).

Highest point: the vertex, $\mathbf{p}_{\rm hi}$, with the largest value of the function: $f(\mathbf{p}_{\rm hi}) = \max_{(k)} f(\mathbf{p}_k)$.

Lowest point: the vertex, $\mathbf{p}_{\rm lo}$, with the smallest value of the function: $f(\mathbf{p}_{\rm lo}) = \min_{(k)} f(\mathbf{p}_k)$.

Centroid: the center of gravity of all points, except for the highest: $\mathbf{p}_{\rm ce} = \frac{1}{n}\sum_{(k\neq{\rm hi})} \mathbf{p}_k$

The simplex is moved downhill by a combination of the following elementary operations:

Reflection: The highest point is reflected against the centroid, $\mathbf{p}_{\rm hi} \to \mathbf{p}_{\rm re} = \mathbf{p}_{\rm ce} + (\mathbf{p}_{\rm ce} - \mathbf{p}_{\rm hi})$.

Expansion: The lowest point doubles its distance from the centroid, $\mathbf{p}_{\rm lo} \to \mathbf{p}_{\rm ex} = \mathbf{p}_{\rm ce} + 2(\mathbf{p}_{\rm lo} - \mathbf{p}_{\rm ce})$.

Contraction: The highest point halves its distance from the centroid, $\mathbf{p}_{\rm hi} \to \mathbf{p}_{\rm co} = \mathbf{p}_{\rm ce} + \frac{1}{2}(\mathbf{p}_{\rm hi} - \mathbf{p}_{\rm ce})$.

Reduction: All points, except for the lowest, move towards the lowest points halving the distance. $\mathbf{p}_{k\neq{\rm lo}} \to \frac{1}{2}(\mathbf{p}_k + \mathbf{p}_{\rm lo})$.

Finally, Table 2 shows a possible algorithm for the downhill simplex method.

Table 2: Downhill simplex (Nelder-Mead) algorithm for non-linear multidimensional optimization.

```
repeat
    try reflection
    if f(p_re) < f(p_lo)
        accept reflection and do expansion
    elseif f(p_re) < f(p_hi)
        accept reflection
    else
        try contraction
        if f(p_co) < f(p_hi)
            accept contraction
        else
            do reduction
until converged (e.g. size(simplex)<tolerance)
```