

# Nonlinear equations

## Introduction

*Non-linear equations* (or *root-finding*) is a problem of finding a set of  $n$  variables  $\mathbf{x} = \{x_1, \dots, x_n\}$  which satisfy a system of  $n$  (generally) non-linear equations

$$f_i(x_1, \dots, x_n) = 0 \Big|_{i=1, \dots, n} . \quad (1)$$

In matrix notation the system is written as

$$\mathbf{f}(\mathbf{x}) = 0 , \quad (2)$$

where  $\mathbf{f} \doteq \{f_1, \dots, f_n\}$ .

In one-dimensional cases it is generally possible to plot the function in the region of interest and see whether the graph crosses the  $x$ -axis. One can then be sure the root exists and even figure out its approximate position to start one's root-finding algorithm with. In multi-dimensions one generally does not know if the root exists at all, until it is found.

The root-finding algorithms generally proceed by iteration, starting from some approximate solution and making consecutive steps (hopefully) in the direction of the suspected root until some convergence criterion is satisfied. The procedure is generally not even guaranteed to converge unless starting from a point close enough to the sought root.

We shall only consider the multi-dimensional case here since i) the multi-dimensional root-finding is more difficult, and ii) the multi-dimensional routines can also be used in a one-dimensional case.

## Newton's method

Newton's method (also referred 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  $\mathbf{x} = \{x_1, \dots, x_n\}$  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

$$f_i(\mathbf{x} + \Delta\mathbf{x}) = 0 \Big|_{i=1, \dots, n} . \quad (3)$$

The first order Taylor expansion of (3) gives a system of linear equations,

$$f_i(\mathbf{x}) + \sum_{k=1}^n \frac{\partial f_i}{\partial x_k} \Delta x_k = 0 \Big|_{i=1, \dots, n} , \quad (4)$$

or, in the matrix form,

$$\mathbf{J}\Delta\mathbf{x} = -\mathbf{f}(\mathbf{x}), \quad (5)$$

where  $\mathbf{f}(\mathbf{x}) \doteq \{f_1(\mathbf{x}), \dots, f_n(\mathbf{x})\}$  and  $\mathbf{J}$  is the matrix of partial derivatives<sup>1</sup>,

$$J_{ik} \doteq \frac{\partial f_i}{\partial x_k} , \quad (6)$$

called the *Jacobian matrix*.

The solution  $\Delta\mathbf{x}$  to the linear system (5)—called the Newton's step—gives the approximate direction and the step-size towards the solution.

---

<sup>1</sup>in practice if derivatives are not available analytically one uses finite differences

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

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

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  $\lambda$  is referred to as *line search*.

It is typically not worth the effort to find  $\lambda$  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

$$\|\mathbf{f}(\mathbf{x} + \lambda \Delta \mathbf{x})\| < \left(1 - \frac{\lambda}{2}\right) \|\mathbf{f}(\mathbf{x})\| \quad (7)$$

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 difficult place and try again. A typical algorithm of the Newton's method with backtracking line search and condition (7) is shown in Table 1.

Table 1: Newton's root-finding algorithm with backtracking.

---

```

repeat
  solve  $\mathbf{J}\Delta \mathbf{x} = -\mathbf{f}(\mathbf{x})$  for  $\Delta \mathbf{x}$ 
   $\lambda \leftarrow 1$ 
  while (  $\|\mathbf{f}(\mathbf{x} + \lambda \Delta \mathbf{x})\| > (1 - \frac{\lambda}{2}) \|\mathbf{f}(\mathbf{x})\|$  and  $\lambda > \frac{1}{64}$  ) do  $\lambda \leftarrow \lambda/2$ 
   $\mathbf{x} \leftarrow \mathbf{x} + \lambda \Delta \mathbf{x}$ 
until converged (e.g.  $\|\mathbf{f}(\mathbf{x})\| < \text{tolerance}$ )

```

---

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

$$g(\lambda) \doteq \frac{1}{2} \|\mathbf{f}(\mathbf{x} + \lambda \Delta \mathbf{x})\|^2 \quad (8)$$

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  $\lambda_{\text{trial}}$  was rejected, we also have  $g(\lambda_{\text{trial}})$ . These three quantities allow to build a quadratic approximation,

$$g(\lambda) \approx g(0) + g'(0)\lambda + c\lambda^2, \quad (9)$$

where

$$c = \frac{g(\lambda_{\text{trial}}) - g(0) - g'(0)\lambda_{\text{trial}}}{\lambda_{\text{trial}}^2}. \quad (10)$$

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

$$\lambda_{\text{next}} = -\frac{g'(0)}{2c}, \quad (11)$$

becomes the next trial step-size.

The procedure is repeated recursively until either condition (7) 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 matrix 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's algorithm estimates the Jacobian  $\mathbf{J} + \Delta \mathbf{J}$  at the point  $\mathbf{x} + \Delta \mathbf{x}$  using the finite-difference approximation,

$$(\mathbf{J} + \Delta \mathbf{J})\Delta \mathbf{x} = \Delta \mathbf{f}, \quad (12)$$

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

The matrix equation (12) is under-determined in more than one dimension as it contains only  $n$  equations to determine  $n^2$  matrix elements of  $\Delta \mathbf{J}$ . Broyden suggested to choose  $\Delta \mathbf{J}$  as a rank-1 update, linear in  $\Delta \mathbf{x}$ ,

$$\Delta \mathbf{J} = \mathbf{c} \Delta \mathbf{x}^T, \quad (13)$$

Table 2: C implementation of Newton's algorithm with backtracking.

```

#include<matrix.h> // garbage collected matrices and vectors
#include<qr.h>
vector* newton
(vector* f(vector*x), vector* startx, vector* dx, double eps)
{
    vector *x=vector_copy(startx), *fx, *Dx;
    int n=x->size;
    matrix *A=matrix_alloc(n,n), *R=matrix_alloc(n,n);
    do
    {
        fx=f(x); // : gets garbage collected
        for(int j=0;j<n;j++) // calculating the Jacobian matrix:
        {
            vector_set(x,j,vector_get(x,j)+vector_get(dx,j));
            vector* df=vector_sum(1,f(x),-1,fx);
            for(int i=0;i<n;i++)
                matrix_set(A,i,j,vector_get(df,i)/vector_get(dx,j));
            vector_set(x,j,vector_get(x,j)-vector_get(dx,j));
        }
        qrdec(A,R);
        vector* minusfx=vector_copy(fx); vector_scale(minusfx,-1);
        Dx=qrback(A,R,minusfx); // : Newton's step
        double lambda=2, normfx=vector_norm(fx);
        vector *z, *fz;
        do // back-tracking line-search:
        {
            lambda/=2; z=vector_sum(1,x,lambda,Dx); fz=f(z);
        }while(vector_norm(fz)>(1-lambda/2)*normfx && lambda>0.02);
        x=z; fx=fz;
    }while(vector_norm(Dx)>vector_norm(dx) && vector_norm(fx)>eps);
    return x;
}

```

where the unknown vector  $\mathbf{c}$  can be found by substituting (13) into (12), which gives

$$\Delta \mathbf{J} = \frac{\Delta \mathbf{f} - \mathbf{J} \Delta \mathbf{x}}{\|\Delta \mathbf{x}\|^2} \Delta \mathbf{x}^T. \quad (14)$$

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