CHAPTER 3. NONLINEAR EQUATIONS

This chapter is based on Chapter 3 of the book by Miranda and Fackler.

Nonlinear equations generally arise in one of two forms. In the nonlinear *rootfinding problem,*a function *f* from R^n to R^n is given, and one must compute a *n*-vector *x*, called a *root* of *f*, that satisfies

$f(x)=0$

In the nonlinear *fixed-point problem*, a function *g* from R^n to R^n is given, and one must compute a *n*-vector *x*, called a *fixed-point* of *g*, that satisfies

$x=g(x)$

The two forms are equivalent. The rootfinding problem may be recast as a fixed-point problem by letting $g(x)=x-f(x)$; conversely, the fixed-point problem may be recast as a rootfinding problem by letting $f(x)=x-g(x)$.

1. Bisection method

The bisection method is the simplest and most robust method for computing the root of a continuous real-valued function defined on a bounded interval of the real line. If *f* is continuous, and $f(a)$ and $f(b)$ have different signs, then *f* must have at least one root *x* in [*a*,*b*].

The bisection method is an iterative procedure. Each iteration begins with an interval known to contain a root of *f*, because the function has different signs at the interval endpoints. The interval is bisected into two subintervals of equal length. One of the two subintervals must have endpoints of different signs and thus must contain a root of *f*. This subinterval is taken as the new interval with which to begin the subsequent iteration. In this manner, a sequence of intervals is generated, each half the width of the preceding one, and each known to contain a root of *f*. The process continues until the width of the bracketing interval containing a root shrinks below an acceptable tolerance.

We will use this method to find the root of the function $f(x) = x^3 - 2$. We have *f(1)=-1<0 and f(2)=8-2=2>0.* So, we set *a=1* and *b=2.*

First, we create a function M-file called f.

```
function y=f(x)
y=x^3-2;
```

Then, we set in the command window

>> a=1;b=2;

Then, we define the convergence tolerance

>>tol=1e-09;

Then, we write the program of the method in a script M-file called bisection

```
s=sign(f(a));
x=(a+b)/2;
d=(b-a)/2;
iter=0;
while d>tol
    d=d/2;d
    if s == sign(f(x))
        x=x+d;
    else
        x=x-d;
    end
    iter=iter+1;
end;
iter, x
```

The function sign returns -1, 0, or 1 if its argument is negative, zero, or positive, respectively.

d will be printed. So, we can see how the interval bracketing the root becomes smaller and smaller.

iter is a counter, which will show how many loops are necessary to compute the root of equation *f*.

We get

```
d =
  9.3132e-010


iter =
    29


x =
    1.2599
```

Instead of writing the bisection program, we could have used the function bisect of Compecon

```
>> x=bisect('f',1,2)
```

The first input is the name of the function M-file, which computes the value of function *f(x).* f is put between quotes '' because the name of this file is a script. The two other inputs are the values of *a* and *b* , which define the original bracket where we look for a root of *f(x).*

2. Function Iteration

We remind that in the nonlinear *fixed-point problem,* a function *g* from R^n to R^n is given, and one must compute a *n*-vector *x,* called a *fixed-point* of *g*, that satisfies

$x=g(x)$

Function iteration begins with the analyst supplying a guess $x^{(0)}$ for the fixed point of *g*. Subsequent iterates are generated using the simple iteration rule

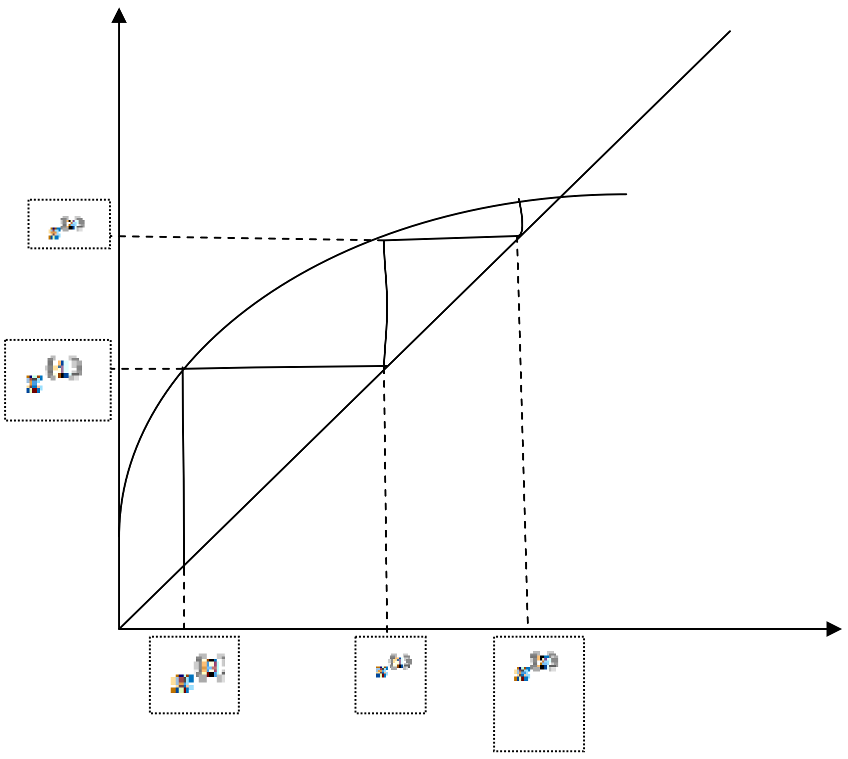
$$x^{(k+1)} = g\left(x^{(k)}\right)$$

If the iterates converge they converge to a fixed point of *g*.

In theory, function iteration is guaranteed to converge to a fixed point of *g* if the initial value of *x* supplied by the analyst is "sufficiently" close to a fixed point *x** of *g* at which $\left|g'(x*)\right| < 1$. Function iteration however, often converges, even when the sufficient conditions are not met. The main advantage of this method is that it is easy to implement.

We can illustrate this method in the case of a univariate function *g(x).* The graph of function *g(x)* intersects the 45 degrees line at point (*x**, *x**).

g



The example will assume that $g(x) = x^{0.5} + 2$. We first create a function M-file g

```
function y=g(x)
y=x^0.5+2;
```

Then, we create the script M-file fixedpoint

```
maxit = 100;
tol   = 1e-10;
x=0;
```

```
for it=1:maxit
   gval = g(x);
   if norm(gval-x)<tol
       it,gval, x
       return
   end
   x = gval;
end
warning('Failure to converge in fixpoint')
```

```
We get the result

>>

it =

    20


gval =

    4.0000


x =

    4.0000
```

3. Newton's Method

Most nonlinear models are solved using *Newton's method* or one of its variants. Newton's method is based on the principle of *successive linearization*. Successive linearization calls for a hard nonlinear problem to be replaced with a sequence of simpler linear problems whose solutions converge to the solution of the nonlinear problem.

Let *f* be a function from R^n to R^n. We must compute a *n*-vector *x* that satisfies *f(x)=0*. The Newton's method begins with the analyst supplying a guess $x^{(0)}$ for the root of *f*. Given $x^{(k)}$, the subsequent iterate $x^{(k+1)}$ is computed by solving the linear rootfinding problem obtained by replacing *f* with its first-order Taylor approximation about $x^{(k)}$:

$$f(x) \approx f(x^{(k)}) + f'(x^{(k)})(x - x^{(k)}) = 0 \qquad (1)$$

This approach yields the iteration rule

$$x^{(k+1)} = x^{(k)} - [f'(x^{(k)})]^{-1} f(x^{(k)}) \qquad (2)$$

We can see that $x^{(k+1)}$ is the root of the linear approximation (1). We can compute it by the methods presented in chapter 2, for example a L-U factorization. $f(x^{(k)})$ is a vector of dimension *n* and $f'(x^{(k)})$, which is the Jacobian of $f(x^{(k)})$, is a matrix of dimension *nxn.*

However, a root of the linear approximation (1) of *f(x)* is not a root of *f(x)*.Thus, we have to go through a sequence of iterations.

In theory, Newton's method converges if *f* is continuously differentiable and if the initial value of *x* is "sufficiently" close to a root of *f* at which *f'* is invertible. There are nice functions, for which the choice of the initial guess has no consequence. However, there are nasty functions, which behave erratically, and such that the Newton's method will diverge except for initial guesses quite near the solution we are trying to compute.

The following MATLAB script, NEWTON.m, computes the root of a function *f* using Newton's method. It assumes that the user has provided an initial guess x for the root, a convergence tolerance tol, and an upper limit maxit on the number of iterations. It calls user-supplied routine ff that computes the value fval and the Jacobian fjac of the function at an arbitrary point x.

```
for it=1:maxit
```

```
    [fval,fjac]=ff(x);
    x=x-fjac\fval;
    if norm(fval)<tol, break, end
end
it
```

This script will be used to compute a very simple Cournot's equilibrium. We have to compute the solution of the two equations

$$f_i(q) = (q_1 + q_1)^{-\frac{1}{\eta}} - \left(\frac{1}{\eta}\right)(q_1 + q_1)^{-\frac{1}{\eta}-1} q_i - c_i q_i = 0$$, for *i=1,2*

q_i is the output of firm *i* , η is the price-elasticity of the demand function and c_i is the marginal cost of firm *i*. We assume $\eta = 1.6$, $c_1 = 0.6$ and $c_2 = 0.8$.

We write the function M-file ff

```
function [fval,fjac]=ff(q)
c=[0.6;0.8]; eta=1.6; e=-1/eta;
fval=sum(q)^e+e*sum(q)^(e-1)*q-diag(c)*q;
fjac=e*sum(q)^(e-1)*ones(2,2)+e*sum(q)^(e-1)*eye(2)...
    +(e-1)*e*sum(q)^(e-2)*q*[1 1]-diag(c);
```

Finally, we type in the command window

```
>> x=[0.2 ;0.2];tol=1e-10;maxit=20;

>> NEWTON; x
```

The initial guess is an output of 0.2 for each firm. We get

```
it =

    6
```

```
x =

    0.8396

    0.6888
```

The algorithm converges in 6 iterations. The outputs of firms 1 and 2 respectively are 0.8396 and 0.6888. You remember that firm 2 has a higher marginal cost than firm 1.

We could also have used the function newton.m of Compecon

```
q=newton('ff',[0.2;0.2]);

>> q
```

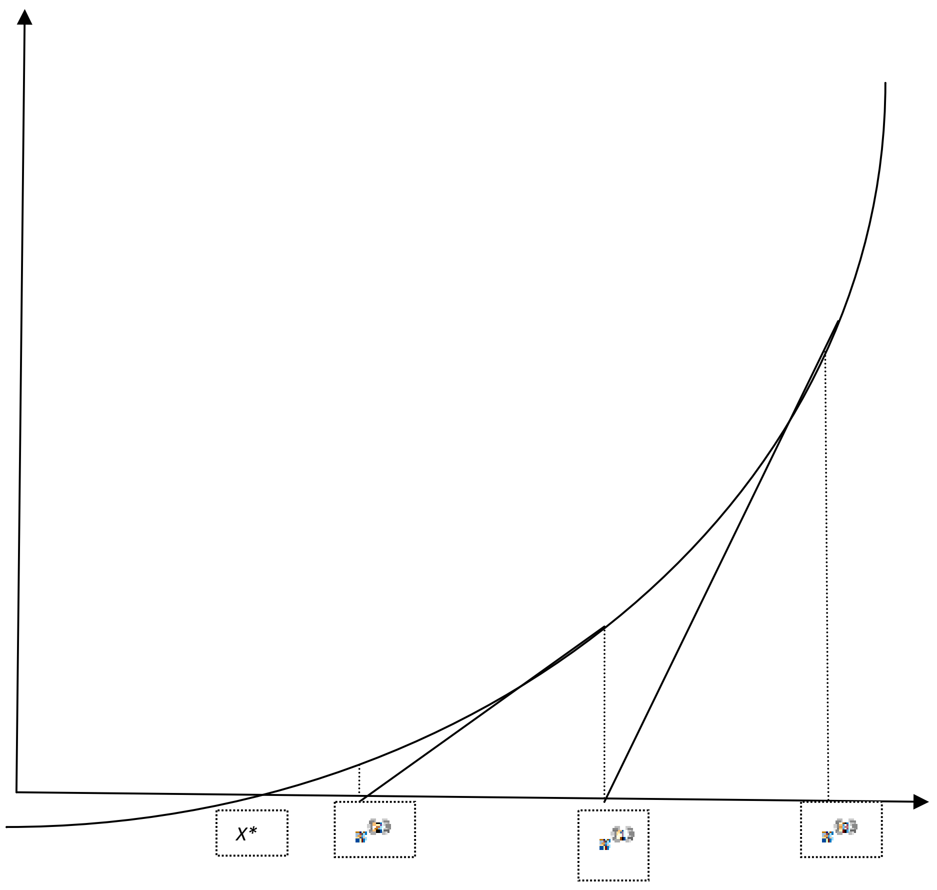
We get

```
q =

    0.8396

    0.6888
```

The following graph gives an illustration of the Newton's method in the univariate case



The algorithm begins with the analyst supplying a guess $x^{(0)}$ for the root of *f*. The function *f* is then approximated by its first-order Taylor series expansion about $x^{(0)}$, which is graphically represented by the line tangent to *f* at $x^{(0)}$. The root $x^{(1)}$ of the tangent line is then accepted as an improved estimate for the root of *f*. The step is repeated, with the root $\mathbf{x}^{(2)}$ of the line tangent to *f* at $x^{(1)}$ taken as an improved estimate for the root of *f* , and so on. The process continues until the roots of the tangent line converge.

4. Quasi-Newton Methods

Quasi-Newton methods are based on the same successive linearization principle as Newton's method, except that they replace the Jacobian *f'* with an approximation that is easier to compute. Computing this Jacobian at each iterations may be cumbersome and the interest of the Quasi-Newton method is avoiding this computation. The price to pay for this simplification is that Quasi-Newton methods converge more slowly than Newton's methods.

The *secant method* is the most widely used univariate quasi-Newton method. The secant method is identical to the univariate Newton method, except that it replaces the derivative of *f* with an approximation constructed from the function values at the two previous iterates

$$f'\left(x^{(k)}\right) \approx \frac{f\left(x^{(k)}\right) - f\left(x^{(k-1)}\right)}{x^{(k)} - x^{(k-1)}}$$

This approximation of derivative $f'\left(x^{(k)}\right)$ is good when $x^{(k)} - x^{(k-1)}$ is small. However, in the first iterations, this difference may be large. Moreover, unlike the Newton method, the secant method requires two starting values rather than one, or a guess on the solution and another guess on the value of the derivative at the solution.

Finally, we use the iteration rule

$$x^{(k+1)} = x^{(k)} - \frac{x^{(k)} - x^{(k-1)}}{f\left(x^{(k)}\right) - f\left(x^{(k-1)}\right)} f\left(x^{(k)}\right)$$

Broyden's method is the most popular multivariate generalization of the univariate secant method. Broyden's method generates a sequence of vectors $x^{(k)}$ and matrices $A^{(k)}$ that approximate the root of *f* and the Jacobian *f'* at the root respectively. Broyden's method begins with the analyst supplying a guess $x^{(0)}$ for the root of the function and a guess $A^{(0)}$ for the Jacobian of the function at the root. Often, $A^{(0)}$ is set equal to the numerical Jacobian of *f* at $x^{(0)}$.

We have to explain what a numerical Jacobian is. We will limit our explanation to the case when *f* and *x* have one dimension. The extension to the case of *n* dimensions is trivial. The definition of a derivative is

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

This definition suggest to compute the derivative of function *f* at *x* by the expression

$f'(x) \approx \dfrac{f(x+h) - f(x)}{h}$, where *h* is a small number.

As *f(x)* has probably been computed yet, the computation of the value of its derivative requires to compute one more values of this function, which is *f(x+h).*

A more precise approximation of the derivative is given by

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}$$

However, this expression requires the computation of two (and not only one) values of *f* : *f(x+h)* and *f(x-h).* Section 5.6 of the book by Miranda and Fackler gives more explanation. Let us come back to Broyden’s method.

Given $x^{(k)}$ and $A^{(k)}$, one updates the root approximation by solving the linear rootfinding problem obtained by replacing *f* with its first-order Taylor approximation about $x^{(k)}$

$$f(x) = f\left(x^{(k)}\right) + A^{(k)}\left(x - x^{(k)}\right) = 0$$

This step yields the root approximation iteration rule

$$x^{(k+1)} = x^{(k)} - \left(A^{(k)}\right)^{-1} f\left(x^{(k)}\right)$$

Broyden's method then updates the Jacobian approximant $A^{(k)}$ by using the following extension of the *secant condition*

$$f\left(x^{(k+1)}\right) - f\left(x^{(k)}\right) = A^{(k+1)}\left(x^{(k+1)} - x^{(k)}\right)$$

However, this condition imposes *n* conditions to compute the *nxn* matrix $A^{(k+1)}$. To solve this indetermination we will retain among all the matrices $A^{(k+1)}$ consistent with this condition, the one, which is the nearest of $A^{(k)}$. Finally, we have the iteration rule (which satisfies the previous condition)

$$A^{(k+1)} = A^{(k)} + \left[f\left(x^{(k+1)}\right) - f\left(x^{(k)}\right) - A^{(k)}d^{(k)}\right]\frac{d^{(k)\prime}}{d^{(k)\prime}d^{(k)}}, \text{ with } d^{(k)} = x^{(k+1)} - x^{(k)}$$

CompEcon includes a routine broyden. We will uses it on the same problem as in last section

```
>> q=broyden('ff',[0.2;0.2]);

>> q



q =

   0.8396

   0.6888
```

5. **Tricks and advices**

We remind that the Newton's method is based on the iteration rule

$$x^{(k+1)} = x^{(k)} + d^{(k)} = x^{(k)} - \left[f'\left(x^{(k)}\right)\right]^{-1} f\left(x^{(k)}\right)$$

This rule can lead to a huge change $x^{(k+1)} - x^{(k)}$, which can lead $x^{(k+1)}$ further from the root *x** than $x^{(k)}$, especially in the beginning of the iteration process if $x^{(0)}$ has been taken far from *x**. Thus, the iteration rule will diverge. We can see that by noticing that the Euclidean norm $\left\| f\left(x^{(k)}\right)\right\|$ increases with *k* instead of decreasing. A solution to this problem is to use the iteration rule

$$x^{(k+1)} = x^{(k)} + \lambda d^{(k)} = x^{(k)} - \lambda\left[f'\left(x^{(k)}\right)\right]^{-1} f\left(x^{(k)}\right), \text{ with } 0 < \lambda < 1,$$

This dampening of the iteration rule will increase the number of steps but will solve the problem of divergence at the beginning of the iteration process.

Miranda and Fackler automate the choice of the value of parameter λ. Its default value is 1. But this value is divided by 2 each time the Euclidean norm $\left\| f\left(x^{(k)}\right)\right\|$ would otherwise increase.

A sequence of iterate $x^{(k)}$ is said to converge to *x** at a rate of order *p* if there is a constant *C*>0 such that

$$\left\|x^{(k+1)} - x^*\right\| \le C\left\|x^{(k)} - x^*\right\|^p$$

for sufficiently large *k.* In particular, the rate of convergence is said to be *linear* if *C*<1 and *p*=1, *superlinear* if 1<*p*<2, and *quadratic* if *p*=2.

The bisection method converges at a linear rate with C=1/2. The function iteration method converges at a linear arte with C equal to $\|f'(x*)\|$. The secant AND Broyden methods converge at a superlinear rate with $p \approx 1.62$. And Newton's method converges at a quadratic rate. The rates of convergence are asymptotically valid, provided that the algorithms are given "goo' initial data.

The rate of convergence is not the only criteria for choosing a method. For instance, each iteration of the Newton's method asks for more computation than the other method. The computation of the Jacobian can sometimes be very difficult, etc.