3.5 Solution of Nonlinear Systems

We now return to the subject of Chapter 3, and discuss the solution of systems of nonlinear equations. An important ingredient will be the multivariate Taylor theorem.

**Theorem 3.9** Let \( D = \{[x_1, x_2, \ldots, x_d]^T \in \mathbb{R}^d : a_i \leq x_i \leq b_i, \ i = 1, \ldots, d\} \) for some \( a_1, a_2, \ldots, a_d, b_1, b_2, \ldots, b_d \in \mathbb{R}. \) If \( f \in C^{n+1}(D) \), then for \( x + h \in D \) \((h = [h_1, h_2, \ldots, h_d]^T)\)

\[
f(x + h) = \sum_{k=0}^{n} \frac{1}{k!} (h \cdot \nabla)^k f(x) + R_n(h), \tag{37}
\]

where

\[
R_n(h) = \frac{1}{(n+1)!} (h \cdot \nabla)^{n+1} f(x + \theta h)
\]

with \(0 < \theta < 1\) and \(\nabla = \left[ \frac{\partial}{\partial x_1}, \frac{\partial}{\partial x_2}, \ldots, \frac{\partial}{\partial x_d} \right]^T\).

**Remark:** See page 11 of the textbook for the bivariate version of Theorem 3.9.

**Example:** We are particularly interested in the linearization of a given function, i.e., \(n = 1\). In this case we have

\[
f(x + h) = f(x) + (h \cdot \nabla) f(x) + \frac{1}{2} (h \cdot \nabla)^2 f(x + \theta h). \]

And, for \(d = 2\), this becomes

\[
f(x_1 + h_1, x_2 + h_2) = f(x_1, x_2) + h_1 \frac{\partial}{\partial x_1} f(x_1, x_2) + h_2 \frac{\partial}{\partial x_2} f(x_1, x_2) + \frac{1}{2} \left( h_1 \frac{\partial}{\partial x_1} + h_2 \frac{\partial}{\partial x_2} \right)^2 f(x_1 + \theta h_1, x_2 + \theta h_2)
\]

Therefore, the linearization of \(f\) is given by

\[
f(x_1 + h_1, x_2 + h_2) = f(x_1, x_2) + \left( h_1 \frac{\partial}{\partial x_1} + h_2 \frac{\partial}{\partial x_2} \right) f(x_1, x_2) + \frac{1}{2} \left( h_1^2 \frac{\partial^2}{\partial x_1^2} + 2h_1h_2 \frac{\partial^2}{\partial x_1 \partial x_2} + h_2^2 \frac{\partial^2}{\partial x_2^2} \right) f(x_1 + \theta h_1, x_2 + \theta h_2).
\]

We now want to solve the following system of nonlinear equations:

\[
\begin{align*}
f_1(x_1, x_2, \ldots, x_n) &= 0, \\
f_2(x_1, x_2, \ldots, x_n) &= 0, \\
&\vdots \\
f_n(x_1, x_2, \ldots, x_n) &= 0. \tag{38}
\end{align*}
\]

To derive Newton’s method for this problem we assume \(r = [r_1, r_2, \ldots, r_n]^T\) is a solution (or root) of (38), i.e., \(r\) satisfies

\[
f_i(r) = 0, \quad i = 1, \ldots, n.
\]
Moreover, we consider $x$ to be an approximate root, i.e.,

$$x + h = r,$$

with a small correction $h = [h_1, h_2, \ldots, h_n]^T$. Then, by linearizing $f_i$, $i = 1, \ldots, n$,

$$f_i(r) = f_i(x + h) \approx f_i(x) + (h \cdot \nabla)f_i(x).$$

Since $f_i(r) = 0$ we get

$$-f_i(x) \approx (h \cdot \nabla)f_i(x) = \left(h_1 \frac{\partial}{\partial x_1} + h_2 \frac{\partial}{\partial x_2} + \ldots + h_n \frac{\partial}{\partial x_n}\right)f_i(x).$$

Therefore, we have a linearized version of system (38) as

$$-f_1(x_1, \ldots, x_n) = \left(h_1 \frac{\partial}{\partial x_1} + \ldots + h_n \frac{\partial}{\partial x_n}\right)f_1(x_1, \ldots, x_n),$$

$$-f_2(x_1, \ldots, x_n) = \left(h_1 \frac{\partial}{\partial x_1} + \ldots + h_n \frac{\partial}{\partial x_n}\right)f_2(x_1, \ldots, x_n),$$

$$\vdots$$

$$-f_n(x_1, \ldots, x_n) = \left(h_1 \frac{\partial}{\partial x_1} + \ldots + h_n \frac{\partial}{\partial x_n}\right)f_n(x_1, \ldots, x_n). \quad (39)$$

Recall that $h = [h_1, \ldots, h_n]^T$ is the unknown Newton update, and note that (39) is a linear system for $h$ of the form

$$J(x)h = -f(x),$$

where $f = [f_1, \ldots, f_n]^T$ and

$$J = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}$$

is called the Jacobian of $f$.

The algorithm for Newton’s method for square nonlinear systems is now

**Algorithm**

Input $f$, $J$, $x^{(0)}$, $M$

for $k = 0$ to $M$ do

Solve $J(x^{(k)})h^{(k)} = -f(x^{(k)})$ for $h^{(k)}$

Update $x^{(k+1)} = x^{(k)} + h^{(k)}$

end

Output $x^{(k+1)}$
**Remark:** If we symbolically write $f'$ instead of $J$, then the Newton iteration becomes

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

which looks just like the Newton iteration formula for the single equation/single variable case.

**Example:** Solve

$$\begin{align*}
x^2 + y^2 &= 4 \\
xy &= 1
\end{align*}$$

which corresponds to finding the intersection points of a circle and a hyperbola in the plane. Here

$$f(x, y) = \begin{bmatrix} f_1(x, y) \\
f_2(x, y) \end{bmatrix} = \begin{bmatrix} x^2 + y^2 - 4 \\
xy - 1 \end{bmatrix}$$

and

$$J(x, y) = \begin{bmatrix} \frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} \\
\frac{\partial f_2}{\partial x} & \frac{\partial f_2}{\partial y} \end{bmatrix} (x, y) = \begin{bmatrix} 2x & 2y \\
y & x \end{bmatrix}.$$ 

This example is illustrated in the Matlab script `run_newtonmv.m`.

**Remarks:**

1. Newton’s method requires the user to input the $n \times n$ Jacobian matrix (which depends on the specific nonlinear system to be solved). This is rather cumbersome.

2. In each iteration an $n \times n$ (dense) linear system has to be solved. This makes Newton’s method very expensive and slow.

3. For “good” starting values Newton’s method converges quadratically to simple zeros, i.e., solutions for which $J^{-1}(r)$ exists.

4. An improvement which removes the strong dependence on the choice of starting values is the so-called line search

$$x^{(k+1)} = x^{(k)} + \lambda_k h^{(k)},$$

where $\lambda_k \in \mathbb{R}$ is chosen so that $f^T f(x^{(k)})$ is strictly monotone decreasing. In this case $x^{(k)}$ converges to the minimum of $f^T f$. This method stems from an interpretation of the solution of nonlinear systems as the minimizer of a nonlinear function (more later).

**Basic Fixed-point Iteration**

We illustrate the use of a general fixed-point algorithm with several examples in the Maple worksheet `577_fixedpointsMV.mws`. However, as we well know, this may not always be possible, and if it is, convergence may be very slow. Sometimes we can use a Gauss-Seidel like strategy to accelerate convergence.

A multivariate version of the Contractive Mapping Theorem (Theorem 3.6 in the notes) is
**Theorem 3.10** Let $C$ be a closed subset of $\mathbb{R}^n$ and $F$ a contractive mapping of $C$ into itself. Then $F$ has a unique fixed point $s$. Moreover, $s = \lim_{k \to \infty} x^{(k)}$, where $x^{(k+1)} = F(x^{(k)})$ and $x^{(0)}$ is any starting point in $C$.

Here a contractive map is defined analogous to Definition 3.3.

**Quasi-Newton Methods**

In the multivariate (systems) setting the extra burden associated with using the derivative (Jacobian) of $f$ becomes much more obvious than in the single equation/single variable case discussed earlier. In the algorithm listed above we need to perform $n^2$ evaluations of derivatives (for the Jacobian) and $n$ evaluations of $f$ (for the right-hand side) in each iteration. Moreover, solving the linear system $J(x)h = f(x)$ usually requires $O(n^3)$ floating point operations per iteration.

In order to reduce the computational complexity we need to apply a strategy analogous to the secant method used for single equations. This will eliminate evaluations of derivatives, and reduce the number of floating point iterations required to compute the Newton update to $O(n^2)$ operations per iteration.

The idea is to provide an initial approximation $B^{(0)}$ to $[J(x^{(0)})]^{-1}$, and then update this approximation from one iteration to the next, i.e.,

$$B^{(k+1)} = B^{(k)} + U^{(k)},$$

where $U^{(k)}$ is an appropriately chosen update.

**Remark:** This replaces solution of the linear system $J(x^{(k)})h = f(x^{(k)})$.

One way of updating was suggested by Broyden and is based on the Sherman-Morrison formula for matrix inversion.

**Lemma 3.11** Let $A$ be a nonsingular $n \times n$ matrix and $x, y \in \mathbb{R}^n$. Then $(A + xy^T)^{-1}$ exists provided that $y^TA^{-1}x \neq -1$. Moreover,

$$(A + xy^T)^{-1} = A^{-1} - \frac{A^{-1}xy^TA^{-1}}{1 + y^TA^{-1}x}.$$  \hspace{1cm} (40)

**Proof:** See final exam. ♠

**Remark:** The Sherman-Morrison formula (40) can be used to compute the inverse of a matrix $A^{(k+1)}$ obtained by a rank-1 update $xy^T$ from $A^{(k)}$, i.e.,

$$[A^{(k+1)}]^{-1} = [A^{(k)}]^{-1} - \frac{[A^{(k)}]^{-1}xy^T[A^{(k)}]^{-1}}{1 + y^T[A^{(k)}]^{-1}x}. \hspace{1cm} (41)$$

Thus, if $A^{(k+1)}$ is a rank-1 modification of $A^{(k)}$ then we need not recompute the inverse of $A^{(k+1)}$, but instead can obtain it by updating the inverse of $A^{(k)}$ (available from previous computations) via (41).
The algorithm for Broyden’s method is

**Algorithm**

Input $f$, $x^{(0)}$, $B^{(0)}$, $M$

for $k = 0$ to $M$ do

$$h^{(k)} = -B^{(k)} f(x^{(k)})$$

$$x^{(k+1)} = x^{(k)} + h^{(k)}$$

$$z^{(k)} = f(x^{(k+1)}) - f(x^{(k)})$$

$$B^{(k+1)} = B^{(k)} - \left( \frac{B^{(k)} z^{(k)} - h^{(k)}}{\|h^{(k)}\|_2} \right) \left[ h^{(k)} \right]^T B^{(k)}$$

end

Output $x^{(k+1)}$

**Remarks:**

1. Only $n$ scalar function evaluations are required per iteration along with $O(n^2)$ floating point operations for matrix-vector products.

2. One can usually use $B^{(0)} = I$ to start the iteration.

In order to see how the formula for $B^{(k+1)}$ in the algorithm is related to (41) we define

$$\left[ J(x^{(k)}) \right]^{-1} \approx \left[ A^{(k)} \right]^{-1} = B^{(k)}$$

$$x = \frac{z^{(k)} - \left[ B^{(k)} \right]^{-1} h^{(k)}}{\|h^{(k)}\|_2^2},$$

$$y = h^{(k)}.$$

Then (41) becomes

$$B^{(k+1)} = B^{(k)} - \frac{B^{(k)} z^{(k)} - h^{(k)}}{\|h^{(k)}\|_2^2 + [h^{(k)}]^T B^{(k)} z^{(k)} - [h^{(k)}]^T B^{(k)} \left[ B^{(k)} \right]^{-1} h^{(k)}}$$

$$= B^{(k)} - \frac{B^{(k)} z^{(k)} - h^{(k)}}{\|h^{(k)}\|_2^2 + [h^{(k)}]^T B^{(k)} z^{(k)} - [h^{(k)}]^T B^{(k)} \left[ B^{(k)} \right]^{-1} h^{(k)}}$$

which is the same as the formula for $B^{(k+1)}$ used in the algorithm.
To see why Broyden’s method can be interpreted as a variant of the secant method, we multiply the formula used to update $B^{(k)}$ in the algorithm by $z^{(k)}$, i.e.,

$$B^{(k+1)}z^{(k)} = B^{(k)}z^{(k)} - \left( B^{(k)}z^{(k)} - h^{(k)} \right) \left[ h^{(k)} \right]^T B^{(k)} \left[ h^{(k)} \right]^T z^{(k)}$$

$$\iff B^{(k+1)}z^{(k)} = h^{(k)}$$

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

which is reminiscent of the secant equation

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

since $B^{(k+1)}$ is an approximation to the inverse of the Jacobian.

We illustrate this algorithm in the Matlab script file `run_broyden.m` with the same example as used earlier for the multivariate Newton method.

**Remarks:**

1. Broyden’s method can also be improved by a line search, i.e., $x^{(k+1)} = x^{(k)} + \lambda_k h^{(k)}$ (see below).
2. Broyden’s method converges only superlinearly.
3. Both Newton’s and Broyden’s methods require good starting values. These can be provided by the steepest descent or conjugate gradient algorithms.

We now discuss the connection between solving systems of nonlinear equations and quadratic minimization problems. The idea is to minimize the 2-norm of the residual of (38) to get the stepsize $\lambda_k$, i.e., to find $[x_1, \ldots, x_n]^T$ such that

$$g(x_1, \ldots, x_n) = \sum_{i=1}^n f_i^2(x_1, \ldots, x_n) = f^T f(x)$$

is minimized.

For the steepest descent method we use

$$x^{(k+1)} = x^{(k)} + \lambda_k (-\nabla g(x^{(k)})).$$

The stepsize $\lambda_k$ is computed such that

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

is minimized. This is an easier problem to solve since it involves only one variable, $\lambda_k$.

Note that since $g(x) = f^T f(x)$ we have $\nabla g(x) = 2 [J(x)]^T f(x)$. This shows that this approach also requires knowledge of the Jacobian. A general line search algorithm is
Algorithm

Input $f, J, x^{(0)}, M$

for $k = 0$ to $M$ do

$h^{(k)} = -\nabla g(x^{(k)}) = -2 \left[ J(x^{(k)}) \right]^T f(x^{(k)})$

Find $\lambda_k$ as a minimizer of $g(x^{(k)} + \lambda_k h^{(k)}) = f^T f(x^{(k)} + \lambda_k h^{(k)})$

Update $x^{(k+1)} = x^{(k)} + \lambda_k h^{(k)}$

end

Output $x^{(k+1)}$

Remarks:

1. The steepest descent method converges linearly.

2. One can replace the steepest descent method by conjugate gradient iteration. This is illustrated in the Matlab script `run_mincg.m`.

3. If only minimization of the quadratic function $g$ is our goal, then we can try solving $\nabla g(x) = 0$ using Newton’s method (which will give us a critical point for the problem).