17 Solution of Nonlinear Systems

We now discuss the solution of systems of *nonlinear* equations. An important ingredient will be the multivariate Taylor theorem.

Theorem 17.1 *Let* $D = \{[x_1, x_2, \ldots, x_m]^T \in \mathbb{R}^m : a_i \leq x_i \leq b_i,\ i = 1, \ldots, m\}$ *for some* $a_1, a_2, \ldots, a_m, b_1, b_2, \ldots, b_m \in \mathbb{R}$. *If* $f \in C^{n+1}(D)$, *then for* $\boldsymbol{x} + \boldsymbol{h} \in D$ ($\boldsymbol{h} = [h_1, h_2, \ldots, h_m]^T$)

$$f(\boldsymbol{x} + \boldsymbol{h}) = \sum_{k=0}^{n} \frac{1}{k!} (\boldsymbol{h}^T \nabla)^k f(\boldsymbol{x}) + R_n(\boldsymbol{h}), \tag{45}$$

where

$$R_n(\boldsymbol{h}) = \frac{1}{(n+1)!} (\boldsymbol{h}^T \nabla)^{n+1} f(\boldsymbol{x} + \theta \boldsymbol{h})$$

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

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

$$f(\boldsymbol{x} + \boldsymbol{h}) = f(\boldsymbol{x}) + (\boldsymbol{h}^T \nabla) f(\boldsymbol{x}) + \frac{1}{2} \underbrace{(\boldsymbol{h}^T \nabla)^2 f(\boldsymbol{x} + \theta \boldsymbol{h})}_{= R_1(\boldsymbol{h})}.$$

And, for $m = 2$, this becomes

$$\begin{aligned} f(x_1 + h_1, x_2 + h_2) &= f(x_1, x_2) + \left(h_1 \frac{\partial}{\partial x_1} + h_2 \frac{\partial}{\partial x_2} \right) f(x_1, x_2) \\ &\quad + \frac{1}{2} \left(h_1 \frac{\partial}{\partial x_1} + h_2 \frac{\partial}{\partial x_2} \right)^2 f(x_1 + \theta h_1, x_2 + \theta h_2) \\ &= f(x_1, x_2) + h_1 \frac{\partial f}{\partial x_1}(x_1, x_2) + h_2 \frac{\partial f}{\partial x_2}(x_1, x_2) \\ &\quad + \frac{1}{2} \left(h_1^2 \frac{\partial^2}{\partial x_1^2} + 2 h_1 h_2 \frac{\partial^2}{\partial x_1 \partial x_2} + h_2^2 \frac{\partial^2}{\partial x_2^2} \right) f(x_1 + \theta h_1, x_2 + \theta h_2). \end{aligned}$$

Therefore, the *linearization* of f is given by

$$f(x_1 + h_1, x_2 + h_2) = f(x_1, x_2) + \left(h_1 \frac{\partial}{\partial x_1} + h_2 \frac{\partial}{\partial x_2} \right) f(x_1, x_2).$$

or

$$f(\boldsymbol{x} + \boldsymbol{h}) = f(\boldsymbol{x}) + (\boldsymbol{h}^T \nabla) f(\boldsymbol{x}).$$

We now want to solve the following (square) *system of nonlinear equations*:

$$\begin{aligned} f_1(x_1, x_2, \ldots, x_m) &= 0, \\ f_2(x_1, x_2, \ldots, x_m) &= 0, \\ &\vdots \\ f_m(x_1, x_2, \ldots, x_m) &= 0. \end{aligned} \tag{46}$$

To derive Newton's method for this problem we assume $\boldsymbol{z} = [z_1, z_2, \ldots, z_m]^T$ is a solution (or *root*) of (46), i.e., $\boldsymbol{z}$ satisfies

$$f_i(\boldsymbol{z}) = 0, \qquad i = 1, \ldots, m.$$

Moreover, we consider $\boldsymbol{x}$ to be an approximate root, i.e.,

$$\boldsymbol{x} + \boldsymbol{h} = \boldsymbol{z},$$

with a small correction $\boldsymbol{h} = [h_1, h_2, \ldots, h_m]^T$. Then, by linearizing f_i, $i = 1, \ldots, m$,

$$f_i(\boldsymbol{z}) = f_i(\boldsymbol{x} + \boldsymbol{h}) \approx f_i(\boldsymbol{x}) + (\boldsymbol{h}^T \nabla) f_i(\boldsymbol{x}).$$

Since $f_i(\boldsymbol{z}) = 0$ we get

$$\begin{aligned} -f_i(\boldsymbol{x}) &\approx (\boldsymbol{h}^T \nabla) f_i(\boldsymbol{x}) \\ &= \left(h_1 \frac{\partial}{\partial x_1} + h_2 \frac{\partial}{\partial x_2} + \ldots + h_m \frac{\partial}{\partial x_m} \right) f_i(\boldsymbol{x}). \end{aligned}$$

Therefore, we have a linearized version of system (46) as

$$\begin{aligned} -f_1(x_1, \ldots, x_m) &= \left(h_1 \frac{\partial}{\partial x_1} + \ldots + h_m \frac{\partial}{\partial x_m} \right) f_1(x_1, \ldots, x_m), \\ -f_2(x_1, \ldots, x_m) &= \left(h_1 \frac{\partial}{\partial x_1} + \ldots + h_m \frac{\partial}{\partial x_m} \right) f_2(x_1, \ldots, x_m), \\ &\vdots \\ -f_m(x_1, \ldots, x_m) &= \left(h_1 \frac{\partial}{\partial x_1} + \ldots + h_m \frac{\partial}{\partial x_m} \right) f_m(x_1, \ldots, x_m). \end{aligned} \tag{47}$$

Recall that $\boldsymbol{h} = [h_1, \ldots, h_m]^T$ is the unknown Newton update, and note that (47) is a linear system for $\boldsymbol{h}$ of the form

$$J(\boldsymbol{x})\boldsymbol{h} = -\boldsymbol{f}(\boldsymbol{x}),$$

where $\boldsymbol{f} = [f_1, \ldots, f_m]^T$ and

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

is called the *Jacobian* of $\boldsymbol{f}$.

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

Algorithm

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

for $k = 0, 1, 2, \ldots$ do

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

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

end

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

Remark If we symbolically write $\boldsymbol{f}'$ instead of J, then the Newton iteration becomes

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} - \underbrace{\left[\boldsymbol{f}'(\boldsymbol{x}^{(k)})\right]^{-1}}_{\text{matrix}} \boldsymbol{f}(\boldsymbol{x}^{(k)}),$$

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

Example Solve

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

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

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

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`.

Remark

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

2. In each iteration an $m \times m$ (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 $J^{-1}(\boldsymbol{z})$ exists.

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

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

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

17.1 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* is

Theorem 17.2 *Let C be a closed subset of $\mathbb{R}^m$ and F a contractive mapping of C into itself. Then F has a unique fixed point $\boldsymbol{z}$. Moreover, $\boldsymbol{z} = \lim_{k\to\infty} \boldsymbol{x}^{(k)}$, where $\boldsymbol{x}^{(k+1)} = F(\boldsymbol{x}^{(k)})$ and $\boldsymbol{x}^{(0)}$ is any starting point in C.*

Here a contractive map is defined as

Definition 17.3 *A function (mapping) F is called* contractive *if there exists a $\lambda < 1$ such that*

$$\|F(\boldsymbol{x}) - F(\boldsymbol{y})\| \leq \lambda \|\boldsymbol{x} - \boldsymbol{y}\| \tag{48}$$

for all $\boldsymbol{x}, \boldsymbol{y}$ in the domain of F.

The property (48) is also referred to as *Lipschitz continuity of F*.

17.2 Quasi-Newton Methods

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

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

The idea is to provide an initial approximation $B^{(0)}$ to $\left[J(\boldsymbol{x}^{(0)})\right]^{-1}$, 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.

This replaces solution of the linear system $J(\boldsymbol{x}^{(k)})\boldsymbol{h} = -\boldsymbol{f}(\boldsymbol{x}^{(k)})$.

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

Lemma 17.4 *Let A be a nonsingular $m \times m$ matrix and $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^m$. Then $(A + \boldsymbol{x}\boldsymbol{y}^T)^{-1}$ exists provided that $\boldsymbol{y}^T A^{-1} \boldsymbol{x} \neq -1$. Moreover,*

$$(A + \boldsymbol{x}\boldsymbol{y}^T)^{-1} = A^{-1} - \frac{A^{-1}\boldsymbol{x}\boldsymbol{y}^T A^{-1}}{1 + \boldsymbol{y}^T A^{-1}\boldsymbol{x}}. \tag{49}$$

The Sherman-Morrison formula (49) can be used to compute the inverse of a matrix $A^{(k+1)}$ obtained by a rank-1 update $\boldsymbol{x}\boldsymbol{y}^T$ from $A^{(k)}$, i.e.,

$$\left[A^{(k+1)}\right]^{-1} = \left[A^{(k)}\right]^{-1} - \frac{\left[A^{(k)}\right]^{-1}\boldsymbol{x}\boldsymbol{y}^T\left[A^{(k)}\right]^{-1}}{1+\boldsymbol{y}^T\left[A^{(k)}\right]^{-1}\boldsymbol{x}}. \tag{50}$$

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 (50).

The algorithm for Broyden's method is

Algorithm

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

for $k = 0, 1, 2, \ldots$ do

$$\begin{aligned}
\boldsymbol{h}^{(k)} &= -B^{(k)}\boldsymbol{f}(\boldsymbol{x}^{(k)}) \\
\boldsymbol{x}^{(k+1)} &= \boldsymbol{x}^{(k)} + \boldsymbol{h}^{(k)} \\
\boldsymbol{z}^{(k)} &= \boldsymbol{f}(\boldsymbol{x}^{(k+1)}) - \boldsymbol{f}(\boldsymbol{x}^{(k)}) \\
B^{(k+1)} &= B^{(k)} - \frac{\left(B^{(k)}\boldsymbol{z}^{(k)} - \boldsymbol{h}^{(k)}\right)\left[\boldsymbol{h}^{(k)}\right]^T B^{(k)}}{\left[\boldsymbol{h}^{(k)}\right]^T B^{(k)}\boldsymbol{z}^{(k)}}
\end{aligned}$$

end

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

Remark 1. Only m scalar function evaluations are required per iteration along with $\mathcal{O}(m^2)$ 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 (50) we define

$$\begin{aligned}
\left[J(\boldsymbol{x}^{(k)})\right]^{-1} \approx \left[A^{(k)}\right]^{-1} &= B^{(k)}, \\
\boldsymbol{x} &= \frac{\boldsymbol{z}^{(k)} - \left[B^{(k)}\right]^{-1}\boldsymbol{h}^{(k)}}{\|\boldsymbol{h}^{(k)}\|_2^2}, \\
\boldsymbol{y} &= \boldsymbol{h}^{(k)}.
\end{aligned}$$

Then (50) becomes

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

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 $\boldsymbol{z}^{(k)}$, i.e.,

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

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

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

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.

Remark 1. Broyden's method can also be improved by a line search, i.e., $\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \lambda_k \boldsymbol{h}^{(k)}$ (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.

17.3 Using the Steepest Descent and Conjugate Gradients with Nonlinear Systems

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 (46) to get the stepsize λ_k, i.e., to find $\boldsymbol{x} = [x_1, \ldots, x_m]^T$ such that

$$g(x_1, \ldots, x_m) = \frac{1}{2}\sum_{i=1}^{m} f_i^2(x_1, \ldots, x_m) = \frac{1}{2}\boldsymbol{f}^T\boldsymbol{f}(\boldsymbol{x})$$

is minimized.

For the steepest descent method we use

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

The stepsize λ_k is computed such that

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

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

Note that since $g(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{f}^T\boldsymbol{f}(\boldsymbol{x})$ we have $\nabla g(\boldsymbol{x}) = [J(\boldsymbol{x})]^T\,\boldsymbol{f}(\boldsymbol{x})$. This shows that this approach also requires knowledge of the Jacobian. A general line search algorithm is

Algorithm

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

for $k = 0, 1, 2, \ldots$ do

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

Find λ_k as a minimizer of $\gamma(\lambda_k) = g(\boldsymbol{x}^{(k)} + \lambda_k \boldsymbol{h}^{(k)}) = \frac{1}{2}\boldsymbol{f}^T \boldsymbol{f}(\boldsymbol{x}^{(k)} + \lambda_k \boldsymbol{h}^{(k)})$

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

end

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

Remark 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 $\nabla g(\boldsymbol{x}) = 0$ using Newton's method (which will give us a critical point for the problem). This is explained in the next section.

It is not easy to come up with a good line search strategy. However, one practical way to determine a reasonable value for λ_k is to use a quadratic interpolating polynomial. The idea is to work with three values $\lambda_k^{(1)}$, $\lambda_k^{(2)}$ and $\lambda_k^{(3)}$ and construct a quadratic polynomial that interpolates the univariate function γ at these points. If we guess these three values reasonably close to the optimum, then the minimum of the parabola on the interval of interest (which is easy to find) tell us how to pick λ_k. For example, one can take $\lambda_k^{(1)} = 0$, then find a $\lambda_k^{(3)}$ such that $\gamma(\lambda_k^{(3)}) < \gamma(\lambda_k^{(1)})$, and then take $\lambda_k^{(2)}$ as the midpoint of the previous values, i.e., $\lambda_k^{(2)} = \lambda_k^{(3)}/2$.

A similar (but interactive) strategy is implemented in the Matlab example `ShowGN_VL.m` explained in the next section.

17.4 Nonlinear Least Squares

To combine the techniques used for systems of linear equations with those for nonlinear systems we consider the problem of finding values of $\boldsymbol{x} = [x_1, \ldots, x_n]^T$ that yield a least squares solution of the (rectangular) nonlinear system

$$\begin{aligned} f_1(x_1, x_2, \ldots, x_n) &= 0, \\ f_2(x_1, x_2, \ldots, x_n) &= 0, \\ &\vdots \\ f_m(x_1, x_2, \ldots, x_n) &= 0. \end{aligned} \tag{51}$$

In other words, we want to minimize

$$g(\boldsymbol{x}) = \frac{1}{2}\sum_{i=1}^{m} f_i^2(\boldsymbol{x}). \tag{52}$$

A problem like this often arises when the components of $\boldsymbol{x}$ are certain control parameters in an objective function that needs to be optimized. Below we will see an example where we need to fit the parameters to a nonlinear system describing the orbit of a planet.

As mentioned at the end of the previous section, we can use Newton's method to solve this problem since a necessary condition for finding a minimal residual (52) is that the gradient of g be zero. Thus, we are attempting to find a solution of $\nabla g(\boldsymbol{x}) = \mathbf{0}$. Since Newton's method is given by

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} - \left[J_{\nabla g}(\boldsymbol{x}^{(k)})\right]^{-1} \nabla g(\boldsymbol{x}^{(k)}),$$

we see that we actually need not only the gradient of S (which leads to the Jacobian) but also the second derivative (which leads to the *Hessian*).

For the particular minimization problem (52) this would result in

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} - \underbrace{\left[J(\boldsymbol{x}^{(k)})^T J(\boldsymbol{x}^{(k)}) + \sum_{i=1}^{m} f_i(\boldsymbol{x}^{(k)}) \nabla^2 f_i(\boldsymbol{x}^{(k)})\right]^{-1}}_{=\text{Hessian}} J(\boldsymbol{x}^{(k)})^T \boldsymbol{f}(\boldsymbol{x}^{(k)}),$$

where now J is the regular (but rectangular) Jacobian with respect to the functions $f_1, \ldots, f_m$.

Since we decided earlier that it would be a good idea to avoid computation of the Jacobian, clearly, computation of the Hessian should be avoided if possible. This motivates the *Gauss-Newton method*. If we drop the summation term in the expression for the Hessian above, then all we need to know is the Jacobian. This step is justified if we are reasonably close to a solution since then $f_i(\boldsymbol{x}^{(k)})$ will be close to zero.

Thus, the Gauss-Newton algorithm is given by

Algorithm

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

for $k = 0, 1, 2, \ldots$ do

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

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

end

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

Note that the update $\boldsymbol{h}^{(k)}$ here is in fact nothing but the solution to the normal equations for the (linear) least squares problem

$$\min_{\boldsymbol{h} \in \mathbb{R}^n} \|J(\boldsymbol{x})\boldsymbol{h} + \boldsymbol{f}(\boldsymbol{x})\|_2.$$

Therefore, any of our earlier algorithms (modified Gram-Schmidt, QR, SVD) can be used to perform this step.

Example Let's consider the nonlinear system

$$\begin{aligned} x(t) &= \frac{P-A}{2} + \frac{A+P}{2}\cos(t) \\ y(t) &= \sqrt{AP}\sin(t) \end{aligned}$$

that describes the elliptical orbit of a planet around the sun in a two-dimensional universe. Here A and P denote the maximum and minimum orbit-to-sun distances, respectively.

In order to estimate the parameters of the orbit of a specific planet we use another relationship between the parameters and the length of the orbit, $r(\theta)$, where θ is the angle to the positive x-axis. This relationship is given by

$$r(\theta) = \frac{2AP}{P(1-\cos(\theta)) + A(1+\cos(\theta))}.$$

Let's assume we have measurements (θ_i, r_i) so that we can define functions

$$f_i(A,P) = r_i - \frac{2AP}{P(1-\cos(\theta_i)) + A(1+\cos(\theta_i))}, \qquad i = 1, \ldots, m.$$

Then the best choices of A and P will be given by minimizing the function

$$g(A,P) = \frac{1}{2}\sum_{i=1}^{m} f_i(A,P)^2.$$

This corresponds exactly to our general discussion above.

The Matlab code `ShowGN_VL.m` written by Charles Van Loan for his book "Introduction to Scientific Computing" provides an implementation of this example. The "measurements" are created by starting with two exact values of A and P, $A_0 = 5$ and $P_0 = 1$, and then sampling from an orbit based on these parameters that has been perturbed by noise. A line search is also included for which the step length λ_k is entered interactively.