Newton-Raphson method (multivariate)

Before discussing how to solve a multivariate systems, it is helpful to review the Taylor series expansion of an N-D function.

The methods discussed above for solving a 1-D equation $f(x) = 0$ can be generalized for solving an N-D multivariate equation system:

$$\begin{cases} f_1(x_1, \cdots, x_N) = f_1(\mathbf{x}) = 0 \\ f_2(x_1, \cdots, x_N) = f_2(\mathbf{x}) = 0 \\ \cdots\cdots\cdots\cdots \\ f_N(x_1, \cdots, x_N) = f_N(\mathbf{x}) = 0 \end{cases}$$

where we have defined

$$\mathbf{x} = [x_1, \cdots, x_N]^T$$

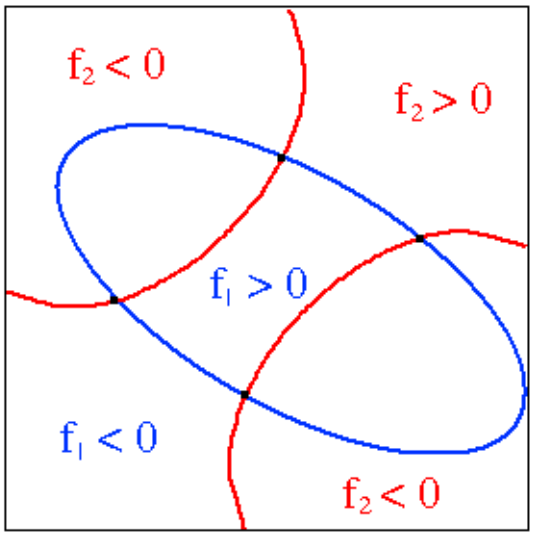
We can further define a vector function

$$\mathbf{f}(\mathbf{x}) = [f_1(\mathbf{x}), \cdots, f_n(\mathbf{x})]^T$$

so that the equation system above can be written as

$$\mathbf{f}(\mathbf{x}) = \mathbf{0}$$

When $N = 2$, the solution of the equation above can be geometrically explained. The equation $f_i(x_1, x_2) = 0$ represents contour curves in the (x_1, x_2) plane that partition the plane into regions in which the function $f_1(x_1, x_2)$ takes either positive or negative values. The solutions that satisfy both equations are the intersections of the contour curves of both $f_1(x_1, x_2)$ and $f_2(x_1, x_2)$.



Newton-Raphson method

The Newton-Raphson method can also be generalized for solving an N-D system $\mathbf{f}(\mathbf{x}) = \mathbf{0}$. We first consider the Taylor expansions of the N functions:

$$f_i(\mathbf{x}+\delta\mathbf{x}) = f_i(\mathbf{x}) + \sum_{j=1}^{N} \frac{\partial f_i(\mathbf{x})}{\partial x_j}\delta x_j + O(\delta\mathbf{x}^2) \approx f_i(\mathbf{x}) + \sum_{j=1}^{N} \frac{\partial f_i(\mathbf{x})}{\partial x_j}\delta x_j, \qquad (i = 1, \cdots, N)$$

These N equations can be written in vector form:

$$\mathbf{f}(\mathbf{x}+\delta\mathbf{x}) = \begin{bmatrix} f_1(\mathbf{x}+\delta\mathbf{x}) \\ \vdots \\ f_N(\mathbf{x}+\delta\mathbf{x}) \end{bmatrix} \approx \begin{bmatrix} f_1(\mathbf{x}) \\ \vdots \\ f_N(\mathbf{x}) \end{bmatrix} + \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_N} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_N}{\partial x_1} & \cdots & \frac{\partial f_N}{\partial x_N} \end{bmatrix} \begin{bmatrix} \delta x_1 \\ \vdots \\ \delta x_N \end{bmatrix}$$

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

where $\mathbf{J}_\mathbf{f}(\mathbf{x})$ is an $N \times N$ *Jacobian matrix* defined over the function vector $\mathbf{f}(\mathbf{x})$ defined as:

$$\mathbf{J}_\mathbf{f}(\mathbf{x}) = \frac{d}{d\mathbf{x}}\mathbf{f}(\mathbf{x}) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_N} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_N}{\partial x_1} & \cdots & \frac{\partial f_N}{\partial x_N} \end{bmatrix}$$

with the ijth element being $\partial f_i/\partial x_j$. Note that the ith row vector of $\mathbf{J}(\mathbf{x})$ is the gradient vector of the ith function $f_i(\mathbf{x})$:

$$\mathbf{g}[f_i(\mathbf{x})] = \left[\frac{\partial f_i(\mathbf{x})}{\partial x_1}, \cdots, \frac{\partial f_i(\mathbf{x})}{\partial x_N}\right]^T$$

Like in the 1-D case, if all equations are linear, the error terms $O_i(\delta\mathbf{x}^2)$ are zero and we have

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

By assuming $\mathbf{f}(\mathbf{x} + \delta\mathbf{x}) = 0$, we can find the roots as $\mathbf{x} + \delta\mathbf{x}$, where $\delta\mathbf{x}$ can be obtained by solving the equation above:

$$\delta\mathbf{x} = \mathbf{J}(\mathbf{x})^{-1}[\mathbf{f}(\mathbf{x} + \delta\mathbf{x}) - \mathbf{f}(\mathbf{x})] = -\mathbf{J}(\mathbf{x})^{-1}\mathbf{f}(\mathbf{x})$$

and the root can be found from any starting point $\mathbf{x}$ as

$$\mathbf{x} + \delta\mathbf{x} = \mathbf{x} - \mathbf{J}(\mathbf{x})^{-1}\mathbf{f}(\mathbf{x})$$

If the equations are nonlinear, this result is only an approximation of the real root, which can be improved iteratively:

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \delta\mathbf{x}_n = \mathbf{x}_n - \mathbf{J}(\mathbf{x}_n)^{-1}\,\mathbf{f}(\mathbf{x}_n)$$

The iteration moves $\mathbf{x}_n$ in the N-D space from an initial position $\mathbf{x}_0 = [x_1, \cdots, x_N]^T$ in such a direction that all functions $f_i(\mathbf{x})$ ($i = 1, \cdots, N$) are reduced.

Newton's method can be further generated to solve over-constrained non-linear equation systems with N unknowns but $M > N$ equations. In the case the inverse matrix of the $M \times N$ Jacobian matrix $\mathbf{J}$ does not exist, but the $N \times M$ pseudo-

inverse $\mathbf{J}^- = (\mathbf{J}^T\mathbf{J})^{-1}\mathbf{J}^T$ can be used in the iteration:

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \mathbf{J}_n^- \, \mathbf{f}(\mathbf{x}_n)$$

or

$$\begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix}_{n+1} = \begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix}_{n} - \begin{bmatrix} \\ \mathbf{J}^-(\mathbf{x}_n) \\ \\ \end{bmatrix} \begin{bmatrix} f_1(\mathbf{x}_n) \\ \vdots \\ \vdots \\ \vdots \\ f_M(\mathbf{x}_n) \end{bmatrix}$$

The iteration attempts to find a solution in the nonlinear least squares sense. This is essentially the Gauss-Newton algorithm to be considered later.

The Newton-Raphson method assumes the analytical expressions of all partial derivatives $\partial f_i(\mathbf{x})/\partial x_j$ can be made available based on the functions $\mathbf{f}(\mathbf{x})$, so that the Jacobian matrix $\mathbf{J}$ can be computed. However, when this is not the case, it is still possible to estimate the ijth component of $J_{ij} = \partial f_i/\partial x_j$ in $\mathbf{J}$ given $\mathbf{x}_n$ and $\mathbf{x}_{n+1}$ in two consecutive iterations:

$$J_{ij} = \frac{\partial f_i(x_1, \cdots, x_N)}{\partial x_j} \approx \frac{f_i(x_1^{(n)}, \cdots, x_j^{(n+1)}, \cdots, x_N^{(n)}) - f_i(x_1^{(n)}, \cdots, x_N^{(n)})}{x_j^{(n+1)} - x_j^{(n)}}$$

This can be considered as the secant method generalized from 1-D to N-D, based on two initial guesses.

Example 1

$$\begin{cases} 3\,x_1 - \cos(x_2 x_3) - 3/2 = 0 \\ 4x_1^2 - 625\,x_2^2 + 2x_3 - 1 = 0 \\ 20\,x_3 + e^{-x_1 x_2} + 9 = 0 \end{cases}$$

$$\mathbf{J} = \begin{bmatrix} 3 & x_3 \sin(x_2 x_3) & x_2 \sin(x_2 x_3) \\ 8\,x_1 & -1250\,x_2 & 2 \\ -x_2\, e^{-x_1 x_2} & -x_1\, e^{-x_1 x_2} & 20 \end{bmatrix}$$

n	$\mathbf{x}$	$error$
0	$(1.000000, 1.000000, 1.000000)$	$6.207e+02$
1	$(1.232701, 0.503132, -0.473253)$	$1.541e+02$
2	$(0.832592, 0.251806, -0.490636)$	$3.884e+01$
3	$(0.833238, 0.128406, -0.494702)$	$9.517e+00$
4	$(0.833275, 0.069082, -0.497147)$	$2.200e+00$
5	$(0.833281, 0.043585, -0.498206)$	$4.063e-01$
6	$(0.833282, 0.036117, -0.498517)$	$3.486e-02$
7	$(0.833282, 0.035343, -0.498549)$	$3.741e-04$
8	$(0.833282, 0.035335, -0.498549)$	$4.498e-08$
9	$(0.833282, 0.035335, -0.498549)$	$5.551e-16$

Example 2

$$\mathbf{f}(\mathbf{x}) = \mathbf{0}, \qquad \begin{cases} f_1(\mathbf{x}) = x_1^2 - 2x_1 + x_2^2 - x_3 + 1 = 0 \\ f_2(\mathbf{x}) = x_1 x_2^2 - x_1 - 3x_2 + x_2 x_3 + 2 = 0 \\ f_3(\mathbf{x}) = x_1 x_3^2 - 3x_3 + x_2 x_3^2 + x_1 x_2 = 0 \end{cases}$$

$$\mathbf{J} = \begin{bmatrix} 2x_1 - 2 & 2x_2 & -1 \\ x_2^2 - 1 & 2x_1 x_2 - 3 + x_3 & x_2 \\ x_3^2 + x_2 & x_3^2 + x_1 & 2x_1 x_3 - 3 + 2x_2 x_3 \end{bmatrix}$$

With $\mathbf{x}_0 = [1,\ 2,\ 3]^T$, we get a root:

n	$\mathbf{x}$	$error$
0	$(1.000000, 2.000000, 3.000000)$	$e = 2.064e+01$
1	$(0.102564, 1.641026, 2.564103)$	$e = 4.303e+00$
2	$(1.520623, 1.411129, 0.198590)$	$e = 2.689e+00$
3	$(1.941234, 0.771343, 0.894651)$	$e = 1.217e+00$
4	$(1.067366, 1.191171, 0.483525)$	$e = 1.144e+00$
5	$(1.268254, 0.951822, 0.880281)$	$e = 3.323e-01$
6	$(0.958988, 1.033836, 0.968126)$	$e = 1.171e-01$
7	$(1.001713, 1.000070, 0.997178)$	$e = 4.162e-03$
8	$(1.000001, 1.000001, 1.000000)$	$e = 6.701e-06$
9	$(1.000000, 1.000000, 1.000000)$	$e = 8.481e-12$

With $\mathbf{x}_0 = [0,\ 0,\ 0]^T$, we get another root:

n	$\mathbf{x}$	$error$
0	$(0.000000, 0.000000, 0.000000)$	$e = 2.236e+00$
1	$(0.500000, 0.500000, 0.000000)$	$e = 5.728e-01$
2	$(0.839506, 0.475309, 0.135802)$	$e = 1.175e-01$
3	$(0.985821, 0.418485, 0.150694)$	$e = 2.639e-02$
4	$(1.054172, 0.387153, 0.147169)$	$e = 6.088e-03$
5	$(1.085652, 0.373392, 0.145578)$	$e = 1.264e-03$
6	$(1.096933, 0.368489, 0.145029)$	$e = 1.618e-04$
7	$(1.098881, 0.367643, 0.144935)$	$e = 4.817e-06$
8	$(1.098943, 0.367617, 0.144932)$	$e = 4.837e-09$

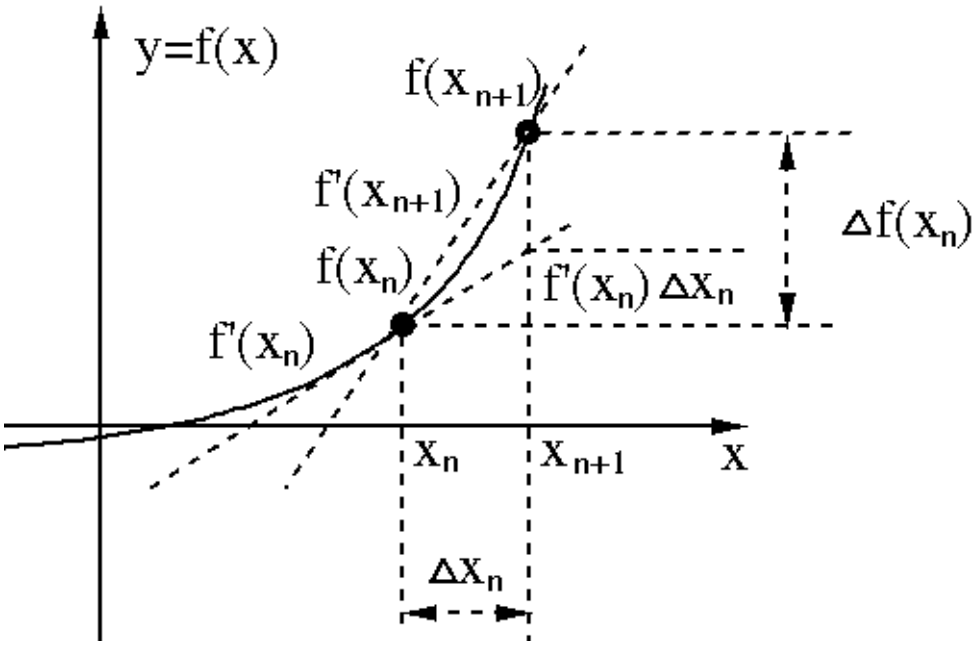
Broyden's method

Recall that the secant method approximate the derivative $f'(x_{n+1})$ based on $\delta x_n = x_{n+1} - x_n$ and $f(x_n) = f(x_{n+1}) - f(x_n)$:

$$\hat{f}'(x_{n+1}) = \frac{f(x_{n+1}) - f(x_n)}{x_{n+1} - x_n} = \frac{\delta f(x_n)}{\delta x_n} = \hat{f}'(x_n) + \frac{\delta f(x_n) - \hat{f}'(x_n)\delta x_n}{\delta x_n}$$

Here $\hat{f}'(x_n)\delta x_n$ is the estimated value of $\delta f(x_n)$ based on the old derivate $\hat{f}'(x_n)$,

which is different from the more accurate estimate $\hat{f}'(x_{n+1})\delta x_n$ based on $\hat{f}'(x_{n+1})$. The difference between these two estimates, when divided by δx_n, approximates the difference between $\hat{f}'(x_{n+1})$ and $\hat{f}'(x_n)$, i.e., the update formula of $\hat{f}'(x_{n+1})$ given above.



The updated $\hat{f}'(x_{n+1})$ can then be used for updating x_{n+1}:

$$x_{n+1} = x_n - \frac{f(x_n)}{\hat{f}'(x_n)}$$

Broyden's method, one of the quasi-Newton methods, can be considered as a generalization of this secant method for solving an N-D system $\mathbf{f}(\mathbf{x}) = \mathbf{0}$. Instead of assuming the availability of the true Jacobian matrix $\hat{\mathbf{J}}$, here we estimate the next Jacobian $\hat{\mathbf{J}}_{n+1}$ by an iteration based on the current one. Let

$$\delta\mathbf{x}_n = \mathbf{x}_{n+1} - \mathbf{x}_n, \qquad \delta\mathbf{f}_n = \mathbf{f}_{n+1} - \mathbf{f}_n$$

then

$$\hat{\mathbf{J}}_{n+1} = \hat{\mathbf{J}}_n + \frac{(\delta \mathbf{f}_n - \hat{\mathbf{J}}_n \delta \mathbf{x}_n)\delta \mathbf{x}_n^T}{||\delta \mathbf{x}_n||^2}$$

With $\mathbf{J}_n$ available, we have the iteration for the solution update as before:

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \hat{\mathbf{J}}_n^{-1}\mathbf{f}(\mathbf{x}_n)$$

Here the inverse of $\mathbf{J}_n$ needs to be computed in each iteration. A better method is to estimate $\mathbf{J}_n^{-1}$ directly. Specifically we apply the Sherman-Morrison formula

$$(\mathbf{A} + \mathbf{u}\mathbf{v}^T)^{-1} = \mathbf{A}^{-1} - \frac{\mathbf{A}^{-1}\mathbf{u}\mathbf{v}^T\mathbf{A}^{-1}}{1 + \mathbf{v}^T\mathbf{A}^{-1}\mathbf{u}}$$

to the iteration above for $\mathbf{J}_n$, with

$$\mathbf{u} = \frac{\delta \mathbf{f}_n - \mathbf{J}_n \delta \mathbf{x}_n}{||\delta \mathbf{x}_n||^2}, \qquad \mathbf{v} = \delta \mathbf{x}_n, \qquad \mathbf{A} = \mathbf{J}_n$$

and get

$$\hat{\mathbf{J}}_{n+1}^{-1} = \hat{\mathbf{J}}_n^{-1} - \frac{\hat{\mathbf{J}}_n^{-1}\left(\delta \mathbf{f}_n - \hat{\mathbf{J}}_n \delta \mathbf{x}_n\right)/||\delta \mathbf{x}_n||^2 \, \delta \mathbf{x}_n^T \hat{\mathbf{J}}_n^{-1}}{1 + \delta \mathbf{x}_n^T \hat{\mathbf{J}}_n^{-1}(\delta \mathbf{f}_n - \hat{\mathbf{J}}_n \delta \mathbf{x}_n)/||\delta \mathbf{x}_n||^2} = \hat{\mathbf{J}}_n^{-1} + \frac{\left(\delta \mathbf{x}_n - \hat{\mathbf{J}}_n^{-1}\delta \mathbf{f}_n\right)\delta \mathbf{x}_n^T \hat{\mathbf{J}}_n^{-1}}{\delta \mathbf{x}_n^T \hat{\mathbf{J}}_n^{-1}\delta \mathbf{f}_n}$$

---

nex up previou

Next: Optimization **Up:** Solving Non-Linear Equations **Previous:** Newton-Raphson method (univariate)

Ruye Wang 2015-02-12