

Multidimensional Rootfinding

Newton's Method for Nonlinear Systems

Multidimensional Rootfinding Problem

Rootfinding Problem: Vector Version

Given a continuous vector-valued function $\mathbf{f} : \mathbb{R}^n \rightarrow \mathbb{R}^n$, find a vector $\mathbf{r} \in \mathbb{R}^n$ such that $\mathbf{f}(\mathbf{r}) = \mathbf{0}$.

The rootfinding problem $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ is equivalent to solving the *nonlinear* system of n scalar equations in n unknowns:

$$\begin{aligned}f_1(x_1, \dots, x_n) &= 0, \\f_2(x_1, \dots, x_n) &= 0, \\&\vdots \\f_n(x_1, \dots, x_n) &= 0.\end{aligned}$$

Multidimensional Taylor Series

If \mathbf{f} is differentiable, we can write

$$\mathbf{f}(\mathbf{x} + \mathbf{h}) = \mathbf{f}(\mathbf{x}) + \mathbf{J}(\mathbf{x})\mathbf{h} + O(\|\mathbf{h}\|^2),$$

where \mathbf{J} is the **Jacobian matrix** of \mathbf{f}

$$\mathbf{J}(\mathbf{x}) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix} = \left[\frac{\partial f_i}{\partial x_j} \right]_{i,j=1,\dots,n}.$$

- The first two terms $\mathbf{f}(\mathbf{x}) + \mathbf{J}(\mathbf{x})\mathbf{h}$ is the “linear approximation” of \mathbf{f} near \mathbf{x} .
- If \mathbf{f} is actually linear, *i.e.*, $\mathbf{f}(\mathbf{x}) = A\mathbf{x} - \mathbf{b}$, then the Jacobian matrix is the coefficient matrix A and the rootfinding problem $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ is simply $A\mathbf{x} = \mathbf{b}$.

Example

Let

$$f_1(x_1, x_2, x_3) = -x_1 \cos(x_2) - 1,$$

$$f_2(x_1, x_2, x_3) = x_1 x_2 + x_3,$$

$$f_3(x_1, x_2, x_3) = e^{-x_3} \sin(x_1 + x_2) + x_1^2 - x_2^2.$$

Then

$$\mathbf{J}(\mathbf{x}) = \begin{bmatrix} -\cos(x_2) & x_1 \sin(x_2) & 0 \\ x_2 & x_1 & 1 \\ e^{-x_3} \cos(x_1 + x_2) + 2x_1 & e^{-x_3} \cos(x_1 + x_2) - 2x_2 & -e^{-x_3} \sin(x_1 + x_2) \end{bmatrix}.$$

Exercise. Write out the linear part of the Taylor expansion of

$$f_1(x_1 + h_1, x_2 + h_2, x_3 + h_3), \quad \text{near } (x_1, x_2, x_3).$$

The Multidimensional Newton's Method

Recall the idea of Newton's method:

If finding a zero of a function is difficult, replace the function with a simpler approximation (linear) whose zeros are easier to find.

Applying the principle:

- Linearize \mathbf{f} at the k th iterate \mathbf{x}_k :

$$\mathbf{f}(\mathbf{x}) \approx L(\mathbf{x}) = \mathbf{f}(\mathbf{x}_k) + \mathbf{J}(\mathbf{x}_k)(\mathbf{x} - \mathbf{x}_k).$$

- Define the next iterate \mathbf{x}_{k+1} by solving $L(\mathbf{x}_{k+1}) = \mathbf{0}$:

$$\mathbf{0} = \mathbf{f}(\mathbf{x}_k) + \mathbf{J}(\mathbf{x}_k)(\mathbf{x} - \mathbf{x}_k) \implies \mathbf{x}_{k+1} = \mathbf{x}_k - [\mathbf{J}(\mathbf{x}_k)]^{-1} \mathbf{f}(\mathbf{x}_k).$$

Note that $\mathbf{J}^{-1}\mathbf{f}$ plays the same role as f/f' in the scalar Newton.

The Multidimensional Newton's Method (cont')

- In practice, we do not compute \mathbf{J}^{-1} . Rather, the k th Newton step $\mathbf{s}_k = \mathbf{x}_{k+1} - \mathbf{x}_k$ is found by solving the square linear system

$$\mathbf{J}(\mathbf{x}_k)\mathbf{s}_k = -\mathbf{f}(\mathbf{x}_k),$$

which is solved using the backslash in MATLAB.

- Suppose `f` and `J` are MATLAB functions calculating \mathbf{f} and \mathbf{J} , respectively. Then the Newton iteration is done simply by

```
% x is a Newton iterate (a column vector).  
% The following is the key fragment  
% inside Newton iteration loop.  
fx = f(x)  
s = -J(x)\fx;  
x = x + s;
```

- Since $\mathbf{f}(x_k)$ is the residual and \mathbf{s}_k is the gap between two consecutive iterates at the k th step, monitor their norms to determine when to stop iteration.

Computer Illustration

- 1 Define f and J , either as anonymous functions or as function m-files.

```
f = @(x) [exp(x(2)-x(1)) - 2;  
         x(1)*x(2) + x(3);  
         x(2)*x(3) + x(1)^2 - x(2)];  
J = @(x) [-exp(x(2)-x(1)), exp(x(2)-x(1)), 0;  
         x(2), x(1), 1;  
         2*x(1), x(3)-1, x(2)];
```

- 2 Define an initial iterate x , say
 $x_0 = (0, 0, 0)^T$.

```
x = [0 0 0]';
```

- 3 Iterate.

```
for k = 1:7  
    s = -J(x) \ f(x);  
    x = x + s;  
end
```


Implementation

```
function x = newtonsys(f,x1)
% NEWTONSYS    Newton's method for a system of equations.
% Input:
%   f          function that computes residual and Jacobian matrix
%   x1         initial root approximation (n-vector)
% Output
%   x          array of approximations (one per column, last is best)

% Operating parameters.
funtol = 1000*eps;  xtol = 1000*eps;  maxiter = 40;

x = x1(:);
[y,J] = f(x1);
dx = Inf;
k = 1;

while (norm(dx) > xtol) && (norm(y) > funtol) && (k < maxiter)
    dx = -(J\y);    % Newton step
    x(:,k+1) = x(:,k) + dx;

    k = k+1;
    [y,J] = f(x(:,k));
end

if k==maxiter, warning('Maximum number of iterations reached. '), end
end
```