Newton's Method

FPI, when convergent, is linearly convergent.

Newton's Method

To find the root of f:

Newton's Method (Algorithm)

- Begin at the point $(x_0, f(x_0))$ on the curve and draw the tangent line at the point using the slope $f'(x_0)$:

$$y = f(x_0) + f'(x_0)(x - x_0).$$

- Find the x-intercept of the line and call it x_1: Set $y=0$ and solve for x.

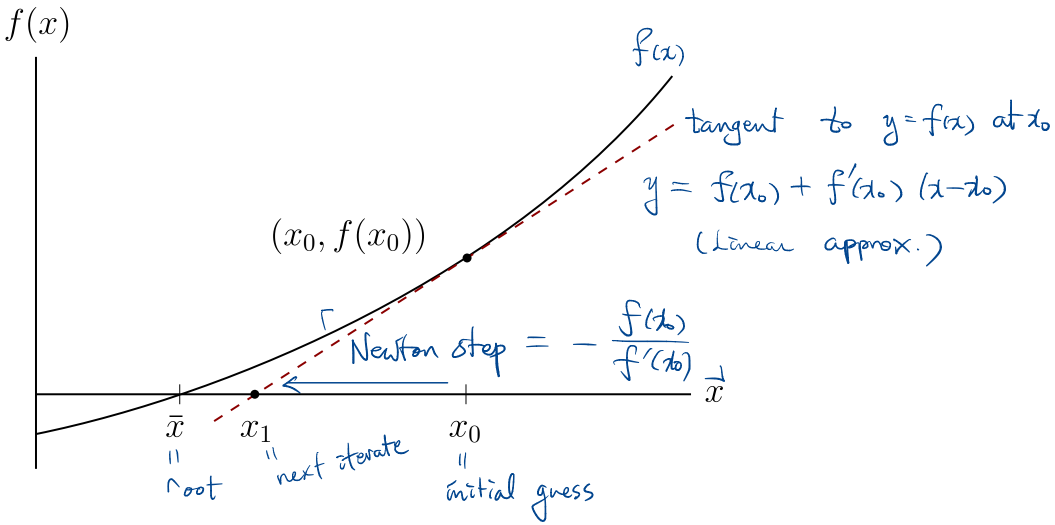
$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}.$$

- Continue this procedure to find $x_2, x_3, \ldots$ until the sequence converges to the root.

General iterative formula: (Newton iteration)

$$\boxed{x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} \quad \text{for } k = 0, 1, 2, \ldots} \qquad (\star)$$

Newton's Method: Illustration



Series Analysis

$\lim_{k\to\infty} x_k$

Let $\epsilon_k = x_k - r$, $k = 1, 2, \ldots$, where r is the limit of the sequence and $f(r) = 0$.

Substituting $x_k = r + \epsilon_k$ into the iterative formula ($\star$):

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

$x_{k+1} = r + \epsilon_{k+1}$

$$\epsilon_{k+1} = \epsilon_k - \frac{f(r+\epsilon_k)}{f'(r+\epsilon_k)}.$$

Taylor-expand f about $x = r$ and simplify (assuming $f'(r) \neq 0$):

$f(r) = 0$

$$\begin{aligned}
\underline{\epsilon_{k+1}} &= \epsilon_k - \frac{\cancel{f(r)}^{\,0} + \epsilon_k f'(r) + \frac{1}{2}\epsilon_k^2 f''(r) + O(\epsilon_k^3)}{f'(r) + \epsilon_k f''(r) + O(\epsilon_k^2)} \\
&= \epsilon_k - \epsilon_k \left[1 + \frac{1}{2}\frac{f''(r)}{f'(r)}\epsilon_k + O(\epsilon_k^2)\right]\left[1 + \frac{f''(r)}{f'(r)}\epsilon_k + O(\epsilon_k^2)\right]^{-1} \\
&= \underline{\frac{1}{2}\frac{f''(r)}{f'(r)}\epsilon_k^2} + O(\epsilon_k^3).
\end{aligned}$$

Geom. Series

Use: $\dfrac{1}{1-\alpha} = 1 + \alpha + \alpha^2 + \ldots$ for $|\alpha| < 1$

Exercise. Check algebra.

Cf $\epsilon_{k+1} \approx C\epsilon_k$

Series Analysis (cont')

- Previous calculation shows that $\boxed{\epsilon_{k+1} \approx C\epsilon_k^2}$, eventually. Written differently,

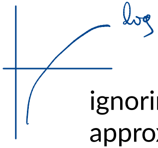
$$\lim_{k\to\infty} \frac{|\epsilon_{k+1}|}{|\epsilon_k|^2} = \text{Const.}$$

$$\left|\epsilon_{k+1}\right| / \left|\epsilon_k\right|^2 \to \text{(some positive number)}, \text{ as } k \to \infty.$$

that is, each Newton iteration roughly squares the previous error. This is **quadratic convergence**[3].

taking log

- Alternately, note that



$$\log|\epsilon_{k+1}| \approx 2\log|\epsilon_k| + \text{(constant)},$$

Note: As $k \to \infty$, $\epsilon_k \to 0$, So the constant term is negligible compared to log terms.

ignoring high-order terms. This means that the number of accurate digits[4] approximately doubles at each iteration.

(related to number of accurate digits of x_{k+1})

(related to number of accurate digits of x_k)

[3] Recall the formal definition given in p. 11.

[4] We say that an iterate is **correct within** p **decimal places** if the error is less than 0.5×10^{-p}.

Convergence of Newton's Method

(formal summary of prev. slides)

Theorem 4 (Quadratic Convergence of Newton's Method)

Let f be twice continuously differentiable and $f(r) = 0$. If $f'(r) \neq 0$, then Newton's method is locally and quadratically convergent to r. The error $\epsilon_k = x_k - r$ at step k satisfies

$$\lim_{k \to \infty} \frac{|\epsilon_{k+1}|}{|\epsilon_k|^2} = \left| \frac{f''(r)}{2f'(r)} \right|.$$

"r is a simple root"

Implementation

(from FNC)

Iteration: $x_1, x_2, x_3, \ldots$

```
function x = newton(f,dfdx,x1)
% NEWTON   Newton's method for a scalar equation.
% Input:
%   f         objective function
%   dfdx      derivative function
%   x1        initial root approximation
% Output
%   x         vector of root approximations (last one is best)

% Operating parameters.
    funtol = 100*eps;  xtol = 100*eps;  maxiter = 40;
    x = x1;
    y = f(x1);
    dx = Inf;   % for initial pass below
    k = 1;

    while (abs(dx) > xtol) && (abs(y) > funtol) && (k < maxiter)
        dydx = dfdx(x(k));
        dx = -y/dydx;            % Newton step
        x(k+1) = x(k) + dx;

        k = k+1;
        y = f(x(k));
    end

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

residual

$|x_{k+1} - x_k|$

# of iteration

"stopping criteria"

iteration formula.

	MATLAB	Math.
(vector)	x	x_k
	y	$f(x_k)$ residual
	dx	$x_{k+1} - x_k$

$$x_{k+1} - x_k = -\frac{f(x_k)}{f'(x_k)}$$

Note: Stopping Criteria

Previous slide.

For a set tolerance, TOL, some example stopping criteria are:

these 3 were used.

- Absolute error:

$$|x_{k+1} - x_k| < \texttt{TOL}.$$

- Relative error: (useful when the solution is not too close to zero)

$$\frac{|x_{k+1} - x_k|}{|x_{k+1}|} < \texttt{TOL}.$$

- Hybrid:

$$\frac{|x_{k+1} - x_k|}{\max(|x_{k+1}|, \theta)} < \texttt{TOL},$$

  for some $\theta > 0$.

- Residual:

$$|f(x_k)| < \texttt{TOL}.$$

Also useful to set a limit on the maximum number of iterations in case convergence fails.

Secant Method

Secant Method

- Newton's method requires calculation and evaluation of $f'(x)$, which may be challenging at times.
- The most common alternative to such situations is the **secant method**.
- The secant method replaces the instantaneous slope in Newton's method by the average slope using the last two iterates.

Secant Method (cont')

Secant Method (Algorithm)

- Begin with two initial iterates x_{-1} and x_0; draw the secant line connecting $(x_{-1}, f(x_{-1}))$ and $(x_0, f(x_0))$:

Slope of secant line → cf. Newton: $f'(x_0)$

$$y = f(x_0) + \boxed{\frac{f(x_0) - f(x_{-1})}{x_0 - x_{-1}}}(x - x_0).$$

- Find the x-intercept of the line and call it x_1:

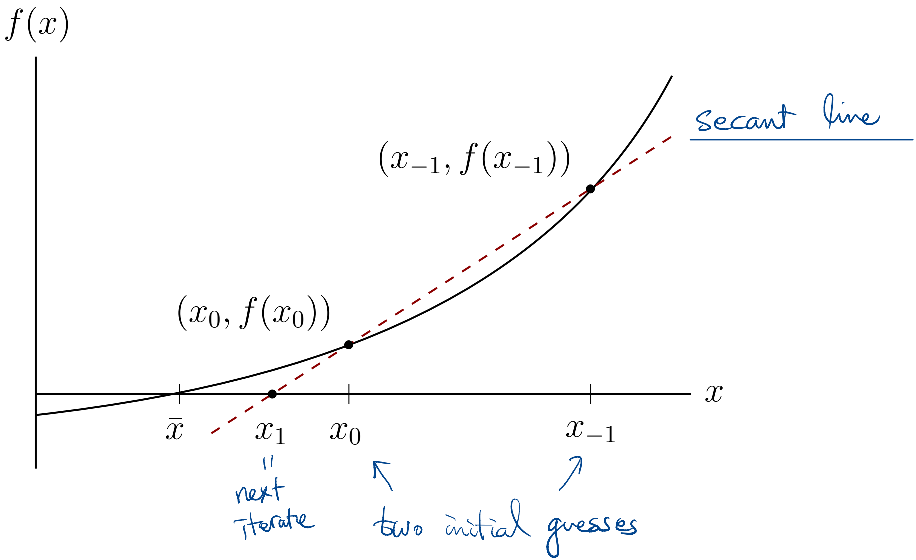
$$x_1 = x_0 - f(x_0)\frac{x_0 - x_{-1}}{f(x_0) - f(x_{-1})}.$$

- Continue this procedure to find $x_2, x_3, \ldots$ until convergence is obtained.

General iterative formula:

$$\boxed{x_{k+1} = x_k - f(x_k)\frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})} \quad \text{for } k = 0, 1, 2, \ldots}$$

Secant Method: Illustration



Series Analysis

Assume that the secant method converges to r and $f'(r) \neq 0$. Let $\epsilon_k = x_k = r$ as before.

It can be shown that

$$|\epsilon_{k+1}| \approx \left|\frac{f''(r)}{2f'(r)}\right| |\epsilon_k|\,|\epsilon_{k-1}|\,,$$

which implies that

$$\boxed{|\epsilon_{k+1}| \approx \left|\frac{f''(r)}{2f'(r)}\right|^{\alpha-1} |\epsilon_k|^{\alpha},}$$

where

$$\alpha = \frac{1+\sqrt{5}}{2} \approx 1.618,$$

the *golden ratio*.

Therefore, the convergence of the secant method is **superlinear**; it lies between linearly and quadratically convergent methods.

Convergence

FPI : linear
Secant : superlinear
Newton : quadratic

Exercise. Confirm the statements in the previous page. Namely, show that

1. The error ϵ_k satisfies the approximate equation

$$|\epsilon_{k+1}| \approx \left|\frac{f''(r)}{2f'(r)}\right| |\epsilon_k| \, |\epsilon_{k-1}| \, .$$

2. If in addition $\lim_{k\to\infty} |\epsilon_{k+1}| / |\epsilon_k|^\alpha$ exists and is nonzero for some $\alpha > 0$, then

$$|\epsilon_{k+1}| \approx \left|\frac{f''(r)}{2f'(r)}\right|^{\alpha-1} |\epsilon_k|^\alpha \, , \quad \text{where } \alpha = \frac{1+\sqrt{5}}{2}.$$

Implementation

```
function x = secant(f,x1,x2)
% SECANT   Secant method for a scalar equation.
% Input:
%   f        objective function
%   x1,x2    initial root approximations
% Output
%   x        vector of root approximations (last is best)

% Operating parameters.
    funtol = 100*eps;  xtol = 100*eps;  maxiter = 40;

    x = [x1 x2];
    dx = Inf;  y1 = f(x1);
    k = 2;  y2 = 100;

    while (abs(dx) > xtol) && (abs(y2) > funtol) && (k < maxiter)
        y2 = f(x(k));
        dx = -y2 * (x(k)-x(k-1)) / (y2-y1);   % secant step
        x(k+1) = x(k) + dx;

        k = k+1;
        y1 = y2;    % current f-value becomes the old one next time
    end

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

Lec 23: Rootfinding Problem – Higher Dimensions

Newton’s Method for Nonlinear Systems

Multidimensional Rootfinding Problem

Rootfinding Problem: Vector Version

Given a continuous vector-valued function $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^n$, 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:

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

Multidimensional Taylor Series

"small"

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

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

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

$\in \mathbb{R}^{n\times n}$

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

- 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

$$
\begin{aligned}
f_1(x_1, x_2, x_3) &= -x_1 \cos(x_2) - 1, \\
f_2(x_1, x_2, x_3) &= x_1 x_2 + x_3, \\
f_3(x_1, x_2, x_3) &= e^{-x_3} \sin(x_1 + x_2) + x_1^2 - x_2^2.
\end{aligned}
$$

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 kth 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 $L(\mathbf{x}_{k+1}) = \mathbf{0}$: → Topic 2

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

→ matrix inverse.

Note that $\mathbf{J}^{-1}$ 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 kth Newton step $\mathbf{s}_k = x_{k+1} - x_k$ 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 kth step, monitor their norms to determine when to stop iteration.

Computer Illustration

Let's find a root of the function introduced in the example on p. 5.

1. Define $\mathbf{f}$ and $\mathbf{J}$, 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)];
```

1. Define an initial iterate x, say $\mathbf{x}_0 = (0,0,0)^{\mathrm{T}}$.

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

1. 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
```