Nonlinear Rootfinding

Contents

Introduction

The Rootfinding Problem

Problem Statement

Rootfinding Problem

Given a continuous scalar function of a scalar variable, find a real number r such that $f(r) = 0$.

- r is a **root** of the function f.
- The formulation $f(x) = 0$ is general enough; *e.g.*, to solve $g(x) = h(x)$, set $f = g - h$ and find a root of f.

Iterative Methods

- Unlike the earlier linear problems, the root cannot be produced in a finite number of operations.
- Rather, a sequence of approximations that formally converge to the root is pursued.

Iteration Strategy for Rootfinding. To find the root of f:

1. Start with an initial iterate, say x_0.
2. Generate a sequence of iterates $x_1, x_2, \ldots$ using an *iteration algorithm* of the form
$$x_{k+1} = g(x_k), \quad k = 0, 1, \ldots$$
3. Continue the iteration process until you find an x_i such that $f(x_i) = 0$. (In practice, continue until some member of the sequence seems to be "good enough".)

MATLAB's FZERO

`fzero` is MATLAB's general purpose rootfinding tool.

Syntax:

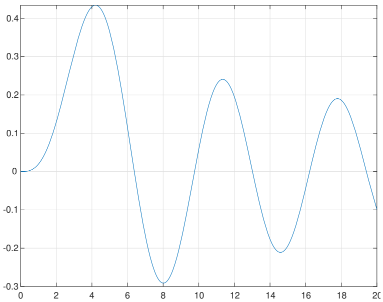
```
x_zero = fzero( <function>, <initial iterate> )
x_zero = fzero( <function>, <initial interval> )
[x_zero, fx_zero] = ....
```

Example

The roots of J_m, a Bessel function of the first kind, is found by

- Plot the function.
- Find approximate locations of roots.

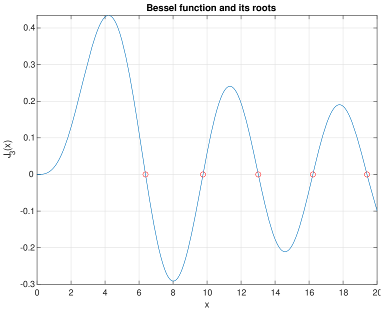
```
J3 = @(x) besselj(3,x);
fplot(J3,[0 20])
grid on
guess = [6,10,13,16,19];
```



Example (cont')

- Then use `fzero` to locate the roots:

```
omega = zeros(size(guess));
for j = 1:length(guess)
  omega(j) = fzero(J3,guess(j));
end
hold on
plot(omega,J3(omega),'ro')
```



Conditioning

- Sensitivity of the rootfinding problem can be measured in terms of the condition number:

$$\text{(absolute condition number)} = \frac{|\text{abs. error in output}|}{|\text{abs. error in input}|},$$

where, in the context of finding roots of f,

  - input: f (function)
  - output: r (root)

- Denote the changes by:
  - error/change in input: ϵg, where $\epsilon > 0$ is small $\quad (f \mapsto f + \epsilon g)$
  - error/change in output: Δr $\quad (r \mapsto r + \Delta r)$

- The *perturbed equation*

$$f(r+\Delta r)+\epsilon g(r+\Delta r)=0$$

  is linearized to (Taylor expansion)

$$f(r)+f'(r)\Delta r+g(r)\epsilon+g'(r)\epsilon\Delta r\approx 0,$$

  ignoring $O((\Delta r)^2)$ terms[1].

- Since $f(r)=0$, we solve for Δr to get

$$\Delta r\approx -\epsilon\frac{g(r)}{f'(r)+\epsilon g'(r)}\approx -\epsilon\frac{g(r)}{f'(r)},$$

  for small ϵ compared with $f'(r)$.

[1]That is, terms involving $(\Delta r)^2$ and higher powers of Δr

Conditioning (cont')

- Therefore, the absolute condition number of the rootfinding problem is

$$\kappa_{f \mapsto r} = \frac{1}{|f'(r)|},$$

  which implies that the problem is highly sensitive whenever $f'(r) \approx 0$.

- In other words, if $|f'|$ is small at the root, a computed *root estimate* may involve large errors.

Residual and Backward Error

- Without knowing the exact root, we cannot compute the error.
- But the **residual** of a root estimate $\tilde{r}$ can be computed:

$$\text{(residual)} = f(\tilde{r}).$$

- Small residual *might* be associated with a small error.
- The residual $f(\tilde{r})$ is the *backward error* of the estimate.

Multiple Roots

Definition 1 (Multiplicity of Roots)

Assume that r is a root of the differentiable function f. Then if

$$0 = f(r) = f'(r) = \cdots = f^{(m-1)}(r) \quad \text{but} \quad f^{(m)}(r) \neq 0,$$

we say that f has a root of **multiplicity** m at r.

- We say that f has a **multiple root** at r if the multiplicity is greater than 1.
- A root is called **simple** if its multiplicity is 1.
- If r is a multiple root, the condition number is infinite.
- Even if r is a simple root, we expect difficulty in numerical computation if $f'(r) \approx 0$.

One Dimension

Fixed Point Iteration

Fixed Point

Definition 2 (Fixed Point)

The real number r is a **fixed point** of the function g if $g(r) = r$.

- The rootfinding problem $f(x) = 0$ can always be written as a fixed point problem $g(x) = x$ by, *e.g.*, setting[2]

$$g(x) = x - f(x).$$

- The fixed point problem is true at, and only at, a root of f.

[2] This is not the only way to transform the rootfinding problem. More on this later.

Fixed Point Iteration

A fixed point problem $g(x) = x$ naturally provides an iteration scheme:

$$\begin{cases} x_0 = \text{initial guess} \\ x_{k+1} = g(x_k), \quad k = 0, 1, 2, \ldots. \end{cases} \qquad \text{(fixed point iteration)}$$

- The sequence $\{x_k\}$ may or may not converge as $k \to \infty$.
- If g is continuous and $\{x_k\}$ converges to a number r, then r is a fixed point of g.

$$g(r) = g\left(\lim_{k\to\infty} x_k\right) = \lim_{k\to\infty} g(x_k) = \lim_{k\to\infty} x_{k+1} = r.$$

Fixed Point Iteration Algorithm

```
function x = fpi(g, x0, n)
% FPI x = fpi(g, x0, n)
% Computes approximate solution of g(x)=x
% Input:
%    g     function handle
%    x0    initial guess
%    n     number of iteration steps
    x = x0;
    for k = 1:n
        x = g(x);
    end
end
```

Examples

- To find a fixed point of $g(x) = 0.3\cos(2x)$ near 0.5 using `fpi`:

```
g = @(x) 0.3*cos(2*x);
xc = fpi(g,0.5,20)
```

```
xc = 0.260266319627758
```

Not All Fixed Point Problems Are The Same

The rootfinding problem $f(x) = x^3 + x - 1 = 0$ can be transformed to various fixed point problems:

- $g_1(x) = x - f(x) = 1 - x^3$
- $g_2(x) = \sqrt[3]{1 - x}$
- $g_3(x) = \dfrac{1 + 2x^3}{1 + 3x^2}$

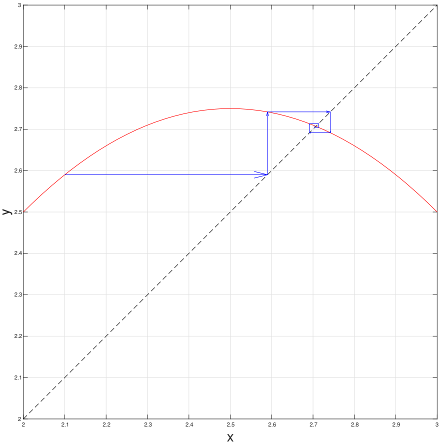
Note that all $g_j(x) = x$ are equivalent to $f(x) = 0$. However, not all these find a fixed point of g, that is, a root of f on the computer.

Exercise. Run `fpi` with g_j and $x_0 = 0.5$. Which fixed point iterations converge?

Geometry of Fixed Point Iteration

The following script[3] finds a root of $f(x) = x^2 - 4x + 3.5$ via FPI.

```
f = @(x) x.^2 - 4*x + 3.5;
g = @(x) x - f(x);
fplot(g, [2 3], 'r');
hold on
plot([2 3], [2 3], 'k--')
x = 2.1;
y = g(x);
for k = 1:5
    arrow([x y], [y y], 'b');
    x = y; y = g(x);
    arrow([x x], [x y], 'b');
end
```



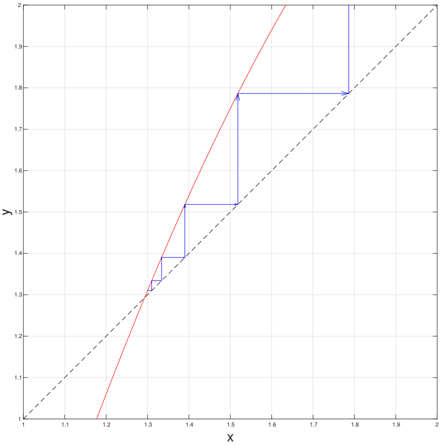
Note the line segments spiral in towards the fixed point.

[3]Modified from FNC.

Geometry of Fixed Point Iteration (cont')

However, with a different starting point, the process does not converge.

```
clf
fplot(g, [1 2], 'r');
hold on
plot([1 2], [1 2], 'k--'),
ylim([1 2])
x = 1.3; y = g(x);
for k = 1:5
    arrow([x y], [y y], 'b');
    x = y; y = g(x);
    arrow([x x], [x y], 'b');
end
```



Custom function: `arrow = @(p1, p2, varargin) quiver(p1(1), p1(2), p2(1)-p1(1), p2(2)-p1(2), 0, varargin{:})`

Series Analysis

Let $\epsilon_k = x_k - r$ be the sequence of errors.

- The iteration formula $x_{k+1} = g(x_k)$ can be written as

$$\begin{aligned} \epsilon_{k+1} + r &= g(\epsilon_k + r) \\ &= g(r) + g'(r)\epsilon_k + \frac{1}{2}g''(r)\epsilon_k^2 + \cdots , \qquad \text{(Taylor series)} \end{aligned}$$

  implying

$$\epsilon_{k+1} = g'(r)\epsilon_k + O(\epsilon_k^2)$$

  assuming sufficient regularity of g.

- Neglecting the second-order term, we have $\epsilon_{k+1} \approx g'(r)\epsilon_k$, which is satisfied if $\epsilon_k \approx C\left[g'(r)\right]^k$ for sufficiently large k.

- Therefore, the iteration converges if $\left|g'(r)\right| < 1$ and diverges if $\left|g'(r)\right| > 1$.

Note: Rate of Convergence

Definition 3 (Linear Convergnece)

Suppose $\lim_{k\to\infty} x_k = r$ and let $\epsilon_k = x_k - r$, the error at step k of an iteration method. If

$$\lim_{k\to\infty} \frac{|\epsilon_{k+1}|}{|\epsilon_k|} = \sigma < 1,$$

the method is said to obey **linear convergence** with rate σ.

Note. In general, say

$$\lim_{k\to\infty} \frac{|\epsilon_{k+1}|}{|\epsilon_k|^p} = \sigma$$

for some $p \geq 1$ and $\sigma > 0$.

- If $p = 1$ and
  - $\sigma = 1$, the convergence is *sublinear*;
  - $0 < \sigma < 1$, the convergence is *linear*;
  - $\sigma = 0$, the convergence is *superlinear*.
- If $p = 2$, the convergence is *quadratic*;
- If $p = 3$, the convergence is *cubic*, ...

Convergence of Fixed Point Iteration

Theorem 4 (Convergence of FPI)

Assume that g is continuously differentiable, that $g(r) = r$, and that $\sigma = \left|g'(r)\right| < 1$. Then the fixed point iterates x_k generated by

$$x_{k+1} = g(x_k), \quad k = 1, 2, \dots,$$

converge linearly with rate σ to the fixed point r for x_0 sufficiently close to r.

In the previous example with $g(x) = x - f(x) = -x^2 + 5x - 3.5$:

- For the first fixed point, near 2.71, we get $g'(r) \approx -0.42$ (convergence);
- For the second fixed point, near 1.29, we get $g'(r) \approx 2.42$ (divergence).

Note. An iterative method is called **locally convergent** to r if the method converges to r for initial guess sufficiently close to r.

Contraction Maps

Lipschitz Condition

A function g is said to satisfy a **Lipschitz condition** with constant L on the interval $S \subset \mathbb{R}$ if

$$|g(s) - g(t)| \leq L\,|s - t| \quad \text{for all } s, t \in S.$$

- A function satisfying the Lipschitz condition is continuous on S.
- If $L < 1$, g is called a **contraction map**.

When Does FPI Succeed?

Contraction Mapping Theorem

Suppose that g satisfies Lipschitz condition on S with $L < 1$, *i.e.*, g is a contraction map on S. Then S contains exactly one fixed point r of g. If $x_1, x_2, \ldots$ are generated by the fixed point iteration $x_{k+1} = g(x_k)$, and $x_1, x_2, \ldots$ all lie in S, then

$$|x_k - r| \leq L^{k-1} |x_1 - r|, \quad k > 1.$$

Newton's Method

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:

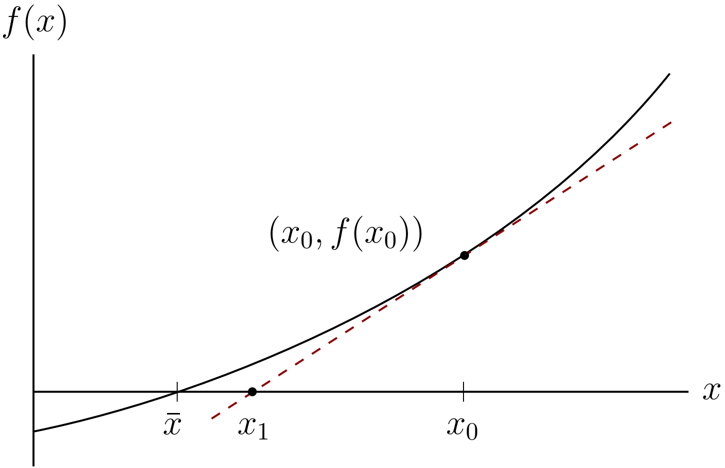
$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}.$$

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

General iterative formula:

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

Newton's Method: Illustration



Series Analysis

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 ($\star$):

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

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

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

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

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

  that is, each Newton iteration roughly squares the previous error. This is **quadratic convergence**[4].

- Alternately, note that

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

  ignoring high-order terms. This means that the number of accurate digits[5] approximately doubles at each iteration.

[4] Recall the formal definition given in p. 25.

[5] 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

Theorem 5 (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

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

Implementation

```
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
```

Note: Stopping Criteria

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

- Absolute error:

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

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

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

- Hybrid:

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

  for some $\theta > 0$.

- Residual:

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

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 $(x_0, f(x_0))$:

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

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

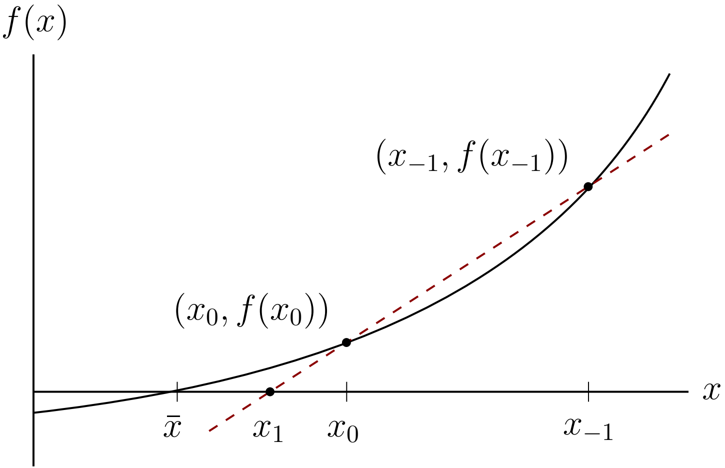
$$x_1 = x_0 - f(x_0)\frac{x_0 - x_{-1}}{f(x_0) - f(x_{-1})}.$$

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

General iterative formula:

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

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

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

which implies that

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

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.

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
```

Other Methods

Inverse Interpolation

The **inverse quadratic interpolation** (IQI) is a generalization of the secant method to parabolas.

- Instead of using two most recent points (to determine a straight line), use three and obtain an quadratic interpolant.
- The parabola of the form $y = p(x)$ may have zero, one, or two x-intercept(s). So use the form $x = p(y)$, a parabola open sideways.

Algorithm.

- Begin with three initial iterates x_{-2}, x_{-1}, x_0; find the parabola of the form $x = p(y)$ passing through the three points $(x_{-2}, f(x_{-2}))$, $(x_{-1}, f(x_{-1}))$, and $(x_0, f(x_0))$.
- Find the x-intercept of the parabola and call it x_1.
- Continue the procedure to find $x_2, x_3, \ldots$ until convergence is obtained.

Inverse Interpolation (cont')

General iterative formula:

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

where

$$q = \frac{f(x_{k-2})}{f(x_{k-1})}, \quad r = \frac{f(x_k)}{f(x_{k-1})}, \quad s = \frac{f(x_k)}{f(x_{k-2})}.$$

Rather than deriving and implementing the formula, try using `polyfit` to perform the interpolation step.

Bisection Method: Bracketing a Root

The following is a corollary to the intermediate value theorem.

Theorem 6 (Existence of a Root)

Let f be a continuous function on $[a, b]$, satisfying $f(a)f(b) < 0$. Then f has a root between a and b, that is, there exists a number $r \in (a, b)$ such that $f(r) = 0$.

Bisection Method (cont')

Algorithm.

- Start with an interval $[a, b]$ where $f(a)f(b) \leq 0$.
- Bisect the interval into $[a, m] \cup [m, b]$ where $m = (a + b)/2$ is the midpoint.
- Select the subinterval in which $f(x)$ changes signs, *i.e.*, calculate $f(a)f(m)$ and $f(m)f(b)$, choose the nonpositive one, and update the values of a and b.
- Repeat the process until you get close enough to the solution.

Notes

Let $[a, b]$ be the initial interval and let $[a_k, b_k]$ be the interval after k bisection steps.

- The length of $[a_k, b_k]$ is $(b - a)/2^k$.
- Using the midpoint $x_k = (a_k + b_k)/2$ as an estimate of the root r, note that

$$|\epsilon_k| = |x_k - r| < \frac{b - a}{2^{k+1}}.$$

- This accuracy is obtained by $k + 2$ function evaluations.

Bisection Method: Pseudocode

```
while <a NOT CLOSE ENOUGH TO b>
  m = (a + b)/2;
  fm = f(m);
  if sign(fa) ~= sign(fm)
    b = m;
    fb = fm;
  else
    a = m;
    fa = fm;
  end
end
x_zero = .5*(a + b);
```

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:

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

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 $\mathbf{f}$

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

- 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 $L(\mathbf{x}_{k+1}) = \mathbf{0}$:

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

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 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

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 $\mathtt{x}$, 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

```matlab
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
```