Outline

1. Nonlinear Equations
2. Numerical Methods in One Dimension
3. Methods for Systems of Nonlinear Equations
Given function $f$, we seek value $x$ for which

$$f(x) = 0$$

Solution $x$ is root of equation, or zero of function $f$

So problem is known as root finding or zero finding
Two important cases

Single nonlinear equation in one unknown, where

\[ f : \mathbb{R} \rightarrow \mathbb{R} \]

Solution is scalar \( x \) for which \( f(x) = 0 \)

System of \( n \) coupled nonlinear equations in \( n \) unknowns, where

\[ f : \mathbb{R}^n \rightarrow \mathbb{R}^n \]

Solution is vector \( x \) for which all components of \( f \) are zero simultaneously, \( f(x) = 0 \)
Examples: Nonlinear Equations

Example of nonlinear equation in one dimension

\[ x^2 - 4 \sin(x) = 0 \]

for which \( x = 1.9 \) is one approximate solution

Example of system of nonlinear equations in two dimensions

\[
\begin{align*}
    x_1^2 - x_2 + 0.25 &= 0 \\
    -x_1 + x_2^2 + 0.25 &= 0
\end{align*}
\]

for which \( \mathbf{x} = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}^T \) is solution vector
Existence and uniqueness of solutions are more complicated for nonlinear equations than for linear equations.

For function $f : \mathbb{R} \rightarrow \mathbb{R}$, a *bracket* is interval $[a, b]$ for which the sign of $f$ differs at endpoints.

If $f$ is continuous and $\text{sign}(f(a)) \neq \text{sign}(f(b))$, then the Intermediate Value Theorem implies there is $x^* \in [a, b]$ such that $f(x^*) = 0$.

There is no simple analog for $n$ dimensions.
Examples: One Dimension

Nonlinear equations can have any number of solutions

\[ \exp(x) + 1 = 0 \] has no solution

\[ \exp(-x) - x = 0 \] has one solution

\[ x^2 - 4 \sin(x) = 0 \] has two solutions

\[ x^3 + 6x^2 + 11x - 6 = 0 \] has three solutions

\[ \sin(x) = 0 \] has infinitely many solutions
Example: Systems in Two Dimensions

\[
\begin{align*}
    x_1^2 - x_2 + \gamma &= 0 \\
    -x_1 + x_2^2 + \gamma &= 0
\end{align*}
\]

\(\gamma = 0.5\) \hspace{2cm} \(\gamma = 0.25\)

\(\gamma = -0.5\) \hspace{2cm} \(\gamma = -1.0\)
If \( f(x^*) = f'(x^*) = f''(x^*) = \cdots = f^{(m-1)}(x^*) = 0 \) but \( f^{(m)}(x^*) \neq 0 \) (i.e., \( m \)th derivative is lowest derivative of \( f \) that does not vanish at \( x^* \)), then root \( x^* \) has multiplicity \( m \).
Sensitivity and Conditioning

Conditioning of root finding problem is opposite to that for evaluating function.

Absolute condition number of root finding problem for root $x^*$ of $f: \mathbb{R} \rightarrow \mathbb{R}$ is $1/|f'(x^*)|$

Root is ill-conditioned if tangent line is nearly horizontal.

In particular, multiple root ($m > 1$) is ill-conditioned.

Absolute condition number of root finding problem for root $x^*$ of $f: \mathbb{R}^n \rightarrow \mathbb{R}^n$ is $\|J_f^{-1}(x^*)\|$, where $J_f$ is Jacobian matrix of $f$,

$$\{J_f(x)\}_{ij} = \partial f_i(x)/\partial x_j$$

Root is ill-conditioned if Jacobian matrix is nearly singular.
Sensitivity and Conditioning

well-conditioned

ill-conditioned
What do we mean by approximate solution $\hat{x}$ to nonlinear system,

\[ \| f(\hat{x}) \| \approx 0 \quad \text{or} \quad \| \hat{x} - x^* \| \approx 0 ? \]

First corresponds to “small residual,” second measures closeness to (usually unknown) true solution $x^*$

Solution criteria are not necessarily “small” simultaneously

Small residual implies accurate solution only if problem is well-conditioned
Convergence Rate

For general iterative methods, define error at iteration $k$ by

$$e_k = x_k - x^*$$

where $x_k$ is approximate solution and $x^*$ is true solution.

For methods that maintain interval known to contain solution, rather than specific approximate value for solution, take error to be length of interval containing solution.

Sequence converges with rate $r$ if

$$\lim_{k \to \infty} \frac{\|e_{k+1}\|}{\|e_k\|^r} = C$$

for some finite nonzero constant $C$. 
Some particular cases of interest

\[ r = 1: \textit{linear} \quad (C < 1) \]

\[ r > 1: \textit{superlinear} \]

\[ r = 2: \textit{quadratic} \]

<table>
<thead>
<tr>
<th>Convergence rate</th>
<th>Digits gained per iteration</th>
</tr>
</thead>
<tbody>
<tr>
<td>linear</td>
<td>constant</td>
</tr>
<tr>
<td>superlinear</td>
<td>increasing</td>
</tr>
<tr>
<td>quadratic</td>
<td>double</td>
</tr>
</tbody>
</table>
Interval Bisection Method

**Bisection** method begins with initial bracket and repeatedly halves its length until solution has been isolated as accurately as desired

```plaintext
while ((b - a) > tol) do
    m = a + (b - a)/2
    if sign(f(a)) = sign(f(m)) then
        a = m
    else
        b = m
    end
end
```

< interactive example >
Example: Bisection Method

\[ f(x) = x^2 - 4 \sin(x) = 0 \]

<table>
<thead>
<tr>
<th>( a )</th>
<th>( f(a) )</th>
<th>( b )</th>
<th>( f(b) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.000000</td>
<td>-2.365884</td>
<td>3.000000</td>
<td>8.435520</td>
</tr>
<tr>
<td>1.000000</td>
<td>-2.365884</td>
<td>2.000000</td>
<td>0.362810</td>
</tr>
<tr>
<td>1.500000</td>
<td>-1.739980</td>
<td>2.000000</td>
<td>0.362810</td>
</tr>
<tr>
<td>1.750000</td>
<td>-0.873444</td>
<td>2.000000</td>
<td>0.362810</td>
</tr>
<tr>
<td>1.875000</td>
<td>-0.300718</td>
<td>2.000000</td>
<td>0.362810</td>
</tr>
<tr>
<td>1.875000</td>
<td>-0.300718</td>
<td>1.937500</td>
<td>0.019849</td>
</tr>
<tr>
<td>1.906250</td>
<td>-0.143255</td>
<td>1.937500</td>
<td>0.019849</td>
</tr>
<tr>
<td>1.921875</td>
<td>-0.062406</td>
<td>1.937500</td>
<td>0.019849</td>
</tr>
<tr>
<td>1.929688</td>
<td>-0.021454</td>
<td>1.937500</td>
<td>0.019849</td>
</tr>
<tr>
<td>1.933594</td>
<td>-0.000846</td>
<td>1.937500</td>
<td>0.019849</td>
</tr>
<tr>
<td>1.933594</td>
<td>-0.000846</td>
<td>1.935547</td>
<td>0.009491</td>
</tr>
<tr>
<td>1.933594</td>
<td>-0.000846</td>
<td>1.934570</td>
<td>0.004320</td>
</tr>
<tr>
<td>1.933594</td>
<td>-0.000846</td>
<td>1.934082</td>
<td>0.001736</td>
</tr>
</tbody>
</table>
Bisection Method, continued

Bisection method makes no use of magnitudes of function values, only their signs.

Bisection is certain to converge, but does so slowly.

At each iteration, length of interval containing solution reduced by half, convergence rate is linear, with $r = 1$ and $C' = 0.5$.

One bit of accuracy is gained in approximate solution for each iteration of bisection.

Given starting interval $[a, b]$, length of interval after $k$ iterations is $(b - a)/2^k$, so achieving error tolerance of $tol$ requires

$$\left\lceil \log_2 \left( \frac{b - a}{tol} \right) \right\rceil$$

iterations, regardless of function $f$ involved.
Newton’s Method

Truncated Taylor series

\[ f(x + h) \approx f(x) + f'(x)h \]

is linear function of \( h \) approximating \( f \) near \( x \)

Replace nonlinear function \( f \) by this linear function, whose zero is \( h = -f(x)/f'(x) \)

Zeros of original function and linear approximation are not identical, so repeat process, giving \textit{Newton’s method}

\[ x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} \]
Newton’s method approximates nonlinear function $f$ near $x_k$ by *tangent line* at $f(x_k)$.
Example: Newton’s Method

Use Newton’s method to find root of

\[ f(x) = x^2 - 4 \sin(x) = 0 \]

Derivative is

\[ f'(x) = 2x - 4 \cos(x) \]

so iteration scheme is

\[ x_{k+1} = x_k - \frac{x_k^2 - 4 \sin(x_k)}{2x_k - 4 \cos(x_k)} \]

Taking \( x_0 = 3 \) as starting value, we obtain

<table>
<thead>
<tr>
<th>( x )</th>
<th>( f(x) )</th>
<th>( f'(x) )</th>
<th>( h )</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.000000</td>
<td>8.435520</td>
<td>9.959970</td>
<td>-0.846942</td>
</tr>
<tr>
<td>2.153058</td>
<td>1.294772</td>
<td>6.505771</td>
<td>-0.199019</td>
</tr>
<tr>
<td>1.954039</td>
<td>0.108438</td>
<td>5.403795</td>
<td>-0.020067</td>
</tr>
<tr>
<td>1.933972</td>
<td>0.001152</td>
<td>5.288919</td>
<td>-0.000218</td>
</tr>
<tr>
<td>1.933754</td>
<td>0.000000</td>
<td>5.287670</td>
<td>0.000000</td>
</tr>
</tbody>
</table>
Newton’s method transforms nonlinear equation $f(x) = 0$ into fixed-point problem $x = g(x)$, where

$$g(x) = x - \frac{f(x)}{f'(x)}$$

and hence

$$g'(x) = \frac{f(x)f''(x)}{(f'(x))^2}$$

If $x^*$ is simple root (i.e., $f(x^*) = 0$ and $f'(x^*) \neq 0$), then $g'(x^*) = 0$

Convergence rate of Newton’s method for simple root is therefore \textit{quadratic} ($r = 2$)

But iterations must start close enough to root to converge

< interactive example >
Newton’s Method, continued

For multiple root, convergence rate of Newton’s method is only linear, with constant $C = 1 - (1/m)$, where $m$ is multiplicity

<table>
<thead>
<tr>
<th>$k$</th>
<th>$f(x) = x^2 - 1$</th>
<th>$f(x) = x^2 - 2x + 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2.0</td>
<td>2.0</td>
</tr>
<tr>
<td>1</td>
<td>1.25</td>
<td>1.5</td>
</tr>
<tr>
<td>2</td>
<td>1.025</td>
<td>1.25</td>
</tr>
<tr>
<td>3</td>
<td>1.0003</td>
<td>1.125</td>
</tr>
<tr>
<td>4</td>
<td>1.0000000005</td>
<td>1.0625</td>
</tr>
<tr>
<td>5</td>
<td>1.0</td>
<td>1.03125</td>
</tr>
</tbody>
</table>
For each iteration, Newton’s method requires evaluation of both function and its derivative, which may be inconvenient or expensive.

In *secant method*, derivative is approximated by finite difference using two successive iterates, so iteration becomes

\[ x_{k+1} = x_k - f(x_k) \frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})} \]

Convergence rate of secant method is normally *superlinear*, with \( r \approx 1.618 \).
Secant method approximates nonlinear function $f$ by secant line through previous two iterates.
Example: Secant Method

Use secant method to find root of

\[ f(x) = x^2 - 4\sin(x) = 0 \]

Taking \( x_0 = 1 \) and \( x_1 = 3 \) as starting guesses, we obtain

<table>
<thead>
<tr>
<th>( x )</th>
<th>( f(x) )</th>
<th>( h )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.000000</td>
<td>-2.365884</td>
<td></td>
</tr>
<tr>
<td>3.000000</td>
<td>8.435520</td>
<td>-1.561930</td>
</tr>
<tr>
<td>1.438070</td>
<td>-1.896774</td>
<td>0.286735</td>
</tr>
<tr>
<td>1.724805</td>
<td>-0.977706</td>
<td>0.305029</td>
</tr>
<tr>
<td>2.029833</td>
<td>0.534305</td>
<td>-0.107789</td>
</tr>
<tr>
<td>1.922044</td>
<td>-0.061523</td>
<td>0.011130</td>
</tr>
<tr>
<td>1.933174</td>
<td>-0.003064</td>
<td>0.000583</td>
</tr>
<tr>
<td>1.933757</td>
<td>0.000019</td>
<td>-0.000004</td>
</tr>
<tr>
<td>1.933754</td>
<td>0.000000</td>
<td>0.000000</td>
</tr>
</tbody>
</table>
Secant method uses linear interpolation to approximate function whose zero is sought.

Higher convergence rate can be obtained by using higher-degree polynomial interpolation.

For example, quadratic interpolation (Muller’s method) has superlinear convergence rate with $r \approx 1.839$.

Unfortunately, using higher degree polynomial also has disadvantages:

- Interpolating polynomial may not have real roots.
- Roots may not be easy to compute.
- Choice of root to use as next iterate may not be obvious.
Safeguarded Methods

Rapidly convergent methods for solving nonlinear equations may not converge unless started close to solution, but safe methods are slow.

Hybrid methods combine features of both types of methods to achieve both speed and reliability.

Use rapidly convergent method, but maintain bracket around solution.

If next approximate solution given by fast method falls outside bracketing interval, perform one iteration of safe method, such as bisection.
Safeguarded Methods, continued

Fast method can then be tried again on smaller interval with greater chance of success

Ultimately, convergence rate of fast method should prevail

Hybrid approach seldom does worse than safe method, and usually does much better
Solving systems of nonlinear equations is much more difficult than scalar case because

Wider variety of behavior is possible, so determining existence and number of solutions or good starting guess is much more complex

There is no simple way, in general, to guarantee convergence to desired solution or to bracket solution to produce absolutely safe method

Computational overhead increases rapidly with dimension of problem
Newton’s Method

In $n$ dimensions, *Newton’s method* has form

$$x_{k+1} = x_k - J(x_k)^{-1} f(x_k)$$

where $J(x)$ is Jacobian matrix of $f$,

$$\{J(x)\}_{ij} = \frac{\partial f_i(x)}{\partial x_j}$$

In practice, we do not explicitly invert $J(x_k)$, but instead solve linear system

$$J(x_k)s_k = -f(x_k)$$

for *Newton step* $s_k$, then take as next iterate

$$x_{k+1} = x_k + s_k$$
Example: Newton’s Method

Use Newton’s method to solve nonlinear system

\[ \mathbf{f}(\mathbf{x}) = \begin{bmatrix} x_1 + 2x_2 - 2 \\ x_2^2 + 4x_2^2 - 4 \end{bmatrix} = 0 \]

Jacobian matrix is \( \mathbf{J}_f(\mathbf{x}) = \begin{bmatrix} 1 & 2 \\ 2x_1 & 8x_2 \end{bmatrix} \)

If we take \( \mathbf{x}_0 = \begin{bmatrix} 1 & 2 \end{bmatrix}^T \), then

\[ \mathbf{f}(\mathbf{x}_0) = \begin{bmatrix} 3 \\ 13 \end{bmatrix}, \quad \mathbf{J}_f(\mathbf{x}_0) = \begin{bmatrix} 1 & 2 \\ 2 & 16 \end{bmatrix} \]

Solving system \( \begin{bmatrix} 1 & 2 \\ 2 & 16 \end{bmatrix} \mathbf{s}_0 = \begin{bmatrix} -3 \\ -13 \end{bmatrix} \) gives \( \mathbf{s}_0 = \begin{bmatrix} -1.83 \\ -0.58 \end{bmatrix} \),

so \( \mathbf{x}_1 = \mathbf{x}_0 + \mathbf{s}_0 = \begin{bmatrix} -0.83 \\ 1.42 \end{bmatrix}^T \).
Example, continued

Evaluating at new point,

\[ f(x_1) = \begin{bmatrix} 0 \\ 4.72 \end{bmatrix}, \quad J_f(x_1) = \begin{bmatrix} 1 & 2 \\ -1.67 & 11.3 \end{bmatrix} \]

Solving system \[ \begin{bmatrix} 1 & 2 \\ -1.67 & 11.3 \end{bmatrix} s_1 = \begin{bmatrix} 0 \\ -4.72 \end{bmatrix} \] gives 

\[ s_1 = \begin{bmatrix} 0.64 \\ -0.32 \end{bmatrix}^T, \quad \text{so} \quad x_2 = x_1 + s_1 = \begin{bmatrix} -0.19 \\ 1.10 \end{bmatrix}^T \]

Evaluating at new point,

\[ f(x_2) = \begin{bmatrix} 0 \\ 0.83 \end{bmatrix}, \quad J_f(x_2) = \begin{bmatrix} 1 & 2 \\ -0.38 & 8.76 \end{bmatrix} \]

Iterations eventually convergence to solution \[ x^* = \begin{bmatrix} 0 \\ 1 \end{bmatrix}^T \]

< interactive example >
Convergence of Newton’s Method

Differentiating corresponding fixed-point operator

\[ g(x) = x - J(x)^{-1} f(x) \]

and evaluating at solution \( x^* \) gives

\[ G(x^*) = I - (J(x^*))^{-1} J(x^*) + \sum_{i=1}^{n} f_i(x^*) H_i(x^*) = O \]

where \( H_i(x) \) is component matrix of derivative of \( J(x)^{-1} \)

Convergence rate of Newton’s method for nonlinear systems is normally \textit{quadratic}, provided Jacobian matrix \( J(x^*) \) is nonsingular

But it must be started close enough to solution to converge
Cost per iteration of Newton’s method for dense problem in $n$ dimensions is substantial

Computing Jacobian matrix costs $n^2$ scalar function evaluations

Solving linear system costs $\mathcal{O}(n^3)$ operations
Secant updating methods reduce cost by

- Using function values at successive iterates to build approximate Jacobian and avoiding explicit evaluation of derivatives
- Updating factorization of approximate Jacobian rather than refactoring it each iteration

Most secant updating methods have superlinear but not quadratic convergence rate

Secant updating methods often cost less overall than Newton’s method because of lower cost per iteration
Broyden’s Method

*Broyden’s method* is typical secant updating method

Beginning with initial guess \( x_0 \) for solution and initial approximate Jacobian \( B_0 \), following steps are repeated until convergence

\[
x_0 = \text{initial guess} \\
B_0 = \text{initial Jacobian approximation} \\
\text{for } k = 0, 1, 2, \ldots \\
\text{Solve } B_k s_k = -f(x_k) \text{ for } s_k \\
x_{k+1} = x_k + s_k \\
y_k = f(x_{k+1}) - f(x_k) \\
B_{k+1} = B_k + ((y_k - B_k s_k) s_k^T) / (s_k^T s_k)
\]

end
Broyden’s Method, continued

Motivation for formula for $B_{k+1}$ is to make least change to $B_k$ subject to satisfying *secant equation*

$$B_{k+1}(x_{k+1} - x_k) = f(x_{k+1}) - f(x_k)$$

In practice, factorization of $B_k$ is updated instead of updating $B_k$ directly, so total cost per iteration is only $O(n^2)$
Example: Broyden’s Method

Use Broyden’s method to solve nonlinear system

\[ f(x) = \begin{bmatrix} x_1 + 2x_2 - 2 \\ x_1^2 + 4x_2^2 - 4 \end{bmatrix} = 0 \]

If \( x_0 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}^T \), then \( f(x_0) = \begin{bmatrix} 3 \\ 13 \end{bmatrix}^T \), and we choose

\[ B_0 = J_f(x_0) = \begin{bmatrix} 1 & 2 \\ 2 & 16 \end{bmatrix} \]

Solving system

\[ \begin{bmatrix} 1 & 2 \\ 2 & 16 \end{bmatrix} s_0 = \begin{bmatrix} -3 \\ -13 \end{bmatrix} \]

gives \( s_0 = \begin{bmatrix} -1.83 \\ -0.58 \end{bmatrix} \), so \( x_1 = x_0 + s_0 = \begin{bmatrix} -0.83 \\ 1.42 \end{bmatrix} \)
Example, continued

Evaluating at new point $x_1$ gives $f(x_1) = \begin{bmatrix} 0 \\ 4.72 \end{bmatrix}$, so

$y_0 = f(x_1) - f(x_0) = \begin{bmatrix} -3 \\ -8.28 \end{bmatrix}$

From updating formula, we obtain

$B_1 = \begin{bmatrix} 1 & 2 \\ 2 & 16 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ -2.34 & -0.74 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ -0.34 & 15.3 \end{bmatrix}$

Solving system

$\begin{bmatrix} 1 & 2 \\ -0.34 & 15.3 \end{bmatrix} s_1 = \begin{bmatrix} 0 \\ -4.72 \end{bmatrix}$

gives $s_1 = \begin{bmatrix} 0.59 \\ -0.30 \end{bmatrix}$, so $x_2 = x_1 + s_1 = \begin{bmatrix} -0.24 \\ 1.120 \end{bmatrix}$
Example, continued

Evaluating at new point $x_2$ gives $f(x_2) = \begin{bmatrix} 0 \\ 1.08 \end{bmatrix}$, so

$$y_1 = f(x_2) - f(x_1) = \begin{bmatrix} 0 \\ -3.64 \end{bmatrix}$$

From updating formula, we obtain

$$B_2 = \begin{bmatrix} 1 & 2 \\ -0.34 & 15.3 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 1.46 & -0.73 \end{bmatrix} = \begin{bmatrix} 1.12 & 2 \\ 1.12 & 14.5 \end{bmatrix}$$

Iterations continue until convergence to solution $x^* = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$
Robust Newton-Like Methods

Newton’s method and its variants may fail to converge when started far from solution.

Safeguards can enlarge region of convergence of Newton-like methods.

Simplest precaution is damped Newton method, in which new iterate is

\[ x_{k+1} = x_k + \alpha_k s_k \]

where \( s_k \) is Newton (or Newton-like) step and \( \alpha_k \) is scalar parameter chosen to ensure progress toward solution.

Parameter \( \alpha_k \) reduces Newton step when it is too large, but \( \alpha_k = 1 \) suffices near solution and still yields fast asymptotic convergence rate.
Another approach is to maintain estimate of trust region where Taylor series approximation, upon which Newton’s method is based, is sufficiently accurate for resulting computed step to be reliable.

Adjusting size of trust region to constrain step size when necessary usually enables progress toward solution even starting far away, yet still permits rapid converge once near solution.

Unlike damped Newton method, trust region method may modify direction as well as length of Newton step.