# Numerical Modeling in Biomedical Systems

BME 125:305

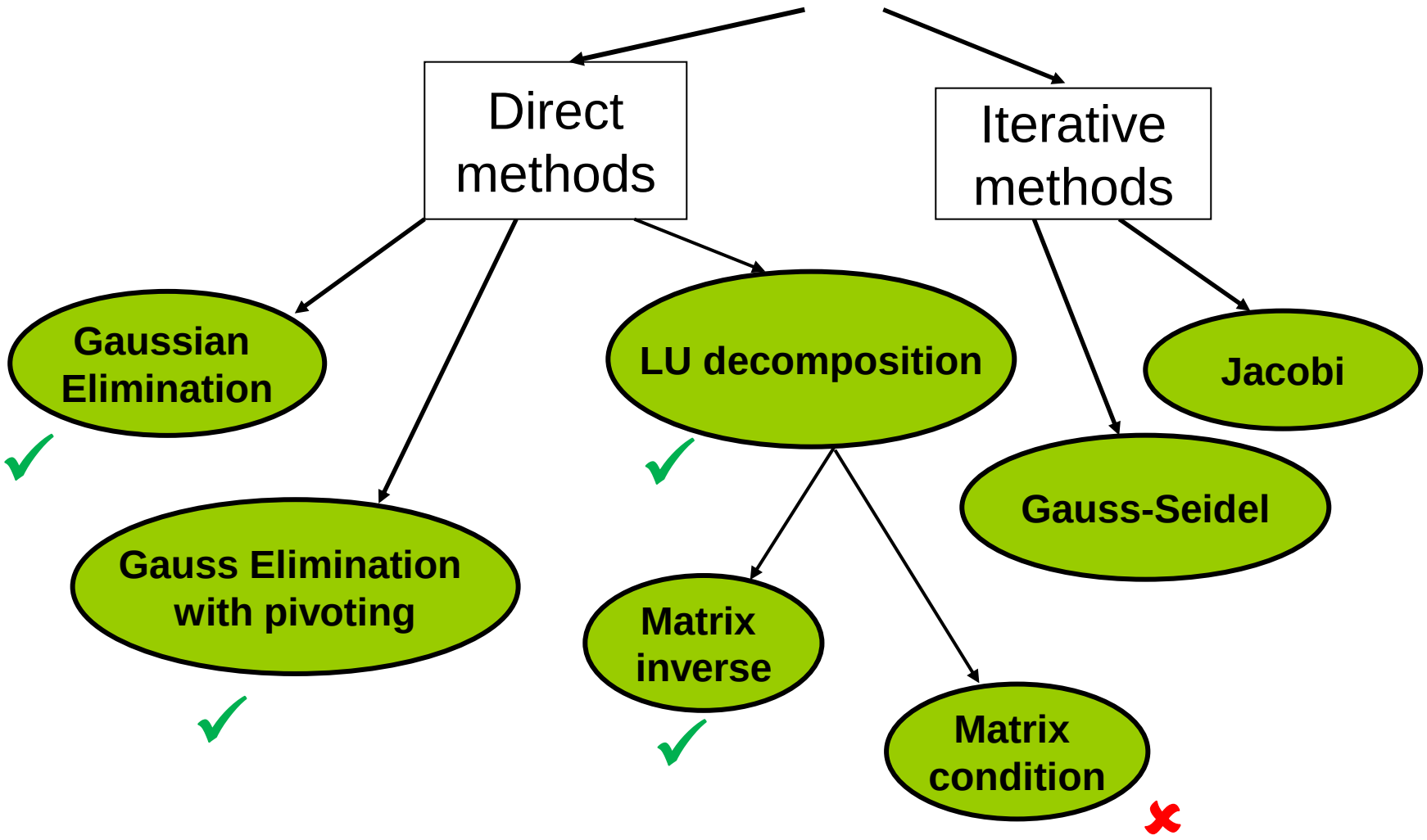
Lecture 6

9/21/17

# Methods for Solving Linear Simultaneous Equations

$$a_{11}x_1 + a_{12}x_2 = b_1$$

$$a_{21}x_1 + a_{22}x_2 = b_2$$



# Iterative Methods (Dunn 4.5)

$$
\begin{aligned}
Q_{11}c_1 + Q_{12}c_2 + \cdots Q_{1n}c_n &= b_1 \\
Q_{21}c_1 + Q_{22}c_2 + \cdots Q_{2n}c_n &= b_2 \\
&\vdots \\
Q_{n1}c_1 + Q_{n2}c_2 + \cdots Q_{nn}c_n &= b_n
\end{aligned}
$$

- Direct methods solve the set of equations for the unknown variables ($c$'s here) by manipulating the equations (usually in matrix form), eliminating and substituting variables until we isolate the $c$'s
  - Gaussian Elimination
  - LU decomposition
- For large systems of equations, we will need many mathematical operations to reach a solution, which can be subject to round-off errors

- Iterative methods solve the set of equations by making an initial guess for the unknown variables ($c$'s here), then iteratively adjusting these estimates *until a desired level of accuracy is reached* – i.e. not limited by round-off error
  - Gauss-Seidel
  - Jacobi method

# Gauss-Seidel (Dunn 4.5.2)

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1$$
$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 = b_2$$
$$a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = b_3$$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

**[A][x] = [b]**

- If we have a set of $n$ equations, we can rearrange equation 1 in terms of $x_1$, rearrange equation 2 in terms of $x_2$, and so on:

With 3 equations:

$$x_1 = \frac{b_1 - a_{12}x_2 - a_{13}x_3}{a_{11}} \qquad x_2 = \frac{b_2 - a_{21}x_1 - a_{23}x_3}{a_{22}} \qquad x_3 = \frac{b_3 - a_{31}x_1 - a_{32}x_2}{a_{33}}$$

- We need to start the iterative solution process by choosing initial guesses for the solutions $x_1$, $x_2$, $x_3$:

  $\rightarrow$ A simple initial guess is $x_1$, $x_2$, $x_3$ = 0

Initial guess: $x_1, x_2, x_3 = 0$

$$x_1 = \frac{b_1 - a_{12}x_2 - a_{13}x_3}{a_{11}} \xrightarrow{x_2,\, x_3 = 0} x_1 = \frac{b_1}{a_{11}} \qquad x_3 = 0$$

- Use our *new guess* for $x_1$ and the *current guess* for $x_3$ to get a *new guess* for $x_2$ :

$$x_2 = \frac{b_2 - a_{21}x_1 - a_{23}x_3}{a_{22}} \longrightarrow x_2 = \frac{b_2 - a_{21}(b_1 / a_{11})}{a_{22}}$$

- Use our *new guess* for $x_2$ and the *current guess* for $x_1$ to get a *new guess* for $x_3$:

$$x_3 = \frac{b_3 - a_{31}x_1 - a_{32}x_2}{a_{33}}$$

$$x_3 = \frac{b_3 - a_{31}(b_1 / a_{11}) - a_{32}\left(b_2 - a_{21}(b_1 / a_{11})\right) / a_{22}}{a_{33}}$$

$\rightarrow$ We can now have <u>new</u> guesses for each of $x_1, x_2, x_3$ , in terms of known parameters ($a$'s and $b$'s)

$\rightarrow$ We can return to use the new guesses for $x_2, x_3$ to obtain a new (new) guess for $x_1$

Initial guess: $x_1, x_2, x_3 = 0$

First iteration (new $x_1, x_2, x_3$)

$$x_1 = \left(b_1 - a_{12}x_2 - a_{13}x_3\right)/a_{11}$$

$$x_2 = \left(b_2 - a_{21}x_1 - a_{23}x_3\right)/a_{22}$$

$$x_3 = \left(b_3 - a_{31}x_1 - a_{32}x_2\right)/a_{33}$$

Second iteration (new $x_1, x_2, x_3$)

$$x_1 = \left(b_1 - a_{12}x_2 - a_{13}x_3\right)/a_{11}$$

$$x_2 = \left(b_2 - a_{21}x_1 - a_{23}x_3\right)/a_{22}$$

$$x_3 = \left(b_3 - a_{31}x_1 - a_{32}x_2\right)/a_{33}$$

$\vdots$

How do we decide when to stop?

- Remember back to Lecture 3:

$$\varepsilon_a = \frac{\text{Current Approximation} - \text{Previous Approximation}}{\text{Current Approximation}}$$

- For **Gauss-Seidel** iteration, our approximation for $x_i$ after iteration $j$ can be written as $x_i^j$

- The approximate error on our values for $x_i$ after iteration $j$ is then $\varepsilon_{a,i}$ where $j$-1 is the previous iteration:

$$\left|\varepsilon_{a,i}\right| = \left|\frac{x_i^j - x_i^{j-1}}{x_i^j}\right| \times 100\%$$

- As we have done previously, the iterative process can continue until the approximate error falls below a pre-defined **stopping criterion**, $\varepsilon_s$

$$\left|\varepsilon_a\right| < \varepsilon_s$$

# Example

- Use the **Gauss-Seidel** method to solve the same set of equations used in L6 (LU decomposition):

$$3x_1 - 0.1x_2 - 0.2x_3 = 7.85$$

$$0.1x_1 + 7x_2 - 0.3x_3 = -19.3$$

$$0.3x_1 - 0.2x_2 + 10x_3 = 71.4$$

The true solution is:
$x_1 = 3$, $x_2 = -2.5$, $x_3 = 7$

1) Rearrange the equations in terms of the unknown on the diagonal:

$$x_1 = \frac{7.85 + 0.1x_2 + 0.2x_3}{3} \qquad x_2 = \frac{-19.3 - 0.1x_1 + 0.3x_3}{7} \qquad x_3 = \frac{71.4 - 0.3x_1 + 0.2x_2}{10}$$

2) Using initial guesses of $x_2$, $x_3$ = 0, generate a new guess for $x_1$:

$$x_1 = \frac{7.85 + 0 + 0}{3} = 2.616667$$

3) Use this new guess for $x_1$, with $x_3$ = 0, to generate a new guess for $x_2$:

$$x_2 = \frac{-19.3 - 0.1(2.616667) + 0}{7} = -2.794524$$

4) Similarly for $x_3$:

$$x_3 = 7.005610$$
$$x_2 = -2.794524$$
$$x_1 = 2.616667$$

After 1 iteration

The true solution is:
$x_1 = 3$, $x_2 = -2.5$, $x_3 = 7$

Our guesses after iteration 1 are already *converging* towards the correct solution:

| Iteration | Variable | Estimated value | True value | True relative error, $\lvert\varepsilon_t\rvert$ |
|---|---|---|---|---|
| 0 | $x_1$ | 0 | 3 | 100% |
| | $x_2$ | 0 | -2.5 | 100% |
| | $x_3$ | 0 | 7 | 100% |
| 1 | $x_1$ | 2.616667 | 3 | 12.78% |
| | $x_2$ | -2.794524 | -2.5 | 11.78% |
| | $x_3$ | 7.005610 | 7 | 0.08% |
| 2 | $x_1$ | 2.990557 | 3 | 0.31% |
| | $x_2$ | -2.499625 | -2.5 | 0.015% |
| | $x_3$ | 7.000291 | 7 | 0.0042% |

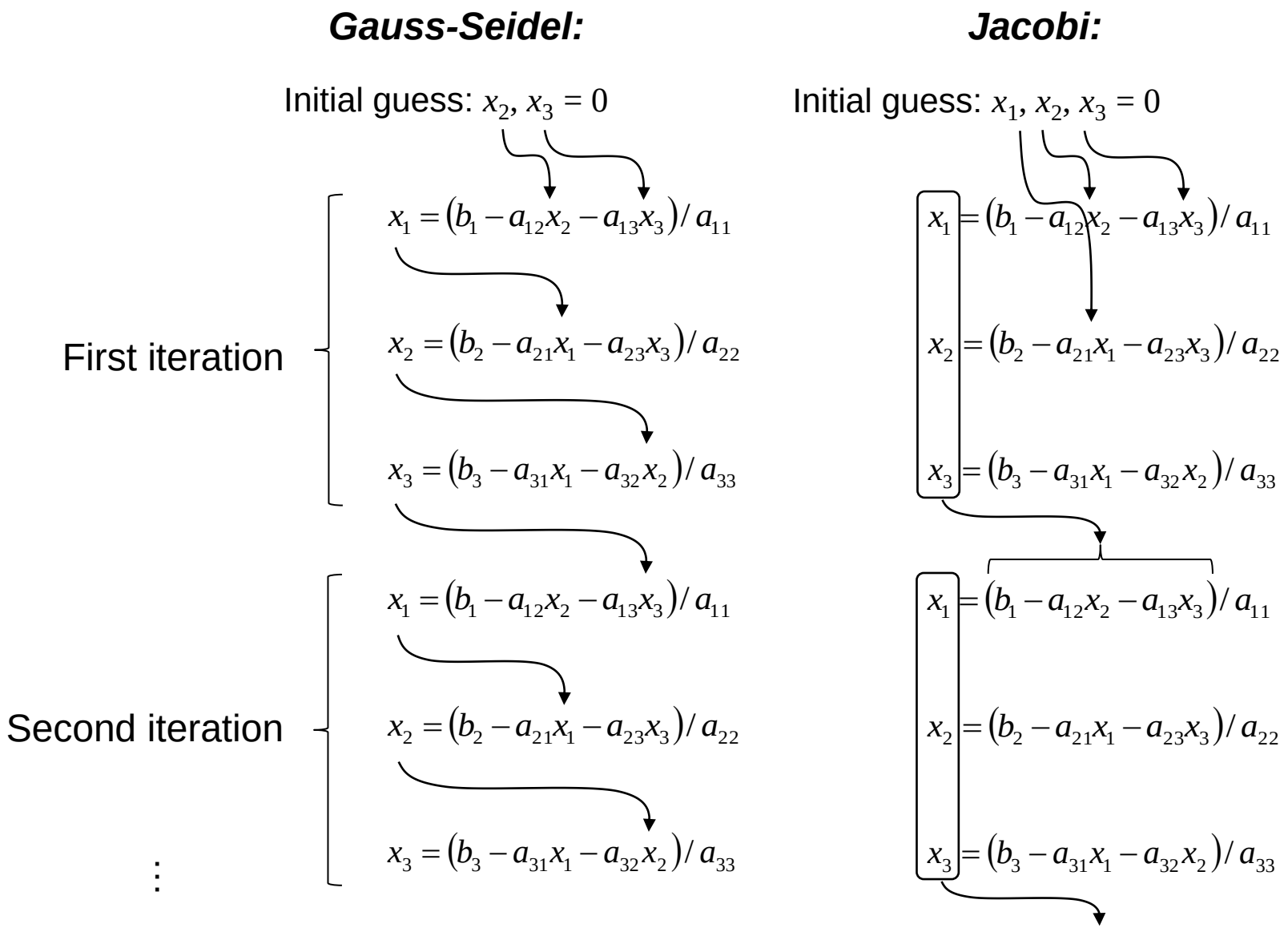
- But as before, we usually don't know the true values we are looking for. We can use our approximate relative error formula to estimate the error

$$\left|\varepsilon_{a,i}\right| = \left|\frac{x_i^j - x_i^{j-1}}{x_i^j}\right| \times 100\%$$

| Iteration ($j$) | Variable ($x_i$) | Estimated value | True relative error, $\lvert\varepsilon_t\rvert$ | Approximate relative error, $\lvert\varepsilon_{a,i}\rvert$ |
|---|---|---|---|---|
| 0 | $x_1$ | 0 | 100% | - |
| | $x_2$ | 0 | 100% | - |
| | $x_3$ | 0 | 100% | - |
| 1 | $x_1$ | 2.616667 | 12.78% | (2.616667 - 0) / 2.616667 = 100% |
| | $x_2$ | -2.794524 | 11.78% | (-2.794524 - 0) / -2.794524 = 100% |
| | $x_3$ | 7.005610 | 0.08% | (7.005610 - 0) / 7.005610 = 100% |
| 2 | $x_1$ | 2.990557 | 0.31% | (2.990557 - 2.616667) / 2.990557 = 12.5% |
| | $x_2$ | -2.499625 | 0.015% | (-2.499625 +2.794524)/-2.499625 = 11.8% |
| | $x_3$ | 7.000291 | 0.0042% | (7.000291-7.005610)/7.000291 = 0.076% |

- Note that $|\varepsilon_a|$ gives a conservative estimate of the error (an overestimate of the true error). This means that if we ensure that $|\varepsilon_a| < \varepsilon_s$, then the true error in our result is probably much lower than $\varepsilon_s$

# Comparison of Gauss-Seidel and Jacobi methods




- Use the new guesses in current iteration (G-S) or wait to use in next iteration (Jacobi)

# Comparison of Gauss-Seidel and Jacobi methods

- Use the new guesses in current iteration (G-S) or wait to use in next iteration (Jacobi)
- ***If*** the solution is converging, G-S will converge faster than the Jacobi method towards the true result

- We can see whether our system of equations will converge when using the Gauss-Seidel algorithm by applying the following test:

Return to our original equations and rearrange:

$$a_{11}x_1 + a_{12}x_2 = b_1 \longrightarrow x_1 = u(x_1, x_2) = \frac{b_1}{a_{11}} - \frac{a_{12}}{a_{11}}x_2$$

$$a_{21}x_1 + a_{22}x_2 = b_2 \longrightarrow x_2 = v(x_1, x_2) = \frac{b_2}{a_{22}} - \frac{a_{21}}{a_{22}}x_1$$

Test for convergence: $\left|\frac{\partial u}{\partial x_1}\right| + \left|\frac{\partial u}{\partial x_2}\right| < 1$ and $\left|\frac{\partial v}{\partial x_1}\right| + \left|\frac{\partial v}{\partial x_2}\right| < 1$

Here, $\left|\frac{\partial u}{\partial x_1}\right| = 0$ $\left|\frac{\partial u}{\partial x_2}\right| = -\frac{a_{12}}{a_{11}}$ and $\left|\frac{\partial v}{\partial x_1}\right| = -\frac{a_{21}}{a_{22}}$ $\left|\frac{\partial v}{\partial x_2}\right| = 0$

$$\left|\frac{a_{12}}{a_{11}}\right| < 1 \qquad \left|\frac{a_{21}}{a_{22}}\right| < 1$$

$$|a_{12}| < |a_{11}| \qquad |a_{21}| < |a_{22}|$$

$$|a_{12}| < |a_{11}| \qquad\qquad |a_{21}| < |a_{22}|$$

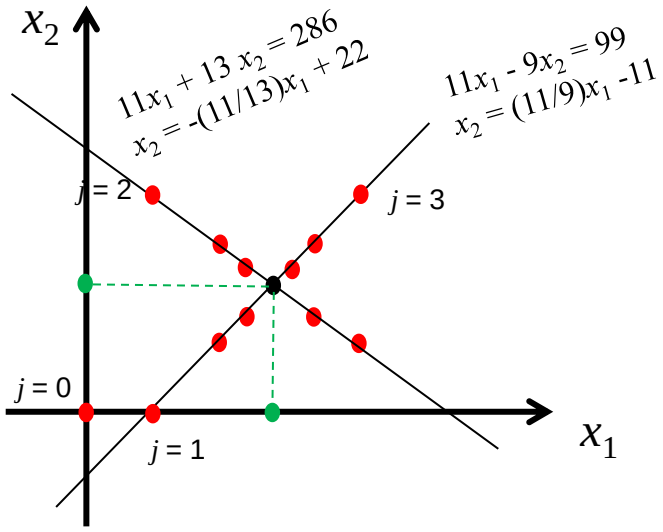
- What is this telling us?
  - The absolute value of the diagonal elements must be greater than that of the off-diagonal elements in each row, in order to ensure convergence

# Convergence

- Sometimes the solution will not converge towards the true result. Consider the graphical interpretation of solving a pair of simultaneous equations:

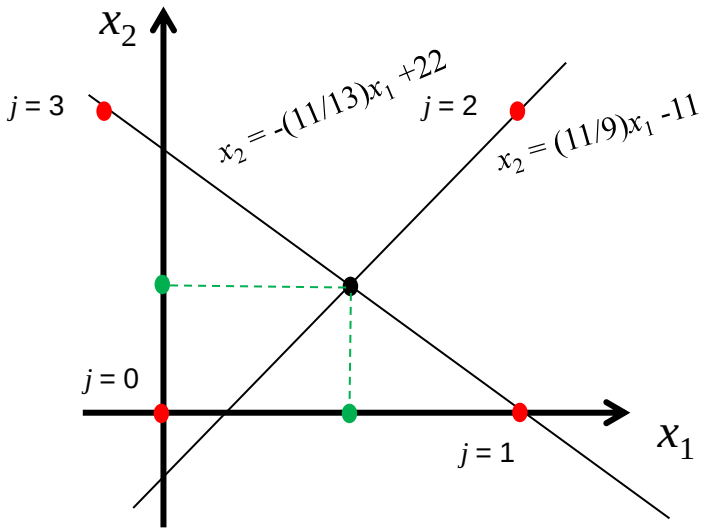
$$11x_1 - 9x_2 = 99$$

$$11x_1 + 13x_2 = 286$$




True result

Iterative result




True result

Iterative result

Gauss-Seidel (1):

$11x_1 - 9x_2 = 99 \quad \rightarrow x_1 = (99 + 9x_2)/11$

$11x_1 + 13x_2 = 286 \quad \rightarrow x_2 = (286 - 11x_1)/13$

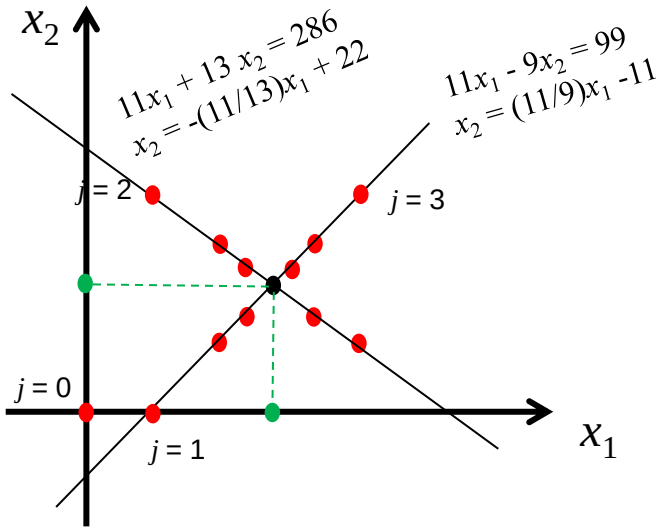
Gauss-Seidel (2):

$11x_1 + 13x_2 = 286 \quad \rightarrow x_1 = (286 - 13x_2)/11$

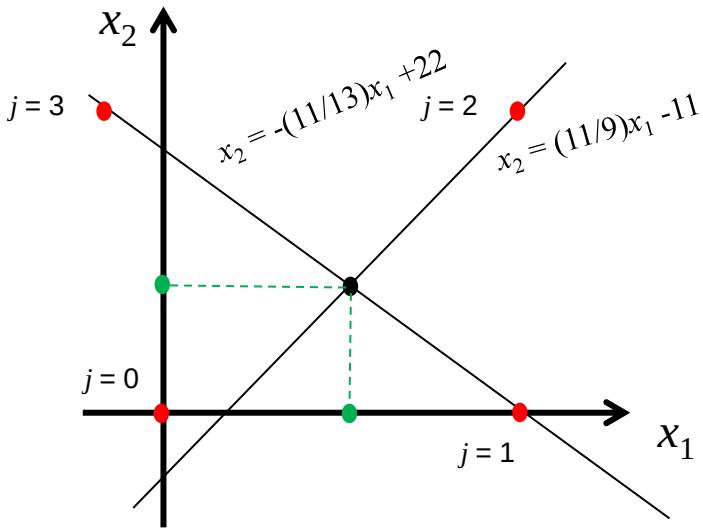
$11x_1 - 9x_2 = 99 \quad \rightarrow x_2 = (11x_1 - 99)/9$

$|a_{12}| < |a_{11}|$ $|a_{21}| < |a_{22}|$

- Remember:
  - The absolute value of the diagonal elements must be greater than that of the off diagonal elements in each row, in order to ensure convergence




- True result
- Iterative result




- True result
- Iterative result

**Converges**

$11x_1 - 9x_2 = 99 \rightarrow x_1 = (99 + 9x_2)/11$

$11x_1 + 13x_2 = 286 \rightarrow x_2 = (286 - 11x_1)/13$

**Does not converge**

$11x_1 + 13x_2 = 286 \rightarrow x_1 = (286 - 13x_2)/11$

$11x_1 - 9x_2 = 99 \rightarrow x_2 = (11x_1 - 99)/9$

# Example – Iterative solutions in Matlab

- Matlab has 11 functions for iteratively solving systems of linear simultaneous equations (choose carefully):

| Function | Method |
|---|---|
| bicg | Biconjugate gradient |
| bicgstab | Biconjugate gradient stabilized |
| bicgstabl | Biconjugate gradient stabilized (l) |
| cgs | Conjugate gradient squared |
| gmres | Generalized minimum residual |
| lsqr | Least squares |
| minres | Minimum residual |
| pcg | Preconditioned conjugate gradient |
| qmr | Quasiminimal residual |
| symmlq | Symmetric LQ |
| tfqmr | Transpose-free quasiminimal residual |

Example (same as slide 8):

```
clear all

A = [3 -0.1 -0.2; 0.1 7 -0.3; 0.3 -0.2 10]
b = [7.85 -19.3 71.4]'

tol = 1e-10;
maxit = 10;

x = symmlq(A,b,tol,maxit)
```

tolerance

max # iterations

```
symmlq converged at iteration 10 to a solution with relative residual 3.5e-11.

x =

    3.0000
   -2.5000
    7.0000
```

# Summary:

- Iterative methods can have advantages over direct methods in some situations
  - For solving large numbers of equations
  - For solving sparse equations

  } We will use later for PDE's

- The Gauss-Seidel and Jacobi methods are common techniques for iterative solution of linear simultaneous equations
- Convergence of iterative methods is not guaranteed, but can be checked by ensuring that the diagonal matrix elements are greater than the off-diagonals in each row:

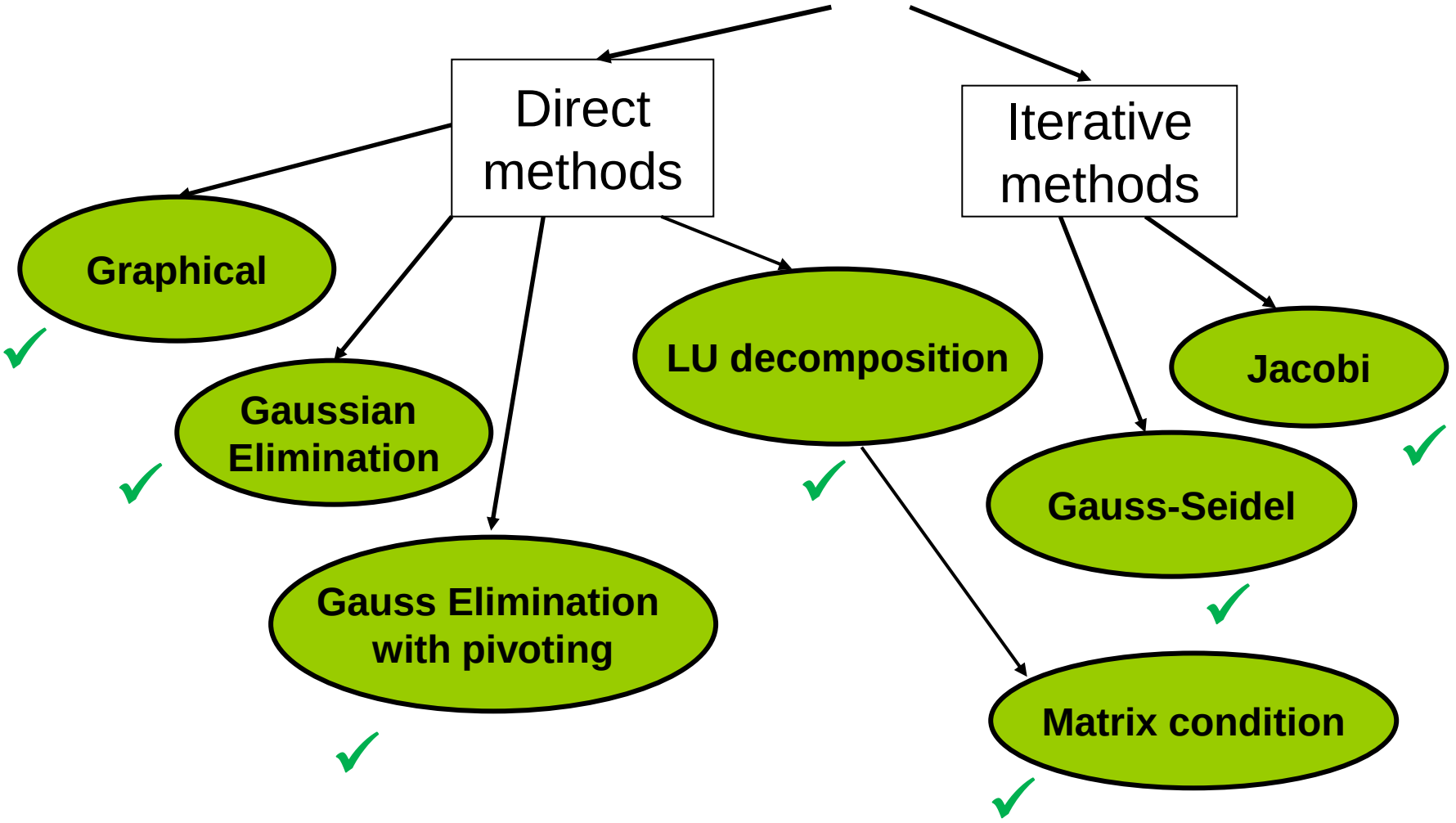
$$|a_{ii}| > \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{ij}|$$

| **Method** | | **Stability / Convergence** | **Precision** | **Comments** |
|---|---|---|---|---|
| Direct | Gaussian elimination / Gauss Jordan | - | Affected by round-off error | Partial pivoting is required to avoid divide-by-zero and large round off errors |
| | LU decomposition | - | Affected by round-off error | Efficient approach for evaluating multiple right-side vectors, computing inverse $A^{-1}$ |
| Iterative | Gauss-Seidel / Jacobi | Must be diagonally dominant | Excellent | More efficient than direct methods for large systems of sparse matrices |

# Methods for Solving Linear Simultaneous Equations

$$a_{11}x_1 + a_{12}x_2 = b_1$$

$$a_{21}x_1 + a_{22}x_2 = b_2$$



# Schedule

| Week | Date | Lecture # | Topic | Book Chapter | Homework / Exams |
|---|---|---|---|---|---|
| 1 | 5-Sep | Lecture 1 | Introduction, course information, mathematical representation of biological systems | 2 | |
| | 7-Sep | Lecture 2 | Digital storage of numbers, round off and truncation errors, Taylor series expansions | 3 | |
| 2 | 12-Sep | Lecture 3 | Linear systems - Direct methods: Gaussian elimination, pivoting, Gauss-Jordan methods | 4 | HW1 set |
| | 14-Sep | Lecture 4 | Linear systems - Direct methods: LU decomposition | 4 | |
| 3 | 19-Sep | Lecture 5 | Linear systems - Iterative methods: Jacobi, Gauss-Seidel methods | 4 | HW1 due |
| | 21-Sep | Lecture 6 | Nonlinear systems - Bracketing methods: Graphical, Bisection, False-Position methods | 5 | HW2 set |
| 4 | 26-Sep | Lecture 7 | Nonlinear systems - Open methods: Fixed-Point Iteration | 5 | |
| | 28-Sep | Lecture 8 | Nonlinear systems - Open methods: Newton-Raphson, Secant, Brent's, Newton's methods | 5 | |
| 5 | 3-Oct | Lecture 9 | Numerical differentiation - Finite differences | 6 | HW2 due |
| | 5-Oct | | Exam 1 - Lectures 2-8 | | Exam 1 |

- Homework 2 will cover linear and nonlinear systems – Due on 10/3

# Linear vs Nonlinear Equations

- In L4-7 we saw that many problems in BME can be described by systems of coupled linear equations, and we studied several approaches for solving these equations:

Linear equations:

$$a_{11}x_1 + a_{12}x_2 = b_1$$

$$a_{21}x_1 + a_{22}x_2 = b_2$$

- Some problems are described by nonlinear equations. We will need to develop methods to solve these equations too.

Nonlinear equations:

$$\ln(x) + x = 4.6$$

$$\frac{1}{\sqrt{x}} = \sin\left(\sqrt{x} + 0.01\right)$$

} Transcendental eg. trig functions, exponentials, logarithms…

$$f(x) = a_0 + a_1x + a_2x^2 + \ldots + a_nx^n$$

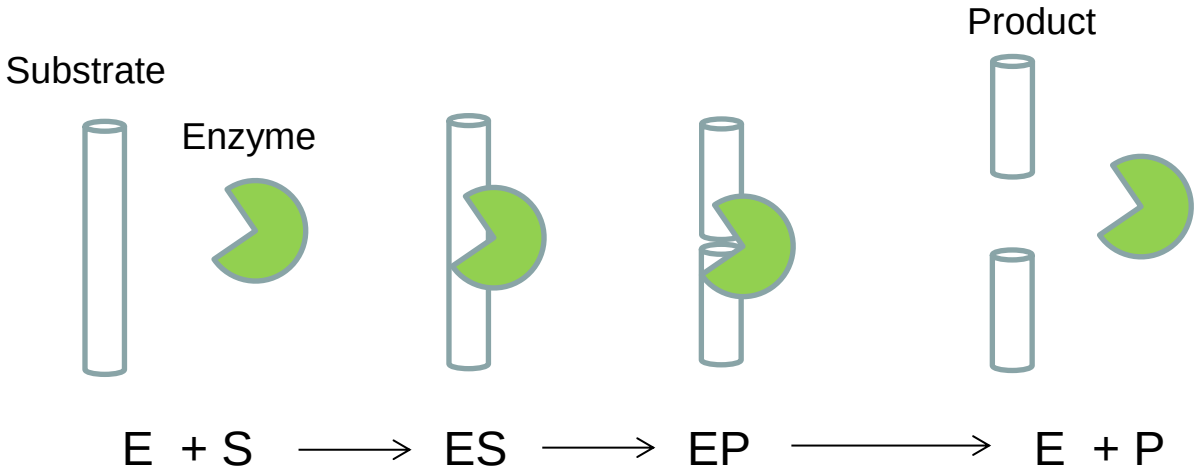
} Algebraic eg. polynomials…

→ Problem: Usually we cannot rearrange nonlinear equations in the form $x$ = …

# Examples of BME systems which lead to nonlinear equations (Dunn 5.3)

Enzyme kinetics:

$$E + S \underset{k_{-1}}{\overset{k_1}{\rightleftharpoons}} ES \xrightarrow{k_{cat}} E + P$$




- Michaelis-Menten equation:
  - Expresses the reaction rate, $r_s$ in terms of substrate concentration $[s]$:

$$r_s = \frac{V_{\max}[s]}{K_m + [s]} \qquad \text{where} \qquad K_m = \frac{k_{-1} + k_{cat}}{k_1}$$

$\rightarrow$ We can rearrange and integrate with respect to $r_s$, to yield an expression for substrate concentration $[s]$ as a function of time, $t$:

$$K_m \ln\left(\frac{[s_0]}{[s]}\right) + ([s_0] - [s]) = V_{\max} t$$

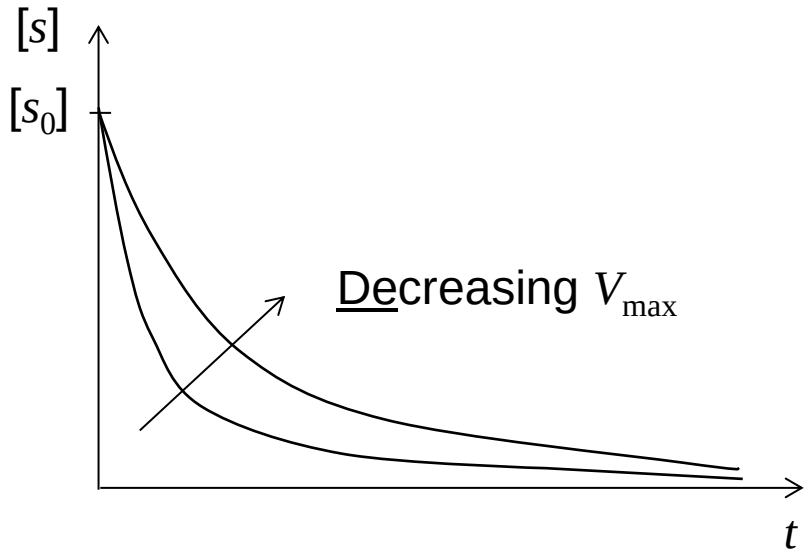
# Nonlinear Equations – Example from Dunn 5.3.1

- How do we *solve* this expression, to determine how [$s$] varies over time?

$$K_m \ln\left(\frac{[s_0]}{[s]}\right) + ([s_0] - [s]) = V_{\max} t \qquad \text{(Michaelis-Menten)}$$

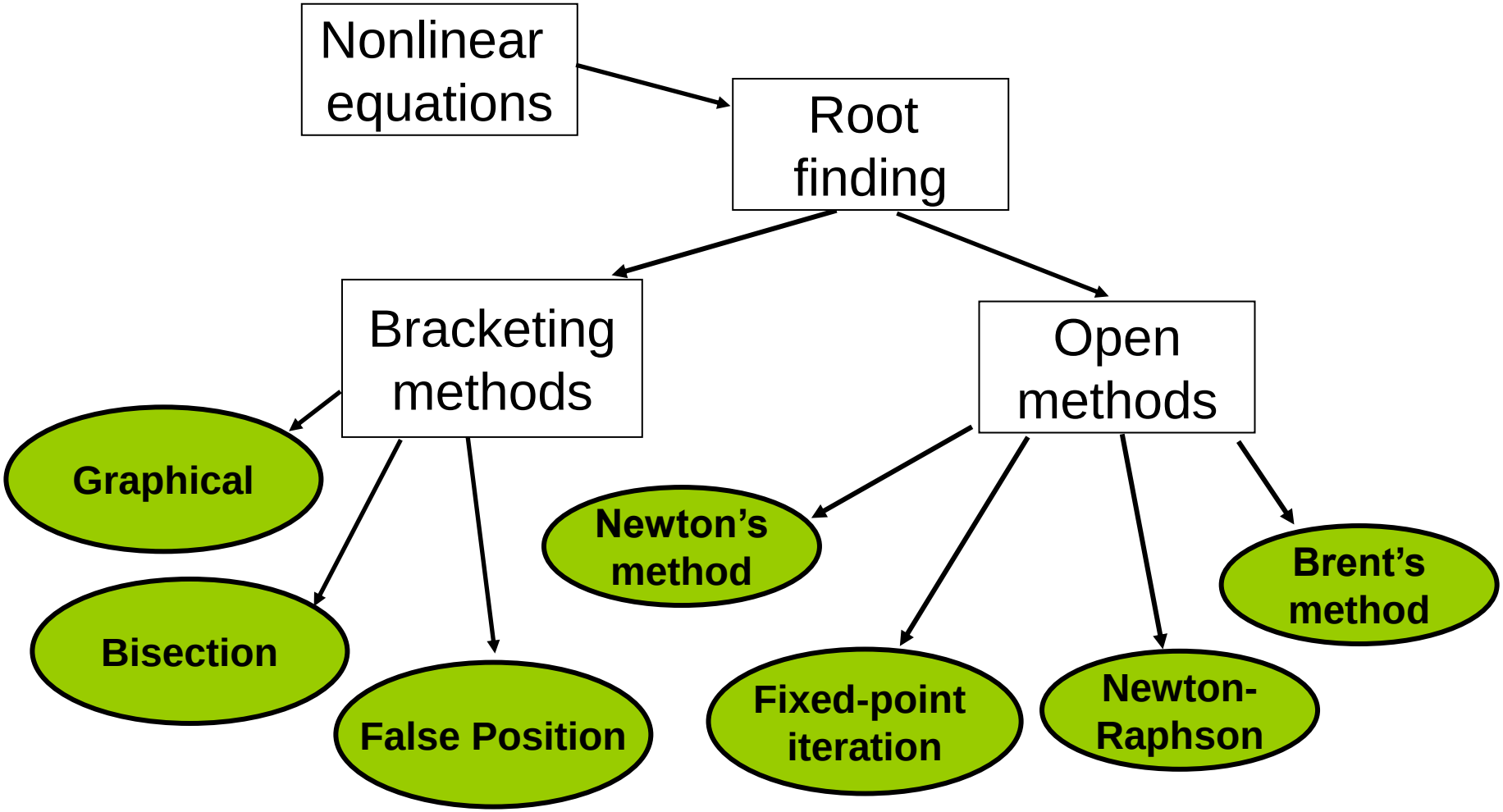
- We cannot rearrange to obtain an analytical expression in the form [$s$] = …

  → We have to solve numerically:



# Nonlinear Systems – Dunn Chapter 5

Today: Bracketing Methods (graphical and bisection)



# Root Finding

- How would you solve this equation for $x$ ? $f(x) = ax^2 + bx + c$

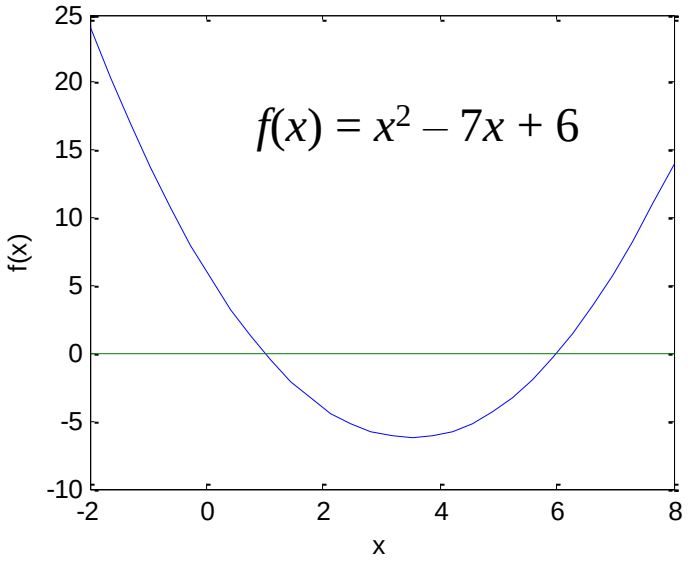
  Can you rearrange in the form $x$ = ... ?

$$x_1, x_2 = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

The formula gives us the "roots" of the quadratic equation

- I.e. the values of the variable ($x$) which satisfy $f(x) = 0$
- If we can get our quadratic in the form $f(x) = 0$, then we *can* solve for $x$

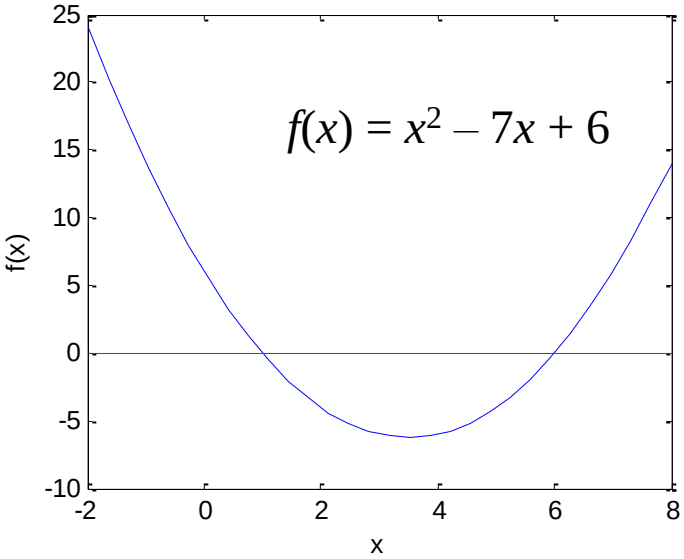
Example:



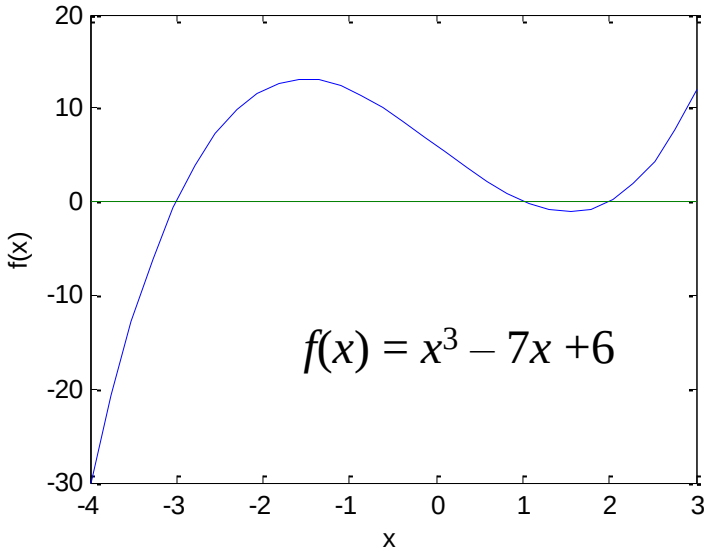

This equation has roots at $x$ = 1 and 6

How do we find the root(s) of nonlinear equations that are more complex than a quadratic?

- Plot the function and see where it crosses zero?




Roots at $x$ = 1, 6




Roots at $x$ = -3, 2, 1

- Graphical methods are not very accurate

- Guess a value for $x$, evaluate whether $f(x) = 0$?
  - This will take a long time

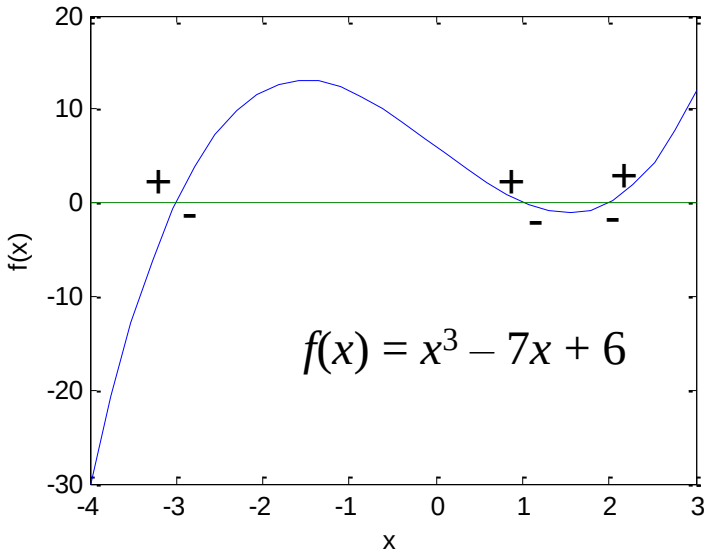
- Better to develop <u>systematic methods</u> to search for $f(x) = 0$

# Bracketing Methods

We want to find the roots of $f(x)$, i.e. the values of $x$ where $f(x) = 0$

Where to start looking?

Bracketing methods use the fact that the value of $f(x)$ *changes sign* at each root



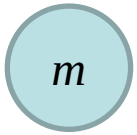

Strategy: Find two values of $x$, $(x_1,x_2)$ giving *positive* and *negative* values for $f(x)$

The root must lie somewhere between these values of $x$

Narrow the distance between $(x_1,x_2)$ to close in on the true root

# Example - Falling object from Lecture 2

$F_{up}$ = drag force



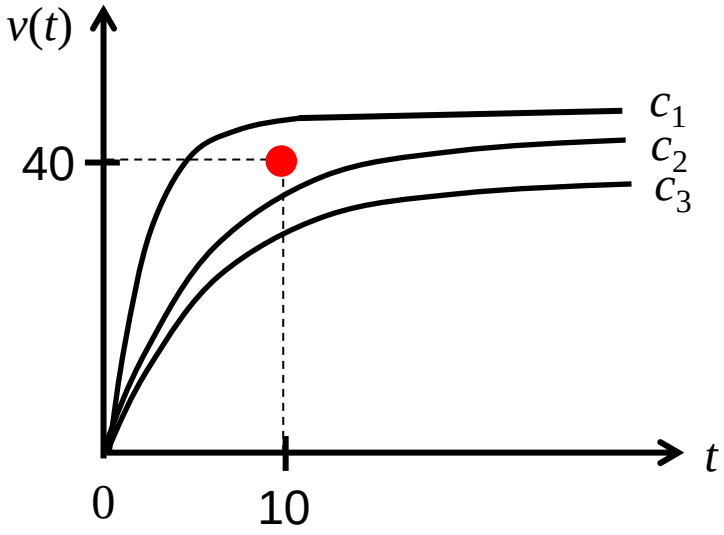

mass, $m$

$F_{down}$ = gravitational force

Using Newton's 2nd law leads to an analytical expression for velocity $v$ as a function of time $t$ and system parameters mass ($m$) and drag coefficient ($c$):

$$v(t) = \frac{gm}{c}\left(1 - e^{-(c/m)t}\right)$$

$\rightarrow$ Given $g,m,c,t$ we can calculate $v$




If an object has mass $m$ = 68.1 kg, what drag coefficient ($c$) is necessary in order to reach a velocity of $v$ = 40 m/s at time $t$ = 10 seconds?

Problem: We can't rearrange the expression in the form $c = f(g,m,v,t)$

$\rightarrow$ We need to find the value(s) of $c$ which satisfies: $40 = \frac{(9.8)(68.1)}{c}\left(1 - e^{-(c/68.1)40}\right)$

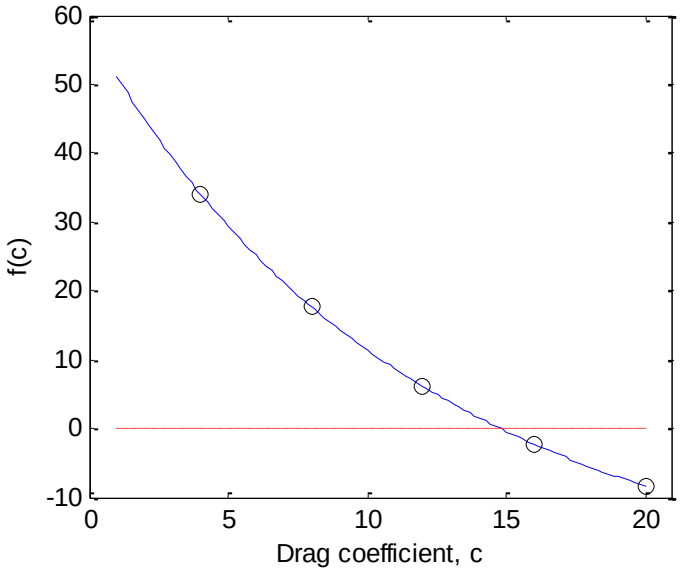
I.e. the value(s) of $c$ which satisfies: $0 = \frac{9.8(68.1)}{c}\left(1 - e^{-(c/68.1)10}\right) - 40$

- We need to find the root(s) of $f(c) = \frac{9.8(68.1)}{c}\left(1 - e^{-(c/68.1)10}\right) - 40$

  I.e. the value(s) of $c$ which lead to $f(c) = 0$

- We can start to make some guesses for $c$, and see what we get for $f(c)$:

| Guess for $c$ | $f(c)$ |
|---|---|
| 4 | 34.115 |
| 8 | 17.653 |
| 12 | 6.067 |
| 16 | -2.269 |
| 20 | -8.401 |




$c \approx 14.75$ looks close to zero. $v(t) = \frac{gm}{c}\left(1 - e^{-(c/m)t}\right)$ gives $v$ = 40.059 m/s when $c$ = 14.75

$\rightarrow$ Graphical methods are not very accurate, but they can provide us with useful initial guesses to use with other methods

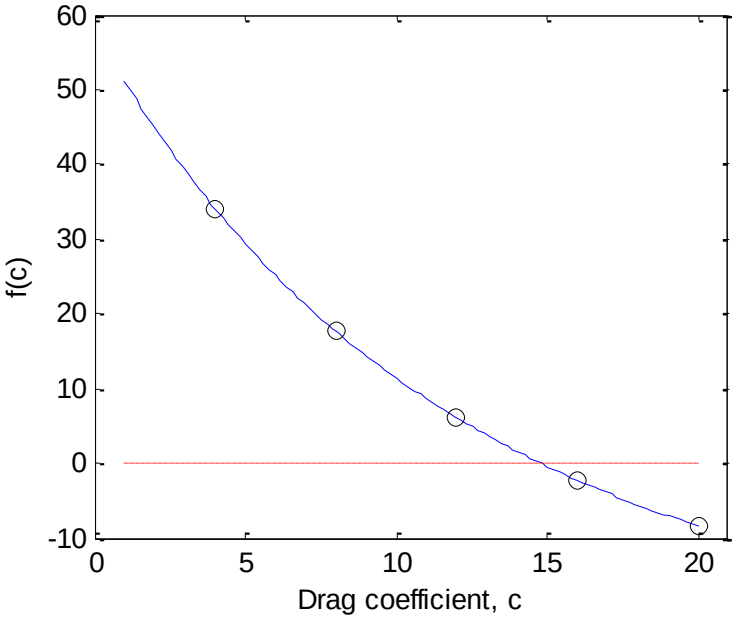
# Bisection Method

- In general:

If $f(x)$ is *real* and *continuous* in the range between $x_l$ and $x_u$, and

$$f(x_l)\, f(x_u) < 0,$$

then there is at least one root in the range $x_l$ to $x_u$




Step 1: Choose lower ($x_l$) and upper ($x_u$) guesses for the location of the root.

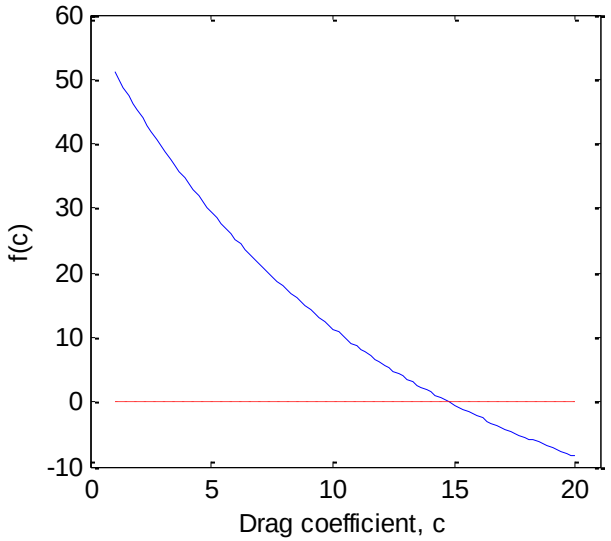
Step 2: If $f(x_l)\, f(x_u) < 0$, then the root is estimated to lie at $x_r = (x_l + x_u) / 2$

Step 3: If $f(x_l)\, f(x_r) < 0$, then the true root must lie between $x_l$ and $x_r$
If $f(x_l)\, f(x_r) > 0$, then the true root must lie between $x_r$ and $x_u$

Return to step 2 with the new guesses for the interval, repeat until the interval width is sufficiently small (i.e. the estimate for the root is within an acceptable tolerance)

# Example - Bisection Method

$$f(c)=\frac{9.8(68.1)}{c}\left(1-e^{-(c/68.1)10}\right)-40$$




- Repeat the previous (falling mass) example using the bisection method:

Step 1: Make initial guesses for $x_l$ and $x_u$

$x_l = 12$, $x_u = 16$

Check: $f(12) \times f(16) = 6.067 \times -2.269$ (i.e. < 0)

$\rightarrow$ a root exists in this range

Step 2: Estimate the root as lying <u>at the midpoint</u> of this range: $x_r = (x_l + x_u) / 2 = 14$

Step 3: Calculate $f(x_l)\, f(x_r) = f(12)\, f(14) = 9.517$. I.e. > 0, the true root does not lie between $x_l$ and $x_r$

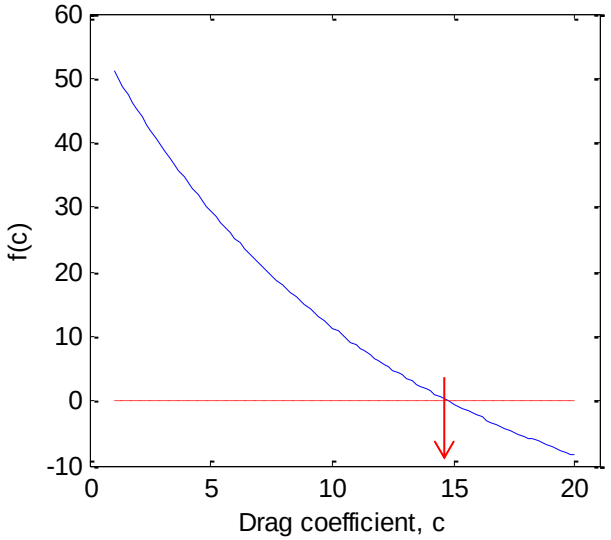
$\rightarrow$ the true root must lie between $x_r$ and $x_u$ (between 14 and 16)

Return to step 2 with the new guesses for the interval:

$\rightarrow$ new estimate for $x_r = (14+16)/2 = 15$

$\rightarrow$ test whether the root lies between 14-15 or 15-16

The true value for the root is $c = 14.7802$




We can evaluate the error in the value generated by the bisection method at each iteration:

As previously when evaluating the error in iterative methods, we can use the *current* and *previous* values to generate an approximate relative error, $\varepsilon_a$ :

$$\varepsilon_a = \left| \frac{x_r^{new} - x_r^{old}}{x_r^{new}} \right| \times 100\%$$

| Iteration | $x_l$ | $x_u$ | $x_r$ | $\varepsilon_a$ [%] | $\varepsilon_t$ [%] |
|---|---|---|---|---|---|
| 1 | 12 | 16 | 14 | - | 5.279 |
| 2 | 14 | 16 | 15 | 6.667 | 1.487 |
| 3 | 14 | 15 | 14.5 | 3.448 | 1.896 |
| 4 | 14.5 | 15 | 14.75 | 1.695 | 0.204 |
| 5 | 14.75 | 15 | 14.875 | 0.840 | 0.641 |
| 6 | 14.75 | 14.875 | 14.8125 | 0.422 | 0.219 |

- After 6 iterations, the approximate relative error $\varepsilon_a < 0.5\%$ (and $\varepsilon_t$ is even lower)

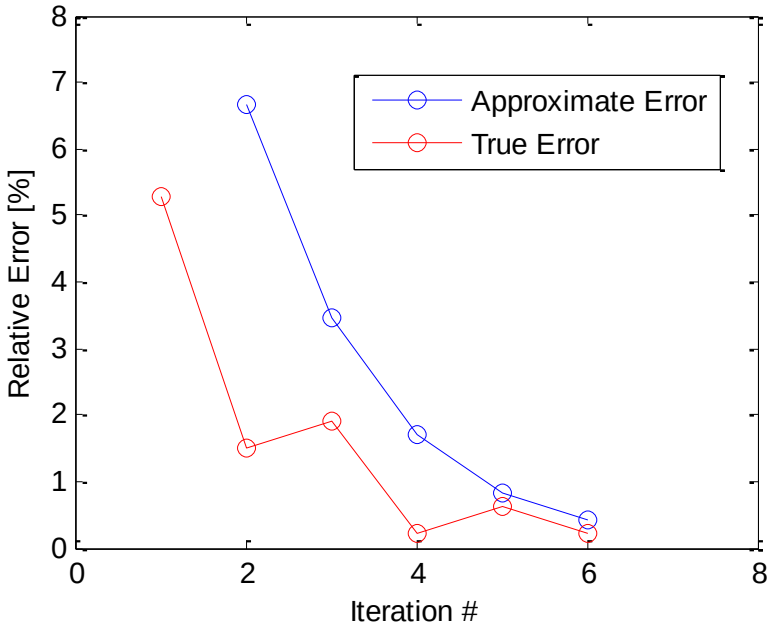
- Compare the approximate relative error to the true relative error:

| Iteration | $\varepsilon_a$ [%] | $\varepsilon_t$ [%] |
|---|---|---|
| 1 | - | 5.279 |
| 2 | 6.667 | 1.487 |
| 3 | 3.448 | 1.896 |
| 4 | 1.695 | 0.204 |
| 5 | 0.840 | 0.641 |
| 6 | 0.422 | 0.219 |

- With the bisection method, the true relative error is always less than the approximate relative error:

(This is a good thing for engineering design / problem solving)

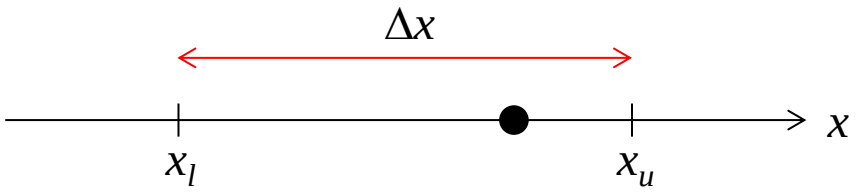
If we terminate the calculation when $\varepsilon_a < \varepsilon_s$, we can be confident that the true value is within our stopping criterion



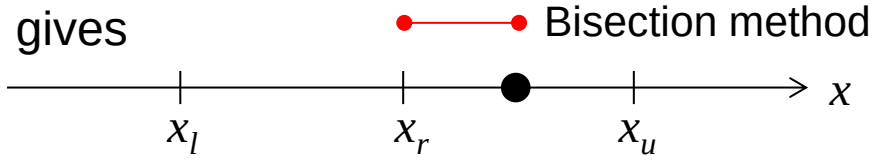
• With the bisection method, "*the true relative error is always less than the approximate relative error*". Why is this the case?

Choose lower and upper guesses for the root:

True root: ●




The bisection method gives a new estimate at $x_r$




→ This true error *must* always be less than $\pm \Delta x/2$:

Our estimated (approximate) error $\varepsilon_a$ is always equal to $\Delta x/2$: $x_r^{new} - x_r^{old} = \frac{x_u - x_l}{2}$

So our relative approximate error will always <u>overestimate</u> the true error:

$$\varepsilon_a = \left| \frac{x_r^{new} - x_r^{old}}{x_r^{new}} \right| \times 100\%$$

# Bracketing Methods - Example: Matlab

## fzero

Find root of continuous function of one variable

### Syntax

```
x = fzero(fun,x0)
x = fzero(fun,x0,options)
[x,fval] = fzero(...)
[x,fval,exitflag] = fzero(...)
[x,fval,exitflag,output] = fzero(...)
```

### Description

`x = fzero(fun,x0)` tries to find a zero of `fun` near `x0`, if `x0` is a scalar. `fun` is a function handle. See Function Handles in the MATLAB Programming documentation for more information. The value `x` returned by `fzero` is near a point where `fun` changes sign, or `NaN` if the search fails. In this case, the search terminates when the search interval is expanded until an `Inf`, `NaN`, or complex value is found.

Parameterizing Functions in the MATLAB Mathematics documentation, explains how to pass additional parameters to your objective function `fun`. See also Example 2 and Example 3 below.

# Review - Functions in Matlab

- Matlab has many built-in functions eg. `sin( ), exp( ), mean( ), bin2dec( ), plot( )..,`
- We can also define our own functions which can be called (by name) from the command line, or from within other scripts / m-files

## function

Declare function

### Syntax

```
function [out1, out2, ...] = myfun(in1, in2, ...)
```

### Description

`function [out1, out2, ...] = myfun(in1, in2, ...)` declares the function `myfun`, and its inputs and outputs. The function declaration must be the first executable line of any MATLAB function.

The existing commands and functions that compose the new function reside in a text file that has a `.m` extension to its filename.

### Example 1

The existence of a file on disk called `stat.m` containing this code defines a new function called `stat` that calculates the mean and standard deviation of a vector:

```
function [mean,stdev] = stat(x)
n = length(x);
mean = sum(x)/n;
stdev = sqrt(sum((x-mean).^2/n));
```

Call the function, supplying two output variables on the left side of the equation:

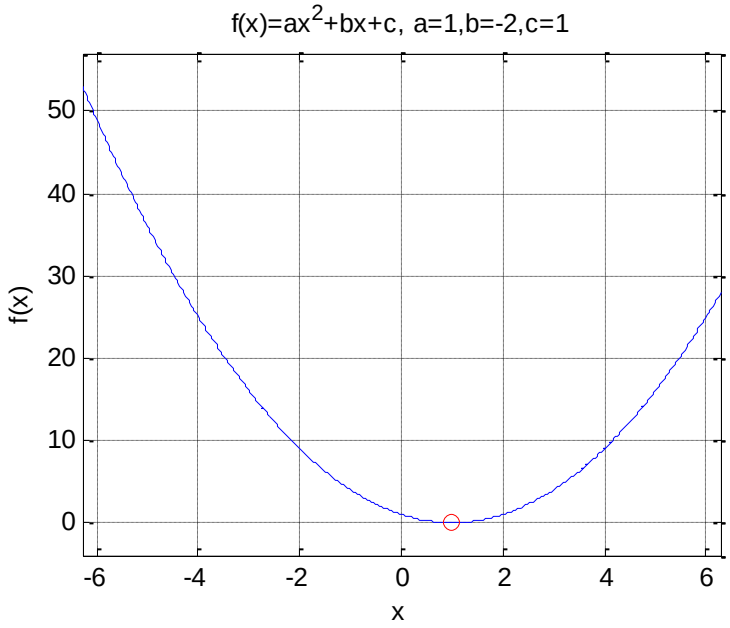
```
[mean stdev] = stat([12.7 45.4 98.9 26.6 53/1])
mean =
   47.3200
stdev =
   29.4085
```

# Anonymous Functions in Matlab

- Suppose we want to find the minimum of the following function

$$f(x) = ax^2 + bx + c$$

when $a$ = 1, $b$ = -2, $c$ = 1




Create an ***anonymous function*** (@):

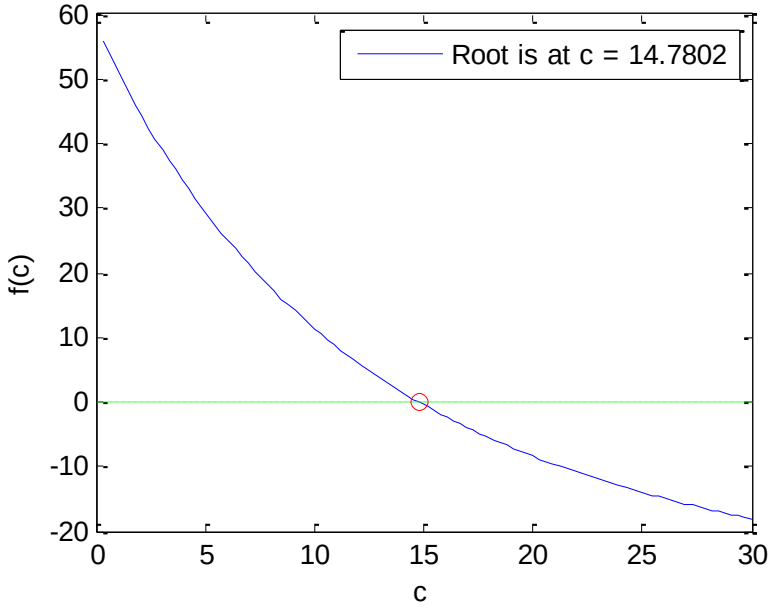
```
clear all;
close all;

a = 1; b = -2; c = 1;

f = @(x)(a*x.^2 + b*x + c);
```

```
ezplot(f);                          % Plots function f(x) over [-2pi < x < 2pi]
title('f(x)=ax^2+bx+c, a=1,b=-2,c=1');
xlabel('x'); ylabel('f(x)');
hold on;

 % Find and plot the minimum
minimum = fminbnd(f,-2,2);      % We can pass our function directly to the minimization routine

plot(minimum,f(minimum),'ro'); % We can evaluate our function at the value "minimum"
grid;
```

Anonymous functions can have more than one variable eg: `f = @(x,y)(a*x.^2+b*y.^2)`

- Back to the root finding problem…

- Same falling object problem solved in Matlab




```
clear all
close all

g = 9.8;
m = 68.1;
v = 40;
t = 10;

fun1 = @(c) (g*m./c).*(1-exp(-c*t/m))-v;
                                    % This is the function we are evaluating
                                    % Expressed here as an anonymous function


x = linspace(0,30,100);             % Define a range of x values
xx = linspace(0,0,100);

figure;
plot(x,fun1(x)); hold on;           % Plot the function
plot(x,xx,'g--');                   % Plot a green dashed line at f(x) = 0
xlabel('c'); ylabel('f(c)');


guess = 16;                         % Our guess for the x-value of the root
[root,fval,exitflag,output] = fzero(fun1,guess)
                                    % Use fzero with initial guess to find the nearest root
hold on; plot(root,0,'ro');         % Plot the location of the root with a red circle

legend(['Root is at c = ',num2str(root)]);
```

# Nonlinear Systems – Dunn Chapter 5

Next time: False-position, Newton's method, Fixed point iteration