PHY604 Lecture 9

September 23, 2025

Today’s lecture:
ODEs and Linear Algebra

- Boundary Value problems

- Eigenvalue problems

- Linear algebra
    - Gaussian elimination
    - LU decomposition

Boundary value problems

- The orbital example we have been studying is an initial value problem: Solving ODEs given some initial value

- Boundary value problems: Conditions needed to specify the solution given at some different (or additional) points to the initial point
  - E.g.: Find a solution for the EOM such that the trajectory passes through a specific point in the future

- Boundary value problems are more difficult to solve
  - Two methods: Shooting method and relaxation method (we will discuss the latter in terms of PDEs later)

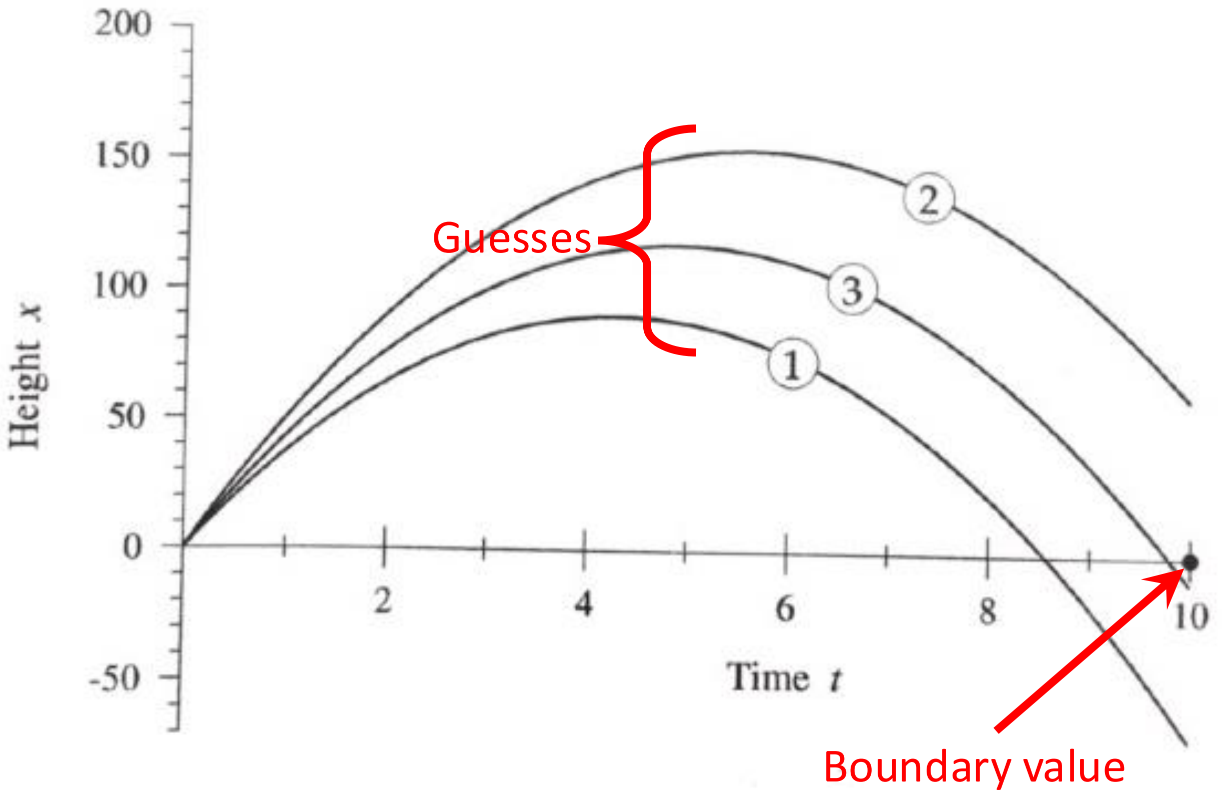
Shooting method example: Ball thrown in the air

- “Trial-and-error” method: Searches for correct values of initial conditions that match a given set of boundary conditions

- Example (from Newman Sec. 8.6): Height of a ball thrown in the air

$$\frac{d^2x}{dt^2} = -G$$

- Guess initial conditions (initial vertical velocity) for which the ball will return to the ground at a given time *t*



How do we modify initial conditions between guesses?

- Write the height of the ball at the boundary t_1 as $x = f(v)$ where v is the initial velocity
- If we want the ball to be at $x = 0$ at t_1, we need to solve $f(v) = 0$

- So, we have reformulated the problem as finding a root of a function
  - We can use, e.g., the bisection method, Newton-Raphson method, secant method
- The function is "evaluated" by solving the differential equation
  - We can use any method discussed previously, e.g., Runge-Kutta, Bulirsch-Stoer, etc.

Today’s lecture:
ODEs and Linear Algebra

- Boundary Value problems
- Eigenvalue problems
- Linear algebra
  - Gaussian elimination
  - LU decomposition

Eigenvalue problems

- Special type of boundary value problem: Linear and homogeneous
  - Every term is linear in the dependent variable
- E.g.: Schrodinger equation:

$$-\frac{\hbar^2}{2m}\frac{d^2\psi}{dx^2} + V(x)\psi(x) = E\psi(x)$$

- Consider the Schrodinger equation in a 1D square well with infinite walls:

$$V(x) = \begin{cases} 0, & \text{for } 0 < x < L \\ \infty, & \text{elsewhere} \end{cases}$$

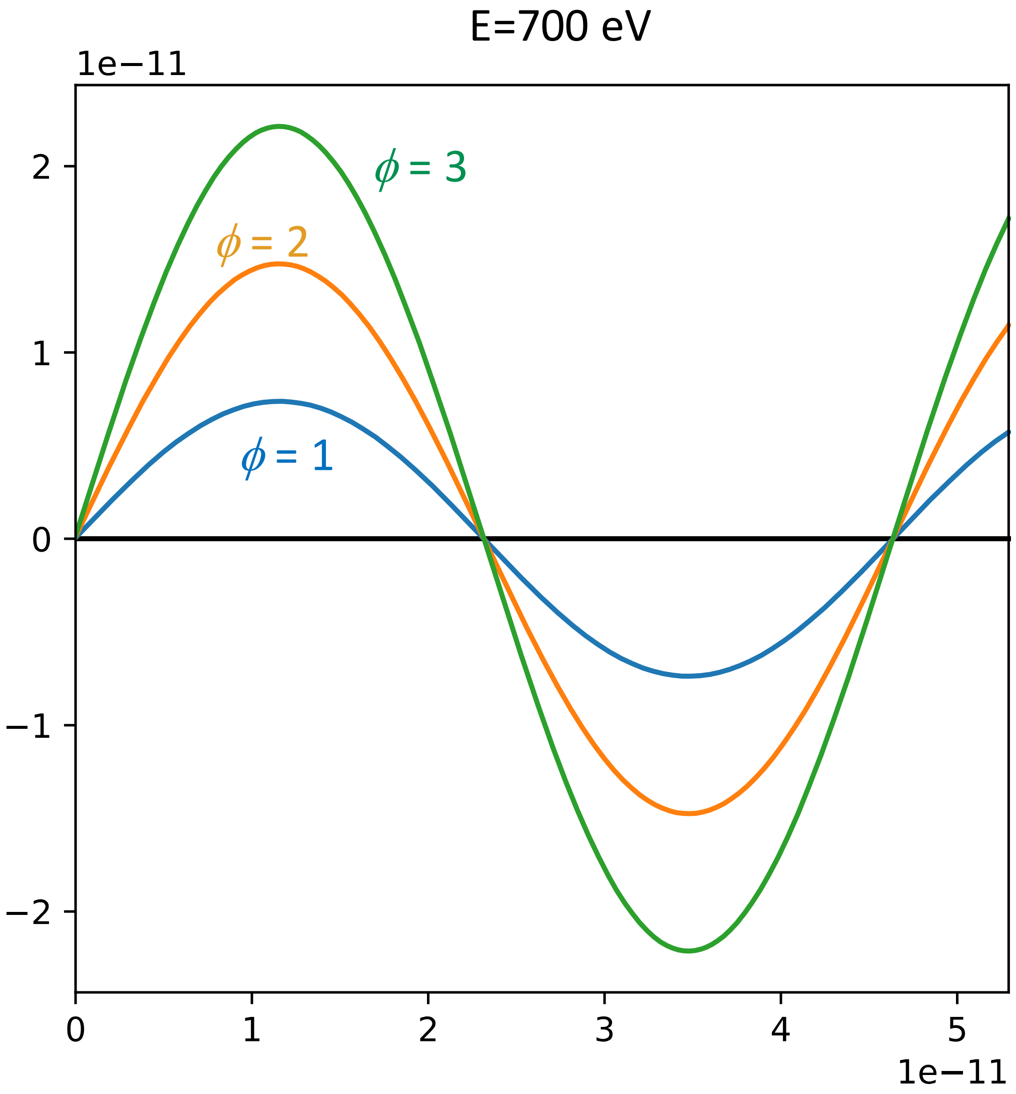
Schrodinger equation in 1D well

- As usual, make into system of 1D ODEs:

$$\frac{d\psi}{dx} = \phi, \quad \frac{d\phi}{dx} = \frac{2m}{\hbar^2}[V(x) - E]\psi$$

- Know that $\psi = 0$ at $x = 0$ and $x = L$, but don't know ϕ
- Let's choose a value of E and solve using some choices for ϕ :
- Since the equation is linear, scaling the initial conditions exactly scales the $\psi(x)$

- **No matter what ϕ, we will never get a valid solution!** (only affects overall magnitude, not shape)



Only specific E has a valid solution

- Solutions only exist for eigenvalues
- Need to vary E, ϕ can be fixed via normalization
- Same strategy, Find the E such that $\psi(L) = 0$

D

Today's lecture:
ODEs and Linear Algebra

- Boundary Value problems
- Eigenvalue problems
- Linear algebra
  - Gaussian elimination
  - LU decomposition

Numerical linear algebra (Garcia Ch. 4)

- Basic problem to solve: **A x** = **b**

- We have already seen many cases where we need to solve linear systems of equations
  - E.g., ODE integration, cubic spline interpolation

- More that we will come across:
  - Solving the diffusion PDE
  - Multivariable root-finding
  - Curve fitting

- We will explore some key methods to understand what they do
  - Mostly, efficient and robust libraries exist, so no need to reprogram

- Often it is illustrative to compare between how we would solve linear algebra by hand and (efficiently) on the computer

Review of matrices: Multiplication

- Matrix-vector multiplication:
  - **A** is *m* x *n* matrix
  - **x** is *n* x 1 (column) vector
  - Result: **b** is *m* x 1 (column vector)
  - Simple scaling: $O(N^2)$ operations

$$b_i = (Ax)_i = \sum_{j=1}^{n} A_{ij} x_j$$

- Matrix-matrix multiplication
  - **A** is *m* x *n* matrix
  - **B** is *n* x *p* matrix
  - Result: **AB** is *m* x *p* matrix
  - Direct multiplication: $O(N^3)$ operations
    - Some faster algorithms exist (make use of organization of sub-matrices for simplification)

$$(AB)_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj}$$

Review of matrices: Determinant

- Encodes some information about a square matrix
  - Used in some linear systems algorithms
  - Solution to linear systems only exists if determinant is nonzero
- Simple algorithm for obtaining determinant is Laplace expansion
- For simple matrices, can be done by hand:

$$\begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc$$

$$\begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix} = a \begin{vmatrix} e & f \\ h & i \end{vmatrix} - b \begin{vmatrix} d & f \\ g & i \end{vmatrix} + c \begin{vmatrix} d & e \\ g & h \end{vmatrix}$$

- What about big matrices?

Review of matrices: Determinant

- Encodes some information about a square matrix
  - Used in come linear systems algorithms
  - Solution to linear systems only exists if determinant is nonzero
- **By hand:** Simple algorithm for obtaining determinant is Laplace expansion
- For simple matrices, can be done by hand:

$$\begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc$$

$$\begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix} = a\begin{vmatrix} e & f \\ h & i \end{vmatrix} - b\begin{vmatrix} d & f \\ g & i \end{vmatrix} + c\begin{vmatrix} d & e \\ g & h \end{vmatrix}$$

- What about big matrices? **Will need a more efficient implementation!**

Review of matrices: Inverse

- $\mathbf{A}^{-1}\mathbf{A}=\mathbf{A}\mathbf{A}^{-1}=\mathbf{I}$

- Formally, the solution to a linear system $\mathbf{A}\,\mathbf{x} = \mathbf{b}$ is $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$
  - Usually less expensive to get the solution without computing the inverse first

- Non-invertible (i.e., singular) if determinant is 0

By hand: Cramer's rule

- One simple way to solve **A x** = **b** is:

$$x_i = \frac{|\mathbf{A}_i|}{|\mathbf{A}|}$$

- Where $\mathbf{A}_i$ is **A** with the *i*th column replaced by **b**

- Comparable speed to calculating the inverse

By hand: Gaussian elimination

- Main general technique for solving **A x = b**
  - Does not involve matrix inversion
  - For “special” matrices, faster techniques may apply
- Involves forward-elimination and back-substitution

- Consider a simple example (from Garcia Ch. 4):

$$
\begin{aligned}
x_1 + x_2 \; + x_3 &= 6 \\
-x_1 + 2x_2 \qquad &= 3 \\
2x_1 \qquad\quad + x_3 &= 5
\end{aligned}
$$

By hand: Forward elimination

- 1. Eliminate x_1 from second and third equation. Add first equation to the second and subtract twice the first equation from the third:

$$
\begin{aligned}
x_1 + x_2 \quad + x_3 &= 6 \\
3x_2 + x_3 &= 9 \\
-2x_2 - x_3 &= -7
\end{aligned}
$$

- 2. Eliminate x_2 from third equation. Multiply the second equation by (-2/3) and subtract it from the third

$$
\begin{aligned}
x_1 + x_2 \quad + x_3 &= 6 \\
3x_2 \quad + x_3 &= 9 \\
-\frac{1}{3} x_3 &= -1
\end{aligned}
$$

By hand: Back substitution

$$\begin{aligned} x_1 + x_2 \quad + x_3 &= 6 \\ 3x_2 \quad + x_3 &= 9 \\ -\frac{1}{3}x_3 &= -1 \end{aligned}$$

- 3. Solve for x_3 = 3.
- 4. Substitute x_3 into the second equation to get x_2 = 2
- 5. Substitute x_3 and x_2 into the first equation to get x_1 = 1

- In general, for *N* variables and *N* equations:
  - Use forward elimination make the last equation provide the solution for x_N
  - Back substitute from the *N*th equation to the first
  - Scales like N^3 (can do better for “sparse” equations)

Pitfalls of Gaussian substitution: Roundoff errors

- Consider a different example (also from Garcia):

$$\begin{aligned} \epsilon x_1 + x_2 + x_3 &= 5 \\ x_1 + x_2 \quad\quad &= 3 \\ x_1 + \quad + x_3 &= 4 \end{aligned}$$

- First, lets take $\epsilon \to 0$ and solve:

Subtract second from third:

$$\begin{aligned} x_2 + x_3 &= 5 \\ x_1 + x_2 \quad\quad &= 3 \\ -x_2 + x_3 &= 1 \end{aligned}$$

Add first to third:

$$\begin{aligned} x_2 + x_3 &= 5 \\ x_1 + x_2 \quad\quad &= 3 \\ 2x_3 &= 6 \end{aligned}$$

Back substitute:

$$\begin{aligned} x_2 &= 2 \\ x_1 &= 1 \\ x_3 &= 3 \end{aligned}$$

Roundoff error example: Now solve with ε

- Forward elimination starts by multiplying first equation by $1/\varepsilon$ and subtracting it from second and third:

$$
\begin{aligned}
\epsilon x_1 + \quad x_2 \quad + x_3 &= 5 \\
(1 - 1/\epsilon)x_2 \quad -(1/\epsilon)x_3 &= 3 - 5/\epsilon \\
-(1/\epsilon)x_2 \; +(1 - 1/\epsilon)x_3 &= 4 - 5/\epsilon
\end{aligned}
$$

- Clearly have an issue if ε is near zero, e.g., if $C - 1/\epsilon \to -1/\epsilon$ for C order unity:

$$
\begin{aligned}
\epsilon x_1 + \quad x_2 \quad + x_3 &= 5 \\
-(1/\epsilon)x_2 \; -(1/\epsilon)x_3 &= -5/\epsilon \\
-(1/\epsilon)x_2 \; -(1/\epsilon)x_3 &= -5/\epsilon
\end{aligned}
$$

Cannot solve, now have two equations, three unknowns

Simple fix: Pivoting

- Interchange the order of the equations before performing the forward elimination

$$
\begin{aligned}
x_1 + x_2 \qquad &= 3 \\
\epsilon x_1 + x_2 + x_3 &= 5 \\
x_1 + \quad + x_3 &= 4
\end{aligned}
$$

- Now the first step of forward elimination gives us:

$$
\begin{aligned}
x_1 + x_2 \qquad\qquad &= 3 \\
(1-\epsilon)x_2 + x_3 &= 5 - 3\epsilon \\
-x_2 \qquad + x_3 &= 1
\end{aligned}
$$

- Now we round off:

$$
\begin{aligned}
x_1 + x_2 \qquad &= 3 \\
x_1 \quad + x_3 &= 5 \\
-x_2 + x_3 &= 1
\end{aligned}
$$

Same as when we initially took ε to 0.

Gaussian elimination with pivoting

- Partial-pivoting:
  - Interchange of rows to move the one with the largest element in the current column to the top
  - (Full pivoting would allow for row and column swaps—more complicated)

- Scaled pivoting
  - Consider largest element relative to all entries in its row
  - Further reduces roundoff when elements vary in magnitude greatly

- Row echelon form: This is the form that the matrix is in after forward elimination

Matrix determinants with Gaussian elimination

- Once we have done forward substitution and obtained a row echelon matrix it is trivial to calculate the determinant:

$$\det(\mathbf{A}) = (-1)^{N_{\text{pivot}}} \prod_{i=1}^{N} A_{ii}^{\text{row-echelon}}$$

- Every time we pivoted in the forward substitution, we change the sign

Matrix inverse with Gaussian elimination

- We can also use Gaussian elimination to fin the inverse of a matrix
- We would like to find $\mathbf{A}\mathbf{A}^{-1} = \mathbf{I}$
- We can use Gaussian elimination to solve: $\mathbf{A}\,\mathbf{x}_i = \mathbf{e}_i$
  - $\mathbf{e}_i$ is a column of the identity:

$$\mathbf{e}_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \end{bmatrix}, \quad \mathbf{e}_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \end{bmatrix}, \quad \mathbf{e}_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ \vdots \end{bmatrix}, \ldots, \; \mathbf{e}_N = \begin{bmatrix} \vdots \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

  - $\mathbf{x}_i$ is a column of the inverse:

$$\mathbf{A}^{-1} = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \mathbf{x}_3 & \ldots & \mathbf{x}_N \end{bmatrix}$$

Singular matrix

- If a matrix has a vanishing determinant, then the system is not solvable
- Common way for this to enter, one equation in the system is a linear combination of some others
- Not always easy to detect from the start

Singular and close to singular matrices

- Condition number: Measures how close to singular we are
  - How much **x** would change with a small change in **b**

$$\text{cond}(\mathbf{A}) = ||\mathbf{A}||\;||\mathbf{A}^{-1}||$$

  - Requires defining a norm of **A**
    - https://en.wikipedia.org/wiki/Matrix_norm
  - See, e.g., numpy implementation:
    - https://numpy.org/doc/stable/reference/generated/numpy.linalg.cond.html

- Rule of thumb: $$\frac{||\mathbf{x}^{\text{exact}} - \mathbf{x}^{\text{calc}}||}{||\mathbf{x}^{\text{exact}}||} \simeq \text{cond}(\mathbf{A}) \cdot \epsilon^{\text{machine}}$$

Tridiagonal and banded matrices

- We saw this type of matrix when solving for cubic spline coefficients:

$$\begin{pmatrix} 4\Delta x & \Delta x & & & & \\ \Delta x & 4\Delta x & \Delta x & & & \\ & \Delta x & 4\Delta x & \Delta x & & \\ & & \ddots & \ddots & \ddots & \\ & & & \Delta x & 4\Delta x & \Delta x \\ & & & & \Delta x & 4\Delta x \end{pmatrix} \begin{pmatrix} p''_1 \\ p''_2 \\ p''_3 \\ \vdots \\ p''_{n-2} \\ p''_{n-1} \end{pmatrix} = \frac{6}{\Delta x} \begin{pmatrix} f_0 - 2f_1 + f_2 \\ f_1 - 2f_2 + f_3 \\ f_2 - 2f_3 + f_4 \\ \vdots \\ f_{n-3} - 2f_{n-2} + f_{n-1} \\ f_{n-2} - 2f_{n-1} + f_n \end{pmatrix}$$

- Often come up in physical situations
- These types of matrices can be efficiently solved with Gaussian elimination

Gaussian elimination for banded matrices

- Only need to do Gaussian elimination steps for *m* nonzero elements below given row (*m* is less than the number of diagonal bands)
- Example:

$$\begin{pmatrix} 2 & 1 & 0 & 0 \\ 3 & 4 & -5 & 0 \\ 0 & -4 & 3 & 5 \\ 0 & 0 & 1 & 3 \end{pmatrix} \rightarrow \begin{pmatrix} 2 & 1 & 0 & 0 \\ 0 & 2.5 & -5 & 0 \\ 0 & -4 & 3 & 5 \\ 0 & 0 & 1 & 3 \end{pmatrix} \rightarrow \begin{pmatrix} 2 & 1 & 0 & 0 \\ 0 & 2.5 & -5 & 0 \\ 0 & 0 & -5 & 5 \\ 0 & 0 & 1 & 3 \end{pmatrix} \rightarrow \begin{pmatrix} 2 & 1 & 0 & 0 \\ 0 & 2.5 & -5 & 0 \\ 0 & 0 & -5 & 5 \\ 0 & 0 & 0 & 4 \end{pmatrix}$$

LU decomposition (Newman Ch. 6)

- Often happens that we would like to solve: $\mathbf{A}\mathbf{x}_i = \mathbf{v}_i$ for the same **A** but many **v**
  - For example, our implementation for the inverse
  - Wasteful to do Gaussian elimination over and over, we will always get the same row echelon matrix, just $\mathbf{v}_i$ will be different
  - Instead, we should keep track of operations we did to $\mathbf{v}_1$ and use them over and over
- Consider a general 4 x 4 matrix:

$$\begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{pmatrix}$$

- Let’s perform Gaussian elimination

LU decomposition: First GE step

- Write the first step of the GE as:

$$\frac{1}{a_{00}}\begin{pmatrix} 1 & 0 & 0 & 0 \\ -a_{10} & a_{00} & 0 & 0 \\ -a_{20} & 0 & a_{00} & 0 \\ -a_{30} & 0 & 0 & a_{00} \end{pmatrix}\begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{pmatrix} = \begin{pmatrix} 1 & b_{01} & b_{02} & b_{03} \\ 0 & b_{11} & b_{12} & b_{13} \\ 0 & b_{21} & b_{22} & b_{23} \\ 0 & b_{31} & b_{32} & b_{33} \end{pmatrix}$$

- Where the *b*'s are some linear combination of *a* coefficients
- The first matrix on the LHS is a lower triangular matrix we call:

$$\mathbf{L}_0 \equiv \frac{1}{a_{00}}\begin{pmatrix} 1 & 0 & 0 & 0 \\ -a_{10} & a_{00} & 0 & 0 \\ -a_{20} & 0 & a_{00} & 0 \\ -a_{30} & 0 & 0 & a_{00} \end{pmatrix}$$

LU decomposition: Second LU step

$$\frac{1}{b_{11}}\begin{pmatrix} b_{11} & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -b_{21} & b_{11} & 0 \\ 0 & -b_{31} & 0 & b_{11} \end{pmatrix}\begin{pmatrix} 1 & b_{01} & b_{02} & b_{03} \\ 0 & b_{11} & b_{12} & b_{13} \\ 0 & b_{21} & b_{22} & b_{23} \\ 0 & b_{31} & b_{32} & b_{33} \end{pmatrix} = \begin{pmatrix} 1 & c_{01} & c_{02} & c_{03} \\ 0 & 1 & c_{12} & c_{13} \\ 0 & 0 & c_{22} & c_{23} \\ 0 & 0 & c_{32} & c_{33} \end{pmatrix}$$

$$\mathbf{L}_1 \equiv \frac{1}{b_{11}}\begin{pmatrix} b_{11} & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -b_{21} & b_{11} & 0 \\ 0 & -b_{31} & 0 & b_{11} \end{pmatrix}$$

LU decomposition: Last two steps for 4x4 matrix

$$\mathbf{L}_2 \equiv \frac{1}{c_{22}} \begin{pmatrix} c_{22} & 0 & 0 & 0 \\ 0 & c_{22} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -c_{32} & c_{22} \end{pmatrix}, \quad \mathbf{L}_3 \equiv \frac{1}{d_{33}} \begin{pmatrix} d_{33} & 0 & 0 & 0 \\ 0 & d_{33} & 0 & 0 \\ 0 & 0 & d_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- So, we can write:

$$\mathbf{L}_3\mathbf{L}_2\mathbf{L}_1\mathbf{L}_0\mathbf{A} = \mathbf{L}_3\mathbf{L}_2\mathbf{L}_1\mathbf{L}_0\mathbf{v}$$

- Afterwards, the equation is ready for back substitution

- Mathematically identical to Gaussian elimination, but we only have to find $\mathbf{L}_0$-$\mathbf{L}_3$ once, and then we can operate on many **v**’s

Slightly different formulation of LU decomposition

- From the properties of upper triangular matrices (same holds for lower):
  - Product of two upper triangular matrices is an upper triangular matrix.
  - Inverse of an upper triangular matrix is an upper triangular matrix

- Consider the lower-diagonal matrix **L** and the upper-diagonal matrix **U**:

$$\mathbf{L} = \mathbf{L}_0^{-1}\mathbf{L}_1^{-1}\mathbf{L}_2^{-1}\mathbf{L}_3^{-1}, \quad \mathbf{U} = \mathbf{L}_3\mathbf{L}_2\mathbf{L}_1\mathbf{L}_0\mathbf{A}$$

- Then trivially: **LU** = **A**, so for **Ax** = **v**,, we can write **LUx** = **v**

Expression for L

- We can confirm that for our 4 x 4 example,

$$\mathbf{L}_0^{-1} = \begin{pmatrix} a_{00} & 0 & 0 & 0 \\ a_{10} & 1 & 0 & 0 \\ a_{20} & 0 & 1 & 0 \\ a_{30} & 0 & 0 & 1 \end{pmatrix}, \quad \mathbf{L}_1^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & b_{11} & 0 & 0 \\ 0 & b_{21} & 1 & 0 \\ 0 & b_{31} & 0 & 1 \end{pmatrix}, \quad \mathbf{L}_2^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & c_{22} & 0 \\ 0 & 0 & c_{32} & 1 \end{pmatrix}, \quad \mathbf{L}_3^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & d_{33} \end{pmatrix}$$

- Multiplying together we get

$$\mathbf{L} = \begin{pmatrix} a_{00} & 0 & 0 & 0 \\ a_{10} & b_{11} & 0 & 0 \\ a_{20} & b_{21} & c_{22} & 0 \\ a_{30} & b_{31} & c_{32} & d_{33} \end{pmatrix}$$

Solving the equation with **L** and **U**

- Break into two steps:
  - 1. **Ly** = **v** can be solved by back substitution:

$$\begin{pmatrix} l_{00} & 0 & 0 & 0 \\ l_{10} & l_{11} & 0 & 0 \\ l_{20} & l_{21} & l_{22} & 0 \\ l_{30} & l_{31} & l_{32} & l_{33} \end{pmatrix} \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \end{pmatrix}$$

  - 2. Now solve **Ux** = **y** by back substitution:

$$\begin{pmatrix} u_{00} & u_{01} & u_{02} & u_{03} \\ 0 & u_{11} & u_{12} & u_{13} \\ 0 & 0 & u_{22} & u_{23} \\ 0 & 0 & 0 & u_{33} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix}$$

Some comments about **LU** decomposition

- Most common method for solving simultaneous equations
- Decomposition needs to be done once, then only back substitution is needed for different **v**
- In general, still may need to pivot
  - Every time you swap rows, you have to do the same to **L**
  - Need to perform the same sequence of swaps on **v**

After class tasks

- Readings:
  - Newman Ch. 6
  - Garcia Ch. 4
  - Pang Sec. 5.3