§2.1 Solving Linear Systems

- Augmented coefficient matrix:

$$A_{11}x_1 + A_{12}x_2 + \ldots + A_{1n}x_n = b_1$$

$$A_{21}x_1 + A_{22}x_2 + \ldots + A_{2n}x_n = b_2$$

....

$$A_{n1}x_1 + A_{n2}x_2 + \ldots + A_{nn}x_n = b_n$$

THE UNIVERSITY of TENNESSEE UT KNOXVILLE

Big Orange. Big Ideas.

Uniqueness of Solution

- Unique if **A** is non-singular or det(**A**) = $|\mathbf{A}| \neq 0$; equivalently, the rows of **A** and columns of **A** must be *linearly* independent.
- If **A** is singular, then **A**x=b has ∞ or no solutions.

Ill-conditioning

- Case when coefficient matrix **A** is *nearly* singular (or when $|\mathbf{A}|$ is very small).
- Can estimate $|A| \ll \|A\|$, where $\|A\|$ is a matrix norm:

THE UNIVERSITY of TENNESSEE KNOXVILLE
Big Orange. Big Ideas.

Ill-conditioning

- The condition number of a **A** is defined by cond(**A**) = $||A|| \times ||A^{-1}||$.
- If cond(**A**) ≅ 1, then we say **A** is well-conditioned; cond(**A**) is not unique – depends on the norm; cond(**A**) is expensive to compute so we typically compare |**A**| to magnitudes of matrix **A** elements.

THE UNIVERSITY of TENNESSEE UT KNOXVILLE
Big Orange. Big Ideas.

Ill-conditioning

- Suppose we modify the second equation of the previous example to be $2x + 1.002y = 0$. What does the solution to the 2 by 2 linear system become?

- So, we see that a ____% change in **A** yields a ____% change in the solution.

Methods of Solution

- **Direct methods** – based on *elementary operations* that do not augment the solution:
  1. Exchange 2 rows (changes sign of $|\mathbf{A}|$);
  2. Multiply equation by nonzero constant α ($|\mathbf{A}|$ becomes $\alpha|\mathbf{A}|$);
  3. Multiply equation by nonzero constant α and subtract it from another equation ($|\mathbf{A}|$ unchanged).

- **Iterative methods** (or indirect methods) – typically used for large and sparse **A** (more zeros than nonzeros); guess solution and improve every iteration.

THE UNIVERSITY of TENNESSEE UT KNOXVILLE
Big Orange. Big Ideas.

Direct Methods

Method	Initial Form	Final Form
Gauss Elimination	**A**x=b	**U**x=c
LU Decomposition	**A**x=b	**LU**x=b
Gauss-Jordan Elim.	**A**x=b	**I**x=c

Sample **U**,**L** for 3-by-3 matrices:

THE UNIVERSITY of TENNESSEE KNOXVILLE
Big Orange. Big Ideas.

LU Decomposition

- Transform **A**x=b given **A**=**LU**:
  (**LU**)x=b ➔ **L**(**U**x)=b; let **U**x=y and solve **L**y=b for y via *forward substitution*, then solve **U**x=y for the solution vector x.

LU Decomposition

- Now that we have $y=(y_1\ y_2\ y_3)^T$ use *back-substitution* to get $x=(x_1\ x_2\ x_3)^T$:

LU Decomposition Example

$$A = \begin{pmatrix} 8 & -6 & 2 \\ -4 & 11 & -7 \\ 4 & -7 & 6 \end{pmatrix}, \quad b = \begin{pmatrix} 28 \\ -40 \\ 33 \end{pmatrix}$$

Gaussian Elimination

- Use elementary row operations that preserve the solution and produce an *upper-triangular* augmented coefficient matrix.

Example: $$\begin{aligned} 4x_1 - 2x_2 + x_3 &= 11 \\ -2x_1 + 4x_2 - 2x_3 &= -16 \\ x_1 - 2x_2 + 4x_3 &= 17 \end{aligned} \quad \rightarrow \begin{array}{c} \left(\begin{array}{ccc|c} 4 & -2 & 1 & 11 \\ -2 & 4 & -2 & -16 \\ 1 & -2 & 4 & 17 \end{array}\right) \end{array} \begin{array}{c} \text{a} \\ \text{b} \\ \text{c} \end{array}$$

Gaussian Elimination

- Now element in (2,2) position is the *pivot*:

$$Eq(c) \leftarrow Eq(c) - (-0.5) \times Eq(b) \rightarrow \left(\begin{array}{ccc|c} 4 & -2 & 1 & 11 \\ 0 & 3 & -1.5 & -10.5 \\ 0 & 0 & 3 & 9 \end{array}\right)$$

THE UNIVERSITY of TENNESSEE UT
KNOXVILLE
Big Orange. Big Ideas.

$$(x_1, x_2, x_3)^T = (\_\_\_, \_\_\_, \_\_\_)^T$$

Gaussian Elimination

- Python code for elimination phase (getting the upper-triangular augmented coefficient matrix):

```
for k in range(0,n-1):
  for i in range(k+1,n):
      if a[i,k] != 0.0
         lam=a[i,k]/a[k,k]    #pivot
         a[i,k+1:n]=a[i,k+1:n]-lam*a[k,k+1:n]
         b[i]=b[i]-lam*b[k]
```

- What happens to $\mathbf{A}_{ij}$ for $i > j$?

THE UNIVERSITY of TENNESSEE KNOXVILLE

Big Orange. Big Ideas.

Gaussian Elimination

- See `gaussElim.py` on page 41 of textbook.
- Can handle multiple right-hand-sides (**AX**=**B**) with minor changes to `gaussElim.py` but method would be inefficient; *back-substitution* has to be repeated for each final column of **B**.

§2.3 LU Decomposition

- Can express any n-by-n matrix $\mathbf{A}$ as $\mathbf{A}=\mathbf{LU}$.

Different forms:

Doolittle	$\mathbf{L}_{ii}=1, i=1,2,\ldots,n$
Crout	$\mathbf{U}_{ii}=1, i=1,2,\ldots,n$
Choleski	$\mathbf{L}=\mathbf{U}^T$ (for symmetric matrices $\mathbf{A}$)

How do we solve $\mathbf{A}x=b$ using $\mathbf{A}=\mathbf{LU}$?

THE UNIVERSITY of TENNESSEE UT KNOXVILLE

Big Orange. Big Ideas.

Doolittle's Decomposition

- Most common **LU** decomposition; matrix (or array **A**) can be overwritten: Implicit that the diagonal of **L** is all ones; element in $\mathbf{L}_{ij}$ is the pivot equation multiplier λ needed to zero the current $\mathbf{A}_{ij}$ element; `LUdecomp.py` on p. 47 in textbook.

$$\begin{pmatrix} U_{11} & U_{12} & U_{13} \\ L_{21} & U_{22} & U_{23} \\ L_{31} & L_{32} & U_{33} \end{pmatrix}$$

THE UNIVERSITY of TENNESSEE UT KNOXVILLE
Big Orange. Big Ideas.

Choleski Decomposition

- **A**=**L**$\mathbf{L}^{T}$
- **A** must be symmetric (i.e., **A**=$\mathbf{A}^{T}$)
- **A** must be positive definite $(\vec{x}^T A \vec{x} > 0 \ \forall \vec{x} \neq \vec{0})$

$$\begin{pmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix} = \begin{pmatrix} L_{11} & 0 & 0 \\ L_{21} & L_{22} & 0 \\ L_{31} & L_{32} & L_{33} \end{pmatrix} \begin{pmatrix} L_{11} & L_{21} & L_{31} \\ 0 & L_{22} & L_{32} \\ 0 & 0 & L_{33} \end{pmatrix}$$

Choleski Decomposition

- Match **A** to the product **LL**^{T}:

$$\begin{pmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix} = \begin{pmatrix} L_{11}^2 & L_{11}L_{21} & L_{11}L_{31} \\ L_{11}L_{21} & L_{21}^2 + L_{22}^2 & L_{21}L_{31} + L_{22}L_{32} \\ L_{11}L_{31} & L_{21}L_{31} + L_{22}L_{32} & L_{31}^2 + L_{32}^2 + L_{33}^2 \end{pmatrix}$$

Choleski Decomposition

- Generalization for *n*-by-*n* matrices:

$$L_{jj} = \sqrt{A_{jj} - \sum_{k=1}^{j-1} L_{jk}^2}, \quad j = 2,3,\ldots,n$$

$$L_{ij} = (A_{ij} - \sum_{k=1}^{j-1} L_{ik} L_{jk}) / L_{jj}, \quad j = 2,3,\ldots,n; i = j+1, j+2,\ldots,n$$

- See `choleski.py` on p. 50 of textbook.
- Applications: Example 2.8 (p. 54), Problem Set 2.1 (#16, p. 57)

THE UNIVERSITY of TENNESSEE UT
KNOXVILLE
Big Orange. Big Ideas.

§2.4 Symmetric & Banded Matrices

- Sparse matrix – most elements are zero.
- Banded matrix – nonzeros are clustered near the main diagonal; tridiagonal matrix **A** has a *bandwidth* of 3:

$$A = \begin{pmatrix} x & x & & & \\ x & x & x & & \\ & x & x & x & \\ & & x & x & x \\ & & & x & x \end{pmatrix}$$

Big Orange. Big Ideas.

Banded Matrices

- For a banded matrix **A** with **A**=**LU**, both **L** and **U** retain the banded *structure* of **A**:

$$A = \begin{pmatrix} x & x & & & \\ x & x & x & & \\ & x & x & x & \\ & & x & x & x \\ & & & x & x \end{pmatrix}, \quad L = \begin{pmatrix} x & & & & \\ x & x & & & \\ & x & x & & \\ & & x & x & \\ & & & x & x \end{pmatrix}, \quad U = \begin{pmatrix} x & x & & & \\ & x & x & & \\ & & x & x & \\ & & & x & x \\ & & & & x \end{pmatrix}$$

Banded Matrices

- How can we exploit symmetry & banded structure in solving **A**x=b?
- For *tridiagonal* coefficient matrices, we only need to eliminate one element below each pivot element, i.e., only one elementary operation is needed per pivot row.
- See `LUdecomp3.py` on pp. 61-62 of the textbook.

Big Orange. Big Ideas.

Banded Matrices

- In `LUdecomp3.py` notice that `b` is overwritten with the solution vector in `LUsolve3(c,d,e,b)`.

- <u>Application</u>: Example 2.11 on p.68 of the textbook.

Symmetric Coefficient Matrices

- It is common to have symmetric and/or banded matrices in engineering applications; $\mathbf{A}=\mathbf{A}^T$ but not necessarily positive definite in this case.

  If $\mathbf{A}=\mathbf{A}^T$, then $\mathbf{A}=\mathbf{LU}=\mathbf{LDL}^T$, where $\mathbf{D}$ is a diagonal matrix.

- We can use `LUdecomp.py` and recover **L** and **D** from the **U** factor.

THE UNIVERSITY of TENNESSEE
KNOXVILLE
Big Orange. Big Ideas.

Symmetric Coefficient Matrices

- Matching elements of **U** we can recover **L** and **D**:

$$\begin{pmatrix} D_1 & & & & \\ & D_2 & & & \\ & & D_3 & & \\ & & & \dots & \\ & & & & D_n \end{pmatrix}\begin{pmatrix} 1 & L_{21} & L_{31} & \dots & L_{n1} \\ & 1 & L_{32} & \dots & L_{n2} \\ & & 1 & \dots & L_{n3} \\ & & & \dots & \dots \\ & & & & 1 \end{pmatrix} = \begin{pmatrix} D_1 & D_1L_{21} & D_1L_{31} & \dots & D_1L_{n1} \\ & D_2 & D_2L_{32} & \dots & D_2L_{n2} \\ & & D_3 & \dots & D_3L_{n3} \\ & & & \dots & \dots \\ & & & & D_n \end{pmatrix}$$

- Application: Problem Set 2.2 (#3, p. 78): given 5-by-5 symmetric tridiagonal matrix **A**, determine **L** and **D** so that **A**=**LDL**^{T}.

§2.5 Pivoting

- Need to reorder equations during the elimination phase in order to avoid very *small* multipliers (pivot elements).
- Suppose the augmented matrix is $\left(\begin{array}{ccc|c} \varepsilon & -1 & 1 & 0 \\ -1 & 2 & -1 & 0 \\ 2 & -1 & 0 & 1 \end{array}\right)$

Pivoting

- We don't always have to pick $\mathbf{A_{kk}}$ for the pivot element when zeroing out elements in column *k* (below the diagonal).
- Instead, we find $\mathbf{A_{pk}}$ of the largest *relative* size (r_{pk}):

$$r_{pk} = \max_j (r_{jk}), j \geq k,$$

$$\text{where } r_{ij} = \frac{|A_{ij}|}{s_i},\ s_i = \max_j |A_{ij}|, i = 1,2,\dots,n.$$

Pivoting

- If $p \neq k$, we interchange rows k and p and proceed with the elimination.
- See `gaussPivot.py` on pp. 72-73; uses the `swap.py` function on p. 67.
- See also the `LUpivot.py` module on pp. 74-75; incorporates pivoting into Doolittle's decomposition and row permutations are stored in the `seq` vector.
- <u>Sample application</u>: Problem Set 2.2 (#12) on pp. 79-80.

Pivoting (Caveats)

- Destroys symmetry and bandedness which do arise in many engineering applications.
- Pivoting is unnecessary when the matrix is *diagonally dominant*:

$$\begin{pmatrix} 4 & -2 & 1 \\ -2 & 4 & -1 \\ 1 & -1 & 3 \end{pmatrix}$$

- Computation time is increased with pivoting.

§2.6 Matrix Inversion

- Can compute $\mathbf{A}^{-1}$ for an *n*-by-*n* matrix A by solving $\mathbf{AX}=\mathbf{I_n}$.
- If $\mathbf{A}$ is banded, $\mathbf{A}^{-1}$ will be dense (lose nonzero structure); if $\mathbf{A}$ is triangular, $\mathbf{A}^{-1}$ will also be triangular.
- Cost of inverting $\mathbf{A}$ to solve $\mathbf{A}x=b$ (i.e., $x=\mathbf{A}^{-1}b$) is much higher than using LU decomposition.

THE UNIVERSITY of TENNESSEE KNOXVILLE
Big Orange. Big Ideas.

§2.7 Iterative Methods

- Sometimes they are called *indirect* methods.
- Take an initial guess at the solution *x* (for **A**x=b) and repeatedly improve *x* until change is negligible.
- <u>Advantages</u>: store only nonzeros of A and methods are self-correcting (round-off errors are corrected in subsequent iterations).

THE UNIVERSITY of TENNESSEE KNOXVILLE
Big Orange. Big Ideas.

Iterative Methods

- Drawback: do not always converge to the exact solution; if the matrix **A** is *diagonally dominant*, convergence is **guaranteed**.
- Initial guess affects the number of iterations (not really if method converges).
- First method we consider is Gauss-Seidel.

Gauss-Seidel

- Write **A**x=b as $\sum_{j=1}^{n} A_{ij} x_j = b_i, i = 1,2,\ldots,n.$
- Extract term for x_i to get $A_{ii} x_i + \sum_{\substack{j=1 \\ j \neq i}}^{n} A_{ij} x_j = b_i, i = 1,2,\ldots,n.$
- Solving for x_i yields:

$$x_i = \frac{1}{A_{ii}} \left(b_i - \sum_{\substack{j=1 \\ j \neq i}}^{n} A_{ij} x_j \right), i = 1,2,\ldots,n.$$

THE UNIVERSITY of TENNESSEE UT KNOXVILLE

Big Orange. Big Ideas.

Gauss-Seidel

- Algorithm: $$x_i^{(k+1)} \leftarrow \frac{1}{A_{ii}}\left(b_i - \sum_{\substack{j=1 \\ j\neq i}}^{n} A_{ij}x_j^{(k)}\right), i = 1,2,\ldots,n.$$

- Start with an initial guess, $x^{(0)}$ and generate successive iterates $x^{(1)}$, $x^{(2)}$,…, etc. until the difference between $x^{(k+1)}$ and $x^{(k)}$ is sufficiently small; how can convergence be improved?

THE UNIVERSITY of TENNESSEE UT KNOXVILLE
Big Orange. Big Ideas.

Gauss-Seidel

- Algorithm with relaxation:

$$x_i^{(k+1)} \leftarrow \frac{\omega}{A_{ii}}\left(b_i - \sum_{\substack{j=1 \\ j\neq i}}^{n} A_{ij}x_j^{(k)}\right) + (1-\omega)x_i^{(k)}, i = 1,2,\ldots,n.$$

- Take weighted average of previous iteration values; $\omega<1$ (under-relaxation), $\omega>1$ (over-relaxation). What is optimal ω?

Gauss-Seidel

- Can be shown that $\omega_{opt} = \dfrac{2}{1+\sqrt{1-\left(\dfrac{\Delta x^{(k+p)}}{\Delta x^{(k)}}\right)^{1/p}}}$,

  where p is a positive integer and $\Delta x^{(k)} = | x^{(k-1)} - x^{(k)} |$.

- Strategy: (1) perform k iterations with $\omega=1$, after k^{th} iteration record $\Delta x^{(k)}$; (2) perform p more iterations and record $\Delta x^{(k+p)}$; and (3) perform subsequent iterations with $\omega=\omega_{opt}$; see `gaussSeidel.py` on p.89 and **Example 2.17** on pp. 95-96.

THE UNIVERSITY of TENNESSEE UT
KNOXVILLE
Big Orange. Big Ideas.

Conjugate Gradient Method

- Consider the function $f(x) = ½\, x^T \mathbf{A} x - b^T x$, where **A** is a symmetric and positive definite (SPD) matrix and x, b are vectors in R^n.
- The minimum for $f(x)$ occurs when the gradient

  $\nabla f = Ax - b = 0$ or when **A**x=b.
- Desire an iteration of the form $x_{k+1} = x_k + \alpha_k s_k$; where s_k is a *search* direction and α_k is the *step length*.

THE UNIVERSITY of TENNESSEE UT KNOXVILLE
Big Orange. Big Ideas.

Conjugate Gradient Method

- To minimize $f(x_{k+1})$ we choose α_k so that $\mathbf{A}(x_k + \alpha_k s_k) = b$, why?
- Let $r_k = b - \mathbf{A}x_k$ (kth residual? vector) so that $\mathbf{A}(x_k + \alpha_k s_k) = b$ after multiplying through by $\mathbf{A}$ yields $\alpha_k \mathbf{A} s_k = b - \mathbf{A}x_k = r_k$.
- Using this last equality and solving for α_k yields $\alpha_k = s_k^T r_k / s_k^T \mathbf{A} s_k$ (a way to compute the step length).
- How do we determine the search direction s_k?

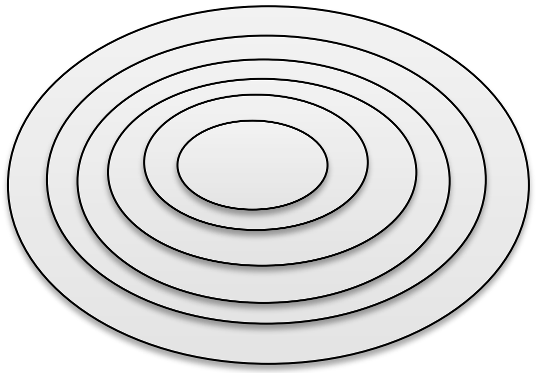
Conjugate Gradient Method

- Could choose $s_k = -\nabla f = r_k$ (i.e., direction of largest negative change in *f*(*x*)); this is called the method of *steepest descent*.
- How would it converge?
- Alternative approach: conjugate gradient with $s_{k+1}=r_{k+1} + \beta_k s_k$, where the constant β_k is chosen so that two successive s_k's are *conjugate*. This means $s_{k+1}^T \mathbf{A} s_k=0$.

THE UNIVERSITY of TENNESSEE UT KNOXVILLE
Big Orange. Big Ideas.

Conjugate Gradient Method

- CG preserves minimizations from previous iterations (no backtracking); not the case for steepest descent.
- Compare convergence paths:



Assuming **exact** arithmetic, CG would converge in at most n steps for an n-by-n linear system.

THE UNIVERSITY of TENNESSEE KNOXVILLE
Big Orange. Big Ideas.

Conjugate Gradient Method

- Since $s_{k+1}=\mathrm{r}_{k+1}+\beta_k s_k$ and we require $s_{k+1}^{\mathrm{T}}\mathbf{A}s_k=0$, then we must have $(r_{k+1}+\beta_k s_k)^{\mathrm{T}}\mathbf{A}s_k = 0$.
- Solving for β_k we obtain… $\beta_k = -r_{k+1}^T\mathbf{A}s_k / s_k^T\mathbf{A}s_k$ (and we can now advance the search direction).

- Initialization of CG algorithm:
  1) Choose x_0
  2) Compute $r_0=b-\mathbf{A}x_0$
  3) Choose $s_0=r_0$ (i.e., start with direction of steepest descent)

THE UNIVERSITY of TENNESSEE KNOXVILLE
Big Orange. Big Ideas.

Conjugate Gradient Method

- Main loop of CG algorithm:
  4) For $k = 0, 1, 2, \ldots$

     $\alpha_k = s_k^T r_k / s_k^T \mathbf{A} s_k$

     $x_{k+1} = \mathrm{x}_k + \alpha_k s_k$

     $\mathrm{r}_{k+1} = b - \mathbf{A}\, x_{k+1}$

     If $|r_{k+1}| \leq \varepsilon$, exit loop. ($\varepsilon$ is the error tolerance.)

     $\beta_k = -r_{k+1}^T \mathbf{A} s_k / s_k^T \mathbf{A} s_k$

     $s_{k+1} = \mathrm{r}_{k+1} + \beta_k s_k$
- Residual vectors $r_1, r_2, r_3, \ldots$ are *mutually orthogonal* (i.e., $r_i^T r_j = 0$ for $i \neq j$).

THE UNIVERSITY of TENNESSEE UT KNOXVILLE
Big Orange. Big Ideas.

Conjugate Gradient Method

- Since the n residual vectors $\{r_1, r_2, r_3, \ldots, r_n\}$ are *mutually orthogonal*, we know that $r_{n+1} = 0$. Why?
- This means that CG should theoretically converge in n iterations of the loop; typically obtain convergence in much less than n iterations.
- See `conjGrad.py` on pp.91-92 of textbook and review Example 2.18 on p.97.