Welcome to CS556!

Goals of the Course

- To obtain working knowledge of sparse and dense matrices
- To build intuition of Krylov methods from a projection approach
- To build intuition of Krylov methods from an efficiency perspective
- To develop an understanding of multigrid to a research level
- To construct and test a working collection of solvers
- To establish a practical understanding of iterative and multigrid methods

By the end of the course, you will be able to

- understand, for a given system or situation, which solver options will be *optimal* or *nearly optimal*
- estimate the performance difference for a variety of methods, so that you may choose the fastest
- estimate development overhead
- estimate communication costs in a parallel setting
- understand vectorization consequences

Text

- Main text: Iterative Methods for Sparse Linear Systems, Second Edition, by Yousef Saad.
- Supplement: Numerical Linear Algebra by Trefethen and Bau
- Supplement: Multigrid Tutorial by Briggs, Henson, McCormick
- Supplement: Iterative Methods by van der Vorst
- Course requirements: CS450
- Course knowledge: programming, basic numerical linear algebra, some numerical PDEs

Assessment (tentative)

- Quizzes 15
- Homework 50
- Midterm Project 15
- Final Project 15
- Attendance and handouts 5

Quizzes should be done on your own.

Weekly/bi-weekly homeworks should be done in teams of 2 (same team for the full term).

The projects will involve presentations to the class in teams of 2.

TOPICS—tentative schedule

Week 1–2:

- Course Overview
- Introduction / Examples
  - A first iterative example
  - A second iterative example: 1D Poisson
  - Grid-function evaluation as matrix-vector products
  - Stencils
  - Kronecker Product Introduction
- Gaussian elimination
  - The Geometry of Linear Systems
  - Full systems
  - Block Methods
  - Banded systems
- Saad Chapter 1, Linear Algebra Basics
  - Norms
  - Diagonally dominant matrices, irreducibly diagonally dominant
  - M matrices
  - Eigenvalues / eigenvectors / Schur form, etc.
  - Projection: 1D / n-D

Week 2–3:

HPC considerations (Saad Chapter 11.2)

- Pipelined/vectorized arithmetic
- Memory hierarchies (caches)
- Interpretive languages
- Examples that slow performance
- Examples with high performance

Saad Chapter 2: Discretization of PDEs

- Finite differences
  - Poisson / Helmholtz / Advection-diffusion 1D
  - Poisson 2D
  - Poisson 3D
  - Eigenvalues
- Finite elements
  - Poisson 2D / 3D

Direct Methods and the Curse of Dimensionality

Fast Poisson Solvers using Kronecker products

Week 4:

- Sparse Matrices
  - Graph representations
  - Sparse-matrix formats (CSR, etc.)
- Reordering and Sparse Direct Methods
  - Minimum degree ordering
  - Nested dissection ordering
  - A-conjugacy of ND orderings
  - Impact of reordering on matrix fill

Week 4–5: Basic Iterative Methods

- Jacobi, GS, SOR
- Jacobi vs GS: Poisson v. Advection-Diffusion
- Some convergence results
- ADI: Poisson / Helmholtz

Week 5-6: Projection Methods

Week 5: Conjugate Gradient Iteration

- Derivation and convergence rate
- Unpreconditioned variant
- Preconditioned variant (inc. Jacobi PCG example)
- Relationship to orthogonal polynomials
- Relationship to Steepest Descent
- Relationship to Lanczos iteration for eigenproblems
  - e.g., solving $f(A)\underline{x} = \underline{b}$
- Flexible CG

Week 6: GMRES

- Full GMRES
- Restarted GMRES
- Flexible GMRES
- Left/Right preconditioning
- Alternatives to GMRES (Saad Chapter 7)

Week 7–11: Preconditioning

- Block Jacobi
- Overlapping Schwarz (additive/multiplicative)
- Multilevel Schwarz
- Substructuring
- ILU

Week 11–13: Multigrid

Remark. We will introduce many of the topics throughout the course with the intent that the principal discussion and exploration of a given topic be covered in the proposed timeline. The overall aim of the course is for students to be familiar with a broad range of tools to efficiently solve large linear systems. Choosing the correct solver can save orders-of-magnitude in computational costs, so coverage beyond iterative methods is important. In particular, the choice of optimal preconditioning strategies warrants familiarity with multiple solution algebra methods.

CS556 Introduction

- In this course we will primarily be interested in *fast* solvers for linear systems of the form

$$A\underline{x} \;=\; \underline{b} \tag{1}$$

(or, often, $A\underline{u} = \underline{f}$).

- Specifically, we will be targeting very large $n \times n$ systems with $n = 10^5$–10^{10}, where A is generally *sparse.*

- In our context, A being **sparse** implies that the maximum (or average) number of nonzeros per row is bounded, *independent of* n.

- The total number of nonzeros is thus $\leq Cn$, for some relatively small constant C (e.g., $C \lesssim 5$–50) and the work to solve the system is thus potentially $O(n)$.

- Note that if A is a full $n \times n$ matrix with n^2 nonzeros, then the best complexity that one can hope for is $O(n^2)$ and, more generally, the complexity can be expected to be $O(n^3)$ if one needs to resort to Gaussian elimination (a.k.a., *direct factorization*).

- Many iterative methods realize $O(n)$ complexity for sparse matrices, which makes them very attractive (i.e., fast) compared to direct methods whose complexity is **superlinear in** n.

  (It turns out that the work and storage for direct factorization of the preceding tridiagonal matrix example is $\approx 8n$ operations and $\approx 4n$ storage, which is optimal $O(n)$ complexity with relatively low constants, so such systems are probably *not* candidates for iterative methods.)

- **A key criterion that dictates the choice of method is the "spatial dimension" of the governing PDE** (*at least for the classic discretizations of PDEs, but there are similar conditions on more general graphs or sparse matrices*).

- An example of a sparse system is the following tridiagonal system, which is a discrete form of the 1D Poisson equation.

$$A\underline{u} \;=\; \frac{1}{\Delta x^2}\begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & -1 & \ddots & \ddots & \\ & & \ddots & \ddots & -1 \\ & & & -1 & 2 \end{pmatrix}\begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ \vdots \\ u_n \end{pmatrix} \;=\; \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ \vdots \\ f_n \end{pmatrix}. \qquad (2)$$

- **Q:** How many nonzeros per row?
- **Q:** What is the total storage required?
- **Q:** *etc.*

Two Important Ideas

- A key ingredient for iterative methods is to be able to effect matrix-vector products in $O(n)$ time (i.e., the number of operations is $O(n)$ and the number of memory references is $O(n)$).

- A second ingredient is to have a scheme that is *rapidly convergent*, so that the number of iterations to reach a desired tolerance is bounded, **independent of** n (and hopefully small).

  We refer to this condition as *order-independent convergence*.

- Let's look at a couple of examples, starting with the second idea.

- Consider the system (from Van der Vorst),

$$A\underline{x} = \begin{bmatrix} 10 & 0 & 1 \\ \frac{1}{2} & 7 & 1 \\ 1 & 0 & 6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 21 \\ 9 \\ 8 \end{bmatrix}. \tag{3}$$

- If we perform one round of Gaussian elimination, we end up with the following *upper-triangular system*,

$$\begin{bmatrix} 10 & 0 & 1 \\ 0 & 7 & 1 - \frac{1}{20} \\ 0 & 0 & 6 - \frac{1}{10} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 21 \\ 9 - \frac{21}{20} \\ 8 - \frac{21}{10} \end{bmatrix}, \tag{4}$$

  which, from backwards substitution, has the solution

$$\underline{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix}. \tag{5}$$

- In this particular case, the solution works out to "nice" integer values.

- But the overall process is somewhat cumbersome and error prone (for humans), especially for large systems.*

- **Gauss** noted that, for *diagonally dominant systems*, one could easily approximate the solution by ignoring the off-diagonal elements of the matrix, e.g.,

$$\begin{bmatrix} 10 & 0 & 1 \\ \frac{1}{2} & 7 & 1 \\ 1 & 0 & 6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \approx \begin{bmatrix} 10 & 0 & 0 \\ 0 & 7 & 0 \\ 0 & 0 & 6 \end{bmatrix} \begin{bmatrix} \hat{x}_1 \\ \hat{x}_2 \\ \hat{x}_3 \end{bmatrix} \approx \begin{bmatrix} 21 \\ 9 \\ 8 \end{bmatrix} \tag{6}$$

$$\Longrightarrow \begin{bmatrix} \hat{x}_1 \\ \hat{x}_2 \\ \hat{x}_3 \end{bmatrix} = \begin{bmatrix} 2.1 \\ \frac{9}{7} \\ \frac{8}{6} \end{bmatrix} \approx \begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix}. \tag{7}$$

*Prof. Philip Davis at Brown University related that, during WWII, his wife was a "computor." She could solve a 5×5 system in a morning, but a 6×6 system would require a full day. This makes sense, since $n^3 = 125$ when $n = 5$, but is 216 when $n = 6$. Other authors report that an expert human could solve an 8×8 system in 8 hours. [Atanasoff, J. V., *Computing Machine for the Solution of Large Systems of Linear Algebraic Equations.*]

- A critical observation about (7) is that the vector $\underline{\hat{x}} = \begin{bmatrix} 2.1 & \frac{9}{7} & \frac{8}{6} \end{bmatrix}$ is *close* to the actual solution, $\underline{x}$.

- We denote the *error* as $\underline{e} := \underline{x} - \underline{\hat{x}}$ and the *residual* as

$$\underline{r} \;=\; A\underline{e} \;=\; A\underline{x} - A\underline{\hat{x}} \;=\; \underline{b} - A\underline{\hat{x}}. \tag{8}$$

- Note that the residual is **computable**, whereas the error is generally not because that would be equivalent to knowing the solution.

- In *test cases*, however, we can of course (and do) compute the error in order to understand the behavior of the method.

- Key points about the residual are the following:
  - $\underline{r}$ is computable.
  - $\underline{r}$ is a measure of the error (the only one we have) and $\underline{r} = 0$ when $\underline{e} = 0$.
  - $\underline{r} = A\underline{e}$, **always.**

- From the relationship $\underline{e} := \underline{x} - \underline{\hat{x}}$ we have

$$\underline{x} \;=\; \underline{\hat{x}} \;+\; \underline{e}. \tag{9}$$

- So, if we could compute $\underline{e}$, we would be able to find $\underline{x}$.

- From the (computable!) residual, we have

$$A\underline{e} \;=\; \underline{r} \;:=\; \underline{b} \;-\; A\underline{\hat{x}}. \tag{10}$$

  So, here we have a *new equation* for the *correction*, $\underline{e}$.

- As before, we can apply the diagonal approximation to A to generate an approximation to $\underline{e}$, which we'll denote as $\hat{\underline{e}}$.

$$\begin{bmatrix} 10 & 0 & 0 \\ 0 & 7 & 0 \\ 0 & 0 & 6 \end{bmatrix} \begin{bmatrix} \hat{e}_1 \\ \hat{e}_2 \\ \hat{e}_3 \end{bmatrix} = \begin{bmatrix} 21 \\ 9 \\ 8 \end{bmatrix} - \begin{bmatrix} 10 & 0 & 1 \\ \frac{1}{2} & 7 & 1 \\ 1 & 0 & 6 \end{bmatrix} \begin{bmatrix} 2.1 \\ \frac{9}{7} \\ \frac{8}{6} \end{bmatrix} = - \begin{bmatrix} 0 & 0 & 1 \\ \frac{1}{2} & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 2.1 \\ \frac{9}{7} \\ \frac{8}{6} \end{bmatrix}.$$

That is,

$$D\hat{\underline{e}} = \underline{b} - A\hat{\underline{e}} \tag{11}$$

$$\hat{\hat{\underline{x}}} = \hat{\underline{x}} + \hat{\underline{e}}, \tag{12}$$

where $D :=\text{diag}(A)$ is the diagonal matrix with entries a_{ii}.

- At this point, it becomes a bit tedious to continue by hand, so let's look at some example matlab (octave) code.

```
%% Matlab/OCTAVE SOURCE CODE:  demo_jac1.m

hdr

xe = [2; 1; 1];              %% Exact solution

A  = [ 10  0  1 ;            %% Test matrix
       .5  7  1 ;
        1  0  6 ];

b  = A*xe                    %% RHS

x=0*b; r=b; D=diag(diag(A));
for k=1:6; disp(' ')

    %%% ALGORITHM

    s=D\r;     % Guess of current error
    x=x+s;     % x = x + D\(b-Ax)
    r=r-A*s;   % Update residual ( := b-Ax)

    %%% DIAGNOSTICS

    e=xe-x;    % Error check for model problem
    format short;  disp(['iter = ' num2str(k)])
    format shorte; disp([ x e ])

end;
```

```
| >> demo_jac1
| b =
|    2.1000e+01
|    9.0000e+00
|    8.0000e+00
| iter = 1
|    2.1000e+00  -1.0000e-01
|    1.2857e+00  -2.8571e-01
|    1.3333e+00  -3.3333e-01
| iter = 2
|    1.9667e+00   3.3333e-02
|    9.4524e-01   5.4762e-02
|    9.8333e-01   1.6667e-02
| iter = 3
|    2.0017e+00  -1.6667e-03
|    1.0048e+00  -4.7619e-03
|    1.0056e+00  -5.5556e-03
| iter = 4
|    1.9994e+00   5.5556e-04
|    9.9909e-01   9.1270e-04
|    9.9972e-01   2.7778e-04
| iter = 5
|    2.0000e+00  -2.7778e-05
|    1.0001e+00  -7.9365e-05
|    1.0001e+00  -9.2593e-05
| iter = 6
|    2.0000e+00   9.2593e-06
|    9.9998e-01   1.5212e-05
|    1.0000e+00   4.6296e-06
```

- Note that the magnitude of $\underline{e}$ is reduced by almost a factor of 10 with each iteration.
- A principal concern throughout the term will be the rate of convergence:
  How fast does $\underline{e} \longrightarrow 0$*?*

1D Poisson Example

- As an example of the type of analysis to be covered this term we will consider Jacobi iteration (i.e., fixed-point iteration with a diagonal preconditioner) to the system,

$$A\underline{u} \;=\; \frac{1}{\Delta x^2}\begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & -1 & \ddots & \ddots & \\ & & \ddots & \ddots & -1 \\ & & & -1 & 2 \end{pmatrix}\begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ \vdots \\ u_n \end{pmatrix} \;=\; \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ \vdots \\ f_n \end{pmatrix}. \tag{13}$$

- With $\Delta x := L/N$, $N := n+1$, this system corresponds to a uniform-grid finite difference (or linear finite element) discretization of the 1D Poisson problem,

$$-\frac{d^2\tilde{u}}{dx^2} \;=\; f(x), \tag{14}$$

  with Dirichlet boundary conditions, $\tilde{u}(0) = \tilde{u}(L) = 0$.

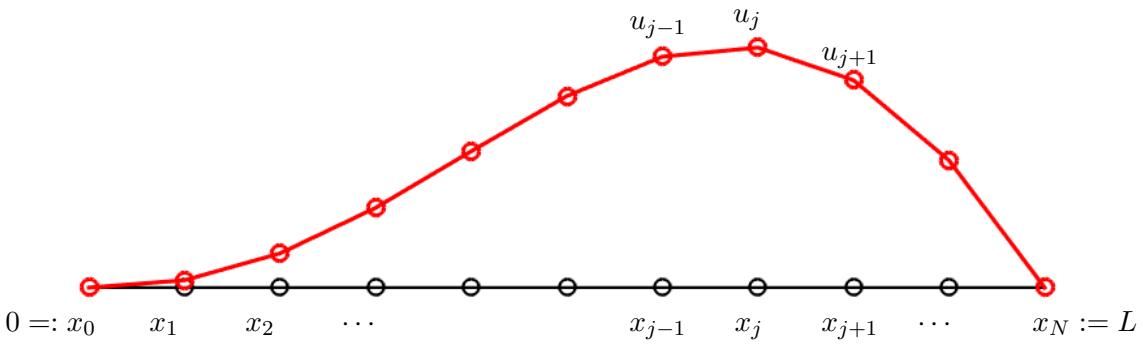

Figure 1: Finite difference grid on $\Omega := [0, L]$ with grid-spacing $\Delta x = L/N$.

- From a Taylor series about $x_j = j\Delta x$, we have

$$-\left.\frac{d^2\tilde{u}}{dx^2}\right|_{x_j} \;=\; -\frac{\tilde{u}_{j-1} - 2\tilde{u}_j + \tilde{u}_{j+1}}{\Delta x^2} \;+\; O(\Delta x^2) \;=\; f(x_j). \tag{15}$$

  If we drop the $O(\Delta x^2)$ we can solve for $u_j \approx \tilde{u}(x_j)$ and (correctly) anticipate an $O(\Delta x^2)$ error between the discrete solution, $\underline{u} = [u_1\ u_2\ \ldots\ u_n]^T$, and its continuous counterpart, $\underline{\tilde{u}} = [\tilde{u}(x_1)\ \tilde{u}(x_2) \ldots \tilde{u}(x_n)]^T$.

- With the boundary conditions $u_0 = 0$ and $u_{n+1} = 0$, applying (15) for $j = 1, \ldots, n$ corresponds to each row of (13).

- Our focus here is on solving the linear system (13), rather than (14), per se.

- *However,* the required accuracy (i.e., $\Delta x^2 <$ tol) *does* determine the size of the system, n, and the total amount of work.

- Let's begin with an analysis of the iteration error for Jacobi iteration applied to $A\underline{u} = \underline{b}$.
- Consider the following iteration,

$$\underline{u}_k \;=\; \underline{u}_{k-1} \;+\; D^{-1}(\underline{b} \;-\; A\underline{u}_{k-1}). \tag{16}$$

- Starting with $\underline{u}_0 = 0$ and $e_k := \underline{u} - \underline{u}_k$, we have $\underline{e}_0 = \underline{u}$, which is of course unknown.
- The important question is, *What happens to the error as we execute this iteration?*
- Let's start by looking at `demo_jac1a.m`.
  - Here, we plot the exact solution (black), the current iterate, $\underline{u}_k$ (blue), and the error, $\underline{e}_k = \underline{u} - \underline{u}_k$.
  - In this case, the error exhibits a slowly decaying mode whos shape is roughly independent of the initial error.

- We can derive an equation for the error by subtracting the iteration scheme from the expression $\underline{u} = \underline{u}$. (Note that $\underline{b} - A\underline{u} = 0$.)

$$\begin{array}{rrcll} - & \underline{u}_k & = & \underline{u}_{k-1} & + D^{-1}(\underline{b} - A\underline{u}_{k-1}) \\ + & \underline{u} & = & \underline{u} & + D^{-1}(\underline{b} - A\underline{u}) \\ \hline & \underline{e}_k & = & \underline{e}_{k-1} & + D^{-1}(0 - A\underline{e}_{k-1}) \\ & & = & \left(I - D^{-1}A\right)\underline{e}_{k-1} & = E\,\underline{e}_{k-1}. \end{array} \tag{17}$$

- Here, we have introduced the *error propagator*,

$$E := (I - D^{-1}A), \tag{18}$$

  which is the matrix that governs the behavior of the error.

- In particular,

$$\underline{e}_k = E\,\underline{e}_{k-1} = E^k\,\underline{e}_0 = E^k\,\underline{u}. \tag{19}$$

  - Notice that D could be any invertible matrix, but that it should ideally have two properties:
    - It should be easy to invert (i.e., to solve $D\underline{s} = \underline{r}$).
    - It should be a close approximation to A.
  - Regarding the latter point, its clear that if $D = A$ then $E = 0$ and the error is zero after just one iteration.
  - Such a choice, however, would not buy us anything.
  - The choice D =diag(A) (Jacobi iteration) is about the least expensive option given that one can precompute the inverse entries of D so that each iteration requires only pointwise multiplication of the residual when applying D^{-1}.

- Are the expressions (16) and (19) convergent?
- How rapidly?
- Two ways to measure:
  - vector/matrix norms
  - eigenvalues

  We will do both, but today we'll start with eigenvalues, making simplifying assumptions where necessary.
- Suppose that E has a full set of linearly-independent eigenvectors, $\underline{z}_j$ with associated eigenvalues, λ_j:

$$E\underline{z}_k \;=\; \underline{z}_k\lambda_k, \quad Z = [\underline{z}_1 \cdots \underline{z}_n] \Longrightarrow Z^{-1} \text{ exists.} \tag{20}$$

- Consequently, *for any* $\underline{u} \in \mathbb{R}^n$,, we can find a vector $\underline{\hat{u}} = [\hat{u}_1 \ldots \hat{u}_n]^T$,

$$\underline{\hat{u}} \;:=\; Z^{-1}\underline{u} \tag{21}$$

  such that

$$\underline{u} \;=\; \sum_{j=1}^{n} \hat{u}_j \underline{z}_j. \tag{22}$$

- That is, the solution is a linear combination of the eigenvectors of E.
- We can assume that the eigenvectors are normalized to have unit norm, so that the norm of $\underline{u}$ is related to the magnitude of the coefficients $\hat{u}_j$.
  (Only if the eigenvectors are *orthogonal*, $\underline{z}_i^T \underline{z}_j = \delta_{ij}$, will we have $\|\underline{u}\| = \sum_j |\hat{u}_j|^2$. Every symmetric matrix has a full set of orthogonal eigenvectors, so this case is rather common.)

- Given that $\underline{e}_0 = \underline{u}$, we have

$$\underline{e}_0 \;=\; \hat{u}_1\,\underline{z}_1 \;+\; \hat{u}_2\,\underline{z}_2 \;+\; \cdots \;+ \hat{u}_n\,\underline{z}_n \tag{23}$$

$$\underline{e}_1 \;=\; E\underline{e}_0 \;=\; \hat{u}_1\lambda_1\,\underline{z}_1 \;+\; \hat{u}_2\lambda_2\,\underline{z}_2 \;+\; \cdots \;+ \hat{u}_n\lambda_n\,\underline{z}_n \tag{24}$$

$$\underline{e}_2 \;=\; E\underline{e}_1 \;=\; \hat{u}_1\lambda_1^2\,\underline{z}_1 \;+\; \hat{u}_2\lambda_2^2\,\underline{z}_2 \;+\; \cdots \;+ \hat{u}_n\lambda_n^2\,\underline{z}_n \tag{25}$$

$$\underline{e}_k \;=\; E^k\underline{e}_0 \;=\; \hat{u}_1\lambda_1^k\,\underline{z}_1 \;+\; \hat{u}_2\lambda_2^k\,\underline{z}_2 \;+\; \cdots \;+ \hat{u}_n\lambda_n^k\,\underline{z}_n \tag{26}$$

- Clearly, if all $|\lambda_j| < 1$ the error will tend towards 0.
- Suppose $|\lambda_n| > |\lambda_j|$, $j \neq n$,

$$\underline{e}_k \;=\; \lambda_n^k \left[\hat{u}_n\,\underline{z}_n \;+\; \sum_{j=1}^{n-1}\left(\frac{\lambda_j}{\lambda_n}\right)^k \hat{u}_j\,\underline{z}_j\right] \tag{27}$$

- As $k \longrightarrow \infty$,

$$\underline{e}_k \;\sim\; \lambda_n^k\,(\hat{u}_n\,\underline{z}_n \;+\; 0)\,. \tag{28}$$

- We define the *spectral radius* of E as

$$\rho(E) \;=\; \max_j |\lambda_j|\,. \tag{29}$$

- The fixed-point iteration will converge iff $\rho(E) < 1$.