Unit 5 – Iterative Methods

Direct vs. Iterative Methods
- Previously when looking at systems of linear equations, we introduced direct methods like Gaussian Elimination and the Cholesky factorization
- The other kind of method is called an iterative method where one starts with an initial guess $x_1$ and iterates through a sequence $x_1, x_2, x_3, \ldots$ ending up with a final guess of $x_m$
- Issues include both finding an initial guess $x_1$ and deciding on a stopping criterion that says that $x_m$, for some $m$, is good enough
- We stop when the error, $e_k = x_k - x_{\text{exact}}$, is small, i.e. when $\|e_k\| < \varepsilon$ (for some small epsilon)

Sparse vs. Dense Matrices
- Often times matrices are very large and very sparse
- Example:
  - Imagine creating a discretization of the air in the room for a Poisson partial differential equation (PDE) governing air flow
  - One may want to solve for the pressure at a resolution of a few hundred samples in each direction, e.g. $200 \times 200 \times 200 = 8$ million unknowns
  - The resulting matrix, 8 million by 8 million, only has non-zero entries between neighboring elements (left/right, top/bottom, front/back, i.e. 7 per row)
  - Thus, $7 \times (8 \text{ million})$ nonzero entries out of $(8 \text{ million}) \times (8 \text{ million})$
  - $64 \text{ trillion} - 56 \text{ million} = \text{ basically } 64 \text{ trillion zeros (almost all zeros!)}$
- One doesn’t want to store the entire matrix, or compute on it
- Thus, there are many sparse storage and sparse solvers methods
- Iterative methods are great for sparse matrices, because they can often be formulated through the matrix’s action on a vector
  - And the zero entries can thus be totally ignored
- Director solvers are more commonly used on dense matrices
Residual

- One can plug any guess at the unknowns $x$ back into the equation $Ax=b$ in order to see how good the guess to the solution is
  - We call this accuracy approximation the residual, $r=b-Ax$
  - It shows how close one is to satisfying the desired equations
- Another way to think about the residual is $r = b - Ax = Ax_{\text{exact}} - Ax = A(x_{\text{exact}} - x) = -Ae$
  - Thus, the residual is the error transformed by $A$ into the space where $b$ resides
- Suppose $A=[a]$ simply maps from 1D to 1D like a linear function
  - Then $Ax=b$ has solution $x_{\text{exact}}=b/a$, error $x-b/a$, and residual $b-ax$
    - Note that $-Ae=-ae=-ax+b=r$
  - The distance an $x$ value guess is horizontally from $x_{\text{exact}}=b/a$ represents the error
  - The distance $ax$ is from $b$ vertically represents the residual
  - We can plot the residual as a straight line $y=b-ax$ with slope $-a$ and $y$-intercept $b$
  - Then for small $a$, the residual changes very little as the $x$-value changes
    - I.e., large errors have much smaller (even small?) residuals
  - Conversely, for large $a$ (large slope), small residuals guarantee much smaller errors
- More generally, we can use the SVD of $A=UDV^T$ to rewrite $Ax=b$ as $D(V^T x) = U^T b$
  - Orthogonal $U$ and $V$ preserve metrics, so we can equivalently think of this as a bunch of decoupled scalar problems $D x^\text{hat} = b^\text{hat}$
  - Then small singular values lead to small residuals with possibly large errors
  - And large singular values have residuals that over-approximate the error
- **POINT** – residual doesn’t do a good job estimating the error for small singular values
  - Small residual could still have large error
Steepest Descent

- Starting from the current guess, we go some distance $\alpha_k$ in the current search direction $s_k$ to get a new $x_{k+1} = x_k + s_k \alpha_k$
- $\alpha_k$ is chosen so that all remaining error is orthogonal to the search direction, i.e. $e_{k+1} \cdot s_k = 0$, or $r_{k+1} \cdot s_k = 0$ (using $r=Ae$)
  - $x_{k+1} - x_{\text{exact}} = x_k - x_{\text{exact}} + s_k \alpha_k$ gives $e_{k+1} = e_k + s_k \alpha_k$
  - so $A(e_k + s_k \alpha_k) \cdot s_k = 0$ yields $\alpha_k = -\frac{s_k \cdot Ae_k}{s_k \cdot As_k}$
- Steepest Descent uses the search direction that appears to be going “downhill” the fastest, which means $s_k$ is chosen as the residual $r_k = b - Ax_k$
- Summary $r_k = b - Ax_k$, $\alpha_k = \frac{r_k \cdot r_k}{r_k \cdot Ar_k}$, $x_{k+1} = x_k + r_k \alpha_k$
  - Iterate until $r_k$ is “small” (prefer the infinity norm!)

Alternately, compute $r_k = b - Ax_k = b - A(x_{k-1} + s_{k-1} \alpha_{k-1}) = r_{k-1} - \alpha_{k-1}Ar_{k-1}$ so that multiplication by $A$ only happens once an iteration (just $Ar$, instead of $Ar$ and $Ax$)
- Problem with steepest decent is that it repeatedly uses the same search directions, and thus takes a long time to converge

Conjugate Gradients (CG) Method

- Amazingly quick/efficient and robust method for SPD systems
- Converges in $n$-steps for an $nxn$ matrix $A$
  - Actually, converges in the number of steps equal to the number of distinct eigenvalues
  - Thus, preconditioning makes a big difference, if it can coalesce eigenvalues
- Motivation: choose search directions *orthogonal* to each other so that one need not search in the same directions over and over as Steepest Descent ineffectually does
  - Difficult to do this!
- Instead: choose search directions to be $A$-orthogonal
  - That is, $s_j \cdot As_k = 0$ for $j \neq k$ (instead of orthogonal with $s_j \cdot s_k = 0$)
  - Motivated by $r=Ae$

Gram-Schmidt

- Orthogonalizes the columns of a matrix
- For each column, subtract its dot product overlap with all prior columns
  - This makes it orthogonal to all prior columns
- A-orthogonal Gram-Schmidt simply replaces all dot products with $A$-weighted dot products
- given a vector $V_k$, construct $s_k$ by subtracting out the “A-overlap” of $V_k$ with $s_1$ to $s_{k-1}$ so that $s_k \cdot As_i = 0$ for $i=1, k-1$
- i.e. $s_k = V_k - \sum_{j=1}^{k-1} \frac{V_k \cdot As_j}{s_j \cdot As_j} s_j$ where the non-normalized $s_j$ require division by their squared norm (the $A$-weighted dot product of $s_j$ with itself)
Error Analysis

- Represent the initial error in the basis of search directions $e_0 = \sum_j a_j s_j$
- $s_k \cdot Ae_0 = s_k \cdot A \sum_j a_j s_j = \sum_j a_j s_k \cdot As_j = a_k s_k \cdot As_k$ since the search directions are A-orthogonal; thus $a_k = \frac{s_k \cdot Ae_0}{s_k \cdot As_k} = \frac{s_k \cdot A(e_0 + \sum_{j=1}^{k-1} \alpha_j s_j)}{s_k \cdot As_k}$ where the summation is identically zero when multiplied by $s_k \cdot A$.
- Then $e_{k+1} = e_k + s_k \alpha_k$ applied recursively yields $e_k = e_0 + \sum_{j=1}^{k-1} \alpha_j s_j$ so that $a_k = \frac{s_k \cdot Ae_k}{s_k \cdot As_k}$ and $a_k = -\alpha_k$.
- That is, $e_0 = -\sum_j \alpha_j s_j$ and $e_k = -\sum_j \alpha_j s_j + \sum_{j=1}^{k-1} \alpha_j s_j$
- After n steps, the second term is equal to the first term and the error is zero.
- Aside: $s_i \cdot Ae_i = -\sum_j \alpha_j s_i \cdot As_j + \sum_{j=1}^{k-1} \alpha_j s_i \cdot As_j$
  - For $i < k$, the single nonzero term in each sum cancels, so $s_i \cdot Ae_i = 0$ and $s_i \cdot r_k = 0$ meaning that the residual is orthogonal to all previous search directions.

Conjugate Gradients (CG) Method

- Use the residuals as *candidate* search directions, $V_k = r_k$, but make them A-orthogonal to all prior search directions with Gram-Schmidt.
- For $i \geq k$, $s_k \cdot r_i = V_k \cdot r_i - \sum_{j=1}^{k-1} \frac{V_k \cdot As_j}{s_j \cdot As_j} s_j \cdot r_i = V_k \cdot r_i$, where the summation vanishes because the residual is orthogonal to all previous search directions.
  - when $i=k$, this gives $\alpha_k = \frac{s_k \cdot r_k}{s_k \cdot As_k} = \frac{V_k \cdot r_k}{s_k \cdot As_k} = \frac{r_k \cdot r_k}{s_k \cdot As_k}$.
  - when $i > k$, $0 = V_k \cdot r_i = r_k \cdot As_k \cdot r_i$, i.e. $r_i$ is orthogonal to all previous $V_k$ and $r_k$ as well.
- Starting with the recursion $r_k = r_{k-1} - \alpha_{k-1} As_{k-1}$:
  - When $i > k$, we have $\alpha_{k-1} r_i \cdot As_{k-1} = r_i \cdot r_{k-1} - r_i \cdot r_k = 0$.
  - When $i=k$, we have $\alpha_{k-1} r_k \cdot As_{k-1} = r_k \cdot r_{k-1} - r_k \cdot r_k = -r_k \cdot r_k$.
  - So, $s_k = r_k - \sum_{j=1}^{k-1} \frac{r_k \cdot As_j}{s_j \cdot As_j} s_j = r_k + \frac{r_k \cdot r_k}{\alpha_{k-1} (s_{k-1} \cdot As_{k-1})} s_{k-1}$, since only the last term in the sum is nonzero.
  - Plugging in the definition of $\alpha_{k-1}$ gives $s_k = r_k + \frac{r_k \cdot r_k}{r_{k-1} \cdot r_{k-1}} s_{k-1}$.
- Summary: $s_1 = r_1 = b - Ax_1$, iterate: $\alpha_k = \frac{r_k \cdot r_k}{s_k \cdot As_k}$, $x_{k+1} = x_k + \alpha_k s_k$ (as usual), $r_{k+1} = r_k - \alpha_k As_k$ (as usual), $s_{k+1} = r_{k+1} + \frac{r_{k+1} \cdot r_{k+1}}{r_k \cdot r_k} s_k$. 
Non-Symmetric Matrices

- GMRES, MINRES, BiCGSTAB
- Generally speaking, these methods are less stable, more error prone, and slower than CG