Lecture 14

Finish Iterative Methods & Cleanup

Owen L. Lewis

Department of Mathematics and Statistics
University of New Mexico

Oct. 3, 2024

Goals for today. . .

- Revisit Jacobi and Gauss Seidel
- Theorem for convergence.
- SOR
- Conjugate Gradient.
- Wrap-up linear solutions methods.
- Begin interpolation (maybe)

Iterative schemes for Linear equations

The goal is to solve

$$Ax = b,$$

but avoid calculating A^{-1} or the product $A^{-1}b$. The basic idea is we come up with some Q such that $Q^{-1} \approx A^{-1}$ and $Q^{-1}b$ is cheap/easy to compute.

Then we just make a guess $x^{(0)}$ and iterate via the scheme

$$x^{(k)} = x^{(k-1)} + Q^{-1} r^{(k-1)}.$$

Hopefully $x^{(k)} \to x^*$ (the true solution) as $k \to \infty$.

Two Popular Choices

Let $A = D - C_U - C_L$, where C_U and C_L are the negative of the strictly upper and lower triangular, respectively, of A.

Example

Jacobi iteration approximates A with $Q = D$.

$$x^{(k)} = x^{(k-1)} + D^{-1} r^{(k-1)}.$$

Example

Gauss-Seidel iteration approximates A with $Q = D - C_U$.

$$x^{(k)} = x^{(k-1)} + \left(D - C_U\right)^{-1} r^{(k-1)}.$$

Again, why do Jacobi and Gauss-Seidel work?

Jacobi, Gauss-Seidel (sufficient) Convergence Theorem

If A is diagonally dominant, then the Jacobi and Gauss-Seidel methods converge for any initial guess $x^{(0)}$.

Definition: Diagonal Dominance

A matrix is *diagonally dominant* if

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

for all i.

Notes

Smart Jacobi Algorithm

The Jacobi algorithm uses the matrix representation:

$$x^{(k)} = D^{-1}(C_L + C_U)x^{(k-1)} + D^{-1}b$$

Or componentwise,

$$x_i^{(k)} = -\sum_{j=1, j\neq i}^{n} \left(\frac{a_{ij}}{a_{ii}}\right) x_j^{(k-1)} + \frac{b_i}{a_{ii}}$$

So each sweep (from $k-1$ to k) uses $\mathcal{O}(n)$ operations per vector element (potentially $\mathcal{O}(n^2)$ total).

If, for each row i, $a_{ij} \neq 0$ for *at most m* values, each sweep uses $\mathcal{O}(mn)$ operations.

Smart Gauss-Seidel Algorithm

The Gauss-Seidel algorithm uses the matrix representation:

$$x^{(k)} = (D - C_L)^{-1} C_U x^{(k-1)} + (D - C_L)^{-1} b$$

Component-wise:

$$x_i^{(k)} = -\sum_{j=1}^{i-1} \left(\frac{a_{ij}}{a_{ii}}\right) x_j^{(k)} - \sum_{j=i+1}^{n} \left(\frac{a_{ij}}{a_{ii}}\right) x_j^{(k-1)} + \frac{b_i}{a_{ii}}$$

So again each sweep (from $k-1$ to k) uses $\mathcal{O}(n)$ operations per vector element.
If, for each row i, $a_{ij} \neq 0$ for *at most* m values, each sweep uses $\mathcal{O}(mn)$ operations.

The difference is that in the Jacobi method, updates are saved (and not used) in a new vector. With Gauss-Seidel, an update to an element $x_i^{(k)}$ is used immediately.

Example:

Lets do Jacobi and Gauss-Seidel:

Example

Consider

$$A = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 3 & -1 \\ 0 & -1 & 2 \end{bmatrix}, \qquad b = \begin{bmatrix} 1 \\ 8 \\ -5 \end{bmatrix}, \qquad x^{(0)} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

See `iterative_methods.m`

Successive Over-Relaxation

The Gauss-Seidel algorithm uses the matrix representation:

$$x^{(k)} = x^{(k-1)} + (D - C_L)^{-1} r^{(k-1)}$$

What if one step moves us in the “right” direction, but not far enough?
Take a step of a different size!

$$x^{(k)} = x^{(k-1)} + \omega (D - C_L)^{-1} r^{(k-1)}$$

Can also be written as

$$x^{(k)} = \Big(\omega (D - C_L)^{-1} C_U + (1 - \omega) \mathbf{I} \Big) x^{(k-1)} + (D - C_L)^{-1} b$$

If $\omega = 1$, this collapses back to Gauss-Seidel.
If $\omega > 1$, this is called “over-relaxation”.

What if A has special structure?

- Suppose that A is $n \times n$ symmetric and positive definite.
- Just like factorizations, there are special iterative methods that take advantage of this special property

Conjugate Gradient!

Notes

Conjugate Gradients

- Suppose that A is $n \times n$ symmetric and positive definite.
- Since A is positive definite, $x^T A x > 0$ for all $x \in \mathbb{R}^n$.
- Define a quadratic function

$$\phi(x) = \frac{1}{2} x^T A x - x^T b$$

- It turns out that $-\nabla\phi = b - Ax = r$, or $\phi(x)$ has a minimum for x such that $Ax = b$.
- Optimization methods look in a "search direction" and pick the best step:

$$x_{k+1} = x_k + \alpha s_k$$

  Choose α so that $\phi(x_k + \alpha s_k)$ is minimized in the direction of s_k.
- Find α so that ϕ is minimized:

$$0 = \frac{d}{d\alpha}\phi(x_{k+1}) = \nabla\phi(x_{k+1})^T \frac{d}{d\alpha} x_{k+1} = -r_{k+1}^T \frac{d}{d\alpha}(x_k + \alpha s_k) = -r_{k+1}^T s_k.$$

Conjugate Gradients

- Find α so that ϕ is minimized:

$$0 = \frac{d}{d\alpha}\phi(x_{k+1}) = \nabla\phi(x_{k+1})^T \frac{d}{d\alpha} x_{k+1} = -r_{k+1}^T \frac{d}{d\alpha}(x_k + \alpha s_k) = -r_{k+1}^T s_k .$$

- We also know

$$r_{k+1} = b - Ax_{k+1} = b - A(x_k + \alpha s_k) = r_k - \alpha A s_k$$

- So, the optimal search parameter is

$$\alpha = -\frac{r_k^T s_k}{s_k^T A s_k}$$

- This is CG: take a step in a search direction

Conjugate Gradients

- Neat trick: We can compute the r without explicitly forming $b - Ax$:

$$r_{k+1} = b - Ax_{k+1} = b - A(x_k + \alpha s_k) = b - Ax_k - \alpha As_k = r_k - \alpha As_k$$

Conjugate Gradients

- How should we pick s_k?
- Note that $-\nabla\phi = b - Ax = r$, so r is the gradient of ϕ (for *any* x), and this is a good direction.
- Thus, pick $s_0 = r = b - Ax_0$.
- What is s_1? This should be in the direction of r_1, but *conjugate* to s_0: $s_1^T A s_0 = 0$.
- (Two vectors u and v are A*-conjugate* if $u^T A v = 0$)
- So, if we let $s_1 = r_1 + \beta s_0$, we can require

$$0 = s_1^T A s_0 = (r_1^T + \beta s_0^T) A s_0 = r_1^T A s_0 + \beta s_0^T A s_0$$

or

$$\beta = -r_1^T A s_0 / s_0^T A s_0.$$

- Holds for s_{k+1} in terms of $r_{k+1} + \beta_{k+1} s_k$
- Further similification (which is *not* simple to carry out) yields a simple method that requires only one matrix-vector product per step:

Conjugate Gradients

```
$x_0 =$ initial guess
$r_0 = b - Ax_0$
$s_0 = r_0$
for $k = 0, 1, 2, \ldots$
    $\alpha_k = \frac{r_k^T r_k}{s_k^T A s_k}$
    $x_{k+1} = x_k + \alpha_k s_k$
    $r_{k+1} = r_k - \alpha_k A s_k$
    $\beta_{k+1} = r_{k+1}^T r_{k+1} / r_k^T r_k$
    $s_{k+1} = r_{k+1} + \beta_{k+1} s_k$
```

See `iterative_methods.m`

Facts about CG

- Each step requires a single matrix-vector multiplication $\mathcal{O}(n^2)$.
- At the step k, $r^{(k)} \cdot r^{(j)} = 0$ for all $j < k$. $\Rightarrow$ the residual is orthogonal to all the residuals at previous steps!
- This means that after n steps, $r^{(n)} = 0$!
- Conj. Grad. is guaranteed to get the solution in n steps. At that point we've done a total of $\mathcal{O}(n^3)$.
- Is Conj. Grad. a "direct" method like Gaussian Elimination? Or is it iterative?
- If $cond(A)$ is "nice", then CG will generally produce tiny tiny residuals after a small number of steps.

Changing Gears

Looping back to something we skipped. . .

More Algorithms for Special Systems

- tridiagonal systems
- banded systems

Tridiagonal

A tridiagonal matrix A

$$\begin{bmatrix} d_1 & c_1 & & & & & & \\ a_1 & d_2 & c_2 & & & & & \\ & a_2 & d_3 & c_3 & & & & \\ & & \dots & \dots & \dots & & & \\ & & & a_{i-1} & d_i & c_i & & \\ & & & & \dots & \dots & \dots & \\ & & & & \dots & \dots & \dots & \\ & & & & & & a_{n-1} & d_n \end{bmatrix}$$

- storage is saved by not saving zeros
- only $n + 2(n-1) = 3n - 2$ places are needed to store the matrix (i.e., $O(n)$ storage) versus n^2 storage for dense system
- can operations be saved? yes!

Tridiagonal

$$\begin{bmatrix}
d_1 & c_1 & & & & & & \\
a_1 & d_2 & c_2 & & & & & \\
 & a_2 & d_3 & c_3 & & & & \\
 & & \dots & \dots & \dots & & & \\
 & & & a_{i-1} & d_i & c_i & & \\
 & & & & \dots & \dots & \dots & \\
 & & & & \dots & \dots & \dots & \\
 & & & & & & a_{n-1} & d_n
\end{bmatrix}$$

Start forward elimination (without any special pivoting)

1. subtract a_1/d_1 times row 1 from row 2
2. this eliminates a_1, changes d_2 and does not touch c_2
3. continuing:

$$d_i = d_i - \left(\frac{a_{i-1}}{d_{i-1}} c_{i-1} \right)$$

for $i = 2 \dots n$

Tridiagonal

$$\begin{bmatrix} \tilde{d}_1 & c_1 & & & & & & \\ & \tilde{d}_2 & c_2 & & & & & \\ & & \tilde{d}_3 & c_3 & & & & \\ & & & \cdots & \cdots & & & \\ & & & & \tilde{d}_i & c_i & & \\ & & & & & \cdots & \cdots & \\ & & & & & \cdots & \cdots & \\ & & & & & & & \tilde{d}_n \end{bmatrix}$$

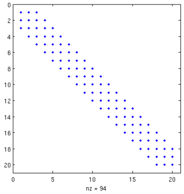
This leaves an upper triangular (2-band). With back substitution:

1. $x_n = \tilde{b}_n / \tilde{d}_n$
2. $x_{n-1} = (1/\tilde{d}_{n-1})(\tilde{b}_{n-1} - c_{n-1} x_n)$
3. $x_i = (1/\tilde{d}_i)(\tilde{b}_i - c_i x_{i+1})$

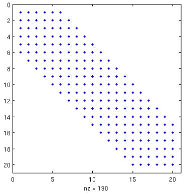
Tridiagonal Algorithm

```
input: $n, a, d, c, b$
for $i = 2$ to $n$
    $xmult = a_{i-1}/d_{i-1}$
    $d_i = d_i - xmult \cdot c_{i-1}$
    $b_i = b_i - xmult \cdot b_{i-1}$
end
$x_n = b_n/d_n$
for $i = n-1$ down to 1
    $x_i = (b_i - c_i x_{i+1})/d_i$
end
```

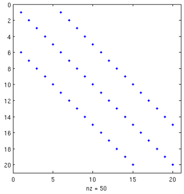
m-band



$m = 5$



$m = 11$



$m = 11$

- the m correspond to the total width of the non-zeros
- after a few passes of GE *fill-in* with occur within the band
- so an empty band costs (about) the same as a non-empty band
- one fix: reordering (e.g. Cuthill-McKee)
- generally GE will cost $\mathcal{O}(m^2 n)$ for m-band systems

Changing Gears

And now for something completely different. . .

Next Topics

1. Interpolation: Approximating a function $f(x)$ by a polynomial $p_n(x)$.
2. Least Squares: More linear algebra!
3. Differentiation: Approximating the derivative of a function $f(x)$.
4. Integration: Approximating an integral $\int_a^b f(x)\,dx$

Interpolation: Introduction

Objective

Approximate an unknown function $f(x)$ by an “easier” function $g(x)$ (perhaps a polynomial?).

Objective (alt)

Approximate some data by a function $g(x)$.

Types of approximating functions:

1. Polynomials
2. Piecewise polynomials
3. Rational functions
4. Trig functions
5. Others (inverse, exponential, Bessel, etc)

Interpolation: Introduction

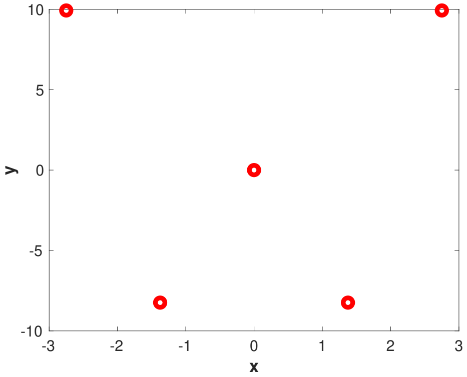
How do we approximate $f(x)$ by $g(x)$? In what sense is the approximation a good one?

1. Interpolation: $g(x)$ must have the same values of $f(x)$ at set of given points.
2. Least-squares: $g(x)$ must deviate as little as possible from $f(x)$ in the sense of a 2-norm: minimize $\int_a^b |f(t) - g(t)|^2 \, dt$
3. Chebyshev: $g(x)$ must deviate as little as possible from $f(x)$ in the sense of the ∞-norm: minimize $\max_{t \in [a,b]} |f(t) - g(t)|$.

Interpolation: Introduction

Given $n+1$ distinct points $x_0, \dots, x_n$, and values $y_0, \dots, y_n$, find a polynomial $p(x)$ of degree n so that

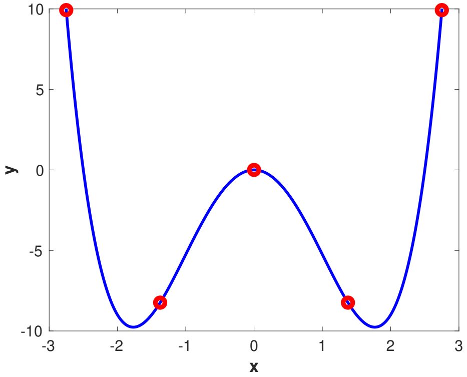
$$p(x_i) = y_i \quad i = 0, \dots, n$$



Interpolation: Introduction

Given $n + 1$ distinct points $x_0, \ldots, x_n$, and values $y_0, \ldots, y_n$, find a polynomial $p(x)$ of degree n so that

$$p(x_i) = y_i \quad i = 0, \ldots, n$$



Interpolation: Introduction

Given $n+1$ distinct points $x_0, \dots, x_n$, and values $y_0, \dots, y_n$, find a polynomial $p(x)$ of degree n so that

$$p(x_i) = y_i \quad i = 0, \dots, n$$

- A polynomial of degree n has $n+1$ degrees-of-freedom:

$$p(x) = a_0 + a_1 x + \cdots + a_n x^n$$

- $n+1$ constraints determine the polynomial uniquely:

$$p(x_i) = y_i, \quad i = 0, \dots, n$$

Theorem (page 142 1stEd)

If points $x_0, \dots, x_n$ are distinct, then for arbitrary $y_0, \dots, y_n$, there is a *unique* polynomial $p(x)$ of degree at most n such that $p(x_i) = y_i$ for $i = 0, \dots, n$.

How can you prove the polynomial is unique? (Hint: What if it isn't?)

Monomials

Obvious attempt: try picking

$$p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

So for each x_i we have

$$p(x_i) = a_0 + a_1 x_i + a_2 x_i^2 + \cdots + a_n x_i^n = y_i$$

OR

$$\begin{aligned}
a_0 + a_1 x_0 + a_2 x_0^2 + \cdots + a_n x_0^n &= y_0 \\
a_0 + a_1 x_1 + a_2 x_1^2 + \cdots + a_n x_1^n &= y_1 \\
a_0 + a_1 x_2 + a_2 x_2^2 + \cdots + a_n x_2^n &= y_2 \\
a_0 + a_1 x_3 + a_2 x_3^2 + \cdots + a_n x_3^n &= y_3 \\
&\vdots \\
a_0 + a_1 x_n + a_2 x_n^2 + \cdots + a_n x_n^n &= y_n
\end{aligned}$$

Monomial: The problem

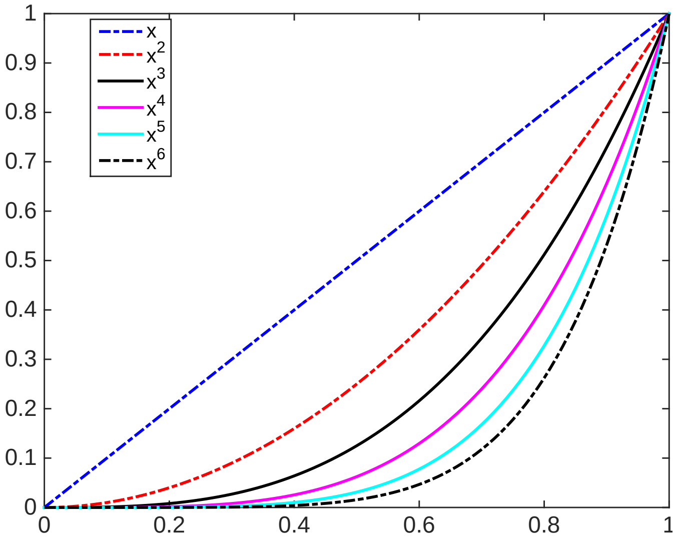
$$\begin{bmatrix} 1 & x_0 & x_0^2 & \dots & x_0^n \\ 1 & x_1 & x_1^2 & \dots & x_1^n \\ 1 & x_2 & x_2^2 & \dots & x_2^n \\ & & & \vdots & \\ 1 & x_n & x_n^2 & \dots & x_n^n \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

The matrix above is known as the Vandermonde matrix.

Question

- Is this a “good” system to solve?

Ill-Conditioning!


x
x^2
x^3
x^4
x^5
x^6
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0
0.2
0.4
0.6
0.8
1