Solving Linear Systems: Iterative Methods and Sparse Systems

COS 323

Direct vs. Iterative Methods

- So far, have looked at *direct methods* for solving linear systems
  - Predictable number of steps
  - No answer until the very end
- Alternative: *iterative methods*
  - Start with approximate answer
  - Each iteration improves accuracy
  - Stop once estimated error below tolerance

Benefits of Iterative Algorithms

- Some iterative algorithms designed for accuracy:
  - Direct methods subject to roundoff error
  - Iterate to reduce error to $O(\varepsilon)$
- Some algorithms produce answer faster
  - Most important class: *sparse matrix* solvers
  - Speed depends on # of *nonzero* elements, not total # of elements
- Today: iterative improvement of accuracy, solving sparse systems (not necessarily iteratively)

Iterative Improvement

- Suppose you've solved (or think you've solved) some system $Ax=b$
- Can check answer by computing *residual*:

$$r = b - Ax_{computed}$$

- If r is small (compared to b), x is accurate
- What if it's not?

Iterative Improvement

- Large residual caused by error in x:

$$e = x_{correct} - x_{computed}$$

- If we knew the error, could try to improve x:

$$x_{correct} = x_{computed} + e$$

- Solve for error:

$$Ax_{computed} = A(x_{correct} - e) = b - r$$
$$Ax_{correct} - Ae = b - r$$
$$Ae = r$$

Iterative Improvement

- So, compute residual, solve for e,
  and apply correction to estimate of x
- If original system solved using LU,
  this is relatively fast (relative to $O(n^3)$, that is):
  - $O(n^2)$ matrix/vector multiplication +
    $O(n)$ vector subtraction to solve for r
  - $O(n^2)$ forward/backsubstitution to solve for e
  - $O(n)$ vector addition to correct estimate of x

Sparse Systems

- Many applications require solution of large linear systems (n = thousands to millions)
  - Local constraints or interactions: most entries are 0
  - Wasteful to store all n^2 entries
  - Difficult or impossible to use $O(n^3)$ algorithms
- Goal: solve system with:
  - Storage proportional to # of *nonzero* elements
  - Running time $<< n^3$

Special Case: Band Diagonal

- Last time: tridiagonal (or band diagonal) systems
  - Storage O(n): only relevant diagonals
  - Time O(n): Gaussian elimination with bookkeeping

Cyclic Tridiagonal

- Interesting extension: cyclic tridiagonal

$$\begin{bmatrix} a_{11} & a_{12} & & & & a_{16} \\ a_{21} & a_{22} & a_{23} & & & \\ & a_{32} & a_{33} & a_{34} & & \\ & & a_{43} & a_{44} & a_{45} & \\ & & & a_{54} & a_{55} & a_{56} \\ a_{61} & & & & a_{65} & a_{66} \end{bmatrix} x = b$$

- Could derive yet another special case algorithm, but there's a better way

Updating Inverse

- Suppose we have some fast way of finding A^{-1} for some matrix A
- Now A changes in a special way:

$$A^* = A + uv^T$$

  for some $n \times 1$ vectors u and v

- Goal: find a fast way of computing $(A^*)^{-1}$
  - Eventually, a fast way of solving $(A^*)\,x = b$

Analogue for Scalars

$Q:$ Knowing $\frac{1}{\alpha}$, how to compute $\frac{1}{\alpha+\beta}$ without division by α?

$$A: \quad \frac{1}{\alpha+\beta} = \frac{1}{\alpha(1+\frac{1}{\alpha}\beta)}$$
$$= \frac{1}{\alpha}\left(\frac{1+\frac{1}{\alpha}\beta-\frac{1}{\alpha}\beta}{1+\frac{1}{\alpha}\beta}\right)$$
$$= \frac{1}{\alpha}\left(1-\frac{\frac{1}{\alpha}\beta}{1+\frac{1}{\alpha}\beta}\right)$$

Sherman-Morrison Formula

$$\mathbf{A}^* = \mathbf{A} + uv^{\mathrm{T}} = \mathbf{A}(\mathbf{I} + \mathbf{A}^{-1}uv^{\mathrm{T}})$$

$$\left(\mathbf{A}^*\right)^{-1} = (\mathbf{I} + \mathbf{A}^{-1}uv^{\mathrm{T}})^{-1}\ \mathbf{A}^{-1}$$

Let $\mathbf{x} = \mathbf{A}^{-1}uv^{\mathrm{T}}$

Note that $\mathbf{x}^2 = \mathbf{A}^{-1}u\ v^{\mathrm{T}}\mathbf{A}^{-1}u\ v^{\mathrm{T}}$

Scalar! Call it λ

$$\mathbf{x}^2 = \mathbf{A}^{-1}u\lambda v^{\mathrm{T}} = \lambda\,\mathbf{A}^{-1}uv^{\mathrm{T}} = \lambda\,\mathbf{x}$$

Sherman-Morrison Formula

$$\mathbf{x}^2 = \lambda\,\mathbf{x}$$

$$\mathbf{x}\left(\mathbf{I} + \mathbf{x}\right) = \mathbf{x}\left(1 + \lambda\right)$$

$$-\mathbf{x} + \frac{\mathbf{x}}{1+\lambda}\left(\mathbf{I} + \mathbf{x}\right) = 0$$

$$\mathbf{I} + \mathbf{x} - \frac{\mathbf{x}}{1+\lambda}\left(\mathbf{I} + \mathbf{x}\right) = \mathbf{I}$$

$$\left(\mathbf{I} - \frac{\mathbf{x}}{1+\lambda}\right)\left(\mathbf{I} + \mathbf{x}\right) = \mathbf{I}$$

$$\therefore \quad \left(\mathbf{I} - \frac{\mathbf{x}}{1+\lambda}\right) = \left(\mathbf{I} + \mathbf{x}\right)^{-1}$$

$$\therefore \left(\mathbf{A}^{*}\right)^{-1} = \mathbf{A}^{-1} - \frac{\mathbf{A}^{-1}uv^{\mathrm{T}}\mathbf{A}^{-1}}{1 + v^{\mathrm{T}}\mathbf{A}^{-1}u}$$

Sherman-Morrison Formula

$$x = \left(\mathbf{A}^*\right)^{-1} b = \mathbf{A}^{-1} b - \frac{\mathbf{A}^{-1} u v^{\mathrm{T}} \mathbf{A}^{-1} b}{1 + v^{\mathrm{T}} \mathbf{A}^{-1} u}$$

Let $y = \mathbf{A}^{-1} b$, $z = \mathbf{A}^{-1} u$, so that $x = y - \dfrac{z v^{\mathrm{T}} y}{1 + v^{\mathrm{T}} z}$

So, to solve $\left(\mathbf{A}^*\right) x = b$,

solve $\mathbf{A} y = b$, $\mathbf{A} z = u$, $x = y - \dfrac{z v^{\mathrm{T}} y}{1 + v^{\mathrm{T}} z}$

Applying Sherman-Morrison

- Let's consider cyclic tridiagonal again:

$$\begin{bmatrix} a_{11} & a_{12} & & & & a_{16} \\ a_{21} & a_{22} & a_{23} & & & \\ & a_{32} & a_{33} & a_{34} & & \\ & & a_{43} & a_{44} & a_{45} & \\ & & & a_{54} & a_{55} & a_{56} \\ a_{61} & & & & a_{65} & a_{66} \end{bmatrix} x = b$$

- Take

$$\mathbf{A} = \begin{bmatrix} a_{11}-1 & a_{12} & & & & \\ a_{21} & a_{22} & a_{23} & & & \\ & a_{32} & a_{33} & a_{34} & & \\ & & a_{43} & a_{44} & a_{45} & \\ & & & a_{54} & a_{55} & a_{56} \\ & & & & a_{65} & a_{66}-a_{61}a_{16} \end{bmatrix}, \quad u = \begin{bmatrix} 1 \\ \\ \\ \\ \\ a_{61} \end{bmatrix}, \quad v = \begin{bmatrix} 1 \\ \\ \\ \\ \\ a_{16} \end{bmatrix}$$

Applying Sherman-Morrison

- Solve $Ay=b$, $Az=u$ using special fast algorithm
- Applying Sherman-Morrison takes a couple of dot products
- Total: O(n) time
- Generalization for several corrections: Woodbury

$$\mathbf{A}^* = \mathbf{A} + \mathbf{U}\mathbf{V}^T$$

$$\left(\mathbf{A}^*\right)^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{U}\left(\mathbf{I} + \mathbf{V}^T\mathbf{A}^{-1}\mathbf{U}\right)^{-1}\mathbf{V}^T\mathbf{A}^{-1}$$

More General Sparse Matrices

- More generally, we can represent sparse matrices by noting which elements are nonzero
- Critical for Ax and A^Tx to be efficient: proportional to # of nonzero elements
  – We'll see an algorithm for solving $Ax=b$ using only these two operations!

Compressed Sparse Row Format

- Three arrays
  - Values: actual numbers in the matrix
  - Cols: column of corresponding entry in values
  - Rows: index of first entry in each row
  - Example: (zero-based! C/C++/Java, not Matlab!)

$$\begin{bmatrix} 0 & 3 & 2 & 3 \\ 2 & 0 & 1 & 3 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 \end{bmatrix}$$

values	3 2 3 2 1 3 1 2 3
cols	1 2 3 0 2 3 1 2 3
rows	0 3 6 6 9

Compressed Sparse Row Format

- Multiplying Ax: $\begin{bmatrix} 0 & 3 & 2 & 3 \\ 2 & 0 & 1 & 3 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 \end{bmatrix}$

values	3 2 3 2 1 3 1 2 3
cols	1 2 3 0 2 3 1 2 3
rows	0 3 6 6 9

```
for (i = 0; i < n; i++) {
    out[i] = 0;
    for (j = rows[i]; j < rows[i+1]; j++)
        out[i] += values[j] * x[ cols[j] ];
}
```

Solving Sparse Systems

- Transform problem to a function minimization!

$$\text{Solve } Ax = b$$

$$\Rightarrow \text{ Minimize } f(x) = x^T A x - 2b^T x$$

- To motivate this, consider 1D:

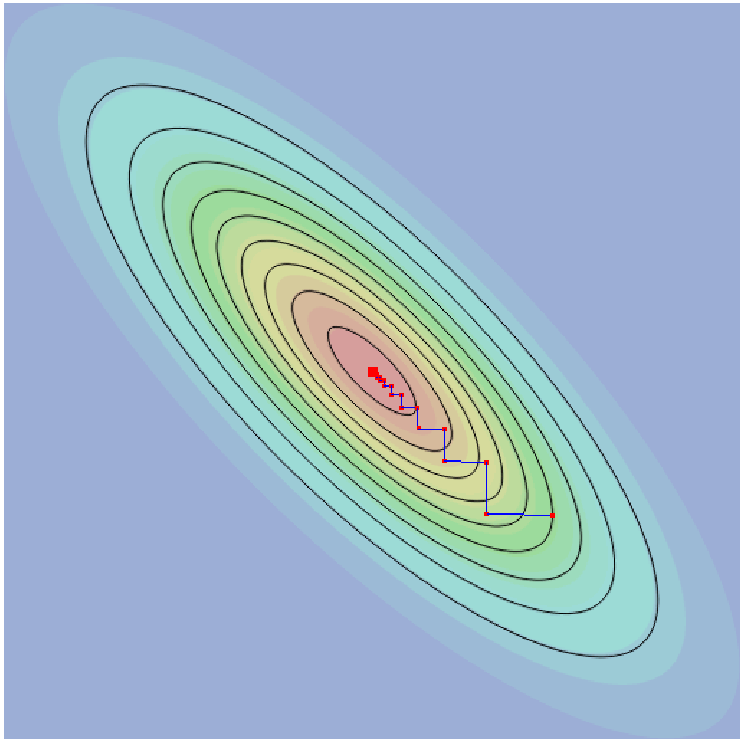
$$f(x) = ax^2 - 2bx$$

$$\frac{df}{dx} = 2ax - 2b = 0$$

$$ax = b$$

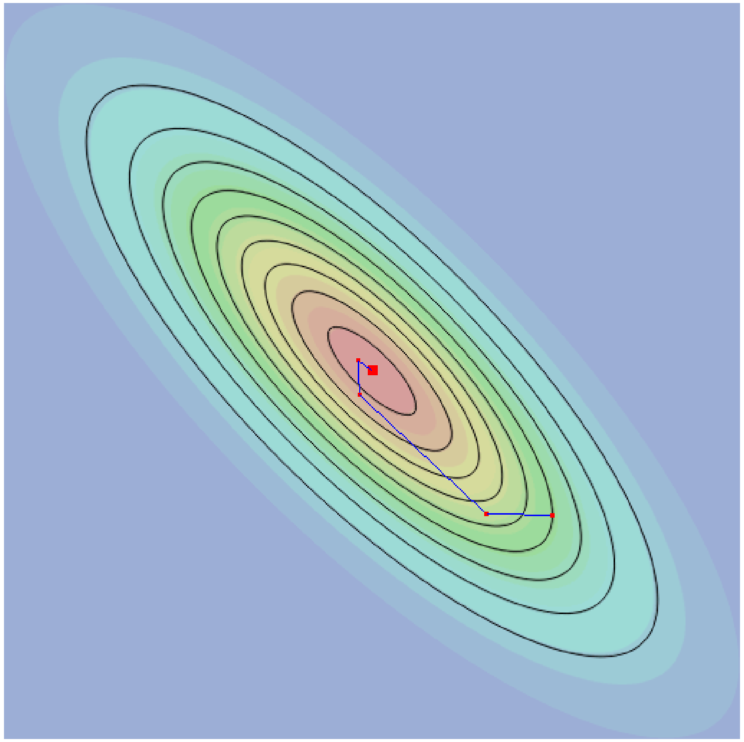
Solving Sparse Systems

- Preferred method: conjugate gradients
- Recall: plain gradient descent has a problem…



Solving Sparse Systems

- … that's solved by conjugate gradients



- Walk along direction

$$d_{k+1} = -g_{k+1} + \beta_k d_k$$

- Polak and Ribiere formula:

$$\beta_k = \frac{g_{k+1}^{\mathrm{T}}(g_{k+1} - g_k)}{g_k^{\mathrm{T}} g_k}$$

Solving Sparse Systems

- Easiest to think about A = symmetric
- First ingredient: need to evaluate gradient

$$f(x) = x^{\mathrm{T}} \mathbf{A}\, x - 2b^{\mathrm{T}} x$$
$$\nabla f(x) = 2\left(\mathbf{A}x - b\right)$$

- As advertised, this only involves A multiplied by a vector

Solving Sparse Systems

- Second ingredient: given point x_i, direction d_i, minimize function in that direction

$$\text{Define } m_i(t) = f(x_i + t\, d_i)$$

$$\text{Minimize } m_i(t): \quad \frac{d}{dt} m_i(t) = 0$$

$$\frac{dm_i(t)}{dt} = 2 d_i^{\mathrm{T}} \left(\mathbf{A} x_i - b \right) + 2t\, d_i^{\mathrm{T}} \mathbf{A}\, d_i \overset{\text{want}}{=} 0$$

$$t_{\min} = -\frac{d_i^{\mathrm{T}} \left(\mathbf{A} x_i - b \right)}{d_i^{\mathrm{T}} \mathbf{A} d_i}$$

$$x_{i+1} = x_i + t_{\min} d_i$$

Solving Sparse Systems

- Just a few sparse matrix-vector multiplies (plus some dot products, etc.) per iteration
- For m nonzero entries, each iteration O(max(m,n))
- Conjugate gradients may need n iterations for "perfect" convergence, but often get decent answer well before then
- For non-symmetric matrices: biconjugate gradient (maintains 2 residuals, requires A^Tx multiplication)