

CS205 – Class 13

Covered in class: 1, 3, 5

Readings: 6.7, 7.2 to 7.3.3

1. Recall the conjugate Gradient Method from previous class: Conjugate Gradient Method - the main idea is to search with conjugate directions where each new direction V is chosen in the steepest decent fashion, i.e. $V_k = -\nabla f(x_k) = r_k$

a. $s_0 = r_0 = b - Ax_0$ which is the steepest decent direction

b.
$$\alpha_k = \frac{r_k \cdot r_k}{s_k \cdot As_k}$$

c. $x_{k+1} = x_k + \alpha_k s_k$ and $r_{k+1} = r_k - \alpha_k As_k$ as always

d.
$$s_{k+1} = r_{k+1} + \frac{r_{k+1} \cdot r_{k+1}}{r_k \cdot r_k} s_k$$

e. This last equation for the new search direction needs some explanation since

we'd expect to have
$$s_{k+1} = r_{k+1} - \sum_{j=1}^k \frac{r_{k+1} \cdot As_j}{s_j \cdot As_j} s_j$$

i. Starting with $r_{k+1} = r_k - \alpha_k As_k$, we have $r_i \cdot r_{k+1} = r_i \cdot r_k - \alpha_k r_i \cdot As_k$ or $\alpha_k r_i \cdot As_k = r_i \cdot r_k - r_i \cdot r_{k+1}$

ii. When $i = k + 1$, $\alpha_k r_{k+1} \cdot As_k = r_{k+1} \cdot r_k - r_{k+1} \cdot r_{k+1} = -r_{k+1} \cdot r_{k+1}$ and thus

$$r_{k+1} \cdot As_k = \frac{-r_{k+1} \cdot r_{k+1}}{\alpha_k}$$

iii. When $i > k + 1$, $\alpha_k r_i \cdot As_k = r_i \cdot r_k - r_i \cdot r_{k+1} = 0$, i.e. $r_i \cdot As_k = 0$

iv. Thus, $s_{k+1} = r_{k+1} - \sum_{j=1}^k \frac{r_{k+1} \cdot As_j}{s_j \cdot As_j} s_j = r_{k+1} + \frac{r_{k+1} \cdot r_{k+1}}{\alpha_k (s_k \cdot As_k)} s_k$ since only

the last term in the sum is nonzero (Note how all the dot products disappear except for one!!)

v. Finally, plugging in the definition of α_k gives $s_{k+1} = r_{k+1} + \frac{r_{k+1} \cdot r_{k+1}}{r_k \cdot r_k} s_k$

as desired.

2. Preconditioning

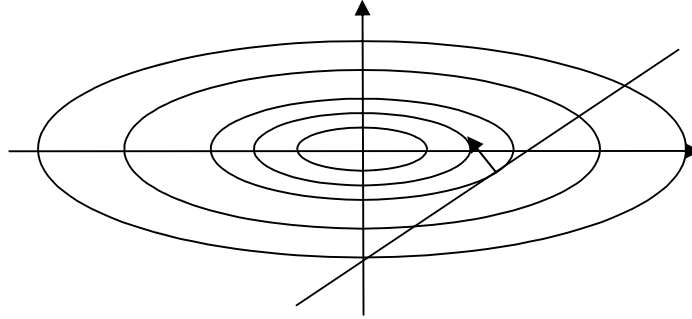
a. If we had an approximate inverse, we can transform $Ax=b$ into $\hat{A}^{-1}Ax = \hat{A}^{-1}b$ or $\hat{I}x = \hat{b}$ where \hat{I} is approximately the identity matrix

b. If all the eigenvalues of \hat{I} are approximately equal to 1, then we have “circles” instead of “ellipses” and CG converges much faster because of the duplicate or near duplicate eigenvalues

c. That is, preconditioning works great!

- d. Diagonal or Jacobi preconditioning scales the quadratic form along the coordinate axis to make it better conditioned (whereas it would be optimal to scale along the *eigenvector* axis)
- e. Incomplete Choleski preconditioning does a Choleski factorization with the caveat that only the nonzero entries are modified, i.e. all the zeros remain zeroes
3. Constrained Optimization
- a. Minimize $f(\bar{x})$ subject to constraints $\bar{g}(\bar{x}) = 0$
- i. Here $\bar{x} \in R^n$ and $\bar{g}(\bar{x}) = 0$ is as system of $m \leq n$ equations
 - ii. One can show that a solution \bar{x} must satisfy $-\nabla f(\bar{x}) = J_g^T(\bar{x})\bar{\lambda}$
 1. $J_g(\bar{x})$ is the Jacobian matrix of g
 2. $\bar{\lambda}$ is an m -vector of *Lagrange multipliers*
 3. This condition says that we cannot reduce the objective function without violating the constraints
 - iii. Define $L(\bar{x}, \bar{\lambda}) = f(\bar{x}) + \bar{\lambda}^T g(\bar{x})$
 1. The critical points are found by setting

$$\nabla L(\bar{x}, \bar{\lambda}) = \begin{bmatrix} \nabla f(\bar{x}) + J_g^T(\bar{x})\bar{\lambda} \\ g(\bar{x}) \end{bmatrix} = \vec{0}$$
 2. Suppose for simplicity that g is a linear function. Then the Hessian is $H(\bar{x}, \bar{\lambda}) = \begin{bmatrix} H_f(\bar{x}) & J_g^T(\bar{x}) \\ J_g(\bar{x}) & 0 \end{bmatrix}$ where the x partial derivatives of $J_g^T(\bar{x})\bar{\lambda}$ vanish because g is linear.
 - a. Note that H is not positive definite
 - b. It turns out that positive definiteness is only needed on the tangent space to the constraint surface, i.e. on the null space of J_g .
 - iv. Consider $f(x) = .5x_1^2 + 2.5x_2^2$ with $g(x) = x_1 - x_2 - 1 = 0$
 1. $L(\bar{x}, \bar{\lambda}) = .5x_1^2 + 2.5x_2^2 + \lambda(x_1 - x_2 - 1)$
 2. $\nabla L(\bar{x}, \bar{\lambda}) = \begin{bmatrix} x_1 + \lambda \\ 5x_2 - \lambda \\ x_1 - x_2 - 1 \end{bmatrix} = \vec{0}$
 3. so we solve $\begin{bmatrix} 1 & 0 & 1 \\ 0 & 5 & -1 \\ 1 & -1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \lambda \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$ to obtain $\begin{bmatrix} x_1 \\ x_2 \\ \lambda \end{bmatrix} = \begin{bmatrix} .833 \\ -.167 \\ -.833 \end{bmatrix}$



The gradient of the function is perpendicular to the constraint surface at the constrained minimum.

4. Linear Programming

- Minimize $\vec{c} \cdot \vec{x}$ subject to constraints $A\vec{x} = \vec{b}$ and $\vec{x} \geq \vec{0}$
- The feasible region is a convex polyhedron in n -dimensional space
- The minimum must occur at one of the vertices of the polyhedron
- Simplex method* - systematically examine a sequence of vertices to find the one yielding the minimum

5. Interpolation

- polynomial of degree n** $y = c_1 + c_2x + c_3x^2 + \dots + c_{n+1}x^n$
 - Monomial basis** - $y = c_1\phi_1 + c_2\phi_2 + c_3\phi_3 + \dots + c_{n+1}\phi_{n+1}$ where the basis function are $\phi_j(x) = x^{j-1}$ for $j = 1, 2, \dots, n+1$

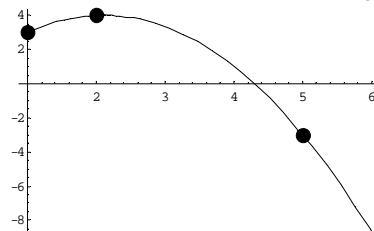
- Polynomial interpolation** given a set of $n+1$ points (x_i, y_i) , find the unique n degree polynomial

$y = c_1 + c_2x + c_3x^2 + \dots + c_{n+1}x^n$ that interpolates them.

- Solve $Ax = y$ where A is the $(n+1) \times (n+1)$ **Vandermonde matrix** with rows $(1, x_i, x_i^2, \dots, x_i^n)$ for each data point (x_i, y_i)
- Example: i.e. for $(1,3), (2,4), (5,-3)$ for quadratic we would

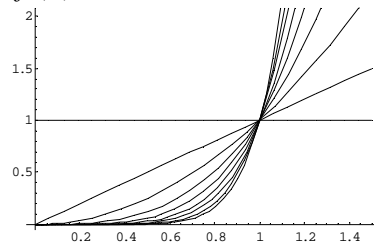
$$\text{have } \begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 4 \\ 1 & 5 & 25 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \end{pmatrix} = \begin{pmatrix} 3 \\ 4 \\ -3 \end{pmatrix} \text{ so we get } \begin{pmatrix} c_0 \\ c_1 \\ c_2 \end{pmatrix} = \begin{pmatrix} 1/3 \\ 7/2 \\ -5/6 \end{pmatrix} \text{ and}$$

thus the equation is $f(x) = \frac{1}{3} + \frac{7}{2}x - \frac{5}{6}x^2$ which looks like:



- But this is not an ideal basis, because as polynomials get higher, the functions have lots of overlap. Plotting

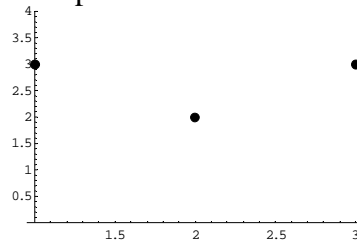
$f(x) = 1, x, x^2, \dots, x^8$ we can see this effect:



ii. **Lagrange interpolation** $y = c_1\phi_1 + c_2\phi_2 + c_3\phi_3 + \dots + c_{n+1}\phi_{n+1}$ with basis

$$\text{functions } \phi_j(x) = \frac{\prod_{k \neq j} (x - x_k)}{\prod_{k \neq j} (x_j - x_k)} \text{ for } j = 1, 2, \dots, n+1$$

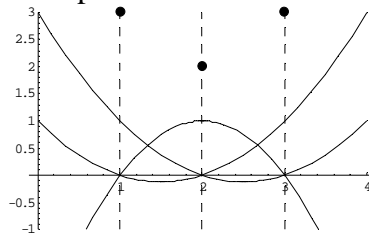
1. $\phi_j(x_j) = 1$ and $\phi_j(x_k) = 0$ where $k \neq j$ – therefore the coefficients are $c_j = y_j$ (easy to compute)
2. No “overlap” problem
3. Evaluation is expensive



4. Example: and we get

$$\phi_1(x) = \frac{(x-2)(x-3)}{(1-2)(1-3)} = \frac{1}{2}(x-2)(x-3) \text{ and we have}$$

$\phi_1(1) = 1, \phi_1(2) = 0, \phi_1(3) = 0$. We have that the polynomial is zero at each of the other data points. And it is one at the point. If we plot all the basis functions we see



iii. **Newton interpolation** $y = c_1\phi_1 + c_2\phi_2 + c_3\phi_3 + \dots + c_{n+1}\phi_{n+1}$ with basis

$$\text{functions } \phi_j(x) = \prod_{k=1}^{j-1} (x - x_k) \text{ for } j = 1, 2, \dots, n+1$$

1. No “overlap” problem
2. **Divided differences** We initially set $f[x_k] = y_k$ at the first level. Then the higher levels are based on

$$f[x_1, x_2, \dots, x_k] = \frac{f[x_2, x_3, \dots, x_k] - f[x_1, x_2, \dots, x_{k-1}]}{x_k - x_1}$$

3. The coefficients are given by $c_j = f[x_1, x_2, \dots, x_j]$

4. Represents a compromise between Lagrange and monomial basis.
 - iv. High order polynomials tend to be oscillatory
 - v. Using unequal data points can help, e.g. Chebyshev points
- b. A better solution is to use **piecewise polynomials** – a different polynomial in each subinterval $[x_i, x_{i+1}]$
 - i. Defined using **control points** (x_i, y_i)
 - ii. **Piecewise linear** – connect the control points with straight lines