Lecture 5: Decomposition Methods for Positive Definite (PD) Matrices

Prof. Krishna R. Pattipati

Dept. of Electrical and Computer Engineering
University of Connecticut
Contact: krishna@engr.uconn.edu (860) 486-2890

ECE 6435

Adv Numerical Methods in Sci Comp

Fall 2008
September 24, 2008

Outline

- Why do we need decomposition methods for *PD* matrices?
- Cholesky decomposition
- LDL^T decomposition
- A special *PD* matrix : Toeplitz System of Equations
  - Application to system identification
  - Levinson- Durbin algorithm
  - Generalized Levinson algorithm
- Conjugate gradient(CG) and pre-conditioned CG methods for sparse positive definite systems

Setting

- In the last lecture, we discussed how a non-singular $n \times n$ matrix A can be decomposed into a product of unit lower Δ matrix and an upper Δ matrix.

  - $PA=LU$ (P is a permutation matrix)

    $$PA\underline{x} = P\underline{b} = \underline{\tilde{b}}$$

  - solve

    $$\Rightarrow LU\underline{x} = \underline{\tilde{b}}$$

- An important special case is when $A=A^T$ and A is PD

  $\Rightarrow \lambda_i(A) > 0$ and $\underline{x}^T A\, \underline{x} > 0$

- Fact: " If A is symmetric PD, then there exist a lower Δ matrix, S with positive diagonal entries such that $A=SS^T$"

Cholesky Decomposition

$$\begin{bmatrix} s_{11} & 0 & 0 & .. & 0 & 0 \\ s_{21} & s_{22} & & & & 0 \\ : & & .. & & & : \\ s_{k1} & .. & .. & s_{kk} & & 0 \\ : & & & & .. & : \\ s_{n1} & s_{n2} & .. & s_{nk} & .. & s_{nn} \end{bmatrix} \begin{bmatrix} s_{11} & s_{21} & .. & s_{k1} & .. & s_{n1} \\ s_{21} & s_{22} & & & & s_{n2} \\ : & & .. & & & : \\ & & .. & s_{kk} & & s_{nk} \\ : & & .. & . & .. & : \\ .. & .. & .. & .. & .. & s_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & .. & a_{1k} & .. & a_{1n} \\ a_{12} & a_{22} & & & & a_{2n} \\ : & & .. & & & : \\ & & .. & s_{kk} & & s_{nk} \\ : & & .. & . & .. & : \\ a_{1n} & a_{2n} & .. & .. & .. & a_{nn} \end{bmatrix}$$

$s_{ii} > 0$ S is called CHOLESKY Δ

- $A=SS^T$ is called CHOLESKY DECOMPOSITION (or) SQUARE ROOT DECOMPOSITION
- Note: since A is symmetric, need to store only the upper (or lower) Δ portion only

Why do we need such decompositions?

- Why do we need such a decomposition ?

1) To test positive definiteness of a symmetric matrix (this will become apparent from the decomposition algorithm)
2) Square root updates of covariance matrices in least squares estimation and Kalman filtering.

- Recall update and propagate equations of Kalman Filtering.
- Measurement update:

$$P_{k|k} = [I - G_k H]P_{k|k-1}$$
$$= P_{k|k-1} - P_{k|k-1}H^T(R + HP_{k|k-1}H^T)^{-1}HP_{k|k-1}$$

- Propagate:

$$P_{k+1|k} = \Phi P_{k|k-1}\Phi^T + EW_dE^T$$

- Update eqn. often results in $(P_{ii})_{k|k}<0$ especially when $\|W_d\|$ is small and / or $\|R\|$ is small

Square root decomposition & κ(*P*)

- One solution: Joseph's form, but requires double the computational load of ordinary update equation

$$P_{k|k} = (I - G_k H) P_{k|k-1} (I - G_k H)^T + G_k R G_k^T$$

- Second solution: recursive square-root update (Lecture 8)

propagate $\sqrt{P_{k|k}} \Rightarrow P_{k|k} = S_k S_k^T$

- Why does second solution work?
  - $P_{k|k}$ is *PD* if S_k is non-singular
  - $\| P_{k|k} \|_2 = \lambda_{\max}(P_{k|k}); \quad \| S_k \|_2 = \sqrt{\lambda_{\max}(S_k S_k^T)} = \sqrt{\lambda_{\max}(P_{k|k})}$

    $\Rightarrow \| S_k \|_2^2 = \| P_{k|k} \|_2$

  - So, $\kappa(P_{k|k}) = \dfrac{\lambda_{\max}(P_{k|k})}{\lambda_{\min}(P_{k|k})}; \quad \kappa(S_k) = \sqrt{\dfrac{\lambda_{\max}(P_{k|k})}{\lambda_{\min}(P_{k|k})}}$

    $\kappa(P_{k|k}) = 10^6 \Rightarrow \kappa(S_k) = 10^3$

Unconstrained Minimization - 1

- So, square root propagation reduces the condition number

→ Can get greater precision with the same computer , or equivalently

→ Can get the same precision with a smaller word length computer … critical in applications with space and weight problems

3) Unconstrained and constrained minimization

- $\underline{x}^*$ is a relative local minimum of $f(\underline{x}) \Rightarrow \nabla^2 f(\underline{x}) \geq 0$
- $\underline{x}^*$ is a strict relative local minimum of $f(\underline{x}) \Rightarrow \nabla^2 f(\underline{x}) > 0$
- Recall modified Newton's method $\nabla^2 f(\underline{x})\, \underline{d}_k = -\nabla f(\underline{x})$

Unconstrained Minimization - 2

$$\Rightarrow \quad \underline{x}_{k+1} = \underline{x}_k + \alpha_k \underline{d}_k ; \alpha_k = \arg\min_{\alpha}[\, f(\underline{x}_k + \alpha \underline{d}_k)]$$

$$\Rightarrow \quad \nabla \underline{f}^T (\underline{x}_k + \alpha_k \underline{d}_k)\, \underline{d}_k = 0$$

$$\Rightarrow \quad f(\underline{x}_k + \alpha \underline{d}_k) \approx f(\underline{x}_k) + \alpha \nabla \underline{f}^T (\underline{x}_k)\, \underline{d}_k$$

$$\Rightarrow \quad \nabla \underline{f}^T (\underline{x}_k)\, \underline{d}_k < 0$$

$$\Rightarrow \quad -\nabla_f^T (\underline{x}_k)[\nabla^2 f(\underline{x}_k)]^{-1} \nabla f(\underline{x}_k) < 0$$

$$\Rightarrow \quad \nabla^2 f(\underline{x}_k) \text{ must be } PD$$

- Cholesky's method will provide a method for testing *PD* of $\nabla^2 f(\underline{x})$ and also to make it *PD* when it is not by adding εI to $\nabla^2 f(\underline{x})$

4) Quasi-Newton methods

$$\underline{x}_{k+1} = \underline{x}_k + \alpha_k \underline{d}_k$$

$$\underline{d}_k = -D_k \nabla \underline{f}(\underline{x}_k), \text{ where } D_k \text{ is } PD$$

Unconstrained Minimization - 3

- $\exists$ a large class of Quasi-Newton methods. But, we restrict ourselves to the so-called Broyden-Fletcher-Goldfard-Shanno (BFGS) class:

$$\underline{p}_k = \underline{x}_{k+1} - \underline{x}_k$$

$$\underline{q}_k = \nabla \underline{f}(\underline{x}_{k+1}) - \nabla \underline{f}(\underline{x}_k)$$

$$D_{k+1} = D_k + \frac{\underline{p}_k \underline{p}_k^T}{\underline{q}_k^T \underline{p}_k} - \frac{D_k \underline{q}_k \underline{q}_k^T D_k}{\underline{q}_k^T D_k \underline{q}_k} + \zeta_k \tau_k \underline{v}_k \underline{v}_k^T$$

$$\underline{v}_k = \underline{p}_k - \frac{1}{\tau_k} D_k \underline{q}_k$$

$$\tau_k = \frac{\underline{q}_k^T D_k \underline{q}_k}{\underline{p}_k^T \underline{q}_k}$$

$$0 \le \zeta_k \le 1$$

Example of Cholesky Decomposition

- Davidon-Fletcher-Powell (DFP) $\Rightarrow \zeta_k=0$
- BFGS $\Rightarrow \zeta_k=1$
- Propagate $\sqrt{D_k}$ to avoid round-off error problems.
- Notice rank two (or three) corrections to go from $D_k \rightarrow D_{k+1}$

❑ Example of Cholesky decomposition

$$A=\begin{bmatrix} 2 & -2 \\ -2 & 5 \end{bmatrix};\ S=\begin{bmatrix} \sqrt{2} & 0 \\ -\sqrt{2} & \sqrt{3} \end{bmatrix} \Rightarrow A=SS^T$$

- Also note that we can write $S=\begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix}\begin{bmatrix} \sqrt{2} & 0 \\ 0 & \sqrt{3} \end{bmatrix}$

$$\Rightarrow A=SS^T=LDL^T \text{ where } S=LD^{1/2}$$

- As with LU decomposition, we evaluate S one column at a time

Cholesky : Set Up

- Consider the situation at the k^{th} column of S (assume done up to column k-1)
  - for $i \geq$ k, we have

$$a_{ik} = \sum_{m=1}^{k} s_{im} s_{km} = \sum_{m=1}^{k-1} s_{im} s_{km} + s_{kk} s_{ik}$$

  - Rearranging this equation, we obtain

$$s_{kk} = \left(a_{kk} - \sum_{m=1}^{k-1} s_{km}^2 \right)^{\frac{1}{2}}$$

$$s_{ik} = (a_{ik} - \sum_{m=1}^{k-1} s_{im} s_{km}) / s_{kk}; \text{ for } i = k+1, \ldots, n$$

- Compute S one column at a time
- Can also compute S one row at a time (see problem set #5)
- Overwrite a_{ij} with s_{ij}; $i \geq j$ (in place computation)

Cholesky Decomposition Algorithm

- Algorithm Cholesky: Column Version

for $k = 1, 2, ..., n$ Do

$$a_{kk} = (a_{kk} - \sum_{m=1}^{k-1} a_{km}^2)^{1/2}$$

for $i = k+1, ..., n$ Do

$$a_{ik} = (a_{ik} - \sum_{m=1}^{k-1} a_{im} a_{km}) / a_{kk}$$

End Do(i)

End Do(k);

- No pivoting is needed if A is PD
- **Accumulate sums in *DP***
- Use of algorithm to test PD of A : If $(-\#)^{1/2} \Rightarrow A$ is not PD !!

- Computational load : n square roots plus

$$\sum_{k=1}^{n} k(n-k) \text{ multiplies}$$

$$= \frac{n^2(n+1)}{2} - \frac{n(2n+1)(n+1)}{6} = \frac{n(n+1)(n-1)}{6} \approx O(n^3/6) \approx \frac{1}{2} \text{ of } LU$$

- $s_{kk}^2 \le \sum_{m=1}^{k} s_{km}^2 \le a_{kk} \Rightarrow s_{kk} < a_{kk} \Rightarrow$ elements are bounded

Pivoting

- Pivoting for positive semi-definite matrices:
  - At step k, find the biggest $a_{ll} - \sum_{i=1}^{k-1} s_{li}^2$; $l = k, ..., n$
  - Pre- and post-multiply A by permutation matrix
  - Why? … because we need to preserve symmetry of A
  - Also recall that the permutation matrix is Symmetric

  $\Rightarrow$ permute S by permutation matrix $P_k^{r_k}$, where r_k is the row with the biggest element in the previous step.

  That is, $A = PSS^T P^T = \tilde{S}\tilde{S}^T$

  - So, we actually find an SS^T factorization of PAP
  - Good to pivot, since can find a reduced rank square-root matrix $S = n \times r$

LDL^T Decomposition - 1

- Problem with Cholesky
  - Need to compute square roots
  - Square roots are more expensive than multiplications and divisions (≈ a factor of 2).

- LDL^T Factorization
  - $A = LDL^T$ is similar to Cholesky decomposition, but avoids square root evaluations. $d_i \geq 0$; $l_{ii} = 1$

$$\begin{bmatrix} 1 & 0 & .. & 0 \\ l_{21} & 1 & .. & 0 \\ : & & & : \\ l_{n1} & l_{n2} & .. & 1 \end{bmatrix} \begin{bmatrix} d_1 & 0 & .. & 0 \\ 0 & d_2 & .. & 0 \\ : & : & & : \\ 0 & 0 & .. & d_n \end{bmatrix} \begin{bmatrix} 1 & l_{21} & .. & l_{n1} \\ 0 & 1 & .. & l_{n2} \\ : & : & & .. \\ 0 & 0 & .. & 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & .. & a_{1n} \\ a_{12} & a_{22} & .. & a_{n2} \\ .. & .. & .. & .. \\ a_{1n} & a_{n2} & .. & a_{nn} \end{bmatrix}$$

LDL^T Decomposition - 2

- For $i \geq k$, we have

$$a_{ik} = \sum_{m=1}^{k} l_{im} d_m l_{km} \Rightarrow d_k = a_{kk} - \sum_{m=1}^{k-1} l_{km} d_m l_{km} \text{ since } l_{kk} = 1$$

- Therefore

$$l_{ik} = (a_{ik} - \sum_{m=1}^{k} l_{im} d_m l_{km}) / d_k$$

- Comments:
  - The term $d_m l_{km}$ is independent of i
  - Overwrite a_{ik} with l_{ik} and a_{kk} with $d_k \Rightarrow$ no need for extra storage
  - Requires $O(n^3/6)$ operations

LDL^T Algorithm

- Algorithm for LDL^T factorization

For $k = 1, 2, \ldots, n$ Do

  For $m = 1, 2, \ldots, k-1$ Do

    $r_m = a_{mm} a_{km}$ ...recall $a_{mm} = d_m$, $a_{km} = l_{km}$

    $a_{kk} \leftarrow a_{kk} - a_{km} r_m$

  end Do(m)

  If $a_{kk} \leq 0$ then

    quit ...A is not PD

  else

    for $i = k+1, k+2, \ldots, n$ Do

      $a_{ik} \leftarrow (a_{ik} - \sum_{m=1}^{k-1} a_{im} r_m) / a_{kk}$ ...recall $a_{kk} = d_k$; $a_{im} = l_{im}$; $a_{ik} = l_{ik}$ at the end.

    End

  Endif

End

Toeplitz System of Equations

- Application to System Identification: Toeplitz system of Equations
  - What is the System Identification problem?

" Given the input and output sequences, determine the transfer function relating the input and output."

  - Restricted problem:
    - Suppose that the input is a white noise sequence $\{w(k)\}$ and output sequence $\{y(k)\}$ is related to input via the autoregressive relation:

$$y(k)+\sum_{i=1}^{n} a_i y(k-i) = gw(k)$$

    - $w(k)$~ zero mean white noise process with unit variance
    - Problem :" Given $\{y(k)\}$ sequence, find $\hat{a}_i,\ i=1,2,...,n$ and $\hat{g}$ such that $J = E\{e^2(k)\}$

$$= E\{[y(k)+\sum_{i=1}^{n} \hat{a}_i\, y(k-i)]^2\} \text{ is a minimum"}$$

Minimization of Cost Function

- $e(k)$ is called the prediction error, $[y(k) - \hat{y}(k/k-1)]$.
- So, want to minimize mean-squared error prediction of $y(k)$ from its past data $\{y(k\text{-}n), y(k\text{-}n\text{-}1), \ldots, y(k\text{-}1)\}$
- This is a Parameter Identification (estimation)problem

- The necessary conditions of optimality yield:

$$\frac{\partial J}{\partial \hat{a}_j} = 0 \Rightarrow E\{[y(k) + \sum_{i=1}^{n} \hat{a}_i y(k-i)]y(k-j)\} = 0;\ j = 1, \ldots, n$$

- $e(k)$ is orthogonal to $y(k - j)\ \forall\ j = 1, 2, \ldots, n$
- $J = E[y(k)e(k)] = \hat{g}^2$

- Expanding the necessary conditions of optimality, we obtain:

$$\sum_{i=1}^{n} \hat{a}_i \phi_y(j-i) = -\phi_y(j);\ j = 1, \ldots, n$$

where $\phi_y(j-i) = E\{y(k-i)y(k-j)\} = E\{y(j)y(i)\}$

due to stationarity which is due to linear time-invariance assumption of the stochastic system

Toeplitz System of Equations

- In matrix form, the necessary conditions are given by:

$$\begin{bmatrix} \phi_y(0) & \phi_y(1) & .. & \phi_y(n-1) \\ \phi_y(1) & \phi_y(0) & .. & \phi_y(n-2) \\ : & : & & : \\ \phi_y(n-1) & \phi_y(n-2) & .. & \phi_y(0) \end{bmatrix} \begin{bmatrix} \hat{a}_1 \\ \hat{a}_2 \\ : \\ \hat{a}_n \end{bmatrix} = - \begin{bmatrix} \phi_y(1) \\ \phi_y(2) \\ : \\ \phi_y(n) \end{bmatrix} \Rightarrow \Phi_n \underline{\hat{a}} = -\underline{b}$$

- The objective function is:

$$J = E\{y(k)e(k)\} = \phi_y(0) + \sum_{i=1}^{n} \hat{a}_i \phi_y(i) = \phi_y(0) - \underline{\hat{a}}^T \Phi_n \underline{\hat{a}} = \hat{g}^2$$

premultiply by Diag $[\phi_y(0)]^{-1}$

$$\begin{bmatrix} 1 & r_1 & r_2 & .. & r_{n-1} \\ r_1 & 1 & r_1 & .. & r_{n-2} \\ r_2 & r_1 & 1 & .. & r_{n-3} \\ : & : & & & : \\ r_{n-1} & r_{n-2} & .. & .. & 1 \end{bmatrix} \begin{bmatrix} \hat{a}_1 \\ \hat{a}_2 \\ : \\ : \\ \hat{a}_n \end{bmatrix} = - \begin{bmatrix} r_1 \\ r_2 \\ : \\ : \\ r_n \end{bmatrix}; \; r_i = \frac{\phi_y(i)}{\phi_y(0)} \text{ correlation coefficient}$$

Properties of Toeplitz Matrix

$$T_n \underline{\hat{a}} = -\underline{r}$$

- These are called **Yule-Walker** equations
- Toeplitz matrix:
  - Symmetric matrix specified by n elements (including *RHS*).
  - *RHS* has a special form
  - This enables us to solve this problem in $O(n^2)$ operations
- Key properties of Toeplitz:
  - T_n is persymmetric $\Rightarrow$ $T_n = E\, T_n\, E$, $E \sim$ *Exchange Matrix*

$$E^{-1} = E;\ E^2 = I;\ E = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

  - T_n^{-1} is also persymmetric
- Physical meaning: The statistical properties of a stationary time series are not modified by reversing time (time-reversibility property)

Solution of Toeplitz Equations - 1

- We will solve $T_n \underline{x} = \underline{b}$ as a solution of two subproblems.
  - Subproblem 1: solve $T_n \underline{a} = -(r_1\ r_2\ \ldots..\ r_n)^T$ (Levinson-Durbin's Algorithm)
  - Subproblem 2: use the solution of 1 to solve $T_n \underline{x} = \underline{b}$ ($\underline{b}$ is general) (Generalized Levinson's algorithm)
- Subproblem 1:
  - Suppose have solved $T_k \underline{a} = -\underline{r} \Rightarrow \underline{a} = -T_k^{-1} \underline{r}$
  - Note that $\underline{a}$ is of dimension k
  - What we are looking for is a recursive way of building up $\underline{a}$ from dimension 1 to n.

$$\begin{bmatrix} r_0 & r_1 & r_2 & .. & r_{k-1} \\ r_1 & r_0 & r_1 & .. & r_{k-2} \\ r_2 & r_1 & r_0 & .. & r_{k-3} \\ : & : & & & : \\ r_{k-1} & r_{k-2} & .. & .. & r_0 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ : \\ : \\ a_k \end{bmatrix} = - \begin{bmatrix} r_1 \\ r_2 \\ : \\ : \\ r_k \end{bmatrix}; \ r_0 = 1$$

Solution of Toeplitz Equations - 2

- Can we solve $T_{k+1}\begin{bmatrix} z \\ \alpha \end{bmatrix} = -\begin{bmatrix} r \\ r_{k+1} \end{bmatrix}$ using $\underline{a}$?

$\Rightarrow$ Can we get next $\underline{a}$?

- Recursion $k \rightarrow k+1$

$$\begin{bmatrix} T_k & E_k \underline{r} \\ \underline{r}^T E_k & 1 \end{bmatrix} \begin{bmatrix} z \\ \alpha \end{bmatrix} = -\begin{bmatrix} r \\ r_{k+1} \end{bmatrix}$$

where $E_k \sim k$ by k exchange matrix

- Given $\underline{a}$, we can solve this problem in $O(k)$ flops. How?

$$T_k \underline{z} + E_k \underline{r} \alpha = -\underline{r}$$

$$\Rightarrow \underline{z} = -T_k^{-1}\underline{r} - \alpha T_k^{-1} E_k \underline{r}$$

- Toeplitz is per symmetric $\Rightarrow T_k^{-1} E_k = E_k T_k^{-1}$

$$\Rightarrow \boxed{\underline{z} = \underline{a} + \alpha E_k \underline{a} = (I + \alpha E_k)\underline{a} \Rightarrow z_i = a_i + \alpha a_{k+1-i}}$$

Solution of Toeplitz Equations - 3

- In signal processing,
  α is termed the reflection coefficient
  $\{a_i\}$ = forward filter coefficients
  $\{a_{k+1-i}\}$ = backward filter coefficients

$\Rightarrow$ next forward filter coefficients = weighted sum of previous forward and backward filter coefficients

- Next, $\alpha = -r_{k+1} - \underline{r}^T E_k \underline{z} = -(r_{k+1} + \underline{r}^T E_k \underline{a} + \alpha \underline{r}^T \underline{a})$

$$\Rightarrow \quad \alpha = -(r_{k+1} + \underline{r}^T E_k \underline{a}) / (1 + \underline{r}^T \underline{a})$$

$$\Rightarrow \quad 1 - \underline{r}^T T_k^{-1} \underline{r} > 0$$

$1 + \underline{r}^T \underline{a} > 0$. This is true because T_{k+1} is PD and

$$\begin{bmatrix} I & E_k \underline{a} \\ 0 & I \end{bmatrix}^T \begin{bmatrix} T_k & E_k \underline{r} \\ \underline{r}^T E_k & I \end{bmatrix} \begin{bmatrix} I & E_k \underline{a} \\ 0 & I \end{bmatrix} = \begin{bmatrix} T_k & 0 \\ 0 & 1 + \underline{r}^T \underline{a} \end{bmatrix}$$

T_{k+1} is $PD \Rightarrow A^T T_{k+1} \mathrm{A}$ is PD, if A is nonsingular

Simplification

- Check:

$$\begin{bmatrix} I & 0 \\ \underline{a}^T E_k & 1 \end{bmatrix}^T \begin{bmatrix} T_k & E_k \underline{r} \\ \underline{r}^T E_k & 1 \end{bmatrix} \begin{bmatrix} I & E_k \underline{a} \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} T_k & E_k \underline{r} \\ \underline{a}^T E_k T_k + \underline{r}^T E_k & 1 + \underline{r}^T \underline{a} \end{bmatrix} \begin{bmatrix} I & E_k \underline{a} \\ 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} T_k & T_k E_k \underline{a} + E_k \underline{r} \\ \underline{a}^T E_k T_k + \underline{r}^T E_k & 1 + \underline{r}^T \underline{a} \end{bmatrix} = \begin{bmatrix} T_k & 0 \\ 0 & 1 + \underline{r}^T \underline{a} \end{bmatrix}$$

since $T_k E_k \underline{a} + E_k \underline{r} = 0$ (recall $\underline{a} = T_k^{-1} \underline{r}$ and persymmetry of T_k)

- Major Simplification:

$$\beta_k = (1 + \underline{r}^{(k)^T} \underline{a}^{(k)}) = 1 + \begin{bmatrix} \underline{r}^{(k-1)^T} & r_k \end{bmatrix} \begin{bmatrix} \underline{a}^{(k-1)} + \alpha_{k-1} E_{k-1} \underline{a}^{(k-1)} \\ \alpha_{k-1} \end{bmatrix}$$

$$= 1 + \underline{r}^{(k-1)^T} \underline{a}^{(k-1)} + \alpha_{k-1} \underline{r}^{(k-1)^T} E_{k-1} \underline{a}^{(k-1)} + \alpha_{k-1} r_k = \beta_{k-1} + \alpha_{k-1} [\underline{r}^{(k-1)^T} E_{k-1} \underline{a}^{(k-1)} + r_k]$$

But, $\beta_{k-1} \alpha_{k-1} = -r_k - \underline{r}^{(k-1)^T} E_{k-1} \underline{a}^{(k-1)}$ ...recall equation for α

$$\Rightarrow \boxed{\beta_k = (1 - \alpha_{k-1}^2) \beta_{k-1}}$$

Levisnon – Durbin Algorithm - 1

- Levinson-Durbin's Problem (subproblem 1)
  - "Given $r_0\ r_1.....r_n,\ r_0 = 1, T = [r_{|i-j|}]$ such that $|r_i| < 1\ \forall i$

    and T is PD, solve $T\underline{a} = -(r_1\ r_2.....r_n)^{T}$"

    $a_1 = -r_1$

    $\beta = 1$

    $\alpha = -r_1$

    For $k = 1, 2, ..., n-1$ Do

    $\beta \leftarrow (1-\alpha^2)\beta$

    $\alpha = -(r_{k+1} + \sum_{i=1}^{k} r_{k+1-i} a_i) / \beta$

    For $i = 1, 2, ..., k$ Do

    $z_i \leftarrow a_i + \alpha a_{k+1-i}$ (Note: can't set $a \leftarrow a$ here because of the exchange matrix E)

Levisnon – Durbin Algorithm - 2

End Do(i)

$a_i \leftarrow z_i \quad i = 1, 2, ..., k$

$a_{k+1} = \alpha$

End Do(k)

$\beta \leftarrow \beta(1-\alpha^2)$

$J = \beta \phi_y(0) = \hat{g}^2$

- Total Flop count over n steps $\approx O(n^2)$ operations

- Reference:

  Durbin, " The Fitting of time series models", Rev. Inst. Int. Statistics, 28, pp. 233-243, 1960 or any Standard book on statistical signal processing (e.g., L. Scharf, Addison – Wesley, 1991).

Example of Toeplitz Equations

Example 1: Solve

$$\begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}\begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} -0.5 \\ -0.2 \end{bmatrix} \Rightarrow \underline{a} = \frac{4}{3}\begin{bmatrix} 1 & -0.5 \\ -0.5 & 1 \end{bmatrix}\begin{bmatrix} -0.5 \\ -0.2 \end{bmatrix} = \begin{bmatrix} -8/15 \\ 1/15 \end{bmatrix}$$

$$a_1 = -0.5,\ \beta = 1,\ \alpha = -0.5$$

$$k = 1 \Rightarrow \beta = 0.75,\ \alpha = -(0.2 - 0.25)(4/3) = 0.067 = 1/15$$

$$z_1 = a_1 + \alpha a_1 = -0.5(1 + 0.067) = -8/15$$

$$a_2 = \alpha = 1/15$$

$$\Rightarrow \text{solution } \underline{\hat{a}} = \begin{bmatrix} \hat{a}_1 \\ \hat{a}_2 \end{bmatrix} = \begin{bmatrix} -8/15 \\ 1/15 \end{bmatrix} \ ; J = 1 + \hat{a}_1 r_1 + \hat{a}_2 r_2 = \frac{56}{75} = \hat{g}^2$$

Forward – Backward Filter Interpretation - 1

- Another way of looking at Levinson- Durbin problem:
  - Recall

$$\Phi_n \underline{a} = -\begin{bmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_n \end{bmatrix} = -\underline{\phi}$$

$$\phi_y(0) + \underline{\phi}^T \underline{a} = J$$

  - So,

$$\begin{bmatrix} \phi_y(0) & \phi_y(1) & .. & \phi_y(n) \\ \phi_y(1) & \phi_y(0) & .. & \phi_y(n-1) \\ \vdots & \vdots & & \vdots \\ \phi_y(n) & \phi_y(n-1) & .. & \phi_y(0) \end{bmatrix} \begin{bmatrix} 1 \\ \underline{a} \end{bmatrix} = \begin{bmatrix} \phi_y(0) & \underline{\phi}^T \\ \underline{\phi} & \Phi_n \end{bmatrix} \begin{bmatrix} 1 \\ \underline{a} \end{bmatrix} = \begin{bmatrix} J \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

  - Initially, let $J = \phi_y(0) \Rightarrow$ zeroth order prediction $(k = 0)$

Forward – Backward Filter Interpretation - 2

- At k^{th} step, suppose have $\underline{a}^{(k)} = (a_1, ..., a_k)$ and have J_k.

  If $k = n$, we are done.

- Want to find:

$$\begin{bmatrix} \phi_y(0) & \phi_y(1) & .. & \phi_y(k+1) \\ \phi_y(1) & \phi_y(0) & .. & \phi_y(k) \\ : & : & & : \\ \phi_y(k+1) & \phi_y(k) & .. & \phi_y(0) \end{bmatrix} \begin{bmatrix} 1 \\ \underline{a}^{(k+1)} \end{bmatrix} = \begin{bmatrix} \phi_y(0) & \underline{\phi}_{k+1}^T \\ \underline{\phi}_{k+1} & \Phi_{k+1} \end{bmatrix} \begin{bmatrix} 1 \\ \underline{a}^{(k+1)} \end{bmatrix} = \begin{bmatrix} J_{k+1} \\ 0 \\ : \\ 0 \end{bmatrix}$$

- Instead, consider two alternative versions of the equations:

1

$$\begin{bmatrix} \Phi_{k+1} & \underline{\phi}_{k+1} \\ \underline{\phi}_{k+1}^T & \phi_y(0) \end{bmatrix} \begin{bmatrix} 1 \\ \underline{a}^{(k)} \\ 0 \end{bmatrix} = \begin{bmatrix} J_k \\ 0 \\ 0 \\ : \\ -\gamma_{k+1} \end{bmatrix}$$

2

$$\begin{bmatrix} \phi_y(0) & \underline{\phi}_{k+1}^T \\ \underline{\phi}_{k+1} & \Phi_{k+1} \end{bmatrix} \begin{bmatrix} 0 \\ E\underline{a}^{(k)} \\ 1 \end{bmatrix} = \begin{bmatrix} -\gamma_{k+1} \\ 0 \\ 0 \\ : \\ J_k \end{bmatrix}$$

where $\gamma_{k+1} = -[\phi_y(k+1) + \sum_{i=1}^{k} \phi_y(k+1-i) a_i^{(k)}]$

Forward – Backward Filter Interpretation - 3

- Key: $\begin{bmatrix} 1 \\ \underline{a}^{(k+1)} \end{bmatrix} = \begin{bmatrix} 1 \\ \underline{a}^{(k)} \\ 0 \end{bmatrix} + \alpha_{k+1} \begin{bmatrix} 0 \\ E\underline{a}^{(k)} \\ 1 \end{bmatrix}$

$$\begin{bmatrix} \phi_y(0) & \underline{\phi}_{k+1}^T \\ \underline{\phi}_{k+1} & \Phi_{k+1} \end{bmatrix} \begin{bmatrix} 1 \\ \underline{a}^{(k+1)} \end{bmatrix} = \begin{bmatrix} J_k - \alpha_{k+1}\gamma_{k+1} \\ 0 \\ : \\ \alpha_{k+1}J_k - \gamma_{k+1} \end{bmatrix} = \begin{bmatrix} J_{k+1} \\ 0 \\ : \\ 0 \end{bmatrix}$$

- So,

  Pick $\alpha_{k+1} = \gamma_{k+1} / J_k$.. called reflection coefficient

$\Rightarrow J_{k+1} = J_k[1-\alpha_{k+1}^2] \sim \beta$ of Durbin's Algorithm

❑ Estimates $\hat{a}_i = a_i^{(n)}$ are ∋ roots of $(1+\sum_{i=1}^{n} \hat{a}_i z^{-i})$ are inside the unit circle, if the reflection coefficients $|\alpha_k| < 1$ for $k = 1, 2, ..., n$.

Guranteed if Toeplitz matrix is PD.

Forward – Backward Filter Interpretation - 4

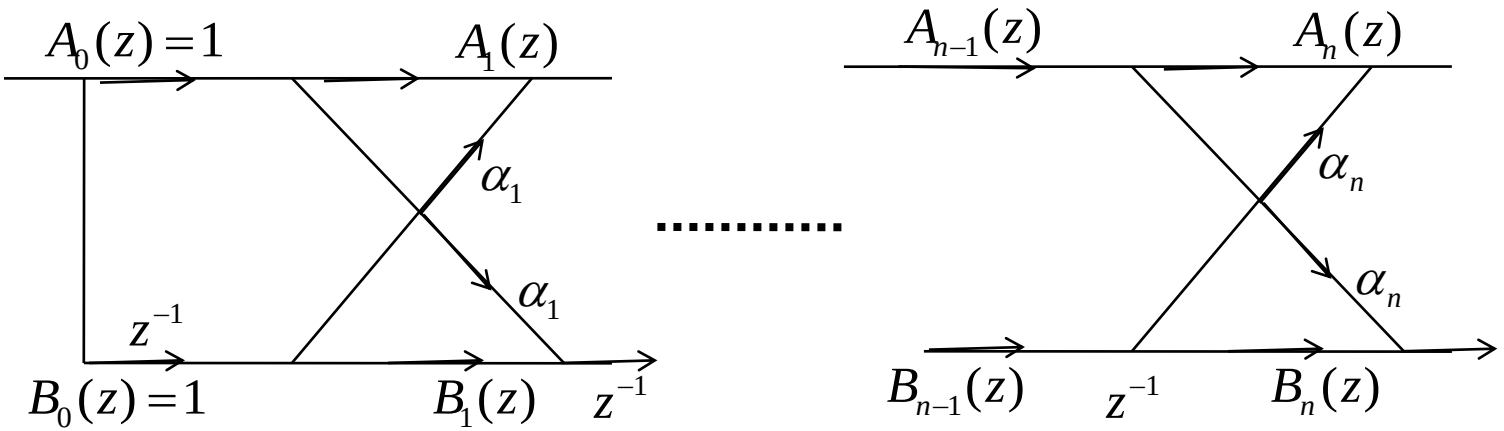
Moving-average whitening filter Interpretation of Levinson-Durbin algorithm

$$A_k(z) = 1 + \sum_{i=1}^{k} a_i^{(k)} z^{-i} \qquad \text{(Forward Filter)}$$

$$A_k(z) = A_{k-1}(z) + \alpha_k z^{-1} B_{k-1}(z)$$
$$B_k(z) = z^{-1} B_{k-1}(z) + \alpha_k A_{k-1}(z)$$

$$B_k(z) = z^{-k} + \sum_{i=1}^{k} a_{k+1-i}^{(k)} z^{-(i-1)} = z^{-1} A_k(z^{-1}) \text{ (Backward Filter)}$$

can show (look at $z_i = a_i + \alpha a_{k+1-i}$ of Levinson -Durbin)



Moving Average Lattice Filter: $A_n(z)\{y(k)\} = \hat{g}\{w(k)\}$

Toeplitz with General RHS

- Generalized Levinson-Durbin's Problem (subproblem 2):
  - What if the *RHS* is arbitrary $\underline{b}$
  - Suppose have solved $T_k \underline{x}^{(k)} = \underline{b}^{(k)} = (b_1 \ b_2 \ \ldots \ b_k)^T$
  - For simplicity, write $T_k \underline{x} = \underline{b}$

Want to solve $T_{k+1} \begin{bmatrix} \underline{\nu} \\ \mu \end{bmatrix} = \begin{bmatrix} \underline{b} \\ b_{k+1} \end{bmatrix} = \begin{bmatrix} T_k & E_k \underline{r} \\ \underline{r}^T E_k & 1 \end{bmatrix} \begin{bmatrix} \underline{\nu} \\ \mu \end{bmatrix} = \begin{bmatrix} \underline{b} \\ b_{k+1} \end{bmatrix}$

$\underline{r} = (r_1 \ r_2 \ \ldots \ r_k)^T$ as before

$$T_k \underline{\nu} + E_k \underline{r} \mu = \underline{b} \ \Rightarrow \ \underline{\nu} = T_k^{-1}[\underline{b} - E_k \underline{r} \mu] = \underline{x} + \mu E_k \underline{a}; \underline{a} = -T_k^{-1} \underline{r}$$

$$\underline{r}^T E_k \underline{\nu} + \mu = b_{k+1} \Rightarrow \ \mu = (b_{k+1} - \underline{r}^T E_k \underline{x}) / (1 + \underline{r}^T \underline{a})$$

- Note: need to solve $T_k \underline{a} = -\underline{r}$ using Levinson-Durbin's algorithm
- If you know $\underline{x}$ and $\underline{a}$, computing $\underline{\nu}$ and μ requires $O(2k)$ operations

Generalized Levinson – Durbin

- Key Idea : to solve $T_n \underline{x} = \underline{b}$

  1) solve $T_k \underline{x}^{(k)} = \underline{b}^{(k)} = (b_1 \; b_2 \; \ldots \; b_k)^T \quad O(n^2)$

  2) solve $T_k \underline{a} = -\underline{r} \quad O(n^2)$

  } Can be done in parallel. Flop count $O(2n^2)$

- "Given $r_0, r_1, \ldots, r_n, \; r_0 = 1, T = [r_{|i-j|}]$ such that $|r_i| < 1 \; \forall i$ and T is PD, solve $T_n \underline{x} = \underline{b}$."

$a_1 = -r_1$

$x_1 = b_1$

$\beta = 1$

$\alpha = -r_1$

for $k = 1, 2, \ldots, n-1$ Do

$\beta \leftarrow (1 - \alpha^2)\beta$

$\mu = (b_{k+1} - \sum_{i=1}^{k} r_i x_{k+1-i}) / \beta$

$x_i = x_i + \mu \, a_{k+1-i}; i = 1, 2, .., k$

$x_{k+1} = \mu$

if $k < n$ -1 then

$\alpha = -(r_{k+1} + \sum_{i=1}^{k} r_i a_{k+1-i}) / \beta$

$z_i \leftarrow a_i + \alpha a_{k+1-i}; \; i = 1, 2, \ldots, k$

$a_i \leftarrow z_i \quad i = 1, 2, \ldots, k$

$a_{k+1} = \alpha$

End if

End Do(k)

Sparse Matrix and Iterative Methods

- Sparse matrix methods for Symmetric Positive Definite Systems
  - Store only non-zero elements (row, column, element value)
  - Two classes of methods:
    - Sparse Cholesky or LDL^T decomposition (intelligent data structures and strategies to minimize fill-in)

    Ref:

    1. I. S. Duff, A. M. Erisman, and J. K. Reid, Direct Methods for Sparse Matrices, Oxford Univ. Press, 1986.
    2. J. A. George and J. W. Liu, Computer Solution of Large Sparse Positive Definite Systems, Prentice-Hall, 1981.

    - Iterative methods
      - Gauss-Seidel method with successive overrelaxation: Convergence critically depends on several parameters that are hard to choose
      - Conjugate Gradient (CG) method: Widely used method for sparse PD systems

Conjugare Gradient Method - 1

- Conjugate Gradient method
  - Consider the problem of minimizing a quadratic function
    $f(\underline{x})=\frac{1}{2}\,\underline{x}^T A\underline{x}-b^T\underline{x}$
    where A is an $n \times n$ symmetric and PD matrix
  - The unique solution to this problem is the solution of

$$\nabla f(\underline{x}) = 0 \Rightarrow A\underline{x} = \underline{b}$$

  - An efficient (especially when A is sparse such as in a large-scale linear programming problem) way of solving the linear equation is the **conjugate gradient** or the **conjugate direction** method.

- Definition: A set of vectors $\{\underline{d}_i\}_{i=1}^k$ are A-orthogonal or mutually conjugate with respect to A, if $\underline{d}_i^T A\,\underline{d}_j=0 \forall i\neq j$, $i=1, 2, \ldots, k$
- Basic idea of conjugate direction method:
  - Given a collection of n mutually A-conjugate directions, $\{\underline{d}_i\}_{i=1}^n$ conjugate direction method generates the solution of $A\underline{x}=\underline{b}$ via:
    $\underline{x}=\alpha_1\underline{d}_1+\alpha_2\underline{d}_2+\ldots+\alpha_n\underline{d}_n$ (1)

Conjugare Gradient Method - 2

- **Key**: $\{\alpha_i\}$ are very easy to obtain :
- Multiply (1) by $\underline{d}_i^T A\underline{x}$ to obtain:
  $\underline{d}_i^T A\underline{x} = \alpha_i \underline{d}_i^T A\underline{d}_i$ recalling: $\underline{d}_i^T A\underline{d}_j = 0 \ \forall \ i \neq j$

$$\Rightarrow \alpha_i = \frac{\underline{d}_i^T \underline{b}}{\underline{d}_i^T A\underline{d}_i}$$

- Q : Is there a simple and iterative (i.e., sequential) way of generating $\{d_i\}$? Yes!!

  Q: What if I start at a point $\underline{x}_i \neq \underline{0}$? No problem… only the equations for $\{\alpha_i\}$ will change.

- In order to solve $A\underline{x}=\underline{b}$, this is what we would like to do
  - Start with $\underline{x}_1$
  - Compute residual $\underline{r}_1 = \underline{b} - A\underline{x}_1 = -\nabla f(x)$

    Negative gradient of quadratic function $f(x)$ is the residual
  - Let $\underline{d}_1 = \underline{r}_1 \Rightarrow \underline{x}_2 = \underline{x}_1 + \alpha_1 \underline{d}_1$
  - Compute new residual $\underline{r}_2 = \underline{b} - A\underline{x}_2$
  - Get $\underline{d}_2 = \underline{r}_2 + \beta_1 \underline{d}_1 \ni \underline{d}_2$ is A-orthogonal to $\underline{d}_1$, etc.
- Since our directions are based on residuals (=negative gradients), the method is called Conjugate gradient (CG) method.

Conjugare Gradient Method - 3

- In general, at the i^{th} step of CG method, we compute
  - residual $\underline{r}_i = \underline{b} - A\underline{x}_i$
  - new direction $\underline{d}_i = \underline{r}_i + \beta_{i-1}\underline{d}_{i-1}$
  - new point $\underline{x}_{i+1} = \underline{x}_i + \alpha_i \underline{d}_i$
- Note: residual $\underline{r}_i$ can be computed from previous residual $\underline{r}_{i-1}$

$$r_i = \underline{b} - A\,\underline{x}_i = \underline{b} - A(\underline{x}_{i-1} + \alpha_{i-1}\underline{d}_{i-1})$$

$$\Rightarrow \underline{r}_i = \underline{r}_{i-1} - \alpha_{i-1} A\,\underline{d}_{i-1}$$

- Key: we will see later that $A\underline{d}_{i-1}$ comes for free because it is used in computing α_i
- So, need expression for α_i and β_i
  - Suppose we are at $\underline{x}_i$ and know the direction $\underline{d}_i$. What is α_i?
  - The best $\alpha = \alpha_i$ must minimize $f(\underline{x})$ along $\underline{d}_i$ starting from $\underline{x}_i$
  - To get α_i, consider $f(\underline{x}_i + \alpha \underline{d}_i)$

$$f(\underline{x}_i + \alpha \underline{d}_i) = \tfrac{1}{2}(\underline{x}_i + \alpha \underline{d}_i)^T A(\underline{x}_i + \alpha \underline{d}_i) - \underline{b}^T(\underline{x}_i + \alpha \underline{d}_i)$$

$$\text{Optimal } \alpha \Rightarrow \frac{\partial f(\underline{x} + \alpha \underline{d}_i)}{\partial \alpha}\Big|_{\alpha=\alpha_i} = 0$$

$$\Rightarrow \left[\alpha \underline{d}_i^T A \underline{d}_i + \underline{d}_i^T (A\underline{x}_i - \underline{b})\right]\Big|_{\alpha=\alpha_i} = 0$$

Conjugare Gradient Method - 4

or, $$\alpha_i = \frac{\underline{d}_i^T \underline{r}_i}{\underline{d}_i^T A \underline{d}_i}$$

$\Rightarrow$The **residual** at the current point, the current **direction** and, of course, the ***A* matrix** are all that are needed in computing α_i.

- Consider the derivative of $f(\underline{x}+\alpha\underline{d}_i)$ at $\alpha=\alpha_i$. This is:

$$\nabla f^T(\underline{x}+\alpha_i\underline{d}_i)\underline{d}_i = -\underline{r}_{i+1}^T\underline{d}_i = 0$$

$\Rightarrow$The current direction $\underline{d}_i$ is orthogonal to the *next* residual $\underline{r}_{i+1}$

$\Rightarrow$Note, however, that the direction $\underline{d}_i$ is a linear combination of $\underline{r}_i$ and $\underline{d}_{i-1}$ (we will formally show this below).

$\Rightarrow$ $\underline{r}_{i+1}$ is orthogonal to $\underline{r}_i$ and $\underline{d}_{i-1}$ as well

- So, we have the important CG relations

$$\underline{r}_{i+1}^T\underline{d}_i = 0$$
$$\underline{r}_{i+1}^T\underline{r}_i = 0$$
$$\underline{r}_{i+1}^T\underline{d}_{i-1} = 0$$

Conjugare Gradient Method - 5

- Key: The residuals in a conjugate gradient method are mutually orthogonal. We will see later (in lecture 11) that they are parallel to the so called Lanczos vectors
  - To further simplify the equation for α_i, let us consider
    $\underline{d}_i = \underline{r}_i + \beta_{i-1}\underline{d}_{i-1}$
  - Taking the inner product with r_i, we get
    $\underline{r}_i^T \underline{d}_i = \underline{r}_i^T \underline{r}_i + \beta_{i-1}\underline{r}_i^T \underline{d}_{i-1} = \underline{r}_i^T \underline{r}_i$
    ($\underline{r}_i^T \underline{d}_{i-1} = 0$)
  - So, we have our final equation for α_i:

$$\alpha_i = \frac{\underline{r}_i^T \underline{r}_i}{\underline{d}_i^T A \underline{d}_i}$$

- Proof of direction update equation: $\underline{d}_i = \underline{r}_i + \beta_{i-1}\underline{d}_{i-1}$

At step i, we have $\underline{x}_i$ and $\underline{r}_i$. What we want to do is this. We seek $\underline{x}_{i+1}$ such that it is a minimum point not merely in the negative gradient direction $\underline{r}_i$, but in a plane passing through $\underline{x}_i$ and spanned by $\underline{r}_i$ and $\underline{d}_{i-1}$. (It turns out that we are effectively minimizing in a subspace spanned by $\underline{r}_i$ and $\{\underline{d}_j\}_{j=1}^{i-1}$ as well)

Conjugare Gradient Method - 6

- Want to find β_{i-1} ə $\underline{d}_i^T A \underline{d}_{i-1} = 0$
- Using the *A*- orthogonality condition

$$\beta_{i-1} = \frac{-\underline{r}_i^T A \underline{d}_{i-1}}{\underline{d}_{i-1}^T A \underline{d}_{i-1}}$$

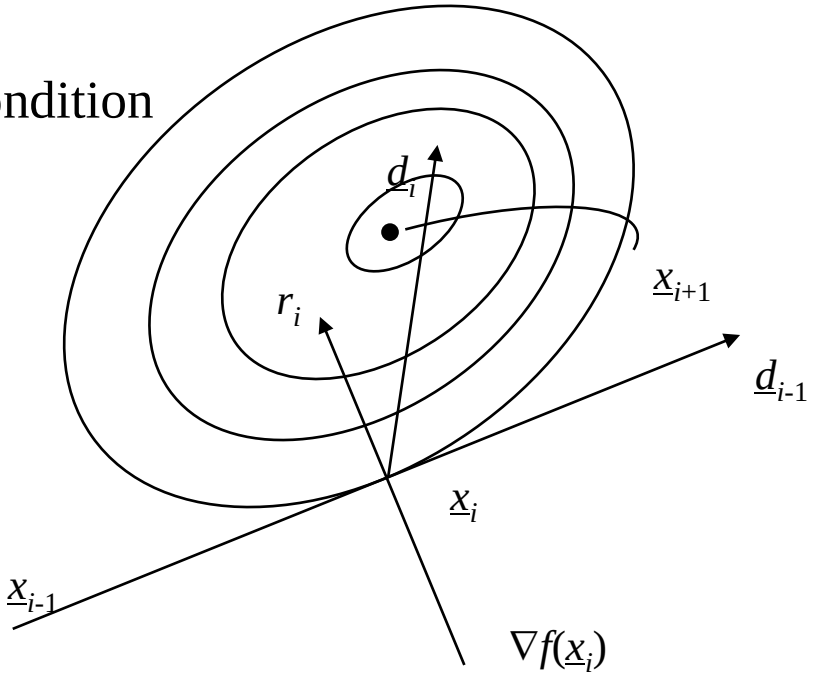
- Using the fact that

$$A\underline{d}_{i-1} = \frac{-1}{\alpha_{i-1}}(\underline{r}_i - \underline{r}_{i-1})$$

and that $\underline{r}_i^T \underline{r}_{i-1} = 0$, we have

$$\beta_{i-1} = \frac{\underline{r}_i^T \underline{r}_i}{\alpha_{i-1}(\underline{d}_{i-1}^T A \underline{d}_{i-1})} = \frac{\underline{r}_i^T \underline{r}_i}{\underline{r}_{i-1}^T \underline{r}_{i-1}}$$

(from the α_i equation)



- So, we have the final equation for β_{i-1} as

$$\beta_{i-1} = \frac{\underline{r}_i^T \underline{r}_i}{\underline{r}_{i-1}^T \underline{r}_{i-1}}$$

CG Algorithm

- All residuals $\underline{r}_i$ are orthogonal $\Rightarrow \underline{r}_i^T\underline{r}_j=0 \ \forall i\neq j$
- All directions $\underline{d}_i$ are A-orthogonal $\Rightarrow \underline{d}_i^T A\underline{d}_j=0 \ \forall i\neq j$

} See Luenberger (1984)

- CG Algorithm:

"Given a PD matrix A, b and a tolerance parameter, $\in$, and maximum number of iterations i_{max}, the following algorithm solves $A\underline{x}=\underline{b}$."

$i=1$

$\underline{x}=\underline{x}_1$ … initial point

$\underline{r}=\underline{b}-A\underline{x}$ … initial residual

$\rho=\|\underline{r}\|_2^2$ … square of norm of residual

$c=\|\underline{b}\|_2^2$ … norm of $\underline{b}$

$\underline{d}=\underline{r}$

DO while $\sqrt{\rho} \geq c \in$ or $i \leq i_{max}$

$\underline{w}=A\underline{d}$

$\alpha=\rho/\underline{d}^T\underline{w}$ … step length

$\underline{x}=\underline{x}+\alpha\underline{d}$ … new point

$\underline{r}=\underline{r}-\alpha\underline{w}$ … new residual

$\beta=\|\underline{r}\|_2^2/\rho$

$\underline{d}=\underline{r}+\beta\underline{d}$ …new direction

$\rho=\|\underline{r}\|_2^2$ … square of norm of residual

$i=i+1$

end DO

Pre-conditioned CG - 1

- Each iteration requires a matrix-vector multiplication +10n operations
  - Exploit sparsity in computing $\underline{w}=A\,\underline{d}$
- Need just four vectors for $\underline{x}$, $\underline{r}$, $\underline{d}$, and $\underline{w}$
- Convergence is faster if $k(A)$ is small … see Luenberger (1984)

$$(\underline{x}-\underline{x}_k)^T Q(\underline{x}-\underline{x}_k) \le 4(\underline{x}-\underline{x}_0)^T Q(\underline{x}-\underline{x}_0)\left(\frac{\sqrt{k(A)}-1}{\sqrt{k(A)}+1}\right)^{2k}$$

$k(A)\approx 1 \Rightarrow$ convergence is faster.

Q : can we make $k(A)\approx 1 \Rightarrow$ pre-conditioned conjugate gradient method.

- Pre-conditioned conjugate gradient (PCG) method
  - Consider

$$A\underline{x}=\underline{b}$$

  - Instead of solving $A\underline{x}=\underline{b}$, we solve

$$L^{-1}A\underline{x}=L^{-1}\underline{b}$$

where L is an approximation to the square-root of A.

$$\Rightarrow L^{-1}A(L^{-1})^T L^T\underline{x}=L^{-1}\underline{b}$$

or

$$\tilde{A}\underline{\tilde{x}}=\underline{\tilde{b}}$$

Pre-conditioned CG - 2

where $\tilde{A} = (L^{-1} A L^{-1})^T$

$\underline{\tilde{x}} = L^T \underline{x}$

$\underline{\tilde{b}} = L^{-1} \underline{b}$

- So, if L is close to the square-root S of A, then
  $\tilde{A} = L^{-1} S S^T (L^{-1})^T \approx I \Rightarrow k(\tilde{A}) \approx 1$

$\Rightarrow$ Fast convergence

- Q1: How to obtain L without actually doing complete Cholesky decomposition? … Incomplete Cholesky decomposition
- Q2: How to solve the modified system of equations?
- We will take up equation 2 first. It turns out that the preconditioner has "local" effect in the sense that it always appears as $M^{-1}=(L^{-1})^T L^{-1}$ in computing inner products related to the computation of β and α
  - See Golub and Van Loan, 1989

PCG Algorithm - 1

- Preconditioned CG algorithm:
  - "Given a PD matrix A, $\underline{b}$, a pre-conditioner L, a tolerance parameter ε and maximum no. of iterations, i_{max}, the following algorithm solves $A\underline{x}=\underline{b}$."

$i=1$

Solve $L\,\underline{y}=\underline{b}$

Solve $L^T\,\underline{x}=\underline{y}$

} Computes initial point. If $L\,L^T \approx A$, we have a good starting solution … initial residual

$\underline{r}_{new}=\underline{b}-A\underline{x}$

$\rho=\|\underline{r}_{new}\|_2$

$c=\|\underline{b}\|_2$

DO while $\rho > c\in$ or $i \le i_{max}$

  solve $L\underline{y}=\underline{r}_{new}$

  solve $L^T\underline{z}_{new}=\underline{y}$

  $\gamma_{new}=\underline{z}^T_{new}\,\underline{r}_{new}$

  If $i=1$

    $\underline{d}=\underline{z}_{new}$

  else

PCG Algorithm - 2

$\beta=\gamma_{new}/\gamma_{old}$

$\underline{d}=\underline{z}_{new}+\beta\underline{d}$

end if

$\underline{y}=A\underline{d}$

$\alpha=\gamma_{new}/\underline{d}^T\underline{y}$

$\underline{x}=\underline{x}+\alpha\underline{d}$

$\gamma_{old}=\gamma_{new}$

$\underline{z}_{old}=\underline{z}_{new}$

$\underline{r}_{old}=\underline{r}_{new}$

$\underline{r}_{new}=\underline{r}_{new}-\alpha\underline{y}$

$i=i+1$

end DO

- Incomplete Cholesky decomposition to obtain *L*
  - Fact: even if *A* is sparse, its "true" Cholesky factor *S* need not be!! This is called "fill-in"
  - So, what incomplete Cholesky decomposition does is to set:

    $l_{ij} = 0$ if $a_{ij} = 0$
  - We can do this with a slightly altered version of Cholesky, where *L* overwrites *A*.

Incomplete Cholesky Algorithm

- Algorithm Incomplete Cholesky

For k=1, 2, …, n, DO

$a_{kk} = \sqrt{a_{kk}}$

For i=k+1, …, n DO

If $a_{ik} \neq 0$

$a_{ik} = a_{ik} / a_{kk}$

end if

end

For j=k+1, …, n DO

For i=j, …, n DO

If $a_{ij} \neq 0$

$a_{ij} = a_{ij} - a_{ik}\, a_{jk}$

end if

end DO (i)

end DO (j)

end DO (k)

- Preconditioning has dramatic effect on convergence of the solution to $A\underline{x}=\underline{b}$ using the conjugate gradient method.

Summary

- Why do we need decomposition methods for *PD* matrices?
- Cholesky decomposition
- LDL^T decomposition
- A special *PD* matrix : Toeplitz System of Equations
  - Application to system identification
  - Levinson- Durbin algorithm
  - Generalized Levinson algorithm
- Conjugate gradient(CG) and pre-conditioned CG methods for sparse positive definite systems