CG Method – Conjugate Gradient (CG) Method

M. R. Hestenes & E. Stiefel, 1952

BiCG Method – Biconjugate Gradient (BiCG) Method

C. Lanczos, 1952
D. A. H. Jacobs, 1981
C. F. Smith et al., 1990
R. Barret et al., 1994

CGS Method – Conjugate Gradient Squared (CGS) Method (MATLAB Function)

P. Sonneveld, 1989

GMRES Method – Generalized Minimal – Residual (GMRES) Method

Y. Saad & M. H. Schultz, 1986
R. Barret et al., 1994
Y. Saad, 1996

QMR Method – Quasi-Minimal-Residual (QMR) Method

R. Freund & N. Nachtigal, 1990
N. Nachtigal, 1991
R. Barret et al., 1994
Y. Saad, 1996

Conjugate Gradient Method

1. Introduction, Notation and Basic Terms

- The CG is one of the most popular iterative methods for solving large systems of linear equations

  $$\mathbf{Ax=b}$$

  which arise in many important settings, such as finite difference and finite element methods for solving partial differential equa-tions, circuit analysis etc.
- It is suited for use with *sparse matrices*. If A is dense, the best choice is to factor A and solve the equation by backsubstitution.
- There is a fundamental underlying structure for almost all the descent algorithms: (1) one starts with an initial point; (2) deter-mines according to a fixed rule a direction of movement; (3) mo-ves in that direction to a relative minimum of the objective function; (4) at the new point, a new direction is determined and the process is repeated. The difference between different algorithms depends upon the rule by which successive directions of movement are selected.

1. Introduction, Notation and Basic Terms (Cont'd)

- A *matrix* is a rectangular array of numbers, called elements.
- The transpose of an $m \times n$ matrix $\mathbf{A}$ is the $n \times m$ matrix $\mathbf{A}^T$ with elements

$$a_{ij}^T = a_{ji}$$

- Two square $n \times n$ matrices $\mathbf{A}$ and $\mathbf{B}$ are similar if there is a nonsingular matrix $\mathbf{S}$ such that

$$\mathbf{B}=\mathbf{S}^{-1}\mathbf{A}\mathbf{S}$$

- Matrices having a single row are called *row vectors*; matrices having a single column are called *column vectors*.

Row vector:	$\mathbf{a} = [a_1, a_2, \ldots, a_n]$
Column vector:	$\mathbf{a} = (a_1, a_2, \ldots, a_n)$

- The inner product of two vectors is written as

$$\mathbf{x}^T\mathbf{y} = \sum_{i=1}^{n} x_i y_i$$

1. Introduction, Notation and Basic Terms (Cont'd)

- A matrix is called *symmetric* if $a_{ij} = a_{ji}$.
- A matrix is *positive definite* if, for every nonzero vector $\mathbf{x}$

$$\mathbf{x}^T\mathbf{A}\mathbf{x} > 0$$

- Basic matrix identities: $(\mathbf{AB})^T=\mathbf{B}^T\mathbf{A}^T$ and $(\mathbf{AB})^{-1}=\mathbf{B}^{-1}\mathbf{A}^{-1}$
- A quadratic form is a scalar, quadratic function of a vector with the form

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} - \mathbf{b}^T\mathbf{x} + c$$

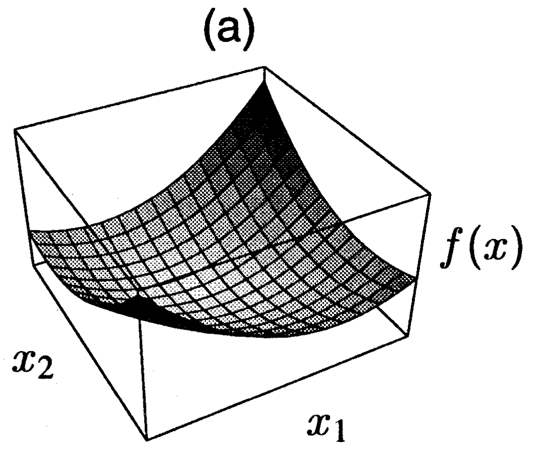
- The gradient of a quadratic form is

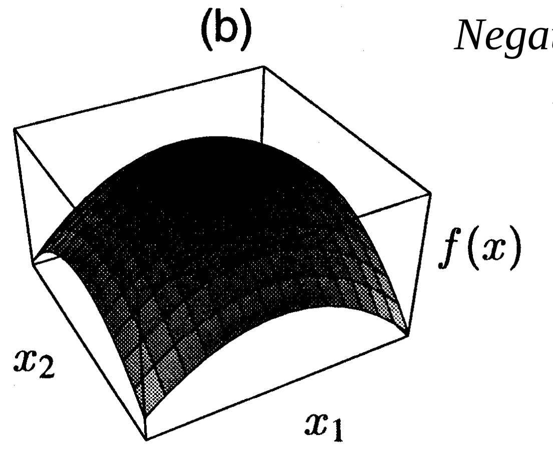
$$f'(\mathbf{x}) = \begin{bmatrix} \frac{\partial}{\partial x_1} f(\mathbf{x}) \\ \vdots \\ \frac{\partial}{\partial x_n} f(\mathbf{x}) \end{bmatrix} = \frac{1}{2}\mathbf{A}^T\mathbf{x} + \frac{1}{2}\mathbf{A}\mathbf{x} - \mathbf{b}$$

1. Introduction, Notation and Basic Terms (Cont’d)

- Various quadratic forms for an arbitrary 2×2 matrix:

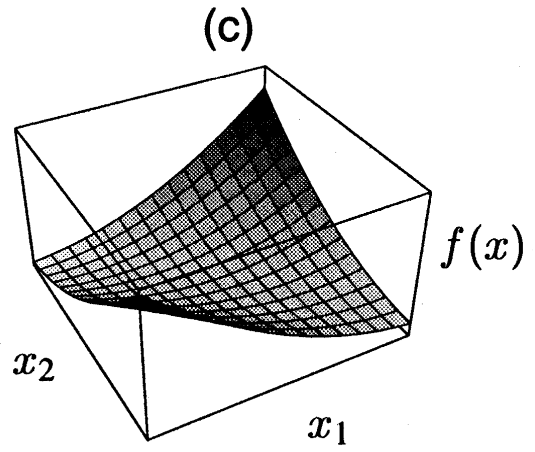
Positive-definite matrix

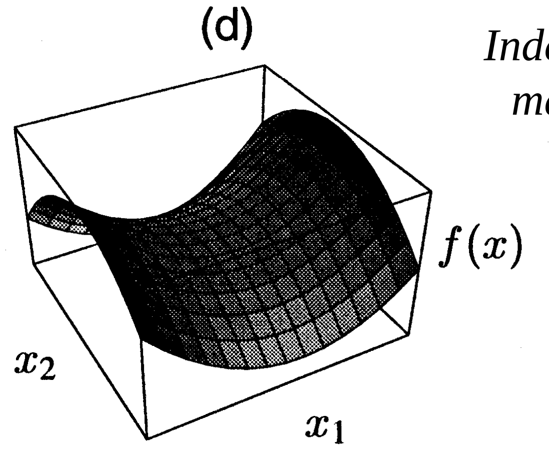




Negative-definite matrix

Singular positive-indefinite matrix



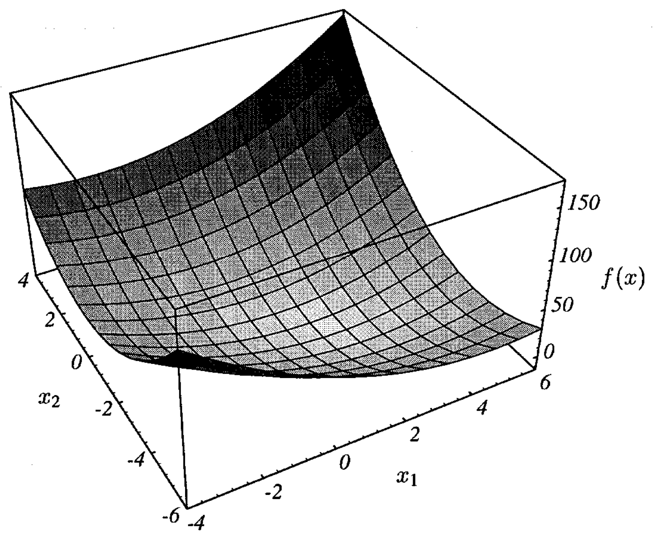


Indefinite matrix

1. Introduction, Notation and Basic Terms (Cont'd)

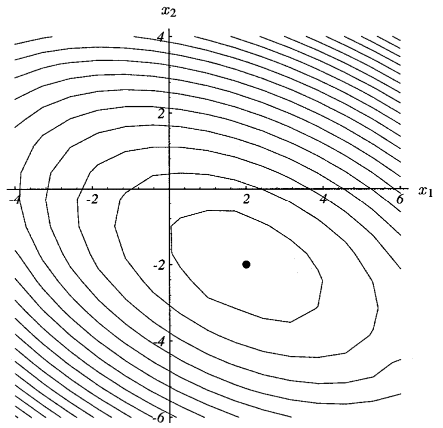
- Specific example of a 2×2 symmetric positive definite matrix:

$$\mathbf{A} = \begin{bmatrix} 3 & 2 \\ 2 & 6 \end{bmatrix},\ \mathbf{b} = \begin{bmatrix} 2 \\ -8 \end{bmatrix},\ c = 0 \ \Rightarrow\ \mathbf{x} = \begin{bmatrix} 2 \\ -2 \end{bmatrix}$$

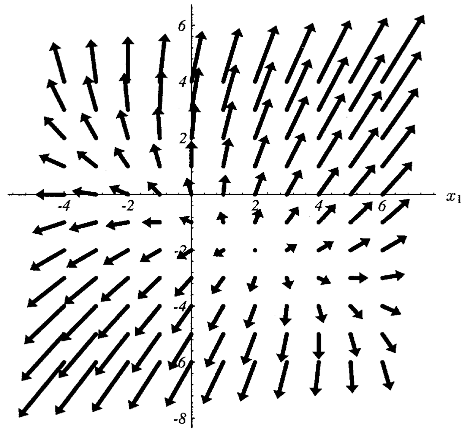


Graph of a quadratic form f(**x**)

Contours of the quadratic form



Gradient of the quadratic form



2. Eigenvalues and Eigenvectors

- For any $n \times n$ matrix $\mathbf{B}$, a scalar λ and a nonzero vector $\mathbf{v}$ that satisfy the equation

$$\mathbf{Bv}=\lambda\mathbf{v}$$

  are said to be the eigenvalue and the eigenvector of $\mathbf{B}$.

- If the matrix is *symmetric*, then the following properties hold:

  (a) the eigenvalues of $\mathbf{B}$ are real

  (b) eigenvectors associated with distinct eigenvalues are orthogonal

- The matrix $\mathbf{B}$ is *positive definite* (or positive semidefinite) if and only if all eigenvalues of $\mathbf{B}$ are positive (or nonnegative).

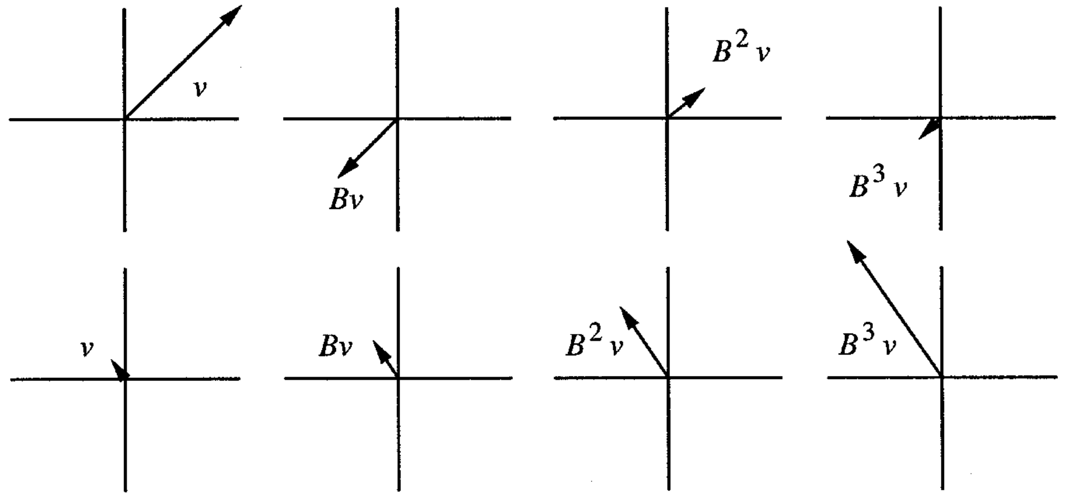
- Why should we care about the eigenvalues? *Iterative methods often depend on applying* $\mathbf{B}$ *to a vector over and over again*:

  (a) If $|\lambda|<1$, then $\mathbf{B}^i\mathbf{v}=\lambda^i\mathbf{v}$ vanishes as i approaches infinity

  (b) If $|\lambda|>1$, then $\mathbf{B}^i\mathbf{v}=\lambda^i\mathbf{v}$ will grow to infinity.

2. Eigenvalues and Eigenvectors (Cont'd)

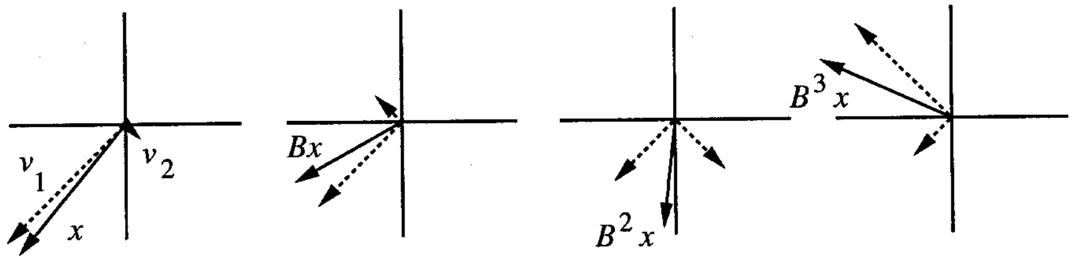
- Examples for $|\lambda|<1$ and $|\lambda| >1$



λ=-0.5

λ=2

- The spectral radius of a matrix is: $\rho(\mathbf{B})= \max|\lambda_i|$



λ_1=0.7
λ_2=-2

2. Eigenvalues and Eigenvectors (Cont'd)

- Example: The Jacobi Method for the solution of **Ax**=**b**
  - split the matrix **A** such that **A**=**D**+**E**
  - Instead of solving the system **Ax**=**b**, one solves the modified system **x**=**Bx**+**z**, where $\mathbf{B}=-\mathbf{D}^{-1}\mathbf{E}$ and $\mathbf{z}=\mathbf{D}^{-1}\mathbf{b}$
  - The iterative method is: $\mathbf{x}_{(i+1)}= \mathbf{B}\mathbf{x}_{(i)}+\mathbf{z}$, or it terms of the error vector: $\mathbf{e}_{(i+1)}= \mathbf{B}\mathbf{e}_{(i)}$
  - For our particular example, we have that the eigenvalues and the eigenvectors of the matrix **A** are:

$$\lambda_1=7, \quad \mathbf{v}_1=[1, 2]^T$$
$$\lambda_2=2, \quad \mathbf{v}_2=[-2, 1]^T$$

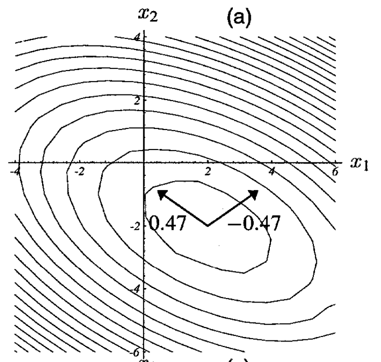
  - The eigenvalues and eigenvectors of the matrix **B** are:

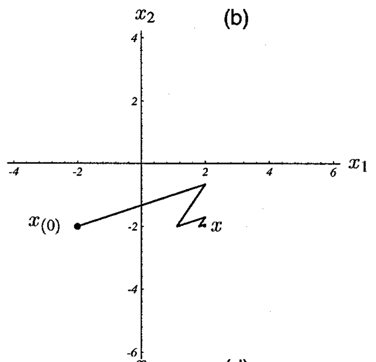
$$\lambda_1=-\sqrt{2}/3=-0.47, \qquad \mathbf{v}_1=[\sqrt{2}, 1]^T$$
$$\lambda_2= \sqrt{2}/3=0.47, \qquad \mathbf{v}_2=[-\sqrt{2}, 1]^T$$

2. Eigenvalues and Eigenvectors (Cont’d)

- Graphical representation of the convergence of the Jacobi method

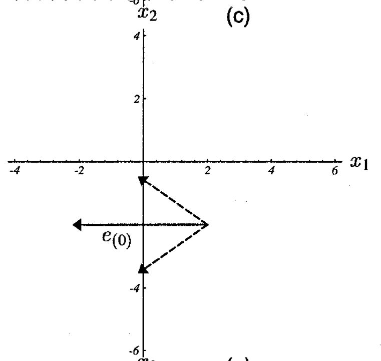
Eigenvectors of **B** *together with their eigenvalues*

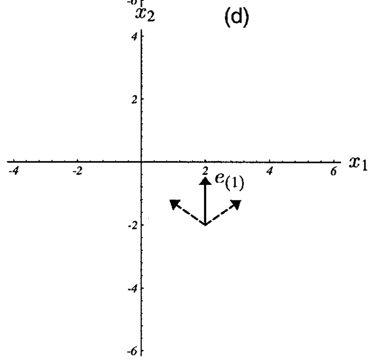




Convergence of the Jacobi method which starts at $[-2,-2]^T$ *and converges to* $[2, -2]^T$

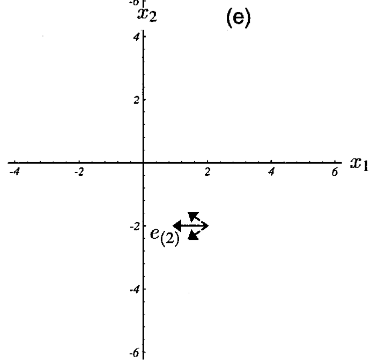
The error vector $\mathbf{e}_{(0)}$





The error vector $\mathbf{e}_{(1)}$

The error vector $\mathbf{e}_{(2)}$



3. The Method of Steepest Descent

- In the method of steepest descent, one starts with an arbitrary point $\mathbf{x}_{(0)}$ and takes a series of steps $\mathbf{x}_{(1)}$, $\mathbf{x}_{(2)}$, … until we are satis-fied that we are close enough to the solution.
- When taking the step, one chooses the direction in which f decreases most quickly, i.e.

$$-f'(\mathbf{x}_{(i)}) = \mathbf{b} - \mathbf{A}\mathbf{x}_{(i)}$$

- Definitions:

error vector: $\mathbf{e}_{(i)}=\mathbf{x}_{(i)}-\mathbf{x}$

residual: $\mathbf{r}_{(i)}=\mathbf{b}-\mathbf{A}\mathbf{x}_{(i)}$

- From **Ax**=**b**, it follows that

$$\mathbf{r}_{(i)}=-\mathbf{A}\mathbf{e}_{(i)}=-f'(\mathbf{x}_{(i)})$$

The residual is actually the direction of steepest descent

3. The Method of Steepest Descent (Cont’d)

- Start with some vector $\mathbf{x}_{(0)}$. The next vector falls along the solid line

$$\mathbf{x}_{(1)} = \mathbf{x}_{(0)} + \alpha \mathbf{r}_{(0)}$$

- The magnitude of the step is determined with a *line search* procedure that minimizes f along the line:

$$\frac{d}{d\alpha} f(\mathbf{x}_{(1)}) = f'(\mathbf{x}_{(1)})^T \frac{d\mathbf{x}_{(1)}}{d\alpha} = 0$$

$$-\mathbf{r}_{(1)}^T \mathbf{r}_{(0)} = 0$$

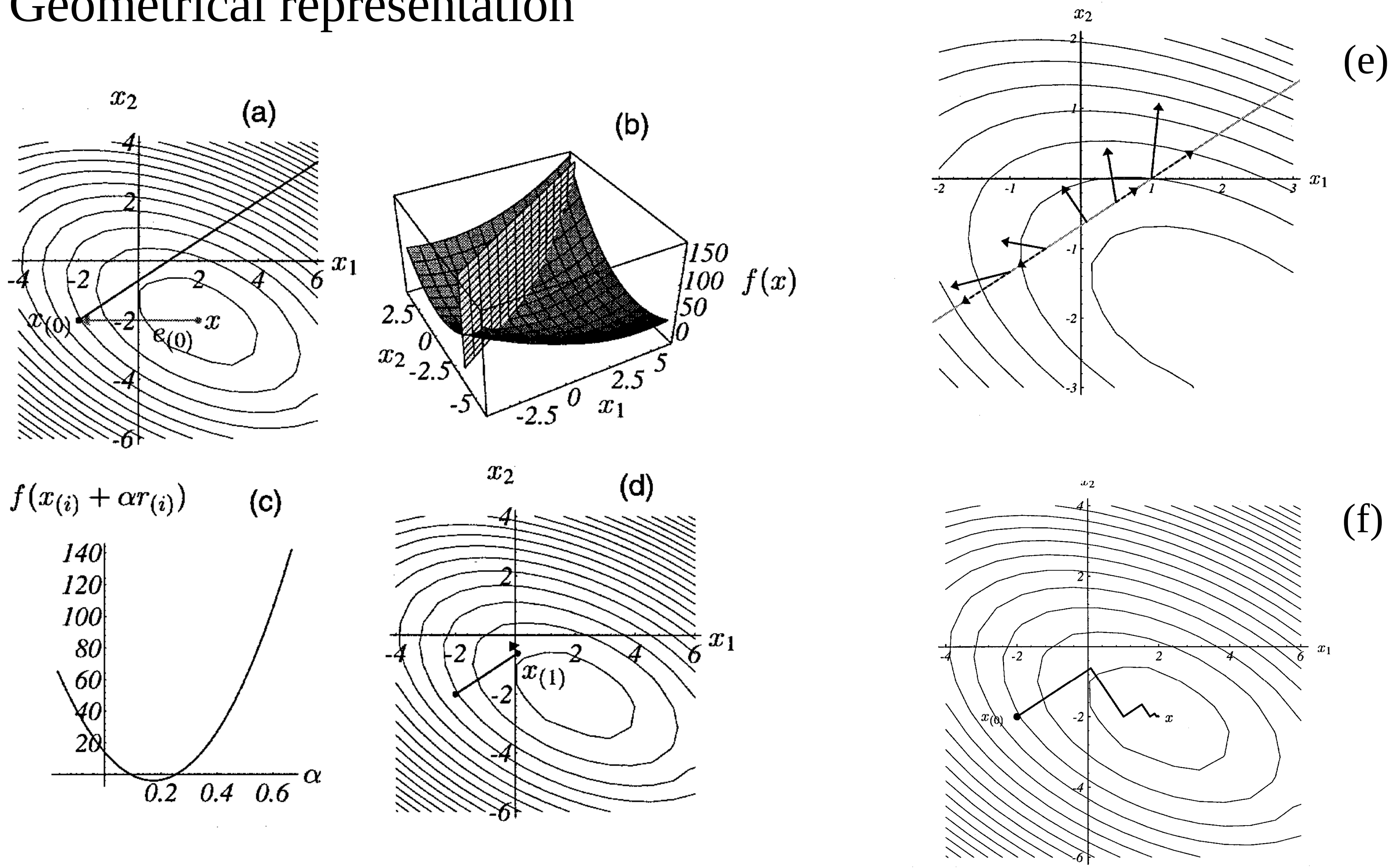
$$\left(\mathbf{b} - \mathbf{A}\mathbf{x}_{(1)}\right)^T \mathbf{r}_{(0)} = 0$$

$$\left(\mathbf{b} - \mathbf{A}\left(\mathbf{x}_{(0)} + \alpha \mathbf{r}_{(0)}\right)\right)^T \mathbf{r}_{(0)} = 0$$

$$\mathbf{r}_{(0)}^{\mathbf{T}} \mathbf{r}_{(0)} - \alpha\left(\mathbf{A}\mathbf{r}_{(0)}\right)^T \mathbf{r}_{(0)} = 0 \qquad \Rightarrow \qquad \alpha = \frac{\mathbf{r}_{(0)}^{\mathbf{T}} \mathbf{r}_{(0)}}{\mathbf{r}_{(0)}^{\mathbf{T}} \mathbf{A}\mathbf{r}_{(0)}}$$

3. The Method of Steepest Descent (Cont’d)

- Geometrical representation



3. The Method of Steepest Descent (Cont'd)

- The algorithm

$$\mathbf{r}_{(i)} = \mathbf{b} - \mathbf{A}\mathbf{x}_{(i)}$$

$$\alpha_{(i)} = \frac{\mathbf{r}_{(i)}^{\mathbf{T}}\mathbf{r}_{(i)}}{\mathbf{r}_{(i)}^{\mathbf{T}}\mathbf{A}\mathbf{r}_{(i)}}$$

Two matrix-vector multiplications are required.

$$\mathbf{x}_{(i+1)} = \mathbf{x}_{(i)} + \alpha_{(i)}\mathbf{r}_{(i)} \quad \Rightarrow \quad \mathbf{e}_{(i+1)} = \mathbf{e}_{(i)} + \alpha_{(i)}\mathbf{r}_{(i)}$$

- To avoid one matrix-vector multiplication, one uses

$$\mathbf{r}_{(i+1)} = \mathbf{r}_{(i)} - \alpha_{(i)}\mathbf{A}\mathbf{r}_{(i)}$$

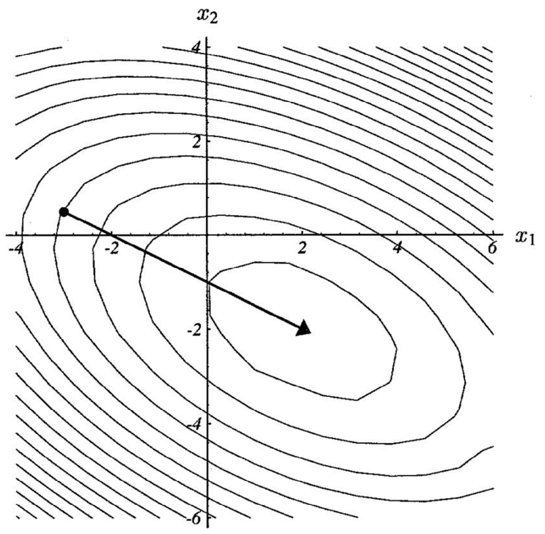
The disadvantage of using this recurrence is that the residual sequence is determined without any feedback from the value of $\mathbf{x}_{(i)}$, so that round-off errors may cause $\mathbf{x}_{(i)}$ to converge to some point near $\mathbf{x}$.

4. Convergence Analysis of the Method of Steepest Descent

- The error vector $\mathbf{e}_{(i)}$ is equal to the eigenvector $\mathbf{v}_j$ of $\mathbf{A}$

$$\mathbf{r}_{(i)} = -\mathbf{A}\mathbf{e}_{(i)} = -\mathbf{A}\mathbf{v}_j = -\lambda_j \mathbf{v}_j$$

$$\alpha_{(i)} = \frac{\mathbf{r}_{(i)}^T \mathbf{r}_{(i)}}{\mathbf{r}_{(i)}^T \mathbf{A}\mathbf{r}_{(i)}} = \frac{1}{\lambda_j}, \quad \mathbf{e}_{(i+1)} = \mathbf{e}_{(i)} + \alpha_{(i)}\mathbf{r}_{(i)} = 0$$

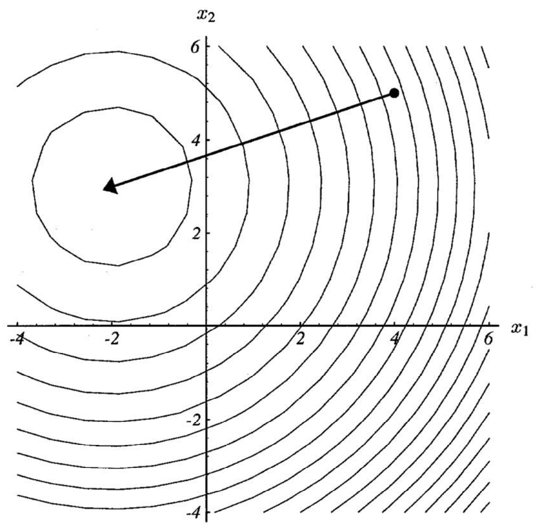


- The error vector $\mathbf{e}_{(i)}$ is a linear combination of the eigenvectors of $\mathbf{A}$, and all eigenvalues are the same

$$\mathbf{e}_{(i)} = \sum_{j=1}^{n} \xi_j \mathbf{v}_j$$

$$\mathbf{r}_{(i)} = -\mathbf{A}\mathbf{e}_{(i)} = -\sum_{j=1}^{n} \xi_j \lambda_j \mathbf{v}_j = -\lambda \sum_{j=1}^{n} \xi_j \mathbf{v}_j$$

$$\alpha_{(i)} = \frac{\mathbf{r}_{(i)}^T \mathbf{r}_{(i)}}{\mathbf{r}_{(i)}^T \mathbf{A}\mathbf{r}_{(i)}} = \frac{1}{\lambda}, \quad \mathbf{e}_{(i+1)} = \mathbf{e}_{(i)} + \alpha_{(i)}\mathbf{r}_{(i)} = 0$$



3. The Method of Steepest Descent

- In the method of steepest descent, one starts with an arbitrary point $\mathbf{x}_{(0)}$ and takes a series of steps $\mathbf{x}_{(1)}$, $\mathbf{x}_{(2)}$, … until we are satis-fied that we are close enough to the solution.
- When taking the step, one chooses the direction in which f decreases most quickly, i.e.

$$-f'(\mathbf{x}_{(i)}) = \mathbf{b} - \mathbf{A}\mathbf{x}_{(i)}$$

- Definitions:

error vector: $\mathbf{e}_{(i)}=\mathbf{x}_{(i)}-\mathbf{x}$

residual: $\mathbf{r}_{(i)}=\mathbf{b}-\mathbf{A}\mathbf{x}_{(i)}$

- From **Ax**=**b**, it follows that

$$\mathbf{r}_{(i)}=-\mathbf{A}\mathbf{e}_{(i)}=-f'(\mathbf{x}_{(i)})$$

The residual is actually the direction of steepest descent

3. The Method of Steepest Descent (Cont'd)

- Start with some vector $\mathbf{x}_{(0)}$. The next vector falls along the solid line

$$\mathbf{x}_{(1)} = \mathbf{x}_{(0)} + \alpha \mathbf{r}_{(0)}$$

- The magnitude of the step is determined with a *line search* procedure that minimizes f along the line:

$$\frac{d}{d\alpha} f(\mathbf{x}_{(1)}) = f'(\mathbf{x}_{(1)})^T \frac{d\mathbf{x}_{(1)}}{d\alpha} = 0$$

$$-\mathbf{r}_{(1)}^T \mathbf{r}_{(0)} = 0$$

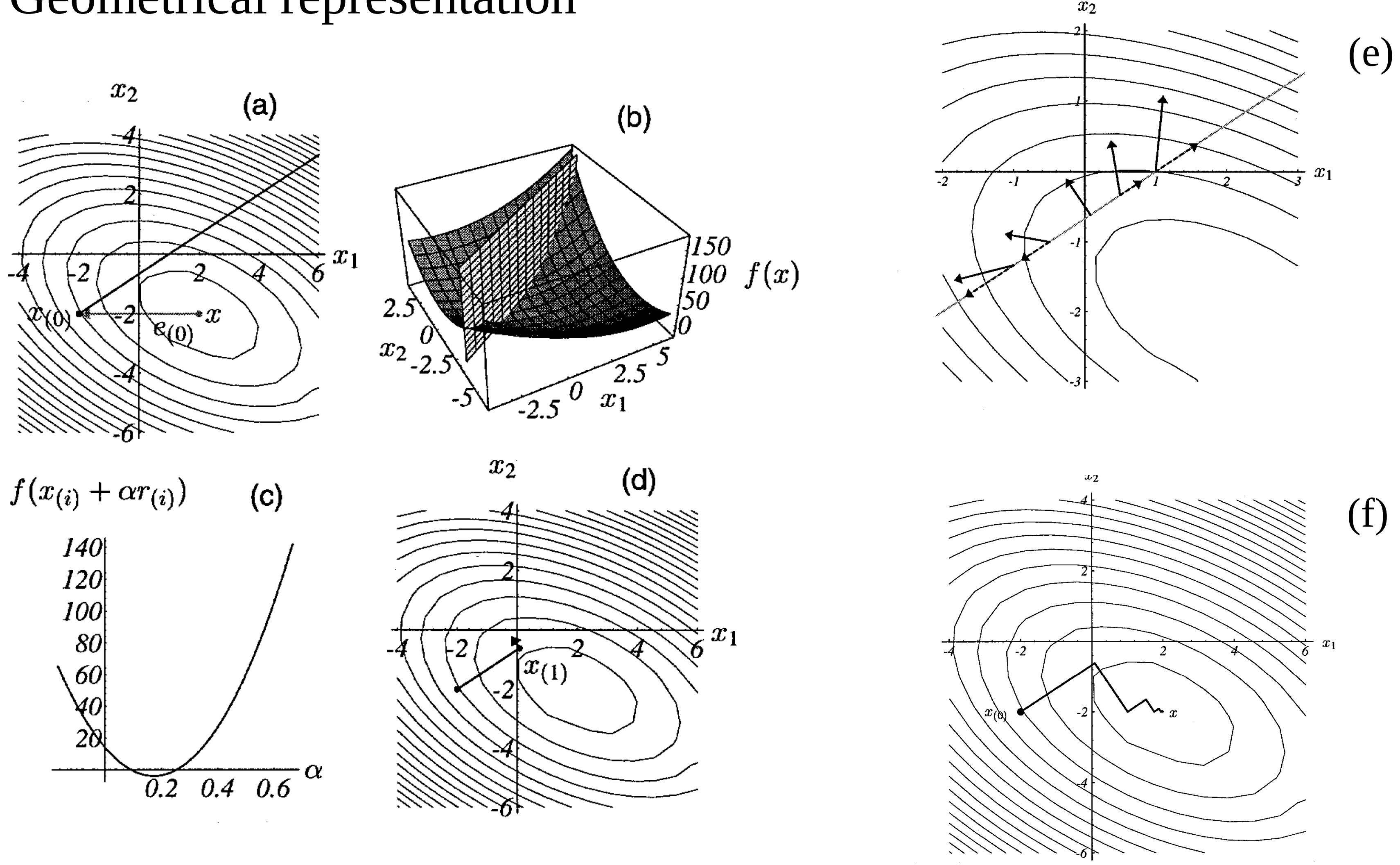
$$\left(\mathbf{b} - \mathbf{A}\mathbf{x}_{(1)}\right)^T \mathbf{r}_{(0)} = 0$$

$$\left(\mathbf{b} - \mathbf{A}\left(\mathbf{x}_{(0)} + \alpha \mathbf{r}_{(0)}\right)\right)^T \mathbf{r}_{(0)} = 0$$

$$\mathbf{r}_{(0)}^{\mathbf{T}} \mathbf{r}_{(0)} - \alpha \left(\mathbf{A}\mathbf{r}_{(0)}\right)^T \mathbf{r}_{(0)} = 0 \qquad \Rightarrow \qquad \alpha = \frac{\mathbf{r}_{(0)}^{\mathbf{T}} \mathbf{r}_{(0)}}{\mathbf{r}_{(0)}^{\mathbf{T}} \mathbf{A}\mathbf{r}_{(0)}}$$

3. The Method of Steepest Descent (Cont’d)

- Geometrical representation



3. The Method of Steepest Descent (Cont'd)

- Summary of the algorithm

$$\mathbf{r}_{(i)} = \mathbf{b} - \mathbf{A}\mathbf{x}_{(i)}$$

$$\alpha_{(i)} = \frac{\mathbf{r}_{(i)}^{\mathbf{T}}\mathbf{r}_{(i)}}{\mathbf{r}_{(i)}^{\mathbf{T}}\mathbf{A}\mathbf{r}_{(i)}}$$

$$\mathbf{x}_{(i+1)} = \mathbf{x}_{(i)} + \alpha_{(i)}\mathbf{r}_{(i)}$$

- To avoid one matrix-vector multiplication, one uses instead

$$\mathbf{r}_{(i+1)} = \mathbf{r}_{(i)} - \alpha_{(i)}\mathbf{A}\mathbf{r}_{(i)}$$

- Efficient implementation of the method of steepest descent

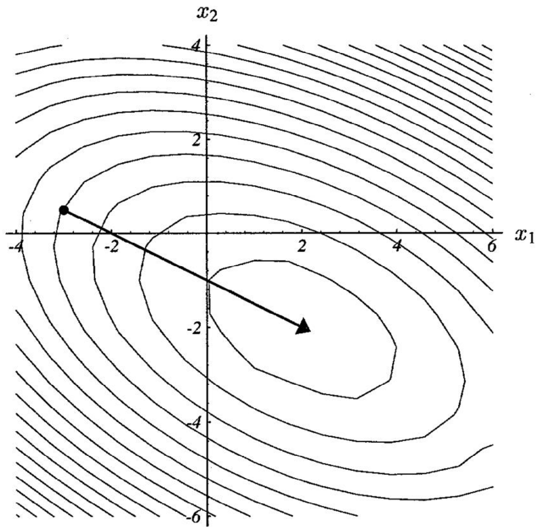
```
$i \Leftarrow 0$
$\mathbf{r} \Leftarrow \mathbf{b} - \mathbf{A}\mathbf{x}$
$\delta \Leftarrow \mathbf{r}^T\mathbf{r}$
While $i < i_{max}$ and $\delta > \varepsilon^2\delta_0$
    $\mathbf{q} \Leftarrow \mathbf{A}\mathbf{r}$
    $\alpha \Leftarrow \frac{\delta}{\mathbf{r}^T\mathbf{q}}$
    $\mathbf{x} \Leftarrow \mathbf{x} + \alpha\mathbf{r}$
    If i is divisible by 50
        $\mathbf{r} \Leftarrow \mathbf{b} - \mathbf{A}\mathbf{x}$
    else
        $\mathbf{r} \Leftarrow \mathbf{r} - \alpha\mathbf{q}$
    endif
    $\delta \Leftarrow \mathbf{r}^T\mathbf{r}$
    $i \Leftarrow i + 1$
```

4. Convergence Analysis of the Method of Steepest Descent

- The error vector $\mathbf{e}_{(i)}$ is equal to the eigenvector $\mathbf{v}_j$ of $\mathbf{A}$

$$\mathbf{r}_{(i)} = -\mathbf{A}\mathbf{e}_{(i)} = -\mathbf{A}\mathbf{v}_j = -\lambda_j \mathbf{v}_j$$

$$\alpha_{(i)} = \frac{\mathbf{r}_{(i)}^{\mathbf{T}} \mathbf{r}_{(i)}}{\mathbf{r}_{(i)}^{\mathbf{T}} \mathbf{A}\mathbf{r}_{(i)}} = \frac{1}{\lambda_j}, \quad \mathbf{e}_{(i+1)} = \mathbf{e}_{(i)} + \alpha_{(i)}\mathbf{r}_{(i)} = 0$$

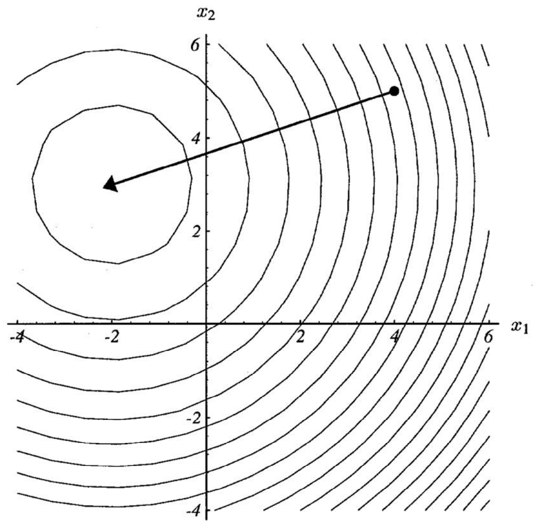


- The error vector $\mathbf{e}_{(i)}$ is a linear combination of the eigenvectors of $\mathbf{A}$, and all eigenvalues are the same

$$\mathbf{e}_{(i)} = \sum_{j=1}^{n} \xi_j \mathbf{v}_j$$

$$\mathbf{r}_{(i)} = -\mathbf{A}\mathbf{e}_{(i)} = -\sum_{j=1}^{n} \xi_j \lambda_j \mathbf{v}_j = -\lambda \sum_{j=1}^{n} \xi_j \mathbf{v}_j$$

$$\alpha_{(i)} = \frac{\mathbf{r}_{(i)}^{\mathbf{T}} \mathbf{r}_{(i)}}{\mathbf{r}_{(i)}^{\mathbf{T}} \mathbf{A}\mathbf{r}_{(i)}} = \frac{1}{\lambda}, \quad \mathbf{e}_{(i+1)} = \mathbf{e}_{(i)} + \alpha_{(i)}\mathbf{r}_{(i)} = 0$$



4. Convergence Analysis of the Method of Steepest Descent (Cont’d)

- General convergence of the method can be proven by calculating the energy norm

$$\left\|\mathbf{e}_{(i+1)}\right\|_A^2 = \mathbf{e}_{(i+1)}^T \mathbf{A}\mathbf{e}_{(i+1)} = \left\|\mathbf{e}_{(i)}\right\|_A^2 \omega^2, \quad \omega^2 = 1 - \frac{\left(\sum_j \xi_j^2 \lambda_j^2\right)^2}{\left(\sum_j \xi_j^2 \lambda_j^3\right)\left(\sum_j \xi_j^2 \lambda_j\right)}$$

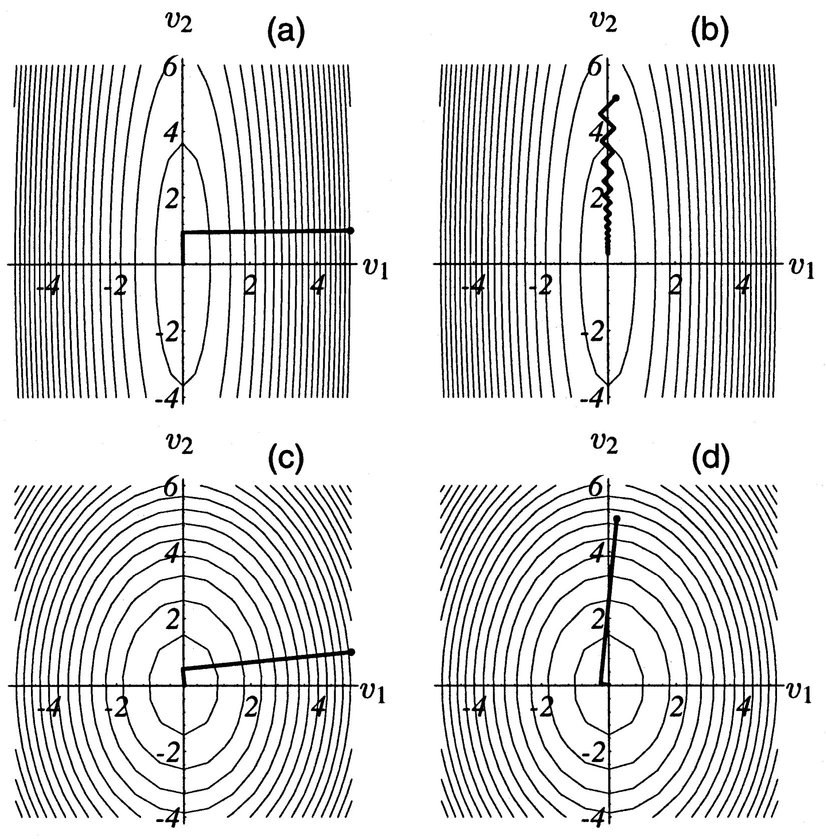
- For n=2, one has that

$$\omega^2 = 1 - \frac{(\kappa^2 + \mu^2)^2}{(\kappa + \mu^2)(\kappa^3 + \mu^2)} \le \frac{\kappa - 1}{\kappa + 1}$$

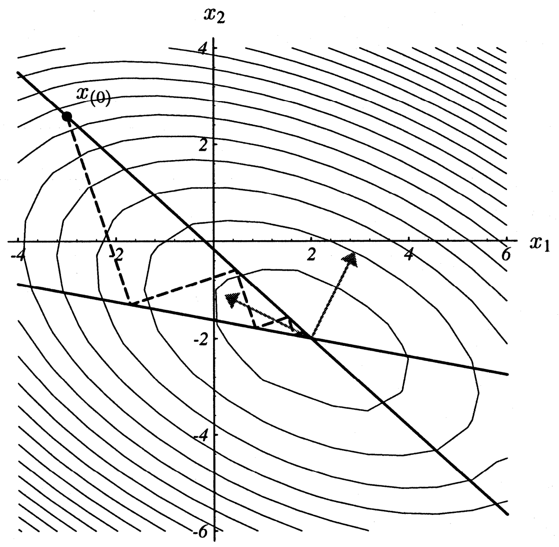
$$\kappa = \lambda_{max} / \lambda_{min}, \quad \mu = \xi_2 / \xi_1$$

Conditioning number

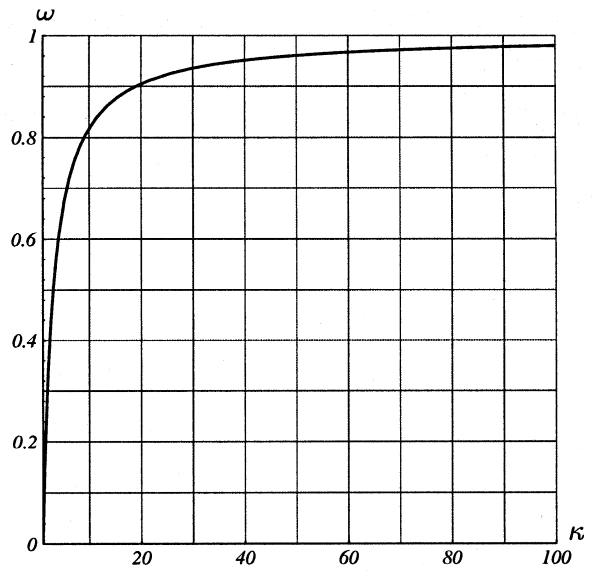
4. Convergence Analysis of the Method of Steepest Descent (Cont'd)



The influence of the spectral condition number on the convergence



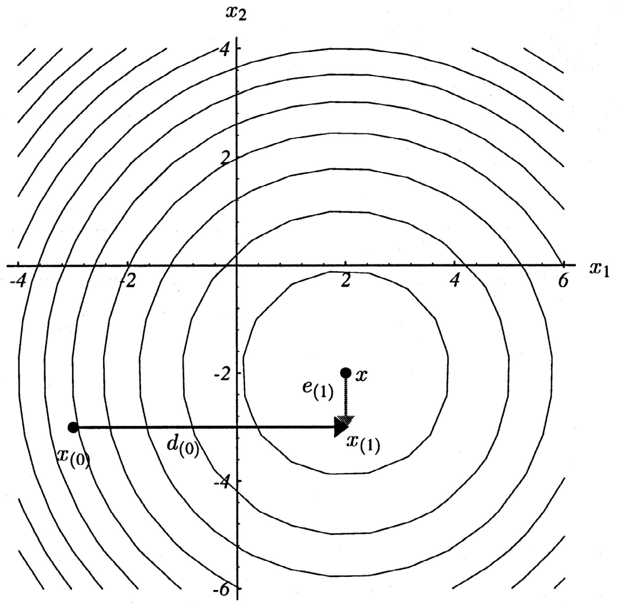
Worst-case scenario: $\mu=\pm\kappa$



5. The Method of Conjugate Directions

- Basic idea:

  1. Pick a set of orthogonal search directions $\mathbf{d}_{(0)}$, $\mathbf{d}_{(1)}$, … , $\mathbf{d}_{(n-1)}$
  2. Take exactly one step in each search direction to line up with $\mathbf{x}$



- Mathematical formulation:

  1. For each step we choose a point

$$\mathbf{x}_{(i+1)}=\mathbf{x}_{(i)}+\alpha_{(i)}\,\mathbf{d}_{(i)}$$

  2. To find $\alpha_{(i)}$, we use the fact that $\mathbf{e}_{(i+1)}$ is orthogonal to $\mathbf{d}_{(i)}$

$$\mathbf{d}_{(i)}^T\mathbf{e}_{(i+1)} = 0$$

$$\mathbf{d}_{(i)}^T\left(\mathbf{e}_{(i)} + \alpha_{(i)}\mathbf{d}_{(i)}\right) = 0$$

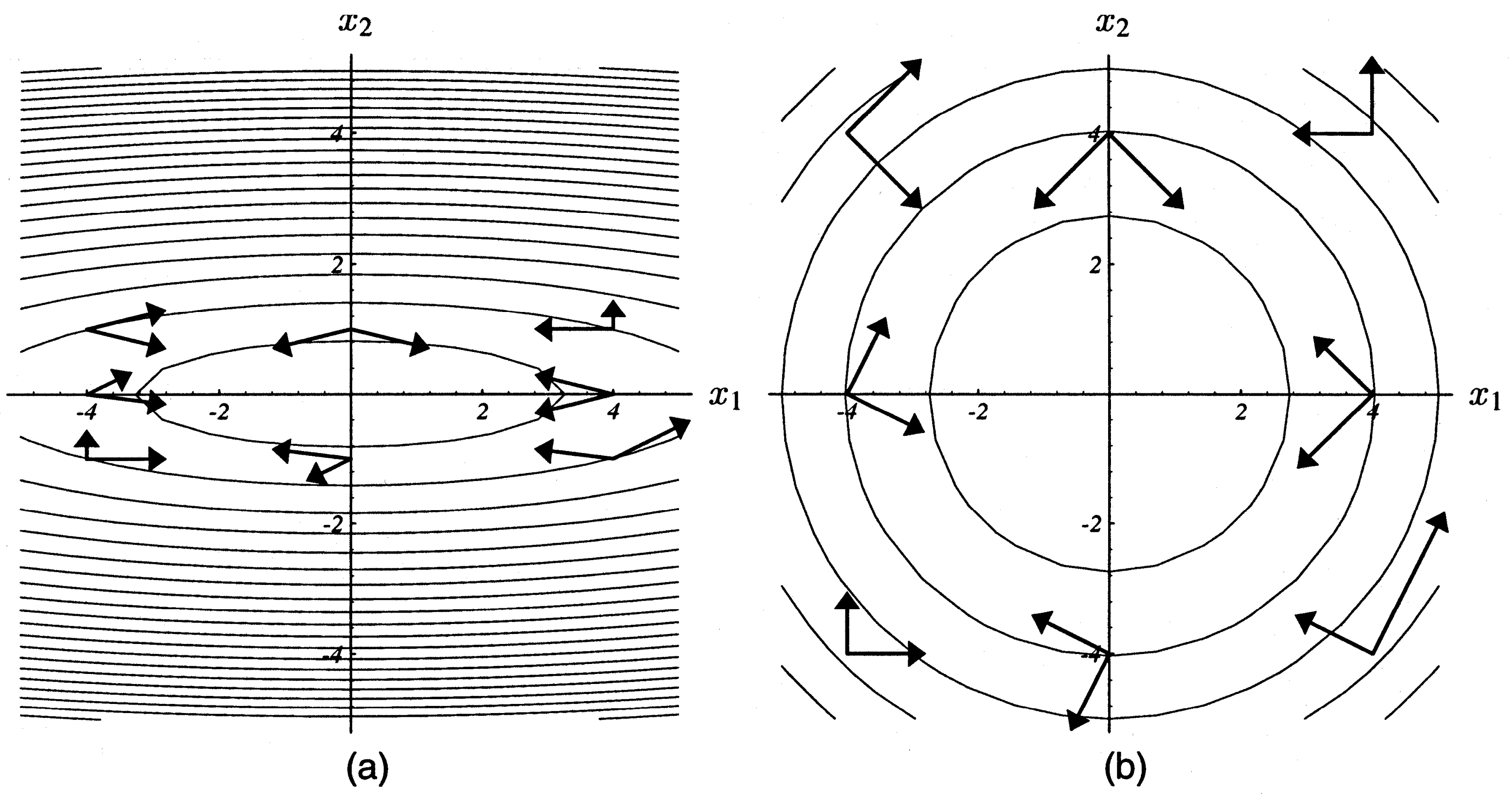
$$\mathbf{d}_{(i)}^T\mathbf{e}_{(i)} + \alpha_{(i)}\mathbf{d}_{(i)}^T\mathbf{d}_{(i)} = 0$$

$$\alpha_{(i)} = -\frac{\mathbf{d}_{(i)}^T\mathbf{e}_{(i)}}{\mathbf{d}_{(i)}^T\mathbf{d}_{(i)}}$$

5. The Method of Conjugate Directions (Cont'd)

- To solve the problem of not knowing $\mathbf{e}_{(i)}$, one makes the search directions to be A-orthogonal rather then orthogonal to each other, i.e.:

$$\mathbf{d}_{(i)}^{T} A \mathbf{d}_{(j)} = 0$$



5. The Method of Conjugate Directions (Cont'd)

- The new requirement is now that $\mathbf{e}_{(i+1)}$ is A-orthogonal to $\mathbf{d}_{(i)}$

$$\frac{d}{d\alpha} f(\mathbf{x}_{(i+1)}) = f'(\mathbf{x}_{(i+1)})^T \frac{d\mathbf{x}_{(i+1)}}{d\alpha} = 0$$

$$\mathbf{r}_{(i+1)}^T \mathbf{d}_{(i)} = 0$$

$$\mathbf{d}_{(i)}^T \mathbf{A} \mathbf{e}_{(i+1)} = 0$$

$$\mathbf{d}_{(i)}^T \mathbf{A}\left(\mathbf{e}_{(i)} + \alpha_{(i)} \mathbf{d}_{(i)}\right) = 0$$

$$\alpha_{(i)} = \frac{\mathbf{d}_{(i)}^{\mathbf{T}} \mathbf{r}_{(i)}}{\mathbf{d}_{(i)}^{\mathbf{T}} \mathbf{A} \mathbf{d}_{(i)}}$$

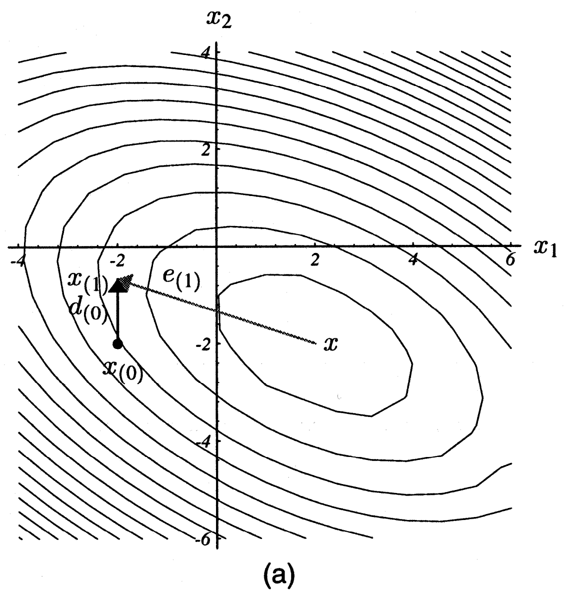
If the search vectors were the residuals, this formula would be identical to the method of steepest descent.

5. The Method of Conjugate Directions (Cont'd)

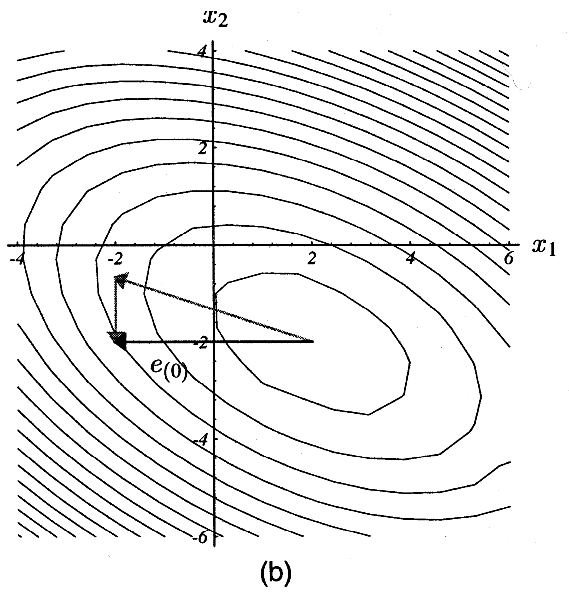
- Proof that this process computes **x** in n-steps
  - express the error terms as a linear combination of the search directions

$$\mathbf{e}_{(0)} = \sum_{j=0}^{n-1} \delta_{(j)} \mathbf{d}_{(j)}$$

  - use the fact that the search directions are A-orthogonal



(a)

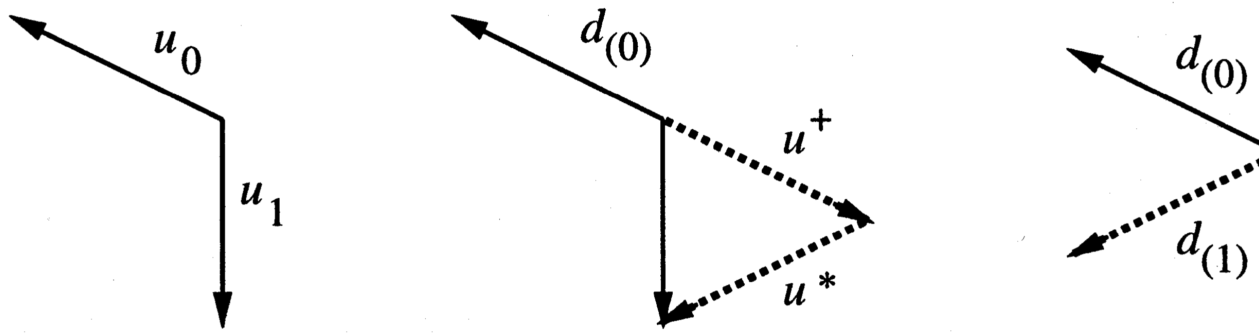


(b)

5. The Method of Conjugate Directions (Cont'd)

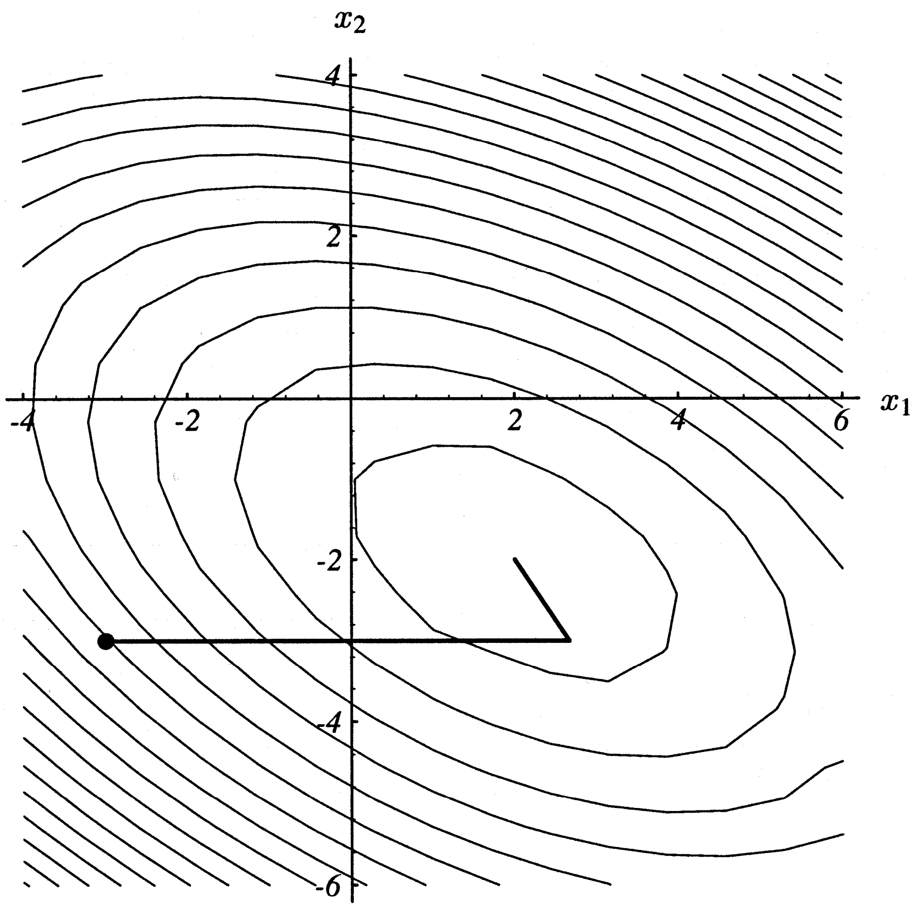
- Calculation of the A-ortogonal search directions by a **conjugate Gram-Schmidth process**
  1. Take a set of linearly independent vectors $\mathbf{u}_0$, $\mathbf{u}_1$, … , $\mathbf{u}_{n-1}$
  2. Assume that $\mathbf{d}_{(0)}=\mathbf{u}_0$
  3. For i>0, take an $\mathbf{u}_i$ and subtracts all the components from it that are not A-orthogonal to the previous search directions

$$\mathbf{d}_{(i)} = \mathbf{u}_{(i)} + \sum_{j=0}^{i-1} \beta_{ij}\mathbf{d}_{(j)} \quad , \quad \beta_{ij} = -\frac{\mathbf{u}_{(i)}^T\mathbf{A}\mathbf{d}_{(j)}}{\mathbf{d}_{(j)}^T\mathbf{A}\mathbf{d}_{(j)}}$$



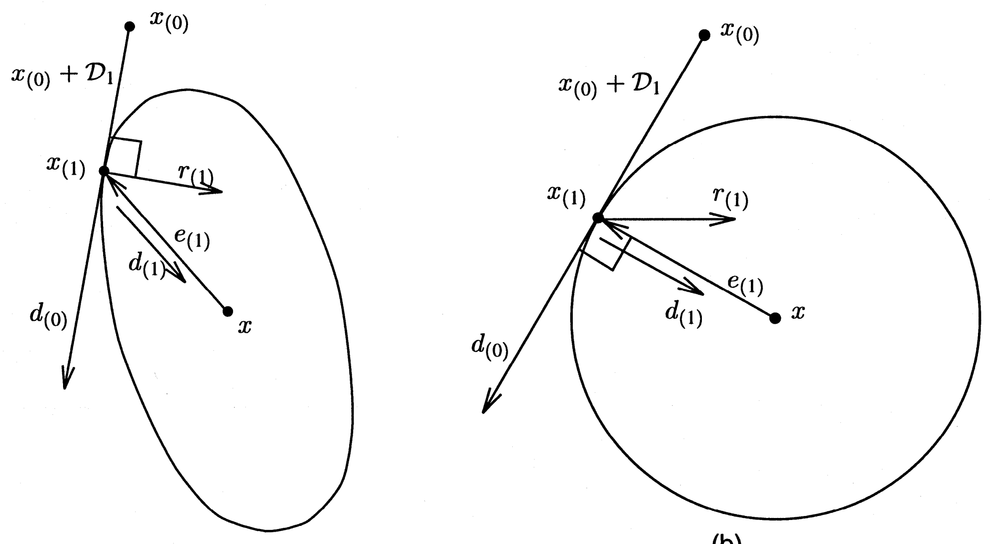
5. The Method of Conjugate Directions (Cont'd)

- The method of Conjugate Directions using the axial unit vectors as basis vectors



5. The Method of Conjugate Directions (Cont'd)

- Important observations:



(a) (b)

1. The error vector is A-orthogonal to all previous search directions
2. The residual vector is orthogonal to all previous search directions
3. The residual vector is also orthogonal to all previous basis vectors.

$$\mathbf{d}_i^T \mathbf{A} \mathbf{e}_j = -\mathbf{d}_i^T \mathbf{r}_j = -\mathbf{u}_i^T \mathbf{r}_j = 0 \quad \text{for} \quad i < j$$

6. The Method of Conjugate Gradients

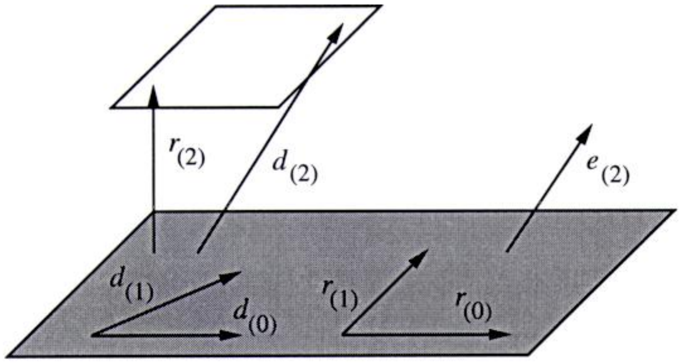
- The method of Conjugate Gradients is simply the method of conjugate directions where the search directions are constructed by conjugation of the residuals, i.e. $\mathbf{u}_i = \mathbf{r}_{(i)}$
- This allows us to simplify the calculation of the new search direction because

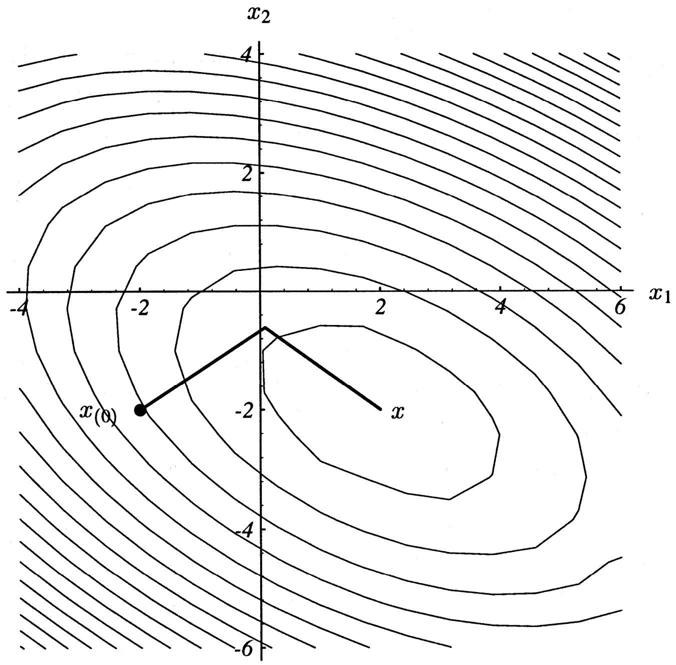
$$\beta_{ij} = \begin{cases} \dfrac{1}{\alpha_{(i-1)}} \dfrac{\mathbf{r}_{(i)}^T \mathbf{r}_{(i)}}{\mathbf{d}_{(i-1)}^T \mathbf{A} \mathbf{d}_{(i-1)}} = \dfrac{\mathbf{r}_{(i)}^T \mathbf{r}_{(i)}}{\mathbf{r}_{(i-1)}^T \mathbf{r}_{(i-1)}} & i = j+1 \\ 0 & i > j+1 \end{cases}$$

- The new search direction is determined as a linear combination of the previous search direction and the new residual

$$\mathbf{d}_{(i+1)} = \mathbf{r}_{(i+1)} + \beta_i \mathbf{d}_{(i)}$$

6. The Method of Conjugate Gradients (Cont'd)





- Efficient implementation

```
$i \Leftarrow 0$
$\mathbf{r} \Leftarrow \mathbf{b} - \mathbf{Ax}$
$\mathbf{d} \Leftarrow \mathbf{r}$
$\delta_{new} \Leftarrow \mathbf{r}^T \mathbf{r}$
$\delta_0 \Leftarrow \delta_{new}$
While   $i < i_{max}$   and   $\delta_{new} > \varepsilon^2 \delta_0$
    $\mathbf{q} \Leftarrow \mathbf{Ad}$
    $\alpha \Leftarrow \frac{\delta_{new}}{\mathbf{d}^T \mathbf{q}}$
    $\mathbf{x} \Leftarrow \mathbf{x} + \alpha \mathbf{d}$
    If i is divisible by 50
        $\mathbf{r} \Leftarrow \mathbf{b} - \mathbf{Ax}$
    else
        $\mathbf{r} \Leftarrow \mathbf{r} - \alpha \mathbf{q}$
    endif
    $\delta_{old} \Leftarrow \delta_{new}$
    $\delta_{new} \Leftarrow \mathbf{r}^T \mathbf{r}$
    $\beta \Leftarrow \frac{\delta_{new}}{\delta_{old}}$
    $\mathbf{d} \Leftarrow \mathbf{r} + \beta \mathbf{d}$
    $i \Leftarrow i + 1$
```

The Landscape of Ax=b Solvers

	Direct A = LU	Iterative y’ = Ay
Non-symmetric	**Pivoting LU**	**GMRES, BiCGSTAB, …**
Symmetric positive definite	**Cholesky**	**Conjugate gradient**

More General

$\updownarrow$

More Robust

More Robust $\longleftrightarrow$ **Less Storage (if sparse)**

Conjugate gradient iteration

```
$x_0 = 0$,   $r_0 = b$,   $d_0 = r_0$
for k = 1, 2, 3, . . .
    $\alpha_k = (r^T_{k-1} r_{k-1}) / (d^T_{k-1} A d_{k-1})$    step length
    $x_k = x_{k-1} + \alpha_k d_{k-1}$    approx solution
    $r_k = r_{k-1} - \alpha_k A d_{k-1}$    residual
    $\beta_k = (r^T_k r_k) / (r^T_{k-1} r_{k-1})$    improvement
    $d_k = r_k + \beta_k d_{k-1}$    search direction
```

- One matrix-vector multiplication per iteration
- Two vector dot products per iteration
- Four n-vectors of working storage

7. Convergence Analysis of the CG Method

- If the algorithm is performed in exact arithmetic, the exact solution is obtained in at most *n*-steps.
- When finite precision arithmetic is used, rounding errors lead to gradual loss of orthogonality among the residuals, and the finite termination property of the method is lost.
- If the matrix **A** has only m distinct eigenvalues, then the CG will converge in at most m iterations
- An error bound on the CG method can be obtained in terms of the A-norm, and after k-iterations:

$$\|\mathbf{x}-\mathbf{x}_k\|_A \leq 2\|\mathbf{x}-\mathbf{x}_0\|_A \left[\frac{\sqrt{\kappa(\mathbf{A})}-1}{\sqrt{\kappa(\mathbf{A})}+1}\right]^k$$

8. Complexity of the CG Method

- Dominant operations of Steepest Descent or Conjugate Gradient method are the matrix vector products. Thus, both methods have spatial complexity $O(m)$.
- Suppose that after I-iterations, one requires that $\|e_{(i)}\|_A \le \varepsilon \|e_{(0)}\|_A$

  (a) Steepest Descent: $i \le \left\lfloor \frac{1}{2} \kappa \ln\left(\frac{1}{\varepsilon}\right) \right\rfloor$

  (b) Conjugate Gradient: $i \le \left\lfloor \frac{1}{2} \sqrt{\kappa} \ln\left(\frac{2}{\varepsilon}\right) \right\rfloor$
- The time complexity of these two methods is:

  (a) Steepest Descent: $O(m\kappa)$

  (b) Conjugate Gradient: $O(m\sqrt{\kappa})$
- Stopping criterion: $\|r_{(i)}\| \le \varepsilon \|r_{(0)}\|$

9. Preconditioning Techniques

References:

- J.A. Meijerink and H.A. van der Vorst, “An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix,” Mathematics of Computation, Vol. 31, No. 137, pp. 148-162 (1977).
- J.A. Meijerink and H.A. van der Vorst, “Guidelines for the usage of incomplete decompositions in solving sets of linear equations as they occur in practical problems,” Journal of Computational Physics, Vol. 44, pp. 134-155 (1981).
- D.S. Kershaw, “The incomplete Cholesky-Conjugate gradient method for the iterative solution of systems of linear equations,” Journal of Computational Physics, Vol. 26, pp. 43-65 (1978).
- H.A. van der Vorst, “High performance preconditioning,” SIAM Journal of Scientific Statistical Computations, Vol. 10, No. 6, pp. 1174-1185 (1989).

9. Preconditioning Techniques (Cont'd)

- Preconditioning is a technique for improving the condition number of a matrix. Suppose that **M** is a symmetric, positive-definite matrix that approximates **A**, but is easier to invert. Then, rather than solving the original system

$$\mathbf{Ax}=\mathbf{b}$$

  one solves the modified system

$$\mathbf{M}^{-1}\mathbf{Ax}=\mathbf{M}^{-1}\mathbf{b}$$

- The effectiveness of the preconditioner depends upon the condition number of $\mathbf{M}^{-1}\mathbf{A}$.
- Intuitively, the preconditioning is an attempt to stretch the quadratic form to appear more spherical, so that the eigenvalues become closer to each other.
- Derivation:

$$\mathbf{M}=\mathbf{E}\mathbf{E}^{T}, \quad \mathbf{E}^{-1}\mathbf{A}\mathbf{E}^{-T}\tilde{\mathbf{x}}=\mathbf{E}^{-1}\mathbf{b}, \; \tilde{\mathbf{x}}=\mathbf{E}^{T}\mathbf{x}$$

9. Preconditioning Techniques (Cont'd)

- Transformed Preconditioned CG (PCG) Algorithm

```
$i \Leftarrow 0,\ \tilde{\mathbf{r}} \Leftarrow \mathbf{E}^{-1}\mathbf{b} - \mathbf{E}^{-1}\mathbf{A}\mathbf{E}^{-T}\tilde{\mathbf{x}},\ \tilde{\mathbf{d}} \Leftarrow \tilde{\mathbf{r}}$
$\delta_{new} \Leftarrow \tilde{\mathbf{r}}^T\tilde{\mathbf{r}},\ \delta_0 \Leftarrow \delta_{new}$
$While\ \ i < i_{\max}\ \ and\ \ \delta_{new} > \varepsilon^2\delta_0$
    $\tilde{\mathbf{q}} \Leftarrow \mathbf{E}^{-1}\mathbf{A}\mathbf{E}^{-T}\tilde{\mathbf{d}}$
    $\alpha \Leftarrow \frac{\delta_{new}}{\tilde{\mathbf{d}}^T\tilde{\mathbf{q}}}$
    $\tilde{\mathbf{x}} \Leftarrow \tilde{\mathbf{x}} + \alpha\tilde{\mathbf{d}}$
    $\tilde{\mathbf{r}} \Leftarrow \mathbf{E}^{-1}\mathbf{b} - \mathbf{E}^{-1}\mathbf{A}\mathbf{E}^{-T}\tilde{\mathbf{x}}\ \ or\ \ \tilde{\mathbf{r}} \Leftarrow \tilde{\mathbf{r}} - \alpha\tilde{\mathbf{q}}$
    $\delta_{old} \Leftarrow \delta_{new}$
    $\delta_{new} \Leftarrow \tilde{\mathbf{r}}^T\tilde{\mathbf{r}}$
    $\beta \Leftarrow \frac{\delta_{new}}{\delta_{old}}$
    $\tilde{\mathbf{d}} \Leftarrow \tilde{\mathbf{r}} + \beta\tilde{\mathbf{d}}$
    $i \Leftarrow i + 1$
```

- Untransformed PCG Algorithm

$$\tilde{\mathbf{r}} = \mathbf{E}^{-1}\mathbf{r},\ \tilde{\mathbf{d}} = \mathbf{E}^T\mathbf{d}$$

```
$i \Leftarrow 0,\ \mathbf{r} \Leftarrow \mathbf{b} - \mathbf{A}\mathbf{x},\ \mathbf{d} \Leftarrow \mathbf{M}^{-1}\mathbf{r}$
$\delta_{new} \Leftarrow \mathbf{r}^T\mathbf{M}^{-1}\mathbf{r},\ \delta_0 \Leftarrow \delta_{new}$
$While\ \ i < i_{\max}\ \ and\ \ \delta_{new} > \varepsilon^2\delta_0$
    $\mathbf{q} \Leftarrow \mathbf{A}\mathbf{d}$
    $\alpha \Leftarrow \frac{\delta_{new}}{\mathbf{d}^T\mathbf{q}}$
    $\mathbf{x} \Leftarrow \mathbf{x} + \alpha\mathbf{d}$
    $\mathbf{r} \Leftarrow \mathbf{b} - \mathbf{A}\mathbf{x}\ \ or\ \ \mathbf{r} \Leftarrow \mathbf{r} - \alpha\mathbf{q}$
    $\delta_{old} \Leftarrow \delta_{new}$
    $\delta_{new} \Leftarrow \mathbf{r}^T\mathbf{M}^{-1}\mathbf{r}$
    $\beta \Leftarrow \frac{\delta_{new}}{\delta_{old}}$
    $\mathbf{d} \Leftarrow \mathbf{M}^{-1}\mathbf{r} + \beta\mathbf{d}$
    $i \Leftarrow i + 1$
```

9. Preconditioning Techniques (Cont’d)

There are sevaral types of preconditioners that can be used:

(a) Preconditioners based on splitting of the matrix **A**, i.e.

$$\mathbf{A} = \mathbf{M} - \mathbf{N}$$

(b) Complete or incomplete factorization of A, e.g.

$$\mathbf{A} = \mathbf{L}\mathbf{L}^{T} + \mathbf{E}$$

(c) Approximation of $\mathbf{M}=\mathbf{A}^{-1}$

(d) Reordering of the equations and/or unknowns, e.g.

Domain Decomposition

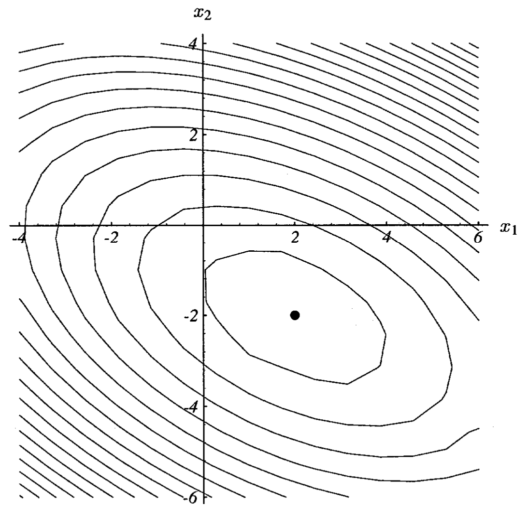
9. Preconditioning Techniques (Cont'd)

(a) Preconditioning based on splittings of **A**

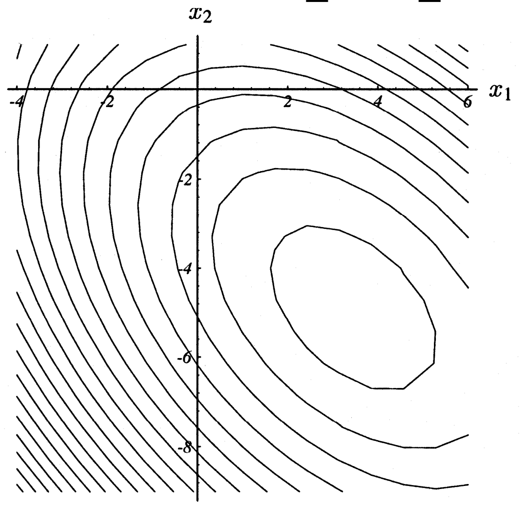
- Diagonal Preconditioning:

$$\mathbf{M}=\mathbf{D}=\text{diag}\{a_{11}, a_{22}, \ldots\, a_{nn}\}$$

$$\mathbf{A}=\begin{bmatrix}3 & 2\\ 2 & 6\end{bmatrix},\quad \mathbf{b}=\begin{bmatrix}2\\ -8\end{bmatrix},\quad c=0 \;\Rightarrow\; \mathbf{x}=\begin{bmatrix}2\\ -2\end{bmatrix}$$



$\Longrightarrow$



- Tridiagonal Preconditioning
- Block Diagonal Preconditioning

9. Preconditioning Techniques (Cont'd)

(b) Preconditioning based on factorization of **A**

- Cholesky Factorization (for symmetric and positive definite matrix) as a direct method:

$$\mathbf{A} = \mathbf{L}\mathbf{L}^{\mathrm{T}}$$

$$\mathbf{x} = (\mathbf{L}^{-\mathrm{T}})(\mathbf{L}^{-1}\mathbf{b})$$

where:

$$L_{ii} = \sqrt{A_{ii} - \sum_{k=1}^{i-1} L_{ik}}$$

$$L_{ji} = \frac{A_{ji} - \sum_{k=1}^{i-1} L_{jk} L_{ik}}{L_{ii}}, \quad j = i+1, i+2, \cdots, n$$

- Modified Cholesky factorization: $\mathbf{A} = \mathbf{L}\mathbf{D}\mathbf{L}^{\mathrm{T}}$

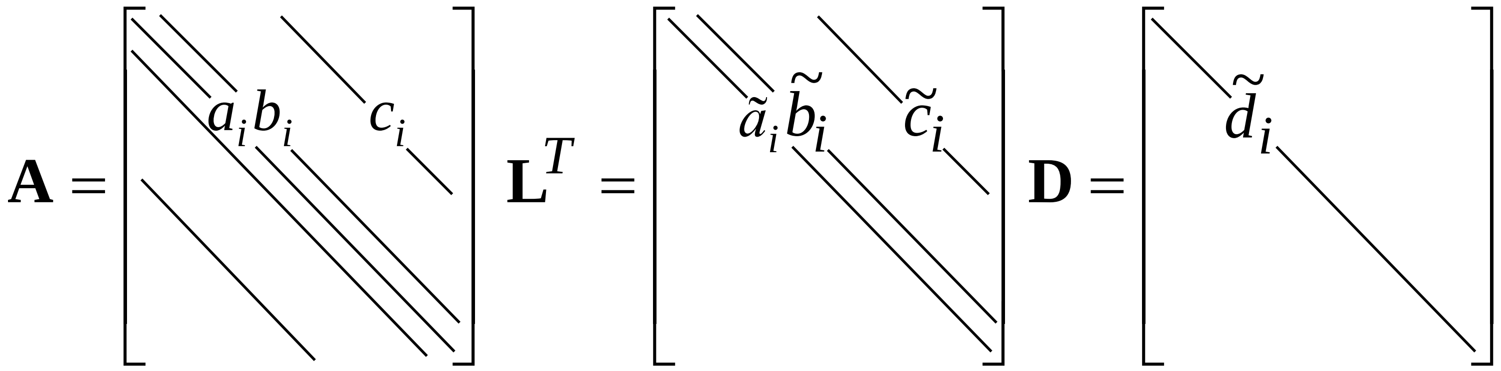
9. Preconditioning Techniques (Cont'd)

(b) Preconditioning based on factorization of **A**

- Incomplete Cholesky Factorization:

$$\mathbf{A} = \mathbf{L}\mathbf{L}^T + \mathbf{E} \text{ or } \mathbf{A} = \mathbf{L}\mathbf{D}\mathbf{L}^T + \mathbf{E}\text{ , where } L_{ij}=0 \text{ if } (i,j)\in P$$

Simplest choice is: P={(i,j)| a_{ij}=0; i,j,=1,2,…,n} => ICCG(0)



$$\tilde{a}_i = \tilde{d}_i^{-1} = a_i - \tilde{b}_{i-1}^2\tilde{d}_{i-1} - \tilde{c}_{i-m}^2\tilde{d}_{i-m} \quad , \quad \tilde{b}_i = b_i,\ \tilde{c}_i = c_i,\ i = 1,2,\cdots,n$$

- Modified ICCG(0): $\mathbf{M} = (\mathbf{L}+\mathbf{D})\mathbf{D}^{-1}(\mathbf{D}+\mathbf{L}^T)$

$$diag(\mathbf{A}) = diag(\mathbf{M}), \quad d_i = a_i - \frac{b_{i-1}^2}{d_{i-1}} - \frac{c_{i-m}^2}{d_{i-m}}$$

9. Preconditioning Techniques (Cont'd)

(c) Preconditioning based on approximation of **A**

- If $\rho(\mathbf{J})<1$, then the inverse of **I-J** is

$$(\mathbf{I}-\mathbf{J})^{-1} = \sum_{k=0}^{\infty} \mathbf{J}^k = \mathbf{I}+\mathbf{J}+\mathbf{J}^2+\mathbf{J}^3+\cdots$$

- Write matrix **A** as:

$$\mathbf{A}=\mathbf{D}+\mathbf{L}+\mathbf{U}=\mathbf{D}(\mathbf{I}+\mathbf{D}^{-1}(\mathbf{L}+\mathbf{U})) \Rightarrow \mathbf{J}= -\mathbf{D}^{-1}(\mathbf{L}+\mathbf{U}))$$

- The inverse of **A** can be expressed as:

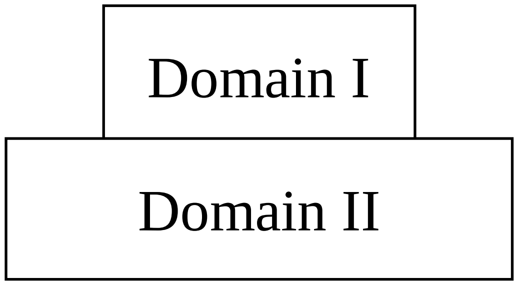
$$\mathbf{A}^{-1} = (\mathbf{D}(\mathbf{I}-\mathbf{J}))^{-1} = (\mathbf{I}-\mathbf{J})^{-1}\mathbf{D}^{-1}$$

$$= \sum_{k=0}^{\infty} (-1)^k \left(\mathbf{D}^{-1}(\mathbf{L}+\mathbf{U})\right)^k \mathbf{D}^{-1}$$

- For k=1, one has that: $\mathbf{M}^{-1} = \mathbf{D}^{-1} - \mathbf{D}^{-1}(\mathbf{L}+\mathbf{U})\mathbf{D}^{-1}$
- For k=m, one usually uses: $\mathbf{M}_m^{-1} = \sum_{k=0}^{m} \gamma_k (-1)^k \left(\mathbf{D}^{-1}(\mathbf{L}+\mathbf{U})\right)^k \mathbf{D}^{-1}$

9. Preconditioning Techniques (Cont'd)

(d) Domain Decomposition Preconditioning



$$\mathbf{Ax} = \mathbf{b}, \quad \begin{bmatrix} \mathbf{A}_1 & \mathbf{0} & \mathbf{B}_1 \\ \mathbf{0} & \mathbf{A}_2 & \mathbf{B}_2 \\ \mathbf{B}_1^T & \mathbf{B}_2^T & \mathbf{Q} \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \mathbf{z} \end{bmatrix} = \begin{bmatrix} \mathbf{d}_1 \\ \mathbf{d}_2 \\ \mathbf{f} \end{bmatrix}$$

$$\mathbf{M} = \mathbf{L} \begin{bmatrix} \mathbf{M}_1^{-1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{M}_2^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{S}^{-1} \end{bmatrix} \mathbf{L}^T, \quad \textit{where} \quad \mathbf{L} = \begin{bmatrix} \mathbf{M}_1 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{M}_2 & \mathbf{0} \\ \mathbf{B}_1^T & \mathbf{B}_2^T & \mathbf{S} \end{bmatrix}$$

$$\mathbf{M} = \begin{bmatrix} \mathbf{M}_1 & \mathbf{0} & \mathbf{B}_1 \\ \mathbf{0} & \mathbf{M}_2 & \mathbf{B}_2 \\ \mathbf{B}_1^T & \mathbf{B}_2^T & \mathbf{S}^* \end{bmatrix}, \quad \mathbf{S}^* = \mathbf{S} + \mathbf{B}_1^T \mathbf{M}_1^{-1} \mathbf{B}_1 + \mathbf{B}_2^T \mathbf{M}_2^{-1} \mathbf{B}_2$$

10. Conjugate Gradient Type Algorithms for Nonsymmetric Matrices

- The **Bi-Conjugate Gradient** (BCG) Algorithm was proposed by Lanczos in 1954.
- It solves not only the original system **Ax**=**b** but also the dual linear system $\mathbf{A}^T\mathbf{x}^*=b^*$.
- Each step of this algorithm requires a matrix-by-vector product with both **A** and $\mathbf{A}^T$.
- The search direction $\mathbf{p}_j^*$ does not contribute to the solution directly.

```
$i \Leftarrow 0,\ \ \mathbf{r} \Leftarrow \mathbf{b} - \mathbf{Ax},\ \ \mathbf{r}^T\mathbf{r}^* \neq \mathbf{0}$
$\mathbf{p} \Leftarrow \mathbf{r},\ \mathbf{p}^* = \mathbf{r}^*,\ \delta_{new} \Leftarrow \mathbf{r}^T\mathbf{r}^*,\ \delta_0 \Leftarrow \delta_{new}$
$While\ \ i < i_{\max}\ \ and\ \ \delta_{new} > \varepsilon^2\delta_0$
    $\mathbf{q} \Leftarrow \mathbf{Ap}$
    $\mathbf{q}^* \Leftarrow \mathbf{A}^T\mathbf{p}^*$
    $\alpha \Leftarrow \frac{\delta_{new}}{\mathbf{p}^{*T}\mathbf{q}}$
    $\mathbf{x} \Leftarrow \mathbf{x} + \alpha\mathbf{p}$
    $\mathbf{r} \Leftarrow \mathbf{r} - \alpha\mathbf{q}$
    $\mathbf{r}^* \Leftarrow \mathbf{r}^* - \alpha\mathbf{q}^*$
    $\delta_{old} \Leftarrow \delta_{new}$
    $\delta_{new} \Leftarrow \mathbf{r}^{*T}\mathbf{r}$
    $\beta \Leftarrow \frac{\delta_{new}}{\delta_{old}}$
    $\mathbf{p} \Leftarrow \mathbf{r} + \beta\mathbf{p}$
    $\mathbf{p}^* \Leftarrow \mathbf{r}^* + \beta\mathbf{p}^*$
    $i \Leftarrow i + 1$
```

10. Conjugate Gradient Type Algorithms for Nonsymmetric Matrices (Cont'd)

- The **Conjugate Gradient Squared** (CGS) Algorithm was developed by Sonneveld in 1984.
- Main idea of the algorithm:

$$\mathbf{r}_j = \phi_j(\mathbf{A})\mathbf{r}_0$$
$$\mathbf{p}_j = \pi_j(\mathbf{A})\mathbf{r}_0$$
$$\mathbf{r}^*_j = \phi_j(\mathbf{A}^T)\mathbf{r}^*_0$$
$$\mathbf{p}^*_j = \pi_j(\mathbf{A}^T)\mathbf{r}^*_0$$

```
$i \Leftarrow 0,\ \mathbf{r}_0 \Leftarrow \mathbf{b} - \mathbf{A}\mathbf{x}_0,\ \mathbf{r}_0^*$ is arbitrary
$\mathbf{p} \Leftarrow \mathbf{r}_0, \mathbf{u} = \mathbf{r}_0$
$\delta_{new} \Leftarrow \mathbf{r}_0^{*T}\mathbf{r},\ \delta_0 \Leftarrow \delta_{new}$
$While\ \ i < i_{\max}\ \ and\ \ \delta_{new} > \varepsilon^2\delta_0$
    $\alpha \Leftarrow \dfrac{\delta_{new}}{\mathbf{r}_0^{*T}\mathbf{A}\mathbf{p}}$
    $\mathbf{q} \Leftarrow \mathbf{u} - \alpha\mathbf{A}\mathbf{p}$
    $\mathbf{x} \Leftarrow \mathbf{x} + \alpha(\mathbf{u} + \mathbf{q})$
    $\mathbf{r} \Leftarrow \mathbf{r} - \alpha\mathbf{A}(\mathbf{u} + \mathbf{q})$
    $\delta_{old} \Leftarrow \delta_{new}$
    $\delta_{new} \Leftarrow \mathbf{r}_0^{*T}\mathbf{r}$
    $\beta \Leftarrow \dfrac{\delta_{new}}{\delta_{old}}$
    $\mathbf{u} \Leftarrow \mathbf{r} + \beta\mathbf{q}$
    $\mathbf{p} \Leftarrow \mathbf{u} + \beta(\mathbf{q} + \beta\mathbf{p})$
    $i \Leftarrow i + 1$
```

10. Conjugate Gradient Type Algorithms for Nonsymmetric Matrices (Cont'd)

- The Bi-**Conjugate Gradient Stabilized (**Bi-CGSTAB) Algorithm was developed by van der Vorst in 1992.
- Rather than producing iterates whose residual vectors are of the form

$$\mathbf{r}_j = \phi_j^2(\mathbf{A})\mathbf{r}_0$$

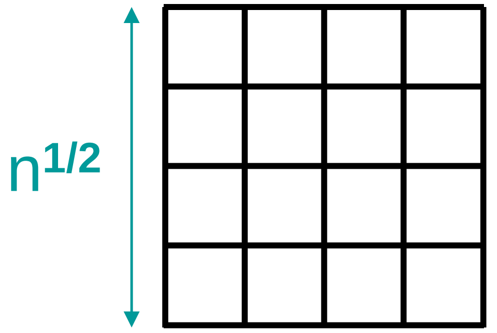
- it produces iterates with residual vectors of the form

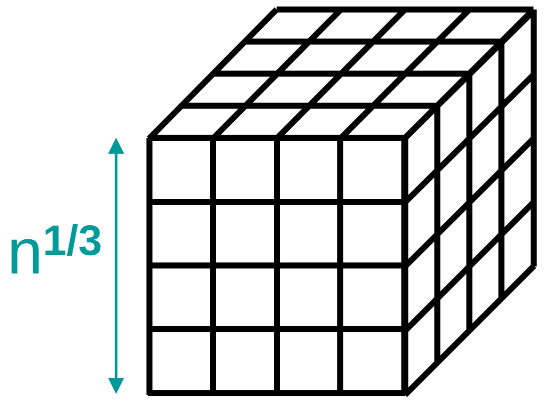
$$\mathbf{r}_j = \psi_j(\mathbf{A})\phi_j(\mathbf{A})\mathbf{r}_0$$
$$\mathbf{p}_j = \psi_j(\mathbf{A})\pi_j(\mathbf{A})\mathbf{r}_0$$

```
$i \Leftarrow 0,\quad \mathbf{r}_0 \Leftarrow \mathbf{b} - \mathbf{A}\mathbf{x}_0,\ \mathbf{r}_0^*$ is arbitrary
$\mathbf{p} \Leftarrow \mathbf{r}_0$
$\delta_{new} \Leftarrow \mathbf{r}_0^{*T}\mathbf{r},\ \delta_0 \Leftarrow \delta_{new}$
$While\ \ i < i_{\max}\ \ and\ \ \delta_{new} > \varepsilon^2\delta_0$
    $\alpha \Leftarrow \dfrac{\delta_{new}}{\mathbf{r}_0^{*T}\mathbf{A}\mathbf{p}}$
    $\mathbf{s} \Leftarrow \mathbf{r} - \alpha\mathbf{A}\mathbf{p}$
    $\omega \Leftarrow \dfrac{\mathbf{s}^T\mathbf{A}\mathbf{s}}{(\mathbf{A}\mathbf{s})^T\mathbf{A}\mathbf{s}}$
    $\mathbf{x} \Leftarrow \mathbf{x} + \alpha\mathbf{p} + \omega\mathbf{s}$
    $\mathbf{r} \Leftarrow \mathbf{s} - \omega\mathbf{A}\mathbf{s}$
    $\delta_{old} \Leftarrow \delta_{new}$
    $\delta_{new} \Leftarrow \mathbf{r}_0^{*T}\mathbf{r}$
    $\beta \Leftarrow \dfrac{\delta_{new}}{\delta_{old}} \times \dfrac{\alpha}{\omega}$
    $\mathbf{p} \Leftarrow \mathbf{r} + \beta(\mathbf{p} - \omega\mathbf{A}\mathbf{p})$
    $i \Leftarrow i + 1$
```

Complexity of linear solvers

Time to solve model problem (Poisson's equation) on regular mesh





	2D	3D
Sparse Cholesky:	$O(n^{1.5})$	$O(n^{2})$
CG, exact arithmetic:	$O(n^{2})$	$O(n^{2})$
CG, no precond:	$O(n^{1.5})$	$O(n^{1.33})$
CG, modified IC:	$O(n^{1.25})$	$O(n^{1.17})$
CG, support trees:	$O(n^{1.20}) \to O(n^{1+})$	$O(n^{1.75}) \to O(n^{1.31})$
Multigrid:	$O(n)$	$O(n)$