The Conjugate Gradient Method:

- Lecture 2:
  - Iterative methods for system of linear equations: The conjugate gradient method
  - Computer Tutorial 3: Implementation

Reference: J. R. Shewchuk, “An introduction to the conjugate gradient method without the agonizing pain,” (1994). available at: http://www.cs.cmu.edu/~jrs/jrspapers.html

2.1: The Conjugate Gradient Method

- Objective: Given a *Hermitian* matrix *A*, and a vector *b*, solve the linear system

$$Ax = b$$

$$\begin{bmatrix} A_{11} & A_{12} & \dots & A_{1n} \\ A_{21} & A_{22} & & A_{2n} \\ \vdots & & \ddots & \vdots \\ A_{n1} & A_{n2} & \dots & A_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

Some linear algebra:

- A: *Hermitian* matrix and symmetric, positive definite

$z^{\mathrm{T}}Az > 0$ for all nonzero vectors, *z,* with real elements.

positive definite example:

$$A=\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix},\quad z=\begin{bmatrix}z_0\\ z_1\end{bmatrix}\rightarrow$$ [redacted] $${}_0 \quad z_1\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}z_0\\ z_1\end{bmatrix}=z_0^2+z_1^2>0.$$

non-positive definite example:

$$A=\begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix},\quad z=\begin{bmatrix}1\\ -1\end{bmatrix}\rightarrow$$ [redacted] $$-1\begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}\begin{bmatrix}1\\ -1\end{bmatrix}=-2<0.$$

Some linear algebra:

- Inner product (dot, scalar) of two vectors, *x*, *y*:

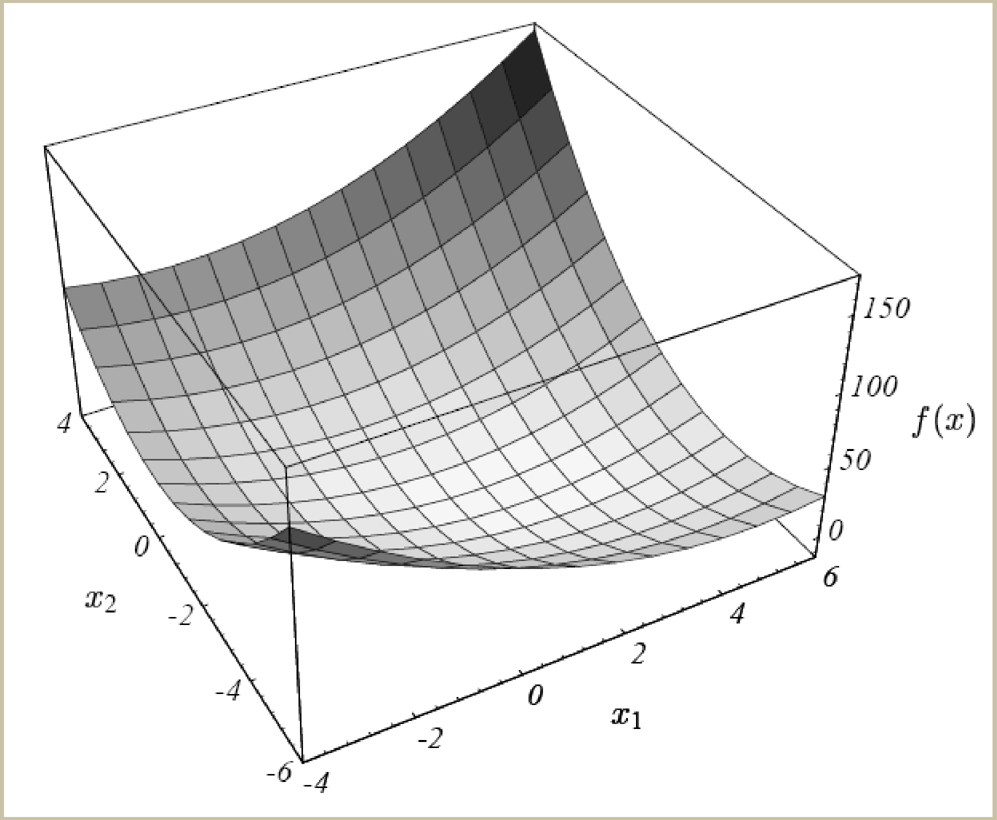
$$\langle x, y \rangle = x^T y = y^T x = \sum_{i=1}^{n} x_i y_i$$

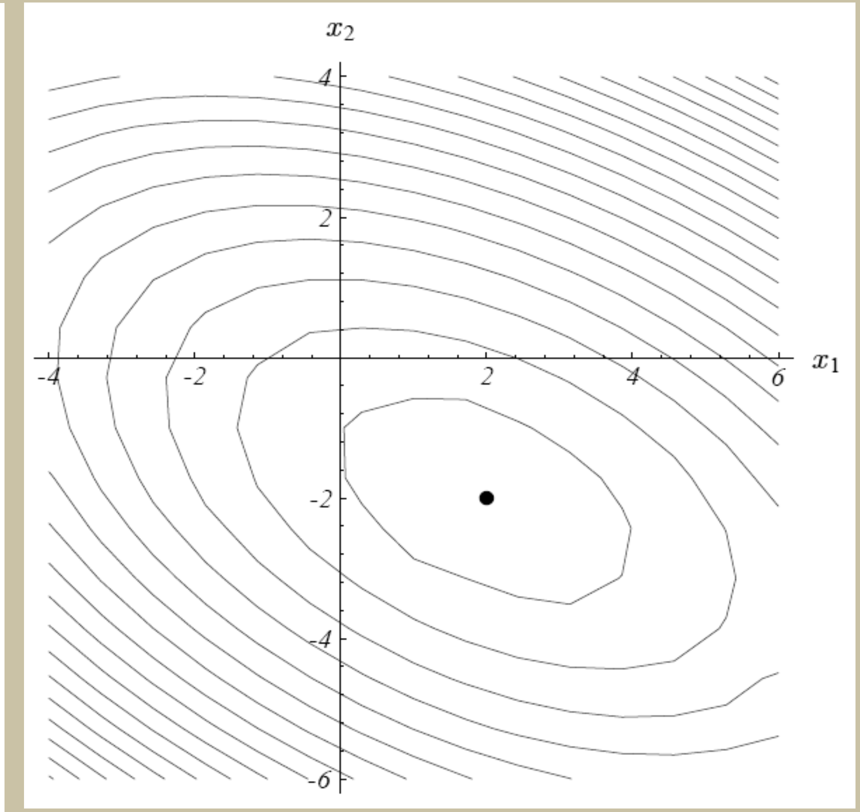
- Orthogonal vectors: <*x*,*y*> = 0.
- Transpose of matrix multiplication: $(AB)^T = B^T A^T$
- Inverse of matrix multiplication: $(AB)^{-1} = B^{-1} A^{-1}$
- The quadratic form:

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

The quadratic form:

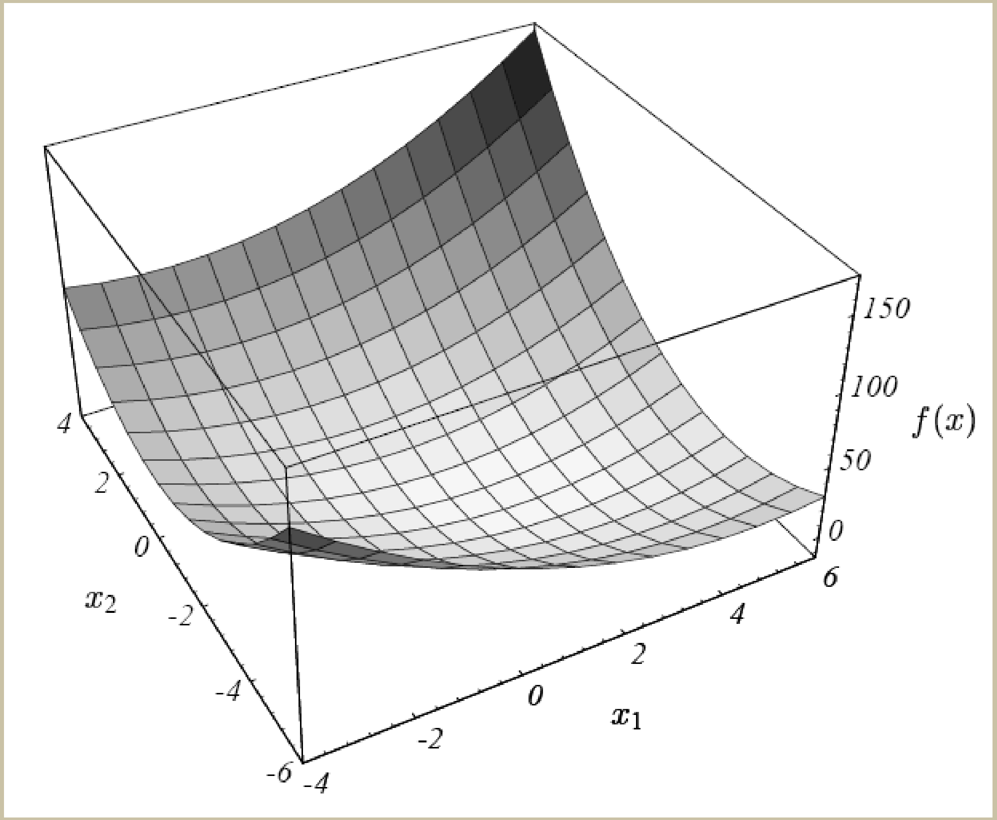
- Example: $A = \begin{bmatrix} 3 & 2 \\ 2 & 6 \end{bmatrix}, \quad b = \begin{bmatrix} 2 \\ -8 \end{bmatrix}, \quad c = 0.$
- Plot of *f(x):*

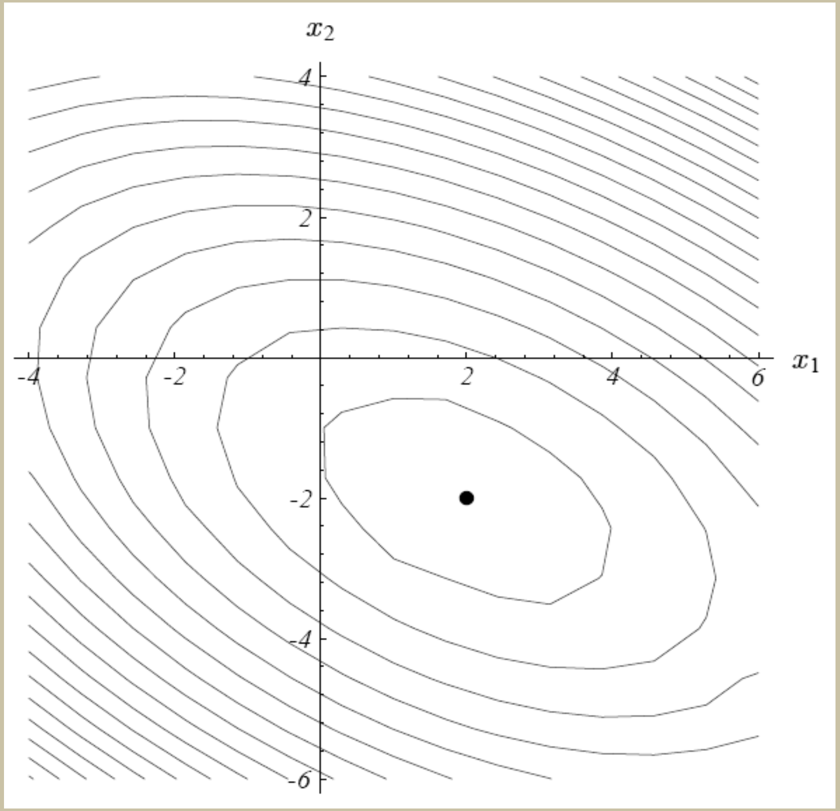




The quadratic form and Ax=b

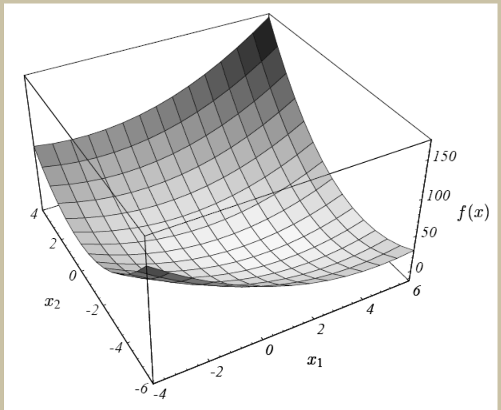
- Solution of $Ax = b$: $x = [2, -2]^T$ Where is it on the figure?

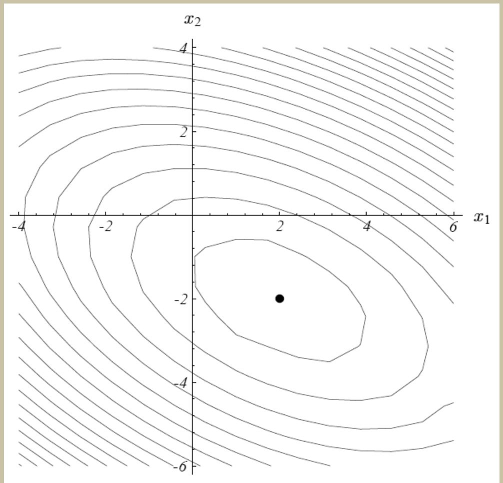


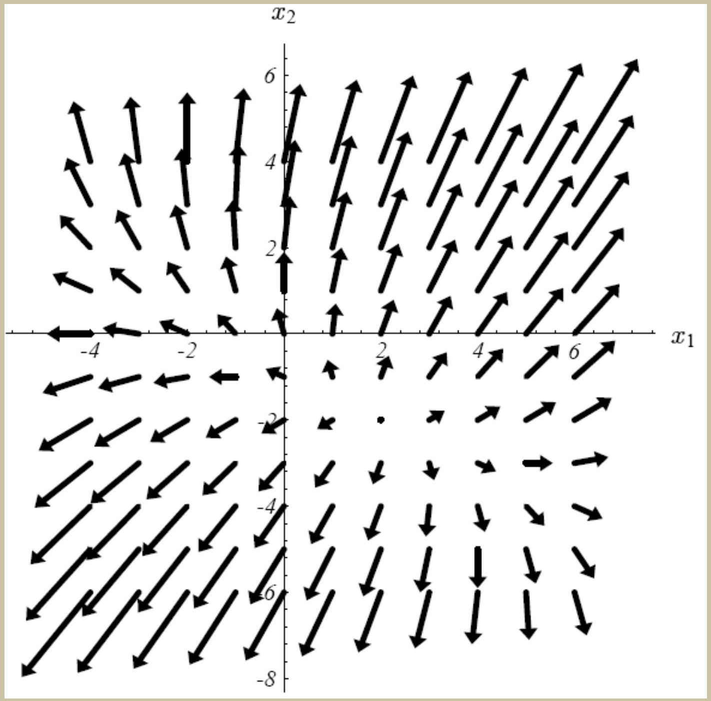


2.1: The Gradient of *f(x)*:

$$f'(x) = \nabla f(x) = \begin{bmatrix} \dfrac{\partial f}{\partial x_1} & \dfrac{\partial f}{\partial x_2} & \cdots & \dfrac{\partial f}{\partial x_n} \end{bmatrix}^T$$







For every x, the gradient points in the direction of <u>steepest increase</u> of *f(x)*, and is orthogonal to the contour lines

Instead of solving Ax=b...

- Inner product (dot, scalar) of two vectors, *x*, *y*:

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

$$f'(x) = \frac{1}{2} A^T x + \frac{1}{2} A x - b = Ax - b$$

A is symmetric:
$A^T = A$

Setting *f'(x) = 0* gives the *minimization* problem for *f(x).* Hence, *Ax* = *b* can be solved by finding *x* that minimizes *f(x).*

Method of Steepest Descent

- Start with an arbitrary point: $x_{(0)}$.
- Find residual vector: $r_{(i)} = b - Ax_{(i)}$

  This indicates how far we are from the correct value of b.

  Note that $r_{(i)} = -f'(x_{(i)})$

  Also, if $e_{(i)} = x_{(i)} - x$ is the (error) vector indicating how far we are from the solution, then $r_{(i)} = -Ae_{(i)}$
- Determine the direction for the next step: move in the direction in which *f*(x) decreases most quickly, i.e. opposite $f'(x)$, that is, $r_{(i)}$.
- How big a step should be taken? $x_{(1)} = x_{(0)} + \alpha r_{(0)}$
- Determine *α* by the condition that it should minimize *f:*

$$\frac{d}{d\alpha} f(x_{(1)}) = f'(x_{(1)})^T \frac{d}{d\alpha} x_{(1)} = f'(x_{(1)})^T r_{(0)} = 0$$

Method of Steepest Descent

- Note that f '$(x_{(1)}) = -r_{(1)}$

$$\begin{aligned}
r_{(1)}^{\;T} r_{(0)} &= 0 \\
(b - Ax_{(1)})^T r_{(0)} &= 0 \\
(b - A(x_{(0)} + \alpha r_{(0)}))^T r_{(0)} &= 0 \\
(b - Ax_{(0)})^T r_{(0)} - \alpha (Ar_{(0)})^T r_{(0)} &= 0 \\
(b - Ax_{(0)})^T r_{(0)} &= \alpha (Ar_{(0)})^T r_{(0)} \\
r_{(0)}^T r_{(0)} &= \alpha r_{(0)}^T (Ar_{(0)}) \\
\alpha &= \frac{r_{(0)}^T r_{(0)}}{r_{(0)}^T A r_{(0)}}
\end{aligned}$$

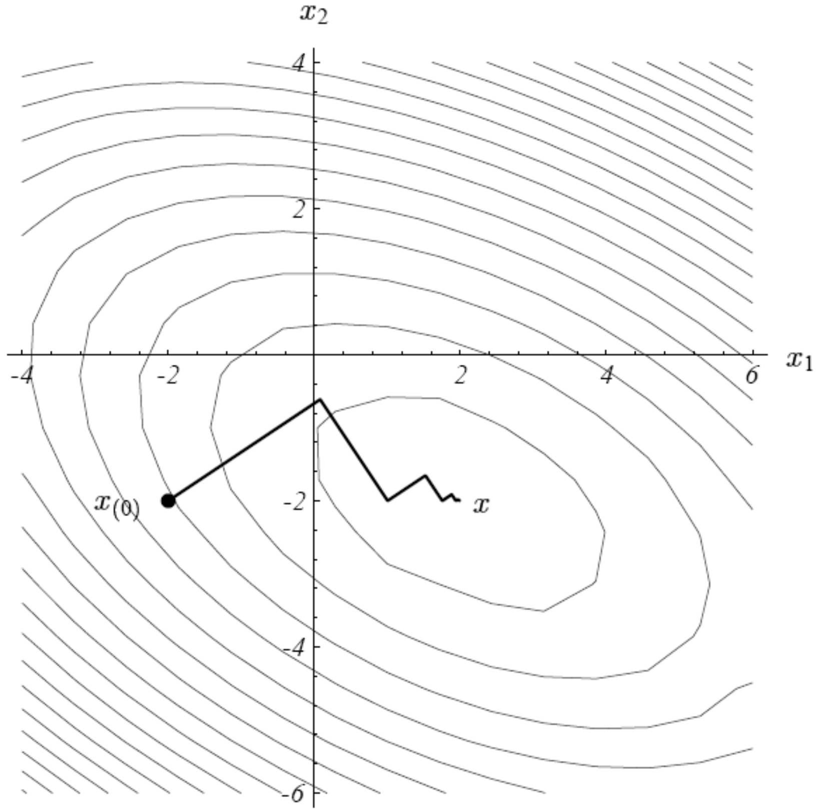
Method of Steepest Descent

- Start with an arbitrary point

$$r_{(i)} = b - Ax_{(i)}$$

$$\alpha_{(i)} = \frac{r_{(i)}^T r_{(i)}}{r_{(i)}^T A r_{(i)}}$$

$$x_{(i+1)} = x_{(i)} + \alpha_{(i)} r_{(i)}$$



Premultiplying last equation by *–A* and adding *b* gives:

$$r_{(i+1)} = r_{(i)} - \alpha_{(i)} A r_{(i)}$$

Use this for *i* > 0. **CAUTION:** Since the feedback from $x_{(i)}$ is not present here, use the form above periodically to prevent misconvergence

Method of Conjugate Gradient

- Method of Steepest Descent was constructing steps with successive residual vectors being orthogonal:

$$r_{(1)}^{T} r_{(0)} = 0$$

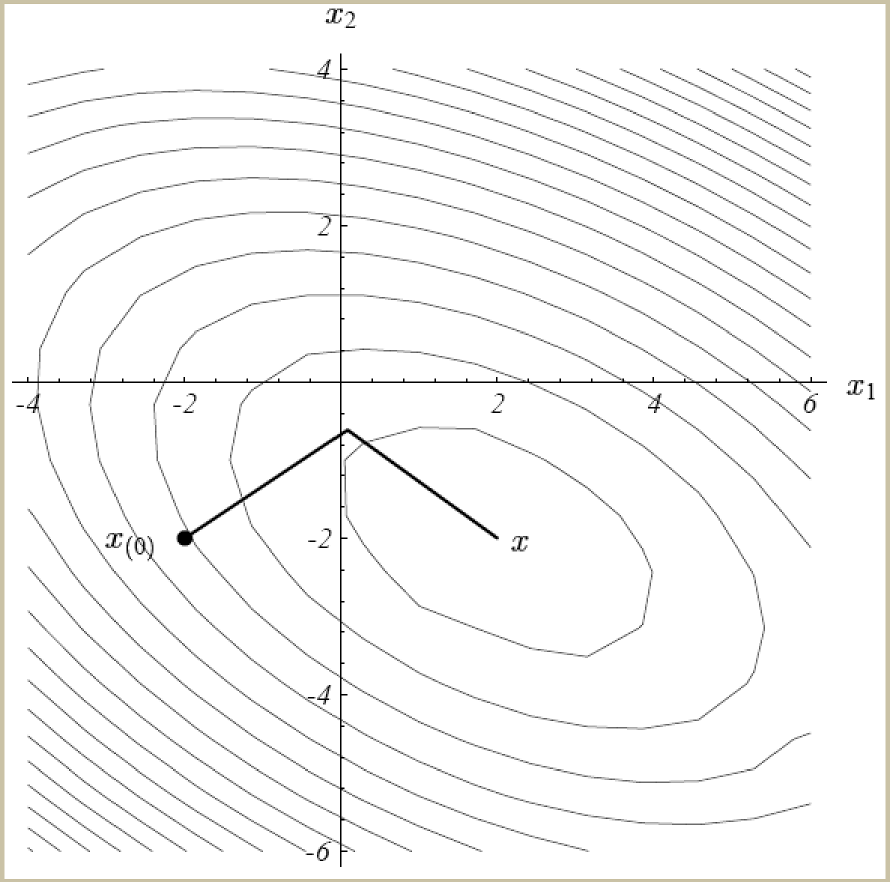
- Conjugate gradient method employs vectors that are *A*-orthogonal (or conjugate)

$$d_{(i)}^{T} A d_{(j)} = 0$$

- Details of the derivation of the method are omitted

Method of Conjugate Gradients

$$
\begin{aligned}
d_{(0)} &= b - Ax_{(0)} \\
\alpha_{(i)} &= \frac{r_{(i)}^T r_{(i)}}{d_{(i)}^T A d_{(i)}} \\
x_{(i+1)} &= x_{(i)} + \alpha_{(i)} d_{(i)} \\
r_{(i+1)} &= r_{(i)} - \alpha_{(i)} A d_{(i)} \\
\beta_{(i+1)} &= \frac{r_{(i+1)}^T r_{(i+1)}}{r_{(i)}^T r_{(i)}} \\
r_{(i+1)} &= r_{(i+1)} + \beta_{(i+1)} d_{(i)}
\end{aligned}
$$



Preconditioned Conjugate Gradient Method

- If the matrix *A* is ill conditioned, the CG method may suffer from numerical errors (rounding, overflow, underflow).

$$\begin{bmatrix} 2 & 1 \\ 2 & 1.001 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 3 \\ 0 \end{bmatrix} \qquad x = 1501.5,\ y = -3000$$

$$\begin{bmatrix} 2 & 1 \\ 2 & 1.002 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 3 \\ 0 \end{bmatrix} \qquad x = 751.5,\ y = -1500$$

- Matrix condition number: $\mathrm{cond}(A) = \|A\| \cdot \|A^{-1}\| \begin{cases} >> 1, \text{ ill conditioned} \\ \approx 1, \text{ well conditioned} \end{cases}$

- Matrix norm: $\|A\| = \max_{1 \le i \le n} \sum_{j=1}^{n} |A_{ij}|$

For this example cond*(A)* = 5001 >>1

Preconditioned Conjugate Gradient Method

- Suppose that *M* is a symmetric positive definite matrix that approximates *A*, but easier to invert (well conditioned). Then we can solve instead: $M^{-1} Ax = M^{-1}x$

$$\begin{aligned}
r_{(0)} &= b - Ax_{(0)} \\
d_{(0)} &= M^{-1} r_{(0)} \\
\alpha_{(i)} &= \frac{r_{(i)}^T M^{-1} r_{(i)}}{d_{(i)}^T A d_{(i)}} \\
x_{(i+1)} &= x_{(i)} - \alpha_{(i)} d_{(i)} \\
r_{(i+1)} &= r_{(i)} - \alpha_{(i)} A d_{(i)} \\
\beta_{(i+1)} &= \frac{r_{(i+1)}^T M^{-1} r_{(i+1)}}{r_{(i)}^T M^{-1} r_{(i)}} \\
d_{(i+1)} &= M^{-1} r_{(i+1)} + \beta_{(i+1)} d_{(i)}
\end{aligned}$$

Preconditioned Conjugate Gradient Method

- Jacobi preconditioner:

$$M_{ij} = \begin{cases} A_{ii} & i = j \\ 0 & i \neq j \end{cases}$$

- Symmetric successive overrelaxation preconditioner:

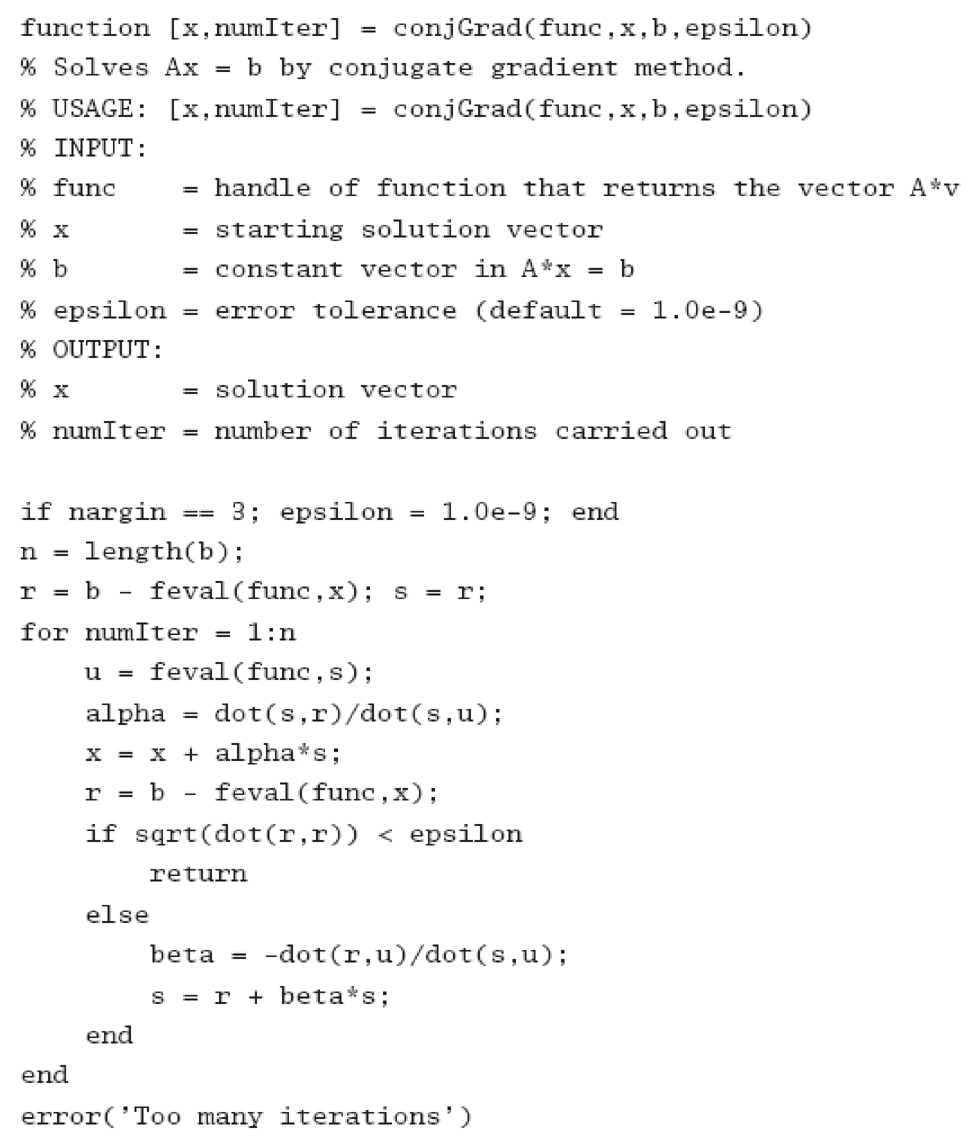
$$A = L + D + L^T$$

where *L* is the strictly lower part of A and D is diagonal of A.

$$M = (\frac{D}{\omega} + L) \frac{\omega}{2 - \omega} D^{-1} (\frac{D}{\omega} + L^T)$$

ω in the interval]0,2[is the relaxation parameter to be chosen.

CG Method: sample code for Matlab



```
function [x,numIter] = conjGrad(func,x,b,epsilon)
% Solves Ax = b by conjugate gradient method.
% USAGE: [x,numIter] = conjGrad(func,x,b,epsilon)
% INPUT:
% func    = handle of function that returns the vector A*v
% x       = starting solution vector
% b       = constant vector in A*x = b
% epsilon = error tolerance (default = 1.0e-9)
% OUTPUT:
% x       = solution vector
% numIter = number of iterations carried out

if nargin == 3; epsilon = 1.0e-9; end
n = length(b);
r = b - feval(func,x); s = r;
for numIter = 1:n
    u = feval(func,s);
    alpha = dot(s,r)/dot(s,u);
    x = x + alpha*s;
    r = b - feval(func,x);
    if sqrt(dot(r,r)) < epsilon
        return
    else
        beta = -dot(r,u)/dot(s,u);
        s = r + beta*s;
    end
end
error('Too many iterations')
```

CG Method: sample problem

- Sample problem:

$$\begin{bmatrix} 4 & -1 & 1 \\ -1 & 4 & -2 \\ 1 & -2 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 12 \\ -1 \\ 5 \end{bmatrix}$$

- exact solution: $x_1 = 3$, $x_2 = x_3 = 1$.