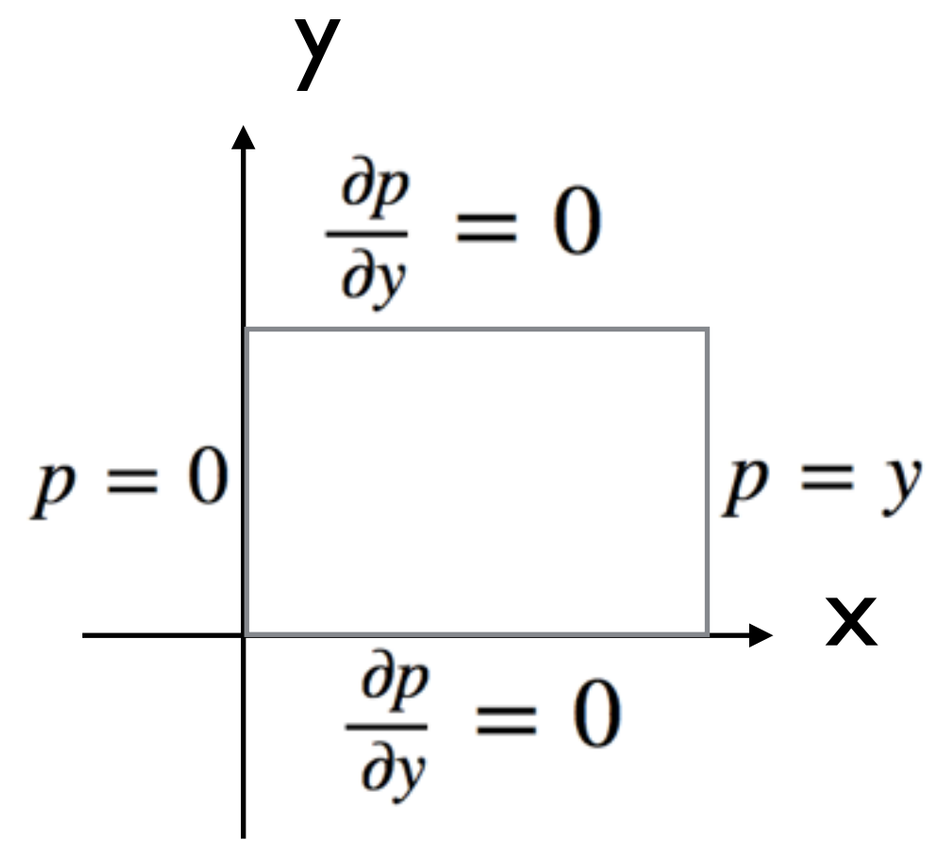
05/09	Class 9	Dense direct solvers	Understand the principle of LU decomposition and the optimization and parallelization techniques that lead to the LINPACK benchmark.
05/12	Class 10	Dense eigensolvers	Determine eigenvalues and eigenvectors and understand the fast algorithms for diagonalization and orthonormalization.
05/16	Class 11	Sparse direct solvers	Understand reordering in AMD and nested dissection, and fast algorithms such as skyline and multifrontal methods.
05/19	Class 12	Sparse iterative solvers	Understand the notion of positive definiteness, condition number, and the difference between Jacobi, CG, and GMRES.
05/23	Class 13	Preconditioners	Understand how preconditioning affects the condition number and spectral radius, and how that affects the CG method.
05/26	Class 14	Multigrid methods	Understand the role of smoothers, restriction, and prolongation in the V-cycle.
05/30	Class 15	Fast multipole methods, H-matrices	Understand the concept of multipole expansion and low-rank approximation, and the role of the tree structure.

2-D Laplace equation

$$\frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2} = 0$$

$$p_{i,j}^n = \frac{\Delta y^2 (p_{i+1,j}^n + p_{i-1,j}^n) + \Delta x^2 (p_{i,j+1}^n + p_{i,j-1}^n)}{2(\Delta x^2 + \Delta y^2)}$$



$$A = \begin{bmatrix} 4 & -1 & & -1 & & & & & \\ -1 & 4 & -1 & & -1 & & & & \\ & -1 & 4 & & & -1 & & & \\ -1 & & & 4 & -1 & & -1 & & \\ & -1 & & -1 & 4 & -1 & & -1 & \\ & & -1 & & -1 & 4 & & & -1 \\ & & & -1 & & & 4 & -1 & \\ & & & & -1 & & -1 & 4 & -1 \\ & & & & & -1 & & -1 & 4 \end{bmatrix}$$

Ax=b

Sparse iterative solvers

SciPy.org Sponsored By ENTHOUGHT

Scipy.org | Docs | SciPy v0.17.1 Reference Guide

Sparse linear algebra (scipy.sparse.linalg)

Solving linear problems

Direct methods for linear equation systems:

spsolve(A, b[, permc_spec, use_umfpack])	Solve the sparse linear system Ax=b, where b may be a vector or a matrix.
factorized(A)	Return a fuction for solving a sparse linear system, with A pre-factorized.
MatrixRankWarning	
use_solver(**kwargs)	Select default sparse direct solver to be used.

Iterative methods for linear equation systems:

bicg(A, b[, x0, tol, maxiter, xtype, M, ...])	Use BIConjugate Gradient iteration to solve A x = b
bicgstab(A, b[, x0, tol, maxiter, xtype, M, ...])	Use BIConjugate Gradient STABilized iteration to solve A x = b
cg(A, b[, x0, tol, maxiter, xtype, M, callback])	Use Conjugate Gradient iteration to solve A x = b
cgs(A, b[, x0, tol, maxiter, xtype, M, callback])	Use Conjugate Gradient Squared iteration to solve A x = b
gmres(A, b[, x0, tol, restart, maxiter, ...])	Use Generalized Minimal RESidual iteration to solve A x = b.
lgmres(A, b[, x0, tol, maxiter, M, ...])	Solve a matrix equation using the LGMRES algorithm.
minres(A, b[, x0, shift, tol, maxiter, ...])	Use MINimum RESidual iteration to solve Ax=b
qmr(A, b[, x0, tol, maxiter, xtype, M1, M2, ...])	Use Quasi-Minimal Residual iteration to solve A x = b

Solvers

$$Ax = b$$

Dense matrix

- Gauss Elimination
- LU decomposition

Sparse matrix

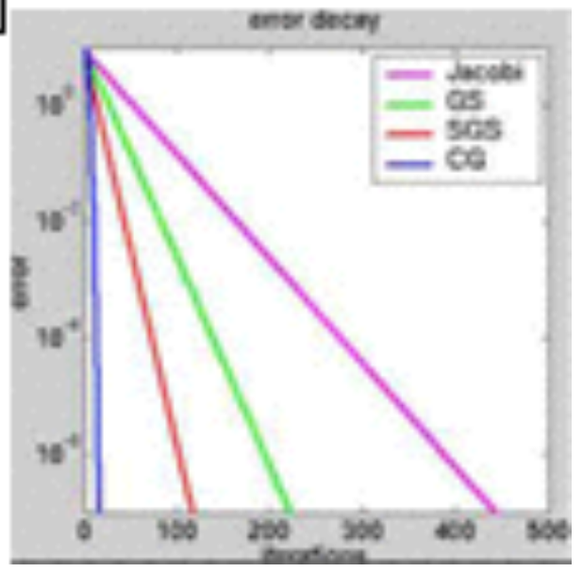
Iterative solver

Stationary method

- Jacobi
- Gauss-Seidel
- SOR

Krylov subspace method

- CG
- BiCG
- BiCGSTAB
- GMRES
- MINRES



Direct solver

- Multifrontal
- Supernodal

Why iterative methods?

For many problems, it is not clear which method is best:

- Direct methods are robust
- $O(n^p)$ can have a very large constant for iterative methods
- Often a combination can be used, e.g. in domain decomposition or iterative refinement

Model problem	Direct	Iterative
Computational costs	$O(n^p)$ $p \approx 2.0$ for 2D $p \approx 2.3$ for 3D	$O(n^p)$ $p \approx 1.4$ for 2D $p \approx 1.2$ for 3D
Memory requirements	$O(n^p)$ $p \approx 1.5$ for 2D $p \approx 1.7$ for 3D	$O(n)$

Why iterative methods?

- Direct solvers are great for dense matrices and can be made to avoid roundoff errors to a large degree. They can also be implemented very well on modern machines.
- **Fill-in** is a major problem for certain sparse matrices and leads to extreme memory requirements (e.g., three-d.
- Some matrices appearing in practice are **too large** to even be represented explicitly (e.g., the Google matrix).
- Often linear systems only need to be **solved approximately**, for example, the linear system itself may be a linear approximation to a nonlinear problem.
- Direct solvers are much harder to implement and use on (massively) **parallel computers**.

Stationary Methods

Jacobi

$$x_i^{(k+1)} = x_i^k - \frac{1}{a_{ii}} \left(\sum_{j=1, j \neq i}^{n} a_{ij}\, x_j^{(k)} - b_i \right) \qquad i = 1, \ldots, n$$

Gauss-Seidel

$$x_i^{(k+1)} = x_i^k - \frac{1}{a_{ii}} \left(\sum_{j=1}^{i-1} a_{ij}\, x_j^{(k+1)} + \sum_{j=i+1}^{n} a_{ij}\, x_j^{(k)} - b_i \right) \qquad i = 1, \ldots, n$$

SOR

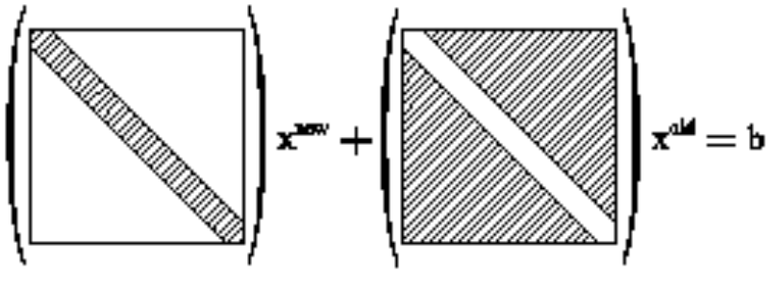
$$x_i^{(k+1)} = x_i^k - \frac{1}{a_{ii}} \omega \left(\sum_{j=1}^{i-1} a_{ij}\, x_j^{(k+1)} + \sum_{j=i+1}^{n} a_{ij}\, x_j^{(k)} - b_i \right) \qquad i = 1, \ldots, n$$

Stationary methods

$$\mathbf{A} = \mathbf{D} - \mathbf{L} - \mathbf{U}$$

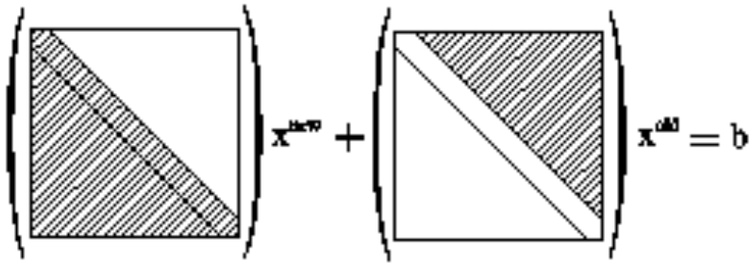
Jacobi Iteration

$$\mathbf{x}^{(k+1)} = \mathbf{D}^{-1}(\mathbf{L} + \mathbf{U})\mathbf{x}^{(k)} + \mathbf{D}^{-1}\mathbf{b}$$



Gauss-Seidel

$$\mathbf{x}^{(k+1)} = (\mathbf{I} - \mathbf{D}^{-1}\mathbf{L})^{-1}\mathbf{D}^{-1}\mathbf{U}\mathbf{x}^{(k)} + (\mathbf{I} - \mathbf{D}^{-1}\mathbf{L})^{-1}\mathbf{D}^{-1}\mathbf{b}$$



Successive Over-Relaxation

$$\mathbf{x}^{(k+1)} = (\mathbf{D} - \omega\mathbf{L})^{-1}((1-\omega)\mathbf{D} + \omega\mathbf{U})\mathbf{x}^{(k)} + (\mathbf{D} - \omega\mathbf{L})^{-1}\omega\mathbf{b}$$

Basic Iterative Methods for Linear Systems

Consider the system of equations

$$Ax = b.$$

Let us split A into

$$A = M - K$$

where M is any non-singular matrix and $K = M - A$.
Hence, $Ax = b$ becomes

$$\begin{aligned}(M - K)x &= b \\ Mx &= Kx + b \\ x &= M^{-1}Kx + M^{-1}b\end{aligned}$$

Basic Iterative Methods for Linear Systems, cont'd

This suggests the iteration scheme: For $k = 1, 2, 3, \ldots$ repeat

$$x^{(k+1)} = M^{-1}Kx^{(k)} + M^{-1}b$$

until convergence.

Of course, for this iteration to be computationally practical, the splitting of A should be chosen such that $M^{-1}K$ and $M^{-1}b$ are easy to calculate.

We will study splittings based on the diagonal, and the upper / lower triangular parts of A:

$$A = D - U - L$$

Jacobi's Method

So-called Jacobi iteration is defined by choosing the splitting

$$A = D - (L + U) = M - K$$

where D is the diagonal of A, $-L$ is the strictly lower triangle of A, and $-U$ is the strictly upper triangle of A.

The iteration scheme takes the form

$$x^{(k+1)} = D^{-1}(L + U)x^{(k)} + D^{-1}b$$

Note that D is easy to invert since it is a diagonal matrix.

The Gauss-Seidel Method

In the (forward) Gauss-Seidel method A is split into

$$A = (D - L) - U = M - K$$

yielding the iteration scheme

$$x^{(k+1)} = (D - L)^{-1}(Ux^{(k)} + b)$$

Since $D - L$ is lower triangular the effect of $(D - L)^{-1}$ can be computed by forward elimination.

The (backward) Gauss-Seidel method instead uses

$$A = (D - U) - L = M - K$$

Successive Over-Relaxation (SOR)

A more sophisticated method is obtained by choosing

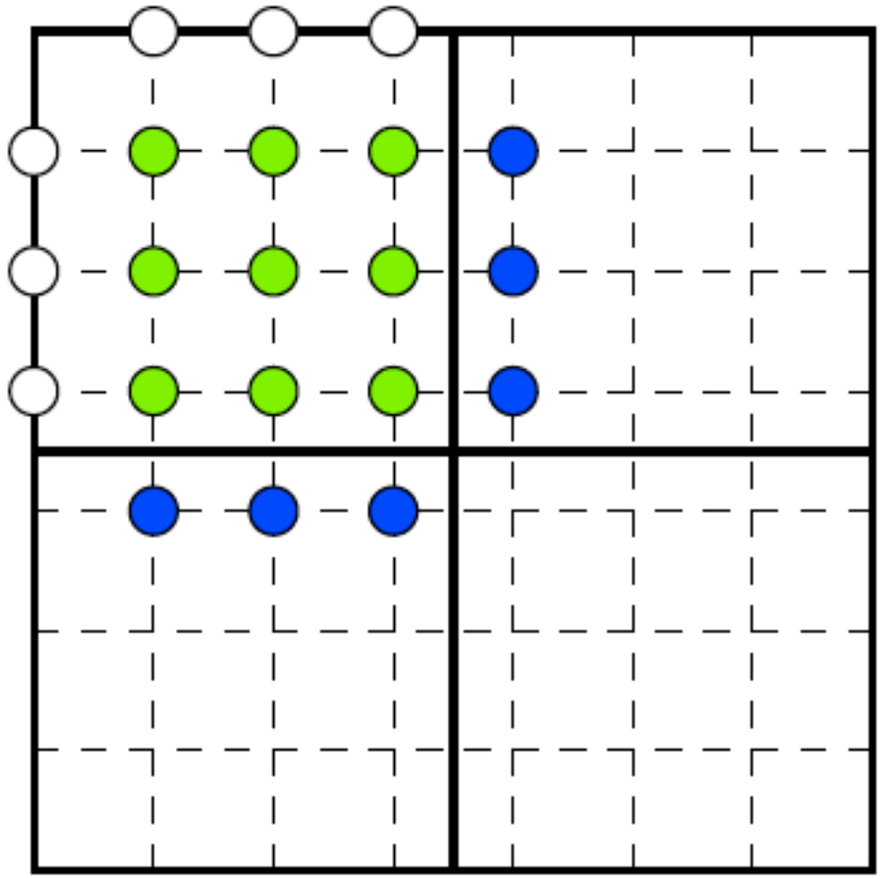
$$A = (\frac{1}{\omega}D - L) - (\frac{1-\omega}{\omega}D + U) = M - K$$

where ω is a relaxation parameter.

This gives the iteration scheme

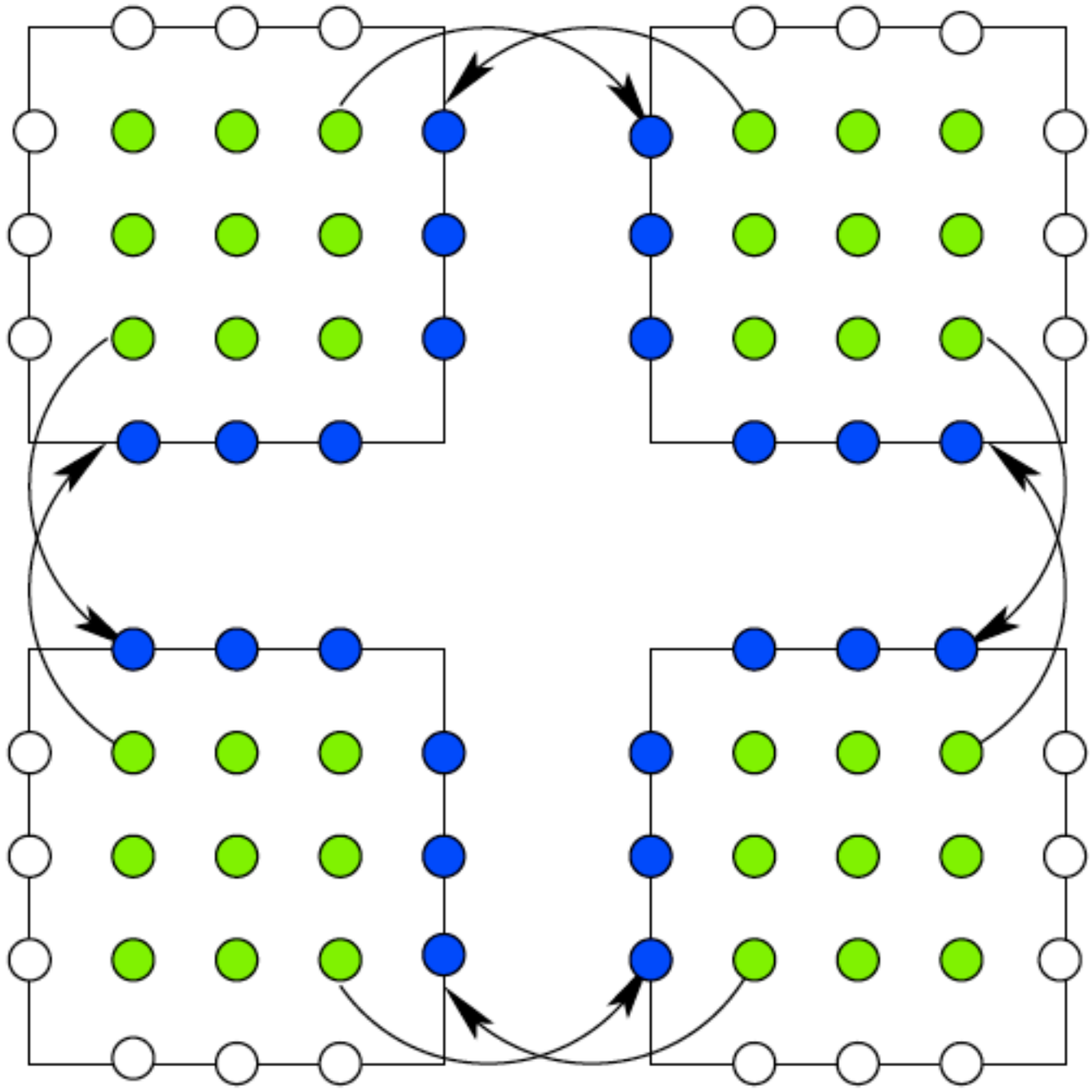
$$x^{(k+1)} = (D - \omega L)^{-1}(\omega U + (1-\omega)D)x^{(k)} + \omega(D - \omega L)^{-1}b$$

Parallel version (5 point stencil)



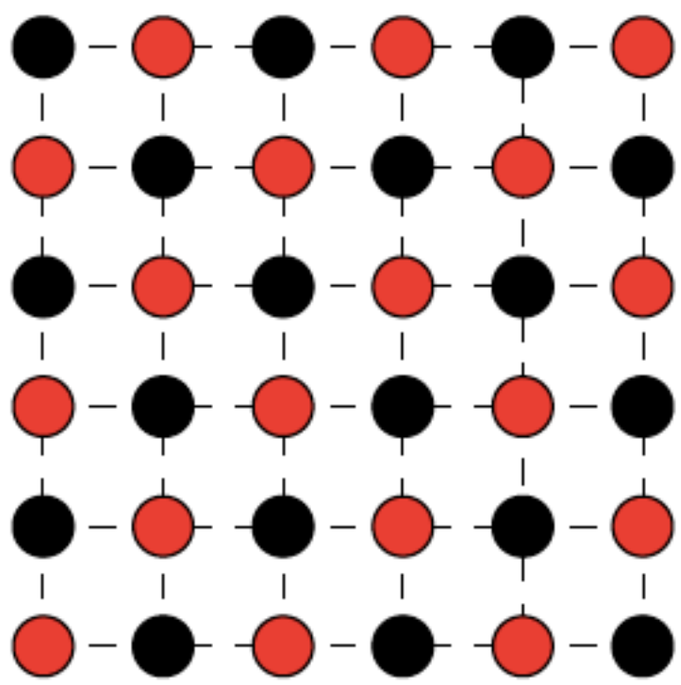
Boundary values:	white
Data on P0:	green
Ghost cell data:	blue

Parallel version (5 point stencil)



Communicate ghost cells before each step.

Red-Black Gauss-Seidel



Red depends only on black, and vice-versa.
Generalization: multi-color orderings

Red black Gauss-Seidel step

```
for i = 2:n-1
  for j = 2:n-1
    if mod(i+j,2) == 0
      u(i,j) = ...
    end
  end
end

for i = 2:n-1
  for j = 2:n-1
    if mod(i+j,2) == 1,
    u(i,j) = ...
  end
end
```

Solvers

$$Ax = b$$

Dense matrix

- Gauss Elimination
- LU decomposition

Sparse matrix

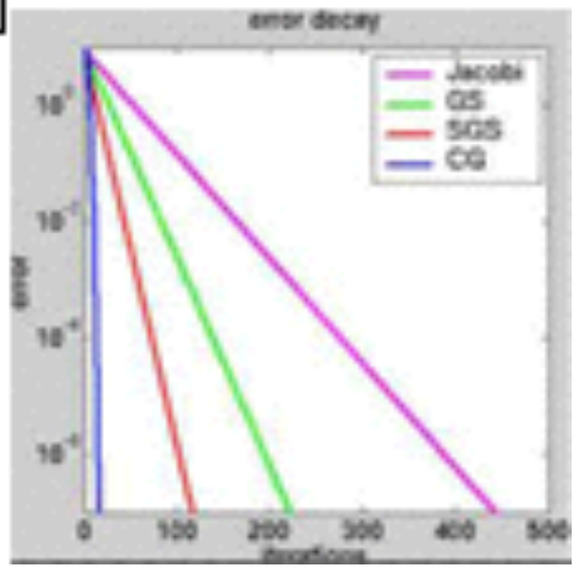
Iterative solver

Stationary method

- Jacobi
- Gauss-Seidel
- SOR

Krylov subspace method

- CG
- BiCG
- BiCGSTAB
- GMRES
- MINRES



Direct solver

- Multifrontal
- Supernodal

Alexei Nikolaevich Krylov

1863-1945

Maritime Engineer

300 papers and books:
shipbuilding, magnetism,
artillery, math, astronomy

1890: Theory of oscillating
motions of the ship

1931: *Krylov subspace methods*

Conjugate gradient method as iterative method

in exact arithmetic

- CG was originally proposed as a direct (non-iterative) method
- in theory, convergence in at most n steps

in practice

- due to rounding errors, CG method can take $\gg n$ steps (or fail)
- CG is now used as an iterative method
- with luck (good spectrum of A), good approximation in $\ll n$ steps
- attractive if matrix-vector products are inexpensive

conjugate gradient

Web Images Videos News Shopping More Search tools

About 770,000 results (0.27 seconds)

In mathematics, the **conjugate gradient** method is an algorithm for the numerical solution of particular systems of linear equations, namely those whose matrix is symmetric and positive-definite.

Conjugate gradient method - Wikipedia, the free encyclopedia
https://en.wikipedia.org/wiki/**Conjugate_gradient**_method

More about Conjugate gradient method

Feedback

Conjugate gradient method - Wikipedia, the free encyclopedia
https://en.wikipedia.org/wiki/**Conjugate_gradient**_method
In mathematics, the **conjugate gradient** method is an algorithm for the numerical solution of particular systems of linear equations, namely those whose matrix is symmetric and positive-definite.
Nonlinear conjugate gradient - Preconditioner - Biconjugate gradient method

[PDF] An Introduction to the Conjugate Gradient Method Without…
www.cs.cmu.edu/~quake-papers/painless-**conjugate-gradient**.pdf
by JR Shewchuk - 1994 - Cited by 1646 - Related articles
The idea of quadratic forms is introduced and used to derive the methods of Steepest Descent, **Conjugate** Directions, and **Conjugate Gradients**. Eigenvectors are explained and used to examine the convergence of the Jacobi Method, Steepest Descent, and **Conjugate Gradients**.

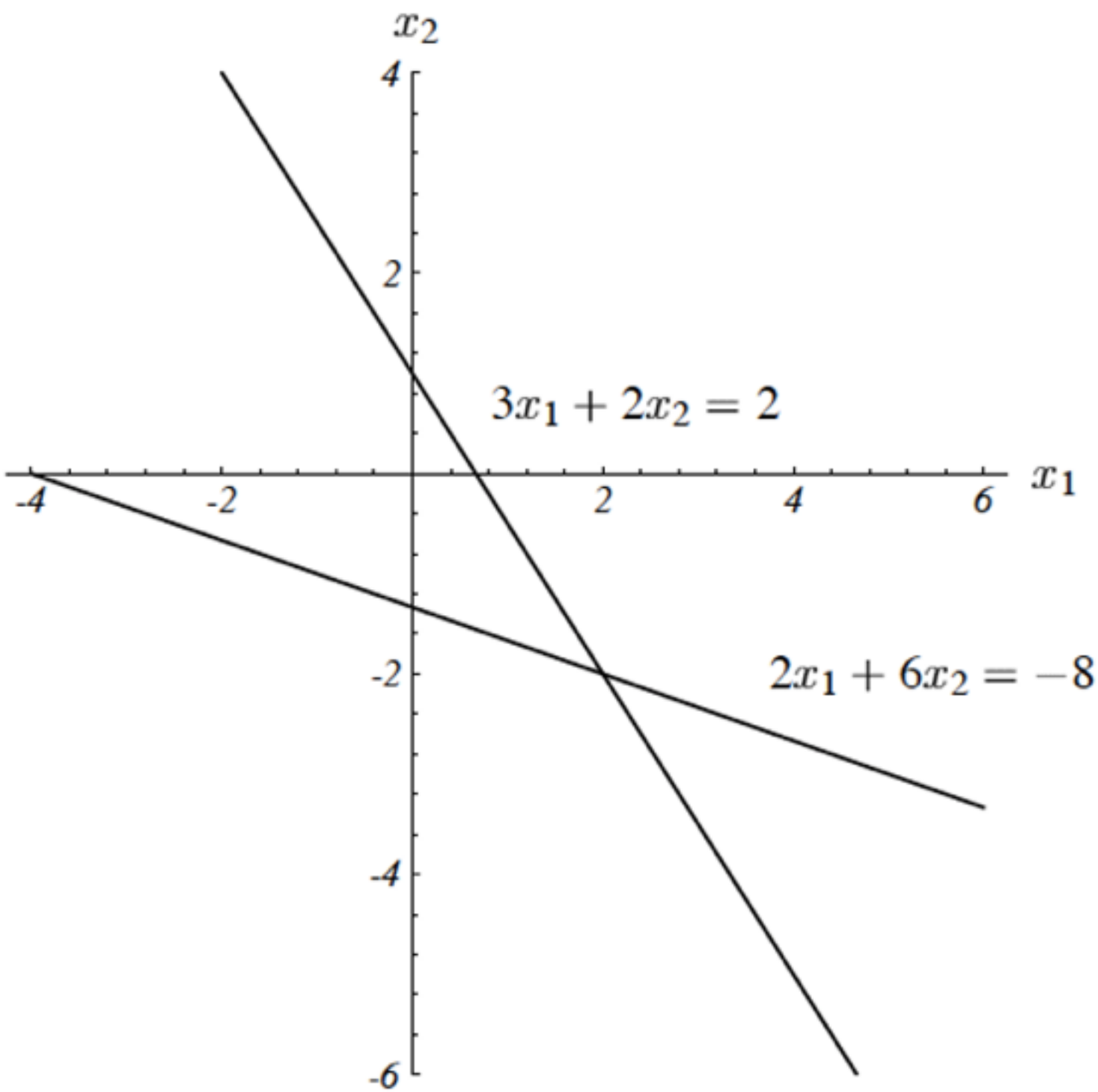
An Introduction to the Conjugate Gradient Method Without the Agonizing Pain

Edition $1\frac{1}{4}$

Jonathan Richard Shewchuk

August 4, 1994

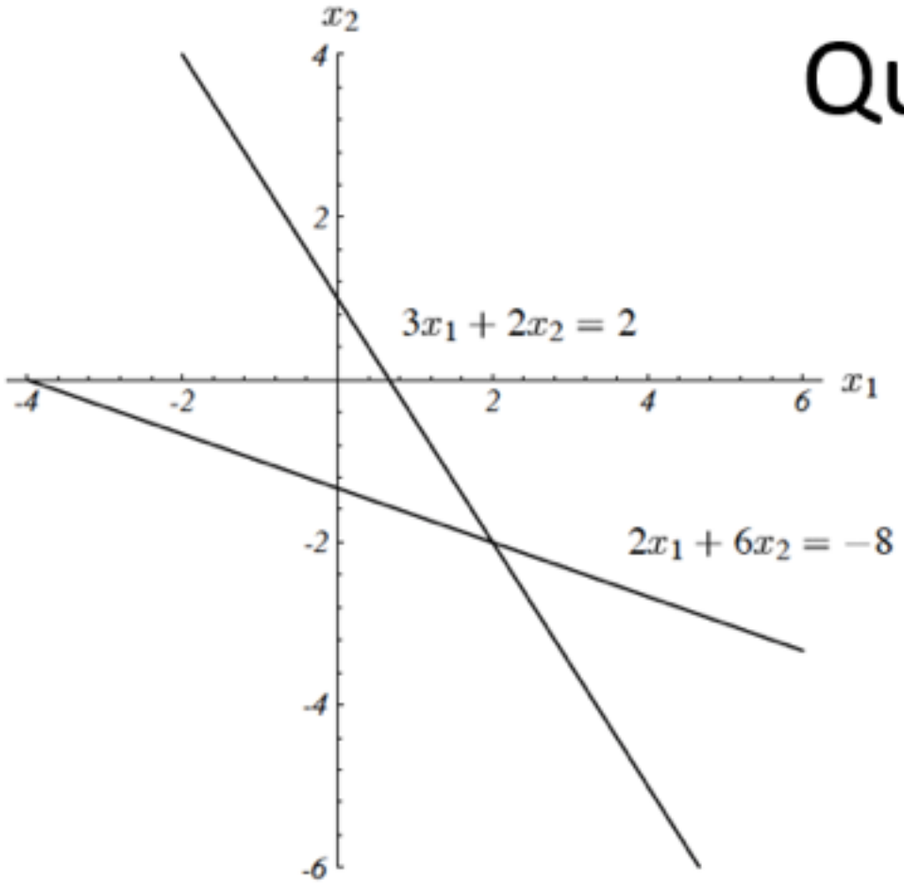
2x2 example



$$Ax = b$$

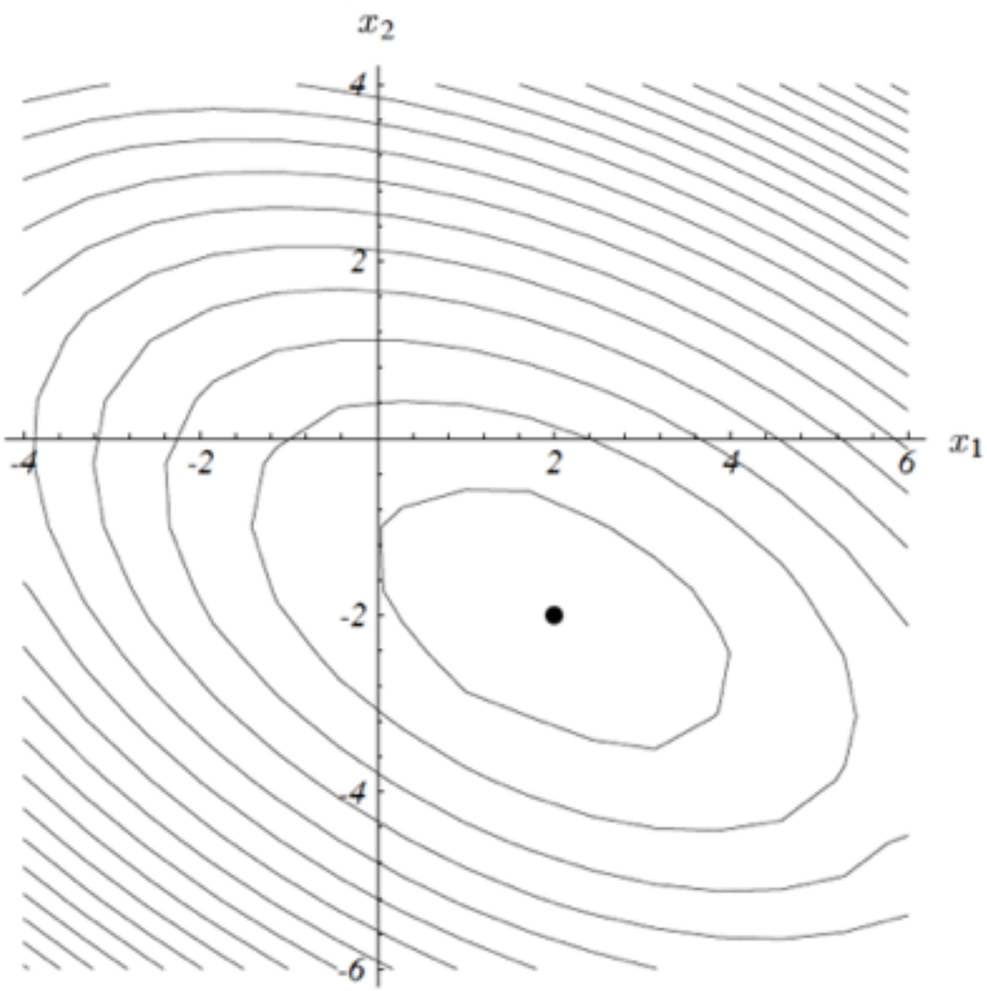
$$A = \begin{bmatrix} 3 & 2 \\ 2 & 6 \end{bmatrix}, \qquad b = \begin{bmatrix} 2 \\ -8 \end{bmatrix}$$

Quadratic function

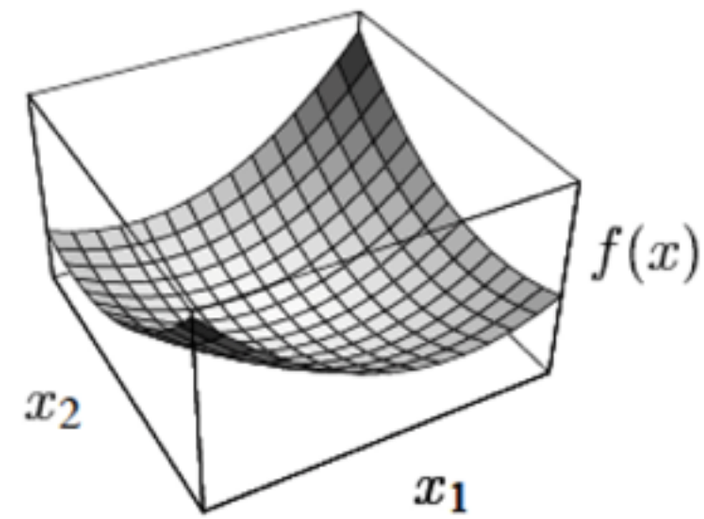


$$f(x) = \frac{1}{2}x^T Ax - b^T x$$

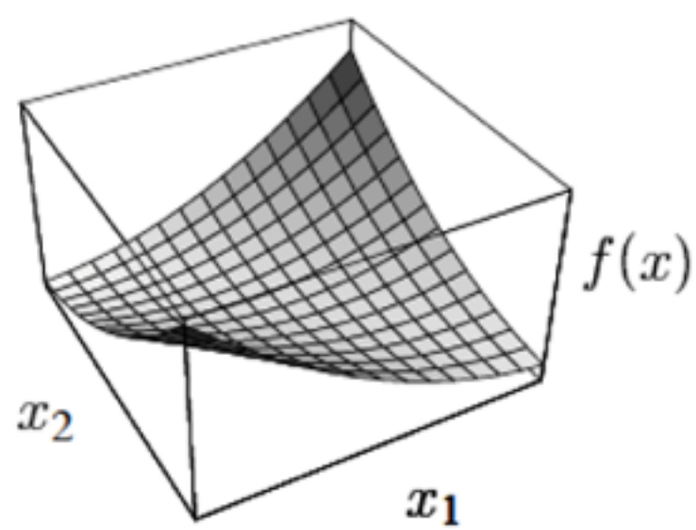
$$f'(x) = Ax - b.$$



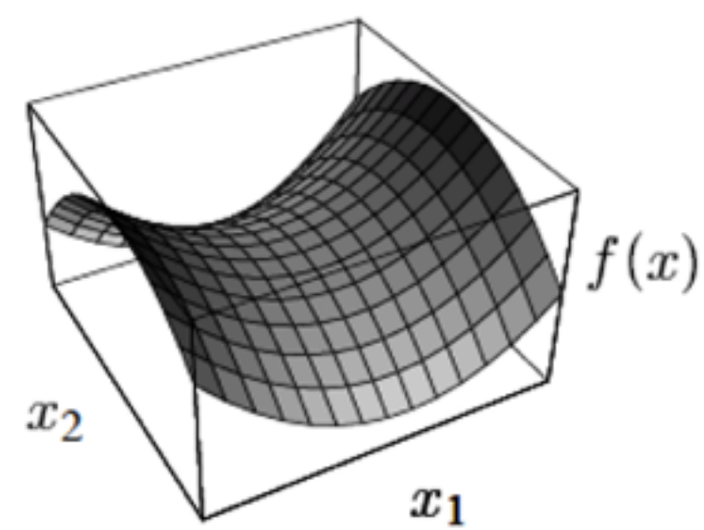
Positive definite $x^T Ax > 0$



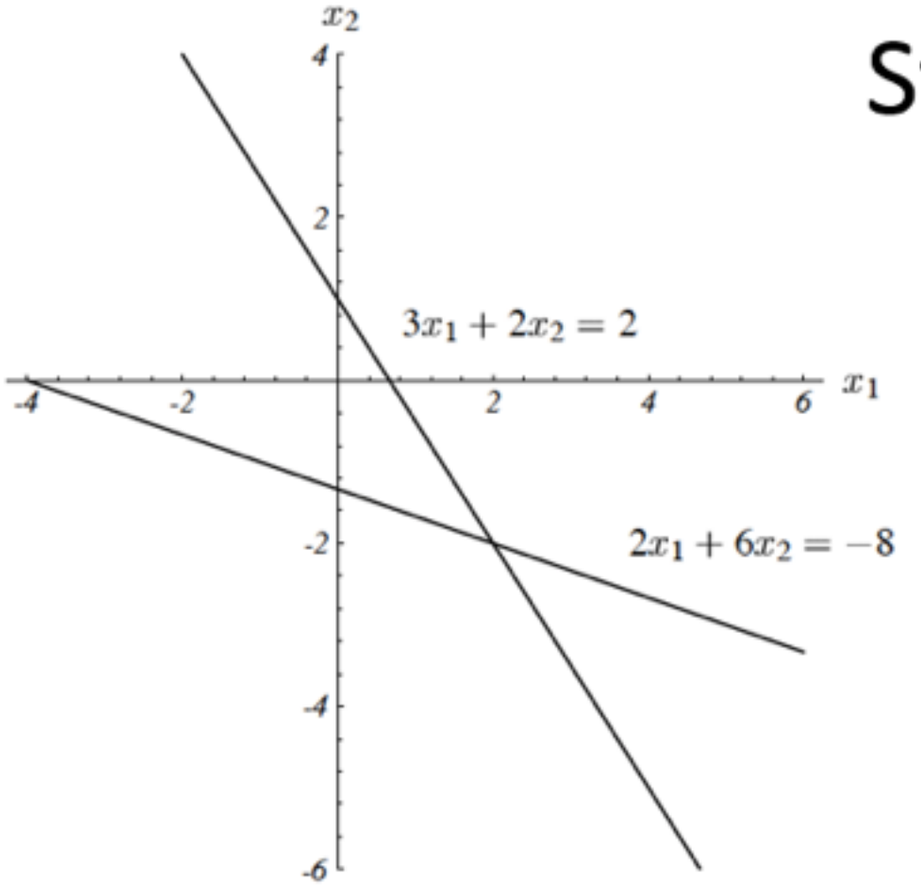
Singular



Indefinite

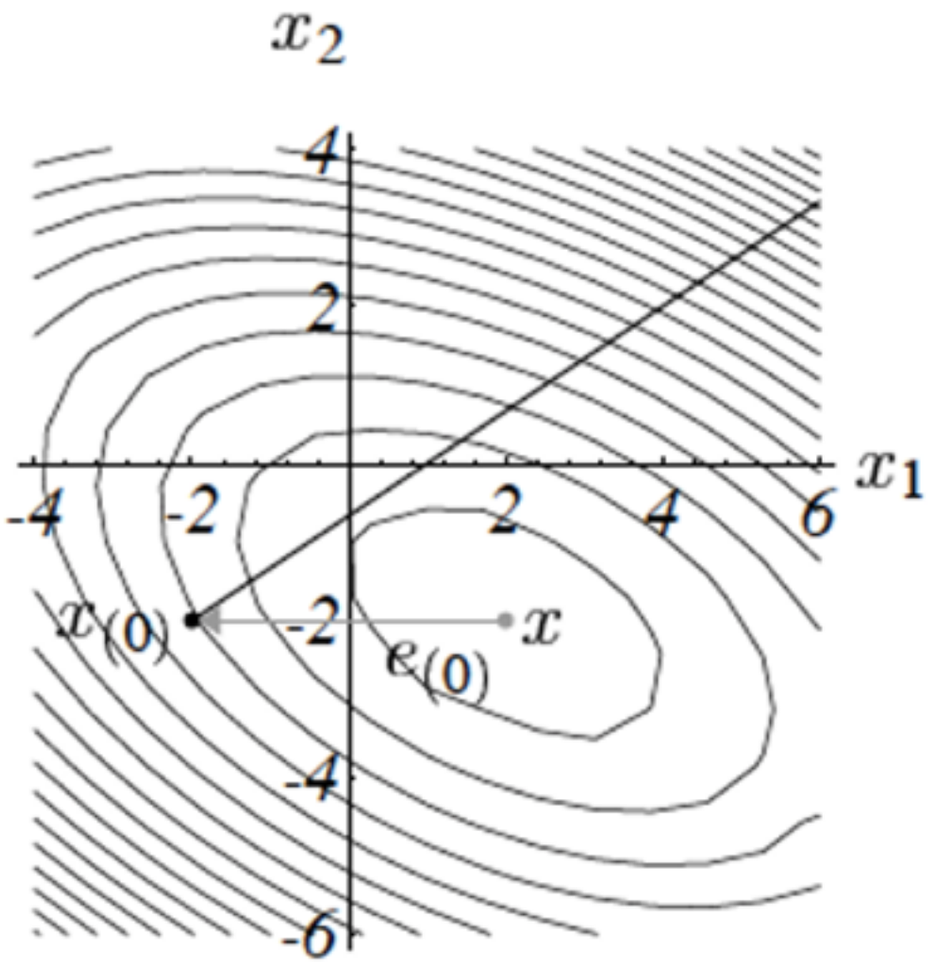


Steepest decent



$$e_{(i)} = x_{(i)} - x$$

$$r_{(i)} = b - Ax_{(i)} = -Ae_{(i)} = -f'(x_{(i)})$$



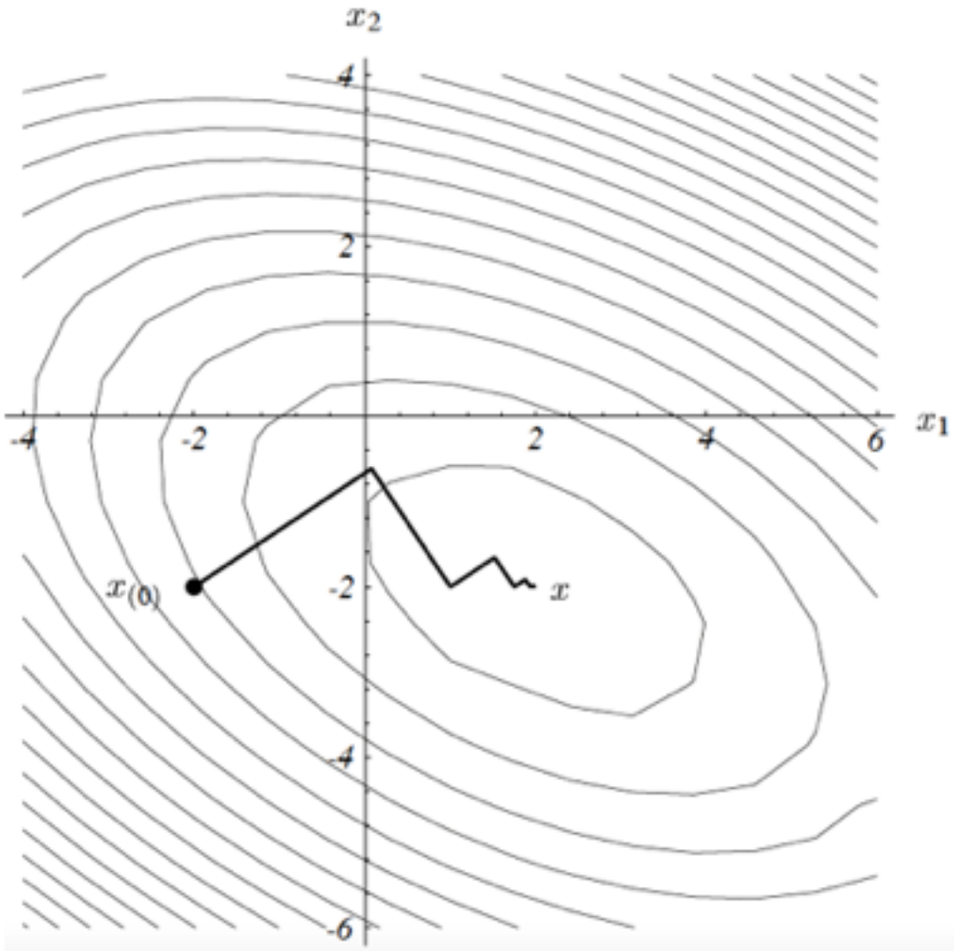
Suppose we start at $x_{(0)} = [-2, -2]^T$

We take a series of steps $x_{(1)}, x_{(2)}, \ldots$

$$x_{(1)} = x_{(0)} + \alpha r_{(0)}$$

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

Steepest decent



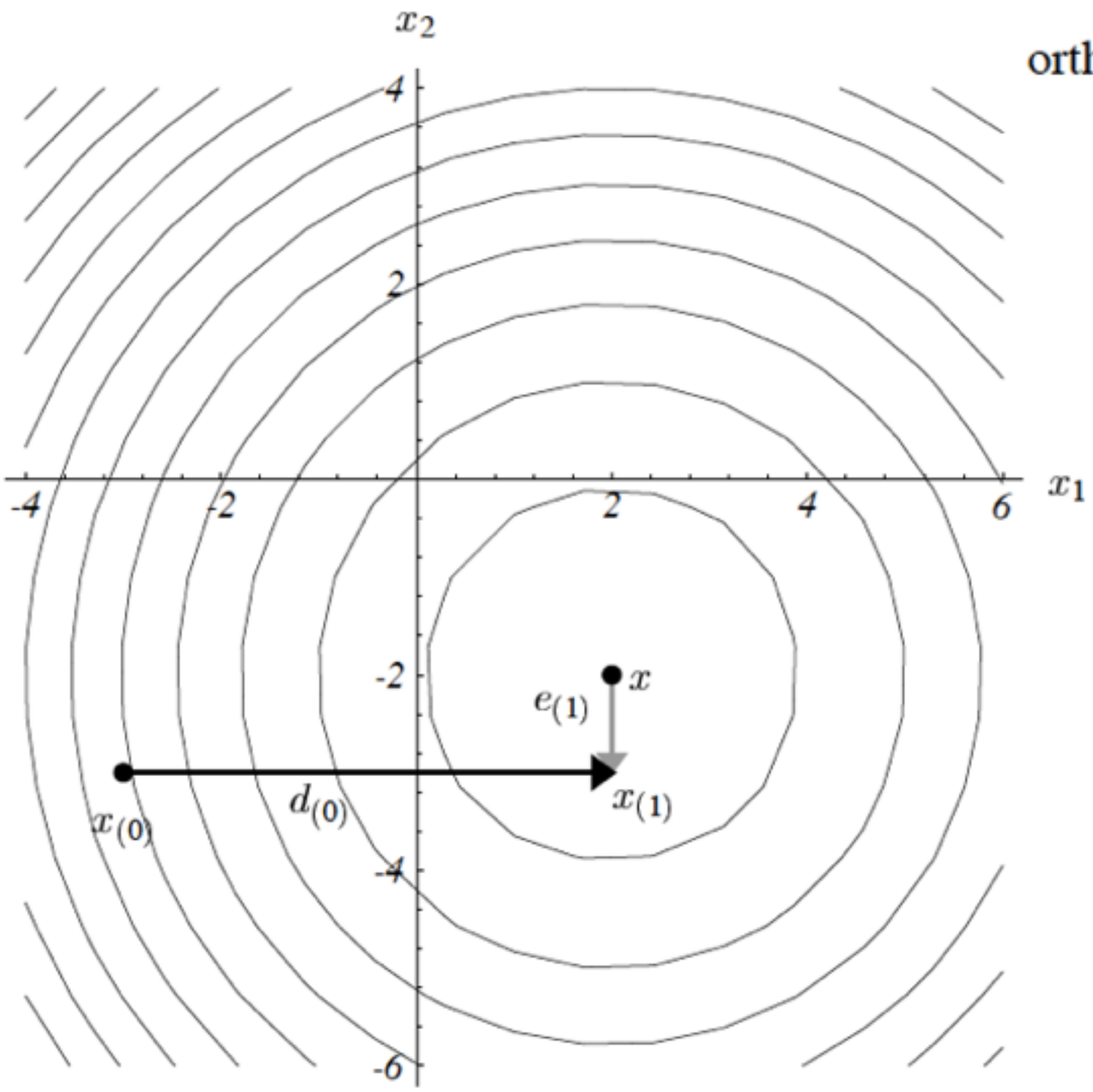
$$
\begin{aligned}
r_{(1)}^T r_{(0)} &= 0 \\
(b - Ax_{(1)})^T r_{(0)} &= 0 \\
(b - A(x_{(0)} + \alpha r_{(0)}))^T r_{(0)} &= 0 \\
(b - Ax_{(0)})^T r_{(0)} - \alpha (Ar_{(0)})^T r_{(0)} &= 0 \\
(b - Ax_{(0)})^T r_{(0)} &= \alpha (Ar_{(0)})^T r_{(0)} \\
r_{(0)}^T r_{(0)} &= \alpha r_{(0)}^T (Ar_{(0)})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

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

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

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

Taking fewer steps



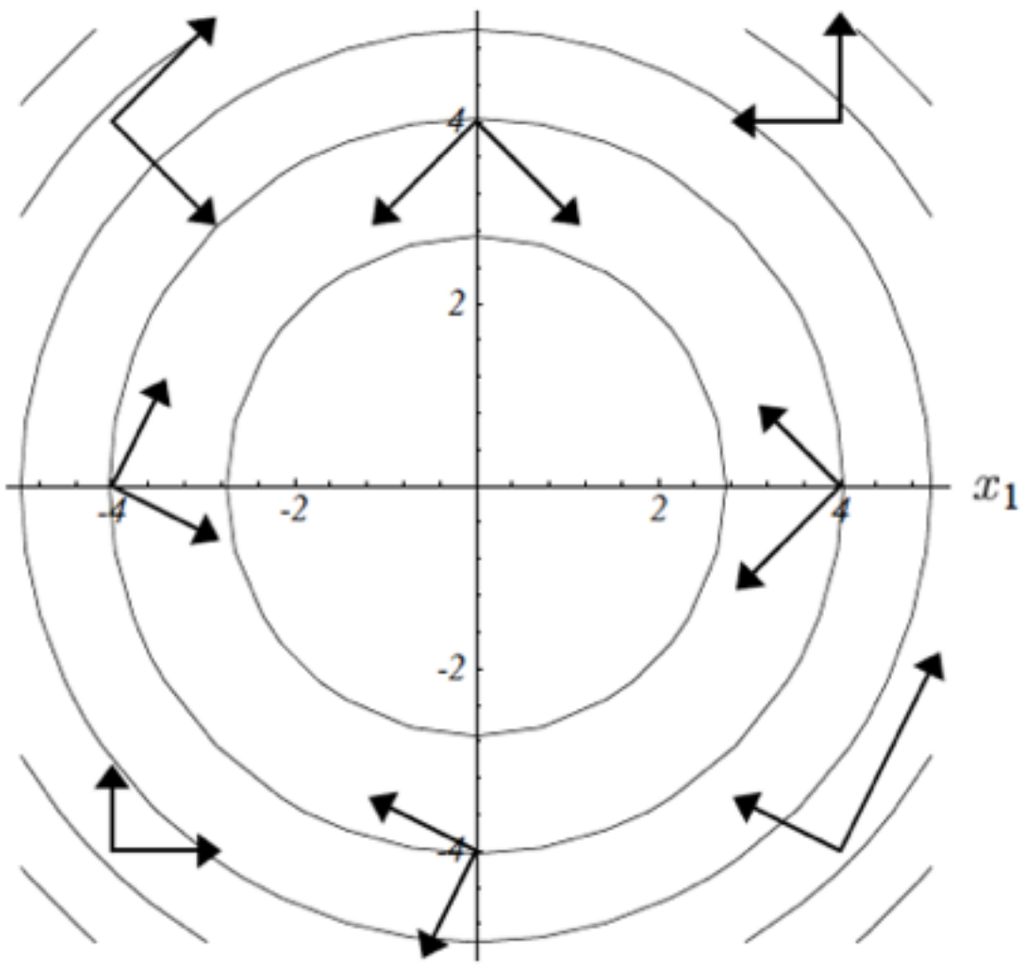
orthogonal *search directions* $d_{(0)}, d_{(1)}, \ldots, d_{(n-1)}$

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

$$\begin{aligned} d_{(i)}^T e_{(i+1)} &= 0 \\ d_{(i)}^T (e_{(i)} + \alpha_{(i)} d_{(i)}) &= 0 \\ \alpha_{(i)} &= -\frac{d_{(i)}^T e_{(i)}}{d_{(i)}^T d_{(i)}} \end{aligned}$$

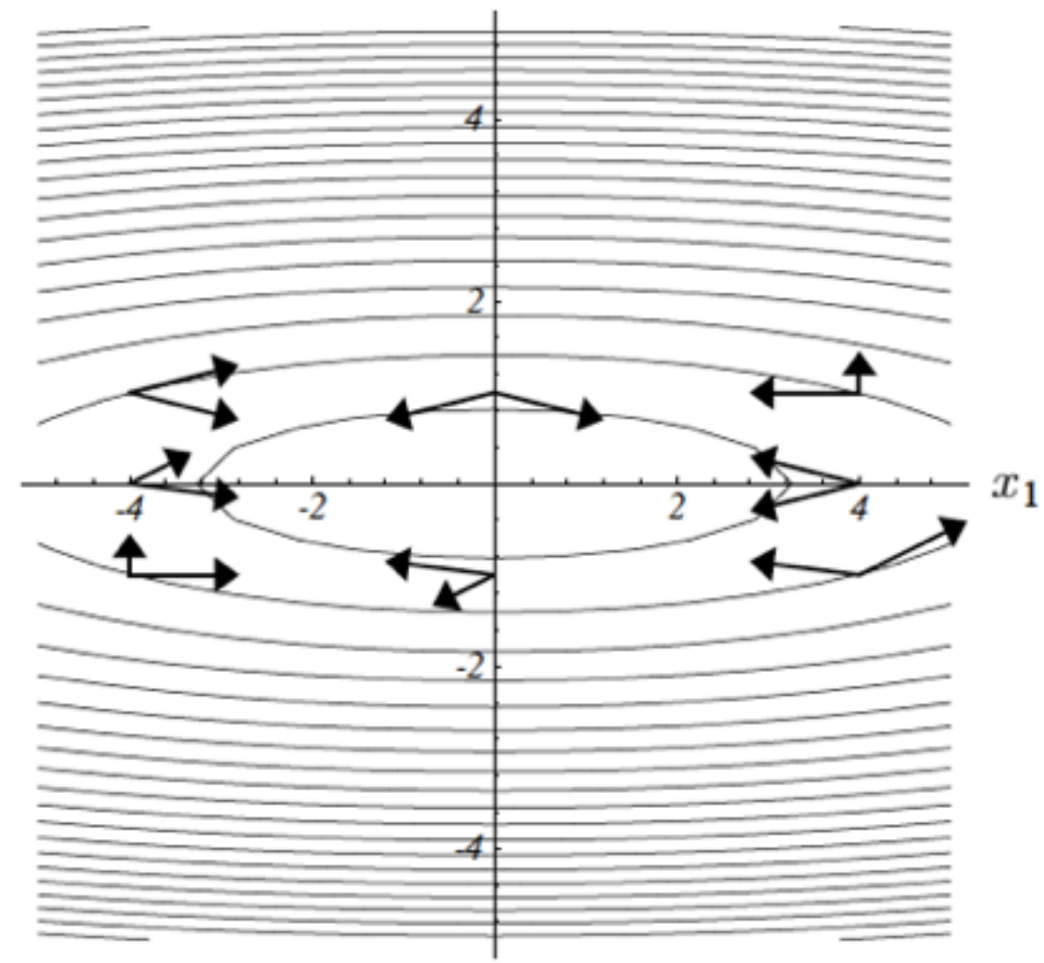
if we knew $e_{(i)}$, the problem would already be solved

Conjugate direction



orthogonal

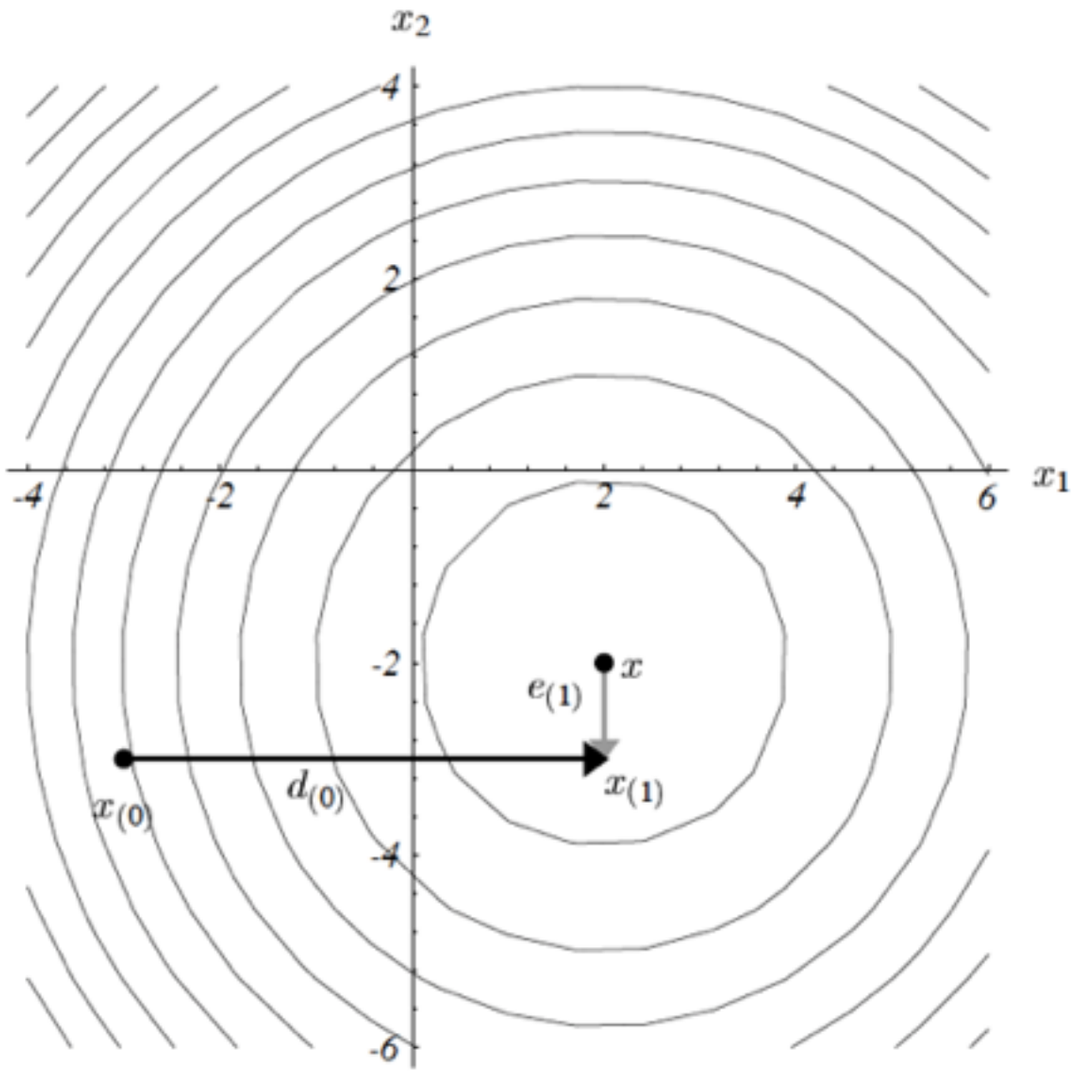
$$d_{(i)}^T d_{(i)} = 0$$

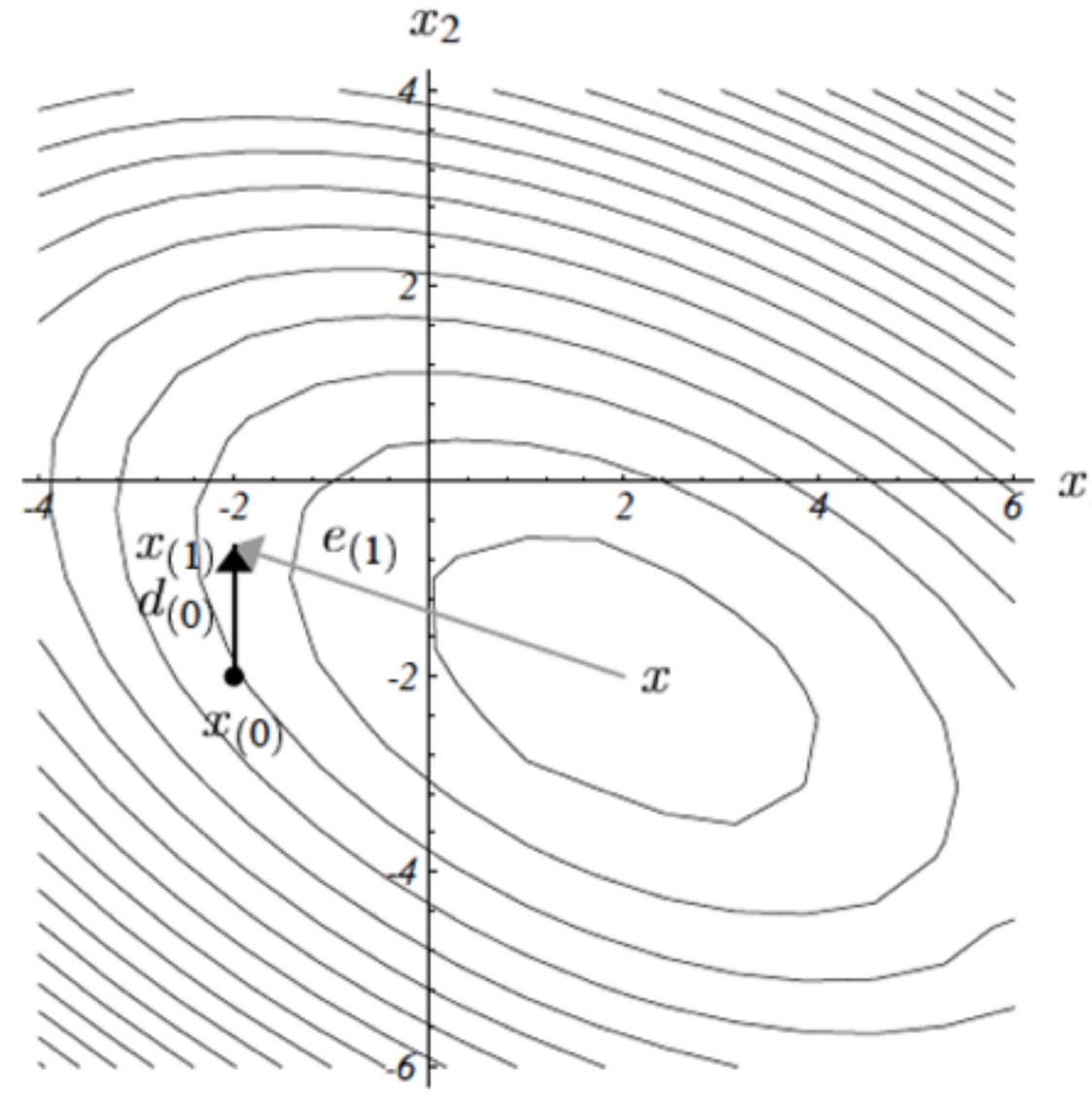


A-orthogonal $=$ *conjugate*

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

Conjugate direction



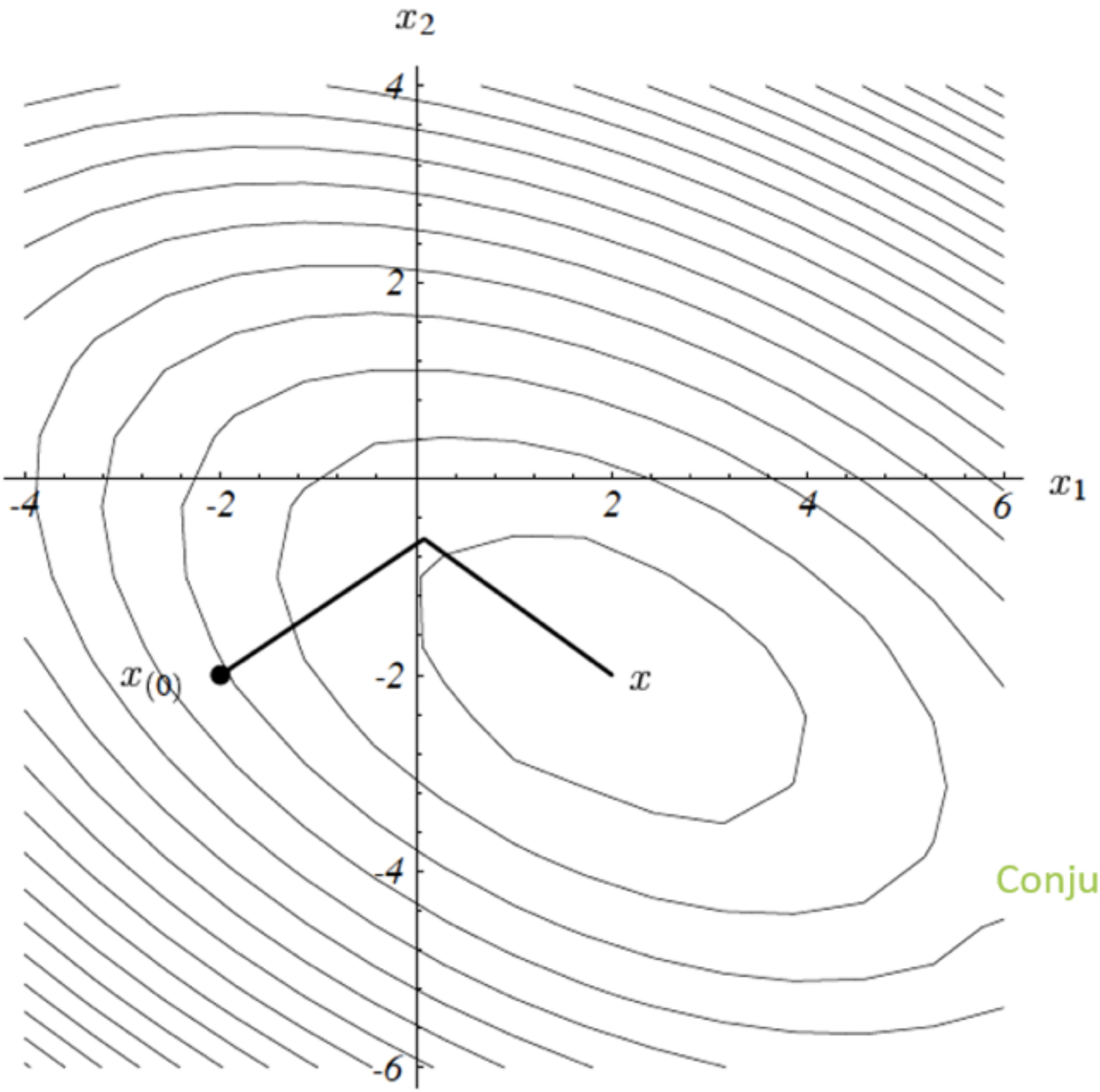


$$\alpha_{(i)} = -\frac{d_{(i)}^T e_{(i)}}{d_{(i)}^T d_{(i)}}$$

$$\alpha_{(i)} = -\frac{d_{(i)}^T A e_{(i)}}{d_{(i)}^T A d_{(i)}} = \frac{d_{(i)}^T r_{(i)}}{d_{(i)}^T A d_{(i)}}.$$

~~if we knew $e_{(i)}$, the problem would already be solved~~

Conjugate gradient



$$d_{(0)} = r_{(0)} = b - Ax_{(0)}$$

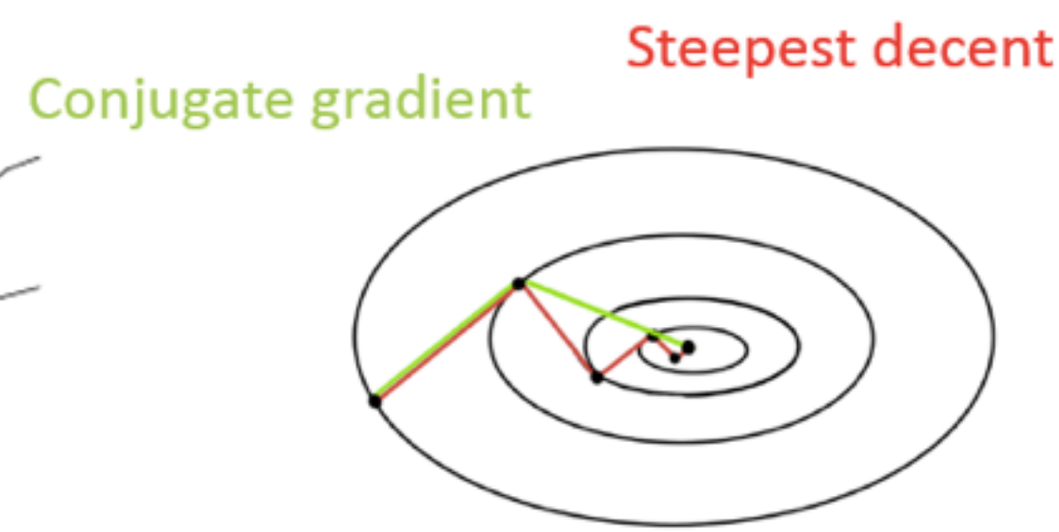
$$\alpha_{(i)} = \frac{r_{(i)}^T r_{(i)}}{d_{(i)}^T A d_{(i)}}$$

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

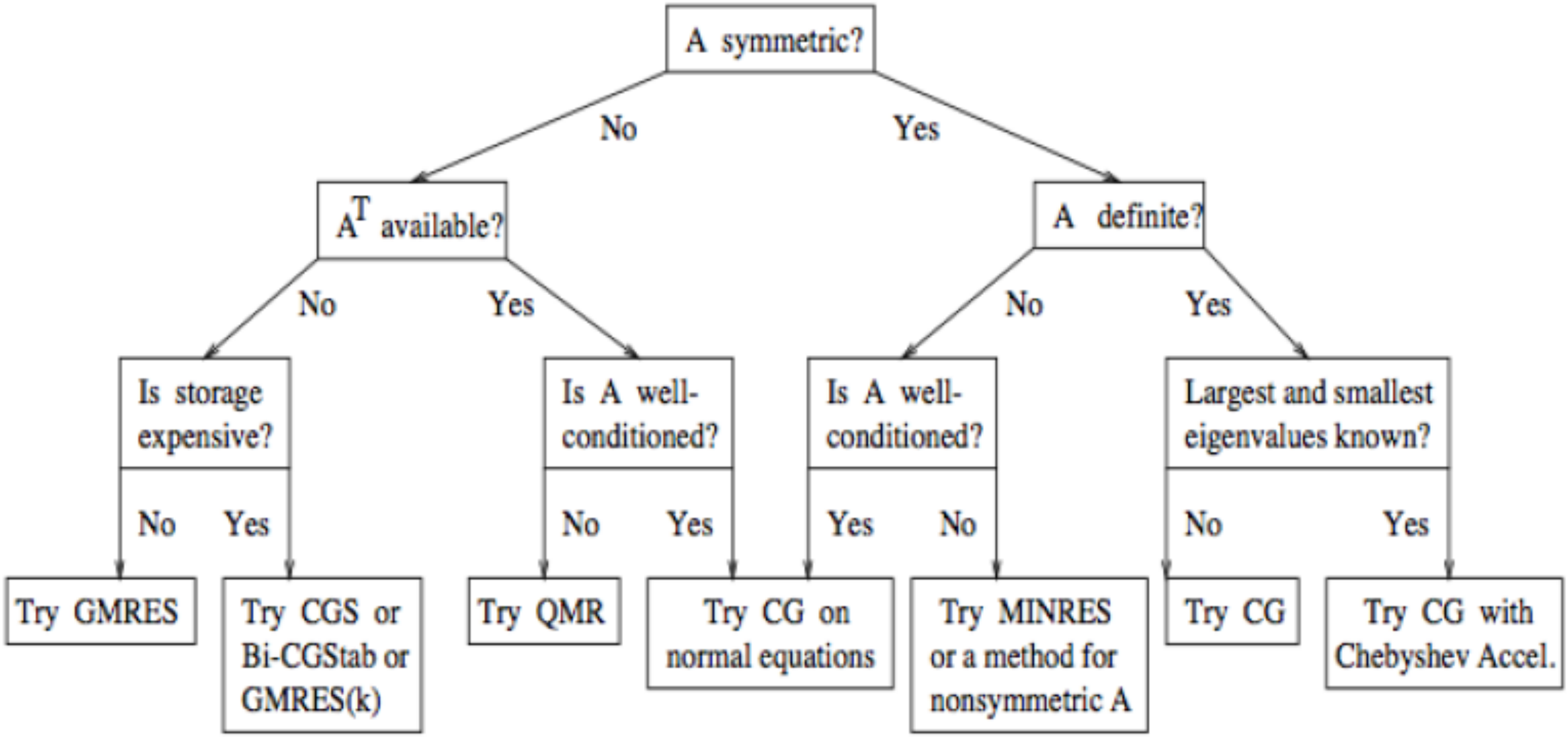
$$r_{(i+1)} = r_{(i)} - \alpha_{(i)} A d_{(i)},$$

$$\beta_{(i+1)} = \frac{r_{(i+1)}^T r_{(i+1)}}{r_{(i)}^T r_{(i)}},$$

$$d_{(i+1)} = r_{(i+1)} + \beta_{(i+1)} d_{(i)}$$

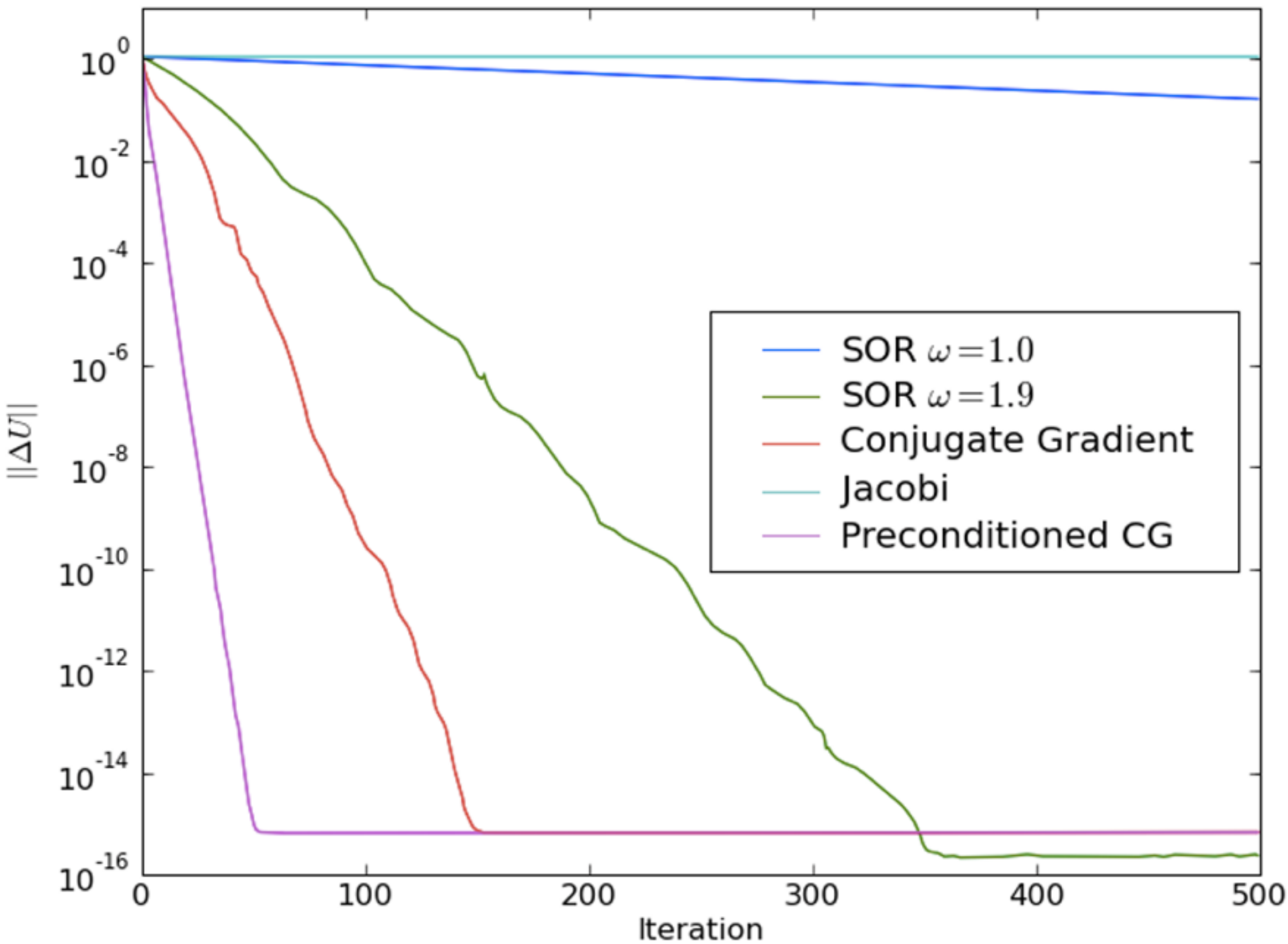


Choosing a Krylov method



source: J. Demmel

Preconditioning



05/09	Class 9	Dense direct solvers	Understand the principle of LU decomposition and the optimization and parallelization techniques that lead to the LINPACK benchmark.
05/12	Class 10	Dense eigensolvers	Determine eigenvalues and eigenvectors and understand the fast algorithms for diagonalization and orthonormalization.
05/16	Class 11	Sparse direct solvers	Understand reordering in AMD and nested dissection, and fast algorithms such as skyline and multifrontal methods.
05/19	Class 12	Sparse iterative solvers	Understand the notion of positive definiteness, condition number, and the difference between Jacobi, CG, and GMRES.
05/23	Class 13	Preconditioners	Understand how preconditioning affects the condition number and spectral radius, and how that affects the CG method.
05/26	Class 14	Multigrid methods	Understand the role of smoothers, restriction, and prolongation in the V-cycle.
05/30	Class 15	Fast multipole methods, H-matrices	Understand the concept of multipole expansion and low-rank approximation, and the role of the tree structure.