Solving linear system

Francesco Sanfilippo

Istituto Nazionale di Fisica Nucleare - Sezione Roma Tre

27 - 31 March 2017

Contents of Day 1

Introduction of the problem

1. Direct solution of linear system $A\,x = b$
2. Quadratic functional minimization

Iterative solver

1. Advantages
2. Comparison of efficiency

Contents of Day 2

Checking the solution

1. Limits, stability and efficiency of various algorithms
2. Convergence criterions

Accelerating the convergence

1. Mixed precision algorithms
2. Choosing a starting guess
3. Preconditioning the problem

Solving similar problems at the same time

1. Shifted problems $A + \sigma \mathrm{Id}$
2. Deflating the problem

Contents of Day 3

Review of Parallelisation

1. Distributed memory
2. Shared memory
3. Vectors

Gather/scatter approaches

$\rightarrow$ 1+2 different examples of gathering of non-local data

Contents of Day 4

More specifically on parallelisation

1. Communication/computation overlap
2. Multithreading
3. Vectorization

An example of a physical application

$\rightarrow$ Lattice QCD

Clear deals, long friendship

Please

1. Pay attention during lectures
2. Work alone
3. Ask questions!!!!!!

The assignment

Write a parallel conjugate gradient solver

for the Laplace problem

The assignment

Write a parallel conjugate gradient solver

for the Laplace problem

Day 1: implement direct and iterative solvers
Day 2: add features and implement the Laplace problem
Day 3-4: parallelize

The assignment

Write a parallel conjugate gradient solver

for the Laplace problem

Day 1: implement direct and iterative solvers
Day 2: add features and implement the Laplace problem
Day 3-4: parallelize

Remarks

- Write a single report per day
- Write in the language you prefer, I suggest C without too many frills
- Well commented code will be appreciated
- Specify how to compile/run your code (makefile, script, command...)
- Optional parts count as a bonus

Let's start...

Which problems are we interested to?

Discretisation of Differential equations

$$A(i,j)\,x(j) = b(i) \quad \rightarrow \boldsymbol{A_{ij}} x_j = b_i$$

Example:

$$\nabla^2 \phi = \psi \quad \rightarrow \quad \underbrace{\left[\delta_{ij} - \frac{\delta_{i,j+\hat{1}} + \delta_{i,j-\hat{1}}}{2}\right]}_{\boldsymbol{A_{ij}}} \underbrace{\phi_j}_{x} = \underbrace{\psi_i}_{b}$$

Which problems are we interested to?

Discretisation of Differential equations

$$A(i,j)\,x(j) = b(i) \quad \rightarrow \boldsymbol{A_{ij}} x_j = b_i$$

Example:

$$\nabla^2\phi = \psi \quad \rightarrow \quad \underbrace{\left[\delta_{ij} - \frac{\delta_{i,j+\hat{1}} + \delta_{i,j-\hat{1}}}{2}\right]}_{\boldsymbol{A_{ij}}} \underbrace{\phi_j}_{x} = \underbrace{\psi_i}_{b}$$

Optimization

- Minimization of a functional: find x such that the functional $F(x)$ is minimum.
- The problem of minimizing F is strictly connected with solving linear system

Which problems are we interested to?

Discretisation of Differential equations

$$A(i,j)\,x(j) = b(i) \quad \rightarrow \boldsymbol{A_{ij}} x_j = b_i$$

Example:

$$\nabla^2\phi = \psi \quad \rightarrow \quad \underbrace{\left[\delta_{ij} - \frac{\delta_{i,j+\hat{1}} + \delta_{i,j-\hat{1}}}{2}\right]}_{\boldsymbol{A_{ij}}} \underbrace{\phi_j}_{x} = \underbrace{\psi_i}_{b}$$

Optimization

- Minimization of a functional: find x such that the functional $F(x)$ is minimum.
- The problem of minimizing F is strictly connected with solving linear system

Focus: how to solve

$$\boldsymbol{A}\,x = b$$

Which problems are we interested to?

Discretisation of Differential equations

$$A(i,j)\,x(j) = b(i) \quad \rightarrow \boldsymbol{A_{ij}} x_j = b_i$$

Example:
$$\nabla^2 \phi = \psi \quad \rightarrow \quad \underbrace{\left[\delta_{ij} - \frac{\delta_{i,j+\hat{1}} + \delta_{i,j-\hat{1}}}{2}\right]}_{\boldsymbol{A_{ij}}} \underbrace{\phi_j}_{x} = \underbrace{\psi_i}_{b}$$

Optimization

- Minimization of a functional: find x such that the functional $F(x)$ is minimum.
- The problem of minimizing F is strictly connected with solving linear system

Focus: how to solve

$$\boldsymbol{A}\,x = b$$

Big problem

- We aim at solving **BIG** linear systems
- Look at way to speed-up solution splitting the problem across multiple computers

Triangular decomposition $A = L \cdot U$

Decompose A in the product of a lower triangular (L) and an upper triangular matrix (U)

For example:

$$\underbrace{\begin{pmatrix} 3 & 2 & 7 \\ 5 & 4 & 2 \\ 5 & 1 & 7 \end{pmatrix}}_{A} = \underbrace{\begin{pmatrix} 1 & & \\ 5/3 & 1 & \\ 5/3 & -7/2 & 1 \end{pmatrix}}_{L} \cdot \underbrace{\begin{pmatrix} 3 & 2 & 7 \\ & 2/3 & -29/3 \\ & & -77/2 \end{pmatrix}}_{U}$$

Direct solution of linear system LU decomposition

Triangular decomposition $A = L \cdot U$

Decompose A in the product of a lower triangular (L) and an upper triangular matrix (U)

For example:

$$\underbrace{\begin{pmatrix} 3 & 2 & 7 \\ 5 & 4 & 2 \\ 5 & 1 & 7 \end{pmatrix}}_{A} = \underbrace{\begin{pmatrix} 1 & & \\ 5/3 & 1 & \\ 5/3 & -7/2 & 1 \end{pmatrix}}_{L} \cdot \underbrace{\begin{pmatrix} 3 & 2 & 7 \\ & 2/3 & -29/3 \\ & & -77/2 \end{pmatrix}}_{U}$$

Solution

- Calling $y = Ux$ rewrite and solve the system: $Ly = b$

In our example:

$$\underbrace{\begin{pmatrix} 1 & & \\ 5/3 & 1 & \\ 5/3 & -7/2 & 1 \end{pmatrix}}_{L} \underbrace{\begin{pmatrix} y_0 \\ y_1 \\ y_2 \end{pmatrix}}_{y} = \underbrace{\begin{pmatrix} 3 \\ 2 \\ -1 \end{pmatrix}}_{b} \rightarrow \begin{cases} y_0 = 3 \\ y_1 = -3 \\ y_2 = 9/2 \end{cases}$$

- Repeat for: $Ux = y$ with the y just computed and obtain finally x

Direct solution of linear system LU decomposition

Triangular decomposition $A = L \cdot U$

Decompose A in the product of a lower triangular (L) and an upper triangular matrix (U)

For example:

$$\underbrace{\begin{pmatrix} 3 & 2 & 7 \\ 5 & 4 & 2 \\ 5 & 1 & 7 \end{pmatrix}}_{A} = \underbrace{\begin{pmatrix} 1 & & \\ 5/3 & 1 & \\ 5/3 & -7/2 & 1 \end{pmatrix}}_{L} \cdot \underbrace{\begin{pmatrix} 3 & 2 & 7 \\ & 2/3 & -29/3 \\ & & -77/2 \end{pmatrix}}_{U}$$

Solution

- Calling $y = Ux$ rewrite and solve the system: $Ly = b$

In our example:

$$\underbrace{\begin{pmatrix} 1 & & \\ 5/3 & 1 & \\ 5/3 & -7/2 & 1 \end{pmatrix}}_{L} \underbrace{\begin{pmatrix} y_0 \\ y_1 \\ y_2 \end{pmatrix}}_{y} = \underbrace{\begin{pmatrix} 3 \\ 2 \\ -1 \end{pmatrix}}_{b} \rightarrow \begin{cases} y_0 & = 3 \\ y_1 & = -3 \\ y_2 & = 9/2 \end{cases}$$

- Repeat for: $Ux = y$ with the y just computed and obtain finally x

Central point: How to perform the LU decomposition?

Doolittle algorithm for $N \times N$ matrix

First step

Consider the matrix $\boldsymbol{A}$ and its element $a_{i,j}$, at first iteration one build the helping matrix:

$$\boldsymbol{L_1} = \begin{pmatrix} 1 & & \\ -a_{21}/a_{11} & 1 & \\ -a_{31}/a_{11} & 0 & 1 \end{pmatrix}$$

and rewrite:

$$\boldsymbol{A} = \boldsymbol{L_1^{-1}} \boldsymbol{L_1} \boldsymbol{A} = \boldsymbol{L_1^{-1}} \boldsymbol{B}$$

Doolittle algorithm for $N \times N$ matrix

First step

Consider the matrix $\boldsymbol{A}$ and its element $a_{i,j}$, at first iteration one build the helping matrix:

$$\boldsymbol{L_1} = \begin{pmatrix} 1 & & \\ -a_{21}/a_{11} & 1 & \\ -a_{31}/a_{11} & 0 & 1 \end{pmatrix}$$

and rewrite:

$$\boldsymbol{A} = \boldsymbol{L_1^{-1}} \boldsymbol{L_1} \boldsymbol{A} = \boldsymbol{L_1^{-1}} \boldsymbol{B}$$

In our example

$$\boldsymbol{A} = \begin{pmatrix} 3 & 2 & 7 \\ 5 & 4 & 2 \\ 5 & 1 & 7 \end{pmatrix}, \boldsymbol{L_1} = \begin{pmatrix} 1 & 0 & 0 \\ -5/3 & 1 & 0 \\ -5/3 & 0 & 1 \end{pmatrix} \text{ so that}$$

$$\boldsymbol{A} = \underbrace{\begin{pmatrix} 1 & 0 & 0 \\ 5/3 & 1 & 0 \\ 5/3 & 0 & 1 \end{pmatrix}}_{\boldsymbol{L_1^{-1}}} \cdot \underbrace{\begin{pmatrix} 3 & 2 & 7 \\ 0 & 2/3 & -29/3 \\ 0 & -7/3 & -14/3 \end{pmatrix}}_{\boldsymbol{A_1}}$$

Doolittle algorithm for $N \times N$ matrix

First step

Consider the matrix $\boldsymbol{A}$ and its element $a_{i,j}$, at first iteration one build the helping matrix:

$$\boldsymbol{L_1} = \begin{pmatrix} 1 & & \\ -a_{21}/a_{11} & 1 & \\ -a_{31}/a_{11} & 0 & 1 \end{pmatrix}$$

and rewrite:

$$\boldsymbol{A} = \boldsymbol{L_1^{-1}} \boldsymbol{L_1} \boldsymbol{A} = \boldsymbol{L_1^{-1}} \boldsymbol{B}$$

In our example

$$\boldsymbol{A} = \begin{pmatrix} 3 & 2 & 7 \\ 5 & 4 & 2 \\ 5 & 1 & 7 \end{pmatrix}, \boldsymbol{L_1} = \begin{pmatrix} 1 & 0 & 0 \\ -5/3 & 1 & 0 \\ -5/3 & 0 & 1 \end{pmatrix} \text{ so that}$$

$$\boldsymbol{A} = \underbrace{\begin{pmatrix} 1 & 0 & 0 \\ 5/3 & 1 & 0 \\ 5/3 & 0 & 1 \end{pmatrix}}_{\boldsymbol{L_1^{-1}}} \cdot \underbrace{\begin{pmatrix} 3 & 2 & 7 \\ 0 & 2/3 & -29/3 \\ 0 & -7/3 & -14/3 \end{pmatrix}}_{\boldsymbol{A_1}}$$

This way we eliminated all elements of $\boldsymbol{A}$ below a_{11}, now consider how to progress

Doolittle algorithm for $N \times N$ matrix

Generic step n

Starting from the matrix: $\boldsymbol{A_n} = \begin{pmatrix} \mathbb{U}_{n\times n} & \mathbb{R}_{n\times(N-n)} \\ 0_{(N-n)\times n} & \mathbb{S}_{(N-n)\times(N-n)} \end{pmatrix}$ we define

$$\boldsymbol{L_{n+1}} = \begin{pmatrix} \mathbb{I}_{n\times n} & & \\ 0 & 1 & \\ 0_{(N-n-1)\times n} & -a^n_{[n+1:N],n}/a^n_{n,n} & \mathbb{I}_{(N-n)\times(N-n)} \end{pmatrix}$$

and rewrite: $\boldsymbol{A = L_1^{-1} \dots L_{n+1}^{-1} L_{n+1} \dots L_1 A = L_1^{-1} \dots L_{n+1}^{-1} A_{n+1}}$

Doolittle algorithm for $N \times N$ matrix

Generic step n

Starting from the matrix: $\boldsymbol{A_n} = \begin{pmatrix} \mathbb{U}_{n\times n} & \mathbb{R}_{n\times(N-n)} \\ 0_{(N-n)\times n} & \mathbb{S}_{(N-n)\times(N-n)} \end{pmatrix}$ we define

$$\boldsymbol{L_{n+1}} = \begin{pmatrix} \mathbb{I}_{n\times n} & & \\ 0 & 1 & \\ 0_{(N-n-1)\times n} & -a^n_{[n+1:N],n}/a^n_{n,n} & \mathbb{I}_{(N-n)\times(N-n)} \end{pmatrix}$$

and rewrite: $\boldsymbol{A = L_1^{-1} \dots L_{n+1}^{-1} L_{n+1} \dots L_1 A = L_1^{-1} \dots L_{n+1}^{-1} A_{n+1}}$

In our example

$$\boldsymbol{A} = \underbrace{\begin{pmatrix} 1 & 0 & 0 \\ 5/3 & 1 & 0 \\ 5/3 & 0 & 1 \end{pmatrix}}_{\boldsymbol{L_1^{-1}}} \cdot \underbrace{\begin{pmatrix} 1 & & \\ 0 & 1 & \\ 0 & -7/2 & 1 \end{pmatrix}}_{\boldsymbol{L_2^{-1}}} \cdot \underbrace{\begin{pmatrix} 1 & & \\ 0 & 1 & \\ 0 & 7/2 & 1 \end{pmatrix}}_{\boldsymbol{L_2}} \cdot \underbrace{\begin{pmatrix} 3 & 2 & 7 \\ 0 & 2/3 & -29/3 \\ 0 & 0 & -14/3 \end{pmatrix}}_{\boldsymbol{B}}$$

$$= \underbrace{\begin{pmatrix} 1 & & \\ 5/3 & 1 & \\ 5/3 & -7/2 & 1 \end{pmatrix}}_{\boldsymbol{L}} \cdot \underbrace{\begin{pmatrix} 3 & 2 & 7 \\ & 2/3 & -29/3 \\ & & -77/2 \end{pmatrix}}_{\boldsymbol{U}}$$

Doolittle algorithm for $N \times N$ matrix

A generic $N \times N$ matrix decomposes as $\boldsymbol{A} = \boldsymbol{L}_1^{-1} \dots \boldsymbol{L}_N^{-1} \boldsymbol{L}_N \dots \boldsymbol{L}_1 \boldsymbol{A}$

Doolittle algorithm for $N \times N$ matrix

A generic $N \times N$ matrix decomposes as $\boldsymbol{A} = \boldsymbol{L}_1^{-1} \dots \boldsymbol{L}_N^{-1} \boldsymbol{L}_N \dots \boldsymbol{L}_1 \boldsymbol{A}$

Proof

- By construction, $\boldsymbol{L}_n \boldsymbol{X}$ is upper Hessenberg (has the the first n column empty below diagonal), so that
$$\boldsymbol{U} = \boldsymbol{L}_N \dots \boldsymbol{L}_1 \boldsymbol{A}$$
is an upper-triangular matrix
- By construction, $\boldsymbol{L}_n^{-1}$ is lower-triangular, and the product of two lower-triangular matrix is triangular as well, so that
$$\boldsymbol{L} = \boldsymbol{L}_1^{-1} \dots \boldsymbol{L}_N^{-1}$$
is lower triangular

Doolittle algorithm for $N \times N$ matrix

A generic $N \times N$ matrix decomposes as $\boldsymbol{A} = \boldsymbol{L}_1^{-1} \dots \boldsymbol{L}_N^{-1} \boldsymbol{L}_N \dots \boldsymbol{L}_1 \boldsymbol{A}$

Proof

- By construction, $\boldsymbol{L}_n \boldsymbol{X}$ is upper Hessenberg (has the the first n column empty below diagonal), so that
$$\boldsymbol{U} = \boldsymbol{L}_N \dots \boldsymbol{L}_1 \boldsymbol{A}$$
is an upper-triangular matrix
- By construction, $\boldsymbol{L}_n^{-1}$ is lower-triangular, and the product of two lower-triangular matrix is triangular as well, so that
$$\boldsymbol{L} = \boldsymbol{L}_1^{-1} \dots \boldsymbol{L}_N^{-1}$$
is lower triangular

Special case: an element of the diagonal is zero

$$\boldsymbol{A} = \begin{pmatrix} 0 & 2 & 7 \\ 5 & 4 & 2 \\ 5 & 1 & 7 \end{pmatrix} \rightarrow \boldsymbol{PA} = \begin{pmatrix} 5 & 4 & 2 \\ 0 & 2 & 7 \\ 5 & 1 & 7 \end{pmatrix} \text{ and solve } \boldsymbol{PA}x = \boldsymbol{P}b$$

$\boldsymbol{P}$ is a permutation matrix $\begin{pmatrix} & 1 & \\ 1 & & \\ & & 1 \end{pmatrix}$

Problems of direct approaches

Complication

- Ignoring sub-dominant scaling: $cost \propto N^3$
- Computation cost increases fast with matrix size!

Problems of direct approaches

Complication

- Ignoring sub-dominant scaling: $cost \propto N^3$
- Computation cost increases fast with matrix size!

Full-solution oriented

- The whole factorization must be carried out to the end
- No simple way to obtain just an approximated solution

Problems of direct approaches

Complication

- Ignoring sub-dominant scaling: $cost \propto N^3$
- Computation cost increases fast with matrix size!

Full-solution oriented

- The whole factorization must be carried out to the end
- No simple way to obtain just an approximated solution

Storing: Even if the matrix $\boldsymbol{A}$ is sparse, $\boldsymbol{U}$ will not be $\rightarrow$ potential memory problem

Problems of direct approaches

Complication

- Ignoring sub-dominant scaling: $cost \propto N^3$
- Computation cost increases fast with matrix size!

Full-solution oriented

- The whole factorization must be carried out to the end
- No simple way to obtain just an approximated solution

Storing: Even if the matrix $\boldsymbol{A}$ is sparse, $\boldsymbol{U}$ will not be $\rightarrow$ potential memory problem

Remarks on Parallelisation

- Can be parallelised splitting column of the matrix across different nodes

$$\left(\begin{array}{cc|cc} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{array}\right)$$

- However the workload is not fully balanced (more and more nodes sleeps toward the end)

Cayley-Hamilton theorem

Any function $f(\boldsymbol{A})$ of a $N \times N$ matrix $\boldsymbol{A}$ is a linear combination of its first N powers

$$f(\boldsymbol{A}) = \sum_{i=0}^{N} c_i \boldsymbol{A}^i$$

Implication:

$$f(\boldsymbol{A})b = \sum_{i=0}^{N} c_i \boldsymbol{A}^i b$$

Cayley-Hamilton theorem

Cayley-Hamilton theorem

Any function $f(\boldsymbol{A})$ of a $N \times N$ matrix $\boldsymbol{A}$ is a linear combination of its first N powers

$$f(\boldsymbol{A}) = \sum_{i=0}^{N} c_i \boldsymbol{A}^i$$

Implication:

$$f(\boldsymbol{A})b = \sum_{i=0}^{N} c_i \boldsymbol{A}^i b$$

Observation

Solving $\boldsymbol{A}x = b$ is equivalent to compute $x = \boldsymbol{A}^{-1}b$, indeed: $\boldsymbol{A}\boldsymbol{A}^{-1}b = b$

Cayley-Hamilton theorem

Cayley-Hamilton theorem

Any function $f(\boldsymbol{A})$ of a $N \times N$ matrix $\boldsymbol{A}$ is a linear combination of its first N powers

$$f(\boldsymbol{A}) = \sum_{i=0}^{N} c_i \boldsymbol{A}^i$$

Implication:

$$f(\boldsymbol{A})b = \sum_{i=0}^{N} c_i \boldsymbol{A}^i b$$

Observation

Solving $\boldsymbol{A}x = b$ is equivalent to compute $x = \boldsymbol{A}^{-1}b$, indeed: $\boldsymbol{A}\boldsymbol{A}^{-1}b = b$

Idea

$$x = \boldsymbol{A}^{-1}b = \sum_{i=0}^{n} c_i \boldsymbol{A}^i b = c_0 b + c_1 \boldsymbol{A}b + c_2 \boldsymbol{A}\left(\boldsymbol{A}b\right) + \cdots + c_N \boldsymbol{A}\left(\boldsymbol{A}^{N-1}b\right)$$

Advantages: If $\boldsymbol{A}$ is a sparse matrix, $\boldsymbol{A}^j b$ can be computed quickly

Solving in the Krylov basis

Krylov space

- For an $N \times N$ matrix $\boldsymbol{A}$ and vector b, the Krylov space of size k is given by:

$$\mathcal{K}^k(\boldsymbol{A}, b) = \mathrm{span}\{b, \boldsymbol{A}b, \boldsymbol{A}^2 b, \ldots \boldsymbol{A}^{k-1} b\}$$

- The solution $x = \boldsymbol{A}^{-1} b$ is contained in $\mathcal{K}^{N+1}(\boldsymbol{A}, b)$
- For $k < N + 1$ the space $\mathcal{K}^k(\boldsymbol{A}, b)$ contains approximated solutions

Solving in the Krylov basis

Krylov space

- For an $N \times N$ matrix $\boldsymbol{A}$ and vector b, the Krylov space of size k is given by:

$$\mathcal{K}^k(\boldsymbol{A}, b) = \text{span}\{b, \boldsymbol{A}b, \boldsymbol{A}^2 b, \dots \boldsymbol{A}^{k-1} b\}$$

- The solution $x = \boldsymbol{A}^{-1} b$ is contained in $\mathcal{K}^{N+1}(\boldsymbol{A}, b)$
- For $k < N + 1$ the space $\mathcal{K}^k(\boldsymbol{A}, b)$ contains approximated solutions

Building the solution in Krylov space of size k

- Build the Krylov space $\mathcal{K}^k(\boldsymbol{A}, b)$ by applying k times A to b
- In such space the "most accurate" solution x_k will be given by

$$x_k = d_0 b + d_1 \boldsymbol{A}b + d_2 \boldsymbol{A}(\boldsymbol{A}b) + \cdots + d_k \boldsymbol{A}\left(\boldsymbol{A}^{k-1} b\right)$$

for some particular set of d_i. When $k = N + 1$ we find the full solution.

Solving in the Krylov basis

Krylov space

- For an $N \times N$ matrix $\boldsymbol{A}$ and vector b, the Krylov space of size k is given by:

$$\mathcal{K}^k(\boldsymbol{A}, b) = \operatorname{span}\{b, \boldsymbol{A}b, \boldsymbol{A}^2 b, \dots \boldsymbol{A}^{k-1} b\}$$

- The solution $x = \boldsymbol{A}^{-1} b$ is contained in $\mathcal{K}^{N+1}(\boldsymbol{A}, b)$
- For $k < N + 1$ the space $\mathcal{K}^k(\boldsymbol{A}, b)$ contains approximated solutions

Building the solution in Krylov space of size k

- Build the Krylov space $\mathcal{K}^k(\boldsymbol{A}, b)$ by applying k times A to b
- In such space the "most accurate" solution x_k will be given by

$$x_k = d_0 b + d_1 \boldsymbol{A}b + d_2 \boldsymbol{A}(\boldsymbol{A}b) + \cdots + d_k \boldsymbol{A}\left(\boldsymbol{A}^{k-1} b\right)$$

for some particular set of d_i. When $k = N + 1$ we find the full solution.

Problems

- What does "most accurate" means?
- How do we find the coefficients d_i?
- Keeping in memory the whole basis $\{b, \boldsymbol{A}b, \boldsymbol{A}^2 b, \dots \boldsymbol{A}^{k-1} b\}$ might be impossible

Iterative solver

Summarising

Having a possibility to find the best solution in the Krylov space $\mathcal{K}^k(\boldsymbol{A}, b)$:

$$x_k = d_0 b + d_1 \boldsymbol{A} b + d_2 \boldsymbol{A}(\boldsymbol{A} b) + \cdots + d_k \boldsymbol{A}\left(\boldsymbol{A}^{k-1} b\right)$$

is attractive, because involve only matrix multiplication $\boldsymbol{A}v$:

- quick operations if $\boldsymbol{A}$ is a sparse matrix
- easily parallelisable (we will see)
- $\mathcal{K}^{k+1} \in \mathcal{K}^k$ so we can extend/improve the approximated solution progressively

Iterative solver

Summarising

Having a possibility to find the best solution in the Krylov space $\mathcal{K}^k(\boldsymbol{A}, b)$:

$$x_k = d_0 b + d_1 \boldsymbol{A} b + d_2 \boldsymbol{A}(\boldsymbol{A} b) + \cdots + d_k \boldsymbol{A}\left(\boldsymbol{A}^{k-1} b\right)$$

is attractive, because involve only matrix multiplication $\boldsymbol{A}v$:

- quick operations if $\boldsymbol{A}$ is a sparse matrix
- easily parallelisable (we will see)
- $\mathcal{K}^{k+1} \in \mathcal{K}^k$ so we can extend/improve the approximated solution progressively

Drawbacks

- Keeping in memory the Krylov space basis $\{b, \boldsymbol{A}b, \boldsymbol{A}^2 b, \ldots \boldsymbol{A}^{k-1} b\}$ can be impossible
- Determining d coefficients looks as difficult as the original problem

Iterative solver

Summarising

Having a possibility to find the best solution in the Krylov space $\mathcal{K}^k(\boldsymbol{A}, b)$:

$$x_k = d_0 b + d_1 \boldsymbol{A} b + d_2 \boldsymbol{A}(\boldsymbol{A} b) + \cdots + d_k \boldsymbol{A}\left(\boldsymbol{A}^{k-1} b\right)$$

is attractive, because involve only matrix multiplication $\boldsymbol{A}v$:

- quick operations if $\boldsymbol{A}$ is a sparse matrix
- easily parallelisable (we will see)
- $\mathcal{K}^{k+1} \in \mathcal{K}^k$ so we can extend/improve the approximated solution progressively

Drawbacks

- Keeping in memory the Krylov space basis $\{b, \boldsymbol{A}b, \boldsymbol{A}^2 b, \ldots \boldsymbol{A}^{k-1} b\}$ can be impossible
- Determining d coefficients looks as difficult as the original problem

Iterative algorithms

We need to design an algorithm that finds the best solution in the space $\mathcal{K}^{k+1}$ starting from the best solution of the space $\mathcal{K}^k$ without the need to hold the whole basis of $\mathcal{K}^k$

Iterative solver

Summarising

Having a possibility to find the best solution in the Krylov space $\mathcal{K}^k(\boldsymbol{A}, b)$:

$$x_k = d_0 b + d_1 \boldsymbol{A} b + d_2 \boldsymbol{A}(\boldsymbol{A} b) + \cdots + d_k \boldsymbol{A}\left(\boldsymbol{A}^{k-1} b\right)$$

is attractive, because involve only matrix multiplication $\boldsymbol{A}v$:

- quick operations if $\boldsymbol{A}$ is a sparse matrix
- easily parallelisable (we will see)
- $\mathcal{K}^{k+1} \in \mathcal{K}^k$ so we can extend/improve the approximated solution progressively

Drawbacks

- Keeping in memory the Krylov space basis $\{b, \boldsymbol{A}b, \boldsymbol{A}^2 b, \ldots \boldsymbol{A}^{k-1} b\}$ can be impossible
- Determining d coefficients looks as difficult as the original problem

Iterative algorithms

We need to design an algorithm that finds the best solution in the space $\mathcal{K}^{k+1}$ starting from the best solution of the space $\mathcal{K}^k$ without the need to hold the whole basis of $\mathcal{K}^k$

To do this, let's look at the problem from **another point of view**...

Functional minimization

Remark

IF A is hermitian and definite positive, the functional

$$F(x) = \frac{1}{2} x^T \boldsymbol{A} x - bx$$

is minimized when $\boldsymbol{A}x = b$

Functional minimization

Remark

IF A is hermitian and definite positive, the functional

$$F(x) = \frac{1}{2} x^T \boldsymbol{A} x - bx$$

is minimized when $\boldsymbol{A}x = b$

Check

- The gradient G of F is:
$$G(x) = \frac{\partial}{\partial x} F(x) = \boldsymbol{A}x - b$$
- Where $G(x) = 0$ the functional F is minimum
(and not a maximum: $\boldsymbol{A}$ is hermitian and definite positive by assumption)
- Therefore
$$G(x) = 0 \quad \rightarrow \quad \boldsymbol{A}x = b$$

Functional minimization

Remark

IF A is hermitian and definite positive, the functional

$$F(x) = \frac{1}{2} x^T \boldsymbol{A} x - bx$$

is minimized when $\boldsymbol{A}x = b$

Check

- The gradient G of F is:
$$G(x) = \frac{\partial}{\partial x} F(x) = \boldsymbol{A}x - b$$
- Where $G(x) = 0$ the functional F is minimum
(and not a maximum: $\boldsymbol{A}$ is hermitian and definite positive by assumption)
- Therefore
$$G(x) = 0 \quad \rightarrow \quad \boldsymbol{A}x = b$$

Idea

Minimize $F(x)$ to solve the linear system

Direct gradient algorithm - steepest descent

Minimize F by following the negative of gradient:

- Define $r_k(x_k) = b - \boldsymbol{A}x_k = -G(x_k)$ (this is the "residual" of the problem)
- Find the optimal step $x_{k+1} = x_k + \alpha_k r_k$ minimizing $F(x_{k+1})$

$$\begin{aligned} F(x_{k+1}) &= \frac{1}{2}(x_k + \alpha_k r_k)^T \boldsymbol{A}(x_k + \alpha_k r_k) - b(x_k + \alpha_k r_k) = \\ &= F(x_k) + -\alpha_k(r_k,\, r_k) + \frac{1}{2}\alpha_k^2(r_k,\, \boldsymbol{A}r_k) \end{aligned}$$

w.r.t α_k:

$$0 = \frac{\partial F(x_{k+1})}{\partial \alpha_k} = -(r_k,\, r_k) + \alpha_k(r_k,\, \boldsymbol{A}r_k) \rightarrow \alpha_k = \frac{(r_k,\, r_k)}{(r_k,\, \boldsymbol{A}r_k)}$$

Direct gradient algorithm - steepest descent

Minimize F by following the negative of gradient:

- Define $r_k(x_k) = b - \boldsymbol{A}x_k = -G(x_k)$ (this is the "residual" of the problem)
- Find the optimal step $x_{k+1} = x_k + \alpha_k r_k$ minimizing $F(x_{k+1})$

$$\begin{aligned} F(x_{k+1}) &= \frac{1}{2}(x_k + \alpha_k r_k)^T \boldsymbol{A}(x_k + \alpha_k r_k) - b(x_k + \alpha_k r_k) = \\ &= F(x_k) + -\alpha_k(r_k,\, r_k) + \frac{1}{2}\alpha_k^2(r_k,\, \boldsymbol{A}r_k) \end{aligned}$$

w.r.t α_k:

$$0 = \frac{\partial F(x_{k+1})}{\partial \alpha_k} = -(r_k,\, r_k) + \alpha_k(r_k,\, \boldsymbol{A}r_k) \rightarrow \alpha_k = \frac{(r_k,\, r_k)}{(r_k,\, \boldsymbol{A}r_k)}$$

Remark: apparently two applications of A per iteration are required, but

- At the first iteration $x_0 = 0$ so $r_0 = b$
- At k iteration $r_{k+1} = b - \boldsymbol{A}x_{k+1} = b - \boldsymbol{A}(x_k + \alpha_k r_k) = r_k - \alpha_k \boldsymbol{A}r_k$
- But $p_k = \boldsymbol{A}r_k$ is what was used to compute α_k:

$$\alpha_k = \frac{(r_k,\, r_k)}{(r_k,\, p_k)}$$

at each step we can compute the residue for following step

Direct gradient algorithm - steepest descent

Algorithm

- At first step $x_0 = 0$, $r_0 = b$
- At each step
  - compute $p_k = \boldsymbol{A} r_k$
  - compute $\alpha_k = \frac{(r_k, r_k)}{(r_k, p_k)}$
  - update x : $x_{k+1} = x_k + \alpha_k r_k$
  - update r : $r_{k+1} = r_k - \alpha_k p_k$

Direct gradient algorithm - steepest descent

Algorithm

- At first step $x_0 = 0$, $r_0 = b$
- At each step
  - compute $p_k = \boldsymbol{A} r_k$
  - compute $\alpha_k = \frac{(r_k, r_k)}{(r_k, p_k)}$
  - update x : $x_{k+1} = x_k + \alpha_k r_k$
  - update r : $r_{k+1} = r_k - \alpha_k p_k$

Remarks

Little computation: At each step we only have to apply once the matrix

Low memory usage: Only last solution and last vector r must be stored

Direct gradient algorithm - steepest descent

Algorithm

- At first step $x_0 = 0$, $r_0 = b$
- At each step
  - compute $p_k = \boldsymbol{A} r_k$
  - compute $\alpha_k = \frac{(r_k, r_k)}{(r_k, p_k)}$
  - update x : $x_{k+1} = x_k + \alpha_k r_k$
  - update r : $r_{k+1} = r_k - \alpha_k p_k$

Remarks

Little computation: At each step we only have to apply once the matrix

Low memory usage: Only last solution and last vector r must be stored

When we do stop?

- We have no way to compute the true error $e_k = x - x_k$ (would require to know x!)
- Minimum $\rightarrow |r| = 0$, r is named *residue* (image of e_k: $\boldsymbol{A} e_k = \boldsymbol{A}(x - x_k) = r_k$)
- Converging iteratively to the solution $\rightarrow$ choose a target residue $|r|_{targ}$
- Absolute residue of little meaning, better to choose: $\hat{r}_{targ} = |r| / |b|$
- Also δF_k can give information on the convergence (but we don't know the absolute minimum $F(x)$

Direct gradient algorithm - steepest descent

Produced chain of solutions

- At step 0: $x_0 = 0$
- At step 1: $x_1 = \underbrace{\frac{(b,b)}{(b,\boldsymbol{A}b)}}_{d_0} b$
- At step 2: $x_2 = \underbrace{\left[\frac{(b,b)}{(b,\boldsymbol{A}b)} + \frac{(b,b)^2}{(b,\boldsymbol{A}b)^2}(b,\boldsymbol{A}^2 b)\right]}_{d_0} b - \underbrace{\left[\frac{(b,b)^3}{(b,\boldsymbol{A}b)^3}(b,\boldsymbol{A}^2 b)\right]}_{d_1} \boldsymbol{A}b$

Direct gradient algorithm - steepest descent

Produced chain of solutions

- At step 0: $x_0 = 0$
- At step 1: $x_1 = \underbrace{\frac{(b,b)}{(b,\boldsymbol{A}b)}}_{d_0} b$
- At step 2: $x_2 = \underbrace{\left[\frac{(b,b)}{(b,\boldsymbol{A}b)} + \frac{(b,b)^2}{(b,\boldsymbol{A}b)^2}(b,\boldsymbol{A}^2 b)\right]}_{d_0} b - \underbrace{\left[\frac{(b,b)^3}{(b,\boldsymbol{A}b)^3}(b,\boldsymbol{A}^2 b)\right]}_{d_1} \boldsymbol{A}b$

Remarks

- Coefficients d gets more and more complicated at each iterations
- This is not a problem: at each iteration they are automatically updated

$$x_{k+1} = x_k + \alpha_k r_k, \quad r_{k+1} = r_k - \alpha_k p_k$$

Direct gradient algorithm - steepest descent

Produced chain of solutions

- At step 0: $x_0 = 0$
- At step 1: $x_1 = \underbrace{\frac{(b,b)}{(b,\boldsymbol{A}b)}}_{d_0} b$
- At step 2: $x_2 = \underbrace{\left[\frac{(b,b)}{(b,\boldsymbol{A}b)} + \frac{(b,b)^2}{(b,\boldsymbol{A}b)^2}(b,\boldsymbol{A}^2 b)\right]}_{d_0} b - \underbrace{\left[\frac{(b,b)^3}{(b,\boldsymbol{A}b)^3}(b,\boldsymbol{A}^2 b)\right]}_{d_1} \boldsymbol{A}b$

Remarks

- Coefficients d gets more and more complicated at each iterations
- This is not a problem: at each iteration they are automatically updated

$$x_{k+1} = x_k + \alpha_k r_k, \quad r_{k+1} = r_k - \alpha_k p_k$$

Is this the best we can do?

- We have not proved that the solution at iteration k we minimized the functional, that is it's not clear that

$$F(x_k) \leq \min [F(x)] \, \forall x \in \mathcal{K}^k$$

- Indeed it turns out **not to be the case** at all!!!

Direct gradient algorithm - Is this the best we can do?

Graphical representation

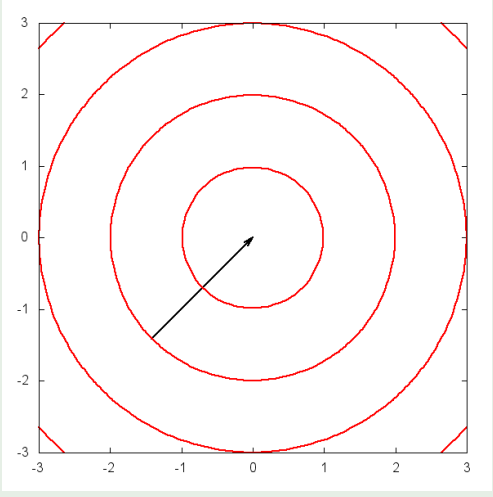
- The quadratic form $F(v) = \frac{1}{2} v^T \boldsymbol{A} v - bv$ is a paraboloid in an N dimension space
- Let's look at minimization in a 2D example

Direct gradient algorithm - Is this the best we can do?

Graphical representation

- The quadratic form $F(v) = \frac{1}{2} v^T \boldsymbol{A} v - bv$ is a paraboloid in an N dimension space
- Let's look at minimization in a 2D example

Example



$$A = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

$$b = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

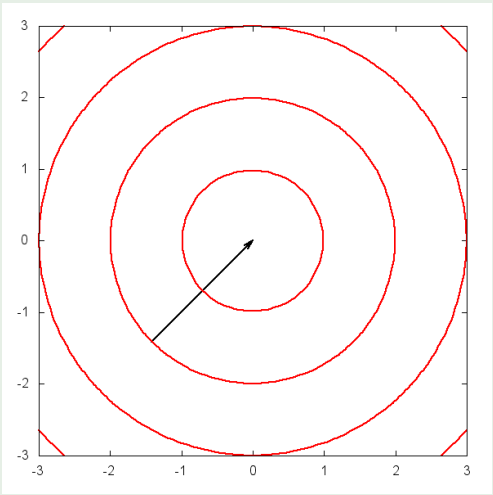
$$x_0 = -\begin{pmatrix} \sqrt{2} \\ \sqrt{2} \end{pmatrix}$$

Direct gradient algorithm - Is this the best we can do?

Graphical representation

- The quadratic form $F(v) = \frac{1}{2} v^T \boldsymbol{A} v - bv$ is a paraboloid in an N dimension space
- Let's look at minimization in a 2D example

Example



$$A = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

$$b = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

$$x_0 = -\begin{pmatrix} \sqrt{2} \\ \sqrt{2} \end{pmatrix}$$

In this case $\boldsymbol{A}$'s eigenvalues are all equals, convergence is very fast

Direct gradient algorithm - Is this the best we can do?

Graphical representation

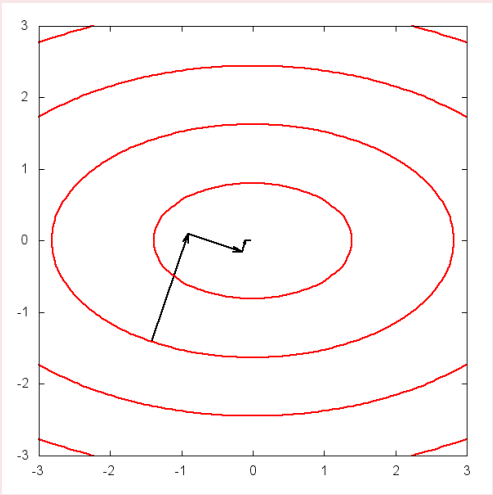
- The quadratic form $F(v) = \frac{1}{2} v^T \boldsymbol{A} v - bv$ is a paraboloid in an N dimension space
- Let's look at minimization in a 2D example

Direct gradient algorithm - Is this the best we can do?

Graphical representation

- The quadratic form $F(v) = \frac{1}{2} v^T \boldsymbol{A} v - bv$ is a paraboloid in an N dimension space
- Let's look at minimization in a 2D example

Another Example



$$A = \begin{pmatrix} 0.5 & 0 \\ 0 & 1.5 \end{pmatrix}$$

$$b = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

$$x_0 = -\begin{pmatrix} \sqrt{2} \\ \sqrt{2} \end{pmatrix}$$

In this case $\boldsymbol{A}$'s eigenvalues are **not the same**, convergence is slow!

Direct gradient algorithm - Is this the best we can do?

Problem

- As the contours of F looks less ellipsoidal, the gradient of F points further and further away from the true minimum
- Minimization is forced to proceed through orthogonal zig-zag steps
- In typical problem the eigenvalues of $\boldsymbol{A}$ will be scattered over several order of magnitude
- The problem is more and more severe as the dimension of the matrix increases
- Immediate convergence if we chose initial guess such that $r_0 \parallel e_0$,
  but this would need to know x in advance

Direct gradient algorithm - Is this the best we can do?

Problem

- As the contours of F looks less ellipsoidal, the gradient of F points further and further away from the true minimum
- Minimization is forced to proceed through orthogonal zig-zag steps
- In typical problem the eigenvalues of $\boldsymbol{A}$ will be scattered over several order of magnitude
- The problem is more and more severe as the dimension of the matrix increases
- Immediate convergence if we chose initial guess such that $r_0 \parallel e_0$, but this would need to know x in advance

Comments

- We forced to move along the direction of the gradient and to minimize F
- Not necessarily this is the best choice

Direct gradient algorithm - Is this the best we can do?

Problem

- As the contours of F looks less ellipsoidal, the gradient of F points further and further away from the true minimum
- Minimization is forced to proceed through orthogonal zig-zag steps
- In typical problem the eigenvalues of $\boldsymbol{A}$ will be scattered over several order of magnitude
- The problem is more and more severe as the dimension of the matrix increases
- Immediate convergence if we chose initial guess such that $r_0 \parallel e_0$, but this would need to know x in advance

Comments

- We forced to move along the direction of the gradient and to minimize F
- Not necessarily this is the best choice

Can we to better staying inside the Krylov space $\mathcal{K}^k$?

Yes: We can impose to arrive to the solution in $N+1$ steps, by allowing to make use of last iteration to improve the step

Direct gradient algorithm - Conjugate basis

How to arrive to the solution in $N+1$ steps $p_{0...}p_N$

- decompose $x = \sum_{j=0}^{N} \alpha_j p_j$ where p_j are independent vectors and α_j scalars
- this way $x_k = x_{k-1} + \alpha_k p_k$

Direct gradient algorithm - Conjugate basis

How to arrive to the solution in $N+1$ steps $p_{0...}p_N$

- decompose $x = \sum_{j=0}^{N} \alpha_j p_j$ where p_j are independent vectors and α_j scalars
- this way $x_k = x_{k-1} + \alpha_k p_k$

How to compute p_k?

- we can build it in term of residue and of previous step p_{k-1}:

$$p_k = r_k + \beta_k p_{k-1}$$

- if we impose $(p_i, \boldsymbol{A} p_j) = 0 \quad \forall j \neq k$ (conjugation), and choose $p_0 = b$:

$$\beta_k = \frac{(r_k,\, r_k)}{(r_{k-1},\, r_{k-1})}$$

Direct gradient algorithm - Conjugate basis

How to arrive to the solution in $N+1$ steps $p_0 \ldots p_N$

- decompose $x = \sum_{j=0}^{N} \alpha_j p_j$ where p_j are independent vectors and α_j scalars
- this way $x_k = x_{k-1} + \alpha_k p_k$

How to compute p_k?

- we can build it in term of residue and of previous step p_{k-1}:

$$p_k = r_k + \beta_k p_{k-1}$$

- if we impose $(p_i,\ \boldsymbol{A} p_j) = 0 \quad \forall j \neq k$ (conjugation), and choose $p_0 = b$:

$$\beta_k = \frac{(r_k,\ r_k)}{(r_{k-1},\ r_{k-1})}$$

How to compute α_j?

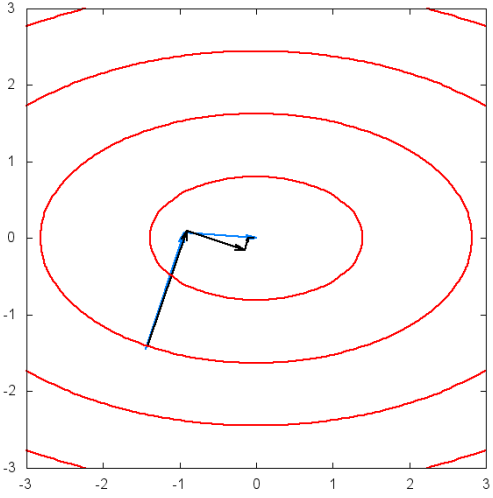
- by minimizing F along p_k: $\alpha_j = \frac{(r_{k-1}, r_{k-1})}{(p_j, \boldsymbol{A} p_j)}$
- r update is achieved as before:

$$r_{k+1} = r_k - \alpha_k p_k$$

Direct gradient algorithm - Conjugate basis

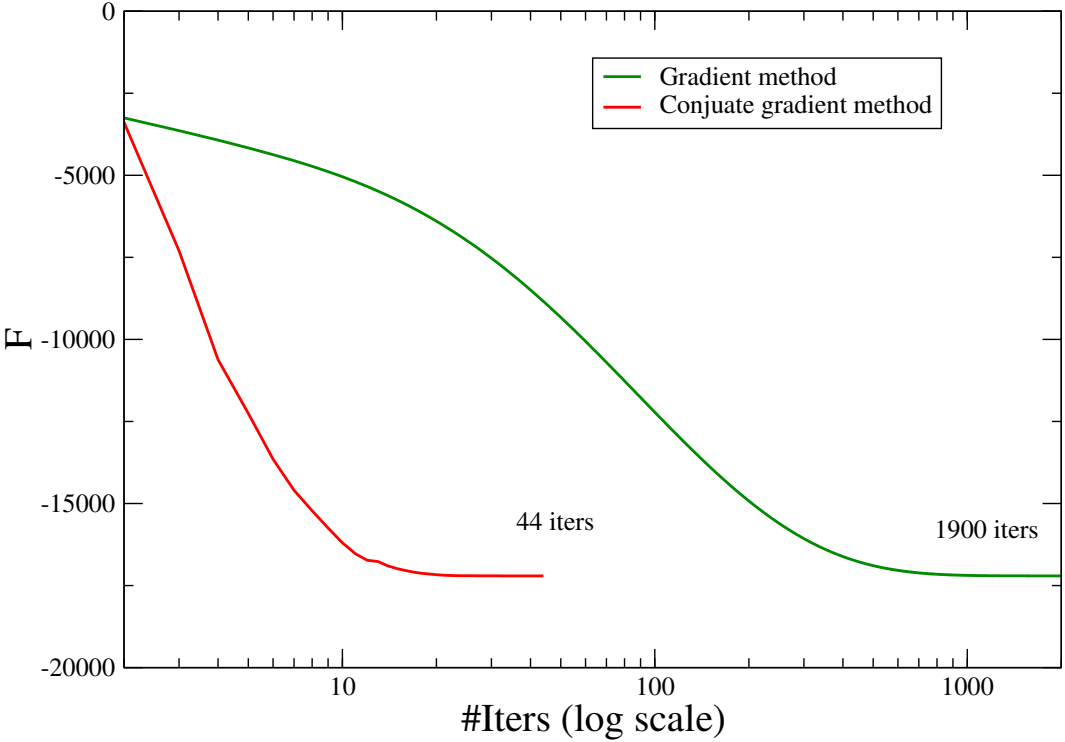
Why is it better?

- At step k we are building a base for $\mathcal{K}^k$
- By construction the method converges in at most N iterations
- It can be proved that this way we minimize F on the k-size Krylov space $\mathcal{K}^k$:
  - at every step k we just have to optimize F in the new direction p_k
  - we never have to go back optimizing past direction again

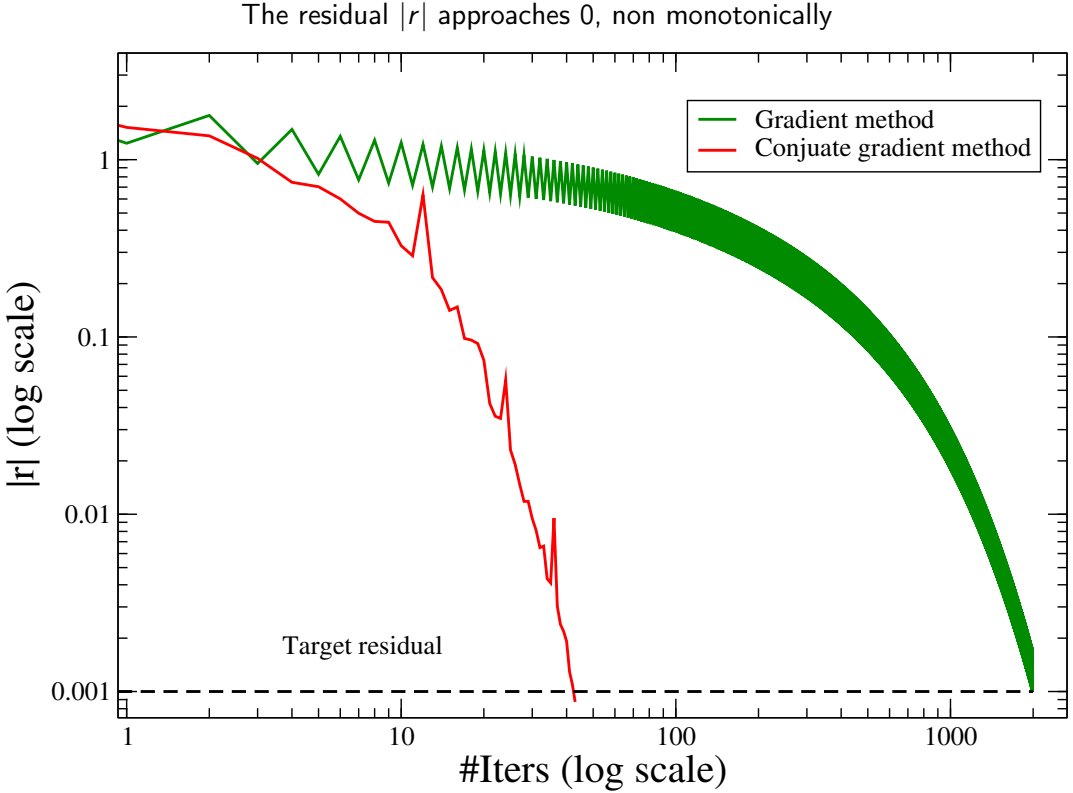


Convergence to $|r|_{targ} = 10^{-3}$ with a 140×140 matrix

The functional F approach its (unknown) minimum monotonically

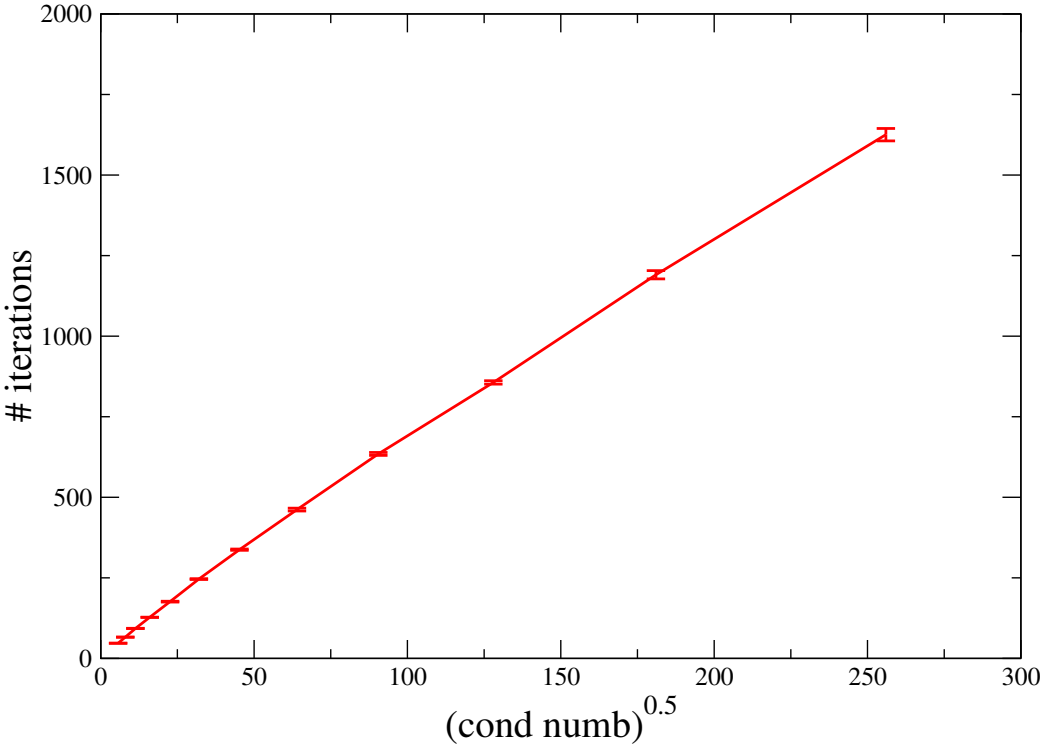


Convergence to $|r|_{targ} = 10^{-3}$ with a 140×140 matrix



Conjugate gradient - convergence rate

At fixed matrix size: $\#iters \propto \sqrt{\kappa}, \quad \kappa = \lambda_{max}/\lambda_{min}$



Conjugate gradient - convergence rate