Modelling and implementation of algorithms in applied mathematics using MPI

Lecture 4: Conjugate Gradient (CG) method, Programming MPI

G. Rapin

Brazil
March 2011

Outline

1 Conjugate Gradient Method

2 Programming MPI

Outline

1 Conjugate Gradient Method

2 Programming MPI

Conjugate Gradient Method

- Today, simple iterative methods like Gauss-Seidel or Jacobi method are not used very often in practice. But they are very important as smoother in multi grid methods.
- Krylov subspace methods have become quite popular.
- The most applied Krylov method for symmetric, positive definite problems is the conjugate gradient (CG) method.
- The most popular Krylov subspace methods for non-symmetric problems are BiCG, GMRES ord BiCGStab.
- The conjugate gradient (CG) method was invented by M.R. HESTENES and E. STIEFEL and was published in 1952.

Problem Statement

Consider the linear system

$$Ax = b. \tag{1}$$

Let the matrix $A \in \mathbb{R}^{n \times n}$ be

- symmetric, i.e. $A = A^T$, and
- positive definite, i.e.

$$x^T A x > 0, \qquad x \setminus \{0\} \in \mathbb{R}^n.$$

The right hand side is given by $b \in \mathbb{R}^n$.

Characterization

The solution $x \in \mathbb{R}^n$ of the linear system (1) can be identified by the solution of the optimization problem

$$\text{Minimize } Q(x) := \frac{1}{2}x^T Ax - b^T x, \quad x \in \mathbb{R}^n \tag{2}$$

Lemma

$x \in \mathbb{R}^n$ is a solution of (1) if and only if it is a solution of (2).

Proof

Is x a solution of (2), then there holds

$$0 = (\nabla Q(x))^T = Ax - b.$$

The condition is also sufficient since the Hesse matrix A is s.p.d.

Idea of the CG method

- Instead of attacking the linear system directly techniques for the solution of the global optimization problem (2) are used.
- Most of the iterative approaches in global optimization are based on sequences

  $$x^{k+1} = x^k + \alpha_k p^k,$$

  There, x^k is the previous iteration, p^k is the *search direction* and α_k is the *step size*.
- The step size α_k can be determined by

  $$Q(x^{k+1}) = Q(x^k + \alpha_k p^k) = \min_{\alpha_k \in \mathbb{R}} Q(x^k + \alpha_k p^k)$$

  for a given search direction.

- Thus, Q is minimized on the straight line $G := \{x = x^k + \beta p^k,\ \beta \in \mathbb{R}\}$.
- Inserting Q yields

$$\begin{aligned} q(\alpha) &:= Q(x^k + \alpha p^k) = \\ & \frac{1}{2}(x^k + \alpha p^k)^T A(x^k + \alpha p^k) - b^T(x^k + \alpha p^k) \\ &= \frac{1}{2}(p^k)^T A p^k \cdot \alpha^2 \\ & +(p^k)^T(Ax^k - b) \cdot \alpha + \frac{1}{2}(x^k)^T(Ax^k - 2b). \end{aligned}$$

- Minimizing q yields

$$0 = q'(\alpha) = (p^k)^T A p^k \cdot \alpha + (p^k)^T(Ax^k - b).$$

Step Size

Theorem

Assume that the search direction $p^k \neq 0$ *has already been determined. Then, we obtain the new iteration* $x^{k+1} = x^k + \alpha_k p^k$ *with*

$$\alpha_k = -\frac{(p^k)^T(Ax^k - b)}{(p^k)^T A p^k}. \tag{3}$$

Remark

Now the search path has to be determined.

Search Direction - Choice 1

- Use circularly the unity vectors e_i as search directions

$$p^0 = e_1,\ p^1 = e_2, \ldots, p^{n-1} = e_n,\ p^n = e_1,\ p^{n+1} = e_2, \ldots$$

- We get

$$(e_i)^T A e_i = a_{ii} \quad \text{and} \quad (e_i)^T (Ax - b) = \sum_{j=1}^{n} a_{ij} x_j - b_i.$$

- Inserting the choice (3) yields

$$x^k = x^{k-1} + \alpha_{k-1} p^{k-1} = x^{k-1} - \frac{1}{a_{kk}} \left(\sum_{j=1}^{n} a_{kj} x_j^{k-1} - b_k \right) e_k$$

for $k = 0, 1, \ldots, n-1$.

Search Direction - Choice 1

- Thus, only the k-th component of the vector is updated.
- If we consider one cycle, we get

$$x_k^k = \frac{1}{a_{kk}} \left(b_k - \sum_{j<k} a_{kj} x_j^k - \sum_{j>k} a_{kj} x_j^0 \right), \quad x_i^k = x_i^{k-1}, k \neq i.$$

- Therefore, one cycle is one Gauss-Seidel iteration.

Search Direction - Choice 2

- The steepest descent is given by the negative gradient of Q.
- Thus, we choose

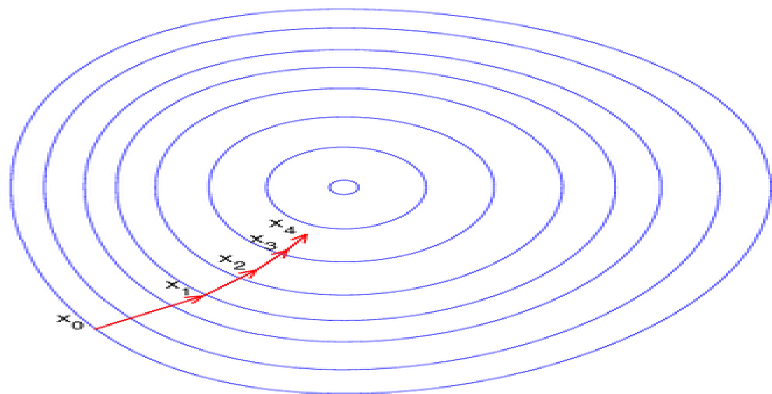
$$p^k = -(\nabla Q(x^k))^T = b - Ax^k.$$

- Then, the step size can be computed as

$$\alpha_k = \frac{(Ax^k - b)^T (Ax^k - b)}{(Ax^k - b)^T A (Ax^k - b)}.$$

- The step size is positive, if x^k is not the solution.

Visualization of Steepest Descent

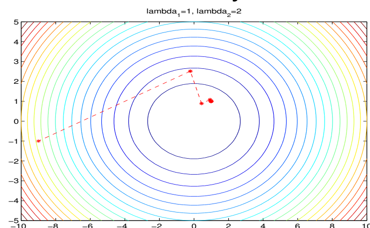


Steepest Descent - Bad if $\lambda_{max}/\lambda_{min}$ is large

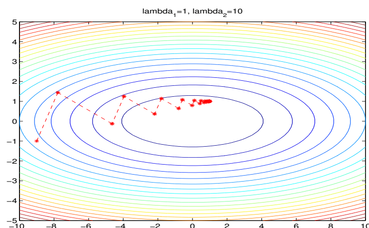
We consider $Ax = b$ with

$$A = \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix}, \qquad b = \begin{pmatrix} \lambda_1 \\ \lambda_2 \end{pmatrix}, \tag{4}$$

using the start vector $(-9, -1)$ and solution $(1, 1)$. Reduction of euclidean norm by 10^{-4}.



$\lambda_1 = 1$, $\lambda_2 = 2$
9 iterations



$\lambda_1 = 1$, $\lambda_2 = 10$
41 iterations

Search Direction - Choice 3

Idea:
Determine the search path p^k in such a way p^k is A-conjugated w.r.t. the previous search paths $p^0, p^1, \ldots, p^{k-1}$.

Definition

Two vectors $x, y \in \mathbb{R}^n$ are called *A-conjugated*, if $x^T Ay = 0$.

We can prove

Theorem

Assume, that $p^0, p^1, \ldots, p^{n-1} \neq 0$ are pairwise A-conjugated vectors. Then, the scheme

$$x^{k+1} = x^k + \alpha_k p^k$$

converges in at most n steps against the exact solution. α_k is given by (3).

Algorithm 1: **Conjugate Gradient (CG) method**

- *Choose initial vector* x^0.
- *Set* $p^0 = r^0 = Ax^0 - b$
- *Compute* $\gamma_0 = (r^0)^T r^0$.
- *For* $k = 0, 1, \ldots$
  1. $z^k = Ap^k$
  2. $\alpha_k = -\gamma_k / ((p^k)^T z^k)$.
  3. $x^{k+1} = x^k + \alpha_k p^k$
  4. $r^{k+1} = r^k + \alpha_k z^k$
  5. $\gamma_{k+1} = (r^{k+1})^T r^{k+1}$
  6. *if* $\gamma_{k+1} < TOL$ *stop*
  7. $\beta_k = \gamma_{k+1} / \gamma_k$
  8. $p^{k+1} = r^{k+1} + \beta_k p^k$
- *end k*

Remarks

- It can be proved, that the vectors p^k are pair-wise A-conjugated.
- Theoretically, after at most n steps the solution can be computed. Due to rounding errors in practice you will not get the solution after n steps.
- In practice we have $n >> 1$. Therefore, the CG method is used as an iterative method.
- In each iteration step there are one matrix-vector product, 2 scalar products and 3 scalar multiplications necessary.
- Besides the matrix A you have to store 4 additional vectors: x^k, r^k, p^k and z^k.

CG as Krylov subspace method

Theorem

The k-th iteration x^k of the CG method minimizes the functional $Q(\cdot)$ w.r.t. the subspace

$$\mathcal{K}_k(A, r^0) = span\{r^0, Ar^0, A^2 r^0, \ldots A^{k-1} r^0\},$$

i.e. there holds.

$$Q(x^k) = \min_{c_i} \; Q(x^0 + \sum_{i=0}^{k-1} c_i A^i r^0).$$

The subspace $\mathcal{K}_k(A, r^0)$ is called Krylov subspace.

Error Estimate

The error is $e^k = x^k - x^*$ is measured in the *energy norm*

$$\|u\|_A := (u^T A u)^{\frac{1}{2}}.$$

We get the estimate

Theorem

$$\|x^k - x^*\|_A \leq 2 \left(\frac{\sqrt{\kappa(A)} - 1}{\sqrt{\kappa(A)} + 1} \right)^k \|x^0 - x^*\|_A \tag{5}$$

with $\kappa(A) := cond_2(A) = \frac{\lambda_{max}(A)}{\lambda_{min}(A)} \geq 1$.

Remark

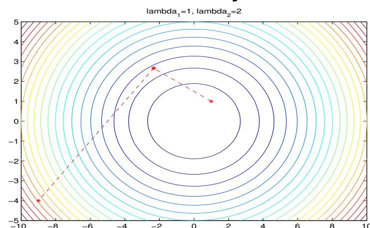
The matrix of the Finite Differences gives $cond_2(A) = \mathcal{O}(h^{-2})$.

Example of CG method

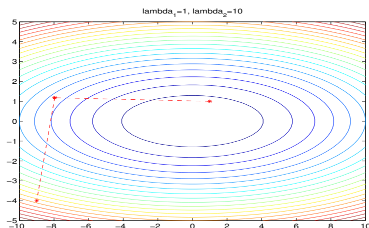
We consider $Ax = b$ with

$$A = \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix}, \qquad b = \begin{pmatrix} \lambda_1 \\ \lambda_2 \end{pmatrix}, \tag{6}$$

using the start vector $(-9, -1)$ and solution $(1, 1)$. Reduction of euclidean norm by 10^{-4}.



$\lambda_1 = 1$, $\lambda_2 = 2$
2 iterations



$\lambda_1 = 1$, $\lambda_2 = 10$
2 iterations

CG method (Parallel Version)

Algorithm 2: **CG – parallel Version (First Part)**

- *Choose initial vector x^0.*
- *Scatter x^0, b on the processes.*
- *Compute $p^0 = r^0 = Ax^0 - b$ in parallel.*
- *Compute $\gamma_0 = (r^0)^T r^0$ with Fan-in (`MPI_Allreduce`).*

CG method (Parallel Version)

Algorithm 3: **CG – parallel Version (Second Part)**

- *For* $k = 0, 1, \dots$
  1. $z^k = Ap^k$ *(parallel)*
  2. $\alpha_k = -\gamma_k / ((p^k)^T z^k)$ *with Fan-in (*`MPI_Allreduce`*).*
  3. $x^{k+1} = x^k + \alpha_k p^k$ *in parallel*
  4. $r^{k+1} = r^k + \alpha_k z^k$ *in parallel*
  5. $\gamma_{k+1} = (r^{k+1})^T r^{k+1}$ *with Fan-in (*`MPI_Allreduce`*).*
  6. *if* $\gamma_{k+1} < TOL$ *stop*
  7. $\beta_k = \gamma_{k+1} / \gamma_k$ *(parallel)*
  8. $p^{k+1} = r^{k+1} + \beta_k p^k$ *(parallel)*
  9. *Interchange components of* p^{k+1} *between the components.*
- *end* k

Outline

1 Conjugate Gradient Method

2 Programming MPI

Grouping of Data

- Sending/ receiving messages is a time consumable operation in a parallel environment.
- One should try to send as few messages as possible.
- It makes sense to combine different data packages to one single package.
- MPI provides several operations and data structures for grouping of data.

Example

We consider the example from the exercises about numerical integration, where we have to send two floats *a*, *b* and one integer *n* to all processes.
We assume that the data is stored on process 0 as follows

Variable	Address	Typ
a	24	`float`
b	40	`float`
n	48	`int`

The relative addresses, called *displacements*, in relation to the start address of *a* are stored. Since *a* has the address `&a=24`, the displacement for *b* is $40 - 24 = 16$ and for *n* we get $48 - 24 = 24$.

Transmitted Information

- 3 elements
- data type of elements
  - The first element is a `float`.
  - The second element is a `float`.
  - The third element is a `int`.
- Displacements
  - The first element has a displacement of 0.
  - The second element has a displacement of 16.
  - The third element has a displacement of 24.
- The starting address is `&a`.

In the derived MPI data type we will store a sequence of pairs

$$\{(t_0, d_0), (t_1, d_1), \ldots, (t_{n-1}, d_{n-1}).\}$$

t_i is one of the basic MPI data types and d_i is the displacement in bytes.

Grouping Data

The derived data type is built using the following command `MPI_Type_struct`:

	int MPI_Type_struct(int count, int block_length[], MPI_Aint displacements[], MPI_Datatype typelist[], MPI_Datatype* new_mpi_t)
IN:	
count	number of blocks, which should be combined.
block_length	Array of block lengths
displacements	Array of displacements
typelist	Array of MPI data types
OUT:	
new_mpi_t	Pointer on the new structure

The arrays `block_length`, `displacements` and `typelist` have the dimension `count`.

Grouping Data

The address of a variable with data type `MPI_Aint` can be obtained using

int MPI_Address(void* location, MPI_Aint address)	
IN:	
location	memory address
OUT:	
address	address in bytes

Example: Computation of displacements

```
MPI_Address(a_ptr, &start_address);
MPI_Address(b_ptr, &address);
displacements[1] = address - start_address;
```

Grouping Data

Finally, you have to start `MPI_Type_commit`:

	int MPI_Type_commit(MPI_Datatype* new_mpi_t)
IN:	
new_mpi_t	new structure
OUT:	
new_mpi_t	new structure

Grouping non-contiguous data of the same type

	int MPI_Type_vector(int count, int block_length, int stride, MPI_Datatype element_type, MPI_Datatype* new_mpi_t)
IN:	
count	number of blocks
block_length	length of a block
stride	number of elements between two blocks new_mpi_t plus 1
element_type	MPI data type
OUT:	
new_mpi_t	new element of type MPI_Type_struct

Staring from the initial address `count` blocks are built. All blocks have the same size `block_length` and consist of elements of type `element_type`. The variable `stride` determines the size of the jump.

Grouping Data

Example Sending the column of a matrix $A[10][10]$.

```
MPI_Datatype column_mpi_t;
MPI_Type_vector(10,1,10,MPI_FLOAT,&column_mpi_t)
MPI_Type_commit(&column_mpi_t);
/* Send 4th column */
MPI_Send(&(A[0][3]), 1, column_mpi_t, 1, 0,
    MPI_COMM_WORLD);
```

Pack/ Unpack

	int MPI_Pack (void* Pack_data, int in_count, MPI_Datatype datatype, void* buffer, int buffer_size, int* position, MPI_Comm comm)
IN:	
Pack_data	pointer on data, which should be added to the buffer
in_count	number of elements
datatype	data type of data
buffer_size	buffer size
position	position in buffer
comm	communicator
OUT:	
buffer	buffer
position	position Puffer

Pack/ Unpack

- With MPI_Pack you can add data to an existing buffer.
- Die Variable position is an input/ output parameter. The data is written in the buffer beginning at position. After return of the function position points to the first position behind the written data.
- buffer_size is the size of the buffer.

Pack/ Unpack

	int MPI_Unpack(void* buffer, int size, int* position, void* unpack_data, int count, MPI_Datatype datatype, MPI_Comm comm)
IN:	
buffer	buffer
size	size of buffer in bytes
position	position in buffer
count	number of elements to be unpacked
datatype	data type
comm	communicator
OUT:	
unpack_data	block of unpacked data
position	position in buffer

Groups and Communicators

- A group is an ordered set of processes. Each process in a group is associated with a unique integer rank. Rank values start at zero and go to N-1, where N is the number of processes in the group.
- A communicator consists of a group of processes that may communicate with each other and a context. All MPI messages must specify a communicator. A context is a system-defined object that uniquely identifies a communicator.
- From the programmer's perspective, a group and a communicator are one. The group routines are primarily used to specify which processes should be used to construct a communicator.

Groups

	int MPI_Comm_group(MPI_Comm comm, MPI_Group* group)
IN:	
comm	communicator
OUT:	
group	group of the communicator

The command `MPI_Comm_group` returns the group of the communicator `comm`.

Groups

	int MPI_Group_incl (MPI_Group old_group, int new_group_size, int ranks[], MPI_Group *newgroup)
IN:	
old_group	old group
new_group_size	size of new group
ranks	array of (old) process numbers
OUT:	
newgroup	new group

The command creates a group with name `new_group` consisting of `new_group_size` processes.

Build Communicator

	int MPI_Comm_create(MPI_Comm old_comm, MPI_Group new_group, MPI_Comm* new_comm)
IN:	
old_comm	old communicator
new_group	name of group
OUT:	
new_comm	new communicator