Numerical Algorithms for HPC

Sparse Linear Algebra: Introduction to Krylov subspace methods

epcc

Overview

- Iterative versus direct methods
- What is a Krylov subspace?
- Measuring the error
  - Residue/residual
- Steepest Descent
- Conjugate Gradient
  - Mathematical overview
  - Toy example
- Other KS methods
  - The GMRES and BiCGSTAB methods

epcc

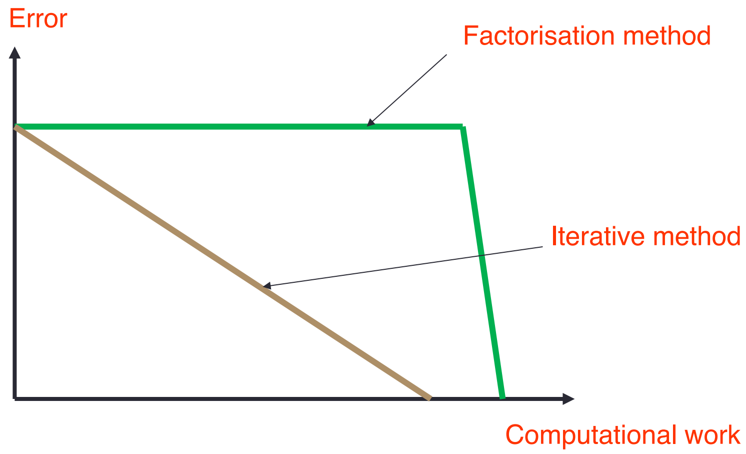
Properties of iterative schemes

- Iterative methods (e.g. Jacobi, Gauss-Seidel, Conjugate Gradient)
  - do not modify source matrix, involve matrix only through matrix-vector multiplication (possibly with transpose of matrix)
  - preserve sparsity and structure
    - Memory: Good for storage
  - progressively refine solution allowing user to impose accuracy constraints interactively
  - operate on individual righthand sides

Properties of direct schemes

- Direct (factorisation) methods (e.g. LU factorisation)
  - act on source matrix, destroying structure such as sparsity
  - involve redundant calculations on zero elements
    - Memory: storage of zero elements
  - produce fixed accuracy solutions in a prescribed number of steps
  - can be used efficiently for multiple righthand sides

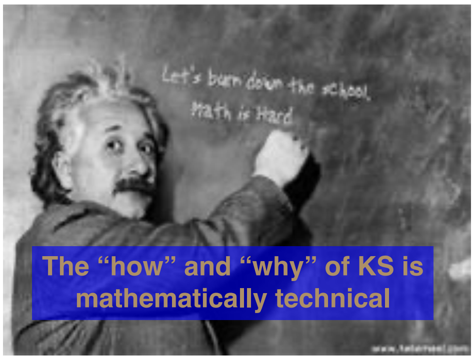
Convergence of schemes

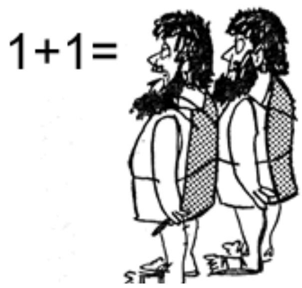


epcc 5

5

Mathematic caveat





Mathematical proofs and derivations are beyond the scope of this course

Implementing KS is easy

epcc 6

6

Notation

- The algorithms involve a number of scalars, vectors and matrices
  - Both sometimes with subscripts!
- For clarity we will use the following notation:
  - Bold upper case for a matrix: $\boldsymbol{A}$
  - Bold lower case for a vector: $\boldsymbol{v}_k$
  - Non-bold lower case for a matrix or vector element: $a_{3,2}, v_1$
  - Non-bold lower case for a scalar: α
  - Non-bold script for Krylov subspace: $\mathcal{K}_m$

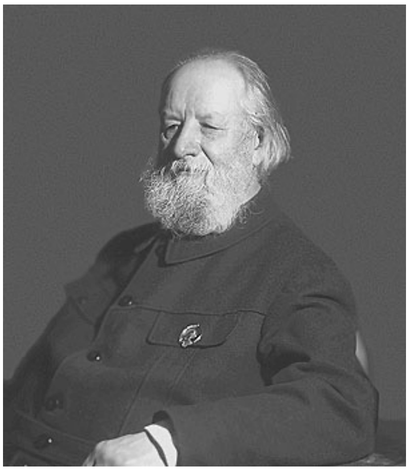
|epcc|

7

7

Alexei Krylov

- Russian Naval engineer and applied mathematician
- Photo taken in 1930s Krylov in his 60s
- One of first people to classify the amount of work required for a given computation
- Krylov subspaces are constructed from linear systems



|epcc|

8

8

Recall what a linear system is

- Recall that a linear system (of size N) can be represented by a matrix equation, of the form:

$$\boldsymbol{Av} = \boldsymbol{b}$$

$N \times N$ matrix

unknown solution to be determined ($\boldsymbol{N}$ components)

known righthand side (with $\boldsymbol{N}$ components)

- This is simply a representation of N equations linking N unknown quantities: $v_1, v_2, \ldots, v_N$

epcc

Krylov Subspace

- Definition: The Krylov subspace (KS) of size m is said to be *spanned* by the vectors $\boldsymbol{v}, \boldsymbol{Av}, \boldsymbol{A}^2\boldsymbol{v}, \ldots, \boldsymbol{A}^{m-1}\boldsymbol{v}$:

$$\mathcal{K}_m(\boldsymbol{A}, \boldsymbol{v}) \equiv \{\boldsymbol{v}, \boldsymbol{Av}, \boldsymbol{A}^2\boldsymbol{v}, \ldots, \boldsymbol{A}^{m-1}\boldsymbol{v}\}$$

- Krylov subspace is a property of matrix $\boldsymbol{A}$ and starting vector $\boldsymbol{v}$
  - e.g. $\boldsymbol{v}$ could be initial guess at solution
- Repeated application of Matrix $\boldsymbol{A}$ to $\boldsymbol{v}$
- Vector $\boldsymbol{v}$ is often known as the starting vector
- The KS is an m-dimensional *subspace* of the N-dimensional space where the matrix lives
  - i.e. $m < N$ for an $N \times N$ matrix

epcc

Building KS is efficient

- Note we never perform "*matrix-matrix*" multiplications
  - Only ever need "*matrix-vector*" multiplications
- Multiplying a matrix by a vector always produces another vector
- If we label the KS vectors as
$$\{\boldsymbol{v}_1, \boldsymbol{v}_2 = \boldsymbol{A}\boldsymbol{v}_1, \boldsymbol{v}_3 = \boldsymbol{A}^2\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m = \boldsymbol{A}^{m-1}\boldsymbol{v}_1\}$$
- considering the third vector for example, we note that
$$\boldsymbol{v}_3 = \boldsymbol{A}^2\boldsymbol{v}_1 = \boldsymbol{A}(\boldsymbol{A}\boldsymbol{v}_1) = \boldsymbol{A}\boldsymbol{v}_2$$
- So even though we have an $\boldsymbol{A}^2$ term in there, we never calculate $\boldsymbol{A}^2$ explicitly.
- Similarly higher powers of $\boldsymbol{A}$ are not calculated explicitly

epcc 11

11

Computing the error

- As before consider the residual
$$\boldsymbol{r} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{v}$$
- And calculate residue
$$\text{residue}=\frac{||\boldsymbol{r}||_2}{||\boldsymbol{b}||_2}$$
- *"Take a candidate solution, $\boldsymbol{v}$, apply matrix $\boldsymbol{A}$ to it, and see how close it is to $\boldsymbol{b}$."*
- If residue is 0, then $\boldsymbol{v}$ is exact solution.
- If residue is "small", then $\boldsymbol{v}$ is <u>likely to be</u> close to solution.

epcc 12

12

What is Krylov Subspace method?

- KS methods are class of iterative *search algorithms*.
- At iteration k, take existing candidate solution $\boldsymbol{v}_k$ and improve it by "minimising error" by moving in some prescribed direction $\boldsymbol{p}_k$:

$$\boldsymbol{v}_{k+1} = \boldsymbol{v}_k + \alpha_k \boldsymbol{p}_k$$

new candidate solution

search direction

old candidate solution

size of correction

epcc

13

13

Mechanics of KS method

Search algorithm ≡ minimisation problem

For example, minimising the quadratic form:

$$\phi(\boldsymbol{v}) = \frac{1}{2}\boldsymbol{v}^T \boldsymbol{A}\boldsymbol{v} - \boldsymbol{v}^T \boldsymbol{b} \quad \textit{[Quadratic form]}$$

$$= \frac{1}{2}\boldsymbol{v}.\boldsymbol{A}\boldsymbol{v} - \boldsymbol{v}.\boldsymbol{b}$$

[only true if $\boldsymbol{A}$ is symmetric, otherwise have $\boldsymbol{A}^T$ term as well!]

minimising (differentiating and making equal to zero), gives

$\boldsymbol{A}\boldsymbol{v} - \boldsymbol{b} = 0$

i.e. minimising $\phi(\boldsymbol{v})$ is equivalent to solving $\boldsymbol{A}\boldsymbol{v} = \boldsymbol{b}$

epcc

14

14

Mechanics of KS method

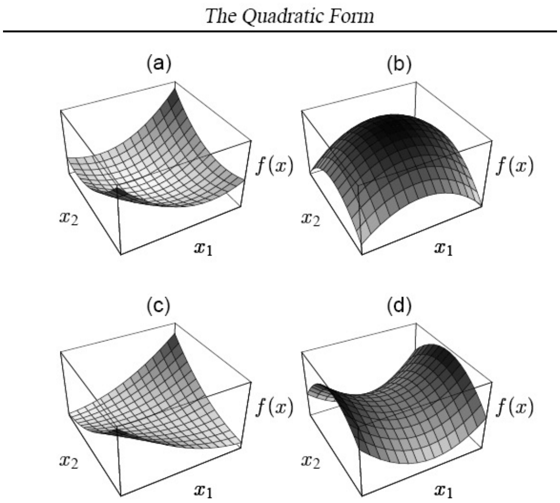
(a) Positive definite

(b) Negative definite

(c) Singular

(d) Indefinite

Choose search directions to locate the minimum of ϕ



epcc

Which direction to go in?

- Given a vector, $\boldsymbol{v}_k$, how should we choose $\boldsymbol{v}_{k+1}$
  - i.e. what should $\boldsymbol{p}_k$ and α_k be?

$$\boldsymbol{v}_{k+1} = \boldsymbol{v}_k + \alpha_k \boldsymbol{p}_k$$

- One obvious choice is to move in the direction of the negative gradient, i.e. “down the hill”
- The gradient happens to be the residual!

$$\boldsymbol{r}_k = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{v}_k$$

- Could simply choose $\boldsymbol{p}_k = \boldsymbol{r}_k$
- Put $\boldsymbol{v}_{k+1} = \boldsymbol{v}_k + \alpha_k \boldsymbol{r}_k$ into $\phi(\boldsymbol{v}_{k+1})$,

$$\phi(\boldsymbol{v}_{k+1}) = \phi(\boldsymbol{v}_k + \alpha_k \boldsymbol{r}_k)$$

- Expand and minimise (via differentiation) to find α_k
- This leads to a simple algorithm…

epcc

Steepest Descent

```
Set k=0 and choose v0
Compute r0 = b - Av0
While (k<maxiter)
  k = k+1
  αk = rk-1.rk-1 / rk-1.Ark-1
  vk = vk-1 + αkrk-1
  rk = b - Avk
  if (||rk||2 / ||b||2 < tol) break
end while
```


Orthogonality and Conjugacy

- The scalar (dot) product of two vectors is

$$\boldsymbol{u}.\boldsymbol{v} = u_1 \times v_1 + \cdots + u_N \times v_N$$

- Notice that

$$||\boldsymbol{u}||_2 = \sqrt{\boldsymbol{u}.\boldsymbol{u}}$$

- We say two vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ are orthogonal if

$$\boldsymbol{u}.\boldsymbol{v} = 0$$

- and conjugate with respect to the matrix $\boldsymbol{A}$ if

$$\boldsymbol{u}.\boldsymbol{A}\boldsymbol{v} = 0$$

Conjugate Gradient Method

- Need to generate "good spread" of search directions. Obvious choice is (method of steepest descent)

$$\boldsymbol{p}_{k+1} = \boldsymbol{r}_{k+1}$$

- Better choice is to use combination of residual and mutually conjugate directions:

$$\boldsymbol{p}_{k+1} = \boldsymbol{r}_{k+1} + \beta \boldsymbol{p}_k$$

- Scalar β chosen to give a "good spread" of conjugate search directions.
- This method is known as the Conjugate Gradient method

[search directions are mutually conjugate]

epcc

19

19

Calculation of residual

- At each CG step we have to calculate the following scalars

$$\alpha_k = \frac{\boldsymbol{r}_k.\boldsymbol{r}_k}{\boldsymbol{p}_k.\boldsymbol{A}\boldsymbol{p}_k}, \quad \beta_k = \frac{\boldsymbol{r}_{k+1}.\boldsymbol{r}_{k+1}}{\boldsymbol{r}_k.\boldsymbol{r}_k}$$

- Residual is defined as

$$\boldsymbol{r}_{k+1} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{v}_{k+1}$$

- We have already worked on $\boldsymbol{A}\boldsymbol{p}_k$ when calculating α_k
- Calculation of $\boldsymbol{A}\boldsymbol{v}_{k+1}$ implies a 2nd matrix multiplication
- Instead use the following relation for residual

$$\boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha \boldsymbol{A}\boldsymbol{p}_k$$

- So only 1 matrix multiplication needed per iteration

epcc

20

20

Desirable properties of KS method

- **Effective:** Method should minimise error (or something associated to error).
- **Bounded convergent:** Method should search solution space effectively, converging within pre-determined number of steps.
- **Efficient:** Cost of individual iteration should be small (and consistent).
- **Progressive:** Each iteration should improve the solution.

epcc 21

21

Conjugate Gradient Method - Properties

The Conjugate Gradient (CG) method:

- converges to solution of equation; effective ✓
- converges in $\leq N$ iterations; bounded convergent ✓
- requires 1 matrix-vector multiplication and 2 scalar products per iteration; efficient ✓
- improves the solution at each iteration. progressive ✓

But only for <u>symmetric, positive definite matrices</u>!

epcc 22

22

CG Algorithm

Set k=0 and choose $\mathbf{v}_0$.
Compute $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{v}_0$, set $\mathbf{p}_0 = \mathbf{r}_0$.

initial setup

While (k<maxiter)

$\alpha = \mathbf{r}_k . \mathbf{r}_k \,/\, \mathbf{p}_k . \mathbf{A}\mathbf{p}_k$
$\mathbf{v}_{k+1} = \mathbf{v}_k + \alpha \mathbf{p}_k$

minimisation: compute correction and apply

$\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha \mathbf{A}\mathbf{p}_k$
if ($||\mathbf{r}_{k+1}||_2 \,/\, ||\mathbf{b}||_2 < \mathrm{tol}$) break

test new solution -- are we close enough?

$\beta = \mathbf{r}_{k+1} . \mathbf{r}_{k+1} \,/\, \mathbf{r}_k . \mathbf{r}_k$
$\mathbf{p}_{k+1} = \mathbf{r}_{k+1} + \beta \mathbf{p}_k$

compute new search direction

k=k+1
end while

epcc

23

23

Toy example

Recall the 'apples and pears' example:

- 2 apples and 3 pears costs 40p
- 3 apples and 5 pears costs 65p.

Solution is apples cost 5p, pears cost 10p.

Associated linear system is:

$$\begin{pmatrix} 2 & 3 \\ 3 & 5 \end{pmatrix} \begin{pmatrix} a \\ p \end{pmatrix} = \begin{pmatrix} 40 \\ 65 \end{pmatrix}$$

A symmetric matrix!

epcc

24

24

CG Algorithm (step by step)

```
Set k=0 and choose v0.
Compute r0 = b - Av0, set p0=r0.
While (k<maxiter)
 α = rk.rk / pk.Apk
 vk+1 = vk + αpk
 rk+1 = rk - αApk
 if (||rk+1||2 / ||b||2 < tol) break
 β = rk+1.rk+1 / rk.rk
 pk+1 = rk+1 + βpk
 k = k+1
end while
```

initial setup

|epcc|

Initial set up

$$\begin{pmatrix} 2 & 3 \\ 3 & 5 \end{pmatrix} \begin{pmatrix} a \\ p \end{pmatrix} = \begin{pmatrix} 40 \\ 65 \end{pmatrix}$$

Initial setup:

Guess $a = 0$ and $p = 0$;

Compute residual: $\boldsymbol{r}_0 = (40{,}65) = \boldsymbol{p}_0$.

|epcc|

CG Algorithm (step by step)

```
Set k=0 and choose v0.
Compute r0 = b - Av0, set p0=r0.
While (k<maxiter)
  α = rk.rk / pk.Apk
  vk+1 = vk + αpk
  rk+1 = rk - αApk
  if (||rk+1||2 / ||b||2 < tol) break
  β = rk+1.rk+1 / rk.rk
  pk+1 = rk+1 + βpk
  k=k+1
end while
```

minimisation: compute correction and apply

epcc

Compute correction: iter 0

$\boldsymbol{r}_0 . \boldsymbol{r}_0 = 40^2 + 65^2 = 5{,}825$

Matrix-vector multiply

$$\boldsymbol{A}\boldsymbol{p}_0 = \begin{pmatrix} 2 & 3 \\ 3 & 5 \end{pmatrix} \begin{pmatrix} 40 \\ 65 \end{pmatrix} = \begin{pmatrix} 2 \times 40 + 3 \times 65 \\ 3 \times 40 + 5 \times 65 \end{pmatrix} = \begin{pmatrix} 275 \\ 445 \end{pmatrix}$$

$\boldsymbol{p}_0 . \boldsymbol{A}\boldsymbol{p}_0 = (40 \times 275) + (65 \times 445) = 39{,}925$

$\alpha = 5{,}825 \div 39{,}925 = 0.145899 \text{ (6 s.f.)}$

$\boldsymbol{v}_1 = \boldsymbol{v}_0 + (0.145899 \times \boldsymbol{p}_0) = \begin{pmatrix} 5.83594 \\ 9.48341 \end{pmatrix}$

Nearly there in just one step!

epcc

CG Algorithm (step by step)

```
Set k=0 and choose v0.
Compute r0 = b - Av0, set p0=r0.
While (k<maxiter)
  α = rk.rk / pk.Apk
  vk+1 = vk + αpk
  rk+1 = rk - αApk
  if (||rk+1||2 / ||b||2 < tol) break
  β = rk+1.rk+1 / rk.rk
  pk+1 = rk+1 + βpk
  k=k+1
end while
```

test new solution -- are we close enough?

Test solution: Iter 0

$$\boldsymbol{r_1} = \boldsymbol{r_0} - \alpha \boldsymbol{A}\boldsymbol{p}_0 = \begin{pmatrix} 40 \\ 65 \end{pmatrix} - 0.145899 \begin{pmatrix} 275 \\ 445 \end{pmatrix} = \begin{pmatrix} -0.122104 \\ 0.0751409 \end{pmatrix}$$

$||\boldsymbol{r_1}||_2 = 0.00187852$
$||\boldsymbol{b}||_2 = 5{,}825$
$\frac{||\boldsymbol{r_1}||_2}{||\boldsymbol{b}||_2} = 3.52885\ \mathrm{e}{-06}$ (very close to stopping already!)

CG Algorithm (step by step)

```
Set k=0 and choose v0.
Compute r0 = b - Av0, set p0=r0.
While (k<maxiter)
 α = rk.rk / pk.Apk
 vk+1 = vk + αpk
 rk+1 = rk - αApk
 if (||rk+1||2 / ||b||2 < tol) break
 β = rk+1.rk+1 / rk.rk
 pk+1 = rk+1 + βpk
 k=k+1
end while
```

compute new search direction

|epcc| 31

31

New search direction: iter 0

$$\beta = \boldsymbol{r}_1.\boldsymbol{r}_1 / \boldsymbol{r}_0.\boldsymbol{r}_0 = 0.00187852/5{,}825$$
$$= 3.52885\text{e-}06$$
$$\boldsymbol{p}_1 = \boldsymbol{r}_1 + \beta \boldsymbol{p}_0$$

$$\boldsymbol{p}_1 = \begin{pmatrix}-0.122104\\0.0751409\end{pmatrix} + 3.52885\text{e}{-}06\begin{pmatrix}40\\65\end{pmatrix} = \begin{pmatrix}-0.121963\\0.0753703\end{pmatrix}$$

Start next iteration

|epcc| 32

32

Iteration 1

$$\begin{pmatrix} 2 & 3 \\ 3 & 5 \end{pmatrix} \begin{pmatrix} a \\ p \end{pmatrix} = \begin{pmatrix} 40 \\ 65 \end{pmatrix}$$

$\boldsymbol{p}_1.\boldsymbol{Ap}_1 = 0.00299902$
$\alpha = 0.0205555 \div 0.00299902 = 6.85408$
$\boldsymbol{v}_2 = \boldsymbol{v}_1 + 6.85408 \times \boldsymbol{p}_1 = \begin{pmatrix} 5.000 \\ 10.000 \end{pmatrix}$
$\|\boldsymbol{r}_2\|_2 / \|\boldsymbol{b}\|_2 = 5.9692\text{e}{-20}$

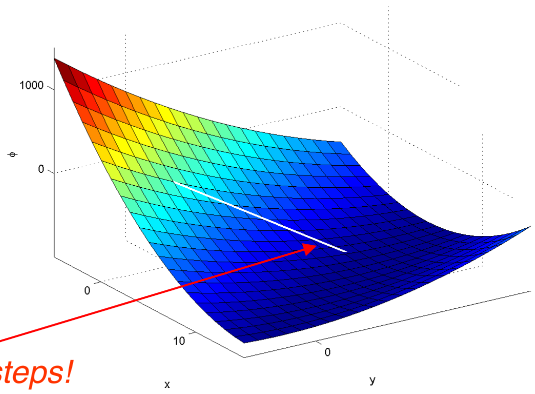
VERY CLOSE TO EXACT SOLUTION!

epcc

33

33

Graphical minimisation



Actually 2 steps!

epcc

34

34

Toy Example comments

- Convergence
  - $N = 2 \rightarrow k \leq 2$ Exact solution **after 2 iterations**
  - $\|\boldsymbol{r}\|_2$
- Precision
  - Rounding errors will produce only approximate solution
  - Single versus double precision

- Real applications
  - Use finite precision
  - Stop when residual $< \varepsilon$
  - Approximate solutions
  - $N_{iter} \ll N$

epcc

35

35

Further comments on CG

- Cost
  - involves one matrix-vector multiplication and two scalar products per iteration.
- CG does not modify the source matrix $\boldsymbol{A}$
- Requires user to provide routine for matrix-vector product
- CG actually builds a KS based first residual $(\boldsymbol{r}_0)$
  - Quite hard to see this in the algorithm!

But CG only for <u>symmetric, positive definite matrices</u>!

epcc

36

36

Other KS methods

- For more general class of matrix, cannot achieve all *desirable properties*, though can fulfil most.

- Two popular methods considered:
  - *Generalised Minimum RESidual method, **GMRES***
    - *Minimises $||\boldsymbol{b} - \boldsymbol{A}\boldsymbol{v}||_2$ via the Krylov Subspace*
  - *Bi-Conjugate Gradient method with STABilisation **(BiCGSTAB)***
    - *Variant of **Bi-Conjugate Gradient**, itself a variant of CG*
    - *Bi-Conjugate Gradient on its own is unstable*
    - *Bi-Conjugate Gradient has to consider both $\boldsymbol{A}$ and $\boldsymbol{A}^T$.*

epcc

37

37

GMRES method

The GMRES method:

- converges to solution of equation; effective ✓
- converges in $\leq N$ iterations; bounded convergent ✓
- requires expensive orthogonalisation of search directions at each iteration, -- depends on all previous iterations/search directions – computationally and memory intensive; **efficient ×**
- improves the solution at each iteration. progressive ✓

epcc

38

38

BiCGSTAB method

The BiCGSTAB method:

- may not converge to solution; **effective ×**
- convergence is unbounded; **bounded convergent ×**
- requires 2 matrix multiplications and 4 scalar products per iteration; efficient ✓
- not guaranteed to improve solution at each iteration. **progressive ×**

However, very often it works!

epcc 39

39

BiCGstab algorithm

Three scalars initialised to 1

```
Set k=0 and choose v₀.
Compute r₀=b-Av₀, l₀=r₀, q₀=p₀=0, ρ₀=α=ω₀=1.
While (k < maxiter)
  ρk+1 = l₀.rk,     β = (ρk+1/ρk) × (α/ωk)
  pk+1 = rk+β(pk- ωkqk),    qk+1 = Apk+1
  α = ρk+1 / (l₀.qk+1), s = rk- αqk+1, t = As
  ωk+1 = (t.s) / (t.t)
  vk+1 = vk+αpk+1+ωk+1s
  rk+1 = s - ωk+1t
  if ( ||rk+1||₂ / ||b||₂ < tol ) break
  k=k+1
end while
```

Appear to have 4 update vectors but l not updated so have l_0 but not l_k. 2 temporary vectors

epcc 40

40

BiCGstab algorithm

Set k=0 and choose $\mathbf{v}_0$.

Compute $\mathbf{r}_0=\mathbf{b}-\mathbf{A}\mathbf{v}_0$, $\mathbf{l}_0=\mathbf{r}_0$, $\mathbf{q}_0=\mathbf{p}_0=\mathbf{0}$, $\rho_0=\alpha=\omega_0=1$.

While (k < maxiter)

$\rho_{k+1} = \mathbf{l}_0.\mathbf{r}_k$, $\beta = (\rho_{k+1}/\rho_k) \times (\alpha/\omega_k)$

$\mathbf{p}_{k+1} = \mathbf{r}_k+\beta(\mathbf{p}_k - \omega_k\mathbf{q}_k)$, $\mathbf{q}_{k+1} = \mathbf{A}\mathbf{p}_{k+1}$

$\alpha = \rho_{k+1} / (\mathbf{l}_0.\mathbf{q}_{k+1})$, $\mathbf{s} = \mathbf{r}_k - \alpha\mathbf{q}_{k+1}$, $\mathbf{t} = \mathbf{A}\mathbf{s}$

$\omega_{k+1} = (\mathbf{t}.\mathbf{s}) / (\mathbf{t}.\mathbf{t})$

$\mathbf{v}_{k+1} = \mathbf{v}_k+\alpha\mathbf{p}_{k+1}+\omega_{k+1}\mathbf{s}$

$\mathbf{r}_{k+1} = \mathbf{s} - \omega_{k+1}\mathbf{t}$

if ($||\mathbf{r}_{k+1}||_2 / ||\mathbf{b}||_2$ < tol) break

k=k+1

end while

2 matrix-vector operations

41

41

BiCGstab algorithm

Set k=0 and choose $\mathbf{v}_0$.

Compute $\mathbf{r}_0=\mathbf{b}-\mathbf{A}\mathbf{v}_0$, $\mathbf{l}_0=\mathbf{r}_0$, $\mathbf{q}_0=\mathbf{p}_0=\mathbf{0}$, $\rho_0=\alpha=\omega_0=1$.

While (k < maxiter)

$\rho_{k+1} = \mathbf{l}_0.\mathbf{r}_k$, $\beta = (\rho_{k+1}/\rho_k) \times (\alpha/\omega_k)$

$\mathbf{p}_{k+1} = \mathbf{r}_k+\beta(\mathbf{p}_k - \omega_k\mathbf{q}_k)$, $\mathbf{q}_{k+1} = \mathbf{A}\mathbf{p}_{k+1}$

$\alpha = \rho_{k+1} / (\mathbf{l}_0.\mathbf{q}_{k+1})$, $\mathbf{s} = \mathbf{r}_k - \alpha\mathbf{q}_{k+1}$, $\mathbf{t} = \mathbf{A}\mathbf{s}$

$\omega_{k+1} = (\mathbf{t}.\mathbf{s}) / (\mathbf{t}.\mathbf{t})$

$\mathbf{v}_{k+1} = \mathbf{v}_k+\alpha\mathbf{p}_{k+1}+\omega_{k+1}\mathbf{s}$

$\mathbf{r}_{k+1} = \mathbf{s} - \omega_{k+1}\mathbf{t}$

if ($||\mathbf{r}_{k+1}||_2 / ||\mathbf{b}||_2$ < tol) break

k=k+1

end while

4 scalar products

1 vector norm

42

42

Summary of KS methods

- Looked at CG method in particular
- Other KS methods exist and are used: Steepest Descent; MINRES; GMRES with restart; Conjugate Residual; BiCG.
- Generally, follow same principles.
- In practice, call numerical library (NAG, PetSC, ARPACK)
- Still need to provide the matrix-vector multiplier!
  - Will look at this in parallel in another lecture

epcc

Conclusions

- Krylov subspace method is a form of iterative improvement. **You decide when to stop!**
- Replace linear solver with minimisation method
- For SPD matrices, standard implementation is Conjugate Gradient method.
- Otherwise, have a choice between **GMRES** and **BiCGSTAB**.
- Other methods do exist.

epcc

References

- O. Axelsson, *Iterative Solution Methods*, Cambridge 1994.
- **G.H. Golub and C.F. Van Loan, *Matrix Computations*, North Oxford Academic 1983.**
- K.W. Morton and D.F. Mayers, *Numerical Solution of Partial Differential Equations*, Cambridge 1994.
- W.H. Press, *Numerical Recipes in C: The Art of Scientific Computing*, Cambridge 1999.
- L.N. Trefethen and D. Bau, *Numerical Linear Algebra*, SIAM 1997.
- A. Greenbaum, *Iterative methods for solving linear systems*, SIAM 1997
- H.A. van der Horst, *Iterative Krylov Methods for Large Linear Systems,* Cambridge 2003
- J.R. Shewchuk. *An Introduction to the Conjugate Gradient Method Without the Agonizing Pain*
- **http://www.cs.cmu.edu/~quake-papers/painless-conjugate-gradient.pdf**