epcc

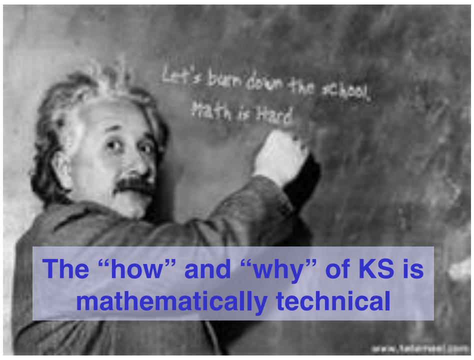
Sparse Linear Algebra: Introduction to Krylov subspace methods

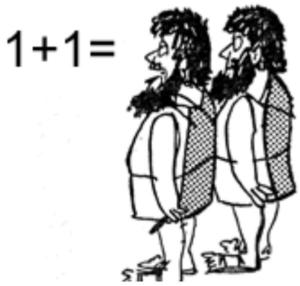
epcc

Overview

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

Mathematical Caveat



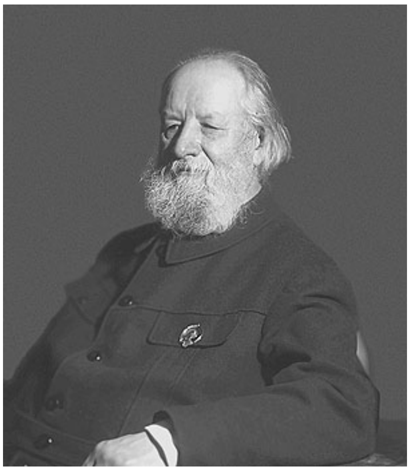


Mathematical proofs and derivations are beyond the scope of this course

Implementing KS is easy

Introduction

- 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

Recall what a linear system is

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

$$A\underline{u} = \underline{b}$$

N×**N** matrix

unknown solution to be determined (**N** components)

known righthand side (with **N** components)

- This is simply a representation of **N** equations linking **N** unknown quantities: $\mathbf{u_1}, \mathbf{u_2}, \ldots, \mathbf{u_N}$.

epcc

Krylov Subspace

- Definition: The Krylov subspace is said to be spanned by the vectors:

$$\left\{ \underline{v}, A\underline{v}, \cdots, A^{N-1}\underline{v} \right\}$$

- Krylov sub-space is a property of matrix and vector
- Repeated application of Matrix *A* to $\underline{v}$

Building KS is efficient

- Note that we never perform "matrix-matrix" operations
- Only ever need "matrix-vector" multiplications
- Multiplying a matrix by a vector always produces another vector
- If label the KS vectors as

$$\{v_1, v_2 = Av_1, v_3 = A^2v_1, \ldots, v_N = A^{N-1}v_1\}$$

- considering the third vector for example, we note that

$$v_3 = A^2v_1 = A(Av_1) = Av_2$$

- So even though we have an A^2 term in there, we never calculate it explicitly.

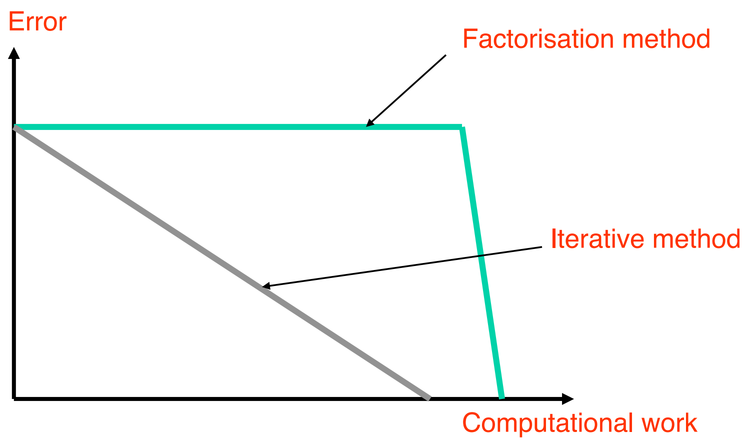
Properties of iterative schemes

- Iterative methods
  - do not modify source matrix, involve matrix only through matrix-vector multiplication (possibly with transpose).
  - 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
  - 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



Computing the error

Given a vector **v**, how close is it to solution **u**?

$$\text{error} = \underline{u} - \underline{v}$$
$$= A^{-1}\underbrace{(\underline{b} - A\underline{v})}_{\text{residual or backward error}}$$

*"Take a candidate solution, apply matrix **A** to it, and see how close it is to **b**."*

$$\text{residual} = \underline{b} - A\underline{v}$$

A norm -- size of a vector

We need a way to quantify size of a vector -- a *norm*.

A number of different versions exist. We will consider the *Euclidean* (or L_2) norm:

$$\|\underline{v}\|_2 = \sqrt{\sum_{i=1}^{N} |v_i|^2}$$

Pythagoras' Theorem!

The residual

Thus, one can quantify the residual:

$$r = \frac{\|\underline{b} - A\underline{v}\|_2}{\|\underline{b}\|_2}$$

normalising factor

Facts:

- If **r=0**, then **$\underline{v}$** is exact solution.
- If **r** is "small", then **$\underline{v}$** is <u>likely to be</u> close to solution.

What is Krylov Subspace method?

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

$$\underline{v}_{k+1} = \underline{v}_k + \alpha\, \underline{p}_k$$

new candidate solution

old candidate solution

size of correction

search direction

Mechanics of KS method

Search algorithm = minimisation problem

For example, aim to minimise:

$$\phi(\underline{v}) = \frac{1}{2}\underline{v}^T A \underline{v} - \underline{v}^T \underline{b}$$

[Quadratic form]

This is equivalent to minimising the error:

$$\left\| \underline{u} - \underline{v} \right\|$$

*[only true if **A** is symmetric!]*

Mechanics of KS method

(a) Positive definite

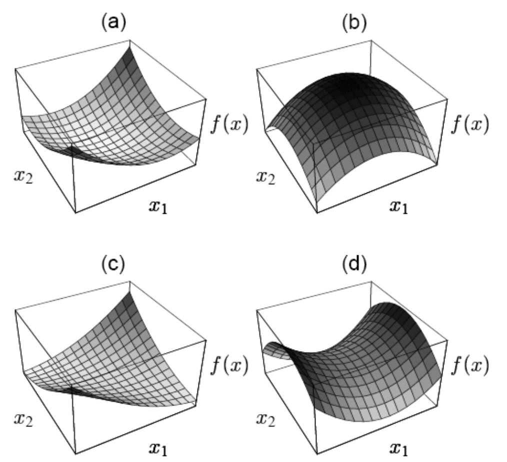
(b) Negative definite

(c) Singular

(d) Indefinite

Choose search directions to locate the minimum of ϕ

The Quadratic Form



epcc

Mechanics of KS method

Need to solve minimisation problem (for new search direction). Choice of

$$\alpha = \frac{\overbrace{\underline{r}_k . \underline{r}_k}^{\text{scalar product}}}{\overbrace{\underline{p}_k . \underbrace{A\underline{p}_k}_{\text{matrix multiplication}}}}$$

residual

will minimise:

$$\phi(\underline{v}) \quad \text{where} \quad \underline{v} = \underline{v}_k + \alpha \underline{p}_k$$

New search direction

epcc

Reminder: scalar product

The scalar (dot) product of two vectors is:

$$\underline{u}.\underline{v} = u_1 \times v_1 + \ldots + u_N \times v_N$$

Notice that: $\|\underline{u}\|_2 = \sqrt{\underline{u}.\underline{u}}$

We say two vectors ***u*** and ***v*** are orthogonal if

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

and conjugate with respect to the matrix ***A*** if

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

Mechanics of KS method

Need to generate "good spread" of search directions. Obvious choice is:

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

Better choice is:

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

Scalar **β** chosen to give a "good spread" of conjugate search directions.

[search directions are mutually conjugate]

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.

Conjugate Gradient Method

The Conjugate Gradient (CG) method:

- converges to solution of equation; effective ✓
- converges in **<= 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>!

CG Algorithm

Set **k=0** and choose $\underline{v}_0$.
Compute $\underline{r}_0 = \underline{b} - A\underline{v}_0$, set $\underline{p}_0 = \underline{r}_0$.
While (**k<maxiter**)

$\quad \alpha = \underline{r}_k \cdot \underline{r}_k / \underline{p}_k \cdot A\underline{p}_k$
$\quad \underline{v}_{k+1} = \underline{v}_k + \alpha\underline{p}_k$
$\quad \underline{r}_{k+1} = \underline{r}_k - \alpha A\underline{p}_k$
$\quad$ if ($\|\underline{r}_{k+1}\|_2 / \|\underline{b}\|_2 <$ tol) **break**
$\quad \beta = \underline{r}_{k+1} \cdot \underline{r}_{k+1} / \underline{r}_k \cdot \underline{r}_k$
$\quad \underline{p}_{k+1} = \underline{r}_{k+1} + \beta\underline{p}_k$
$\quad$ **k=k+1**

end while

initial setup

minimisation: compute correction and apply

test new solution -- are we close enough?

compute new search direction

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!

CG Algorithm (step by step)

$\left.\begin{array}{l}\text{Set } \mathbf{k=0} \text{ and choose } \underline{\mathbf{v}}_0. \\ \text{Compute } \underline{\mathbf{r}}_0\mathbf{=}\underline{\mathbf{b}}\mathbf{-A}\underline{\mathbf{v}}_0, \text{ set } \underline{\mathbf{p}}_0\mathbf{=}\underline{\mathbf{r}}_0.\end{array}\right\} \leftarrow$ initial setup

While ($\mathbf{k<maxiter}$)

$\alpha = \underline{r}_k \cdot \underline{r}_k \,/\, \underline{p}_k \cdot A\underline{p}_k$

$\underline{v}_{k+1} = \underline{v}_k + \alpha\underline{p}_k$

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

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

$\beta = \underline{r}_{k+1} \cdot \underline{r}_{k+1} \,/\, \underline{r}_k \cdot \underline{r}_k$

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

$\mathbf{k=k+1}$

end while

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: $\underline{\mathbf{r}}_0 = (40,65) = \underline{\mathbf{p}}_0$.

CG Algorithm (step by step)

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

Compute $\underline{r}_0 = \underline{b} - A\underline{v}_0$, set $\underline{p}_0 = \underline{r}_0$.

While (**k<maxiter**)

$\alpha = \underline{r}_k \cdot \underline{r}_k \,/\, \underline{p}_k \cdot A\underline{p}_k$ — minimisation: compute correction and apply

$\underline{v}_{k+1} = \underline{v}_k + \alpha \underline{p}_k$

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

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

$\beta = \underline{r}_{k+1} \cdot \underline{r}_{k+1} \,/\, \underline{r}_k \cdot \underline{r}_k$

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

k=k+1

end while

Compute correction: iter 0

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

Matrix-vector multiply

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

$\underline{p}_0 . \mathbf{A}\underline{p}_0 = 40\times275 + 65\times445 = 39{,}925$

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

$\underline{v}_1 = \underline{v}_0 + 0.145899\times\underline{p}_0 = (5.83594\ ,\ 9.48341)$

Nearly there in one step!

CG Algorithm (step by step)

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

Compute $\underline{r}_0 = \underline{b} - \mathbf{A}\underline{v}_0$, set $\underline{p}_0 = \underline{r}_0$.

While ($\mathbf{k<maxiter}$)

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

end while

test new solution -- are we close enough?

Test solution: Iter 0

$$\underline{\mathbf{r}}_1 = \underline{\mathbf{r}}_0 - \alpha \mathbf{A}\underline{\mathbf{p}}_0 = \begin{pmatrix} 40 \\ 65 \end{pmatrix} - 0.145899 \begin{pmatrix} 275 \\ 445 \end{pmatrix} = \begin{pmatrix} -0.122104 \\ 0.0751409 \end{pmatrix}$$

$\|\underline{\mathbf{r}}_1\|_2 = 0.00187852$

$\|\underline{\mathbf{b}}\|_2 = 5825$

$\|\underline{\mathbf{r}}_1\|_2 / \|\underline{\mathbf{b}}\|_2 = 3.52885\text{e-}06$ *(very close to stopping!)*

CG Algorithm (step by step)

Set **k**=0 and choose $\underline{\mathbf{v}}_0$.
Compute $\underline{\mathbf{r}}_0 = \underline{\mathbf{b}} - \mathbf{A}\underline{\mathbf{v}}_0$, set $\underline{\mathbf{p}}_0 = \underline{\mathbf{r}}_0$.
While (**k<maxiter**)

- $\alpha = \underline{\mathbf{r}}_k \cdot \underline{\mathbf{r}}_k / \underline{\mathbf{p}}_k \cdot \mathbf{A}\underline{\mathbf{p}}_k$
- $\underline{\mathbf{v}}_{k+1} = \underline{\mathbf{v}}_k + \alpha \underline{\mathbf{p}}_k$
- $\underline{\mathbf{r}}_{k+1} = \underline{\mathbf{r}}_k - \alpha \mathbf{A}\underline{\mathbf{p}}_k$
- if ($\|\underline{\mathbf{r}}_{k+1}\|_2 / \|\underline{\mathbf{b}}\|_2 <$ **tol**) **break**
- $\beta = \underline{\mathbf{r}}_{k+1} \cdot \underline{\mathbf{r}}_{k+1} / \underline{\mathbf{r}}_k \cdot \underline{\mathbf{r}}_k$
- $\underline{\mathbf{p}}_{k+1} = \underline{\mathbf{r}}_{k+1} + \beta \underline{\mathbf{p}}_k$
- **k=k+1**

end while

compute new search direction

New search direction: iter 0

$\beta = \underline{r}_1 . \underline{r}_1 / \underline{r}_0 . \underline{r}_0 = 0.00187852/5825$

$= 3.52885e\text{-}06$

$\underline{p}_1 = \underline{r}_1 + \beta \underline{p}_0$

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

Start next iteration

Iteration 1

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

$\underline{p}_1 . A \underline{p}_1 = 0.00299902$

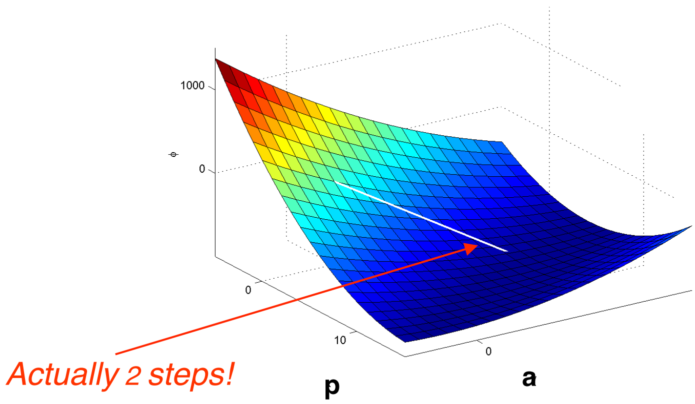
$\alpha = 0.0205555 \div 0.00299902 = 6.85408$

$\underline{v}_2 = \underline{v}_1 + 6.85408 \times \underline{p}_1 = (5.000 , 10.000)$

$\|\underline{r}_2\|_2 / \|\underline{b}\|_2 = 5.9692e\text{-}20$

EXACT SOLUTION!

Graphical minimisation



Toy Example comments

- Convergence
  - N=2 → k≤2 Exact solution after 2 iterations
  - $||r^2||$ is very small
- Precision
  - Rounding errors will produce only approximate solution
  - Single versus double precision

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

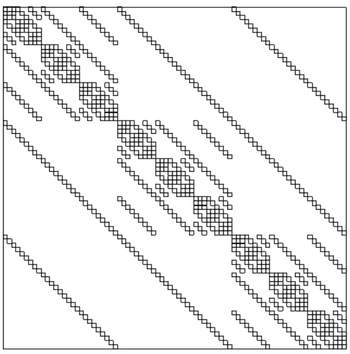
Further comments

- Cost
  - involves one matrix-vector multiplication and two scalar products per iteration.
- does not modify the source matrix ***A***
- requires user to provide routine for matrix-vector product

But only for symmetric, positive definite matrices!

A Real Example

- Particle physics example
  - Quantum Chromodynamics (QCD)
- Matrix is highly structured
  - $N \sim O(10^3)$ →
    - SM Pickles PhD thesis UoE 1998
- Real calculation $N > O(10^7)$
- CG converge $\leq$ N iters
  - 4096 cores of HECToR
  - $O(10^5)$ iters takes 30 mins
  - Calculation done in double-prec appropriate finite residual 10^{-8}



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***
  - *Bi-Conjugate Gradient method with STABilisation, **BiCGSTAB***

GMRES method

The GMRES method:

- converges to solution of equation; effective ✓
- converges in <= **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 ✓

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!

BiCGstab algorithm

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

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

While **(k < maxiter)**

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

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

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

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

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

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

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

k=k+1

end while

Three scalars initialised to 1

4 update vectors
2 temporary vectors

$\underline{l}$ is not updated (*i.e.* $\underline{l}_0$ not $\underline{l}_k$)

BiCGstab algorithm

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

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

While **(k < maxiter)**

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

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

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

$\omega_{k+1}=(\underline{t}\cdot\underline{s})/(\underline{t}\cdot\underline{t})$

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

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

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

k=k+1

end while

2 matrix-vector operations

BiCGstab algorithm

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

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

While **(k < maxiter)**

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

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

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

$\omega_{k+1}=\boxed{(\underline{t}\cdot\underline{s})}/\boxed{(\underline{t}\cdot\underline{t})}$

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

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

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

k=k+1

end while

4 scalar products

1 vector norm

Summary of KS methods

- 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!

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