CSR format with empty row

$$\boldsymbol{A} = \begin{pmatrix} 1.0 & 0.1 & 0.5 & \cdot \\ 0.4 & \cdot & \cdot & 0.7 \\ \cdot & \cdot & \cdot & \cdot \\ 0.3 & \cdot & 0.2 & 0.8 \end{pmatrix}$$

```
nRow = 4; nCol = 4; nzMax = 8;
value[8] = {1.0, 0.1, 0.5, 0.4, 0.7, 0.3,
0.2, 0.8};
colIdx[8] = {0, 1, 2, 0, 3, 0, 2, 3};
rowStart[5] = {0, 3, 5, 5, 8};
```

epcc

1

Numerical Algorithms for HPC

KS methods in parallel, Preconditioning, matrix splitting and KS methods in action

epcc

2

Outline

- KS method in parallel
  - Matrix-vector multiplication
- Preconditioning
  - Basic concept
  - Matrix splitting
- CG and BiCGStab in action

3

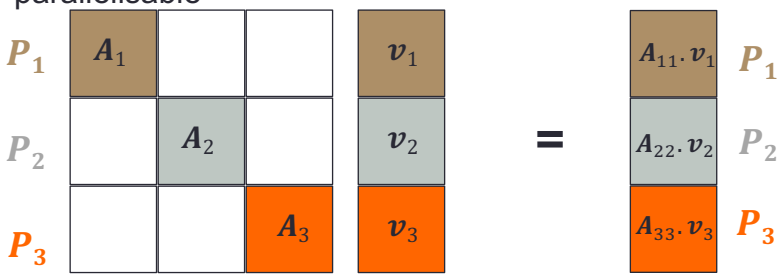
Matrix vector multiplication in parallel

- Krylov Subspace methods involve one or more matrix-vector multiplications at each iteration
- The matrix-vector multiplication is the most computationally expensive part of most KS methods
- It makes sense to carry this out in parallel where possible
- We first look at trivially parallel case of block-diagonal matrix
- Then look at more general case for
  - Replicated vector
  - Distributed vector

4

Parallel matrix vector – block diagonal matrix

- How to decompose matrix across parallel machine?
  - Depends on how the matrix is stored
  - Depends on what the matrix is
- Block diagonal matrix-vector multiply routine is completely parallelisable



- Could further parallelise via threads within blocks

epcc

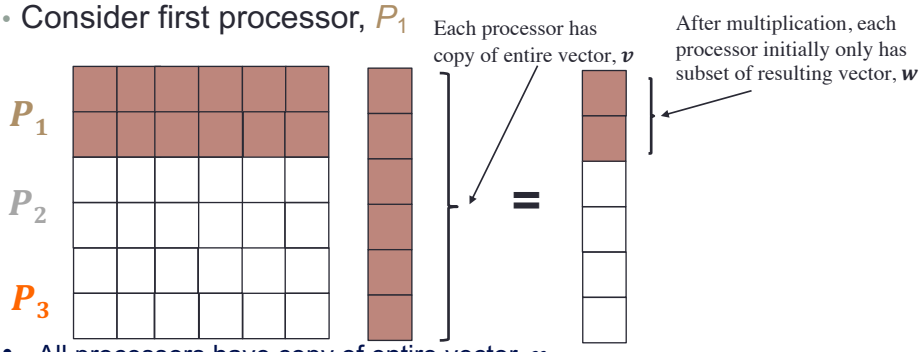
General parallel matrix-vector

- Multiplication of $\boldsymbol{Av} = \boldsymbol{w}$
- General approach is 1D decomposition of matrix $\boldsymbol{A}$ by row
  - each processor has $m = Nrows/Nprocs$ rows of matrix
- Then have a choice as to whether the vector $\boldsymbol{v}$ is
  - Replicated across all processors – all processors can see all the entire vector at each iteration
  - Distributed across processors – processors only see subset of data at any one time

epcc

Parallel matrix-vector – Replicated data

- Consider first processor, P_1



- All processors have copy of entire vector, v
- Need to broadcast data at end of each iteration to complete the vector w
- Vector w then replicated across procs and ready for next iteration
- Simple to implement with no communication during calculation
  - but high memory requirement and requires broadcast at the end

Distributed vector: 2 processor 2×2 example

Both matrix **and** vector are distributed:

$$\begin{matrix} P_1 \\ P_2 \end{matrix} \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} = \begin{pmatrix} a_{11}b_1 + a_{12}\textcircled{b_2} \\ a_{21}\textcircled{b_1} + a_{22}b_2 \end{pmatrix}$$

- Circled elements not initially available to relevant processor
- Calculate $a_{11}\,b_1$ on P_1 and $a_{22}\,b_2$ on P_2
- Then swap elements b_1 and b_2 between processors
- Then calculate $a_{12}\,b_2$ on P_1 and $a_{21}b_1$ on P_2
- Can generalise to more processors
  - e.g. pass around a ring rather than simply "swap" elements
- Can also generalise to bigger matrices
  - Then deal with blocks of rows instead
- No need to gather at the end - vectors remain distributed!

Parallel matrix-vector – Distributed data

- Distributed vector
  - more complicated than replicated and more communication during calculations
  - but more efficient use of memory for storing both matrix **and** vectors
  - answer ends up distributed as before but can stay distributed!
    - No need for final broadcast of data
  - locally accumulate matrix times vector block (i.e. partial dot products)
  - processors must exchange vector blocks to complete dot products
  - Sparse: may not need all processors to exchange blocks
- Smart implementations for specific matrix types
- Global sum still required if scalar products needed (e.g. residues)

epcc

Preconditioning: Overview

- Motivation
- What is preconditioning?
- What is its purpose?
- Common preconditioners

epcc

What is preconditioning?

"a preconditioner is any form of implicit or explicit modification of an original linear system which makes it "easier" to solve by a given iterative method"

Y Saad, Iterative methods for Sparse Linear Systems

- Examples of preconditioners
  - Scaling all rows so that diagonal entries are equal to 1 (Jacobi preconditioner)
    - See block Jacobi a few slides later…
  - Pre-multiplying the matrix by a given matrix, e.g. $\boldsymbol{A} \rightarrow \boldsymbol{M}^{-1}\boldsymbol{A}$
    - Unlikely $\boldsymbol{M}$ or $\boldsymbol{M}^{-1}\boldsymbol{A}$ ever computed directly
    - $\boldsymbol{M}^{-1}$ may be complicated. E.g. result of some FFT transformations or integral calculations

What is preconditioning? cont ...

- Two extremal cases:
  - Choice $\boldsymbol{M} = \boldsymbol{I}$ is equivalent to no preconditioning
  - choice $\boldsymbol{M} = \boldsymbol{A}$ is equivalent to factorising the problem directly
- We seek an intermediate $\boldsymbol{M}$ which ensures that KS method will converge and reduces (minimises) the cost of solver.
- Idea to preserve structure of $\boldsymbol{A}$ (particularly symmetry!)

Motivation

- We would like Krylov subspace method
  - to converge (smoothly)
  - to converge in as few iterations as possible
    - reduce the effects of rounding error
    - make method more tractable for multiple righthand sides
- Number of iterations is affected by condition number, *C*, of the matrix

  $C = \frac{\lambda_{\max}}{\lambda_{\min}}$, where $\lambda_{\max}$ and $\lambda_{\min}$ are the maximum and minimum eigenvalues of the matrix respectively
  - a lower condition number implies fewer KS iterations
- Choose preconditioner which reduces the condition number

Preconditioning in linear systems

- Solving system

$$\boldsymbol{Au} = \boldsymbol{b}$$

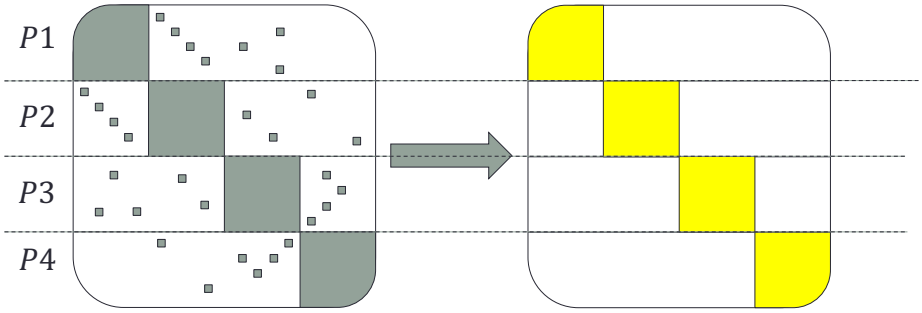
- is equivalent to solving

$$\boldsymbol{M}^{-1}\boldsymbol{Au} = \boldsymbol{M}^{-1}\boldsymbol{b}$$

- where $\boldsymbol{M}$ is a SPD matrix (for CG, at least).
- Idea is to choose $\boldsymbol{M}$ such that similar to $\boldsymbol{A}$ but easier to invert.
- Jacobi/Gauss-Seidel can be thought of as pre-conditioner (see matrix splitting later)

Block Jacobi preconditioning

- Ignore entries that are not in block diagonals
- Invert blocks locally (often incomplete inverse)

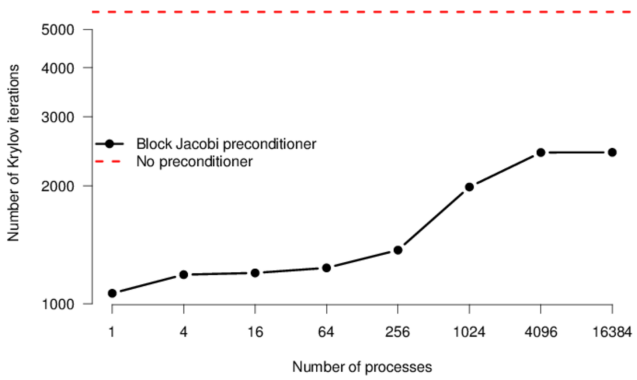


- Blocks get smaller with more processors so less is inverted

epcc

More processors give worse approximation

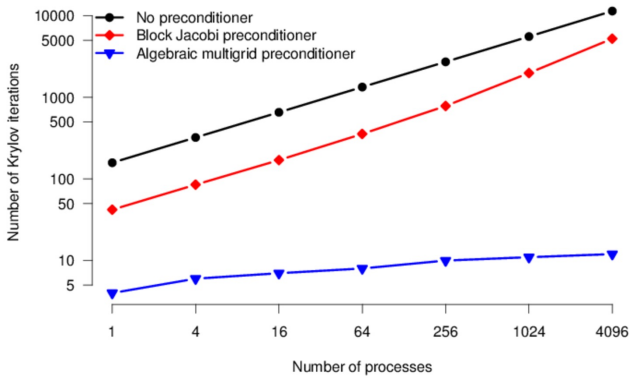
- Throws away more information



epcc

Other preconditioners exist

- Multigrid (algorithmically optimal)
  - computationally more difficult and more expensive



epcc

Incomplete LU factorisation

- A more powerful/expensive preconditioner is ILU factorisation
- Elements are computed as in LU factorisation, but those that fall outwith the sparsity pattern are discarded

$$\boldsymbol{A = LU - R}$$

- $M = LU$ can then be used as the preconditioner
- This preserves sparsity pattern
- No guarantee of existence of non-singular ILU factors.
- Many modifications exist

epcc

Preconditioned CG

- Remember, we were originally solving the linear system
$$\boldsymbol{Av} = \boldsymbol{b}$$
- With the preconditioner, we are now solving the equally valid linear system
$$\boldsymbol{M}^{-1}\boldsymbol{Av} = \boldsymbol{M}^{-1}\boldsymbol{b}$$
- Remember the original definition of the residual
$$\boldsymbol{r} = \boldsymbol{b} - \boldsymbol{Av}$$
which would become, with preconditioning
$$\boldsymbol{M}^{-1}\boldsymbol{r} = \boldsymbol{M}^{-1}(\boldsymbol{b} - \boldsymbol{Av})$$
- We can introduce a preconditioned analogue, $\boldsymbol{s}$:
$$\boldsymbol{s} = \boldsymbol{M}^{-1}\boldsymbol{r} \Longrightarrow \boldsymbol{Ms} = \boldsymbol{r}$$

Preconditioned CG algorithm

Choose $\mathbf{v}_0$, compute $\mathbf{r}_0=\mathbf{b}-\mathbf{Av}_0$, k=0,
Solve $\mathbf{Ms}_0=\mathbf{r}_0$ (using a direct method), $\mathbf{p}_0=\mathbf{s}_0$
While (k<maxiter)
 $\alpha = \mathbf{r}_k.\mathbf{s}_k \;/\; \mathbf{p}_k.\mathbf{Ap}_k$
 $\mathbf{v}_{k+1} = \mathbf{v}_k + \alpha\mathbf{p}_k$
 $\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha\mathbf{Ap}_k$
 if ($||\mathbf{r}_{k+1}||_2 \;/\; ||\mathbf{b}||_2 <$ tol) break
 Solve $\mathbf{Ms}_{k+1}=\mathbf{r}_{k+1}$
 $\beta = \mathbf{r}_{k+1}.\mathbf{s}_{k+1} \;/\; \mathbf{r}_k.\mathbf{s}_k$
 $\mathbf{p}_{k+1} = \mathbf{s}_{k+1} + \beta\mathbf{p}_k$
 k=k+1
end while

Matrix Splitting

- Returning to Jacobi and Gauss Seidel…
- Any linear problem is of the form $\boldsymbol{A}\,\boldsymbol{x} = \boldsymbol{b}$
  - $\boldsymbol{A}$ encodes the precise form of the PDE
  - $\boldsymbol{b}$ contains any fixed boundary conditions
- Could split $\boldsymbol{A}$ into three parts
  - Diagonal, Strictly Upper and Strictly Lower triangular: $\boldsymbol{A} = \boldsymbol{L} + \boldsymbol{D} + \boldsymbol{U}$
  - *not* the same as the LU factors!

$$(\boldsymbol{L} + \boldsymbol{D} + \boldsymbol{U})\boldsymbol{x} = \boldsymbol{b}, \qquad \boldsymbol{D}\boldsymbol{x} = -(\boldsymbol{L} + \boldsymbol{U})\boldsymbol{x} + \boldsymbol{b}$$

  - view these as iterative expressions, e.g. Jacobi corresponds to

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

Consider 1D Pollution Problem

- $\boldsymbol{A}$ represents: $-\frac{d^2}{dx^2}$
- Splitting into $\boldsymbol{L}, \boldsymbol{D}$ and $\boldsymbol{U}$
  - Jacobi iteration is actually given by: $\boldsymbol{D}\boldsymbol{x}^{(n+1)} = -(\boldsymbol{L} + \boldsymbol{U})\boldsymbol{x}^{(n)} + \boldsymbol{b}$

$$A = \begin{bmatrix} 2 & -1 & \cdot & \cdot \\ -1 & 2 & -1 & \cdot \\ \cdot & -1 & 2 & -1 \\ \cdot & \cdot & -1 & 2 \end{bmatrix}$$

$$\begin{bmatrix} 2 & \cdot & \cdot & \cdot \\ \cdot & 2 & \cdot & \cdot \\ \cdot & \cdot & 2 & \cdot \\ \cdot & \cdot & \cdot & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}^{(n+1)} = \begin{bmatrix} \cdot & 1 & \cdot & \cdot \\ 1 & \cdot & 1 & \cdot \\ \cdot & 1 & \cdot & 1 \\ \cdot & \cdot & 1 & \cdot \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}^{(n)} + \begin{bmatrix} b_1 \\ 0 \\ 0 \\ b_4 \end{bmatrix}$$

Jacobi Equations

- Equations the same as in previous lectures
  - with u_i replaced by x_i
  - exterior boundary values u_0 and u_{N+1} replaced by b_1 and b_N

$$
\begin{aligned}
x_1^{(n+1)} &= \tfrac{1}{2}\,(\,b_1 \;+\; x_2^{(n)}\,)\\
x_2^{(n+1)} &= \tfrac{1}{2}\,(\,x_1^{(n)} + x_3^{(n)}\,)\\
x_3^{(n+1)} &= \tfrac{1}{2}\,(\,x_2^{(n)} + x_4^{(n)}\,)\\
x_4^{(n+1)} &= \tfrac{1}{2}\,(\,x_3^{(n)} + b_4\,)
\end{aligned}
$$

- Procedure
  - impose PDE at each interior point
  - new value is the average of the old neighbouring points

Gauss Seidel

- Keep both $\boldsymbol{D}$ and $\boldsymbol{L}$ on the LHS: $(\boldsymbol{D}+\boldsymbol{L})\boldsymbol{x}^{(n+1)} = -\boldsymbol{U}\,\boldsymbol{x}^{(n)} + \boldsymbol{b}$

$$
\begin{bmatrix} 2 & \cdot & \cdot & \cdot \\ -1 & 2 & \cdot & \cdot \\ \cdot & -1 & 2 & \cdot \\ \cdot & \cdot & -1 & 2 \end{bmatrix}
\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}^{(n+1)}
=
\begin{bmatrix} \cdot & 1 & \cdot & \cdot \\ \cdot & \cdot & 1 & \cdot \\ \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot \end{bmatrix}
\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}^{(n)}
+
\begin{bmatrix} b_1 \\ 0 \\ 0 \\ b_4 \end{bmatrix}
$$

$$
\begin{aligned}
x_1^{(n+1)} &= \tfrac{1}{2}\,(\,b_1 \;+\; x_2^{(n)}\,)\\
x_2^{(n+1)} &= \tfrac{1}{2}\,(\,x_1^{(n+1)} + x_3^{(n)}\,)\\
x_3^{(n+1)} &= \tfrac{1}{2}\,(\,x_2^{(n+1)} + x_4^{(n)}\,)\\
x_4^{(n+1)} &= \tfrac{1}{2}\,(x_3^{(n+1)} + b_4\,)
\end{aligned}
$$

  - equivalent to solving Jacobi equations *in-place* in order $1, 2, \ldots, N$

Jacobi and (over-relaxed) Gauss-Seidel

- Connection to matrix-splitting
  - Jacobi

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

  - Gauss-Seidel

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

  - Over-relaxed Gauss-Seidel

$$\boldsymbol{x}^{(n+1)} = (\boldsymbol{D}+\omega\boldsymbol{L})^{-1}\left(\left((1-\omega)\boldsymbol{D}-\omega\boldsymbol{U}\right)x^{(n)} + \omega\boldsymbol{b}\right)$$

  - All have the form

$$\boldsymbol{x}^{(n+1)} = \boldsymbol{E}^{-1}\boldsymbol{F}\boldsymbol{x}^{(n)} + \boldsymbol{E}^{-1}\boldsymbol{b}$$
$$\boldsymbol{E}\boldsymbol{x}^{(n+1)} = \boldsymbol{F}\boldsymbol{x}^{(n)} + \boldsymbol{b}$$

epcc

Note: Convergence of Splitting

- General matrix splitting eqns: $\boldsymbol{E}\boldsymbol{x}^{(n+1)} = \boldsymbol{F}\boldsymbol{x}^{(n)} + \boldsymbol{b}$
  - solution at iteration n is perfect solution $\hat{\boldsymbol{x}}$ plus correction $\boldsymbol{\delta x}^{(n)}$
    $\boldsymbol{x}^{(n)} = \hat{\boldsymbol{x}} + \boldsymbol{\delta x}^{(n)}$ where $\boldsymbol{E}\hat{\boldsymbol{x}} = \boldsymbol{F}\hat{\boldsymbol{x}} + \boldsymbol{b}$
  - substituting into main equation gives $\boldsymbol{E}\boldsymbol{\delta x}^{(n+1)} = \boldsymbol{F}\boldsymbol{\delta x}^{(n)}$
  - error in solution evolves according to $\boldsymbol{\delta x}^{(n)} = (\boldsymbol{E}^{-1}\boldsymbol{F})^n\boldsymbol{\delta x}^{(0)}$
- Convergence depends on eigenvalues of $\boldsymbol{E}^{-1}\boldsymbol{F}$
  - must all be less than one in order to get a solution
  - speed of convergence depends on condition number
- Iteration matrix
  - $\boldsymbol{E}^{-1}\boldsymbol{F}$ is $-\boldsymbol{D}^{-1}(\boldsymbol{L}+\boldsymbol{U})$ for Jacobi and $-(\boldsymbol{D}+\boldsymbol{L})^{-1}\boldsymbol{U}$ for Gauss Seidel
  - can show that latter is better conditioned
  - heuristically, GS inverts more of the matrix at each step

epcc

CG and BiCGstab in action

- Next few slides look at KS in action for pollution problem
- Consider asymmetry (wind)
- Look at convergence

epcc

Asymmetry - wind

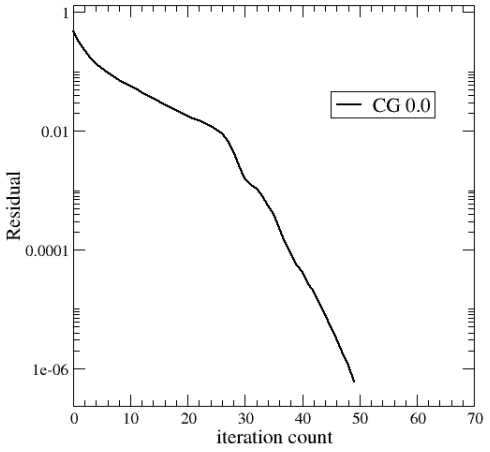
- With wind the matrix A is
  - not symmetric
  - positive definite
- CG properties no longer guaranteed
- Gentle breeze,
  - CG still works
  - Takes longer to converge
- Increase wind until CG breaks down
  - What happens to the norm of the residual?
- Implement BiCGstab
  - Play with a hurricane!

epcc

A still day

Wind = F (1.0 x + 0.5 y)

F = 0.0



epcc

29

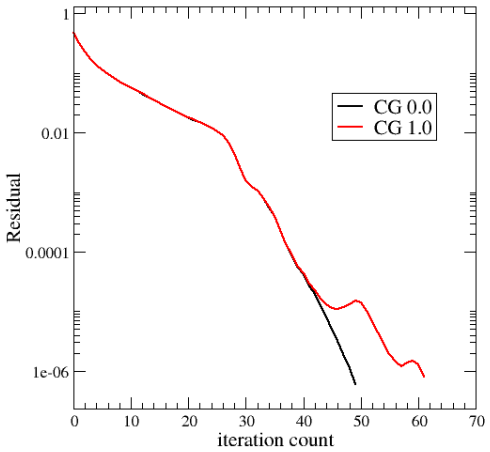
A gentle breeze

Wind = F (1.0 x + 0.5 y)

F = 1.0

Change in residual no longer monotonic

Algorithm is no longer progressive



epcc

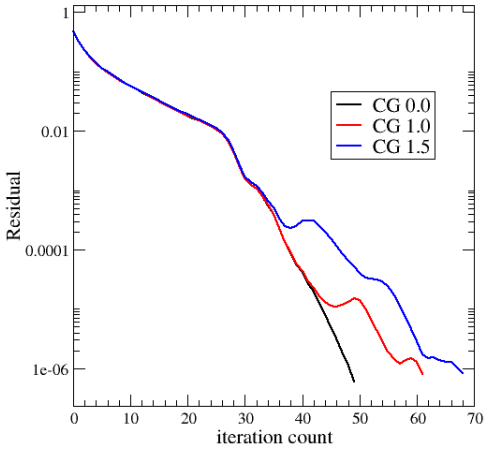
30

A moderate breeze

Wind = F (1.0 x + 0.5 y)

F = 1.5

Number of iterations required to reach residual increases as wind strength increases



epcc

31

A stiff wind

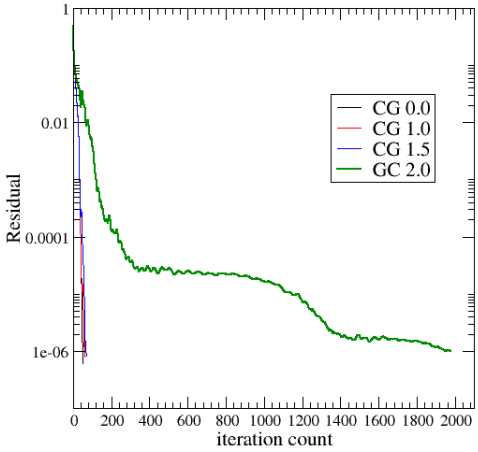
Wind = F (1.0 x + 0.5 y)

F = 2.0

Matrix further from SPD

CG is starting to break down

Now takes thousands of iterations



epcc

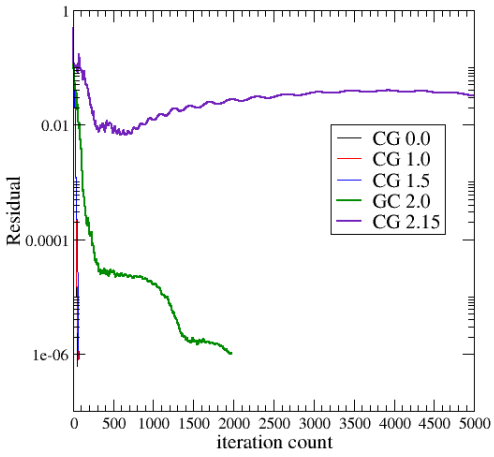
32

Slightly stronger wind

Wind = F (1.0 x + 0.5 y)

F = 2.15

CG fails to converge!



epcc

33

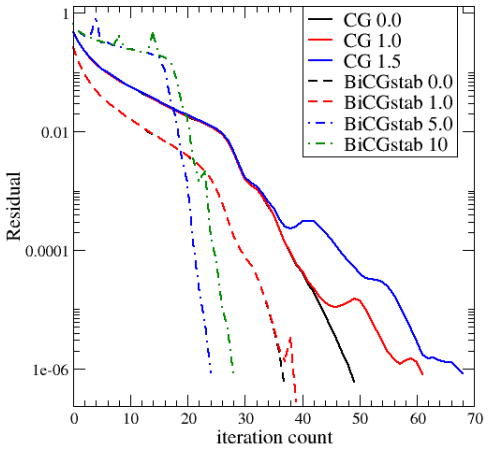
BiCGstab

Wind = F (1.0 x + 0.5 y)

BiCGstab converges in few iterations for even large winds

Large winds off diagonal terms dominate

BiCGstab does twice as much work per iteration as CG



epcc

34

Conclusions

- Matrix-vector multiplication in parallel with vectors stored as
  - replicated data
  - distributed data
- Preconditioning of linear system can improve reliability of KS method, reduce iteration count and computational costs
- Simple preconditioners based on stationary splitting methods
- More complex methods such as ILU, or preconditioning using Fourier transforms are more effective but may not work at all.
- Also looked at CG and BiCGstab in action

Remaining exercises

- Pollution model
  - Write a method that converts a COO-formatted matrix into a CSR-formatted matrix
  - Write a method that performs a matrix-vector multiplication for a CSR-formatted matrix.
  - Implement CG
- Matrix-vector in parallel – no practical session for this
  - Take serial “power method” eigensolver and parallelise the matrix-vector routine for both replicated data and distributed data