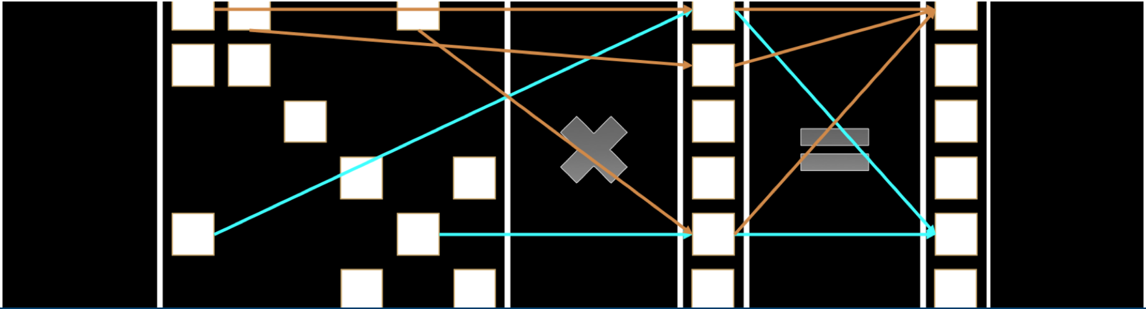


High-Performance Matrix Computations

Sparse Matrix Applications: CG & PageRank

January 26, 2022 | Xinzhe Wu (xin.wu@fz-juelich.de) | Jülich Supercomputing Centre

RWTH AACHEN UNIVERSITY

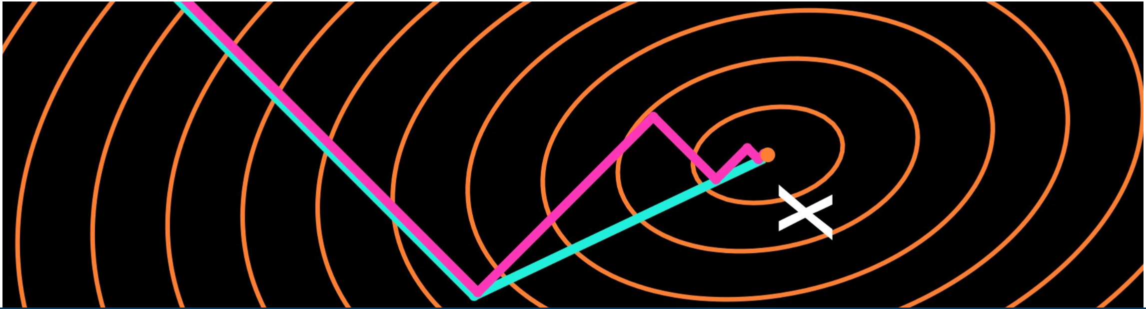
JÜLICH Forschungszentrum | JÜLICH SUPERCOMPUTING CENTRE

Organisation

Topics: High-Performance Computations of Sparse Matrices

- Module 1 (Jan. 24): Sparse Matrix Representations and Computations
- Module 2 (Jan. 26): Applications of Sparse Matrix:
  - Iterative linear solver: Conjugate Gradient method (CG)
  - Graph analytics: PageRank algorithm to rank webpages

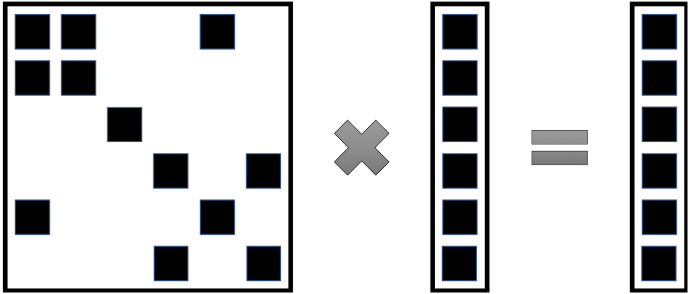
- Lectures based on slides
- Practical examples and exercises
  1. Module 1: C codes on Laptop and CLAIX
     - numerical kernel implementation
     - calling of high-performance libraries for sparse matrices
     - testing and benchmarking
  2. Module 2: Jupyter notebooks with Julia on Laptop
     - Questions in sequence during the execution of Jupyter notebooks



Part I: Conjugate Gradient Method

Sparse Linear Solvers

- Solve sparse linear system ($Ax = b$) in which A is a sparse matrix
- Variety of direct and iterative methods



Three classes of linear solvers

The methods to solve linear system $Ax = b$, with $A \in \mathcal{R}^{n \times n}$ can be split into thee classes

- dense direct solver
  - factor-solve method
  - runtime depends on size; independent of A and b, and structure of A
  - work well for n up to 10^4
- sparse direct solver
  - factor-solve based
  - runtime depends on size, sparsity pattern of A; (almost) independent of data
  - can work well for n up to 10^5 (or more).
  - requires good heuristic for ordering
- indirect (iterative methods)
  - runtime depends on data (A and b), size, sparsity, desired accuracy
  - requires tuning, preconditioning, $\cdots$
  - good choice in many cases; only chose for $n = 10^6$ or larger

Direct solvers vs Iterative solvers

Direct Solver

- Robust
- Black-box operation
- Difficult to parallelize
- Memory consumption
- Limited scalability

Iterative Solver

- Breakdown issues
- lots of parameters
- easy to parallelize
- low memory footprint
- scalable

Some Iterative Solvers

To solve $Ax = b$ with splitting $A = L + D + U$, with a iterate such that $x_{t+1} = Gx_t + f$, it converges only with the spectrum radius $\rho(G) < 1$.

- **Jacobi method**: $x_{i+1} = -D^{-1}(L + U)x_t + D^{-1}b$
- **Gauss-Seidel method**: $x_{i+1} = -(D + L)^{-1}Ux_t + (D + L)^{-1}b$
- **Successive over-relaxation (SSOR):** $x_{i+1} = (D + \omega L)^{-1}[(1 - \omega)D - \omega U]x_t + (D + \omega L)^{-1}(D + L)^{-1}\omega b$

Krylov Subspace Methods: CG, GMRES, BiCGstab $\cdots$

$$\mathcal{K}_r(A, b) = span(b, Ab, A^2b, \ldots, A^{r-1}b)$$

Symmetric Positive Definite (s.p.d.) Linear Systems

s.p.d. linear systems

$$Ax = b, \quad A \in \mathcal{R}^{n\times n}, \quad A = A^T, \quad \text{and } x^T Ax > 0 \text{ for all non-zero } x \in \mathcal{R}^n$$

CG overview

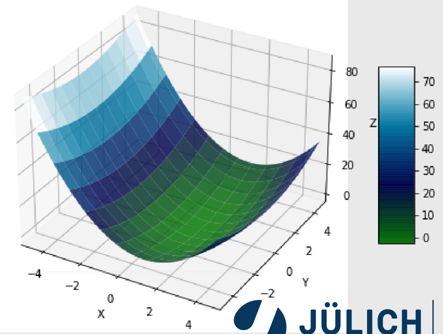
- invented by Hestenes and Stiefel in 1952 as a direct method
- Solve s.p.d. linear system
- Theoretically, converge in n iterations
- Each iteration includes a matrix-vector multiply and a few inner products
- If A is dense, each step costs n^2, so total cost is n^3, same as direct method
- get advantage over dense with a cheaper matrix-vector product operation (SpMV)
- It can work poorly in reality due to round-off error
- for "good" linear systems, can get approximation in far less than n iterations.

CG methodology

Idea

- $f(x) = \frac{1}{2}x^T Ax - b^T x$
- $r = b - Ax$ $\rightarrowtail$ Find x $s.t$ $Ax = b \Leftrightarrow$ Find x $s.t$ $f(x)$ is minimum
- $-\nabla f = Ax - b = r$ with A s.p.d.

$$\begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 3 \\ 2 \end{pmatrix}$$

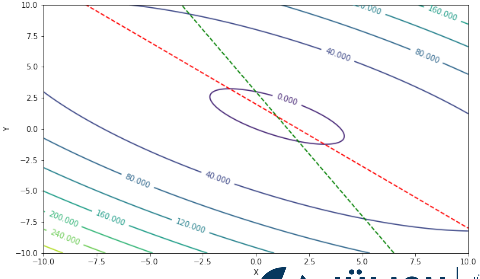


CG methodology

Idea

- $f(x) = \frac{1}{2}x^T Ax - b^T x$
- $r = b - Ax$ $\quad \rightarrowtail$ Find x *s.t* $Ax = b \Leftrightarrow$ Find x *s.t* $f(x)$ is minimum
- $-\nabla f = Ax - b = r$ with A s.p.d.

$$\begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 3 \\ 2 \end{pmatrix}$$

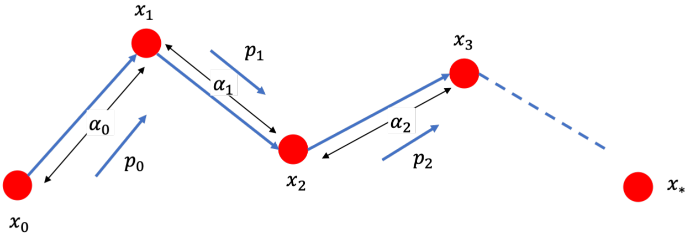


CG methodology

Method

Given x_0 as a starting point:

- **Searching iterate**: $x_{k+1} = x_k + \alpha_k p_k$
- **Search direction**: $p_0, p_1, p_2, \cdots$
- **Step length**: $\alpha_0, \alpha_1, \alpha_2, \cdot$



How to determine step length α_k

$x_{k+1} = x_k + \alpha_k p_k$

For a given x_k and a given direction p_k, find α s.t $f(x)$ is minimized

$$\frac{\mathrm{d}f(x_{k+1})}{\mathrm{d}\alpha} = [\nabla f(x_{k+1})]^T \frac{\mathrm{d}x_{k+1}}{\mathrm{d}\alpha} = -r_{k+1}^T \frac{\mathrm{d}x_{k+1}}{\mathrm{d}\alpha} = -r_{k+1}^T p_k \Rightarrow -r_{k+1}^T p_k = 0$$

$$-r_{k+1}^T p_k = 0 \Rightarrow (b - Ax_{k+1})^T p_k = 0 \Rightarrow (b - A(x_k + \alpha p_k)) = 0 \Rightarrow (r_k - \alpha A p_k)^T p_k = 0$$

$$\Rightarrow \alpha_k = \frac{r_k^T p_k}{p_k^T A p_k}$$

How to pick search direction p

Gradient Descent Method: $p_k = -\nabla f(x_k) = r_k$

Gradient Descent Algorithm

for $k = 0, \cdot .maxIter - 1$ **do**
  $r = b - Ax$
  $\alpha = \frac{r^T r}{r^T Ar}$
  $x = x + \alpha r$
  if $r^T r$ is sufficiently small **then**
    exit loop
  end if
end for

How to pick search direction p

Gradient Descent Method: $p_k = -\nabla f(x_k) = r_k$

Gradient Descent Algorithm

$r = b - Ax$
for $k = 0, \cdot .maxIter - 1$ **do**
  $\alpha = \frac{r^T r}{r^T Ar}$
  $x = x + \alpha r$
  if $r^T r$ is sufficiently small **then**
    exit loop
  end if
  $r = r - \alpha Ar$
end for

How to pick search direction p

Conjugate Gradient Method

given x_0, $p_0 = -\nabla f = r$ (r is the gradient of f)
given x_k, $p_{k+1} = r_{k+1} + \beta_k p_k$, in which p_{k+1} and p_k are A-conjugate ($p_{k+1}^T A p_k = \langle p_{k+1}, p_k \rangle_A = 0$)

$$\langle p_{k+1}, p_k \rangle_A = p_{k+1}^T A p_k = (r_{k+1} + \beta_k p_k)^T A p_k = 0$$

$$\Rightarrow \beta_k = -\frac{r_{k+1}^T A p_k}{p_k^T A p_k}$$

Summary

Finally we have

- $x_{k+1} = x_k + \alpha p_k$
- $p_{k+1} = r_{k+1} + \beta_k p_k$
- $\alpha = \frac{r_k^T p_k}{p_k^T A p_k}$
- $\beta_k = -\frac{r_{k+1}^T A p_k}{p_k^T A p_k}$
- $\langle r_k, p_j \rangle = 0, \quad j < k$

- $\langle r_k, r_j \rangle = 0, \quad j < k$
- $\langle A p_k, p_j \rangle = 0, \quad j < k$
- $r_0, r_1, r_2, \cdots$: Orthogonal
- $p_0, p_1, p_2, \cdots$: A-orthogonal
- $span(r_0, \cdots, r_{k-1}) = span(p_0, \cdots, p_{k-1}) = K(A, r_0)$

CG algorithm: preliminary version

$r_0 = b - Ax_0$ ▷ SpMV + BLAS 1: AXPY
$p_0 = r_0$ ▷ BLAS 1: COPY
for $k = 0, ..maxIter - 1$ **do**
  $\omega_k = Ap_k$ ▷ SpMV
  $\alpha_k = \frac{{r_k}^T p_k}{{p_k}^T \omega_k}$ ▷ BLAS 1: DOT
  $x_{k+1} = x_k + \alpha_k p_k$ ▷ BLAS 1: AXPY
  $r_{k+1} = b - Ax_{k+1}$ ▷ SpMV + BLAS 1: AXPY
  if $||r_{k+1}|| < tol$ **then**
    break
  end if
  $\beta_k = -\frac{r_{k+1}^T \omega_k}{p_k^T \omega_k}$ ▷ BLAS 1: DOT
  $p_{k+1} = r_{k+1} + \beta_k p_k$ ▷ BLAS 1: AXPY
end for

Summary

Finally we have

- $x_{k+1} = x_k + \alpha p_k$
- $r_{k+1} = r_k - \alpha_k A p_k$
- $p_{k+1} = r_{k+1} + \beta_k p_k$
- $\alpha = \frac{r_k^T p_k}{p_k^T A p_k} = \frac{{r_k}^T r_k}{{p_k}^T \omega_k}$
- $\beta_k = -\frac{r_{k+1}^T A p_k}{p_k^T A p_k} = \frac{{r_{k+1}}^T r_{k+1}}{{r_k}^T r_k}$
- $\langle r_k, p_j \rangle = 0, \quad j < k$

- $\langle r_k, r_j \rangle = 0, \quad j < k$
- $\langle A p_k, p_j \rangle = 0, \quad j < k$
- $r_0, r_1, r_2, \cdots$: Orthogonal
- $p_0, p_1, p_2, \cdots$: A-orthogonal
- $span(r_0, \cdots, r_{k-1}) = span(p_0, \cdots, p_{k-1}) = K(A, r_0)$

CG algorithm: economical version

$r_0 = b - Ax_0$ ▷ SpMV + BLAS 1: AXPY
$p_0 = r_0$ ▷ BLAS 1: COPY
for $k = 0, \cdot .maxIter - 1$ **do**
  $\omega_k = Ap_k$ ▷ SpMV
  $\alpha_k = \frac{{r_k}^T r_k}{{p_k}^T \omega_k}$ ▷ BLAS 1: DOT
  $x_{k+1} = x_k + \alpha_k p_k$ ▷ BLAS 1: AXPY
  $r_{k+1} = r_k - \alpha_k \omega_k$ ▷ BLAS 1: AXPY
  if $||r_{k+1}|| < tol$ **then**
    break
  end if
  $\beta_k = \frac{{r_{k+1}}^T r_{k+1}}{{r_k}^T r_k}$ ▷ BLAS 1: DOT
  $p_{k+1} = r_{k+1} + \beta_k p_k$ ▷ BLAS 1: AXPY
end for

CG: 4×4 matrix

Example: Solve

$$\begin{pmatrix} 12 & -1 & 2 & 0 \\ -1 & 14 & -1 & 3 \\ 2 & -1 & 9 & -1 \\ 0 & 3 & -1 & 8 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 8 \\ 31 \\ -10 \\ 15 \end{pmatrix}$$

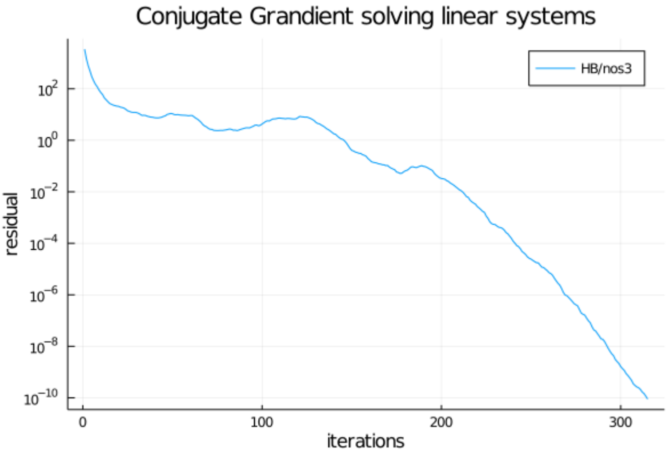
Exact solution:

$$x^* = \begin{pmatrix} 1 \\ 2 \\ -1 \\ 1 \end{pmatrix}$$

	k=0	k=1	k=2	k=3	k=4
x_0	0	0.545014	1.007874	1.000058	1.000000
x_1	0	2.111929	2.008764	1.999956	2.000000
x_2	0	-0.681267	-0.984438	-1.000113	-1.000000
x_3	0	1.021901	1.026010	1.000067	1.000000
$\|\|r_k\|\|$	36.742346	5.553680	0.328046	0.001235	0.000000

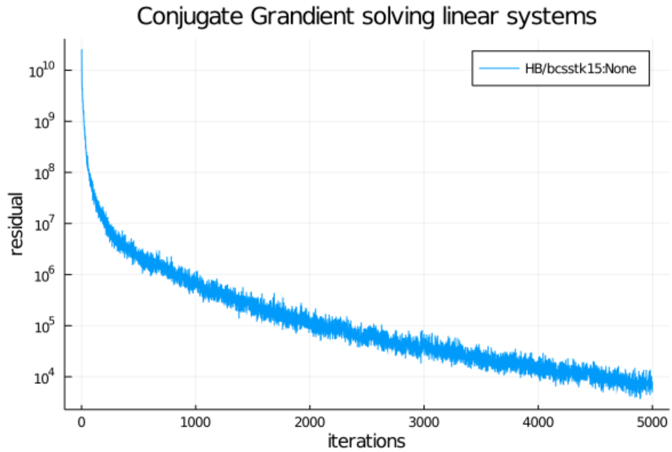
CG example 1: HB/nos3

960×960 symmetric matrix, FE for Biharmonic operator on Plate



CG example 2: HB/bcsstk15

3948 $\times$ 3948 matrix - module of an offshore platform



Convergence Bounds of CG

Let $\lambda_1 \leq \lambda_2 \cdots \leq \lambda_n$ be the ordered eigenvalues of a s.p.d. A:

- $||X_{t+1} - x_*||_A^2 \leq (\frac{\lambda_{n-t}-\lambda_1}{\lambda_{n-t}+\lambda_1})^2 ||x_0 - x_*||_A^2$
- $||X_{t+1} - x_*||_A^2 \leq 2(\frac{\sqrt{\kappa(A)}-1}{\sqrt{\kappa(A)}+1})^t ||x_0 - x_*||_A^2$,

  where $\kappa(A) = \frac{\lambda_n}{\lambda_1}$ is the condition number of A.

Important messages:

- Roughly speaking, if the eigenvalues of A occur in r distinct clusters, the CG iterates will approximately solve the problem after $\mathcal{Q}(r)$ steps.
- A with a small condition number (a single cluster of eigenvalues) converges fast
  - e.g., condition number of *nos*3 matrix is 37723.6, and the one of *bcsstk*15 is $6.53819e + 09$.

Preconditioned Conjugate Gradient algorithm (PCG)

- idea: apply CG after linear change of coordinates $x = Ty$, with $det(T) \neq 0$
- use standard CG to solve $T^T ATy = T^T b$, then $x^* = T^{-1} y^*$
- $M = TT^T$ is called a preconditioner
- can re-arrange computation so each iteration requires one multiply by M (and A), and no final solve $x^* = T^{-1} y^*$
- if spectrum of $T^T AT$ (which is the same as the one of MA) is clustered or $\kappa(A)$ is small, PCG converges fast
  - extreme case: $M = A^{-1}$, which makes MA an identity matrix

Preconditioned CG: algorithm

with preconditioner $M \approx A^{-1}$ (hopefully)

$r_0 = b - Ax_0$ ▷ SpMV + BLAS 1: AXPY
$p_0 = r_0$ ▷ BLAS 1: COPY
for $k = 0, \cdot\cdot maxIter - 1$ **do**
    $\omega_k = Ap_k$ ▷ SpMV
    $\alpha_k = \frac{{r_k}^T r_k}{{p_k}^T \omega_k}$ ▷ BLAS 1: DOT
    $x_{k+1} = x_k + \alpha_k p_k$ ▷ BLAS 1: AXPY
    $r_{k+1} = r_k - \alpha_k \omega_k$ ▷ BLAS 1: AXPY
    if $||r_{k+1}|| < tol$ **then**
        break
    end if
    $\beta_k = \frac{{r_{k+1}}^T r_{k+1}}{{r_k}^T r_k}$ ▷ BLAS 1: DOT
    $p_{k+1} = r_{k+1} + \beta_k p_k$ ▷ BLAS 1: AXPY
end for

Preconditioned CG: algorithm

with preconditioner $M \approx A^{-1}$ (hopefully)

$r_0 = b - Ax_0$ ▷ SpMV + BLAS 1: AXPY
$p_0 = r_0$ ▷ BLAS 1: COPY
$z_0 = Mr_0$ ▷ SpMV? BLAS 2 GEMV?
for $k = 0, \cdot\cdot maxIter - 1$ **do**
    $\omega_k = Ap_k$ ▷ SpMV
    $\alpha_k = \frac{{r_k}^T z_k}{{p_k}^T \omega_k}$ ▷ BLAS 1: DOT
    $x_{k+1} = x_k + \alpha_k p_k$ ▷ BLAS 1: AXPY
    $r_{k+1} = r_k - \alpha_k \omega_k$ ▷ BLAS 1: AXPY
    if $||r_{k+1}|| < tol$ **then**
        break
    end if
    $z_k = Mr_k$ ▷ SpMV? BLAS 2 GEMV?
    $\beta_k = \frac{{r_{k+1}}^T r_{k+1}}{{r_k}^T r_k}$ ▷ BLAS 1: DOT
    $p_{k+1} = r_{k+1} + \beta_k p_k$ ▷ BLAS 1: AXPY
end for

Some generic preconditioners

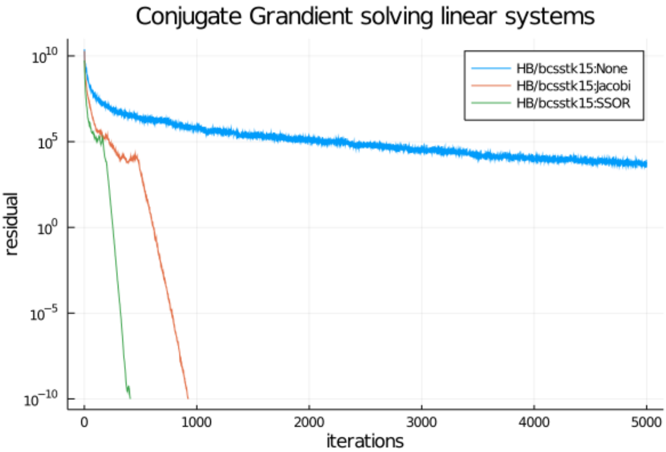
For a symmetric positive definite matrix A, some generic preconditioners are:

- **Jacobi**: $M = D^{-1}$, with D is the diagonal of matrix A
- **SSOR**[1]: $M = P^{-1}$, with $P = (D + L)D^{-1}(D + L)^T$
  - D refers to the diagonal of A
  - L refers to the lower triangular part of A
- **Incomplete Cholesky factorization**: use $M = \hat{A}^{-1}$, where $\hat{A} = \hat{L}\hat{L}^T$ is an approximation of A with cheap Cholesky factorization
  - Compute $\hat{A} = \hat{L}\hat{L}^T$
    - $\hat{A}$ can be central k wide band of A
    - $\hat{L}$ obtained by sparse Cholesky factorization of A, ignoring small elements in A, or refusing to create excessive fill-in.
  - at each iteration, compute $Mz = \hat{L}^{-T}\hat{L}^{-1}z$ with forward/backward substitution

[1] Symmetric successive over-relaxation

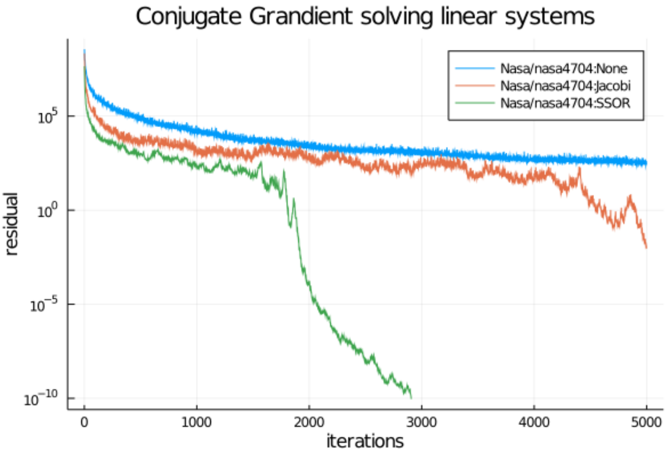
PCG example 1: HB/bcsstk15

3948 × 3948 matrix - module of an offshore platform



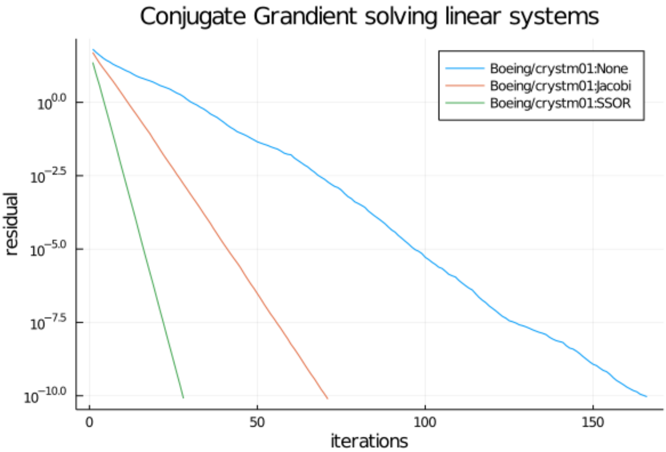
PCG example 2: Nasa/nasa4704

4704×4704 matrix - from NASA Langley



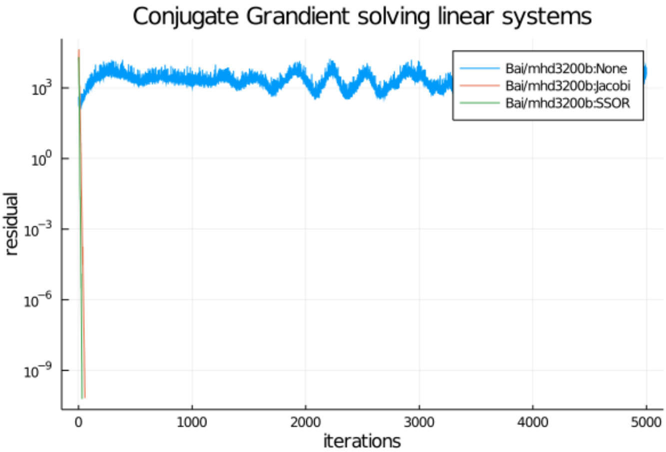
PCG example 3: Boeing/crystm01

4875×4875 **FEM Crystal free vibration mass matrix**



PCG example 4: Bai/mhd3200b

3200 × 3200matrixforAlfvenspectrainMagnetohydrodynamics



Choice of preconditioner

- trade-off between enhanced convergence, and extra cost of multiplication by M at each step
  - **SpMV** if M could be sparse, e.g., Jacobi preconditioner
  - **BLAS 2 GEMV** if M could be dense, e.g., SSOR preconditioner
- goal is to find M that is cheap to multiply, and approximate inverse of A (or at least has a more clustered spectrum than A)

Choice of preconditioner

- trade-off between enhanced convergence, and extra cost of multiplication by M at each step
  - **SpMV** if M could be sparse, e.g., Jacobi preconditioner
  - **BLAS 2 GEMV** if M could be dense, e.g., SSOR preconditioner
- goal is to find M that is cheap to multiply, and approximate inverse of A (or at least has a more clustered spectrum than A)

This strategy of this trade-off will be demonstrated in **homework 1** by exercises

(P)CG summary

- in theory (with exact arithmetic) converges to solution in n steps
  - the bad news: due to numerical round-off errors, can take more than n steps (or fail to converge)
  - the good news: with luck (i.e., good spectrum of A), can get good approximate solution in $\ll n$ steps
- each step requires $v \to Av$ multiplication
  - can exploit a variety of structure in A
  - in many cases, never form or store the matrix A explicitly
- A good choice of preconditioner will significantly speedup the solving procedure
- compared to direct (factor-solve) methods, CG is less reliable, data dependent; often requires good (problem-dependent) preconditioner
- but, when it works, can solve extremely large systems

Part II: PageRank Method

Problem Statement

Not all web pages are equally ”important”.

- https://www.bbc.com (BBC)

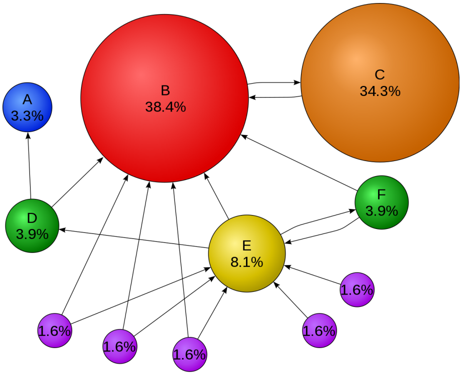
vs

- https://brunowu.github.io (My personal webpage)

PageRank (PR):

- an algorithm used by Google Search to rank web pages in their search engine results
  - mesuring the importance of webpages..
- introduced by Larray Page. the co-founder of Google.

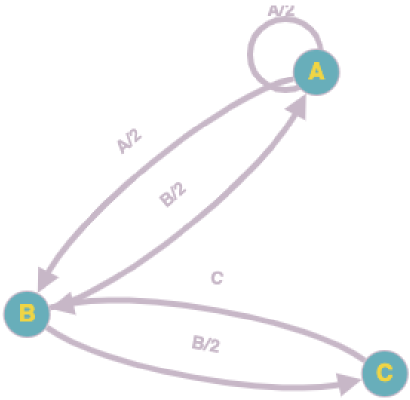
PageRank: Links as votes



[source: https://en.wikipedia.org/wiki/PageRank]

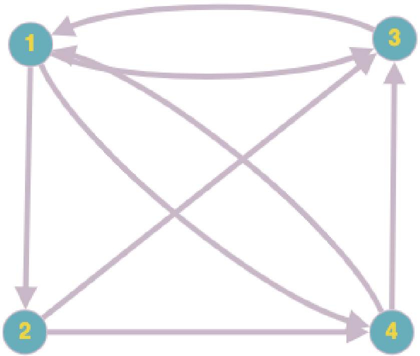
- Links as votes
- In-links as votes
- In-links are not equal:
  - Links from important pages count more
  - Recursive definition

PageRank: Links as votes



- Each link's vote is proportional to the importance of its source page
- Page j with importance r_j has n outlinks, each links gets $\frac{r_j}{n}$
- Page's own importance is the sum of the votes on its in-links
  - a "rank" r_j for page j is $r_j = \sum_{i \to j} \frac{r_i}{d_i}$
  - additional constraint $\sum_j r_j = 1$

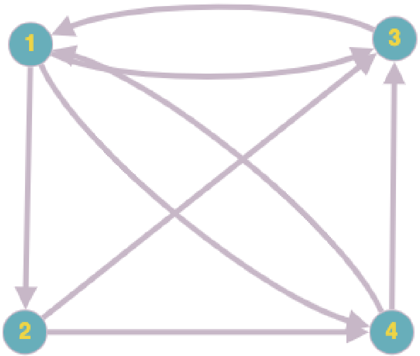
Construct Stochastic Adjacency Matrix from Graph



For a stochastic adjacency matrix M

- Page i has d_i out-links
- If $i \rightarrow j$, the $M_{ji} = \frac{1}{d_i}$, else $M_{ji} = 0$
- columns sum to 1

Construct Stochastic Adjacency Matrix from Graph



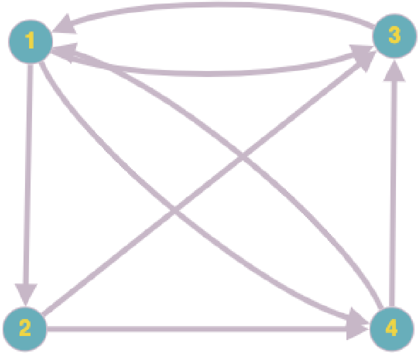
For a stochastic adjacency matrix M

- Page i has d_i out-links
- If $i \rightarrow j$, the $M_{ji} = \frac{1}{d_i}$, else $M_{ji} = 0$
- columns sum to 1

Adjacency Matrix

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

Construct Stochastic Adjacency Matrix from Graph



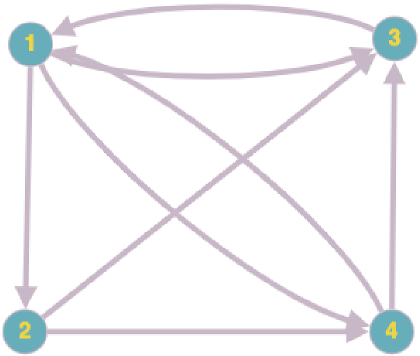
For a stochastic adjacency matrix M

- Page i has d_i out-links
- If $i \rightarrow j$, the $M_{ji} = \frac{1}{d_i}$, else $M_{ji} = 0$
- columns sum to 1

Stochastic Adjacency Matrix

$$\begin{pmatrix} 0 & 0 & 1 & 1/2 \\ 1/3 & 0 & 0 & 0 \\ 1/3 & 1/2 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{pmatrix}$$

Construct Stochastic Adjacency Matrix from Graph

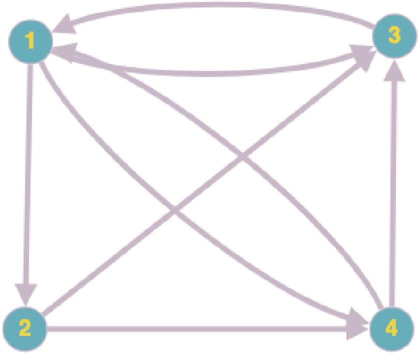


For a stochastic adjacency matrix M

- Page i has d_i out-links
- If $i \rightarrow j$, the $M_{ji} = \frac{1}{d_i}$, else $M_{ji} = 0$
- columns sum to 1

$$\Rightarrow r = Mr$$

Construct Stochastic Adjacency Matrix from Graph



$$\Rightarrow r = Mr$$

- To solve it is to find the eigenvectors with corresponding eigenvalue 1
- Luckily, Largest eigenvalue of a stochastic matrix with non-negative entries is 1
- We can use Power Iteration method.

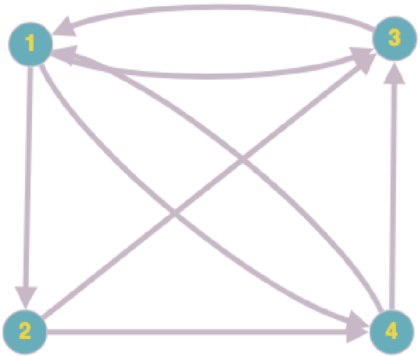
A PageRank solver based on Power Iteration

Power iteration method to solve PageRank graph

- At $t = 0$, an initial probability distribution V is randomly generated.
- At each time step, the computation, $r_{t+1} = Mr_t$
- Convergence is assumed when $|V_{t+1} - V_t| < \epsilon$ for some small ϵ.

The most important kernel of this solver is $r_{t+1} = Mr_t$, SpMV.

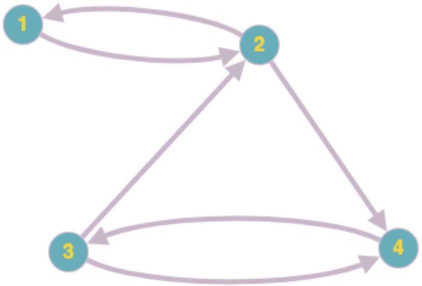
A PageRank solver based on Power Iteration



Try to interpret the result

$$\begin{pmatrix} 0.39 \\ 0.13 \\ 0.29 \\ 0.19 \end{pmatrix}$$

Spider traps

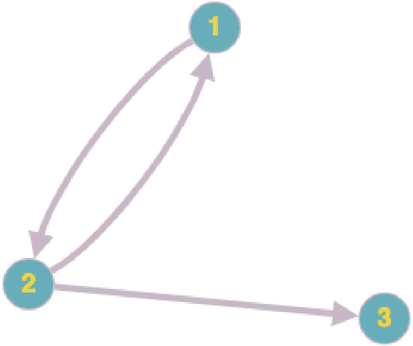


all out-links are within a group

- Random walk gets “stuck” in a trap
- it absorbs all importance

$$\begin{pmatrix} 0 & 0.5 & 0 & 0 \\ 1 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0.5 & 0.5 & 0 \end{pmatrix} \text{ with } \begin{pmatrix} 0.142 \\ 0.286 \\ 0.286 \\ 0.286 \end{pmatrix}$$

Dead ends



all out-links are within a group

- "No where to go" for some random walk
- "leaking" the importance

$$\begin{pmatrix} 0 & 0.5 & 0 \\ 1 & 0 & 0 \\ 0 & 0.5 & 0 \end{pmatrix} \text{ with } \begin{pmatrix} 0.0 \\ 0.0 \\ 0.0 \end{pmatrix}$$

Google's solution: introducing a Damping factor

a "rank " r_j of a webpage j with a damping factor β

$$r_j = \sum_{i \to j} \beta \frac{r_i}{d_i} + (1 - \beta)\frac{1}{N}$$

At each time step, the random surfer has two options:

- follow a link at random with probability β
- jump to some random page with probability $1 - \beta$
- Common value for β is between 0.8 and 0.9

Try the PageRank with Damping factor in the homework.

PageRank Summary

- ”Normal” PageRank
- Topic-specific PageRank (Personalized PageRank)
- Random walk with restarts

Homework

- CG: `./tasks/homework-1/LinearSolver.ipynb`
- PageRank: `./tasks/homework-2/PageRank.ipynb`