Revisit of Monte Carlo Methods on Solving Large-Scale Linear Systems

YAOHANG LI, PH.D.

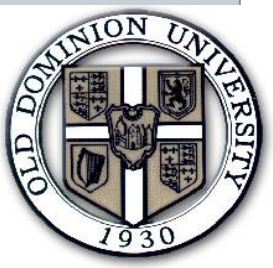
DEPARTMENT OF COMPUTER SCIENCE
OLD DOMINION UNIVERSITY

yaohang@cs.odu.edu

Nov. 4, 2014 @ NIST

Agenda

- Numerical Linear Algebra in Big Data Era
- Revisit of Ulam-von Neumann Scheme
- Breakdown-Free BCG
- Randomized SVD
- BCG with Adaptive Deflation
- Results and Applications
- Summary

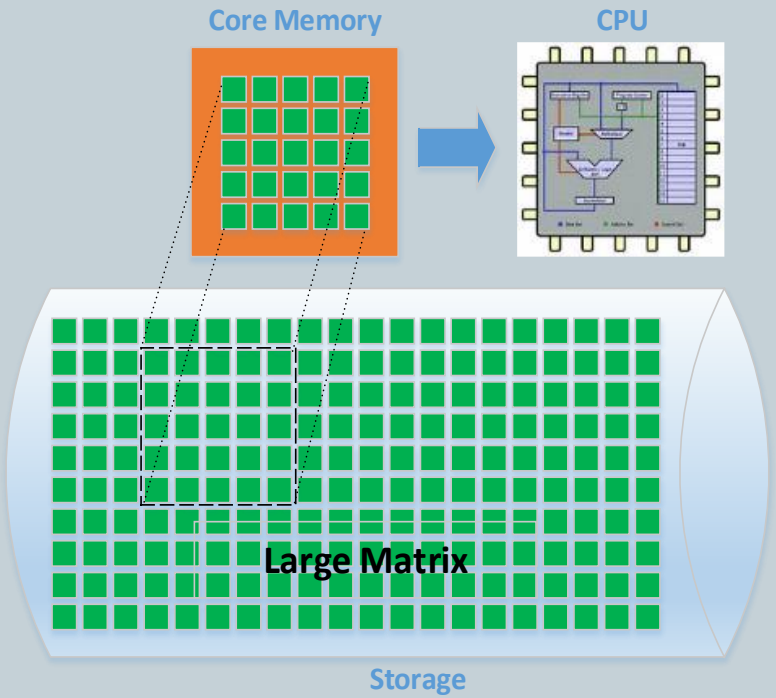


Numerical Linear Algebra for BIGDATA

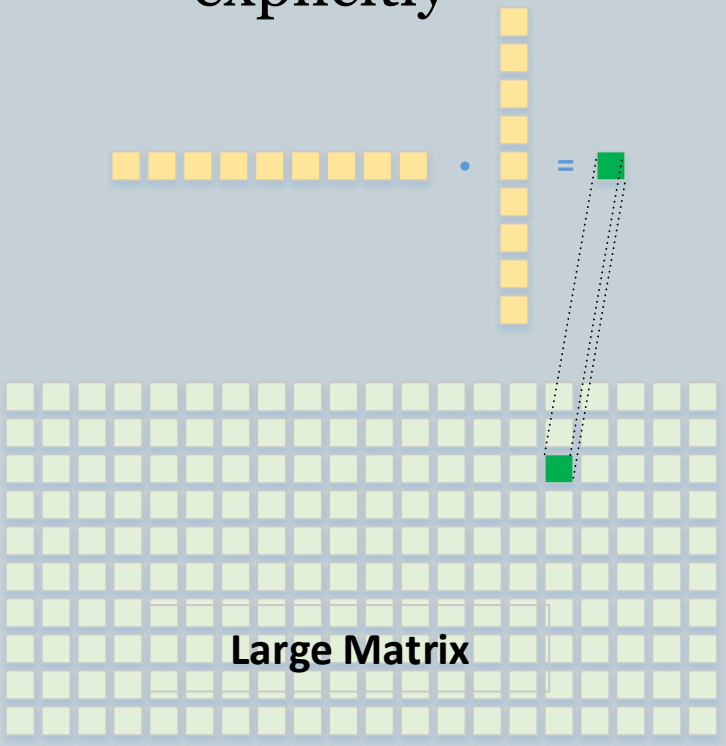
- Numerical Linear Algebra in Big Data Era
  - Characterizing by Large Matrices
    - Million x Million to Billion x Billion
    - Most likely sparse
  - Operations
    - Approximating Matrix-Matrix/Matrix-Vector Multiplications
    - Solving Linear Systems with Large Coefficient Matrices
    - Finding Extreme Eigenvalues/Eigenvectors
      - Top-*k* Eigenvalues/Eigenvectors
      - Low-*k* Eigenvalues/Eigenvectors
    - Estimating the Determinant

Characteristics of Numerical Linear Algebra in Big Data Era (1)

Matrices may not fit in the main memory



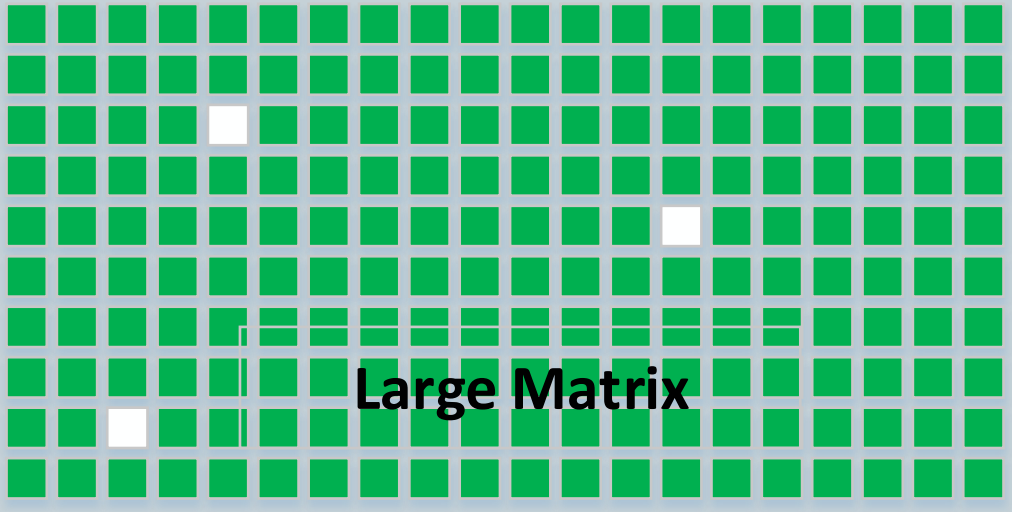
Matrices may not exist explicitly



Accessing Matrix Elements Becomes the New Bottleneck

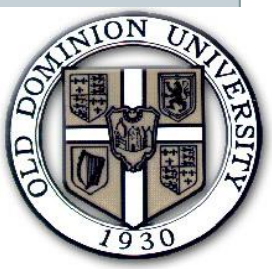
Characteristics of Numerical Linear Algebra in Big Data Era (2)

- Matrices may be incomplete
  - Some elements may be missing
  - Chance of memory errors increases



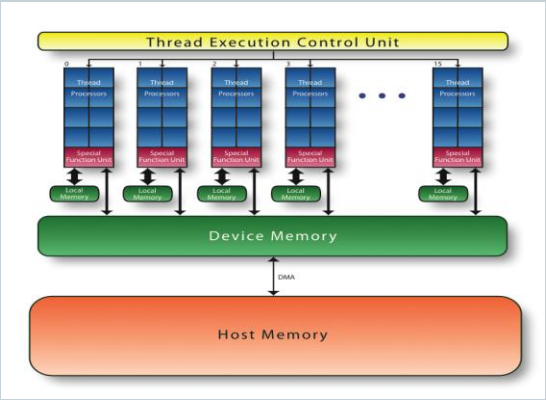
Characteristics of Numerical Linear Algebra in Big Data Era (3)

- Solution precision requirement is usually not very high
  - 10^{-2} even 10^{-1} is enough, sometimes
- Vice versa, computational speed (response time) is usually more important, instead

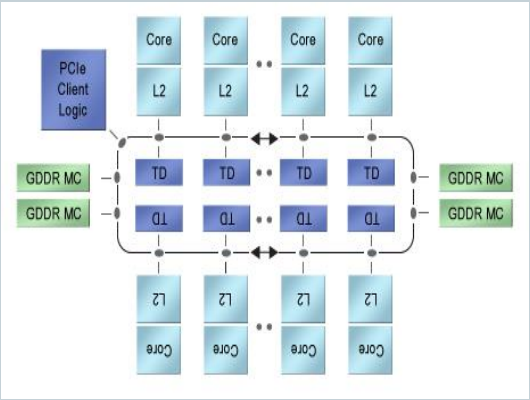


Characteristics of Numerical Linear Algebra in Big Data Era (4)

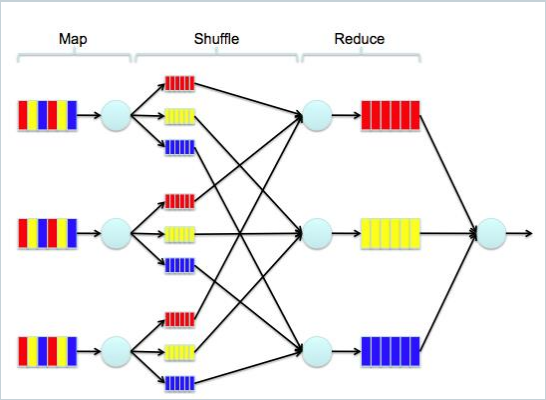
- Modern Parallel/Distributed Computing Paradigms/Architectures



Nvidia GPGPU



Intel Xeon Phi

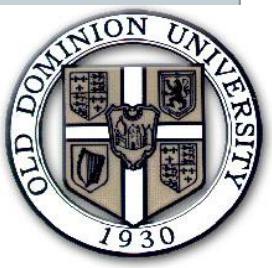


Map/Reduce Paradigm

A Little Bit of History

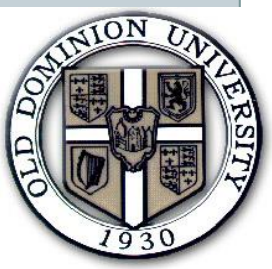
- Conjugate Gradient (CG) Methods
  - Developed in 1950's
    - Not considered as efficient methods compared to direct methods
  - Start to gain attention in 1970's
    - Bigger matrices with sparsity
    - Efficient matrix-vector multiplication
    - Powerful iterative method for sparse linear systems
    - Catalyst for subsequent work
      - Krylov subspace methods
      - Preconditioning
      - Parallel computing
      - etc.

[Golub & O'Leary, SIAM Review, 1989]



Can History Repeat Itself?

- Monte Carlo Methods for Solving Linear Systems
  - Developed in 1950s by Ulam and von Neumann
  - Statistical sampling
  - Not considered as efficient methods compared to deterministic methods
- Can Monte Carlo Methods be Resurrected in the Big Data Era?
  - Sampling matrices (reduced visits to matrix elements)
  - Fast computation with low relative accuracy of 10^{-1} or 10^{-2} residual error
  - Natural parallel



Ulam-von Neumann Scheme

- Ulam-von Neumann Scheme
  - Construct Markov chains to sample the Neumann Series
- Consider a Linear System

$$\boldsymbol{x} = \boldsymbol{H}\boldsymbol{x} + \boldsymbol{b}$$

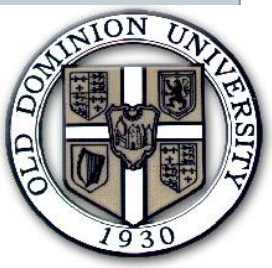
- Neumann Series

$$\boldsymbol{I} + \boldsymbol{H} + \boldsymbol{H}^2 + \boldsymbol{H}^3 + \ldots$$

  - If $\rho(\boldsymbol{H}) < 1$

$$\boldsymbol{I} + \boldsymbol{H} + \boldsymbol{H}^2 + \boldsymbol{H}^3 + \ldots = (\boldsymbol{I} - \boldsymbol{H})^{-1}$$

[Hammersley & Handscomb, 1964]



Ulam-von Neumann Scheme (cont.)

- Transition Matrix P

$$P_{ij} \geq 0;$$

$$\sum_j P_{ij} \leq 1;$$

$$H_{ij} \neq 0 \rightarrow P_{ij} \neq 0$$

- Termination Probability T_i

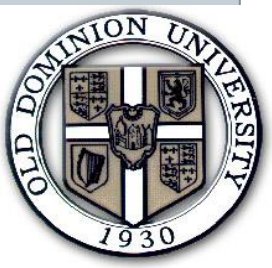
$$T_i = 1 - \sum_j P_{ij}$$

- Random walk γ

$$\gamma : r_0 \rightarrow r_1 \rightarrow r_2 \rightarrow \ldots \rightarrow r_k$$

- Then

$X(\gamma) = \dfrac{H_{r_0 r_1} H_{r_1 r_2} \ldots H_{r_{k-1} r_k}}{P_{r_0 r_1} P_{r_1 r_2} \ldots P_{r_{k-1} r_k}} b_{r_k} / T_{r_k}$ is an unbiased estimator of x_{r0}



An Improved Monte Carlo Algorithm

- Monte Carlo Almost Optimal (MAO)
- An alternative transition matrix P

$$P_{ij} = \frac{|H_{ij}|}{\sum_k |H_{ik}|}$$

- Adaptive termination

$$W_0 = 1;$$

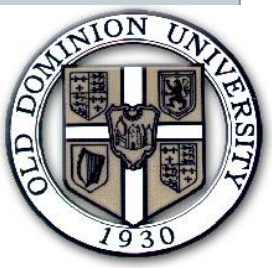
$$W_k = W_{k-1} \frac{H_{r_{k-1} r_k}}{P_{r_{k-1} r_k}} \ldots;$$

- An alternative estimator of u_{r0}

$$X(\gamma) = \sum_k W_k b_{k r_k}$$

- Better accuracy (smaller variance)

[Dimov et al., 1998]



Perform one trajectory:

Set initial values

$X = 0, W = 1, index = i;$

Calculate $X = X + W b_{index}$;

While $(|W| > \varepsilon)$

Generate an uniformly distributed random number $\xi \in [0,1)$;

Set $j = 1$;

While $(\xi > \sum_{s=1}^{j} p_{index,s})$

$j = j + 1$;

End

$W = W * sign(h_{index,j}) * hsum(index)$;

$X = X + W * b_j$;

$index = j$;

End

return X;

For $x(1)$, $\varepsilon = 0.002$

$$\mathrm{H} = \begin{bmatrix} 0.1 & 0.45 & 0.225 \\ -0.15 & 0.1 & -0.3 \\ -0.18 & 0.36 & 0.1 \end{bmatrix} \quad b = \begin{bmatrix} 0.225 \\ 1.35 \\ 0.72 \end{bmatrix}$$

$$hsum = \begin{bmatrix} 0.775 \\ 0.55 \\ 0.64 \end{bmatrix}$$

$$P = \begin{bmatrix} 0.129032 & 0.580645 & 0.290323 \\ 0.272727 & 0.181818 & 0.545455 \\ 0.28125 & 0.5625 & 0.15625 \end{bmatrix}$$

index	ξ	W	X	j
1		1	0	

Perform one trajectory:

Set initial values

$X = 0, W = 1, index = i;$

Calculate $X = X + Wb_{index};$

While $(|W| > \varepsilon)$

Generate an uniformly distributed random number $\xi \in [0,1)$;

Set $j = 1$;

While $(\xi > \sum_{s=1}^{j} p_{index,s})$

$j = j + 1;$

End

$W = W * sign(h_{index,j}) * hsum(index);$

$X = X + W * b_j;$

$index = j;$

End

return X;

For $x(1)$, $\varepsilon = 0.002$

$$\mathrm{H} = \begin{bmatrix} 0.1 & 0.45 & 0.225 \\ -0.15 & 0.1 & -0.3 \\ -0.18 & 0.36 & 0.1 \end{bmatrix} \quad b = \begin{bmatrix} 0.225 \\ 1.35 \\ 0.72 \end{bmatrix}$$

$$hsum = \begin{bmatrix} 0.775 \\ 0.55 \\ 0.64 \end{bmatrix}$$

$$P = \begin{bmatrix} 0.129032 & 0.580645 & 0.290323 \\ 0.272727 & 0.181818 & 0.545455 \\ 0.28125 & 0.5625 & 0.15625 \end{bmatrix}$$

index	θ	W	X	j
1		1	0.225	

Perform one trajectory:

- Set initial values
- $X = 0, W = 1, index = i;$
- Calculate $X = X + W b_{index};$
- While $(|W| > \varepsilon)$
  - Generate an uniformly distributed random number $\xi \in [0,1)$;
  - Set $j = 1;$
  - While $(\xi > \sum_{s=1}^{j} p_{index,s})$
    - $j = j + 1;$
  - End
  - $W = W * sign(h_{index,j}) * hsum(index);$
  - $X = X + W * b_j;$
  - $index = j;$
- End
- return $X;$

For $x(1)$, $\varepsilon = 0.002$

$$H = \begin{bmatrix} 0.1 & 0.45 & 0.225 \\ -0.15 & 0.1 & -0.3 \\ -0.18 & 0.36 & 0.1 \end{bmatrix} \quad b = \begin{bmatrix} 0.225 \\ 1.35 \\ 0.72 \end{bmatrix}$$

$$hsum = \begin{bmatrix} 0.775 \\ 0.55 \\ 0.64 \end{bmatrix}$$

$$P = \begin{bmatrix} 0.129032 & 0.580645 & 0.290323 \\ 0.272727 & 0.181818 & 0.545455 \\ 0.28125 & 0.5625 & 0.15625 \end{bmatrix}$$

index	ξ	W	X	j
1		1	0.225	

Perform one trajectory:

Set initial values

$X = 0, W = 1, index = i;$

Calculate $X = X + W b_{index}$;

While $(|W| > \varepsilon)$

Generate an uniformly distributed random number $\xi \in [0,1)$;

Set $j = 1$;

While $(\xi > \sum_{s=1}^{j} p_{index,s})$

$j = j + 1$;

End

$W = W * sign(h_{index,j}) * hsum(index)$;

$X = X + W * b_j$;

$index = j$;

End

return X;

For $x(1)$, $\varepsilon = 0.002$

$$\mathrm{H} = \begin{bmatrix} 0.1 & 0.45 & 0.225 \\ -0.15 & 0.1 & -0.3 \\ -0.18 & 0.36 & 0.1 \end{bmatrix} \quad b = \begin{bmatrix} 0.225 \\ 1.35 \\ 0.72 \end{bmatrix}$$

$$hsum = \begin{bmatrix} 0.775 \\ 0.55 \\ 0.64 \end{bmatrix}$$

$$P = \begin{bmatrix} 0.129032 & 0.580645 & 0.290323 \\ 0.272727 & 0.181818 & 0.545455 \\ 0.28125 & 0.5625 & 0.15625 \end{bmatrix}$$

index	ξ	W	X	j
1	0.3937	1	0.225	

Perform one trajectory:

Set initial values

$X = 0, W = 1, index = i;$

Calculate $X = X + W b_{index};$

While $(|W| > \varepsilon)$

Generate an uniformly distributed random number $\xi \in [0,1)$;

Set $j = 1;$

While $(\xi > \sum_{s=1}^{j} p_{index,s})$

$j = j + 1;$

End

$W = W * sign(h_{index,j}) * hsum(index);$

$X = X + W * b_j;$

$index = j;$

End

return $X;$

For $x(1), \quad \varepsilon = 0.002$

$$\mathrm{H} = \begin{bmatrix} 0.1 & 0.45 & 0.225 \\ -0.15 & 0.1 & -0.3 \\ -0.18 & 0.36 & 0.1 \end{bmatrix} \qquad b = \begin{bmatrix} 0.225 \\ 1.35 \\ 0.72 \end{bmatrix}$$

$$hsum = \begin{bmatrix} 0.775 \\ 0.55 \\ 0.64 \end{bmatrix}$$

$$P = \begin{bmatrix} 0.129032 & 0.580645 & 0.290323 \\ 0.272727 & 0.181818 & 0.545455 \\ 0.28125 & 0.5625 & 0.15625 \end{bmatrix}$$

index	ξ	W	X	j
1	0.3937	1	0.225	1

Perform one trajectory:

Set initial values

$X = 0, W = 1, index = i;$

Calculate $X = X + W b_{index};$

While $(|W| > \varepsilon)$

Generate an uniformly distributed random number $\xi \in [0,1)$;

Set $j = 1;$

While $(\xi > \sum_{s=1}^{j} p_{index,s})$

$j = j + 1;$

End

$W = W * sign(h_{index,j}) * hsum(index);$

$X = X + W * b_j;$

$index = j;$

End

return $X;$

For $x(1), \quad \varepsilon = 0.002$

$$\mathrm{H} = \begin{bmatrix} 0.1 & 0.45 & 0.225 \\ -0.15 & 0.1 & -0.3 \\ -0.18 & 0.36 & 0.1 \end{bmatrix} \qquad b = \begin{bmatrix} 0.225 \\ 1.35 \\ 0.72 \end{bmatrix}$$

$$hsum = \begin{bmatrix} 0.775 \\ 0.55 \\ 0.64 \end{bmatrix}$$

$$P = \begin{bmatrix} 0.129032 & 0.580645 & 0.290323 \\ 0.272727 & 0.181818 & 0.545455 \\ 0.28125 & 0.5625 & 0.15625 \end{bmatrix}$$

index	ξ	W	X	j
1	0.3937	1	0.225	2

Perform one trajectory:

Set initial values

$X = 0, W = 1, index = i;$

Calculate $X = X + W b_{index}$;

While $(|W| > \varepsilon)$

Generate an uniformly distributed random number $\xi \in [0,1)$;

Set $j = 1$;

While $(\xi > \sum_{s=1}^{j} p_{index,s})$

$j = j + 1$;

End

$W = W * sign(h_{index,j}) * hsum(index)$;

$X = X + W * b_j$;

$index = j$;

End

return X;

For $x(1)$, $\varepsilon = 0.002$

$$H = \begin{bmatrix} 0.1 & 0.45 & 0.225 \\ -0.15 & 0.1 & -0.3 \\ -0.18 & 0.36 & 0.1 \end{bmatrix} \quad b = \begin{bmatrix} 0.225 \\ 1.35 \\ 0.72 \end{bmatrix}$$

$$hsum = \begin{bmatrix} 0.775 \\ 0.55 \\ 0.64 \end{bmatrix}$$

$$P = \begin{bmatrix} 0.129032 & 0.580645 & 0.290323 \\ 0.272727 & 0.181818 & 0.545455 \\ 0.28125 & 0.5625 & 0.15625 \end{bmatrix}$$

index	ξ	W	X	j
1	0.3937	1	0.225	2
2		0.775	1.271	

Perform one trajectory:

Set initial values

$X = 0, W = 1, index = i;$

Calculate $X = X + W b_{index};$

While $(|W| > \varepsilon)$

Generate an uniformly distributed random number $\xi \in [0,1)$;

Set $j = 1;$

While $(\xi > \sum_{s=1}^{j} p_{index,s})$

$j = j + 1;$

End

$W = W * sign(h_{index,j}) * hsum(index);$

$X = X + W * b_j;$

$index = j;$

End

return $X;$

For $x(1), \quad \varepsilon = 0.002$

$$\mathrm{H} = \begin{bmatrix} 0.1 & 0.45 & 0.225 \\ -0.15 & 0.1 & -0.3 \\ -0.18 & 0.36 & 0.1 \end{bmatrix} \qquad b = \begin{bmatrix} 0.225 \\ 1.35 \\ 0.72 \end{bmatrix}$$

$$hsum = \begin{bmatrix} 0.775 \\ 0.55 \\ 0.64 \end{bmatrix}$$

$$P = \begin{bmatrix} 0.129032 & 0.580645 & 0.290323 \\ 0.272727 & 0.181818 & 0.545455 \\ 0.28125 & 0.5625 & 0.15625 \end{bmatrix}$$

index	ξ	W	X	j
1	0.3937	1	0.225	2
2		0.775	1.271	

Perform one trajectory:

Set initial values

$X = 0, W = 1, index = i;$

Calculate $X = X + W b_{index}$;

While $(|W| > \varepsilon)$

Generate an uniformly distributed random number $\xi \in [0,1)$;

Set $j = 1$;

While $(\xi > \sum_{s=1}^{j} p_{index,s})$

$j = j + 1$;

End

$W = W * sign(h_{index,j}) * hsum(index)$;

$X = X + W * b_j$;

$index = j$;

End

return X;

For $x(1)$, $\varepsilon = 0.002$

$$\mathrm{H} = \begin{bmatrix} 0.1 & 0.45 & 0.225 \\ -0.15 & 0.1 & -0.3 \\ -0.18 & 0.36 & 0.1 \end{bmatrix} \quad b = \begin{bmatrix} 0.225 \\ 1.35 \\ 0.72 \end{bmatrix}$$

$$hsum = \begin{bmatrix} 0.775 \\ 0.55 \\ 0.64 \end{bmatrix}$$

$$P = \begin{bmatrix} 0.129032 & 0.580645 & 0.290323 \\ 0.272727 & 0.181818 & 0.545455 \\ 0.28125 & 0.5625 & 0.15625 \end{bmatrix}$$

index	ξ	W	X	j
1	0.3937	1	0.225	2
2	0.2765	0.775	1.271	

Perform one trajectory:

- Set initial values
- $X = 0, W = 1, index = i;$
- Calculate $X = X + W b_{index};$
- While $(|W| > \varepsilon)$
  - Generate an uniformly distributed random number $\xi \in [0,1)$;
  - Set $j = 1;$
  - While $(\xi > \sum_{s=1}^{j} p_{index,s})$
    - $j = j + 1;$
  - End
  - $W = W * sign(h_{index,j}) * hsum(index);$
  - $X = X + W * b_j;$
  - $index = j;$
- End
- return $X;$

For $x(1), \quad \varepsilon = 0.002$

$$\mathrm{H} = \begin{bmatrix} 0.1 & 0.45 & 0.225 \\ -0.15 & 0.1 & -0.3 \\ -0.18 & 0.36 & 0.1 \end{bmatrix} \qquad b = \begin{bmatrix} 0.225 \\ 1.35 \\ 0.72 \end{bmatrix}$$

$$hsum = \begin{bmatrix} 0.775 \\ 0.55 \\ 0.64 \end{bmatrix}$$

$$P = \begin{bmatrix} 0.129032 & 0.580645 & 0.290323 \\ 0.272727 & 0.181818 & 0.545455 \\ 0.28125 & 0.5625 & 0.15625 \end{bmatrix}$$

index	ξ	W	X	j
1	0.3937	1	0.225	2
2	0.2765	0.775	1.271	1

Perform one trajectory:

Set initial values

$X = 0, W = 1, index = i;$

Calculate $X = X + W b_{index};$

While $(|W| > \varepsilon)$

Generate an uniformly distributed random number $\xi \in [0,1)$;

Set $j = 1;$

While $(\xi > \sum_{s=1}^{j} p_{index,s})$

$j = j + 1;$

End

$W = W * sign(h_{index,j}) * hsum(index);$

$X = X + W * b_j;$

$index = j;$

End

return $X;$

For $x(1), \quad \varepsilon = 0.002$

$$H = \begin{bmatrix} 0.1 & 0.45 & 0.225 \\ -0.15 & 0.1 & -0.3 \\ -0.18 & 0.36 & 0.1 \end{bmatrix} \quad b = \begin{bmatrix} 0.225 \\ 1.35 \\ 0.72 \end{bmatrix}$$

$$hsum = \begin{bmatrix} 0.775 \\ 0.55 \\ 0.64 \end{bmatrix}$$

$$P = \begin{bmatrix} 0.129032 & 0.580645 & 0.290323 \\ 0.272727 & 0.181818 & 0.545455 \\ 0.28125 & 0.5625 & 0.15625 \end{bmatrix}$$

index	ξ	W	X	j
1	0.3937	1	0.225	2
2	0.2765	0.775	1.271	2

Perform one trajectory:

Set initial values

$X = 0, W = 1, index = i;$

Calculate $X = X + W b_{index}$;

While $(|W| > \varepsilon)$

Generate an uniformly distributed random number $\xi \in [0,1)$;

Set $j = 1$;

While $(\xi > \sum_{s=1}^{j} p_{index,s})$

$j = j + 1$;

End

$W = W * sign(h_{index,j}) * hsum(index)$;

$X = X + W * b_j$;

$index = j$;

End

return X;

For $x(1)$, $\varepsilon = 0.002$

$$\mathrm{H} = \begin{bmatrix} 0.1 & 0.45 & 0.225 \\ -0.15 & 0.1 & -0.3 \\ -0.18 & 0.36 & 0.1 \end{bmatrix} \quad b = \begin{bmatrix} 0.225 \\ 1.35 \\ 0.72 \end{bmatrix}$$

$$hsum = \begin{bmatrix} 0.775 \\ 0.55 \\ 0.64 \end{bmatrix}$$

$$P = \begin{bmatrix} 0.129032 & 0.580645 & 0.290323 \\ 0.272727 & 0.181818 & 0.545455 \\ 0.28125 & 0.5625 & 0.15625 \end{bmatrix}$$

index	ξ	W	X	j
1	0.3937	1	0.225	2
2	0.2765	0.775	1.271	2
2		0.426	1.846	

Perform one trajectory:

Set initial values

$X = 0, W = 1, index = i;$

Calculate $X = X + W b_{index};$

While $(|W| > \varepsilon)$

Generate an uniformly distributed random number $\xi \in [0,1)$;

Set $j = 1;$

While $(\xi > \sum_{s=1}^{j} p_{index,s})$

$j = j + 1;$

End

$W = W * sign(h_{index,j}) * hsum(index);$

$X = X + W * b_j;$

$index = j;$

End

return $X;$

For $x(1), \quad \varepsilon = 0.002$

$$\mathrm{H} = \begin{bmatrix} 0.1 & 0.45 & 0.225 \\ -0.15 & 0.1 & -0.3 \\ -0.18 & 0.36 & 0.1 \end{bmatrix} \qquad b = \begin{bmatrix} 0.225 \\ 1.35 \\ 0.72 \end{bmatrix}$$

$$hsum = \begin{bmatrix} 0.775 \\ 0.55 \\ 0.64 \end{bmatrix}$$

$$P = \begin{bmatrix} 0.129032 & 0.580645 & 0.290323 \\ 0.272727 & 0.181818 & 0.545455 \\ 0.28125 & 0.5625 & 0.15625 \end{bmatrix}$$

index	ξ	W	X	j
1	0.3937	1	0.225	2
2	0.2765	0.775	1.271	2
2		0.426	1.846	

Perform one trajectory:

Set initial values

$X = 0, W = 1, index = i;$

Calculate $X = X + W b_{index};$

While $(|W| > \varepsilon)$

Generate an uniformly distributed random number $\xi \in [0,1)$;

Set $j = 1$;

While $(\xi > \sum_{s=1}^{j} p_{index,s})$

$j = j + 1;$

End

$W = W * sign(h_{index,j}) * hsum(index);$

$X = X + W * b_j;$

$index = j;$

End

return X;

For $x(1), \quad \varepsilon = 0.002$

$$\mathrm{H} = \begin{bmatrix} 0.1 & 0.45 & 0.225 \\ -0.15 & 0.1 & -0.3 \\ -0.18 & 0.36 & 0.1 \end{bmatrix} \qquad b = \begin{bmatrix} 0.225 \\ 1.35 \\ 0.72 \end{bmatrix}$$

$$hsum = \begin{bmatrix} 0.775 \\ 0.55 \\ 0.64 \end{bmatrix}$$

$$P = \begin{bmatrix} 0.129032 & 0.580645 & 0.290323 \\ 0.272727 & 0.181818 & 0.545455 \\ 0.28125 & 0.5625 & 0.15625 \end{bmatrix}$$

index	ξ	W	X	j
1	0.3937	1	0.225	2
2	0.2765	0.775	1.271	2
2		0.426	1.846	
⋮				

Perform one trajectory:

Set initial values

$X = 0, W = 1, index = i;$

Calculate $X = X + W b_{index}$;

While $(|W| > \varepsilon)$

Generate an uniformly distributed random number $\xi \in [0,1)$;

Set $j = 1$;

While $(\xi > \sum_{s=1}^{j} p_{index,s})$

$j = j + 1$;

End

$W = W * sign(h_{index,j}) * hsum(index)$;

$X = X + W * b_j$;

$index = j$;

End

return X;

The random walk terminates after 13 moves

For $x(1)$, $\varepsilon = 0.002$

$$\mathrm{H} = \begin{bmatrix} 0.1 & 0.45 & 0.225 \\ -0.15 & 0.1 & -0.3 \\ -0.18 & 0.36 & 0.1 \end{bmatrix} \quad b = \begin{bmatrix} 0.225 \\ 1.35 \\ 0.72 \end{bmatrix}$$

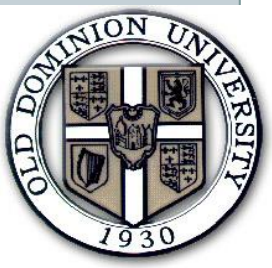
$$hsum = \begin{bmatrix} 0.775 \\ 0.55 \\ 0.64 \end{bmatrix}$$

$$P = \begin{bmatrix} 0.129032 & 0.580645 & 0.290323 \\ 0.272727 & 0.181818 & 0.545455 \\ 0.28125 & 0.5625 & 0.15625 \end{bmatrix}$$

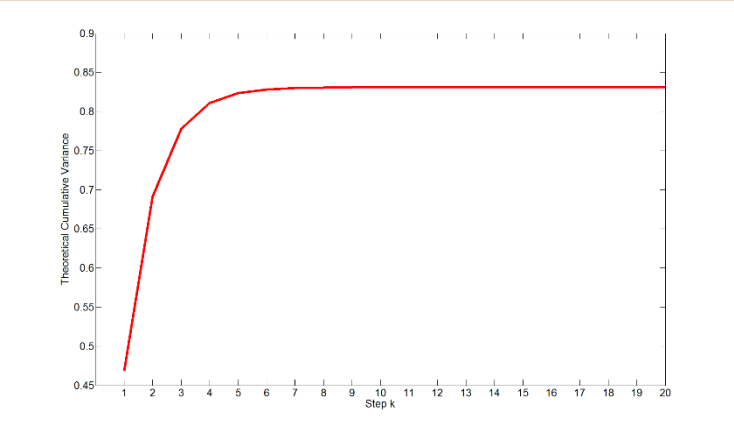
#	index	ξ	W	X	j
1	1	0.3937	1	0.225	2
2	2	0.2765	0.775	1.271	2
3	2	0.7866	0.426	1.846	3
	⋮				
13	1		-0.001	1.837	

Convergence of Ulam-von Neumann Scheme

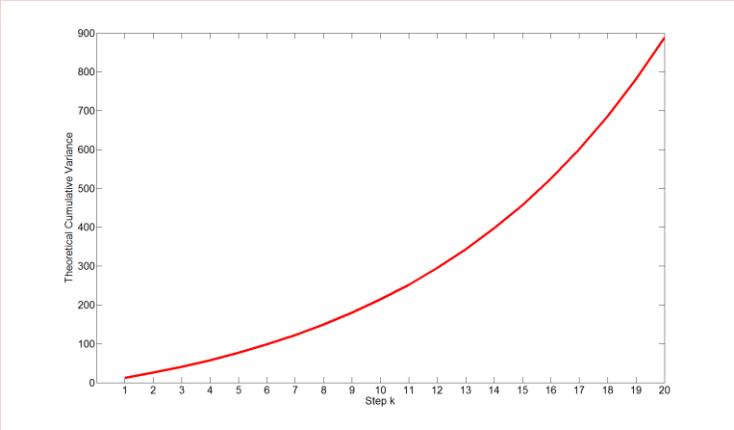
- Confusions in the literature
  - "If $||H||>1$, the Monte Carlo method breaks down."
    - [Curtiss, 1956]
    - [Hammersley & Handscomb, 1964]
  - If the underlying Neumann series converge, i.e., $\rho(H) < 1$
    - [Wang et al., 2008]
    - [Srinivasan, 2010]
    - [Estep, 2009]
    - [Ginting, 2010]
    - etc.
  - $\rho(H) \leq ||H||$



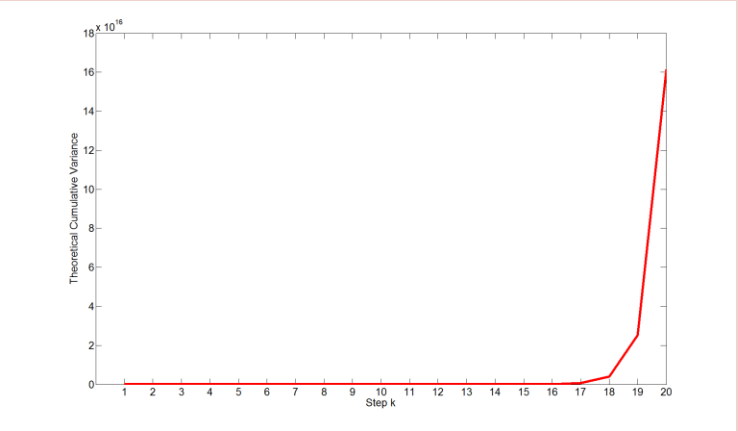
Case Study 1

Case	*H* and *P*	Conditions	Converged?	$\mathrm{Var}\left(\sum_k X(\gamma_k)\right)$
1	$H = \begin{bmatrix} 0.1 & 0.3 \\ 0.3 & -0.05 \end{bmatrix}$ $P = \begin{bmatrix} 0.1 & 0.3 \\ 0.3 & 0.05 \end{bmatrix}$	$\|H\| < 1$ $\rho(H) < 1$	Yes	

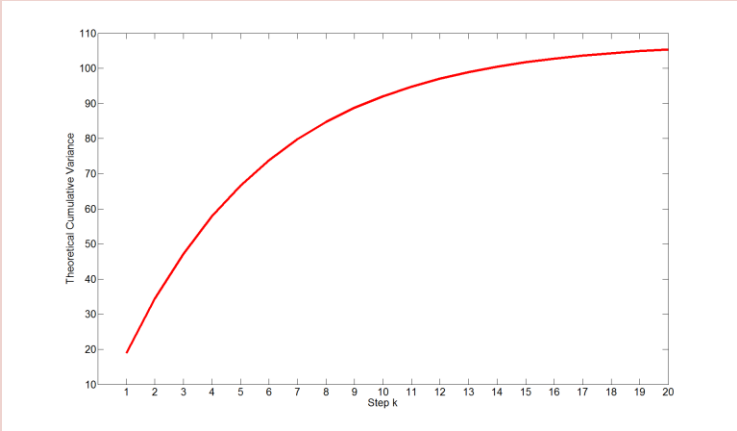
Case Study 2

Case	*H* and *P*	Conditions	Converged?	$\mathrm{Var}\left(\sum_k X(\gamma_k)\right)$
2	$H = \begin{bmatrix} 0.1 & 0.3 \\ 0.3 & -0.05 \end{bmatrix}$ $P = \begin{bmatrix} 0.009 & 0.891 \\ 0.8 & 0.1 \end{bmatrix}$	$\|H\| < 1$ $\rho(H) < 1$	No	

Case Study 3

Case	*H* and *P*	Conditions	Converged?	$\mathrm{Var}\left(\sum_k X(\gamma_k)\right)$
3	$H = \begin{bmatrix} 0.8 & 0.35 \\ 0.1 & -0.01 \end{bmatrix}$ $P = \begin{bmatrix} 0.1 & 0.8 \\ 0.7 & 0.2 \end{bmatrix}$	$\|H\| > 1$ $\rho(H) < 1$	No	

Case Study 4

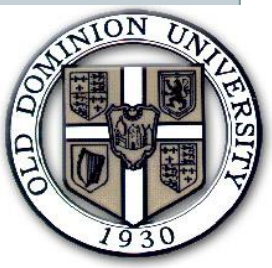
Case	*H* and *P*	Conditions	Converged?	$\mathrm{Var}\left(\sum_k X(\gamma_k)\right)$
4	$\mathbf{H} = \begin{bmatrix} 0.8 & 0.35 \\ 0.1 & -0.01 \end{bmatrix}$ $\mathbf{P} = \begin{bmatrix} 0.8 & 0.1 \\ 0.7 & 0.2 \end{bmatrix}$	$\|\mathbf{H}\| > 1$ $\rho(\mathbf{H}) < 1$	Yes	

Necessary and Sufficient Condition of Convergence

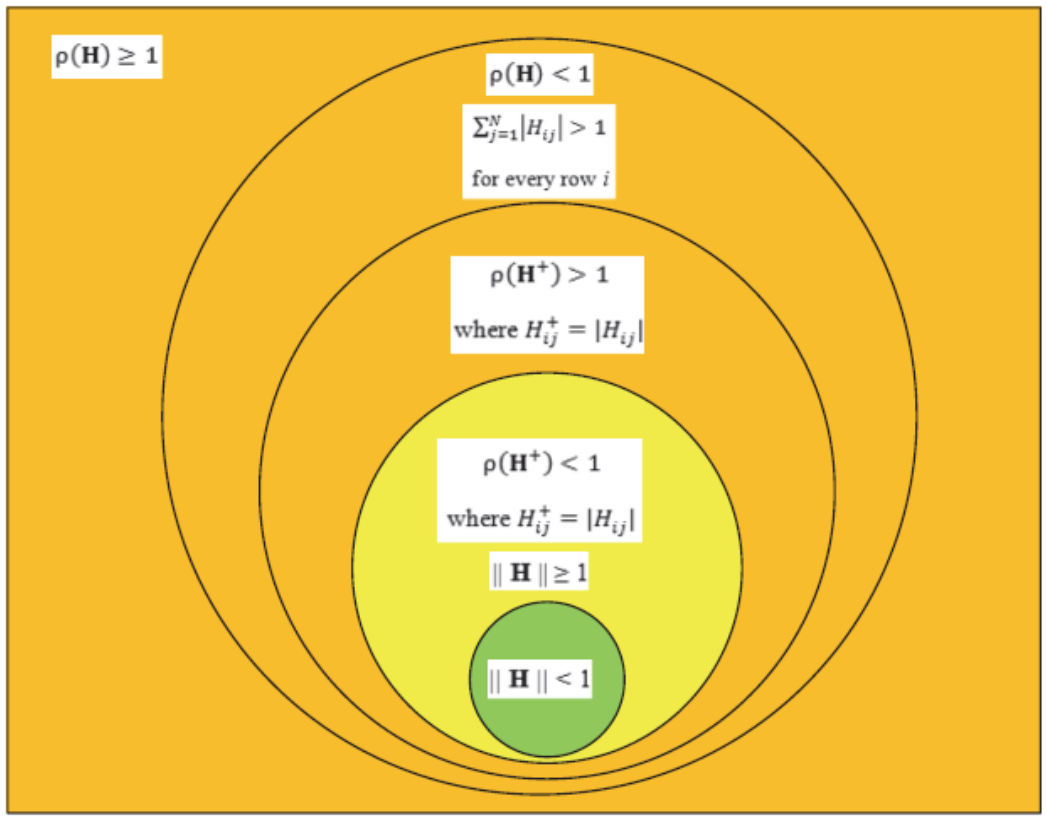
- Convergence depends on not only the coefficient matrix H, but also the transition matrix P
- Necessary and Sufficient Condition

> "Given an $N \times N$ nonsingular matrix H such that $\rho(H)<1$, a nonzero vector b, and a transition matrix P, the necessary and sufficient condition for convergence of the Monte Carlo linear solver using the Ulam-von Neumann scheme is $\rho(H^*) < 1$, where H^* is an $N \times N$ matrix such that $H^*_{ij} = H^2_{ij}/P_{ij}$."

[Ji, Mascagni, Li, SIAM J. NUMER. ANAL., 2013]



Finding the Transition Matrix

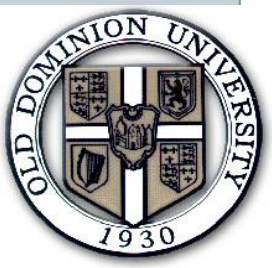


[Ji, Mascagni, & Li, SIAM J. NUMER. ANAL., 2013]

Pros and Cons of Markov Chain Monte Carlo Algorithms

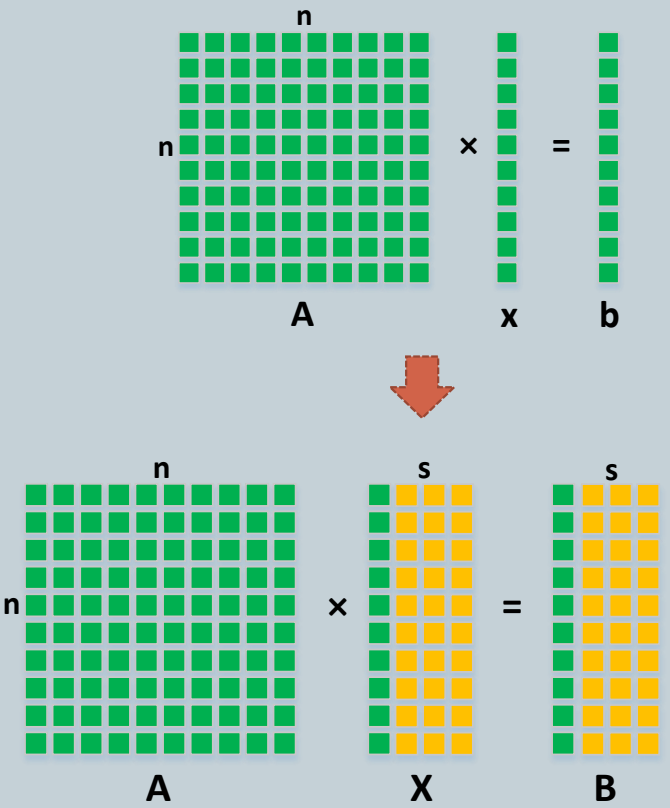
- Advantages
  - Matrix structure (symmetric or non-symmetric, dense or sparse) is not important
  - Don't need to access all elements of the matrix at a time
  - Naturally parallel
  - Can estimate a single element instead of the whole solution vector
  - Robustness
    - A few errors wouldn't hurt the computation
- Disadvantages
  - Slow convergence
    - $O(N^{-1/2})$ where N is the number of samples
  - Costly to provide solutions in high precision
  - Limited applicability

[Li and Mascagni, IJHPCA, 2003]



Reduce Passes

- From CG to Block CG
  - Less Passes of Coefficient Matrix A
    - Theoretical Convergence Steps
      - CG: n
      - BCG: $[n/s]$
  - Better Convergence
  - More Suitable for Parallel/Distributed Computing Paradigm



[O'Leary, 1980]

Rank Deficiency

Original BCG Method

Input: matrix $A \in \mathbb{R}^{n \times n}$, matrix $B \in \mathbb{R}^{n \times s}$, initial guess $X_0 \in \mathbb{R}^{n \times s}$, preconditioner $M \in \mathbb{R}^{n \times n}$, tolerance $tol \in \mathbb{R}$ and maximum number of iterations $maxit \in \mathbb{R}$

Output: an approximate solution $X_{sol} \in \mathbb{R}^{n \times s}$

$R_0 = B - AX_0$

$Z_0 = MR_0$

$P_0 = Z_0\gamma_0$

For $i = 0, \ldots, maxit$

$\alpha_i = (P_i{}^T A P_i)^{-1} \gamma_i{}^T (Z_i{}^T R_i)$

$X_{i+1} = X_i + P_i\alpha_i$

$R_{i+1} = R_i - AP_i\alpha_i$

If converged, **then** stop.

$Z_{i+1} = MR_{i+1}$

$\beta_i = \gamma_i{}^{-1} (Z_i{}^T R_i)^{-1} (Z_{i+1}{}^T R_{i+1})$

$P_{i+1} = (Z_{i+1} + P_i\beta_i)\gamma_{i+1}$

End

$X_{sol} = X_{i+1}$

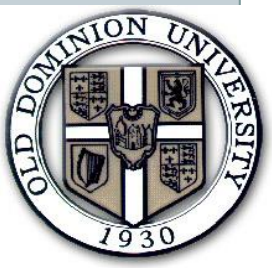
Potential Breakdown

Two or more vector components in the initial block residue R_0 are linearly dependent

Convergence of one or more vector components in the block residue R_i

Two or more vector components in the block residue R_i at iteration i become linearly dependent

[O'Leary, 1980]



Breakdown-Free BCG

Original BCG Algorithm

Input: matrix $A \in \mathbb{R}^{n\times n}$, matrix $B \in \mathbb{R}^{n\times s}$, initial guess $X_0 \in \mathbb{R}^{n\times s}$, preconditioner $M \in \mathbb{R}^{n\times n}$, tolerance $tol \in \mathbb{R}$ and maximum number of iterations $maxit \in \mathbb{R}$

Output: an approximate solution $X_{sol} \in \mathbb{R}^{n\times s}$

$R_0 = B - AX_0$
$Z_0 = MR_0$
$P_0 = Z_0\gamma_0$
For $i = 0, \dots, maxit$
$\alpha_i = \left(P_i{}^T AP_i\right)^{-1}\gamma_i{}^T\left(Z_i{}^T R_i\right)$
$X_{i+1} = X_i + P_i\alpha_i$
$R_{i+1} = R_i - AP_i\alpha_i$
If converged, **then** stop.
$Z_{i+1} = MR_{i+1}$
$\beta_i = \gamma_i{}^{-1}\left(Z_i{}^T R_i\right)^{-1}\left(Z_{i+1}{}^T R_{i+1}\right)$
$P_{i+1} = (Z_{i+1} + P_i\beta_i)\gamma_{i+1}$
End
$X_{sol} = X_{i+1}$

Breakdown-Free BCG Algorithm

Input: matrix $A \in \mathbb{R}^{n\times n}$, right hand side matrix $B \in \mathbb{R}^{n\times s}$, initial guess $X_0 \in \mathbb{R}^{n\times s}$, preconditioner $M \in \mathbb{R}^{n\times n}$, tolerance $tol \in \mathbb{R}$ and maximum number of iterations $maxit \in \mathbb{R}$

Output: an approximate solution $X_{sol} \in \mathbb{R}^{n\times s}$

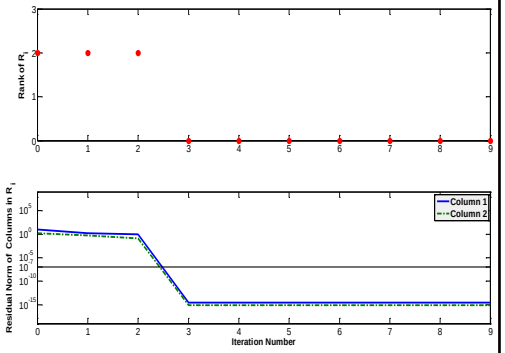
$R_0 = B - AX_0$
$Z_0 = MR_0$
$\tilde{P}_0 = orth(Z_0)$
For $i = 0, \dots, maxit$
$Q_i = A\tilde{P}_i$
$\tilde{\alpha}_i = \left(\tilde{P}_i{}^T Q_i\right)^{-1}\left(\tilde{P}_i{}^T R_i\right)$
$X_{i+1} = X_i + \tilde{P}_i\tilde{\alpha}_i$
$R_{i+1} = R_i - Q_i\tilde{\alpha}_i$
If converged, **then** stop.
$Z_{i+1} = MR_{i+1}$
$\tilde{\beta}_i = -\left(\tilde{P}_i{}^T Q_i\right)^{-1}\left(Q_i{}^T Z_{i+1}\right)$
$\tilde{P}_{i+1} = orth\left(Z_{i+1} + \tilde{P}_i\tilde{\beta}_i\right)$
End
$X_{sol} = X_{i+1}$

[Ji and Li, 2014]

Results of Breakdown-Free BCG

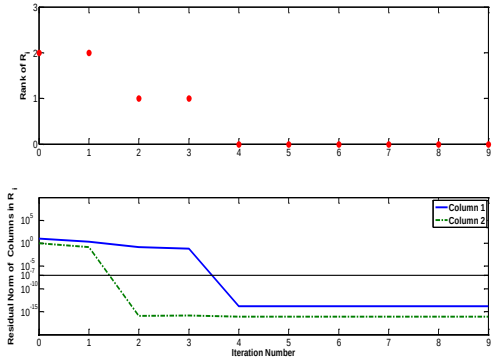
$$A = \begin{bmatrix} 15 & 5 & 4 & 3 & 2 & 1 \\ 5 & 35 & 9 & 8 & 7 & 6 \\ 4 & 9 & 46 & 12 & 11 & 10 \\ 3 & 8 & 12 & 50 & 14 & 13 \\ 2 & 7 & 11 & 14 & 19 & 15 \\ 1 & 6 & 10 & 13 & 15 & 45 \end{bmatrix}$$

$$R_0 = \begin{bmatrix} 1 & 0.537266261211281 \\ 2 & 0.043775211060964 \\ 3 & 0.964458562037146 \\ 4 & 0.622317517840541 \\ 5 & 0.552735938776748 \\ 6 & 0.023323943544997 \end{bmatrix}$$



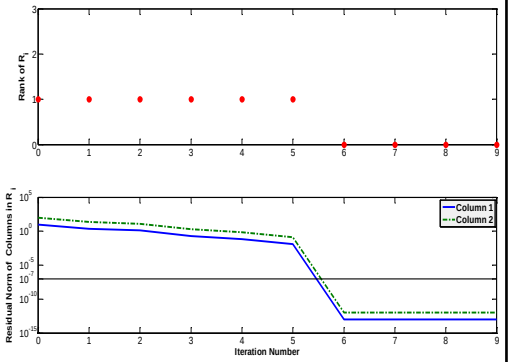
Case 1: The block residue R_i without rank deficiency

$$R_0 = \begin{bmatrix} 1 & 0.027212780358615 \\ 2 & 0.117544343373396 \\ 3 & 0.140184539179715 \\ 4 & 0.605659566833592 \\ 5 & 0.323269030695212 \\ 6 & 0.590821508384101 \end{bmatrix}$$



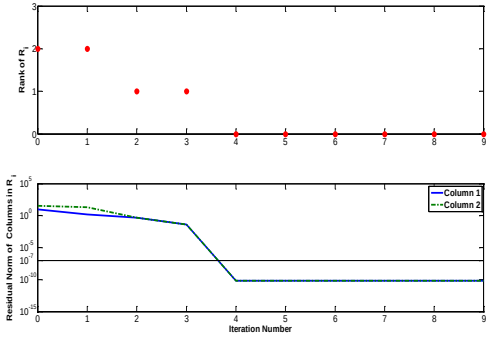
Case 3: Convergence of one or more but not all columns in the block residue R_i

$$R_0 = \begin{bmatrix} 1 & 10 \\ 2 & 20 \\ 3 & 30 \\ 4 & 40 \\ 5 & 50 \\ 6 & 60 \end{bmatrix}$$



Case 2: Columns in the initial block residual R_0 are linearly dependent

$$R_0 = \begin{bmatrix} 1 & -8.888614458250306 \\ 2 & -10.999025290685955 \\ 3 & -19.339674247091921 \\ 4 & -10.289152668326622 \\ 5 & 18.107579559267656 \\ 6 & -8.930794511222629 \end{bmatrix}$$

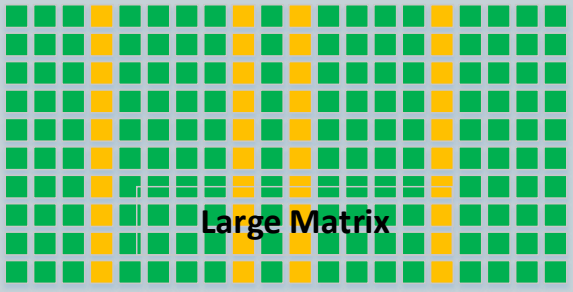


Case 4: Columns in the block residue R_i become linearly dependent during iterations

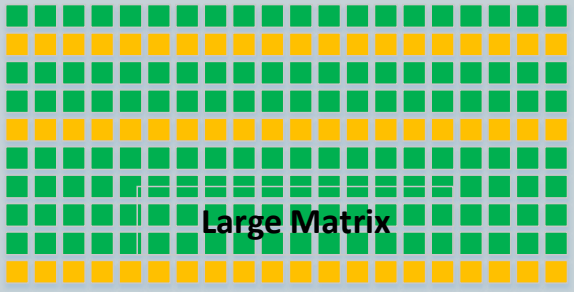
[Ji and Li, 2014]

Randomized Singular Value Decomposition

- Statistically Sampling Large Matrix to Approximate Top-*k* Singular Values/Vectors

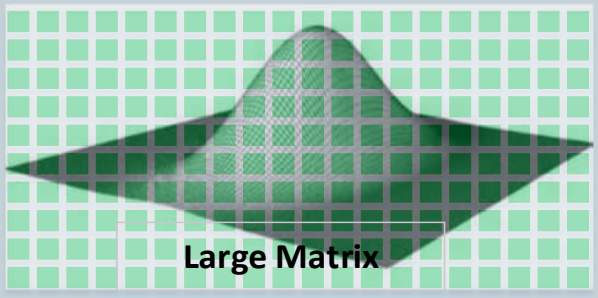


Column Sampling



Row Sampling

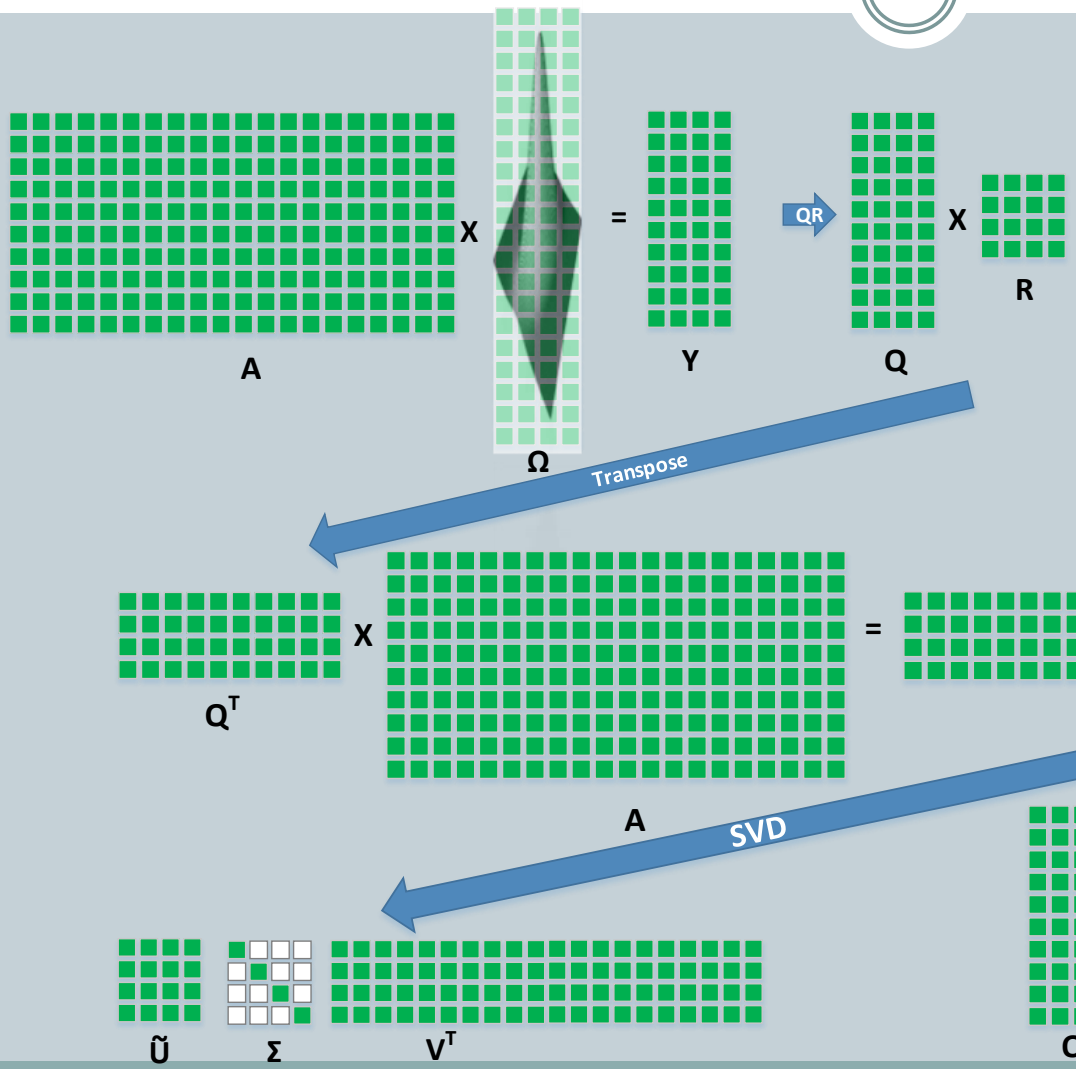
[Drineas et al. 2006]



Gaussian Sampling

[Halko et al. 2011]

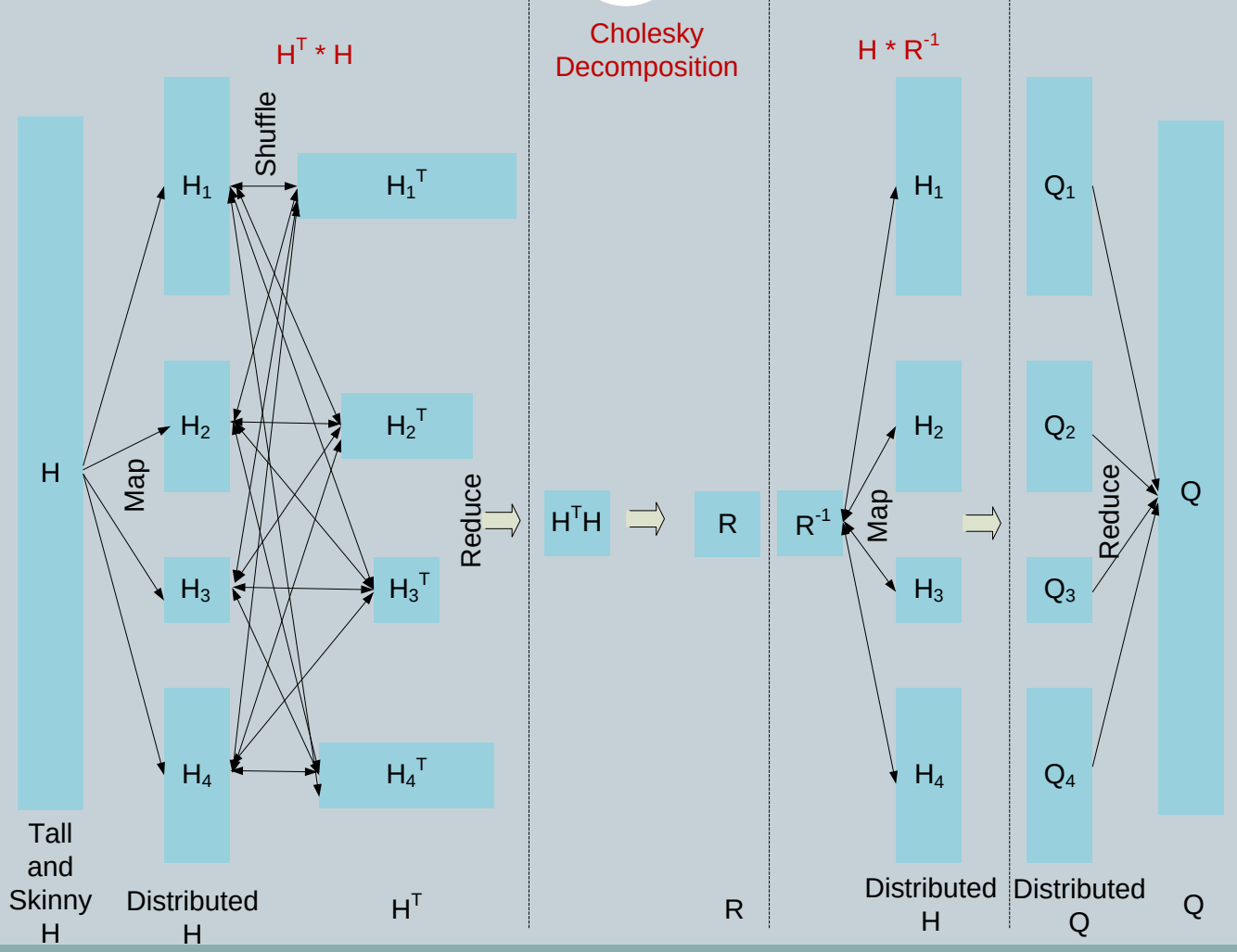
Gaussian Sampling



Two passes of A

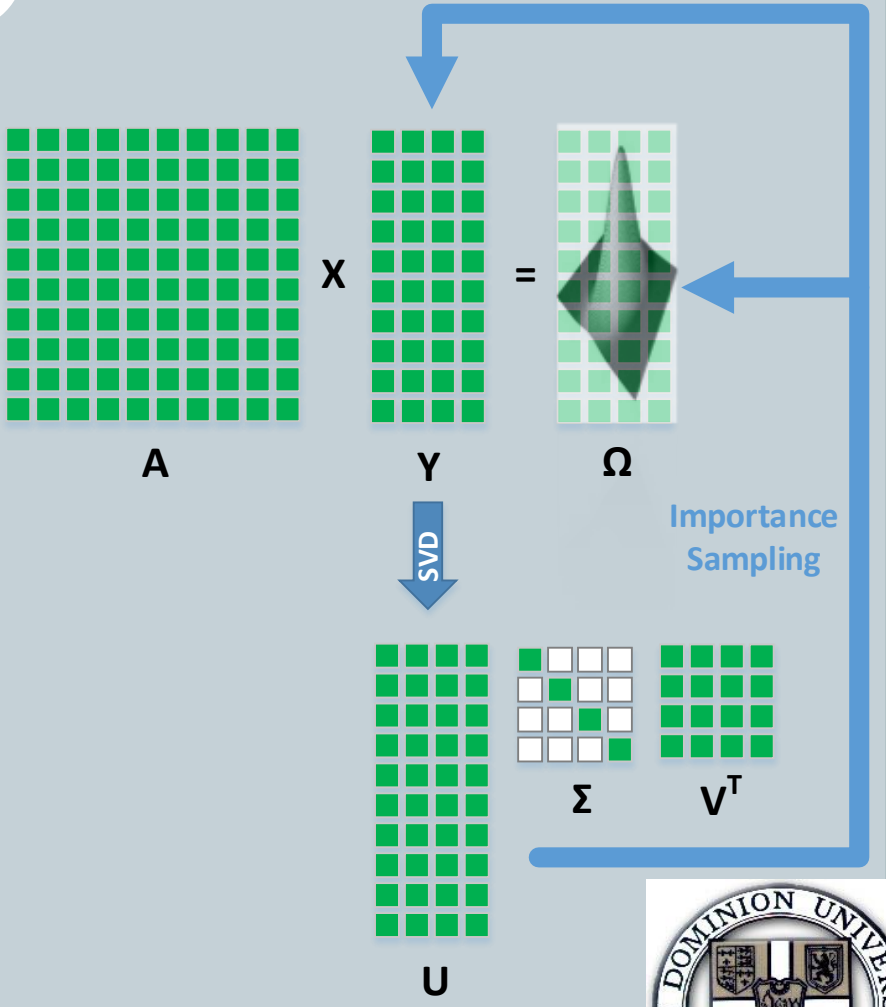
[Halko et al. 2011]

Tall-and-Skinny Matrices



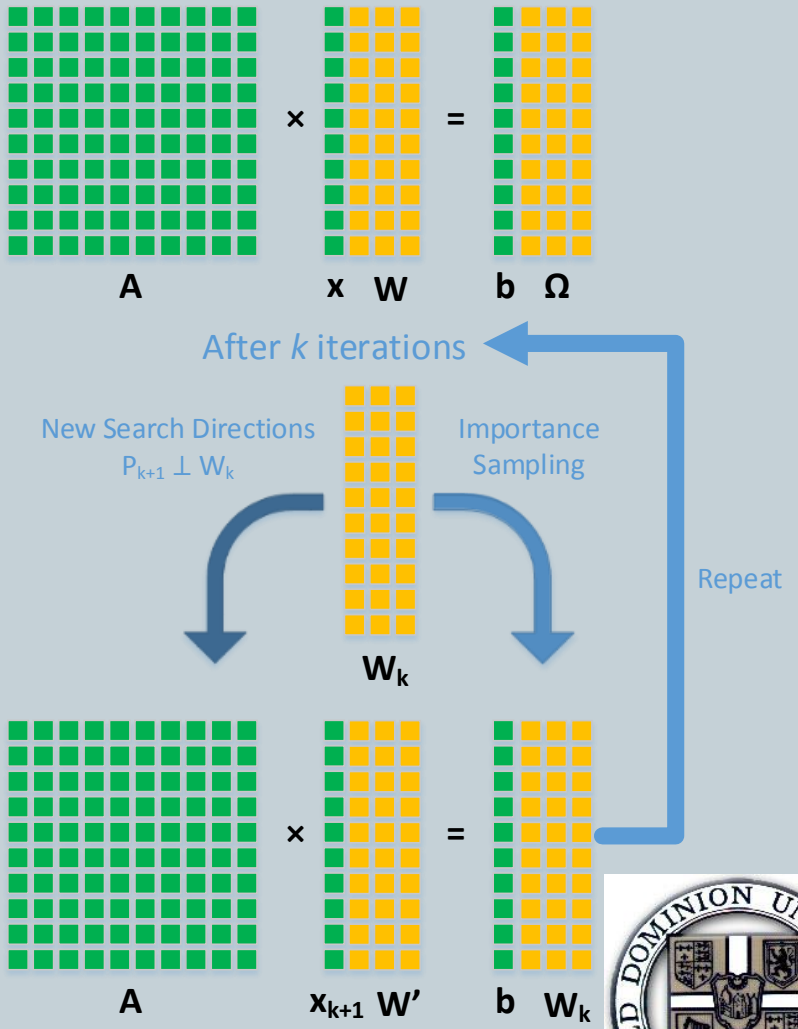
Inverse Randomized SVD

- Inverse Process of Randomized SVD
  - Approximate low-*k* singular values/vectors (eigenvalues/eigenvectors)

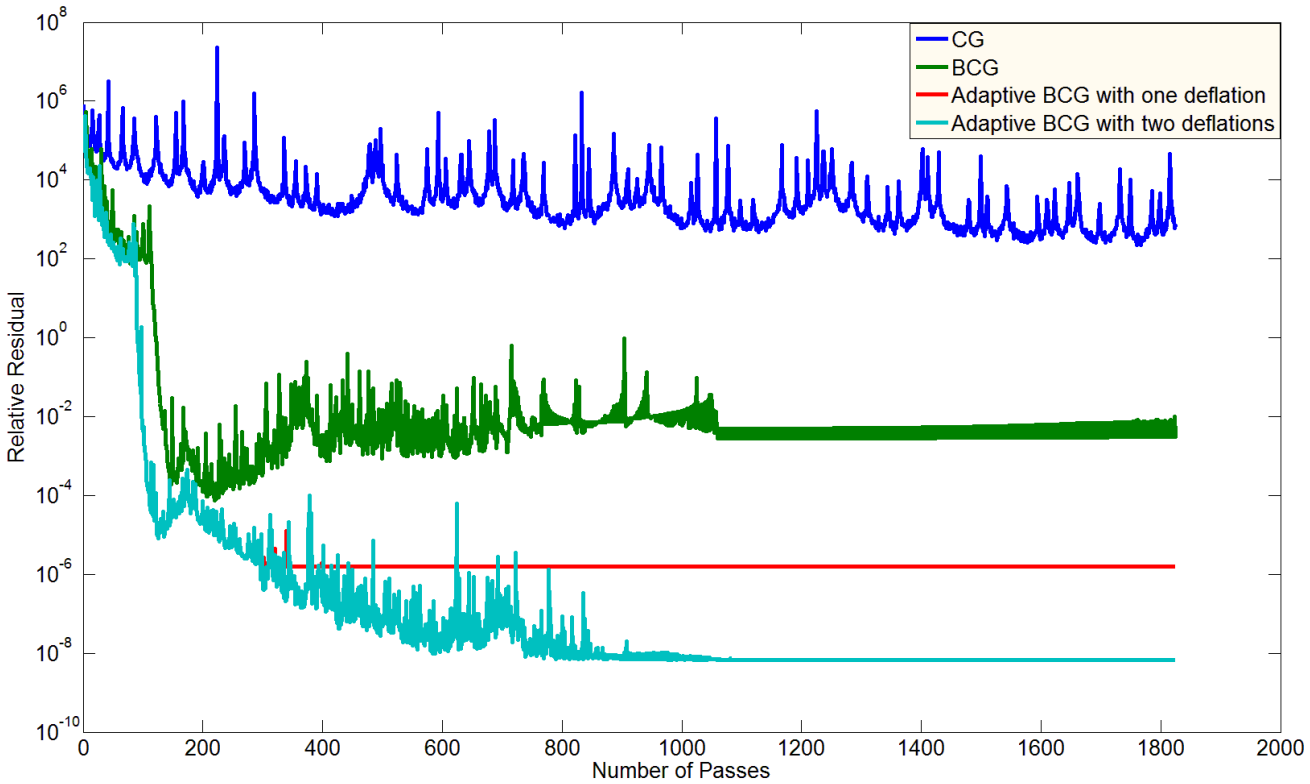


BCG with Adaptive Deflation

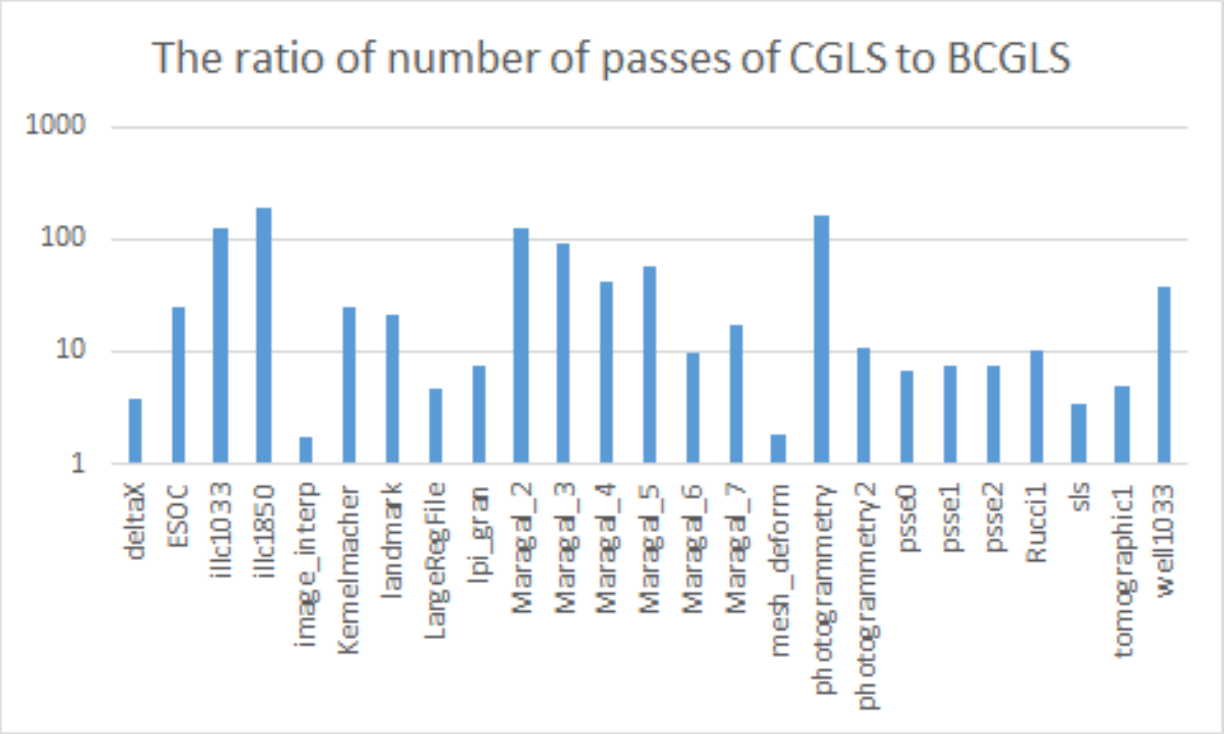
- BCG with Adaptive Deflation
  - BCG + inverse randomized SVD
  - Deflation
    - Approximated low-*k* eigenvectors
  - Importance Sampling
    - Refined approximated low-*k* eigenvectors
- Advantages
  - Reduced passes
  - Fixed memory bound
  - Accelerated convergence
  - No preconditioning is needed



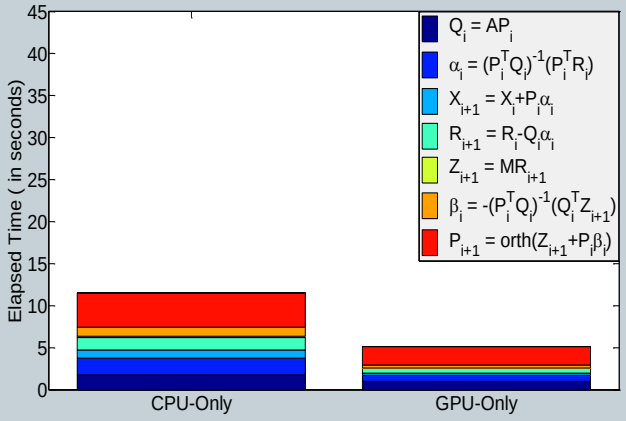
Performance of BCG with Adaptive Deflation



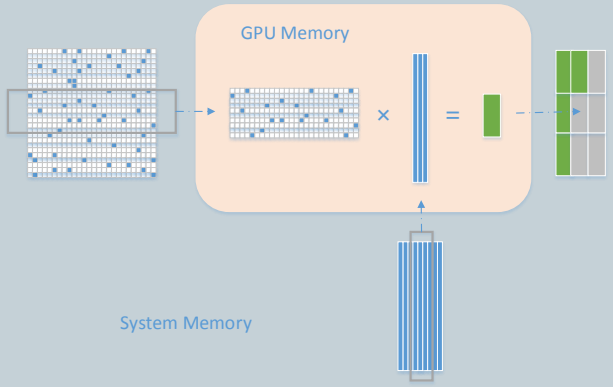
Number of Passes

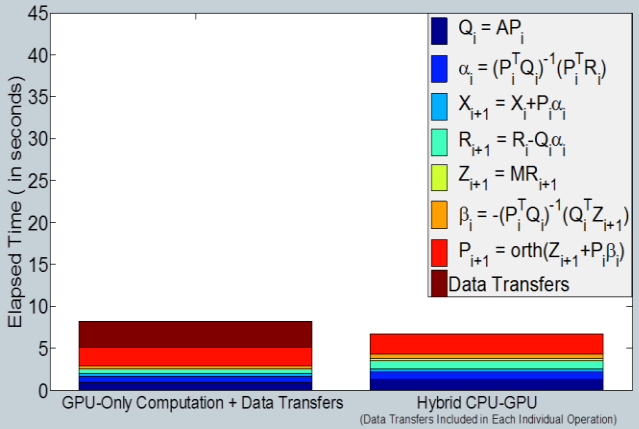


BCG with Adaptive Deflation using GPU as Co-Processor



2.63 speedup
GPU (K20) Peak Performance = 1134 GFLOPS
CPU (Xeon E5) Peak Performance = 345 GFLOPS



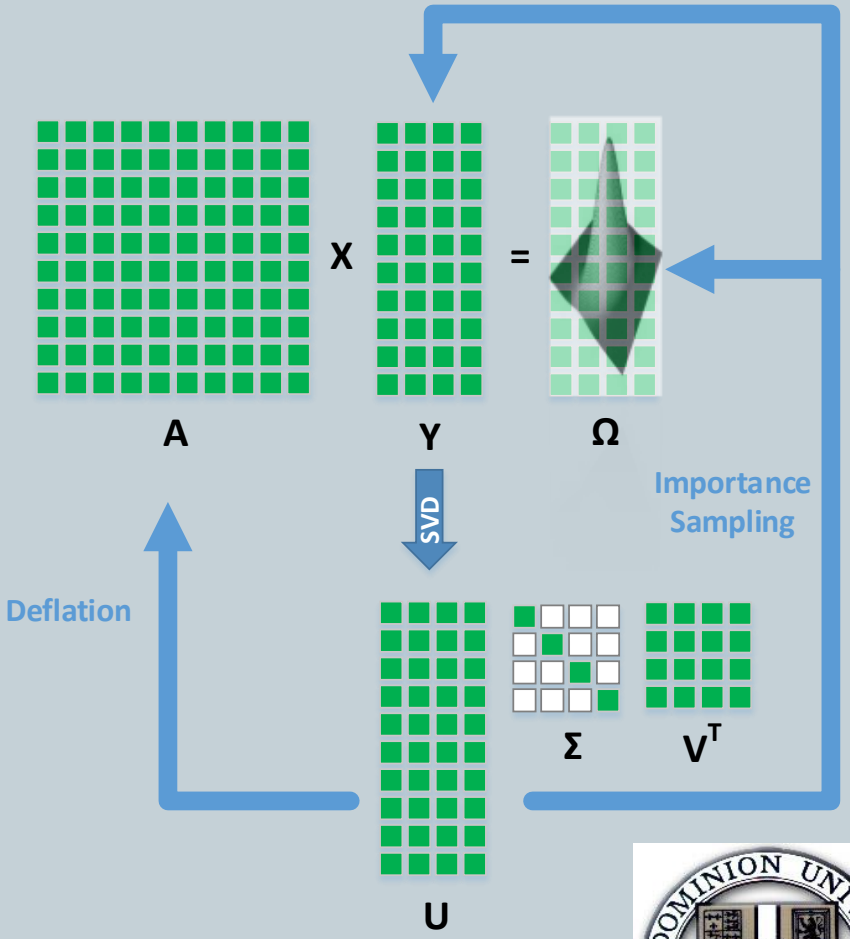


Data transferring time is hidden

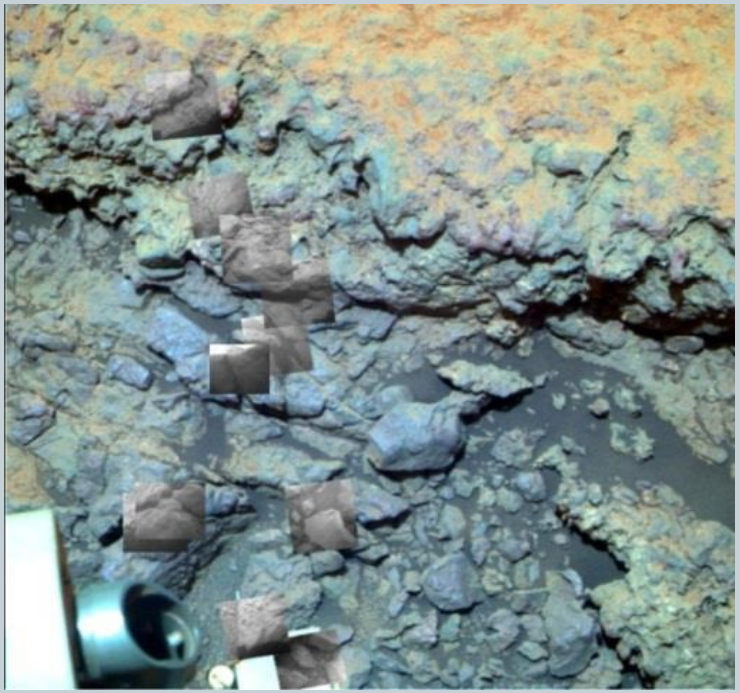
[Ji, Sosonkina, Li, Co-HPC, 2014]

Finding the Smallest Eigenvalues/Eigenvectors

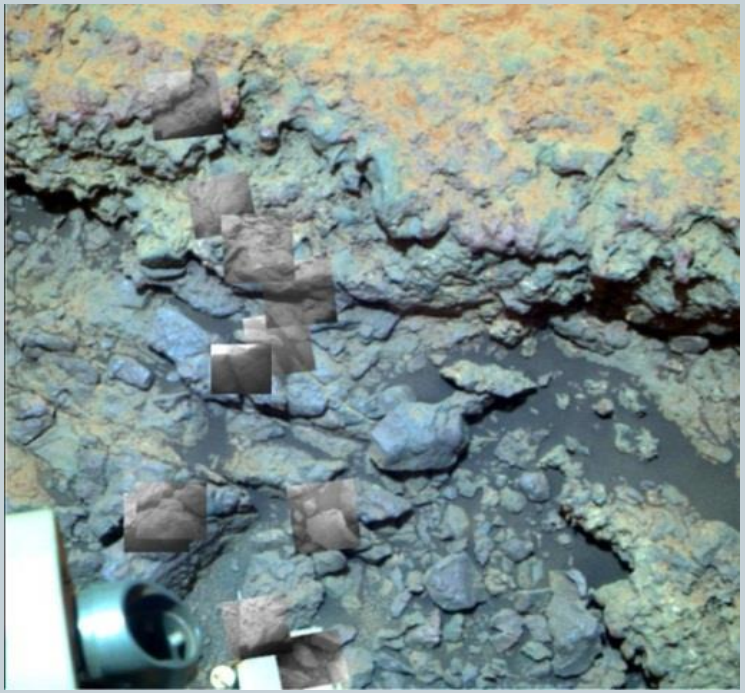
- Find the smallest *k* eigenvalues/eigenvectors using BCG with adaptive deflation
  - Approximating the Smallest *k* Eigenvalues/Eigenvectors
  - Reduced Access to Big Matrix *A*
  - Memory Efficient
  - Accelerated Convergence with Deflation



Mars Image Compression



(a) Original Image (7680x7671)

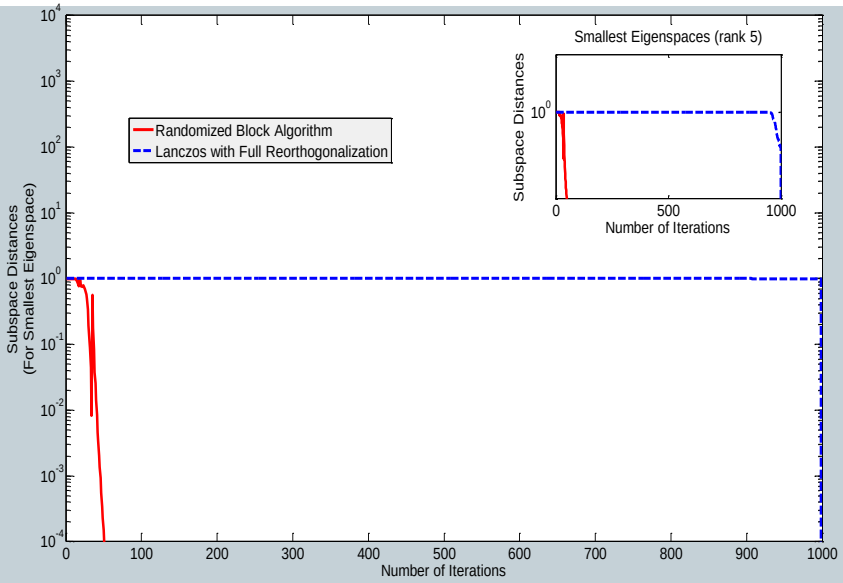


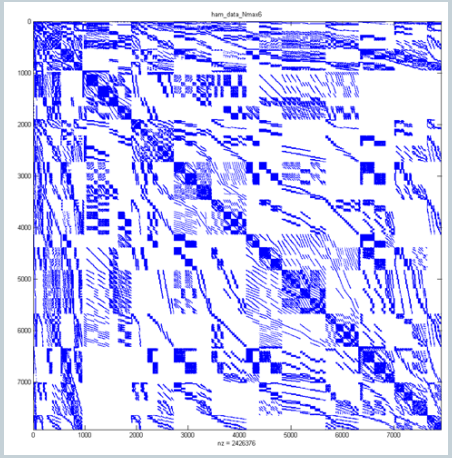
(b) Reproduced Image after Reconstruction (Err = 0.99%)

The overall volume of which is less than 1/8 of the original image with less than 1% error in Frobenius norm. (k = 472)

Application in *ab initio* Nuclear Physics Computation

- Expand wave function in basis states
- Express Hamiltonian matrix ***H*** in basis
- Diagonalize Hamiltonian matrix
- Complete basis → exact result
  - Caveat: complete basis is infinite dimensional
- In practice
  - Truncate basis to obtain many body problem with 2-body interactions
  - Study behavior of observables as function of truncation
- Computational challenges
  - Construct large sparse symmetric real matrix H
  - Get lowest eigenvalues & eigenvectors





[Collaboration with Dr. Sosonkina]

OLD DOMINION UNIVERSITY 1930

Image Segmentation

- Normalized Cut
  - Graph cut

Given a partition (A, B) of the vertex set V.

$$Ncut(A,B) = \frac{cut(A,B)}{assoc(A,V)} + \frac{cut(A,B)}{assoc(B,V)}$$

$$cut(A,B) = \sum_{u \in A, v \in B} w(u,v) \qquad assoc(A,V) = \sum_{u \in A, t \in V} w(u,t)$$

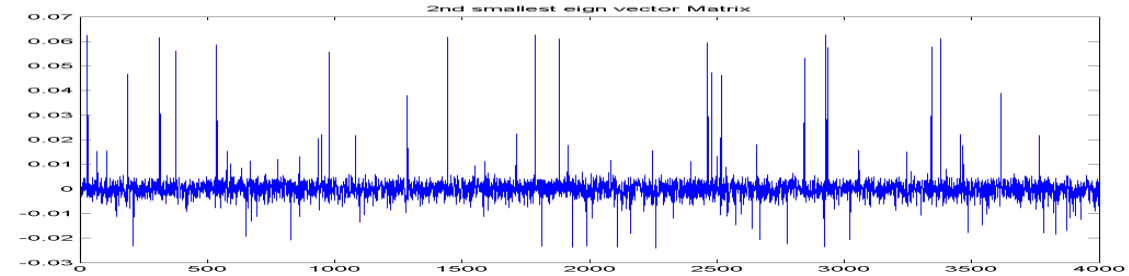
Ncut(A, B) measures similarity between two groups.

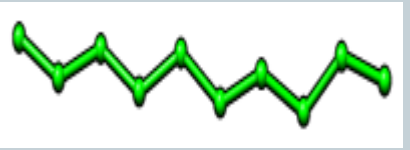
  - Optimal cut

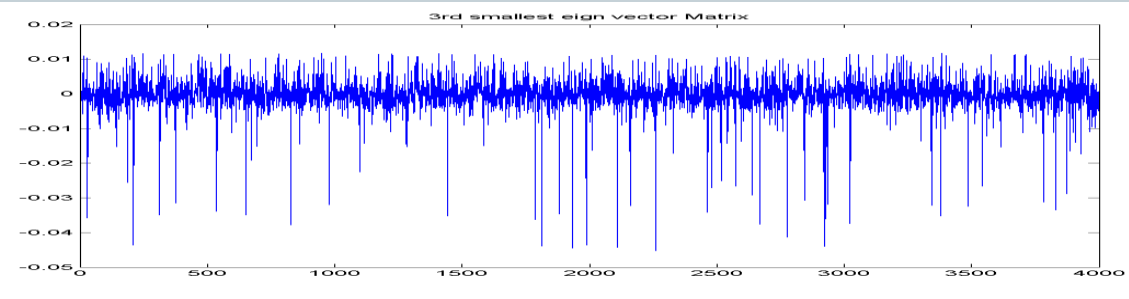
$$MinNcut(G) = \min_y \frac{y^t(D-W)y}{y^tDy}$$

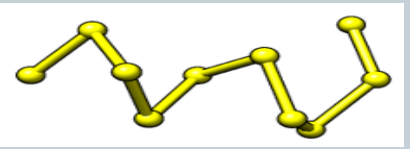
  - Segmentation based on the eigenvectors of the smallest *k* eigenvalues

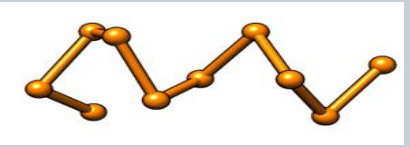
Discovery of Protein Fragment Motifs

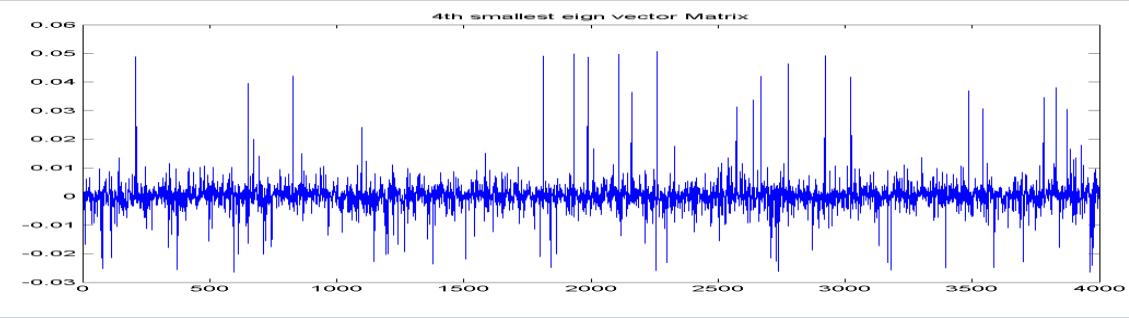


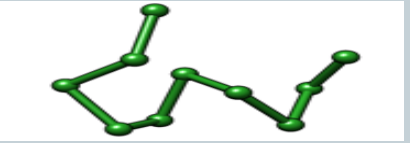


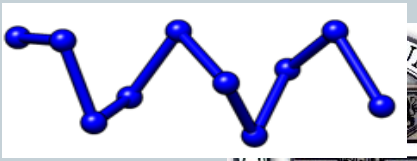






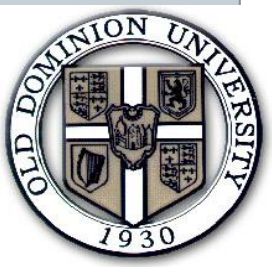






Summary

- Revisit of Ulam-von Neumann Scheme
  - Necessary and Sufficient Condition for Convergence
- Breakdown-Free BCG
  - Avoid Potential Breakdowns due to Rank Deficiency
- BCG with Adaptive Deflation
  - Inverse Randomized SVD for Adaptive Deflation
- Applications
  - Image Compression
  - Image Segmentation
  - Nuclear Physics
  - Protein Fragments Discovery



Research Team @

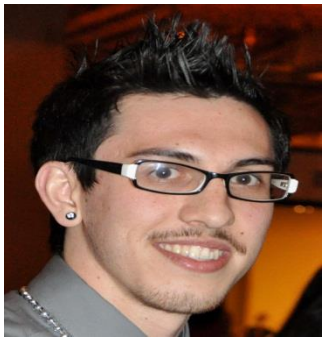
Dr. Ashraf Yaseen

Hao Ji

Wessam Elhefhawy

Dr. Yun Han

Maha Abdelaal



Kyle Wessells

Akeem Edwards

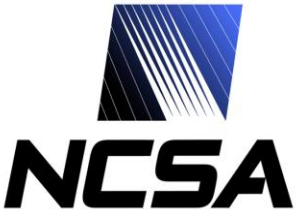
Thomas Goldsmith

Adam Boudion

Acknowledgements

NSF HRD-0450203
NSF EMT CCF-0829382
NSF CAREER CCF-1066471

ODU MSF Seed Grant, 2013
ODU Undergraduate
Research Program, 2011, 2012



NCSA Faculty Fellowship, 2007

Ralph E. Powe Young Faculty Award, 2005
Summer Faculty Participation Program, 2006 and 2008

References

- H. Ji, M. Mascagni, Y. Li, *Convergence Analysis of Markov Chain Monte Carlo Linear Solvers using Ulam-von Neumann Algorithm*, SIAM J. Numer. Anal., 51(4): 2107-2122, 2013.
- H. Ji, Y. Li, *Breakdown-free Block Conjugate Gradient*, SIAM J. Numer. Anal., under review, 2014.
- J. M. Hammersley, D. C. Handscomb, *Monte Carlo Methods*, Chapman and Hall, London, 1964.
- J. H. Curtiss, *A theoretical comparison of the efficiencies of two classical methods and a Monte Carlo method for computing one component of the solution of a set of linear algebraic equations*, in Proceedings of Symposium on Monte Carlo Methods, John Wiley and Sons, New York, pp. 191–233, 1956.
- Q. Wang, D. Gleich, A. Saberi, N. Etemadi, P. Moin, *A monte carlo method for solving unsteady adjoint equations*, J. Comp. Phys., 227(12): 6184-6205, 2008.
- A. Srinivasan, *Monte Carlo linear solvers with non-diagonal splitting*, Mathematics and Computers in Simulation, 80(6): 1133–1143, 2010.
- D. Estep, A. M˚alqvist, S. Tavener, *Nonparametric density estimation for randomly perturbed elliptic problems II: Applications and adaptive modeling*, International Journal for Numerical Methods in Engineering, 80: 846–86, 2009.
- V. Ginting, A. Målqvist, M. Presho, *A Novel Method for Solving Multiscale Elliptic Problems with Randomly Perturbed Data*, SIAM Multiscale Model. Simul., 8(3), 977-996, 2010.
- P. Drineas, R. Kannan, M. W. Mahoney. *Fast Monte Carlo algorithms for matrices II: Computing a low-rank approximation to a matrix*. SIAM J. Comput., 36(1), 2006
- N. Halko, P. G. Martinsson, J. A. Tropp, *Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions*, SIAM Rev., 53(2): 217-288, 2011.
- H. Ji, M. Sosonkina, Y. Li, *An Implementation of Block Conjugate Gradient Algorithm on CPU-GPU Processors*, Proc. of Hardware-Software Co-Design for High Performance Computing (Co-HPC14), 2014.
- H. Ji, Y. Li, *Block least square*, in preparation, 2014.
- H. Ji, Y. Li, *Block conjugate gradient with adaptive deflation using inverse randomized singular value decomposition*, in preparation, 2014.
- D. P. O'Leary, *The block conjugate gradient algorithm and related methods*, Linear Algebra Appl., 29: 293–322, 1980.
- J. Chen, *A deflated version of the block conjugate gradient algorithm with an application to Gaussian process maximum likelihood estimation*, Preprint ANL/MCS-P1927-0811, Argonne National Laboratory, Argonne, IL, 2011.
- G. H. Golub, D. P. O'Leary, *Some history of the conjugate gradient and lanczos methods*, SIAM Rev., 31(1): 50-102, 1989.
- I. T. Dimov, T. T. Dimov, T. V. Gurov, *A new iterative Monte Carlo Approach for Inverse Matrix Problem*, J. Comput. Appl. Math., 92: 15–35, 1998.
- Y. Li, M. Mascagni, *Analysis of Large-scale Grid-based Monte Carlo Applications*, International Journal of High Performance Computing Applications (IJHPCA), 17(4): 369-382, 2003.