juliacon

Markov Chain-Monte Carlo Methods for Linear Algebra using Julia v.1.0

Oscar A. Esquivel-Flores[1] and Héctor Benítez-Pérez[1]

[1]Universidad Nacional Autónoma de México, IIMAS-DISCA.

ABSTRACT

A well known probabilistic method for solving systems of linear algebraic equations (SLAE) is Markov Chain-Monte Carlo (MCMC) method. Some works about this method have focused on estimating the solution vector, inverse matrix of the system and computing efficient preconditioner in order to obtain a good approximate solution. The iterative process, random sampling, compute a Markov Chain with appropriete length, compute the estimators of solutuion represent an exhaustive computational effor for MCMC method as the size of SLAE increases. Some proposals to tackle the above are serial implementation using an efficient programming language and parallel implementation of MCMC on different parallel architecture.`Julia`programming language has captured our attention since it has consolidated, relatively fast, as an excellent tool for scientific computation, therefore we have open a new route in our work by implementing the MCMC in `Julia v.1.0` in order to accelerate the precision of the method without to paralelize it. This work presents our first iteration over MCMC using this alluring programming language.

Proceedings of JuliaCon

Keywords

Julia, Monte Carlo Methods, Linear Algebra

1. Introduction

Solving systems of linear algebraic equations (SLAE) in the form of $Ax = b$ or inverting a real matrix A is of unquestionable importance in many scientific fields. Iterative solvers are used widely to compute the solutions of these systems and such approaches are often the method of choice due to their predictability and reliability when considering accuracy and speed. They are, however, prohibitive for large-scale problems as they can be very time consuming to compute. These methods are dependent on the size of the matrix and so the computational effort grows with the problem size. The complexity of these methods is $O(kn^2)$ for dense matrices in the iterative case and $O(n^3)$ for direct methods with dense matrices while solving SLAE if common elimination or annihilation schemes (e.g. Gaussian elimination, Gauss-Jordan methods) are employed [16]. Therefore, these algorithms often rely on preconditioners to speed up the computations and/or to ensure faster convergence.

Monte Carlo (MC) methods on the other hand can quickly yield a rough estimate of the solution. This is done by performing random sampling of a certain variable whose mathematical expectation is the desired solution. For some problems an estimate is sufficient or even favourable, due to the accuracy of the underlying data. For example we would not need to process data with a higher precision than the one of the input data. In addition, Monte Carlo methods help to qualify the uncertainties while obtaining solution vector or performing matrix inversion. For example, Monte Carlo approaches enable to estimate the non-zero elements of the solution vector or inverse matrix with a given precision, and with certain probability.

Therefore, we concentrate on Monte Carlo method that only require $O(NL)$ steps to find a estimate of vector solution. Here N is the number of Markov chains and L is an estimate of the chain length in the stochastic process. These computations are independent of the matrix size n and also inherently parallel. Note that in order to find the inverse matrix or the full solution vector in the serial case, $O(nNL)$ steps are required.

Depending on the method used to compute the vector solution or inverse matrix, the savings and end-results vary. Compute vector solution or inverse matrix of a very sparse system is quickly, but it is unlikey to improve the quality of the solution. On the other hand, compute the solution vector or inverse matrix of a dense system is computationally expensive and might be time or cost prohibitive. Therefore, finding a good solution for both problems that is computationally efficient, while still providing substantial improvement to the iterative solution process, is a worthwile research topic.

The next section 2 gives and overview of related work. Monte Carlo methods, and the specific matrix inversion algorithm that is discussed are presented in Section 3. Section 4 provides background information on the underlying computing systems and the experimental set ups. Results and findings from experiments with matrices of varying sizes and sparsity, as well as outcomes from running the SLAE solvers with Monte Carlo inverse matrix are discussed in Section 5. The last Section 6 consists of the conclusion and outlines the future work.

2. Related Work

Research efforts in the past have been directed towards optimizing the approach of computing the solution of SLAE through sparse approximate inverse matrix.

The Monte Carlo code shown in this paper is part of a bigger family, it includes a serial algorithm for finding the vector solution of sparse systems of linear algebraic equations. The proposed Monte Carlo algorithm has been developed and enhanced upon in the last decades, and several key advances in serial and parallel Monte Carlo methods for solving such problems have been made [1, 4, 8, 11, 12, 14].

In the past there have been differing aproaches and advances towards a parallelisation of Monte Carlo methods using different

kinds of programming languages in order to get a efficient vector solution and efficient matrix inversion used as preconditioner.
Several improvements of the Monte Carlo methods have been presented, some approaches use quasi-random sequences in order to increase the rate of convergence of stochastic simulation algorithms [5, 3]. Recently a new walk on equations Monte Carlo algorithm for solving SLAEs is proposed, this algorithm relies on a non-discounted sum of an absorbed random walk [13].
Recently parallel approaches of MCMC method have been made using advanced accelerator architectures [17].

3. Monte Carlo Approach

Monte Carlo methods are probabilistic methods, that use random numbers to either simulate a stochastic behaviour or to estimate the solution of a problem. Method involves many independent samples used to estimate the solution, accuracy could be better when this sample increases, this consideration stress the computational effort.

3.1 General MCMC method for solving SLAE

The following procedure has been presented by Fathi and Hassanzadeh in [15], they provide the basic background of the Monte Carlo Methods for solving SLAE and some modifications on the hybrid MC method to obtain accurate Matrix Inversion.
Assume that the System of Linear Algebraic Equations has the following form:

$$Ax = b \tag{1}$$

where $A \in \mathbb{R}^{n\times n}$ is a nonsingular matrix and $x \in \mathbb{R}^n$ is a solution vector, $B \in \mathbb{R}^n$ is a known vector.
Consider the separation of matrix A as $A = M - N$ and subtitute in Equation (1, as shown in [14] the system could be transformed in the following iterative scheme:

$$x^{(k+1)} = Tx^{(k)} + f, k = 0, 1, 2, ... \tag{2}$$

where $T = M^{-1}N$ and $f = M^{-1}b$.
The outcome of reducing the norm $\|T\| < 1$ (for a some matrix norm $\|\cdot\| < 1$) produces the convergence of iterative relation (2) to exact solution of (1) regardless of the initial vector x^0 and thus reducing the number of Markov chains requiered to reach a given precision.
Assume that $\|T\| < 1$ and system is tranformed in (2) consider a homogeneous discrete Markov chain

$$\gamma : r_0 \to r_1 \to \dots r_k \to \cdots,$$

where $r_i\, i = 1, 2, ..., k$ belongs to the state space $R = 1, 2, ..., n$, which is built using random walks on the indices of T in order to sample the solution of (1).
Random walks are based on transition probabilities, wich requires the intital probability which make the initial probability vector $p \in \mathbb{R}^n$ (starting point of random walks) and transition probability matrix $P \in \mathbb{R}^{n\times n}$ (transition from the current point to the next point) to satisfy the following conditions:

$$\begin{aligned} &p_{ij} \geq 0, \quad \sum_{j=1}^{n} p_{ij} = 1, \quad \text{if } t_{ij} \neq 0 \quad \text{then } p_{ij} \neq 0 \\ &p_i \geq 0, \quad \sum_{i=1}^{n} p_i = 1, \quad \text{if } h_i \neq 0 \quad \text{then } p_i \neq 0 \end{aligned} \tag{3}$$

MCMC method is considered with uniform transition probability (UM) $P_{ij} = \frac{1}{n}$. Probability $P_{ij} = \frac{|T_{ij}|}{\sum_{j=1}^{n}|T_{ij}|}$ hs been used in Almost Optimal MCMC method (MAO) which satisfies transition conditions (3).
The distribution p is acceptable for a given nonzero vector h, and that the distribution P is acceptable for matrix T [9]. The role of h is restricted to the construction of the initial probability.
Fathi and Hassanzadeh [15] define the following unbiased estimator for the stochastic trajectory γ:

$$X(\gamma) = \sum_{m=0}^{\infty} W_m f_{r_m} \tag{4}$$

where $W_m = W_{m-1}w_{r_{m-1}r_m}, w_{r_{m-1}r_m} = \frac{t_{r_{m-1}r_m}}{p_{r_{m-1}r_m}}$ and $W_0 = \frac{h_{r_0}}{p_{r_0}}$ so that mathematical expectation of $X(\gamma)$ is the Euclidean inner product $\langle h, x\rangle$, *i.e.* $E[X_i(\gamma)] = x_i$ (see [13] and references therein).
Under condition $\|T\| < 1$ the corresponding Neumann series converges for any given h:

$$x = \sum_{k=0}^{\infty} T^k f \tag{5}$$

Thus to find an arbitrary component of the solution, for example the $i - th$ component of x, it should choose $h = e_i = (\underbrace{0, ..., 1}_{i}, ..., 0)$ and $p_i = 1$. They represent the single component of x in the form:

$$x_i = f_i + \sum_{m=1}^{\infty}\sum_{r_1=1}^{n}\sum_{r_2=1}^{n}\cdots\sum_{r_m=1}^{n} t_{ir_1}t_{r_1r_2}\dots t_{r_{m-1}r_m}f_{r_m} \quad , \tag{6}$$

it can be proved that mathematical expectation of $X_i(\gamma) = \sum_{m=0}^{\infty} W_m f_{r_m}(h = e_i, p_i = 1)$ is the $i-$th component of the solution, i.e., $E[X_i(\gamma)] = x_i$ [13].
Using the following notation for the partial sum $X_i(\gamma_k) = \sum_{m=0}^{k} W_m f_{r_m}$ on finite Markov chain $\gamma : r_0 \to r_1 \to \dots r_k$ whose mathematical expectation tends to x_i by choosing k large enough. This approximation leads to the systematic error.

3.2 Systematic and Statistical Errors

An stopping criterion for the random walk to create the Markov chain γ_k is given by *systematic error* $|W_k| < \epsilon$ where $\epsilon > 0$ is the given parameter. An upper bound for length of Markov chain γ_k can be computed sustituting the value of $|W_k| < \epsilon$, thus:

$$\begin{aligned} |W_k| = |\frac{t_{r_0r_1}t_{r_1r_2}\dots t_{r_{k-1}r_k}}{p_{r_0r_1}p_{r_1r_2}\cdots p_{r_{k-1}r_k}}| &\leq \frac{t_{r_0r_1}t_{r_1r_2}\dots t_{r_{k-1}r_k}}{\frac{t_{r_0r_1}}{\|T\|}\frac{t_{r_1r_2}}{\|T\|}\cdots\frac{t_{r_{k-1}r_k}}{\|T\|}} \\ &< \|T\|^k \end{aligned} \tag{7}$$

Accordingly $k < \frac{log(\epsilon)}{log\|T\|}$.
The fact of $\|T\| \leq 1$ reduces the length of Markov chain and convergence speed of the MCMC method increases. Simulating N independent sample paths of the Markovian Chain

$$r_0{}^{(s)} \to r_1{}^{(s)} \to \cdots \to r_k{}^{(s)}, \quad s = 1, 2, \cdots, N$$

and consider the sample mean of $X_i(\gamma_k^{(s)})$ to estimate the real mean $E[X_i(\gamma_k)]$ *i.e.* $\overline{X}_i(\gamma_k) = \frac{1}{N}\sum_{s=1}^{N} X_i(\gamma_k^{(s)}) \approx x_i$.

The *probable error* of the method is defined as $r_N = 0.6745\sqrt{\frac{var(X)}{N}}$ where:

$$P\{\left|\overline{X} - E\left[X\right]\right| < r_N\} \approx 1/2 \approx P\{\left|\overline{X} - E\left[X\right]\right| > r_N\}$$

if it has N independent realizations of random variable $X_i(\gamma_k)$, with mathematical expectation $E[X] = E\left[X_i\left(\gamma_k\right)\right]$ and average $\overline{X} = \overline{X}_i(\gamma_k)$ [9]. Probable error is employed to estimate the statistical error. The *statistical error* is expressed as: $\left|\overline{X}_i\left(\gamma_k\right) - E\left[X_i\left(\gamma_k\right)\right]\right| < \delta$, where δ is the given parameter. Using the precision the precision $r_N \leq \delta$ and employing the upper bound for $var(X_i(\gamma))$, it results:

$$N \geq \frac{(0.6745)^2}{\delta^2} \frac{\|f\|^2}{(1 - \|f\|)^2} \tag{8}$$

Algorithm 1 presents the MCMC method [15].

Algorithm 1: MCMC Algoritm for solving SLAE

Data: T, f, ϵ, δ
Result: $X_i(\gamma_k)$

Compute $N = \left[\frac{(0.6745)^2}{\delta^2} \frac{\|f\|^2}{(1-\|T\|)^2}\right] + 1$
Compute P based on the type of probability transition matrix
for $i = 1$ *to* n **do**
  for $s = 1$ *to* N **do**
    Set $W_0 = 1, k = 0, point = i, X_i^{(s)} = W_0 f_i$
    while $|W_k| \geq \epsilon$ **do**
      Generate an r.v. $nextpoint$, distributed on $i - th$ row of matrix P as:
      Set $nextpoint = 1, u = rand$
      while $u > P_{ponint,nextpoint}$ **do**
        $nextpoint = nexpoint + 1$
      Set $k = k + 1$
      if $t_{point,nextpoint} \neq 0$ **then**
        Compute
        $W_k = W_{k-1} \frac{t_{point,nextpoint}}{P_{point,nextpoint}}$,
        $X_i^{(s)} = X_i^{(s)} + W_k f_{nextpoint}$
      Set $point = nextpoint$
  Compute $X_i(\gamma_k) = \frac{1}{N} \sum_{s=1}^{N} X_i^{(s)}$

3.3 Markov Chain-Monte Carlo method in Julia

Implementation of the MCMC method in Julia is transparent and very similar to the code presented in (1) in practice, the main packages requiered were: `Statistics`,`LinearAlgebra`,and `SparseArrays`.

```
# Compute T and f

M = diagm(0 => diag(A));
N = M-A;
T = inv(M) * N;
f = inv(M) * b;

# Compute P and p probability matrices

nT, mT = size(T);
S = fill(0, nT);
[S[i] += 1 for i in 1:nT, j in 1:mT if T[i,j] != 0];
P = fill(0., nT, mT);
[P[i,j]= 1/S[i] for i in 1:nT, j in 1:mT if T[i,j] != 0 ];

# Compute number of chains
N = floor((0.6745/δ)^2*((norm(f)^2)/(1-norm(T))^2)) +
1 # Se calcula N

# Compute MCMC method
Xs = fill(0., mT)
for i in 1:mT
  W_0 = 1
  for s in 1:N
     W = W_0; point = i; X = W_0 * f[i]
     while abs(W) >= ϵ
        nextpoint  = 1
        u = rand()
        while u >= sum(P[point, 1:nextpoint])
           nextpoint = nextpoint + 1
        end
        if T[point, nextpoint] != 0
           W_new = W *(T[point, nextpoint]/P[point, nextpoint])
           X = X + W_new * f[nextpoint]
        end
        point = nextpoint
        W = W_new
     end
     Xs[i] += X
  end
end
x_mc = Xs/N;
```

4. Experiments

In this paper we focus on the precision of the algorithm that works on sparse matrices. We are interested in testing how varying the precision, e.g. the systematic error δ and the probabilistic error ϵ, influences the quality of the Monte Carlo approximate solution. Tests were made in an x86 CPU architecture consist in a Intel Core i7 vPro CPU, 8Gb RAM.
In order to studying the Marko Chain-Monte Carlo method for solving systems of linear algebraic equations it is relevant not only to take abstract matrices but matrices which represent a concrete problem to solve. This section presents some tests about a matrix that represents the physical behavior of the temperature in a bidimiensional square domain which experiences diffusion. Mathematically, the system $Ax = b$ is defined by the following partial equation

$$\frac{\partial(\rho C_v T)}{\partial t} = \nabla \cdot (\kappa \nabla T)$$

where ρ, t, C_v, T and κ represent density, time, specific heat, temperature and thermal conductivity respectively. It's worth to mention that test matrix was obtained with finite volume method; dimesions of test matrix A are of 120×120 and vector b is a column vector of size 120. Figure1 shows the sparsity pattern of the test matrix.
Matrix A is used as input to the algorithm 1 to compute the accuracy of the approximate solutiton variying the ϵ and δ parameters. In order to show the output of `BenchmarkTools` macro some experiments was developed using values of $\epsilon = 0.1$ and $\delta = 0.1$

```
BenchmarkTools.Trial:
  memory estimate:  375.43 MiB
  allocs estimate:  977796
  --------------
  minimum time:     178.927 ms (21.18% GC)
  median time:      192.165 ms (20.70% GC)
```

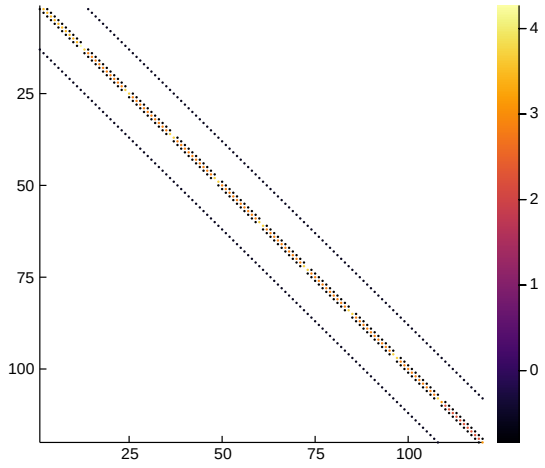

Fig. 1. Sparsity pattern and entrie's values of test matrix

```
mean time:          198.074 ms (22.14% GC)
maximum time:       280.477 ms (47.87% GC)
--------------
samples:            26
evals/sample:       1
```

Table 1 shows the compute times and the 2-norms of $\|b - A\hat{x}\|_2$ for different values of parameters.

Table 1. Number of Markov Chains N, $\|b - A\hat{x}\|_2$ and execution time for different values of ϵ and δ

ϵ	δ	N	$\|\cdot\|_2$	time(s)
0.1	0.1	4	6.729	0.188
0.1	0.05	14	5.174	0.553
0.1	0.01	330	1.05	12.07
0.1	0.005	1319	0.56	47.62
0.1	0.001	32961	0.22	1203.2
0.05	0.1	4	6.69	0.263
0.05	0.05	14	5.67	0.662
0.05	0.01	330	0.94	14.168
0.05	0.005	1319	0.50	56.96
0.05	0.001	32961	0.154	1435.9
0.01	0.1	4	6.65	0.292
0.01	0.05	14	3.60	0.839
0.01	0.01	330	1.06	19.12
0.01	0.005	1319	0.55	76.25
0.01	0.001	32961	0.154	2142.0
0.005	0.1	4	6.92	0.323
0.005	0.05	14	3.21	0.948
0.005	0.01	330	1.05	20.79
0.005	0.005	1319	0.69	84.62
0.005	0.001	32961	0.11	2149.7
0.001	0.1	4	6.36	0.394
0.001	0.05	14	3.05	1.14
0.001	0.01	330	1.00	25.60
0.001	0.005	1319	0.62	102.25
0.001	0.001	32961	0.18	2611.7

Previous results shows the influence of the parameter associated with the error statistic δ as the one that most strongly influences the accuracy and time of execution of the implementation. The proposed case of temperature diffusion over a domain it was observed that the accuracy of the method improves in each temporary step although the results suggest that this precision is limited.
Figure 2 shows the impact of parameters ϵ and δ on executime time. Results show that parameter δ increses the execution time when it takes values less than or equal to 0.005 because this parameter defines the number of markov chains that will be constructed to estimate each input of the solution vector. Figure 3 shows the impact of parameters ϵ and δ on accuray of the method. Results show there is not any impact of parameter ϵ on accuracy, however, the accuracy increases ($\|b - A\hat{x}\|_2$ decreses) when the δ parameter is decreasing and the number of markov chains increases.
Finally, figure shows the relationship between the precision of MCMC method and execution time. It is unquestionable that accuracy depends on the number of Markov chains N built for estimating each entry of the solution vector, which sacrifices the efficiency in orden to achieve some precision.

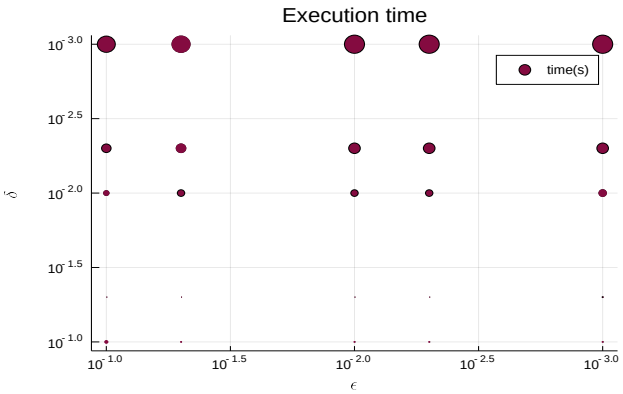

Fig. 2. MCMC execution times for different values of ϵ and δ parameters

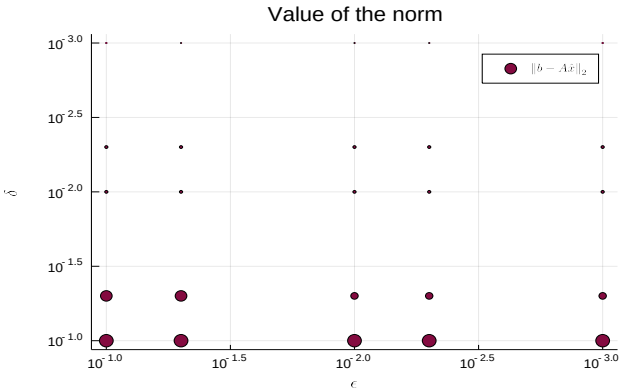

Fig. 3. MCMC accuracy for different values of ϵ and δ parameters

5. Evaluation

Further experiments were carried out to investigate how varying the precision parameters impacts on the approximate Monte Carlo delivering an efficient solution to SLAEs.

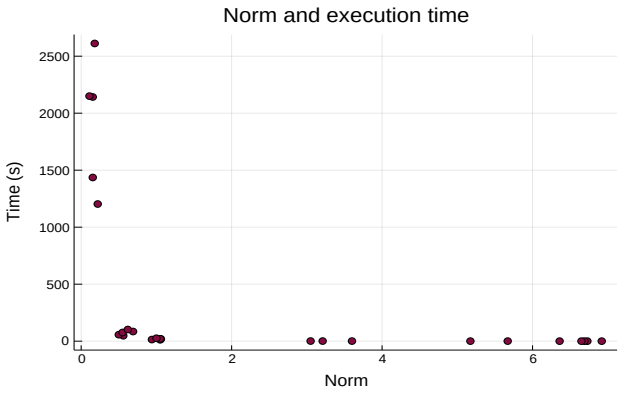

Fig. 4. Relationship

Consider the computational experiments on selected matrices given in table 1. First, it is evident that increasing the precision ϵ does not lead to any significant increase of the execution time of Monte Carlo algorithm. Therefore in many cases it is beneficial to obtain the Monte Carlo approximate solution using very rough precision, and also keeping the systematic and probable errors to be of the same magnitude, so that to minimize the execution time of the algorithm.
Second, it should be noted that increasing the precision δ does lead to significant increase of the execution time and precision of MCMC algorithm. This shows a susbstantial gain, it is possible to obtain the MCMC approximate solution of SLAE much more accurate by sacrificing the number of iterations to achieve this precision.

6. Conclusions and Future Work

A Markov Chain-Monte Carlo method based approximate solution and the corresponding algorithm for solving SLAEs have been presented as an alternative strategy. MCMC algorithm has been implemented in Julia v.1.0 which enables increasing the time in analysis. Several adavantages by implementing the MCMC method in Julia v.1.0. are worth to note. Transparent syntax, very fast even in a single multicore processor. Packages as `LinearAlgebra`,`Statistics`,and `SparseArray`were very useful in order to compute norms, random variables, and optimize calculation over `sparse arrays`.
The study of the parameters that define the accuracy of the method allows us to know its limitations and make decisions about the level of precision desired, which can be low and be fine-tuned by means of other iterative techniques.
As future work we can mention tha Ssome works have proposed MCMC algorithm is well suited for calculating preconditioners for SLAE solvers [18, 2] using classes of matrices such as non-diagonally dominant and non-symmetric matrices; particularly taken matrices from different sets obtained from *Suite Sparse Matrix Collection* [1] (this repository actually merge two previous collections -The Matrix Market [7] and The University of Florida Sparse Matrix Collection [10]). `MatrixDepot` and `SparseArrays` Julia packages are efficient tools in order to get and easily manipulate benchmark sparse matrices, future work deals with manipulate test matrices using this packages.

[1] https://sparse.tamu.edu/

Finally, MCMC method is highly parallelizable. The support for NVIDIA GPUs to the Julia programming language [6] promises to be an excellent help to program GPUs devices in order to increase the accuracy of the method and the speed of execution.

Acknowledgements

We would like to thank to José Antonio Borrás Gutiérrez (post graduate student in UNAM) by sharing the test matrices, we hope Antonio evolves for Python to Julia very soon.

7. References

[1] V. Alexandrov, E. Atanassov, I. Dimov, S. Branford, A. Thandavan, and C. Weihrauch. Parallel Hybrid Monte Carlo Algorithms for Matrix Computations. In Vaidy Sunderam, editor, *Computational Science – ICCS 2005*, volume 3516 of *Lecture Notes in Computer Science*, pages 752–759. Springer Berlin / Heidelberg, 2005.

[2] V. Alexandrov and O. Esquivel-Flores. Towards Monte Carlo Preconditioning Approach and Hybrid Monte Carlo Algorithms for Matrix Computations. *Computers and Mathematics with Applications*, (in publication), 2015.

[3] Vassil Alexandrov, Diego Davila, Oscar Esquivel-Flores, Aneta Karaivanova, Todor Gurov, and Emanouil Atanassov. On monte carlo and quasi-monte carlo for matrix computations. In Ivan Lirkov and Svetozar Margenov, editors, *Large-Scale Scientific Computing*, pages 249–257, Cham, 2018. Springer International Publishing.

[4] Vassil Alexandrov and Aneta Karaivanova. Parallel Monte Carlo algorithms for sparse SLAE using MPI. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pages 283–290. Springer, 1999.

[5] E. Atanassov, T. Gurov, S. Ivanovska, A. Karaivanova, and T. Simchev. On the parallel implementation of quasi-monte carlo algorithms. In Ivan Lirkov and Svetozar Margenov, editors, *Large-Scale Scientific Computing*, pages 258–265, Cham, 2018. Springer International Publishing.

[6] Tim Besard, Christophe Foket, and Bjorn De Sutter. Effective extensible programming: Unleashing julia on gpus. *CoRR*, abs/1712.03112, 2017.

[7] Ronald F Boisvert, Roldan Pozo, Karin A Remington, Richard F Barrett, and Jack Dongarra. Matrix market: a web resource for test matrix collections. In *Quality of Numerical Software*, pages 125–137, 1996.

[8] S. Branford, C. Sahin, A. Thandavan, C. Weihrauch, V.N. Alexandrov, and I.T. Dimov. Monte Carlo Methods for Matrix Computations on the Grid. *Future Generation Computer Systems*, 24(6):605 – 612, 2008.

[9] Simon Branford, Christian Weihrauch, and Vassil Alexandrov. A Sparse Parallel Hybrid Monte Carlo Algorithm for Matrix Computations. In Vaidy Sunderam, Geert Albada, Peter Sloot, and Jack Dongarra, editors, *Computational Science – ICCS 2005*, volume 3516 of *Lecture Notes in Computer Science*, pages 743–751. Springer Berlin / Heidelberg, 2005.

[10] Timothy A Davis and Yifan Hu. The University of Florida Sparse Matrix Collection. *ACM Transactions on Mathematical Software (TOMS)*, 38(1):1, 2011.

[11] I.T. Dimov and V. Alexandrov. A New Highly Convergent Monte Carlo Method for Matrix Computations. *Mathematics and Computers in Simulation*, 47(2-5):165–181, Aug 1998.

[12] I.T. Dimov, V.N. Alexandrov, R. Papancheva, and C. Weihrauch. Monte Carlo Numerical Treatment of Large Linear Algebra Problems. In *Lecture Notes in Computing Sciences: Computational Science - ICCS 2007*, volume 4487, pages 747–754, Berlin, 2007. Springer-Verlag GmbH.

[13] Ivan Dimov, Sylvain Maire, and Jean Michel Sellier. A new walk on equations monte carlo method for solving systems of linear algebraic equations. *Applied Mathematical Modelling*, 39(15):4494 – 4510, 2015.

[14] B. Fathi Vajargah, B. Liu, and V.N. Alexandrov. Mixed Monte Carlo Parallel Algorithms for Matrix Computation. In *Lecture Notes in Computational Science 2330 - ICCS 2002*, pages 609–618, Berlin Heidelberg, 2002. Springer Verlag.

[15] Behrouz Fathi-Vajargah and Zeinab Hassanzadeh. Improvements on the hybrid monte carlo algorithms for matrix computations. *Sādhanā*, 44(1):1, Dec 2018.

[16] G.H. Golub and C. Van Loan. *Matrix computations*. Johns Hopkins studies in the mathematical sciences. Johns Hopkins University Press, 1996.

[17] A. Lebedev and V. Alexandrov. On advanced monte carlo methods for linear algebra on advanced accelerator architectures. In *2018 IEEE/ACM 9th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (scalA)*, pages 81–90, Nov 2018.

[18] J. Strassburg and V. Alexandrov. Enhancing Monte Carlo Preconditioning Methods for Matrix Computations. *Procedia Computer Science*, 29:1580–1589, 2014.