2011 50th IEEE Conference on Decision and Control and
European Control Conference (CDC-ECC)
Orlando, FL, USA, December 12-15, 2011

Solving a positive definite system of linear equations via the matrix exponential

Ammar Hasan, Eric C. Kerrigan, George A. Constantinides

***Abstract*— We present a new direct algorithm for solving a system of linear equations with a positive definite matrix by discretizing a continuous-time dynamical system for a large sampling time. The obtained algorithm is highly fine-grain parallelizable and its computational complexity grows logarithmically with respect to the condition number of the system of linear equations. When the parallelism is fully exploited, the algorithm is shown to be more efficient in terms of computational speed in comparison to other popular methods for solving a positive definite system of linear equations, especially for large and ill-conditioned problems.**

I. Introduction

Iterative numerical algorithms can be viewed as discrete-time dynamical systems [1], [2]. Based on this observation many authors have proposed to obtain new iterative algorithms by discretizing a suitably-defined continuous-time dynamical system [3]–[5]. In contrast to the existing literature, we present a scheme for obtaining a direct (non-iterative) algorithm by discretizing a continuous-time dynamical system for a large sampling time. We apply the proposed scheme to obtain a new algorithm for solving a system of linear equations with a positive definite matrix.

Systems of linear equations arise in many scientific problems and are central to many numerical algorithms. Algorithms for the solution of systems of linear equations are an important topic in numerical analysis and many algorithms have been developed and continue to be extensively studied. One of the important properties of an algorithm is the execution time, i.e. the total time taken by the algorithm to compute a solution. The execution time of an algorithm depends on many factors, which include the architecture of the computational hardware. Advances in parallel computer architecture allow highly parallel and customized circuits that can lend to high-speed solutions [6]–[9]. Graphics processing units (GPUs) and field programmable gate array (FPGA)s are two examples of the current popular parallel architectures. The potential parallelism in an algorithm and its exploitation in circuit design is an important factor in increasing the computational speed of an algorithm. Therefore, for a parallel computational hardware an algorithm that is easily parallelized is more suitable.

A. Hasan and G. Constantinides are with the Department of Electrical and Electronic Engineering, Imperial College London, SW7 2AZ, U.K. {ammar.hasan07,g.constantinides}@imperial.ac.uk

E. Kerrigan is with the Department of Aeronautics and the Department of Electrical and Electronic Engineering, Imperial College London, SW7 2AZ, U.K. e.kerrigan@imperial.ac.uk

This work was funded by the EPSRC under grant number EP/G031576/1, EP/I020357/1, EP/I010236/1 and the EC FP7 grant EMBOCON.

We show that the algorithm proposed in this paper for solving a system of linear equations is highly parallelizable. We compare the proposed algorithm with the conjugate gradient method and the Cholesky decomposition method in terms of some of the factors that are important for a parallel computing architecture. We argue that a fully parallel implementation of the proposed algorithm will provide a faster computational time in comparison to the other algorithms.

II. Deriving an algorithm from a continuous-time dynamical system

In this section we outline a general scheme for deriving an algorithm using a continuous-time dynamical system. Consider a numerical problem with solution $x^* \in \mathbb{R}^n$. Assume that we can synthesize a continuous-time dynamical system

$$\frac{d}{dt}x(t) = f(x),$$

where $x \in \mathbb{R}^n$ is the state and $f : \mathbb{R}^n \to \mathbb{R}^n$, such that it has a globally asymptotically stable equilibrium x_e that is equal to the solution x^* of the numerical problem. A discretization of such a dynamical system would give a discrete-time dynamical system that converges to the solution of the numerical problem for any initial condition. This discrete-time dynamical system can serve as an iterative algorithm for the numerical problem.

While discretization for all sampling times can be used as an algorithm, the discretized system obtained for a larger time step will converge in fewer iterations to the equilibrium. We illustrate this with the help of Figure 1. The figure shows the solution of a linear single-input single-state continuous-time system and the states of the discretized systems for sampling times $h = 1$ and $h = 2$. As can be seen the discretized system for $h = 2$ requires less than half the number of iterations to get to the same value as the discretized system with $h = 1$. Therefore, for a higher rate of convergence a larger value of h should be used. A higher rate of convergence is desirable as it could lead to a computationally fast algorithm.

Following the same argument, in the limit as $h \to \infty$ one should obtain a discretized system that converges to the solution from any initial condition in just one iteration, i.e. a direct algorithm. Therefore, by discretizing for a large enough sampling time we should get an algorithm that gives an approximate solution after a single iteration.

Many authors have discussed the potential of using continuous-time dynamical systems to develop iterative algorithms and applied this approach to obtain new algorithms;

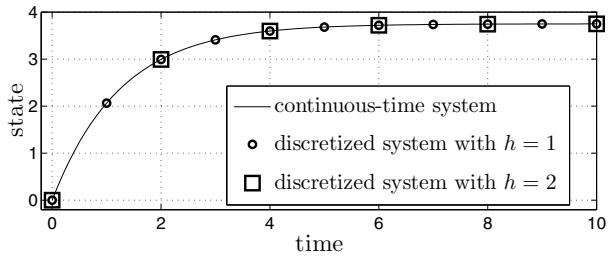

Fig. 1. Sampling time and the speed of convergence

see [3]–[5] and the references therein. Although not in the same context, the effect of sampling time on the rate of convergence has also been identified earlier [3]. However, to the best of our knowledge, this paper is the first to advocate the derivation of a direct algorithm by discretizing for a large enough h or to use a value of h as large as feasible to obtain a fast converging iterative algorithm. A reason for this could be that the overhead of discretization for a large (or arbitrary) h was considered computationally prohibitive. In the next section we consider an algorithm for solving a system of linear equations that can be discretized efficiently using a parallel computing architecture.

III. Algorithm for solving a system of linear equations

In this section we will derive an algorithm for solving a system of linear equations

$$Fx^* = g, \tag{1}$$

where $F \in \mathbb{R}^{n\times n}$ is symmetric and positive-definite, $x^* \in \mathbb{R}^n$ is the solution, and $g \in \mathbb{R}^n$. Since the matrix F is positive definite, it is also invertible and a unique solution will exist for (1).

Consider the linear continuous-time system

$$\frac{d}{dt}x(t) = Ax(t) + b, \tag{2}$$

where $x \in \mathbb{R}^n$ is the state, $A := -F$, and $b := g$. The eigenvalues of A will be the negative of the eigenvalues of F. Since F is a positive definite matrix all its eigenvalues are positive. Hence, all eigenvalues of A will lie in the left half plane and system (2) will be asymptotically stable. An equilibrium x_e of (2) will satisfy

$$0 = Ax_e + b \Leftrightarrow 0 = -Fx_e + g \Leftrightarrow Fx_e = g.$$

Since the above equation is the same as (1), system (2) will have a unique equilibrium that is equal to the solution of (1). The continuous-time system (2) has also been proposed by other authors [4] for the solution of (1).

The exact solution of the continuous-time system (2) at time $t = kh$ for $k = 0, 1, 2, \cdots$, where h is the sampling interval, is given by [10, Sec. 3.4.2]

$$x((k+1)h) = \exp(Ah)x(kh) + \int_0^h \exp(A(h-\tau))bd\tau \tag{3}$$

for all k, where $\exp(\cdot)$ represents the exponential function. The solution (3) suggests the following discrete-time model for the continuous-time system:

$$x_{k+1} = \Phi_h x_k + \gamma_h, \tag{4}$$

where

$$\Phi_h := \exp(Ah), \tag{5}$$

$$\gamma_h := \int_0^h \exp(A(h-\tau))bd\tau, \tag{6}$$

and $x_k := x(kh)$ is the state at time index k. The obtained discrete-time system (4) for all sampling times will have the same equilibrium as the continuous-time system. Since the equilibrium is equal to x^*, the discrete-time system can be used as an algorithm for solving (1).

Based on the discussion in the previous section, we will obtain a faster converging iterative algorithm for a larger sampling time in comparison to an algorithm obtained for a smaller sampling time. Moreover, the algorithm should converge to the solution in a single iteration for $h \to \infty$. In fact, for (4) we have

$$\lim_{h\to\infty} \Phi_h = 0_{n\times n} \tag{7}$$

and

$$\lim_{h\to\infty} \gamma_h = x^*. \tag{8}$$

Therefore, to solve problem (1) we just have to compute $\lim_{h\to\infty}\gamma_h$. When A is invertible, which is the case here, the analytic expression for the integral in (6) is given by

$$\gamma_h = A^{-1}(\exp(Ah) - I_n)b. \tag{9}$$

Since this analytic expression involves a matrix inverse in itself, it is not feasible to attempt to evaluate γ_h in this manner. In the next section we use another method for calculating γ_h. However, the method can only be used for a finite h. Therefore, we propose to use a large, finite value of h such that $\gamma_h \approx x^*$.

A. Computations in discretization

In this section we will present an efficient way of computing γ_h. Let

$$X := \begin{bmatrix} A & b \\ 0_{1\times n} & 0 \end{bmatrix}$$

be an $(n+1)\times(n+1)$ matrix. Using the results in [11] it can be shown that the exponential of matrix Xh is equal to

$$\begin{bmatrix} \exp(Ah) & \int_0^h \exp(A(h-\tau))bd\tau \\ 0_{1\times n} & 1 \end{bmatrix} = \begin{bmatrix} \Phi_h & \gamma_h \\ 0_{1\times n} & 1 \end{bmatrix}.$$

Therefore, if we calculate the exponential of the matrix Xh, we will compute γ_h as well as Φ_h and hence the complete discretization.

In the control systems literature we usually discretize systems for small values of h, in which case several approximations of matrix exponential are possible, e.g. Tustin's approximation, etc. [10]. In contrast, here we are interested in large values of h, for which these approximations are not valid.

In numerical analysis, the computation of the matrix exponential is extensively studied and several algorithms have been developed. A comprehensive survey of these algorithms can be found in [12]. One of the most effective techniques of computing the matrix exponential is the scaling and squaring method [12]. It is based on the fact that for a matrix Z and a scalar ζ

$$\exp(Z) = \left(\exp\left(\frac{Z}{\zeta}\right)\right)^{\zeta}. \tag{10}$$

The scaling factor ζ is chosen so that $\rho(Z/\zeta) \leq 1$, in which case the Padé approximant and the Taylor series method are very efficient methods for approximating $\exp(Z/\zeta)$ [12]. Here $\rho(\cdot)$ denotes the spectral radius of a matrix, which is defined as the largest absolute eigenvalue of a matrix. Although the Padé approximant is considered to be more efficient than the Taylor series method [12], we will choose the first order Taylor series approximate for its low computational complexity. The first order Taylor series approximate is given by

$$\exp\left(\frac{Z}{\zeta}\right) \approx I + \frac{Z}{\zeta}. \tag{11}$$

To simplify computations, the scaling factor ζ is often chosen equal to 2^s for some positive integer s. This allows the matrix power in (10) to be evaluated by squaring s times.

Following the above discussion we approximate our discrete-time system calculation by

$$\exp(Xh) \approx \left(I_{n+1} + \frac{Xh}{2^s}\right)^{2^s}.$$

B. Values for h and s

In this section we state results that help in choosing the values of h and s, but first we introduce some notation. We use $\|\cdot\|_2$ to denote the vector Euclidean norm and the matrix spectral norm (or the induced matrix 2-norm). Let us denote the smallest eigenvalue of the symmetric and positive definite matrix F by $\lambda_{\min}(F)$, the largest eigenvalue of F by $\lambda_{\max}(F)$, the condition number of F by $\kappa(F) := \|F\|_2\|F^{-1}\|_2$, an upper bound for $\kappa(F)$ by $\overline{\kappa}(F)$, a lower bound on $\lambda_{\max}(F)$ by $\underline{\lambda}_{\max}(F)$, and an upper bound on $\lambda_{\max}(F)$ by $\overline{\lambda}_{\max}(F)$. Since F is a symmetric and positive definite matrix, $\kappa(F) = \lambda_{\max}(F)/\lambda_{\min}(F)$, $\lambda_{\max}(F) = \rho(F)$, and all its eigenvalues are real and positive.

Theorem 1: If x^* and g are not zero and

$$h \geq \alpha/\lambda_{\min}(F) = \alpha\kappa(F)\lambda_{\max}(F), \tag{12}$$

where α is any positive scalar, then the approximate solution γ_h satisfies

$$\frac{\|x^* - \gamma_h\|_2}{\|x^*\|_2} \leq \exp(-\alpha), \tag{13}$$

$$\frac{\|g - F\gamma_h\|_2}{\|g\|_2} \leq \exp(-\alpha). \tag{14}$$

Proof: See Appendix I. ∎

Corollary 1: If x^* is not zero and

$$h \geq \alpha\overline{\kappa}(F)/\underline{\lambda}_{\max}(F), \tag{15}$$

for some $\alpha > 0$, then (13) is satisfied.

The expression on the left hand side of (13) and (14) is called the relative error and the relative residual, respectively. Given a desired upper bound on the relative error or the relative residual, Theorem 1 can be used to choose a value of h.

As discussed earlier, for a good approximation of the matrix exponential, the value of s should be chosen such that $\rho(Xh/2^s) \leq 1$ [12]. The theorem given below helps the choice of s.

Theorem 2: If s is an integer such that

$$s \geq \log_2(h\lambda_{\max}(F)) = \log_2(\alpha\kappa(F)), \tag{16}$$

Algorithm 1 New Algorithm for solving (1)

Input: Matrix F, vector g and scalar $\alpha > 0$
Output: x (an approximation to x^*)

Algorithm:

1: $h \leftarrow \alpha\kappa(F)/\lambda_{\max}(F)$
2: $s \leftarrow \lceil \log_2(h\lambda_{\max}(F)) \rceil$
3: $Y \leftarrow \begin{bmatrix} I_n - F(h/2^s) & g(h/2^s) \\ 0_{1\times n} & 1 \end{bmatrix}$
4: **for** $i = 1$ to s **do**
5: $\quad Y \leftarrow Y^2$
6: **end for**
7: $x \leftarrow Y_{12}$

then

$$\rho(Xh/2^s) \leq 1. \tag{17}$$

Proof: See Appendix II. ∎

Corollary 2: If

$$s \geq \log_2(h\overline{\lambda}_{\max}(F)), \tag{18}$$

then (17) is satisfied.

The bounds for h and s involve the values of $\kappa(F)$ and $\lambda_{\max}(F)$. In practice these values may not be readily available and for computational efficiency it might be desirable to avoid their computation. In this case we can use the corollaries to choose the value of h and s. Possible expressions for bounds on $\lambda_{\max}$ are

$$\overline{\lambda}_{\max}(F)) := \sqrt{n}\|F\|_\infty,$$

and

$$\underline{\lambda}_{\max}(F)) := \frac{1}{\sqrt{n}}\|F\|_\infty,$$

where $\|\cdot\|_\infty$ denotes the induced ∞-norm of a matrix.

C. The complete algorithm

We now have all the ingredients to state the complete algorithm. It is given in Algorithm 1. In the algorithm, Y_{12} denotes the $n \times 1$ submatrix of Y when partitioned as

$$Y =: \begin{bmatrix} Y_{11} & Y_{12} \\ Y_{21} & Y_{22} \end{bmatrix},$$

where $Y_{11} \in \mathbb{R}^{n\times n}$, $Y_{21} \in \mathbb{R}^{1\times n}$, and $Y_{22} \in \mathbb{R}^{1\times 1}$. Note that Y is initialized with the value

$$I_{n+1} + \frac{Xh}{2^s},$$

which is the first order Taylor series approximation of $\exp(Xh)$.

In Step 2 of Algorithm 1, $\lceil\cdot\rceil$ denotes the ceiling function. It is required because s should be an integer.

IV. Comparison of time complexity

In this section we find the time complexity of the newly proposed algorithm and compare it with the time complexities of the preconditioned conjugate gradient (PCG) method and the Cholesky decomposition method. The PCG method and the Cholesky decomposition method are arguably the best known iterative and direct algorithms, respectively, for solving a system of linear equations with a positive definite matrix. Due to limitations on space, we will assume that the reader has access to the description of these methods, which can be found in most standard books on linear algebra,

e.g. [13], [14]. We will use the LDL factorization variant of the Cholesky decomposition method and the Chebyshev preconditioner for the PCG method because they are considered to be more suitable for parallel computing [9], [15].

The time complexity of an algorithm quantifies the time taken by an algorithm on a computational hardware. A common measure of time complexity of algorithms for sequential computing is the total number of floating point operations (flops) required by the algorithm. A measure of time complexity for parallel computing is the least time taken by an algorithm assuming that 1) all basic floating point operations take unit time, 2) a sufficiently large number of processing elements are available, and 3) the communication between processing elements takes zero time [16, Sec. 1.2]. Here sufficiently large means that enough processing elements are available such that the parallelization of operations in the algorithm is not limited by the number of processing elements. We will follow the method in [16, Sec. 1.2] to find the time complexity of the algorithms. For the purposes of clarity, we will only focus on the steps of the algorithms that are computationally intensive.

1) Cholesky decomposition method: The most computationally intensive tasks in the Cholesky decomposition method are the forward/back substitutions of unit triangular matrices. Forward and back substitution are very similar algorithms with the same time complexity. Therefore, we will only analyze the forward substitution. Figure 2 shows a graph that represents the forward substitution algorithm for solving a system of linear equations with a unit lower triangular matrix

$$Lz = y,$$

where

$$L := \begin{bmatrix} 1 & 0 & \cdots & 0 \\ l_{2,1} & 1 & \cdots & 0 \\ \vdots & & \ddots & \\ l_{n,1} & l_{n,2} & \cdots & 1 \end{bmatrix}, z := \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{bmatrix}, y := \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}.$$

If we assume that both addition and multiplication take a unit of time, then using the figure we can find that the time taken by the forward substitution algorithm is

$$T_{fs}(n) := 2(n-1).$$

For the back substitution we have

$$T_{bs}(n) := T_{fs}(n).$$

The Cholesky decomposition method involves forward substitution for problem sizes of 2 to n and a single backward substitution of problem size n. Since neither of these forward/back substitutions can be executed in parallel, we add the time complexities of all forward/back substitutions to get the time complexity of the Cholesky decomposition method:

$$T_{ch}(n) := \sum_{i=2}^{n} \left(T_{fs}(i)\right) + T_{bs}(n) = n^2 + n - 2. \qquad (19)$$

2) Preconditioned conjugate gradient: The computationally intensive tasks in the PCG method are the matrix-vector products and vector dot products. Matrix-vector products involve n dot products, all of which can be computed in parallel. Therefore, the time complexity of matrix-vector

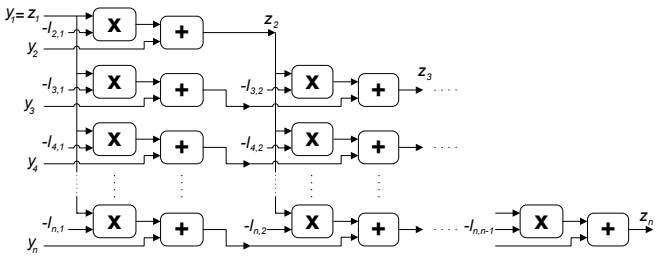

Fig. 2. Graph for the forward substitution algorithm

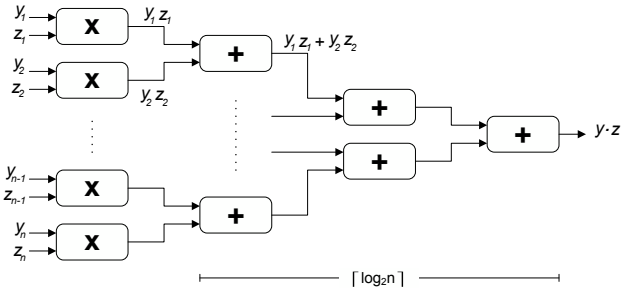

Fig. 3. Graph for the dot product algorithm

product is the same as that of a vector dot product. Figure 3 shows a graph that represents the dot product algorithm. The dot product of two vectors $y \in \mathbb{R}^n$ and $z \in \mathbb{R}^n$ is defined as

$$y \cdot z := \sum_{i=1}^{n} y_i z_i,$$

where y_i and z_i denote the ith component of the corresponding vector. Using the figure we can find that the time complexity for the dot product algorithm is

$$T_{dp}(n) := 1 + \lceil \log_2 n \rceil .$$

Using the calculated time complexity of the dot product algorithm, the time complexity of the PCG method turns out to be

$$T_{pcg}(n, \sigma) := 4\sigma T_{dp}(n) = 4\sigma(1 + \lceil \log_2 n \rceil). \qquad (20)$$

where σ denotes the number of iterations taken by the algorithm.

3) Proposed algorithm: In the proposed algorithm the most computationally intensive task is finding the square of matrix Y. A matrix-matrix multiplication involves n^2 dot products. For the specific structure of matrix Y it can be shown that only $(n^2 + 3n)/2$ dot products will be required. All of these dot products can be computed in parallel. Therefore, the time complexity of computing the square of the matrix Y is equal to the time complexity of a vector dot product $T_{dp}(n)$. Since we have to compute the matrix square s times, the time complexity of the proposed algorithm is

$$T_{pa}(s, n) := sT_{dp}(n) := s\left(1 + \lceil \log_2(n) \rceil\right). \qquad (21)$$

4) Comparison: The time complexity of the all the algorithms (19)-(21) are a function of the problem size n. For the proposed algorithm it also depends on the scaling factor s. As seen in (16) a suitable value of s is a function of the condition of the problem and the parameter α. For the purposes of comparison we will take $\alpha = \left\lceil -\ln 2^{-52} \right\rceil$, where $\ln$ denotes the natural logarithmic. The constant 2^{-52} is the machine epsilon for IEEE double precision and our choice of α will guarantee the relative error due to finite h

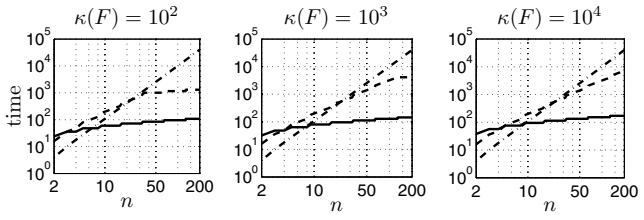

Fig. 4. Time complexity of algorithms as a function of problem size n. (dash-dot) Cholesky decomposition method, (dashed) preconditioned conjugate gradient, (solid) proposed algorithm

to be less than or equal to the machine epsilon.

Besides the problem size n, the time complexity of the PCG method also depends on the number of iterations σ. The number of iterations in the PCG method can be bounded by a function of the condition number [15], [17]. The bound depends on the condition number of the original (unconditioned) matrix, the desired relative residual, and the order of the Chebyshev preconditioner k. The order of the Chebyshev preconditioner is often taken between 5 and 10 [15]. In this paper, throughout our analysis we will take $k = 10$ or $k = n$ in the case that $n < 10$. We will find the bound for a relative residual of less than 10^{-6}, which is the default value used in MATLAB [18].

Using the values and bounds stated above, we have plotted the time complexities of all the algorithms for different values of the condition number and problem sizes in Figure 4. It can be seen in the figure that for problem sizes less than about 10, the Cholesky decomposition algorithm has the lowest time complexity. Whereas, for larger problem sizes the proposed algorithm has the lowest time complexity.

V. Numerical examples

For numerical comparison we consider matrices in the University of Florida sparse matrix collection [19] as the matrix F and a unit vector with the n^{th} element equal to 1 as the right hand side vector g. We select all matrices that are real, symmetric, positive definite and have a size less than or equal to 200. We limited our selection to matrices of sizes 200 or less because of space limitations and the selection is enough to show the important trends.

For the purpose of numerical comparison we will apply the proposed algorithm and the PCG method on the Jacobi preconditioned system [14, Sec 10.2]. The Jacobi preconditioned system is easy to compute and could potentially have a lower condition number than the original system.

Table I show a comparison of the time complexities of the algorithm. The table is sorted in ascending order of condition number. The lowest time complexity for a problem is highlighted in bold. It can be seen in the table that for all of the considered problems the Cholesky decomposition method has the largest time complexity. For most of the considered problems, the proposed algorithm gives the lowest time complexity, especially when the problem is ill-conditioned. In some cases we see that the CG method is computationally faster. In these cases either the condition number is very low or the eigenvalues are clustered, which we found upon inspection. The CG method is known to perform well when

TABLE I

Comparison of time complexity of algorithms for a subset of University of Florida sparse matrix collection

Name	$\kappa(F)$	n	s	σ	T_{ch}	T_{pa}	T_{pcg}	Relative Residual Cholesky	Relative Residual proposed
mesh1e1	8.2E+00	48	8	3	2350	**56**	84	5.0E-16	1.9E-16
bcsstm02	8.8E+00	66	6	1	4420	48	**32**	0.0E+00	0.0E+00
mesh1em6	9.3E+00	48	9	4	2350	**63**	112	1.4E-16	4.8E-16
bcsstm05	1.3E+01	153	6	1	23560	54	**36**	0.0E+00	0.0E+00
mesh1em1	3.4E+01	48	10	6	2350	**70**	168	2.4E-16	7.7E-16
Trefethen_20b	4.4E+01	19	8	2	378	**48**	**48**	9.1E-18	1.4E-17
Trefethen_20	9.6E+01	20	9	2	418	54	**48**	1.1E-16	5.6E-17
Trefethen_200b	7.3E+02	199	8	1	39798	72	**36**	2.2E-16	2.2E-16
bcsstm22	9.4E+02	138	6	1	19180	54	**36**	0.0E+00	6.7E-16
Trefethen_150	1.1E+03	150	9	2	22648	81	**72**	1.1E-16	1.6E-18
Trefethen_200	1.6E+03	200	9	1	40198	81	**36**	9.0E-19	3.3E-16
nos4	2.7E+03	100	17	24	10098	**136**	768	5.8E-15	9.7E-15
bcsstk02	1.3E+04	66	18	6	4420	**144**	192	1.1E-14	1.8E-14
Journals	1.9E+04	124	12	5	15498	**96**	160	9.4E-16	2.8E-14
bcsstk05	3.5E+04	153	19	44	23560	**171**	1584	3.5E-15	3.2E-14
lund_b	6.0E+04	147	15	20	21754	**135**	720	2.8E-14	1.1E-12
bcsstk22	1.7E+05	138	19	30	19180	**171**	1080	5.2E-15	1.1E-13
bcsstk01	1.6E+06	48	17	17	2350	**119**	476	7.9E-16	9.1E-14
LF10	5.1E+06	18	18	10	340	**108**	240	8.3E-13	8.7E-11
lund_a	5.4E+06	147	21	24	21754	**189**	864	2.8E-12	8.2E-11
bcsstk04	5.6E+06	132	18	9	17554	**162**	324	3.6E-16	1.1E-13
bcsstk03	9.5E+06	112	21	33	12654	**168**	1056	2.0E-15	5.9E-13
ex5	1.3E+08	27	33	52	754	**198**	1248	1.7E-10	1.5E-09
LFAT5	1.7E+08	14	14	5	208	**70**	100	1.4E-13	3.9E-09

most of the eigenvalues are clustered together [17, Ch. 4]. However, as evident from the tables, this will not be the case in general.

Table I also show the accuracy of the obtained solutions by the Cholesky decomposition algorithm and the newly proposed algorithm. We have not included the PCG method in this comparison because the method is usually terminated earlier at a lower accuracy. The table lists the relative residual for the obtained solutions. The relative residual indicates how accurate the obtained solution is; a smaller value means a more accurate solution. As seen in the table, the relative residual for the proposed method is generally greater than the relative residual of the Cholesky decomposition method by one order of magnitude. This means that the proposed method is slightly less accurate than the Cholesky decomposition. However, for most applications the accuracy of the proposed method would be enough.

Note that for some values of s in Table I, 2^s will be a very large number. While this may not be an issue for common floating point number representations – the largest representable power of 2 in IEEE single precision and IEEE double precision is 2^{127} and 2^{1023}, respectively – for some number representations the number 2^s can exceed the maximum range of representable numbers. Note that in Algorithm 1, the number 2^s always appears as part of the expression $h/2^s$, therefore, the explicit calculation of 2^s can be avoided. Moreover, $h/2^s$ is not a large number for the values of h and s used in Algorithm 1.

VI. Conclusions

By discretizing a synthesized continuous-time dynamical system for a large sampling time, we have obtained a new direct algorithm for solving a positive definite system of linear equations. The newly proposed algorithm is highly parallelizable and suitable for computational hardware that can exploit this parallelism.

The proposed method is not suitable for large and sparse matrices, since the method involves a matrix exponential that can destroy the sparsity structure in a matrix. However, for dense problems, the newly proposed algorithm is arguably the most efficient algorithm in terms of time complexity.

VII. Future work

In this paper all analysis is done in IEEE double precision. The hardware resources utilized grow with the precision. Due to limited hardware resources it might be desirable to implement the algorithm in a lower precision. Therefore, it would be useful to establish that the algorithm is numerically stable and hence also suitable for lower precisions.

Appendix I
Proof of Theorem 1

We will only give the proof for conclusion (13). Proof for conclusion (14) is similar.

The difference between the exact solution and the approximate solution γ_h is $x^* - \gamma_h$. Using the analytic expression of γ_h, given in (9), we get

$$\begin{aligned} x^* - \gamma_h &= x^* - A^{-1}(\exp(Ah) - I)g \\ &= x^* - A^{-1}\exp(Ah)g + A^{-1}g \\ &= x^* - \exp(Ah)A^{-1}g + A^{-1}g \\ &= x^* + \exp(Ah)x^* - x^* \\ &= \exp(Ah)x^*. \end{aligned} \tag{22}$$

Here we have used the fact that $A^{-1}\exp(Ah) = \exp(Ah)A^{-1}$, $A = -F$ and $x^* = F^{-1}g$. The property of the exponential can be verified by writing the matrix exponential in Taylor series form. Taking the 2-norm of both sides of the equation,

$$\begin{aligned} \|x^* - \gamma_h\|_2 &= \|\exp(Ah)x^*\|_2 \\ &\leq \|\exp(Ah)\|_2\|x^*\|_2, \\ \Leftrightarrow \|x^* - \gamma_h\|_2/\|x^*\|_2 &\leq \|\exp(Ah)\|_2. \end{aligned}$$

To complete the proof, we have to show that $\|\exp(Ah)\|_2 \leq \exp(-\alpha)$ for the value of h in (12). The eigenvalues of the exponential of a matrix are equal to the exponential of the eigenvalues of the matrix. This can be seen by multiplying the Taylor series form of a matrix exponential with an eigenvector of the matrix. Therefore, we have

$$\begin{aligned} \operatorname{spec}(\exp(Ah)) &= \{\exp(\lambda) : \lambda \in \operatorname{spec}(Ah)\} \\ &= \{\exp(-\lambda h) : \lambda \in \operatorname{spec}(F)\}, \end{aligned}$$

where $\operatorname{spec}(\cdot)$ denotes the spectrum of a matrix and is defined as the set of all the eigenvalues of a matrix.

Since the spectral radius is the maximum absolute eigenvalue of a matrix, we have

$$\rho(\exp(Ah)) = \max\{|\exp(-\lambda h)| : \lambda \in \operatorname{spec}(F)\},$$

where $|\cdot|$ denotes the absolute value of a scalar. All the eigenvalues of F are positive, therefore,

$$\rho(\exp(Ah)) = |\exp(-\lambda_{\min}(F)h)| = \exp(-\lambda_{\min}(F)h).$$

Substituting the lower bound on h from (12) we get

$$\rho(\exp(Ah)) \leq \exp\left(-\lambda_{\min}(F)\left(\alpha/\lambda_{\min}(F)\right)\right) = \exp(-\alpha).$$

Since F is symmetric, $\exp(Ah) = \exp(-Fh)$ will also be symmetric. For a symmetric matrix, the induced 2-norm is equal to its spectral radius, therefore,

$$\|\exp(Ah)\|_2 \leq \exp(-\alpha).$$

Appendix II
Proof of Theorem 2

Since X is a block diagonal matrix with A and 0 as the diagonal entries, we have

$$\rho(X) = \max\{\rho(A), 0\} = \rho(A) = \rho(-F) = \rho(F).$$

Therefore,

$$\rho(Xh/2^s) = (h/2^s)\rho(X) = \rho(F) = \lambda_{\max}(F).$$

Substituting the lower bound on s in (16), we get

$$\rho(Xh/2^s) \leq \frac{h\lambda_{\max}}{2^{\log_2(h\lambda_{\max})}} = \frac{h\lambda_{\max}}{h\lambda_{\max}} = 1.$$

References

[1] James M. Ortega. Stability of difference equations and convergence of iterative processes. *SIAM Journal on Numerical Analysis*, 10(2):268–282, 1973.

[2] Amit Bhaya and Eugenius Kaszkurewicz. *Control Perspectives on Numerical Algorithms and Matrix Problems*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2006.

[3] J. M. Ortega and W. C. Rheinboldt. *Iterative solution of nonlinear equations in several variables*. Academic Press, 1970.

[4] J. P. Chehab and Jacques Laminie. Differential equations and solution of linear systems. *Numerical Algorithms*, 40(2):103–124, 2005.

[5] M. T. Chu. Linear algebra algorithms as dynamical systems. *Acta Numerica*, 17:1–87, 2008.

[6] George A. Constantinides. Tutorial paper: Parallel architectures for model predictive control. In *Proceedings of the European Control Conference*, pages 138–143, Budapest, Hungary, 2009.

[7] A. Roldao Lopes and G. A. Constantinides. A high throughput FPGA-based floating point conjugate gradient implementation. In *Proceedings of Applied Reconfigurable Computing*, pages 75–86, London, UK, 2008.

[8] D. Boland and G. A. Constantinides. An FPGA based implementation of the MINRES algorithm. In *Proceedings of the IEEE International Conference on Field-Programmable Logic and Applications*, pages 379–384, Heidelberg, Germany, 2008.

[9] D. Yang, G. Peterson, and H. Li. High performance reconfigurable computing for Cholesky decomposition. In *Symposium on Application Accelerators in High Performance Computing, University of Illinois at Urbana-Champaign*, Urbana, Illinois, USA, 2009.

[10] R. H. Middleton and G. C. Goodwin. *Digital Control and Estimation: A Unified Approach*. Prentice Hall, 1990.

[11] C. Van Loan. Computing integrals involving the matrix exponential. *IEEE Transactions on Automatic Control*, 23(3):395–404, 1978.

[12] C. Moler and Charles Van Loan. Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. *SIAM Review*, 45(1):3–46, 2003.

[13] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, USA, 1996.

[14] Yousef Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, 2000.

[15] Steven F. Ashby, Thomas A. Manteuffel, and James S. Otto. A comparison of adaptive chebyshev and least squares polynomial preconditioning for hermitian positive definite linear systems. *SIAM Journal on Scientific and Statistical Computing*, 13:1–29, January 1992.

[16] Dimitri P. Bertsekas and John N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*. Athena Scientific, 1997.

[17] Anne Greenbaum. *Iterative Methods for Solving Linear Systems*. Society for Industrial and Applied Mathematics, 1997.

[18] The Mathworks, inc. MATLAB (R2010a), 2010.

[19] T. A. Davis and Y. Hu. The University of Florida Sparse Matrix Collection. ACM Transactions on Mathematical Software (to appear), http://www.cise.ufl.edu/research/sparse/matrices.