Journal of Computational Physics 231 (2012) 7730–7738

Contents lists available at SciVerse ScienceDirect

Journal of Computational Physics

journal homepage: www.elsevier.com/locate/jcp

A fast Galerkin method with efficient matrix assembly and storage for a peridynamic model

Hong Wang [a,b,*], Hao Tian [a]

[a] School of Mathematics, Shandong University, Jinan, Shandong 250100, China
[b] Department of Mathematics, University of South Carolina, Columbia, SC 29208, USA

ARTICLE INFO

Article history:
Received 18 February 2012
Received in revised form 4 June 2012
Accepted 10 June 2012
Available online 4 August 2012

Keywords:
Dense matrices
Fast methods
Peridynamics

ABSTRACT

Peridynamic theory provides an appropriate description of the deformation of a continuous body involving discontinuities or other singularities, which cannot be described properly by classical theory of solid mechanics. However, the operators in the peridynamic models are nonlocal, so the resulting numerical methods generate dense or full stiffness matrices. Gaussian types of direct solvers were traditionally used to solve these problems, which requires $O(N^3)$ of operations and $O(N^2)$ of memory where N is the number of spatial nodes. This imposes significant computational and memory challenge for a peridynamic model, especially for problems in multiple space dimensions. A simplified model, which assumes that the horizon of the material $\delta = O(N^{-1})$, was proposed to reduce the computational cost and memory requirement to $O(N)$. However, the drawback is that the corresponding error estimate becomes one-order suboptimal. Furthermore, the assumption of $\delta = O(N^{-1})$ does not seem to be physically reasonable since the horizon δ represents a physical property of the material that should not depend on computational mesh size.

We develop a fast Galerkin method for the (non-simplified) peridynamic model by exploiting the structure of the stiffness matrix. The new method reduces the computational work from $O(N^3)$ required by the traditional methods to $O(N\log^2 N)$ and the memory requirement from $O(N^2)$ to $O(N)$ without using any lossy compression. The significant computational and memory reduction of the fast method is better reflected in numerical experiments. When solving a one-dimensional peridynamic model with $2^{14} = 16,384$ unknowns, the traditional method consumed CPU time of 6 days and 11 h while the fast method used only 3.3 s. In addition, on the same computer (with 128 GB memory), the traditional method with a Gaussian elimination or conjugate gradient method ran out of memory when solving the problem with $2^{16} = 131,072$ unknowns. In contrast, the fast method was able to solve the problem with $2^{28} = 268,435,456$ unknowns using 3 days and 11 h of CPU time. This shows the benefit of the significantly reduced memory requirement of the fast method.

1. Introduction

Classical theory of solid mechanics assumes that all internal forces act through zero distance, which leads to mathematical models described by partial differential equations. These models do not provide a proper description of problems with spontaneous formation of discontinuities which conclude points with singularity. Peridynamic model was proposed as a

* Corresponding author at: Department of Mathematics, University of South Carolina, Columbia, SC 29208, USA.
E-mail address: hwang@math.sc.edu (H. Wang).

0021-9991/$ - see front matter © 2012 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.jcp.2012.06.009

reformation of solid mechanics [11], which leads to a non-local framework that does not explicitly involve the notion of deformation gradients.

Extensive research has been conducted on peridynamic theory [5–7,9–11,13,12,14–19]. In particular, finite element methods have been developed for peridynamic models [3,7,10] with proved quasi-optimal order error estimates [5,19]. In contrast to those for classical models for solid mechanics, the finite element methods for peridynamic models usually generate dense or full stiffness matrices which typically require $O(N^3)$ operations and $O(N^2)$ memory storage with N being the number of unknowns. A simplified peridynamic model was proposed to reduce the computational cost and memory requirement of the finite element method, in which the horizon of the material δ in the peridynamic model was assumed to be $\delta = O(N^{-1})$ [3]. The advantage of the simplified model is that it reduced the computational cost and memory requirement to $O(N)$, but at the cost of a reduced convergence rate of only first-order for linear finite element method.

In this paper we study a fast Galerkin method for the (non-simplified) peridynamic model by fully utilizing the structure of the stiffness matrix, which reduces the computational cost from $O(N^3)$ to $O(N\log^2 N)$ and memory requirement from $O(N^2)$ to $O(N)$ without using any lossy compression. In the context of a one-dimensional peridynamic model with $2^{14} = 16,384$ unknowns, the traditional method consumed CPU time of 6 days and 11 h while the fast method used only 3.3 s. In addition, on the same computer (with 128 GB memory), the traditional method with Gaussian elimination or conjugate gradient method ran out of memory for the problem with $2^{16} = 131,072$ unknowns. In contrast, the fast method solved the problem with $2^{28} = 268,435,456$ unknowns using 3 days and 11 h of CPU time. This shows the benefit of the significantly reduced memory requirement of the fast method. The rest of the paper is organized as follows. In Section 2 we go over the finite element method for the peridynamic model. In Section 3 we develop a fast method with an efficient matrix assembly and storage. In Section 4 we conduct numerical experiments to investigate the computational benefits of the fast method.

2. A peridynamic model and its finite element discretization

We outline a Galerkin finite element method for a steady-state peridynamic model.

2.1. A peridynamic model and its Galerkin formulation

A linear steady-state peridynamic model for microelastic materials on a finite bar is given by the following pseudo-differential equation [9,11]

$$\begin{cases} \int_\alpha^\beta \frac{u(x)-u(y)}{|y-x|}dy = b(x), & x \in (\alpha,\beta), \\ u(\alpha) = u_\alpha, & u(\beta) = u_\beta. \end{cases} \tag{1}$$

Here $b(x)$ and $u(x)$ represent the prescribed forcing term and the displacement of the material, respectively.

Let $H^\kappa(\alpha,\beta)$, with $\kappa > 1/2$, be the fractional Sobolev space of order κ and $H_0^\kappa(\alpha,\beta)$ be its subspace with trace 0 at the boundary α and β. We multiply the governing equation in (1) by any function $v \in H_0^\kappa(\alpha,\beta)$ and integrate the resulting equation on (α,β) to obtain the following Galerkin formulation: Seek $u \in H^\kappa(\alpha,\beta)$ with $u(\alpha) = u_\alpha$ and $u(\beta) = u_\beta$ such that

$$\begin{aligned} &a(u,v) = l(v), \quad \forall v \in H_0^\kappa(\alpha,\beta), \\ &a(u,v) := \int_\alpha^\beta v(x)\int_\alpha^\beta \frac{u(x)-u(y)}{|y-x|}dy\,dx, \quad l(v) := \int_\alpha^\beta b(x)v(x)dx. \end{aligned} \tag{2}$$

It was proved that the bilinear form $c(\cdot,\cdot)$ is coercive and bounded on some nonconventional Hilbert space [9,19]. The norm equipped on this space is equivalent to the L^2 norm in the two and three space dimensions. But the equivalence is not true in the current case of one space dimension (see also Table 2 in Section 5).

By the symmetry of x and y the bilinear form $a(u,v)$ can be rewritten as

$$a(u,v) = \int_\alpha^\beta\int_\alpha^\beta \frac{v(x)(u(x)-u(y))}{|y-x|}dy\,dx = \int_\alpha^\beta\int_\alpha^\beta \frac{v(y)(u(y)-u(x))}{|y-x|}dy\,dx, \tag{3}$$

which concludes that

$$\int_\alpha^\beta\int_\alpha^\beta \left(\frac{v(x)(u(x)-u(y)) - v(y)(u(y)-u(x))}{|y-x|}\right)dy\,dx = 0. \tag{4}$$

The numerator of the integrand in (4) can be decomposed as

$$\begin{aligned} v(x)(u(x)-u(y)) - v(y)(u(y)-u(x)) &= v(x)[(u(x)-u(y)) - (u(y)-u(x))] - (v(y)-v(x))(u(y)-u(x)) \\ &= 2v(x)(u(x)-u(y)) - (v(y)-v(x))(u(y)-u(x)). \end{aligned} \tag{5}$$

We incorporate (5) into (4) to derive an alternative expression for $a(u,v)$

$$a(u,v) = \int_\alpha^\beta\int_\alpha^\beta \frac{(u(y)-u(x))(v(y)-v(x))}{2|y-x|}dy\,dx, \tag{6}$$

which shows the symmetry of $a(u, v)$.

2.2. A finite element discretization

A finite element method was developed in [3,10], in which a uniform spatial partition of (α, β)

$$x_i := \alpha + ih, \quad 0 \leqslant i \leqslant N, \quad h := \frac{\beta - \alpha}{N} \tag{7}$$

was assumed for a given positive integer N. Let S^h be the linear finite element space defined on $[\alpha, \beta]$ with respect to the given partition (7). Let $\{\phi_j(x)\}_{j=0}^N$ be the standard "hat" basis functions for S^h. Let $u^h \in S^h$ be the finite element approximation to the true solution u of problem (1). Then u^h can be expressed in terms of the hat functions $\{\phi_j\}_{j=0}^N$ as follows

$$u^h(x) = \sum_{j=1}^{N-1} u_j \phi_j(x) + u(\alpha)\phi_0(x) + u(\beta)\phi_N(x). \tag{8}$$

If we choose the test function $v^h(x) = \phi_i(x)$ for $i = 1, \ldots, N-1$, then the finite element formulation derived from the weak formulation (2) is expressed as follows

$$\sum_{j=1}^{N-1} a(\phi_i, \phi_j) u_j = l(\phi_i) - a(\phi_i, \phi_0)u(\alpha) - a(\phi_i, \phi_N)u(\beta), \quad 1 \leqslant i \leqslant N-1. \tag{9}$$

Let $u := [u_1, u_2, \ldots, u_{N-1}]^T$, $f := [f_1, f_2, \ldots, f_{N-1}]^T$, and $A := [A_{ij}]_{i,j=1}^{N-1}$ with A_{ij} and f_i being defined by

$$\begin{aligned} A_{ij} &:= \frac{1}{h} a(\phi_i, \phi_j), \quad 1 \leqslant i \leqslant N-1, \quad 0 \leqslant j \leqslant N, \\ f_i &:= \frac{1}{h}\left(l(\phi_i) - a(\phi_i, \phi_0)u(\alpha) - a(\phi_i, \phi_N)u(\beta)\right). \end{aligned} \tag{10}$$

Then the finite element method (9) can be expressed in a matrix form

$$Au = f. \tag{11}$$

2.3. An error estimate and its impact on computational complexity

The following error estimate was proved in [19] for the linear finite element method under the assumption that the true solution $u \in H^2(\alpha, \beta)$. For any $0 < \varepsilon \ll 1$, there exists a positive constant $C = C(\varepsilon)$ which is independent of h such that

$$\|u - u^h\|_{L^2(\alpha,\beta)} \leqslant Ch^{2-\varepsilon}\|u\|_{H^2(\alpha,\beta)}. \tag{12}$$

The estimate looks similar to that for the finite element method for classical second-order differential equations in which the stiffness matrix is a sparse matrix, which has $O(N)$ nonzero entries. In the case of a finite bar, the coefficient matrix is tridiagonal, which can be inverted in $O(N)$ operations. However, in the current peridynamic model for the finite bar, the displacement $u(x)$ at location x is determined by the value of u over the interval (α, β). This yields a full coefficient matrix.

The combination of this observation with the estimate (12) shows that the finite element method to the peridynamic model (1) can reach a quasi-optimal order convergence rate $O(h^{2-\varepsilon})$. But the coefficient matrix of the finite element method of the peridynamic model is dense or full, the computational cost of inverting the stiffness matrix of the finite element method is $O(N^3)$ with a memory requirement of $O(N^2)$. This represents a significant increase of computational cost and memory requirement of the finite element method compared to its analogue for the classical models by partial differential equations.

A simplified peridynamic model was proposed to reduce its computational cost, by introducing a parameter δ that describes the horizon of the material and further assuming $\delta = Mh$ with M being a fixed positive integer [3]. The benefit of this assumption is that it significantly reduces the computational cost of the finite element method to $O(M^2N)$ and the memory requirement to (MN). It is not clear how this simplification is asymptotically consistent with the physics, as the horizon of the material is a physical property of the material of the finite bar and should be independent of the computational mesh size h. This inconsistency is also reflected in the error estimate of the corresponding finite element method, which now reduced to the following first-order estimate

$$\|u - u^h\|_{L^2(\alpha,\beta)} \leqslant Ch\|u\|_{H^2(\alpha,\beta)}. \tag{13}$$

In this paper we develop a fast solution method with an efficient matrix assembly and memory requirement for the finite element method for the full peridynamic model (1). The fast method results in a computational cost of $O(N\log^2 N)$ and memory requirement of $O(N)$, while still retaining the full convergence rate of $O(h^{2-\varepsilon})$ of the method.

3. A fast solution method

We present a fast solution method to solve the linear system (11).

3.1. A second look at the conjugate gradient method

The conjugate gradient method for (11) is formulated as follows [1]: Let u_0 be an initial guess, then compute $r_0 := f - Au_0$ and $d_1 := r_0$

for $k = 2, 3, \ldots$
$\gamma_k := r_{k-1}^T r_{k-1} / r_{k-2}^T r_{k-2}$
$d_k := r_{k-1} + \gamma_k d_{k-1}$
$\omega_k := r_{k-1}^T r_{k-1} / d_k^T A d_k$
$u_k := u_{k-1} + \omega_k d_k$
$r_k := r_{k-1} - \omega_k A d_k$
Check for convergence; continue if necessary
end
$u := u_k$

In the CG formulation, the evaluation of the matrix–vector multiplication Ad_k requires $O(N^2)$ computational work. Furthermore, the evaluation and storage of the coefficient matrix A also requires $O(N^2)$ of computational work and memory requirement. All other computations in the CG formulation require only $O(N)$ computational work and $O(N)$ memory. Therefore, we need only to accelerate the matrix–vector multiplication Ad for any vector d and to store A efficiently.

3.2. Efficient evaluation and storage of the coefficient matrix

To develop a mechanism for the efficient evaluation and storage of the matrix A and fast matrix–vector multiplication Ad, we carefully explore the structure of the stiffness matrix A. By the symmetry of the coefficient matrix, we need only evaluate the upper triangular part of the matrix.

For $j \geqslant i+2$, the entries A_{ij} of the matrix are given by

$$A_{ij} = \frac{1}{h}\left(\int_{x_{i-1}}^{x_i} \frac{x - x_{i-1}}{h} \int_{x_{j-1}}^{x_j} \frac{0 - \frac{y - x_{j-1}}{h}}{y - x} dy\, dx + \int_{x_{i-1}}^{x_i} \frac{x - x_{i-1}}{h} \int_{x_j}^{x_{j+1}} \frac{0 - \frac{x_{j+1} - y}{h}}{y - x} dy\, dx + \int_{x_i}^{x_{i+1}} \frac{x_{i+1} - x}{h} \int_{x_{j-1}}^{x_j} \frac{0 - \frac{y - x_{j-1}}{h}}{y - x} dy\, dx \right.$$
$$\left. + \int_{x_i}^{x_{i+1}} \frac{x_{i+1} - x}{h} \int_{x_j}^{x_{j+1}} \frac{0 - \frac{x_{j+1} - y}{h}}{y - x} dy\, dx \right), \tag{14}$$

$$= -\int_0^1 t \int_0^1 \frac{s}{j - i + s - t} ds\, dt - \int_0^1 t \int_0^1 \frac{s}{j - i + 2 - s - t} ds\, dt - \int_0^1 t \int_0^1 \frac{s}{j - i - 2 + s + t} ds\, dt$$
$$- \int_0^1 t \int_0^1 \frac{s}{j - i - s + t} ds\, dt. \tag{15}$$

On the right-hand side of the second equal sign, $t = (x - x_{i-1})/h$ in the first two integrals and $t = (x_{i+1} - x)/h$ in the last two integrals. $s = (y - x_{j-1})/h$ in the first and third integrals and $s = (x_{j+1} - y)/h$ in the second and fourth integrals.

We evaluate these integrals to obtain

$$A_{i,i+2} = 18 \ln 3 - \frac{88}{3} \ln 2 \tag{16}$$

and for $j \geqslant i+3$

$$A_{ij} = -\frac{1}{6}\left((j-i)^3 - 6(j-i)^2 + 12(j-i) - 8\right) \ln\left(1 - \frac{2}{j-i}\right) + \frac{2}{3}\left((j-2)^3 - 3(j-i)^2 + 3(j-i) - 1\right) \ln\left(1 - \frac{1}{j-i}\right)$$
$$+ \frac{2}{3}\left((j-2)^3 + 3(j-i)^2 + 3(j-i) + 1\right) \ln\left(1 + \frac{1}{j-i}\right)$$
$$- \frac{1}{6}\left((j-i)^3 + 6(j-i)^2 + 12(j-i) + 8\right) \ln\left(1 + \frac{2}{j-i}\right). \tag{17}$$

That is, the entries on each descending diagonal from left to right but the tridiagonals are constant. The entries on the tridiagonals are given by

$$A_{i,i} = \frac{(i-1)^3}{3}\ln\left(1+\frac{1}{i-1}\right) - \frac{(i+1)^3}{3}\ln\left(1-\frac{1}{i+1}\right) - \frac{2i^2}{3} + \frac{2\ln i}{3} + \frac{(N-i-1)^3}{3}\ln\left(1+\frac{1}{N-i-1}\right) - \frac{(N-i+1)^3}{3}\ln\left(1-\frac{1}{N-i+1}\right) - \frac{2(N-i)^2}{3} + \frac{2\ln(N-i)}{3} - \frac{8\ln 2}{3}, \quad 2 \leqslant i \leqslant N-2, \tag{18}$$

$$A_{1,1} = \frac{(N-2)^3}{3}\ln\left(1+\frac{1}{N-2}\right) - \frac{N^3}{3}\ln\left(1-\frac{1}{N}\right) - \frac{2(N-1)^2}{3} + \frac{2\ln(N-1)}{3} - \frac{2}{3}, \tag{19}$$

$$A_{N-1,N-1} = \frac{(N-2)^3}{3}\ln\left(1+\frac{1}{N-2}\right) - \frac{N^3}{3}\ln\left(1-\frac{1}{N}\right) - \frac{2(N-1)^2}{3} + \frac{2\ln(N-1)}{3} - \frac{2}{3}. \tag{20}$$

The super-diagonal entries are given by

$$A_{i,i+1} := \frac{1}{6}\left((-3i^2-2i+1)\ln\left(1+\frac{1}{i}\right) + 2(N-i)^2 - 2(N-i) + 2i^2 + 2i + (-2(N-i)^3 + 3(N-i)^2)\ln\left(1+\frac{1}{N-1-i}\right) + \ln i - 4 + \ln(N-i-1) + 4\ln 2\right) + 1 + \frac{14\ln 2}{3} - \frac{9\ln 3}{2}. \tag{21}$$

Thus, instead of evaluating and storing $N(N-1)/2$ matrix entries, we need only evaluate and store $3N$ entries. Finally, the coercivity of the bilinear form $a(\cdot,\cdot)$ implies the positive-definiteness of the coefficient matrix A. We emphasize that the storage scheme actually stores the stiffness matrix A^m exactly and is not a lossy compression.

3.3. A fast matrix–vector multiplication

It remains to develop a fast matrix–vector multiplication algorithm to evaluate Ad for any vector d. We notice that the stiffness matrix A is a symmetric and positive-definite tridiagonal-plus-Toeplitz matrix, while it is known that a fast Toeplitz-matrix–vector multiplication can be achieved. We therefore split the matrix A as follows:

$$A = A_{tr} + A_o. \tag{22}$$

Here A_{tr} consists of all the tridiagonal entries of A and zero elsewhere, and $A_o = [A^o_{i,j}]_{i,j=1}^{N-1}$ contains all remaining nonzero entries of A and is a symmetric Toeplitz matrix. We let q_{j-i} denote the common entry in the $(j-i)$th descending diagonal of A_o from left to right. Namely,

$$A^o_{i,j} = q_{j-i}, \quad j \geqslant i. \tag{23}$$

Then the symmetric Toeplitz matrix A_o can be embedded into a $2N \times 2N$ circulant matrix C as follows [2,8]

$$C := \begin{pmatrix} A_o & B \\ B & A_o \end{pmatrix} \quad B := \begin{pmatrix} 0 & q_{N-2} & \cdots & q_2 & q_1 \\ q_{N-2} & 0 & q_{N-2} & \cdots & q_2 \\ \vdots & q_{N-2} & 0 & \ddots & \vdots \\ q_2 & \vdots & \ddots & \ddots & q_{N-2} \\ q_1 & q_2 & \cdots & q_{N-2} & 0 \end{pmatrix}. \tag{24}$$

The circulant matrix C has the following decomposition [4,8]

$$C = F^{-1}\mathrm{diag}(Fc)F, \tag{25}$$

where c is the first column vector of C and F is the $2(N-1) \times 2(N-1)$ discrete Fourier transform matrix. It is well known that the matrix–vector multiplication Fw for $w \in \mathbb{R}^{2N-2}$ can be carried out in $O(2N\log(2N)) = O(N\log N)$ operations via the fast Fourier transform (FFT). Eq. (25) shows that Cw can be evaluated in $O(N\log N)$ operations. So (24) implies that $A_o d$ can be evaluated in $O(N\log N)$ operations for any $d \in \mathbb{R}^{N-1}$. On the other hand, since A_{tr} is a symmetric tridiagonal matrix, $A_{tr}d$ can be evaluated in $O(N\log N)$ operations. The combination of these results shows that Ad can be evaluated in $O(N\log N)$ operations for any $d \in \mathbb{R}^{N-1}$. So the conjugate gradient method can be evaluated in $O(N\log N)$ operations per iteration! The numerical experiments in Section 4 indicate that the number of iterations is $O(\log N)$, which suggests that the overall computational cost of the fast conjugate gradient method is $O(N\log^2 N)$. Finally we point out that the fast matrix–vector multiplication is computed exactly without using any lossy compression.

4. Numerical experiments

We conduct numerical experiments to investigate the performance of the fast methods. In the numerical example runs, the spatial domain $(\alpha, \beta) = (0, 1)$. A choice of $u(x) = x^2(1-x)^2$ corresponds to

$$b(x) = \frac{25}{6}x^4 - \frac{25}{3}x^3 + \frac{9}{2}x^2 - \frac{1}{3}x - \frac{1}{12}. \tag{26}$$

We use Gaussian elimination, conjugate gradient (CG) method, and the fast conjugate gradient (FCG) method to solve the system (11) to investigate the performance of these methods. We implement these methods in Matlab and run all the experiments on a 128 GB computer. In Table 1 we present the L^1, L^2 and L^∞ errors of the numerical solutions solved by these methods with gradually decreasing mesh size h from $h = 2^{-3}$ to $h = 2^{-11}$. We observe that all of the three methods generate numerical solutions with the same accuracy. We use a linear regression to fit the convergence rates and the associated constants in the estimates

$$\|u_h - u\|_{L^p(\alpha,\beta)} \leqslant M_\gamma h^\gamma, \quad p = 2, \infty. \tag{27}$$

We find that these methods have second-order convergence rates. Due to the effect of round-off errors and the already very small truncation errors on the order of 10^{-8}, the numerical solutions gradually lose the second-order convergence as the mesh size h is further refined.

In Table 2 we present the CPU time consumed by all the three methods, and the number of iterations in the conjugate gradient method and the fast conjugate gradient method for gradually reduced mesh size h. We drop the CPU times consumed by these methods for the coarse mesh size $h = 2^{-3}$ to 2^{-7} since the corresponding CPU times are almost not detectable. In contrast, we run exhaustive numerical methods to solve the system (11) for as fine mesh size h as possible on the same computer with 128 GB memory. We measure the CPU times consumed by the three methods at these very fine meshes and the number of iterations needed by the conjugate gradient method and the fast conjugate gradient method. From the accuracy point of view, these extremely fine mesh sizes are computationally unnecessary. The purpose of these exhaustive

Table 1
Convergence of the Gaussian elimination, the conjugate gradient (CG) method, and the fast conjugate gradient (FCG) method.

	h	L_2	L_∞
Gauss	2^{-3}	2.5000×10^{-3}	3.5396×10^{-3}
	2^{-4}	6.8787×10^{-4}	9.4787×10^{-4}
	2^{-5}	1.7844×10^{-4}	2.4422×10^{-4}
	2^{-6}	4.5329×10^{-5}	6.1855×10^{-5}
	2^{-7}	1.1404×10^{-5}	1.5543×10^{-5}
	2^{-8}	2.8559×10^{-6}	3.8912×10^{-6}
	2^{-9}	7.1168×10^{-7}	9.7030×10^{-7}
	2^{-10}	1.7343×10^{-7}	2.3766×10^{-7}
	2^{-11}	3.5497×10^{-8}	5.0572×10^{-8}
CG	2^{-3}	2.5000×10^{-3}	3.5396×10^{-3}
	2^{-4}	6.8787×10^{-4}	9.4787×10^{-4}
	2^{-5}	1.7844×10^{-4}	2.4422×10^{-4}
	2^{-6}	4.5329×10^{-5}	6.1855×10^{-5}
	2^{-7}	1.1404×10^{-5}	1.5543×10^{-5}
	2^{-8}	2.8559×10^{-6}	3.8912×10^{-6}
	2^{-9}	7.1168×10^{-7}	9.7030×10^{-7}
	2^{-10}	1.7343×10^{-7}	2.3766×10^{-7}
	2^{-11}	3.5497×10^{-8}	5.0572×10^{-8}
FCG	2^{-3}	2.5000×10^{-3}	3.5396×10^{-3}
	2^{-4}	6.8787×10^{-4}	9.4787×10^{-4}
	2^{-5}	1.7844×10^{-4}	2.4422×10^{-4}
	2^{-6}	4.5329×10^{-5}	6.1855×10^{-5}
	2^{-7}	1.1404×10^{-5}	1.5543×10^{-5}
	2^{-8}	2.8559×10^{-6}	3.8912×10^{-6}
	2^{-9}	7.1168×10^{-7}	9.7030×10^{-7}
	2^{-10}	1.7343×10^{-7}	2.3766×10^{-7}
	2^{-11}	3.5497×10^{-8}	5.0572×10^{-8}
M_γ		0.1792	0.2479
γ		2.0034	2.0034

Table 2
The CPU consumed by Gaussian elimination, conjugate gradient (CG) method, and the fast conjugate gradient (FCG) method for different mesh sizes.

h	Gauss	CG		FCG	
	CPU (s)	Iterations	CPU (s)	Iterations	CPU (s)
2^{-8}	1.01	32	0.10	32	0.09
2^{-9}	6.25	34	0.31	34	0.12
2^{-10}	46.8	36	1.19	36	0.18
2^{-11}	383 = 6 m 23 s	38	5.17	38	0.32
2^{-12}	3852 = 1 h 4 m	40	33.09	40	0.69
2^{-13}	40,123 = 11 h 9 m	43	137 = 2 m 17 s	43	1.50
2^{-14}	558,190 = 6 days 11 h	46	9 m 50 s	46	3.30
2^{-15}	We terminated test	49	2512 = 41 m 52 s	49	8.08
2^{-16}	We terminated test	53	11,311 = 3 h 9 m	53	17.84
2^{-17}	Out of memory	N/A	Out of memory	56	89 = 1 m 29 s
2^{-18}	N/A	N/A	N/A	58	171 = 2 m 51 s
2^{-19}	N/A	N/A	N/A	61	360 = 6 m
2^{-20}	N/A	N/A	N/A	64	695 = 11 m 35 s
2^{-21}	N/A	N/A	N/A	67	1282 = 21 m 22 s
2^{-22}	N/A	N/A	N/A	70	2991 = 49 m 51 s
2^{-23}	N/A	N/A	N/A	74	6409 = 1 h 47 m
2^{-24}	N/A	N/A	N/A	78	11,339 = 3 h 9 m
2^{-25}	N/A	N/A	N/A	81	22,024 = 6 h 7 m
2^{-26}	N/A	N/A	N/A	85	60,039 = 16 h 41 m
2^{-27}	N/A	N/A	N/A	89	142,040 = 1 day 15 h
2^{-28}	N/A	N/A	N/A	92	302,740 = 3 days 11 h

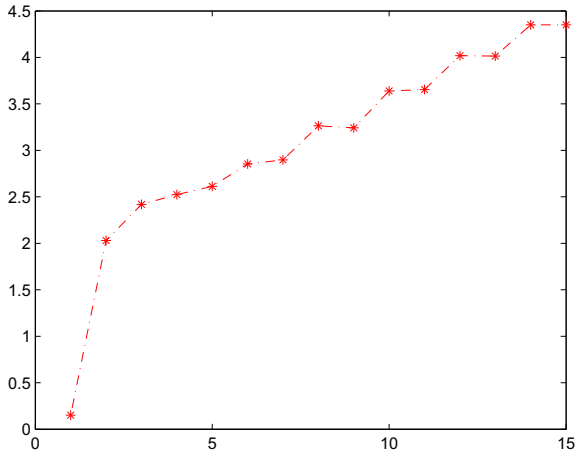

Fig. 1. Distribution of eigenvalues under $N = 16$. The smallest eigenvalue is 0.1514, and the largest one is 4.3504. The corresponding condition number is 28.7345.

experiments is to investigate the savings of the CPU times and the memory requirement of the fast conjugate gradient method over Gaussian elimination which has been the solver used in the numerical methods for peridynamic models. In particular, these exhaustive experiments is crucial in the study of the convergence behavior of the fast conjugate gradient method as the number of unknowns increases significantly. From these results we observe that with a fine mesh size $h = 2^{-14}$, i.e., 16,384 nodes, Gaussian elimination consumed CPU of 6 days and 11 h. With the same size, conjugate gradient methods consumed CPU of only 9 min and 50 s while the fast conjugate gradient method used only 3.3 s. We indeed ran numerical experiments using the conventional Galerkin method with Gauss elimination on finer mesh size h but did not complete these runs due to the significantly increased CPU time. We also observed that as we reduced the mesh size h to $h = 2^{-16}$, i.e., 65,536 nodes, conjugate gradient method consumed a CPU time of 3 h and 9 min, while the fast conjugate gradient method consumed CPU time of only 18 s. Moreover, as we further reduced the mesh size h to $h = 2^{-17}$, both Gaussian elimination and conjugate gradient method ran out of memory, while the fast conjugate gradient method solved the problem using CPU time of 1 min and 29 s. We continued to refine the mesh size h all the way to $h = 2^{-28}$, i.e., 268,435,456 nodes, the fast

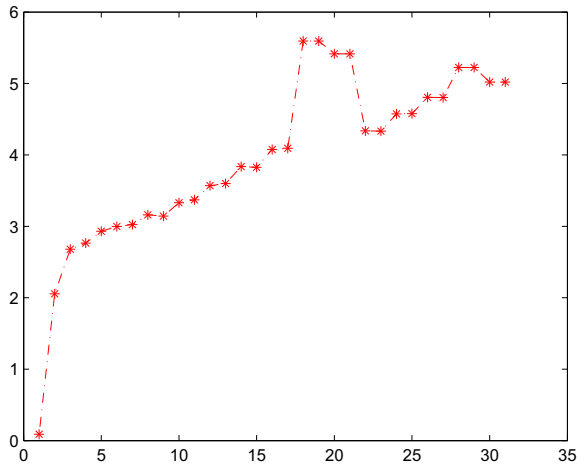

Fig. 2. Distribution of eigenvalues under $N = 32$. The smallest eigenvalue is 0.0877, and the largest one is 5.5941. The corresponding condition number is 63.7868.

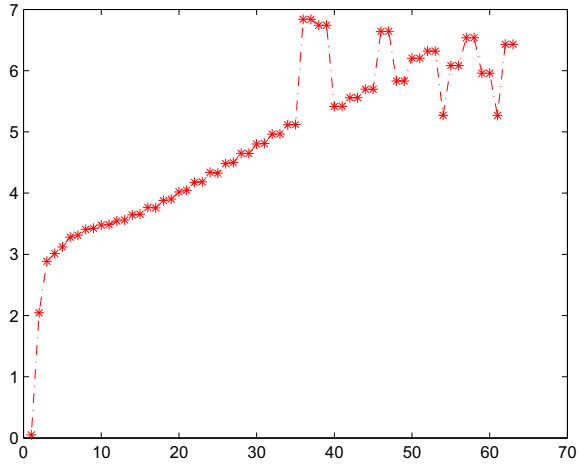

Fig. 3. Distribution of eigenvalues under $N = 64$. The smallest eigenvalue is 0.05, and the largest one is 6.8389. The corresponding condition number is 136.7780.

conjugate gradient method completed the simulation in 3 days and 11 h of CPU times. Moreover, the number of iterations had been steadily increasing logarithmically with respect to N without any preconditioner or special treatment involved. The fast method ran out of memory at $h = 2^{-29}$, i.e., 536,870,912 nodes.

From Table 2 we also observed that the number of iterations in the conjugate gradient method and the fast conjugate gradient method increases logarithmically with respect to the number of unknowns. This implies that the stiffness matrix is just weakly ill-conditioned, see Figs. 1–3. A regular conjugate gradient method has a computational cost of $O(N^2)$ per iteration and for the evaluation of the coefficient matrix A, and so has an overall computational cost of $O(N^2 \log N)$. This represents one order less of computational work compared to Gaussian elimination and has been reflected in the CPU time consumed. These show that for the numerical methods for peridynamic models, iterative methods are more efficient than Gaussian elimination, even though the stiffness matrix is dense or full. The fast conjugate gradient method further reduces the evaluation of the coefficient matrix to $O(N)$ and the computational work to $O(N \log N)$ per iteration, which leads to an overall computational work of $O(N\log^2 N)$ since the number of iterations is $O(\log N)$, as observed in Table 2.

In the numerical experiments the Gaussian elimination, the conjugate gradient squared method, and the fast conjugate gradient squared method were all implemented using the Matlab subroutines without any fine tuning. A properly "tuned" implementation of Gaussian elimination method, e.g., a parallel version, might speed up its numerical performance significantly. The same is of course true for conjugate gradient squared method and the fast conjugate gradient squared method. The corresponding comparison of the "fine tuned" versions of the methods is beyond the scope of this paper.

5. Concluding remarks

Peridynamic theory provides an appropriate description of the deformation of a continuous body involving discontinuities or other singularities, which cannot be described properly by classical theory of solid mechanics. However, peridynamic models are nonlocal and so yield numerical schemes with dense or full coefficient matrices. Gaussian types of direct solvers were traditionally used solve these problems, which requires $O(N^3)$ of operations and $O(N^2)$ of memory where N is the number of spatial nodes. This imposes significant computational and memory challenge for a peridynamic model, especially for problems in multiple space dimensions.

In this paper we develop a fast linear Galerkin finite element method for the (non-simplified) peridynamic model. The new method reduces the computational work from $O(N^3)$ required by the traditional methods to $O(N\log N)$ per iteration. Numerical experiments show that the number of iterations increases logarithmically with N. Thus, this leads to an overall computational cost of $O(N\log^2 N)$ without any lossy compression used in the computations. Moreover, the fast method reduces the memory requirement from $O(N^2)$ to $O(N)$ without any lossy compression used. Numerical experiments show the utility of the fast method.

In this paper we focus on linear finite element approximation in the development for the simplicity of the expression and implementation. In fact, the fast method can also be developed for high-order finite element methods. In this case, the coefficient matrix still has a Toeplitz structure except that the number of diagonals in which the entries are not constant increases as the degree of finite element approximations increases. See Figs.1–3.

Acknowledgments

The authors would like to express their sincere thanks to Professors Max Gunzburger, Qiang Du, and Lili Ju for the very helpful discussions. The authors would like to express their sincere thanks to the referees for their very helpful comments and suggestions, which greatly improved the quality of this paper. This work was supported in part by the National Science Foundation under Grant No. EAR-0934747 and by the National Natural Science Foundation of China under the Grant No. 91130010.

References

[1] F. Bobaru, M. Yang, L. Alves, S. Silling, E. Askari, J. Xu, Convergence adaptive refinement and scaling in 1D peridynamics, Int. J. Numer. Methods Engrg. 77 (2009) 852–857.
[2] A. Böttcher, B. Silbermann, Introduction to Large Truncated Toeplitz Matrices, Springer, New York, 1999.
[3] X. Chen, M. Gunzburger, Continuous and discontinuous finite element methods for a peridynamics model of mechanics, Comput. Methods Appl. Mech. Engrg. 200 (2011) 1237–1250.
[4] P.J. Davis, Circulant Matrices, Wiley-Intersciences, New York, 1979.
[5] Q. Du, K. Zhou, Mathematical analysis for the peridynamic nonlocal continuum theory, ESIAM M2AN Math. Mod. Numer. Anal. 45 (2011) 217–234.
[6] E. Emmrich, O. Weckner, Analysis and numerical approximation of an integrodifferential equation modeling non-local effects in linear elasticity, Math. Methods Solids 12 (2007) 363–384.
[7] E. Emmrich, O. Weckner, The peridynamic equation and its spatial discretization, Math. Model. Anal. 12 (2007) 17–27.
[8] R.M. Gray, Toeplitz and circulant matrices: a review, Found. Trends Commun. 2 (2006) 155–239.
[9] M. Gunzburger, R. Lehoucq, A nonlocal vector calculus with application to nonlocal boundary value problems, Multiscale Model. Simul. 8 (2010) 1581–1598.
[10] R. Maceka, S. Silling, Peridynamics via finite element analysis, Finite Elem. Anal. Design 183 (2006) 365–372.
[11] S. Silling, Reformulation of elasticity theory for discontinuities and long-range forces, N. Mech. Phys. Solids 48 (2000) 175–209.
[12] S. Silling, E. Askari, A meshfree method based on the peridynamic model of solid mechanics, Comput. Struct. 83 (2005) 1584–1611.
[13] S. Silling, Linearized theory of peridynamic states, J. Elast. 99 (2010) 85–111.
[14] S. Silling, M. Epton, O. Weckner, J. Xu, E. Askari, Peridynamic states and constitutive modeling, J. Elast. 88 (2007) 151–184.
[15] S. Silling, R. Lehoucq, Peridynamic theory of solid mechanics, Adv. Appl. Mech. 44 (2010) 73–168.
[16] S. Silling, M. Zimmermann, R. Abeyaratne, Deformation of a peridynamic bar, J. Elast. 73 (2003) 173–190.
[17] O. Weckner, E. Emmrich, The effect of long-range forces on the dynamics of a bar, J. Mech. Phys. Solids 53 (2005) 705–728.
[18] O. Weckner, E. Emmrich, Numerical simulation of the dynamics of a nonlocal, inhomogeneous, infinite bar, J. Comput. Appl. Mech. 6 (2005) 311–319.
[19] K. Zhou, Q. Du, Mathematical and numerical analysis of linear peridynamic models with nonlocal boundary condition, SIAM N. Numer. Anal. 48 (2010) 1759–1780.