A COMPARISON OF DIFFERENT METHODS FOR CALCULATING TANGENT-STIFFNESS MATRICES IN A MASSIVELY PARALLEL COMPUTATIONAL PERIDYNAMICS CODE

APPROVED BY SUPERVISING COMMITTEE:

John T. Foster, Ph.D., chair

Harry R. Millwater, Ph.D.

Yusheng Feng, Ph.D.

Accepted:

Dean, Graduate School

DEDICATION

To my family, Christa, Brian and Mark, for their love, belief and support.

A COMPARISON OF DIFFERENT METHODS FOR CALCULATING TANGENT-STIFFNESS MATRICES IN A MASSIVELY PARALLEL COMPUTATIONAL PERIDYNAMICS CODE

by

MICHAEL BROTHERS, B.S.

THESIS
Presented to the Graduate Faculty of
The University of Texas at San Antonio
In Partial Fulfillment
Of the Requirements
For the Degree of

MASTER OF SCIENCE IN MECHANICAL ENGINEERING

THE UNIVERSITY OF TEXAS AT SAN ANTONIO
College of Engineering
Department of Mechanical Engineering
December 2013

ACKNOWLEDGEMENTS

I would like to thank John T. Foster not only for introducing me to the fields of high performance computing and computational mechanics, but for his indespensable advice and effort on my behalf.

I would also like to thank Harry R. Millwater for his help in creating the idea for the topic of this thesis as well as his invaluable input and support.

Additionally I would like to thank Yusheng Feng whose lessons in linear algebra and complex analysis were helpful.

This work was funded in part by grants from the United States Air Force Office of Scientific Research grant number W911NF-11-1-0208 and National Energy Technology Laboratory grant number DE-FE0010808. The authors acknowledge the Texas Advanced Computing Center (TACC) at The University of Texas at Austin for providing HPC resources that have contributed to the research results reported within this thesis. URL: http://www.tacc.utexas.edu

December 2013

A COMPARISON OF DIFFERENT METHODS FOR CALCULATING TANGENT-STIFFNESS MATRICES IN A MASSIVELY PARALLEL COMPUTATIONAL PERIDYNAMICS CODE

Michael Brothers, M.S.
The University of Texas at San Antonio, 2013

Supervising Professor: John T. Foster, Ph.D.

In order to maintain the quadratic convergence properties of the first-order Newton's method in quasi-static nonlinear analysis of solid structures it is crucial to obtain accurate, algorithmically consistent tangent-stiffness matrices. For an extremely small class of nonlinear material models, these consistent tangent-stiffness operators can be derived analytically; however, most often in practice, they are found through numerical approximation of derivatives. A goal of the study described in this thesis was to establish the suitability of an under-explored method for computing tangent-stiffness operators, referred to here as 'complex-step'. Compared are four methods of numerical derivative calculation: automatic differentiation, complex-step, forward finite difference, and central finite difference in the context of tangent-stiffness matrix calculation in a massively parallel computational peridynamics code. The complex-step method was newly implemented in the peridynamics code for the purpose of this comparison. The methods were compared through *in situ* profiling of the code for Jacobian accuracy, solution accuracy, speed, efficiency, Newton's method convergence rate and parallel scalability. The performance data was intended to serve as practical guide for code developers and analysts faced with choosing which method best suit the needs of their application code. The results indicated that complex-step produces Jacobians very similar, as measured by a low l^2 norm of element wise difference, to automatic differentiation. The values for this accuracy metric computed for forward finite difference and central finite difference indicated orders of magnitude worse Jacobian accuracy than complex-step, but convergence

study results showed that convergence rate and solution was not strongly affected. Ultimately it was speculated that further studies on the effect of Jacobian accuracy may better accompany experiments conducted on plastic material models or towards the evaluation of approximate and Quasi-Newton's methods

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1: INTRODUCTION

In order to maintain the quadratic convergence properties of the first-order Newton's method [2] [34, Ch. 13] in quasi-static nonlinear analysis of solid structures it is crucial to obtain accurate, algorithmically consistent tangent-stiffness matrices. For an extremely small class of nonlinear material models, these consistent tangent-stiffness operators can be derived analytically; however, most often in practice, they are found through numerical approximation of derivatives.

A goal of this study was to develop and evaluate a new, accurate, and practical method for calculating tangent-stiffness matrices against established methods. This new method is based on a complex number Taylor series expansion and referred to as CTSE or the "complex-step" method. The distinction of 'accurate' is defined by comparison of the new method to popular finite differencing techniques used for computing tangent-stiffness matrices and with the exact to machine precision algorithmically consistent derivatives computed with *automatic differentiation*.

Comparative data regarding the accuracy and compute cycle timing for the complex step, finite-difference, central-difference, and automatic differentiation methods are included to aide developers and analysts in computational mechanics code design who are faced with implementing a general method for tangent stiffness matrix calculation. A comparison and discrimination of the methods was achieved through in-situ instrumentation of a massively parallel computational mechanics code.

The scope of the application component of the study was limited to a single material model, a perfectly elastic state-based peridynamic solid, previously implemented in the computational peridynamics code, *Peridigm* [20]. Identifying a specific application provided a practical framework for implementing the new method and solving engineering problems to generate the data needed to compare the methods. In particular, *Peridigm* was chosen because it combined several helpful characteristics: the utilization of Newton's method and tangent-stiffness matrices for solving nonlinear quasi-static problems, prior inclusion of finite-difference, central-difference, and automatic-differentiation methods needed for comparison to the new complex-step method implementation,

and being an agile-components code that makes use of distributed computing data structures via Trilinos [10] for efficient parallelization on large clusters. Agile-components is a software development term meaning to structure a software project for continuous development to respond to evolving specifications by making much use of techniques from object-oriented programming and reusing proven pre-existing code to reduce development time and increase reliability. The modularity of *Peridigm* allowed the adding of new features, such as required by the study.

The aim of this thesis is not only to introduce the new complex-step method in the context of evaluating tangent-stiffness matrices, but to serve as a review for other differentiation techniques useful for solving non-linear systems with Newton's method. After presenting background information on the underlying methods to be discussed, presented in this thesis are: A) a description of tangent-stiffness matrices, B) detailed directions for producing tangent-stiffness matrices with each the methods identified above, C) a description and justification of the new complex-step method for calculating tangent-stiffness matrices, D) a description of implementing complex-step in the *Peridigm* software, E) a description of the quantities-of-interest used to rank the methods, F) a presentation and analysis of the results of the comparative study, and G) conclusions and thoughts on potential future work. Finally, in the interest of replicable research, the reader is referred to the corresponding author's website for *Peridigm* C++ source-code and data necessary to reproduce the results presented here. Additionally, included on the website is an example serial C++ library with classes for solving non-linear systems using the techniques discussed here, as well as two validation and verification example problems which show how to use the library. The purpose of the serial C++ library is to demonstrate in a simpler manner than in the *Peridigm* code how one would use AD or complex-step to better encourage their use and discussion.

1.1 Differentiation Techniques

1.1.1 Forward and central difference

Since the derivation of first-order finite-difference techniques from Taylor series expansion is considered well known, it is not described here. Instead the reader is referred to [5, Chap. 4.1.3] or their favorite numerical methods text.

1.1.2 The "complex-step" method.

It is possible to approximate derivatives quite accurately with a technique based on a complex variable Taylor series expansion of a function. This method was first described by Lyness and Moler [17, 18] and has more recently been rediscovered [1, 19, 30, 33] for use in engineering analysis. The basic idea is that a model parameter can be be made complex and expanded in a Taylor series about a small perturbation along the imaginary axis by some arbitrary value h as follows

$$\begin{aligned} f(x+ih) =& f(x) + \frac{\partial f(x)}{\partial x}\frac{ih}{1!} \\ &+ \frac{\partial f^2(x)}{\partial x^2}\frac{(ih)^2}{2!} + \frac{\partial f^3(x)}{\partial x_n^3}\frac{(ih)^3}{3!} + \cdots , \end{aligned} \tag{1.1}$$

Taking the imaginary part of both sides of equation (1.1) and solving for the first derivative and ignoring terms of $\mathcal{O}\left(h^2\right)$ yields an estimate

$$\frac{\partial f(x)}{\partial x} \approx \frac{\text{Im}\left(f(x+ih)\right)}{h}. \tag{1.2}$$

Thus, an estimate of the first derivative of a function can be made by only utilizing one functional evaluation of a perturbed model parameter along the imaginary axis. The step-size h is arbitrary and can be made as small as practical (even to machine precision without the dangers of roundoff error as in finite-differences) to yield an accurate estimate of this derivative. The only disadvantage of this technique is the necessity of requiring the functions to accept complex numbers as argu-

ments. Appendix A includes a simple example that illustrates the use of the complex-step method for the derivative calculation of a function.

1.1.3 Automatic-differentiation

Automatic-differentiation (AD) is a computerized method for computing exact derivatives based the chain-rule from calculus. AD takes advantage of the fact that any mathematical function executed on a computer, no matter how complicated, is a "composition of simple operations" (add, multiply, power, transcendental and the like) each having known analytical derivatives [23]. For reference, the AD implementation used in the study is part of the *Sacado* package from the *Trilinos* agile component libraries developed at Sandia National Laboratories [11].

An AD system evaluates composition functions in the expected manner: it works by first evaluating the innermost function of the composition, then presenting that function's output as input to the next level function until all levels are complete, observing commonly expected order of operations. However, an AD system does additional work during a function evaluation in that as each nested function is evaluated, the function's partial derivative with respect to the designated variables of the given input is also calculated. This is possible because the elementary math functions are hard-coded into the AD source-code along with their analytical derivatives, and linked by special instructions, so that when the elementary math functions are called upon for computation, their partial derivatives may be computed and stored in a sequence. The AD system then multiplies the final sequence of partial derivatives together to produce the exact equivalent to taking a partial derivative of the corresponding composition function with respect to a designated variable at a particular value. It is obvious, but bears mentioning, that the AD system could simply store one value for partial derivatives, modifying it as appropriate for every function evaluation rather than keeping a sequence. This is significant because modifying a single value rather than keeping a list of values corresponding to every level of a composition function, which itself may not be the sole one needing to be evaluated, represents a savings in memory usage, which at large scales is an active concern. In the literature, the AD scheme described here is called *forward automatic-differentiation*.

Only forward AD will be covered here since it is the implementation used in *Sacado* and therefore in this study; however, the reader is referred to the introduction section of [7] and its citations for further information on AD, and particularly [8] which is foundational. Appendix B includes an example that walks the reader through the process a computer uses to compute derivatives via AD. Some things to note about AD are that no approximation of derivatives is being made because the analytical forms of the partials of the elementary math functions are defined alongside them. The accuracy of AD is then limited by the precision of the AD system's definition of the elementary math functions and their partial derivatives.

1.2 Tangent-stiffness

In solid and structural computational mechanics the tangent-stiffness matrix is a linearization operator that describes the stiffness of a system in response to small displacements imposed upon the current configuration of the system. Mathematically speaking, the tangent-stiffness represents the gradient of a high-dimensional energy surface that "points" in the minimum direction. While we've adopted the terminology of solid mechanics, it's important to note that these operators appear in other physical settings and are known by other names, e.g. a *transmissibility matrix* in the context of a Poisson problem or, more generally, a *Jacobian matrix* in mathematical optimization.

While, for simplicity, the examples shown in the sequel refer only to linear problems, tangent-stiffness matrices generally arise in the context of non-linear analysis where Newton's method or quasi-Newton's method's are used in the minimization of a given residual function. In this context, the tangent-stiffness matrix can be thought of as the linearization of the system about about a particular configuration. It has been shown [12, 13] that in order to preserve the quadratic convergence properties of the Newton's methods, it is necessary that this linearization is carried out in a manner that is *consistent* with the algorithmic constraint equations used when computing the internal forces that arise due to deformations. An example of these constraint equations would be the Kuhn-Tucker conditions [29] that are used in integration of a flow rule for plasticity modeling. If the continuum tangent-moduli are used in place of the *consistent* or *algorithmic tangent moduli*

in the solution of a non-linear solid mechanics problem, convergence may still be achieved, but not in the quadratic manner that makes Newton's method attractive for this class of problems. In the setting of non-linear solid mechanics, there is a very small class of material model algorithms in which consistent tangent-stiffness operators can be derived analytically. Perfect plasticity and plasticity with isotropic hardening used in combination with a general nearest-point projection or *radial return algorithm* are a few examples. Therefore, when general models (and certainly more complex ones) are implemented into a general purpose computational mechanics code, the tangent-stiffness operators are typically defined via a numerical approximation.

A tangent-stiffness matrix can be described as a collection of the first-order partial derivatives of a vector valued function with respect to each independent vector component of the function for a given vector. A general mathematical formula for a tangent-stiffness matrix, K_{ij}, can be expressed in indicial notation as

$$K_{ij} = \left. \frac{\partial F_i}{\partial X_j} \right|_{\vec{X}_0}, \tag{1.3}$$

where F_i is the i^{th} component of the vector valued function, X_j is the j^{th} component of the vector argument to F, and a particular value of the vector,$\vec{X}_0$ is chosen as the linearization point. One then evaluates the expression for each combination (i, j) corresponding to a (row, column) location in the tangent-stiffness matrix. The elements of a tangent-stiffness matrix can be estimated using any of the complex-step, AD, or common finite-difference techniques for functions $F : R^1 \rightarrow R^1$, since taking partial derivatives entails holding all but a single independent vector component of the function constant, and each element of the function can be evaluated independently of the others.

1.2.1 Differencing formulas for tangent-stiffness matrix evaluation.

The *forward-difference* (FD) formula for calculating a tangent-stiffness matrix in the setting of a solid structural mechanics problem is

$$K_{ij} = \frac{F_i^{int}(\vec{u} + h\hat{e}_j) - F_i^{int}(\vec{u})}{h}, \tag{1.4}$$

Where K_{ij} is the indicial notation representation of an element of the tangent-stiffness matrix at row i, column j. The first of the two terms in the numerator is the internal force, F_i^{int} evaluated in the current deformed configuration, $\vec{u}$, plus a small perturbation h in the direction $\hat{e}_j$ where $\hat{e}_j$ represents a unit-vector corresponding to the j^{th} degree-of-freedom. The second of the two terms is F_i^{int} evaluated in the current configuration. The denominator is the magnitude of perturbation called *probe-distance*.

The *central-difference* (CD) formula for calculating a tangent-stiffness matrix is

$$K_{ij} = \frac{F_i^{int}(\vec{u} + h\hat{e}_j) - F_i^{int}(\vec{u} - h\hat{e}_j)}{2h}. \tag{1.5}$$

Together, the FD and CD methods, are sometimes termed *finite-difference probing* techniques for tangent-stiffness calculation in literature [32]. From the well-known 1-dimensional formulas for FD and CD, it is understood that the accuracy of these expressions is dependent theoretically upon selecting a small step-size h, however experiments shown in [30] demonstrate that too small an h can lead to inaccuracy from 'subtractive-cancellation'. Therefor it is fair to say that selecting h in practice can be problematic.

The complex-step (CS) formula for calculating a tangent-stiffness matrix is

$$K_{ij} = \frac{\text{Im}(F_i^{int}(\vec{u} + ih\hat{e}_j))}{h}, \tag{1.6}$$

where the internal force, F^{int} is now treated as function of complex vectors, $\vec{u}$. The small pertur-

bation, h, is now carried out on the imaginary axis and only the real coefficient to the imaginary part of the vector returned by F^{int} is kept, hence the $\mathrm{Im}(\cdot)$ operation. As highlighted in Section 1.1.2, notice that there is no subtraction operation taking place; therefore h can be made very small to yield highly accurate derivatives. It should be mentioned that using complex-step to compute tangent-stiffness matrices was done in [21, 22], but those references did not investigate the performance of complex-step in a massively-parallel computational peridynamics code.

When using automatic-differentiation to calculate a tangent-stiffness matrix, one only needs to follow the definition of the tangent-stiffness matrix and issue the correct commands to the AD system. The formula is the same as the continuum formula

$$K_{ij} = \frac{\partial F_i^{int}(\vec{u})}{\partial u_j}, \tag{1.7}$$

but, with the caveat that F^{int} is evaluated in a way that is algorithmically consistent, and therefore should yield quadratic convergence with Newton's method.

1.3 Applying Newton's method

A popular iterative nonlinear solution technique is the famous Newton's method for n unknowns, here reproduced from [2, Chap. 6],

$$r_a(\mathbf{d}_\nu) + \frac{\partial r_a(d_\nu)}{\partial \mathbf{d}_b}\Delta d_b + O(\Delta \mathbf{d})^2 = \mathbf{0}. \tag{1.8}$$

The residual or function with a fixed point to be identified is linearized via a Taylor series expansion about an initial guess.

$$A = \frac{\partial \mathbf{r}}{\partial \mathbf{d}} \tag{1.9}$$

The system Jacobian or tangent-stiffness matrix is the slope of the tangent plane to the linearization of the residual.

$$\mathbf{r} + \mathbf{A}\Delta\mathbf{d} = \mathbf{0} \tag{1.10}$$

Substituting the residual derivative for the system Jacobian allows the minimization problem to be neatly written as a linear model. Higher order terms are dropped. The linear model may be solved for update $\Delta\mathbf{d}$ using an appropriate linear solver.

$$\mathbf{A}\Delta\mathbf{d} = -\mathbf{r}(\mathbf{d}_\nu, t^{n+1}) \tag{1.11}$$

The update is added to the current $\mathbf{d}_\nu$ to form the next iterate.

$$\mathbf{d}_{\nu+1} = \mathbf{d}_\nu + \Delta\mathbf{d} \tag{1.12}$$

While the essential method is usually as presented, the method is often augmented with additional steps such as to account for plastic changes in the material or to conserve on Jacobian evaluations by line searching for better residual reducing updates in the direction of the tangent plane. For these considerations [2] offers thorough discussion.

1.4 Working examples of CS, AD and FD for Newton's method

In order to illustrate the complex-step technique in a simpler form than it appears in the *Peridigm* code written for this study, an example header-only library and two example problem definition programs that use it have been created. All the materials described here are available in the associated project repository.

1.4.1 Common code

The source code for the header-only library is entitled *NewtonRaphson.hpp*. Note that the library is an interface to functions and data structures made available in the *Trilinos* library, mentioned earlier, such that it is required in order for the examples described here to work. *Trilinos* is freely available for download as of this writing at: *trilinos.sandia.gov*. Generally the header-only library

works by taking in the users initial guessed equilibrium solution, target dependent variable values, system constants, and a function pointer to a model evaluation function. Solver parameters including an update scaling factor, tolerance, maximum iterations and probe length are chosen after the solver is instantiated as a new object. The behavior of the solver is specialized through a coding feature called inheritance to perform each of CS, AD and FD. But, in common, each version of the solver works by computing the system Jacobian, solving a linear system as in that step of the Newton Raphson method, computing the residual and checking for convergence or overrun. Here the linear solution method is LU decomposition, while the chief convergence criterion is change in residual between iterations. Additionally the problem can be partially respecified to allow for running a series of load steps having progressing target values, which is a typical mode also seen in *Peridigm*. Applying boundary conditions incrementally in load steps assists convergence [24].

1.4.2 First example

The first example program, *NRexamples.cpp*, solves a simple nonlinear system describing the intersection of three infinite paraboloids in threespace. It works by defining force computation functions for each of the models, and then passing an address called a function pointer corresponding to each of these methods to a solver object which coordinates solving the problem and reporting output for the user based on their convergence criteria and other preferences like step-size and maximum number of iterations. CS, AD and FD are each represented and the user is free to adjust input settings by specifying command line arguments as the compiled program instructs, or perhaps create a shell script that varies parameters for each of the methods in order to perform a comparative study.

1.4.3 Second example and related studies

The second program,a *NRComparison.cpp*, follows a case study that appears in [24]. The example program departs from the cited literature by applying each of CS, AD and FD and verifying them against an analytical model of a nonlinear mechanical system involving a connected beam and

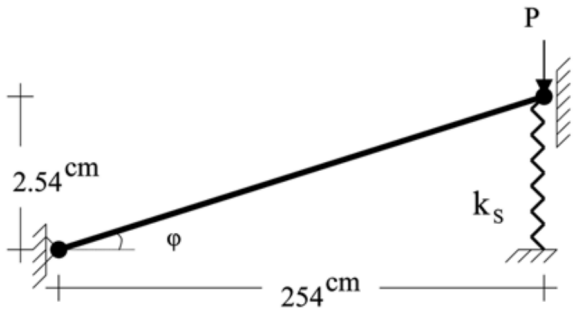

Figure 1.1: Beam and spring simulated in example program.

spring. Fig. 1.1 is a schematic of the beam and spring system.

The equations describing this system and results for verification were obtained in [24]. They include a force function, Eqn. 1.13, and an analytical force derivative or stiffness, Eqn. 1.14.

$$f(D) = .5AE(cos^2\phi)(\frac{D}{L_0})^2[\frac{D}{L_0}cos^2\phi - 3sin\phi] + k_sD + (AE\frac{D}{L_0})sin^2\phi, \tag{1.13}$$

$$S_T(D) = 1.5AE(cos^2\phi)[\frac{D}{L_0}cos^2\phi - 2sin\phi](\frac{D}{L_0^2}) + k_s + \frac{AEsin^2\phi}{L_0}. \tag{1.14}$$

Where ϕ is a constant, beam angle, AE is rigidity, D is displacement, L_0 is original beam length, and k_s is the Hooke's constant for the coil spring attached to the beam. For a verification of each of the methods as well as a convergence rate study, this truss and spring model was analyzed in the second program. $f(D)$ is restorative force to balance downward force loading at the connection point of the beam and the spring, parallel to the wall. As in the reference, equilibrium solutions for 12 incremental load steps of 4.448 Newtons were sought. Differing from the reference, Newton's method was used rather than 'Dynamic Relaxation'. Solver parameters common to all tests were a limit of 100 iterations for convergence, a residual change tolerance of 1×10^{-9}pm and an update multiplier of 1.0. Data collected included number of iterations to converge summed over all load-steps and final displacement, D, after the final load-step. Additionally finite-difference step-size was varied in order to measure the impact on each of the Jacobian calculation methods. Naturally, varying step size does not affect AD or the analytically derived Jacobian. The accuracy results for

verifying the Jacobian calculation methods are summarized in Table 1.2.

Table 1.1: Displacement, D, after load-step 12

Step exponent	Actual cm	AD cm	CS cm	FD cm
0	5.07996	"	"	"
-4	5.07996	"	"	"
-8	5.07996	"	"	"
-12	5.07996	"	"	"
-16	5.07996	"	"	failed
-20	5.07996	"	"	failed

Table 1.1 shows that each of the methods solved the problem as accurately as one another within the precision displayed on the table. The exception to this is that the finite difference method failed due to an arithmetic error in association with extremely small step size, suggesting subtractive cancellation. However, since there is no accuracy reason for choosing an extremely small step size, each of the methods can be used to produce acceptable solutions to the nonlinear truss system. Table 1.2 shows the total number of iterations taken over all load-steps for each of the methods.

Table 1.2: Summed iterations over all load steps

Step exponent	Actual cm	AD cm	CS cm	FD cm
0	63	"	213	417
-4	63	"	"	"
-8	63	"	"	"
-12	63	"	"	65
-16	63	"	"	fail
-20	63	"	"	fail

Table 1.2 indicates that when using AD or the 'actual' analytically derived Jacobian, a minimum number of iterations is achieved corresponding to a maximum convergence rate. CS and FD are similarly affected by a grossly large step-size, while subtractive cancellation invalidates the results for FD for very small step-sizes. Inaccuracy for large step-sizes is predicted by the FD and CS formulas, which are contingent upon a small step-size to allow the neglect of higher order

terms. The results of this test show that step-size selection is not unimportant when using FD, but when properly configured any of the methods may return the same maximum convergence rate for this problem.

Overall conclusions regarding the methods that may be kept in mind for problems besides this example are that the accuracy of the Jacobian should not be confused with the accuracy of the equilibrium solution, since each of the methods discussed here may be equally capable given enough iterations. Additionally, while the accuracy of the Jacobian may strongly affect convergence rates, this relationship was not quantified here.

Chapter 2: DESCRIPTION OF ANALYSIS TOOLS AND APPROACH.

2.1 Peridigm and peridynamics.

Comparisons of the different tangent-stiffness calculation techniques were carried out through code-development and *in situ* instrumentation of the computational peridynamics code *Peridigm* [20]. *Peridigm* is distributed by Sandia National Laboratories as its primary open-source computational peridynamics code. It is a massively-parallel simulation code for implicit and explicit multi-physics simulations primarily used for solid mechanics and material failure. *Peridigm* is a C++ code utilizing agile software components from Sandia's *Trilinos* project [10]. It's important to note that there is no specialization of the tangent-stiffness calculation methods described previously to this particular code or more generally to a computational peridynamics approach and the analysis carried out in this study should provide insight into the speed and accuracy of these methods when implemented into other computational mechanics analysis tools as well. *Peridigm* was chosen primarily because the FD, CD, and AD methods for tangent-stiffness calculation where already implemented in the code, leaving only the CS method for development. Also, because peridynamics is a nonlocal theory, the tangent-stiffness matrices have a much higher bandwidth (i.e. less sparsity) than traditional finite-element tangent-stiffness computations and the time required to construct the tangent-stiffness is a majority of the total computation time in any quasi-static or implicit dynamics analysis; therefore, the *Peridigm* development team has an interest in profiling the performance of the tangent-stiffness computation methods.

Briefly, peridynamics [26–28] is a nonlocal reformulation of the partial-differential equations that provide the statement of momentum balance in classical continuum mechanics. Its primary goal is to avoid the use of spatial derivatives in the balance equations or constitutive laws such that discontinuous displacements, i.e. cracks, are mathematically consistent with the governing equations. It has demonstrated great promise in modeling problems with massive pervasive failure [15], interesting phenomenon such as dynamic crack branching [9], and is being extended to multiphyics

phenomena [3, 14] and the problem of multi-scale coupling to molecular simulations [25].

2.1.1 Implementing the complex-step method in Peridigm

The CS method was implemented in *Perdigm* in a manner that follows closely the implementation of the FD technique. However, because of *Peridigm*'s reliance on *Trilinos* and specifically the *Epetra* package, special steps had to be followed to allow the use of complex number data types. This is because *Epetra* vectors, which can be thought of as distributed memory parallel cousins to standard C++ STL vectors, are hard coded to be of type `double` and can not be simply be declared as having a complex data type. Therefore, program methods that were used in the course of applying the FD method were copied, renamed and re-written to follow the CS formula. This involved changing the data type of intermediate variables which were locally scoped, and dynamically casting persistent memory variables to complex data types where necessary. No attempt was made to "tune" the CS code in any way for performance. The algorithm was implemented in such as way that it was in one-to-one correspondence with the FD and CD methods; however, it must be noted that the overhead associated with the dynamic casts in the CS method likely hurts its overall performance and this issue could be alleviated by using the next generation of templated *TPetra* vectors available in *Trilinos* that are capable of taking an explicit instantiation of any data type including complex. The version of *Peridigm* modified to implement CS can be found on the corresponding author's website.

Peridigm was primarily written by computational scientists; therefore, many advanced techniques in C++ such as multiple virtual inheritance, pointer arithmetic, and distributed data structures were used in the basic code and by necessity in the modified code. To see CS and other methods in a more narrow context, it is advised that readers also take a look at the simple examples provided on the corresponding author's website.

2.2 The Comparative Study

The methods were compared by running two sets of test problems where each method would solve a problem concurrently. In this context, concurrently means successively, drawing from the same independent variables, yet within the same computational process; this definition is characterized in further detail in 2.2.1. Each test problem simulated a $4\,\mathrm{m}$ by $0.5\,\mathrm{m}$ by $0.5\,\mathrm{m}$ meter block of a material undergoing tension along the axis parallel to the long dimension of the block. The test problems were set up in *Perdigm* as quasistatic equilibrium problems, where the displacement boundary conditions used to apply tension were applied gradually in 'load steps', and an equilibrium solution for each load step was achieved before applying the next load step. The purpose of applying the load in gradual steps is a widely used technique to ensure the Newton's method is always initialized near the actual solution and therefore likely to converge. The fictional material represented in the model had values of $1.515 \times 10^4\,\mathrm{MPa}$ for bulk modulus, and $7.813 \times 10^4\,\mathrm{MPa}$ for shear modulus. The peridynamic horizon, a length scale that effectively sets the bandwidth of tangent-stiffness in the computation, used in the test problems was always set to three times the nominal discretization size (i.e. node spacing in the particle discretization scheme). This results in a minimum tangent-stiffness matrix bandwidth of 7 (for an optimal node numbering scheme).

A series of nine single compute core (i.e. serial) test problems were run with the discretization size being refined with every test in the series. The aim of increasing the refinement was to exam differences between the methods in terms of accuracy and speed. This series would comprise the single core runs. The specific parameters of these tests can be found in *Peridigm* input `xml` files included in the archive for this thesis which can be found in the corresponding author's website. These `xml` files allow users with appropriately modified versions of *Peridigm* to reproduce the results found in this thesis.

Another series of four test problems were simulated, however this time the number of compute cores used to solve the problem was increased from test to test while the discretization level was held constant at 1 million computational nodes (3 million degrees of freedom). This allowed

for sufficient computational nodes per core even at the highest level of parallelization such that message passing computation did not overwhelm the simulation. The specific parameters of these tests can be found in *Perdigm* input `xml` files also included in the archive for this thesis on the corresponding author's website.

A final test problem was run using the same input mesh as the largest single compute core test problem. This test consisted of running each of the studied methods separately to conclusion such that convergence rate and solution accuracy could be measured. The trend-less nature of the Jacobian accuracy results indicated this final test with a single discretization size could be considered representative of the single core study. That is to say finite-difference and complex step had the same Jacobian accuracy ranking in every single-iteration test, so to be judicious the single largest problem was chosen for the convergence and solution accuracy study.

2.2.1 Quantities of interest

As stated in the Introduction, a goal of the study was to compare CS, FD, and CD on the basis of accuracy. AD was omitted from the comparison because it served as the standard of accuracy for the other methods in the absence of appropriate analytical forms for the tangent-stiffness matrix associated with the system solved in the study. The assumption that AD is accurate enough to serve as a standard is supported by ADs implementation as a computerized chain rule as explained in Section 1.1.3.

It would be a poor comparative study to compare tangent-stiffness matrices from different problems, load steps that start with different current configurations, or from different iterations; because of this, it was necessary to solve one load step and conclude each Newton iteration within that load step by updating the displacement iterate with only the results of the AD method and to subsequently feed all four methods the same updated displacement in the following iteration. This decision precluded a comparison of Newton iteration convergence rate, since if the methods were allowed to solve a problem at their individual pace, differences in accuracy would produce differences in guess updates and therefore the number and nature of Newton iterations performed

(e.g. lower accuracy predictions of the algorithmically correct tangent-stiffness could cause a loss of quadratic convergence). Different guesses from iterations started with different previous guesses could not be data for a valid comparison of tangent-stiffness accuracy between methods. Additionally, having each of the methods physically operate in the same process, serial or one of associated parallel, allowed the comparison of tangent-stiffness matrix calculation time as cached from random access memory rather than from the hard-disk. Caching from RAM gives the benefit of vastly greater speed and simpler programming compared to some other solution speculatively involving dynamic file management on files which for one of the components of the study would be on the order of a terabyte in size. However, the price of running the methods together in the same process and avoiding the use of the hard-disk was that at least two tangent-stiffness matrices had to be stored in RAM during the simulation, which meant that special high-memory compute nodes were needed for the 1 million peridynamic node test series.

Another goal was to compare the four methods, including AD, on the basis of speed. Speed is defined as the total compute-time of one iteration of a Newton step which includes the tangent-stiffness calculation and parallel assembly. The speed of iteration was measured because it could be so done at the same time as accuracy was being measured given a single tangent-stiffness calculation. It was assumed that the order that each method was evaluated in parallel was unimportant, that evaluating each method successively within each solver iteration did not affect their performance individually and that speed of computation did not change with time. These assumptions allowed the test program to run the same problem with each of the methods at effectively the same time and generate an equal volume of data from each method. It was also assumed that the tangent-stiffness matrix calculation time for each of the methods did not vary based on the current configuration, as it changes slightly from iteration to iteration as the Newton solution iterate is updated. This assumption allowed calculation time measurements to be averaged over all iterations within a single load step. The purpose of averaging calculation time measurements was to informally address the extraneous variables of parallel evaluation order and computer system load due to system processes not associated with the study. While not part of the study, the example

programs available on the corresponding author's website allow the reader to make a comparison of the methods for themselves on the basis of accuracy and convergence rate for two example nonlinear systems. In these examples, one will notice that the finite difference methods will lose quadratic convergence when the step size selected is too large such that accuracy relative to AD is decreased.

A final goal was to place the per-iteration performance results in context with a convergence rate study. Each of the methods would be allowed to solve the same problem separately and their average number of iterations to converge per load step would be computed. This measurement would give context to the Jacobian accuracy results and reveal if they were practically significant. Additionally, solution accuracy would be measured as the order of magnitude of the greatest individual element difference between AD and another method.

The goals of collecting accuracy and speed measurements were achieved by developing and implementing metrics within the simulation program used in the study. The metric used to measure the accuracy of a tangent-stiffness matrix was the Frobenius norm (analogous to an l^2-norm except defined on a matrix) of the element-wise difference between the tangent-stiffness matrix produced by the method being evaluated and the tangent-stiffness matrix produced by the AD based method (i.e., the *exact* derivatives), given that both methods were set upon the same problem. The lower the value of this metric, the more accurate the method was. The expression for the accuracy metric was

$$D = \sqrt{\sum_i \sum_j (K_{ij}^{AD} - K_{ij}^{M})^2} \tag{2.1}$$

Where D is distance, K^{AD} is the tangent-stiffness matrix produced by the AD based method, M is replaced with either CS, FD, or CD depending on the method being compared. The metric used to compare the speed of the different methods was the time in seconds required to calculate the tangent-stiffness matrix. A final basis of comparison used in the study called *computational efficiency* is based on specific calculation time per tangent-stiffness matrix element.

2.2.2 Computers and other software

Discretization for the test problems run in the study was done using the software package *Cubit* developed at Sandia National Laboratories [31]. *Cubit* generates a finite element mesh that is internally converted to a "particle" discretization internally when calling *Peridigm*. The number of computational nodes in *Peridigm* correspond to the number of finite elements in the discretization, not the number of finite element nodes. An example journal script can be found on the corresponding author's website.

The test problems were run on the *Stampede* HPC cluster computer housed at *TACC* (Texas Advanced Computing Center) at The University of Texas at Austin. The single core test series was run using the `normal` queue compute nodes, while the parallel test series was run using one to four `large memory` nodes each having one terabyte of local RAM. This large amount of memory was required for reasons explained in Section 2.2.1.

2.2.3 Data Reduction

The output produced by *Perdigm* including the additional accuracy and speed measurements was redirected from the console to text files by the resource manager, *SLURM* (Simple Linux User Resource Manager). Directories and filenames were chosen so that data would be easy to sort by individual test run. Extraneous information was manually deleted from copied versions of the output text files so that they would be regular enough to be easily read by scripts written in *Python*. These scripts averaged accuracy or speed data for a tests corresponding to a particular peridynamic node density over all iterations for that run, and then plotted this averaged data as a function of peridynamic node density as it varied from test to test in the series. Similarly, for the parallel tests, the data was averaged and plotted in much the same way, but as a function of number of compute cores used to solve the problem rather than as a function peridynamic node density which was held constant. Confidence intervals were not calculated on the measurements. The main reason is that network traffic from other users makes running tests on the HPC cluster a time varying process.

These data reduction and plotting scripts can be found on the corresponding author's website.

Chapter 3: RESULTS AND DISCUSSION

Table 3.1 shows the compute core count, number of nonzero tangent-stiffness (TS) non-zero matrix elements, load step average calculation time and load step average accuracy. The units for accuracy are derived from the expression used to calculate the TS matrix elements and the units used to define bulk material properties, mentioned in Section 2.2. The results of the *Peridigm* convergence studies are reported in 3.4.

Table 3.1: Averaged Results, Each Test

Cores	TS Elements	Calculation Time (s)				Accuracy Difference (MPa)		
	$\times 10^6$	CS	CD	FD	AD	CS	CD	FD
1	1.99	3.5	2.7	1.7	1.6	1.92×10^{-10}	1.21×10^{-4}	.137
1	4.25	6.2	4.9	3.2	2.9	2.28×10^{-10}	9.94×10^{-4}	.148
1	7.79	11.2	8.9	5.7	5.2	2.38×10^{-10}	1.59×10^{-4}	.145
1	15.4	26.7	21.0	13.4	12.2	2.33×10^{-10}	4.61×10^{-4}	.12
1	20.0	28.1	22.2	14.4	13.1	2.76×10^{-10}	1.05×10^{-3}	.145
1	31.4	47.6	37.7	24.2	21.9	2.64×10^{-10}	1.65×10^{-3}	.133
1	40.1	55.6	44.1	28.4	25.9	3.03×10^{-10}	1.92×10^{-3}	.148
1	83.9	138.9	109.6	70.2	64.0	3.63×10^{-10}	1.64×10^{-3}	.123
1	165.0	277.3	218.1	139.4	126.5	3.26×10^{-10}	2.18×10^{-3}	.128
32	1670	336.1	277.7	200.6	233.1	6.21×10^{-10}	1.52×10^{-2}	.176
64	1670	169.9	140.7	102.0	119.7	6.20×10^{-10}	1.50×10^{-2}	.177
96	1670	114.7	95.0	69.1	79.7	6.18×10^{-10}	1.50×10^{-2}	.177
128	1670	86.4	71.8	52.4	58.8	6.16×10^{-10}	1.47×10^{-2}	.177

3.1 Speed Data

Speed measurements were taken according to the methods mentioned in Section 2.2. The averaged results of these speed measurements for the serial tests can be seen in Fig. 3.1. Average time in seconds to compute a TS matrix is plotted as a function of the number of nonzero TS elements. CS was the slowest of the four methods, taking the longest average time per iteration, followed by CD, FD and AD in that order. Calculation time has a power-law relationship with problem size, with a power-law index of roughly 1.0×10^{-6} for each of the methods. This bodes well for the

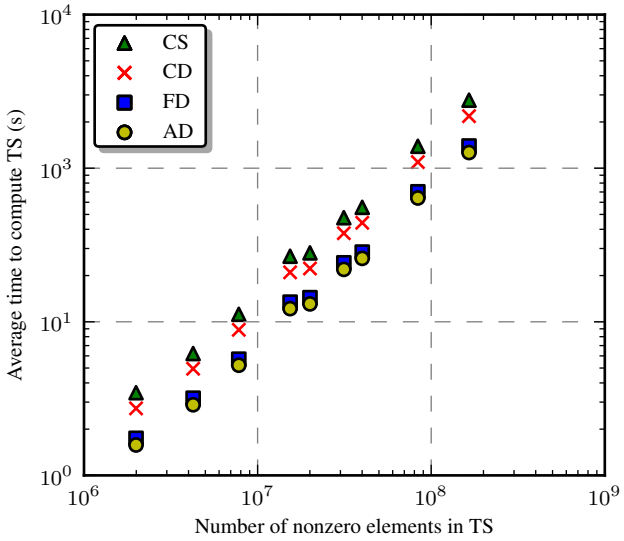

Figure 3.1: Serial test series speed measurements.

scalability of each of the methods by themselves in *Peridigm*.

For the parallel simulations, averaged speed results appear in Fig. 3.2. For this series, rather than varying the number of nonzero TS matrix elements the number of compute cores used to solve the problem is increased, while the number of nonzero TS matrix elements remained constant at 1.67×10^9 . CS again was the slowest method, followed by central-difference, automatic-differentiation and then forward-difference. There is a somewhat surprise reversal of FD and AD for these parallel simulations. Figure 3.2 indicates that when looking at average iteration time, the factor of speed improvement scales linearly with the number of compute cores used for the given conditions of this group of tests. Looking at Figure 3.2, the reader can see that when core count doubles, compute time halves, for each method. Again, these results show that each of the methods as implemented in *Peridigm* scale well when additional processors are used. A speculative explanation as to why FD and AD have reversed places in the speed tests with respect to the serial results is that the AD data-structures carry a list of partial derivatives of the functional evaluations along side the functional evaluations themselves only to be processed at the end of the tangent-stiffness calculation when the derivatives are requested. This represents additional load on the hardware memory subsystem, such that may exceed that of FD yet still fall under a supposed

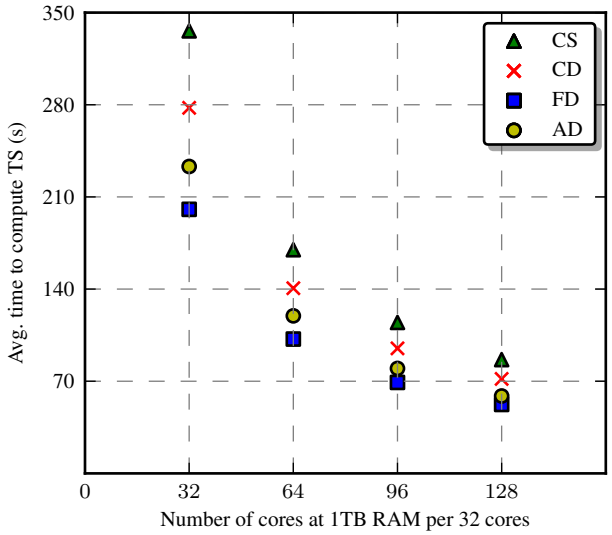

Figure 3.2: Multicore test series speed measurements.

memory bandwidth threshold during serial operation. Cores within a multicore processor use the same shared bus to access memory, such that cores compete for memory access. Supposing that exceeding the memory bandwidth threshold penalizes additional requested bandwidth, it is likely that the decrease in available memory bandwidth per core caused by parallel operation coupled with AD's additional resource requirements causes the AD method to be slower than FD.

3.2 Accuracy Data

Data for accuracy measurements were taken according to the methods mentioned in Section 2.2, and reduced using 2.1. The averaged results of these accuracy measurements for the serial tests can be seen in Fig. 3.3. Averaged Frobenius norm (referred to as l^2-norm in the figure labels for conciseness) of the element-wise difference between a given TS matrix and the TS matrix produced by the AD method is plotted as a function of the number of nonzero TS matrix elements. CS was shown to be the most accurate of the methods when compared to AD and was 6 orders of magnitude more accurate than CD and 9 orders of magnitude more accurate than FD. This accuracy ranking order was maintained for each test in the series. It should be pointed out that compared to the l^2-norms of any of the TS matrices by themselves, the magnitude of the accuracy

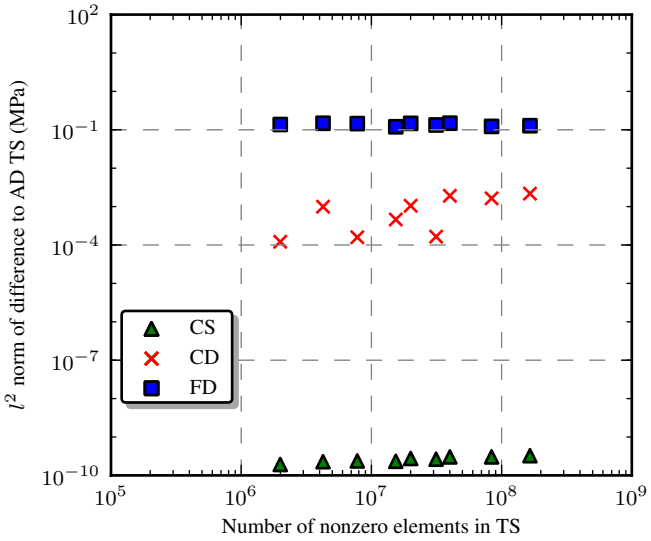

Figure 3.3: Serial test series accuracy measurements.

metric calculated with Eqn. 2.1 for any of the methods is relatively small. These values appear in the simulation data which is available on the corresponding author's website.

Perhaps accuracy of FD and CD could be improved by using a smaller probe distance, h; however, if h is too small, then the methods can suffer from severe effects, including solution divergence, of the round-off error associated with subtracting two numbers that are very close to one another. The value of h used in this study is the default value used in *Peridigm* which is heuristically derived based on computational node spacing. One of the beauties of the CS method is that since dependence on h decreases for small h, which a numerical example in [30, Table 1] demonstrates, h, cannot be too small, therefore it can be set to machine epsilon and the analysis performed without the issues associated with round-off error. The value chosen for h for CS used in this study was 1.0×10^{-100} to demonstrate this. However, as is seen 3.4 the step-size used was near enough optimal to return optimal convergence for all of the methods.

The averaged results of the accuracy measurements taken for the parallel test series can be seen in Fig. 3.4. The ordinate axis units remain unchanged from Fig. 3.3, but the absicca now indicates number of compute cores used to solve the test problem. Accuracy trends matched those for the single core tests.

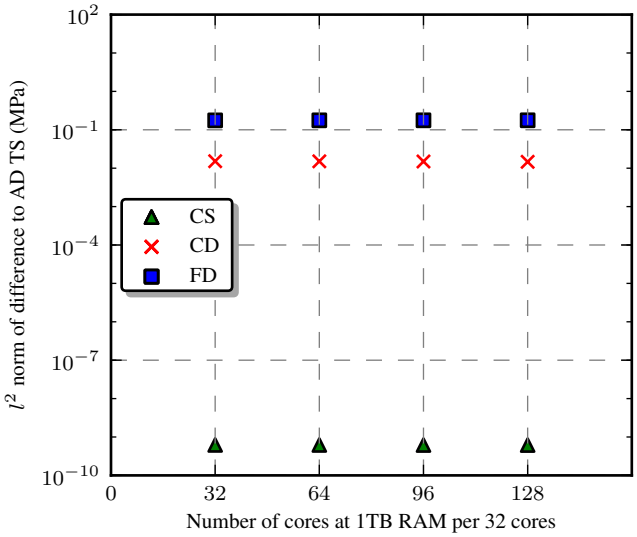

Figure 3.4: Multicore test series accuracy measurements.

3.3 Efficiency Data

Efficiency, calculated as the average number of seconds taken per nonzero Jacobian matrix element, was plotted as a function of the number nonzero TS matrix elements. The single core results are shown in Fig. 3.5. The ranking order of the methods matched their speed ranking in Fig. 3.1. This plot indicates that efficiency is insensitive to the number of nonzero TS elements. The parallel efficiency results are shown in Fig. 3.6. The parallel efficiency plot shows that AD was slower than FD because for the given test problem it had a lower efficiency. Strangely AD appears to gain in efficiency over the last few data points in the plot. The reason behind this trend was not determined.

3.4 Convergence and displacement accuracy studies

While the previous sections profiled the per-iteration performance of each of the methods, it was necessary to conduct convergence rate studies in order to get a complete picture. It is necessary for an analyist to know if the accuracy differences measured in the per-iteration tests would amount to differences in convergence rates among the methods if they were allowed to solve problems

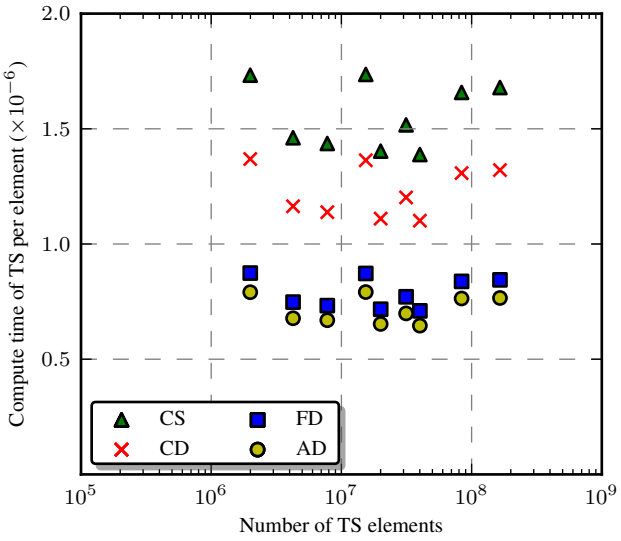

Figure 3.5: Serial test series efficiency measurements.

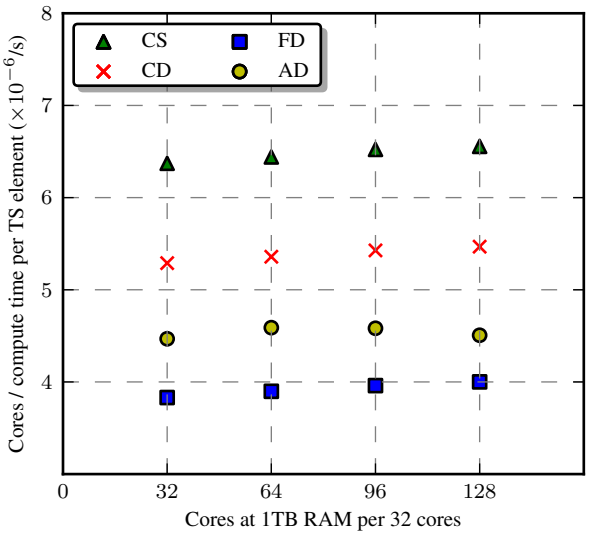

Figure 3.6: Multicore test series efficiency measurements.

normally. The question is posed, given that step-size h is heuristically selected by *Perdigm* for the finite difference methods and presumable makes a good choice, do the single-iteration Jacobian accuracy results predict the ranking of the methods in terms of convergence rate for a given study? For brevity, convergence rate was measured for each of the methods in solving just the largest problem of the single-core test series. As was seen in the per-iteration results, Jacobian accuracy did not vary greatly from test to test, such that any pair of tests from the two series could have been considered reasonably representative. As in the validation problem shown in 1.4.3, would each of the methods return an equal convergence rate so long as h were optimally selected?

The results shown in Table 3.2 indicate that this was the case. The difference in Jacobian accuracy between the methods seen in the per-iteration tests was not sufficient to lead to differences in the convergence rate of the methods.

Table 3.2: Average iterations/ load-step

Test type	AD	CS	CD	FD
single-core	63	"	"	"
multi-core	63	"	"	"

The end displacement results shown in 1.4.3 indicated that up to a certain precision the solutions produced by each of the methods were identical. It was examined if this was the case for the solutions produced in *Peridigm*. Here each of the solutions produced by each of the methods were compared to the AD produced solutions, under the assumption that the AD produced solutions would be closest to the solution produced by using an 'exact' Jacobian. Table 3.3 shows that the methods did indeed produce differing solutions in terms of nodal force density, although only in decimal places at least 9 orders of magnitude smaller than the force density value at the node where the difference was detected.

Table 3.3: Nodal force density maximum difference

	CS	CD	FD
Order of magnitude	1.0×10^{-8}	1.0×10^{-8}	1.0×10^{-5}

The solution accuracy results do follow the same ranking seen in the Jacobian accuracy results, however, again as in the validation problem the results differ only slightly. Speculatively this degree of error may be important to those running probabilistic simulations such that the propagation of error may be a concern. Also it is conceivable that if this error is a fixed quantity, insensitive to material properties or loading conditions, it may have a greater impact on simulations involving very low force densities such as for small displacements or very lax materials. In this case it could be recommended that forward-difference be avoided.

Chapter 4: CONCLUSIONS

4.1 For further studies of Jacobian accuracy in Newton's method

The question is raised; what is the significance of Jacobian estimation accuracy to the convergence of Newton's method? The per-iteration and even convergence studies shown here indicate the effect of Jacobian accuracy for only a single datum, a very accurate Jacobian estimate, while the validation problem indicates the effect of using a grossly inaccurate Jacobian. The results here show that for very small Jacobian estimate inaccuracies, convergence rate is not affected, while for grossly inaccurate Jacobians Newton's method fails. Is it worthwhile then to investigate the effect of using moderately inaccurate Jacobians, produced deliberately or otherwise, on the convergence rate of Newton's method?

The answer may be no at least unless a more highly non-linear class of problem is selected for study or a controllable method of producing a moderately inaccurate Jacobian is found along with a reason for using it. While it is possible to incorrectly apply finite-differencing and CS by choosing an inappropriate step-size, there is no reason that this should be done. The experiments show that step-size h can be properly selected and that the methods are capable of producing accurate Jacobians. It should be noted that the model used to perform the simulation could always be substituted for a more highly non-linear one and perhaps this would somewhat increase the importance of Jacobian accuracy. The study presented here failed to pose the question of how wrong a correctly implemented Jacobian calculation scheme could go for plastic or otherwise non-linear material models, here instead non-linearity was imposed by the nonlocal interaction between material points essential to peridynamic models, like the linear model used in the study.

Nevertheless, Newton's method is not defined for using an inaccurate or 'wrong' Jacobian, so using an inaccurate Jacobian is failing to perform Newton's method by definition. But it is also true that it should not be expected that every differentiation technique performs flawlessly under all situations. The pragmatic strategy to study this may be to compare a prospective method's Jacobian

accuracy to that produced by AD and develop a rule of thumb based on observed convergence rates from prior simulations as they relate to the percent difference of the prospective and AD produced Jacobians. Also it is possible to calculate a sensitivity of Jacobian error to step-size alongside a sensitivity of convergence rate to step-size, and then to relate these two quantities as a map of Jacobian error to convergence rate change from ideal. Additionally it is still possible to look at the convergence conditions for Newton's method and suggest a relationship between Jacobian accuracy and convergence rate in a theoretical manner. To this point, there are a set of convergence conditions derived in [4, Chap. 2, p.41] which relate quality of Jacobian inverse approximations to convergence rate classification, here reproduced,

$$|| \ (r\prime(x_k)^{-1} - J_k^{-1})y_k \ ||\leq K, \tag{4.1}$$

for worse than linear convergence,

$$|| \ (r\prime(x_k)^{-1} - J_k^{-1})y_k \ ||\leq \Theta_1^k K; \Theta_1 \in (0,1), \tag{4.2}$$

for linear convergence,

$$|| \ (r\prime(x_k)^{-1} - J_k^{-1})y_k \ ||\leq minM_2 \ || \ x_k - x_{k-1} \ ||, K; M_2 > 0 \tag{4.3}$$

for two step quadratic,

$$|| \ (r\prime(x_k)^{-1} - J_k^{-1})y_k \ ||\leq minM_3 \ || \ r(x_k) \ ||, K; M_3 > 0 \tag{4.4}$$

and lastly for quadratic convergence. Here $r\prime$ is the true system Jacobian and J is the approximation. Essentially the conditions measure how well the difference of the true Jacobian inverse and approximation works as a contraction map within the closure of a radius of convergence about the true solution. This property is necessary for the solution method using the approximation to converge rather than to diverge, as is true of the exact Newton method.

It may be interesting to artificially perturb the Jacobian calculations with random error in order to better control experimentally the onset of inaccuracy to develop a sensitivity of convergence rate to this type of error. Perhaps an analyst could use this knowledge to devise an adaptive differentiation technique selection method which economizes on accuracy until a significant Jacobian error is measured during a special doubly calculated iteration.

Additionally, there are Quasi-Newton methods such as Broyden's method which uses an approximate inverse Jacobian and updates it with information from the iterate and residual [6]. A method called Incomplete Jacobian Newton's method may the closest to the idea of deliberately making use of an inaccurate Jacobian in the sense that arbitrary elements of the full Jacobian are set to zero to reduce computational effort [16]. The Quasi-Newton methods represent a compromise of Newton's method for the promise of faster iterations in exchange for a lower convergence rate. For both of these methods, the iterate and residual are computed as in Newton's method. As to the question of algorithmic consistency, these numerical methods evaluate the same linearized constitutive law and governing equation as for the exact Newton's method which in the case of a plastic simulation are involved in iteratively determining algorithmic tangent moduli. In fact, a secant equation based method is appropriate for Newton's method with algorithmic tangent moduli since the Jacobian ought only depend on known converged values such that the Newton iterate could not artificially drive the plasticity of the model [2, Chap. 6]. It should be noted that constraints or a line search algorithm to minimize the residual are necessary whereas the pure secant method does not return unique solutions in multi-dimensions.

4.2 Grain of salt for speed results

The per-iteration speed results shown here indicate that the AD method is faster than CD or FD. While this may be the case for the implementation in the stock version of *Peridigm* this is not reflective of any limitation of these methods. In a side experiment, the finite-difference code in *Peridigm* was re-written to take advantage of the benefits of storing the results of duplicated intermediate calculations. These modifications were general, low memory footprint and could be

adopted in any numerical simulation simulation software as good practice. An informal performance test was done using a personal AMD Phenom-2 X4 workstation using GCC 4.7 compilers rather than a Intel Xeon E5 node with Intel Cluster Studio 13.0 compilers as provided by TACC. The problem being run was the largest single core test from the main study, instead run on four compute cores. Table 4.1 shows the timing results for AD as reference along with an unmodified CD, and then modified CD and FD.

Table 4.1: Total time spent computing Jacobian

	AD	CD	Modded CD	Modded FD
Total seconds	430	570	430	320

The results indicated that in exchange for a modest programming effort, a performance increase of around a quarter was available for the finite-difference methods and that FD could eclipse AD. What allowed the performance modifications to enhance the finite-difference methods was their lack of sophistication compared to AD in that their explicitly laid out and repetitive code could be readily optimized.

4.3 Remarks

Based on the experimental results, it appears that for users of *Peridigm* there is a case to be made for using AD for reasons of speed. However, certain parallel tests done in this study showed a small performance gain over AD with the FD method. The results showed that CS produced more accurate tangent-stiffness matrices than CD and FD under the parameters of the tests, however it was determined that this difference in accuracy was not sufficient to alter convergence rate for a test problem. In order for Jacobian accuracy to have an impact on convergence rate, and therefor serve as a predictor of convergence rate in per-iteration results, the amount of error detected between a supposed perfectly accurate Jacobian and the test Jacobian needs to be greater than allowed by the conditions seen in [4], reproduced in 4.1.

It should be mentioned that another work, [22], compares the speed and accuracy of complex-

step to finite difference for producing tangent-stiffness matrices in solid mechanics simulations using the 'MRS-Lade' material model, where the researchers similarly found CS to be accurate yet relatively computationally expensive compared to FD and CD. In that work, full convergence studies were also done and indicated that quadratic convergence was achieved with the methods [22, p.28].

The CS method was shown to be highly accurate in calculating tangent-stiffness matrices by the standards of this study, and it is slower than other methods for a naive implementation. In the context of future use of CS, CD and FD in *Peridigm*, the methods hold the advantage of relying on byte-copyable data types, while AD requires a complicated serialization and deserialization in order to function as byte-copyable. This is fine for copying in between MPI processes, but deserialization on an accelerator or GPU is very difficult, as this would require compiling a version of the AD library being used that were compatible with the accelerator or GPU being used. What this means is that if *Peridigm* or any simulation software were to be reformulated to take advantage of accelerators or GPUs, the flexibility of complex-step owed to its simple implementation may make it more viable for that application given its high accuracy, should it be required. Additional performance gains may be achieved through the use of the templated *Tpetra* package in *Trilinos* which would allow the use of complex data types natively, and alleviate the expense of the dynamic-casts that were utilized in this study. However, at least for the models producing the results shown here, each of the methods returned quadratic convergence, indicating each was sufficiently accurate. 'Marching orders' for the analyst would be to select the swiftest method, while it would seem accuracy above what is adequate does not enhance the convergence rate or iteration speed until either classical Newton's convergence conditions or the approximate case conditions are violated.

Appendix A: COMPLEX-STEP EXAMPLE

Using $f(x) = \cos(x)$, we have

$$f(x + ih) = \cos(x + ih),$$

or through trigonometric identities equivalently

$$f(x + ih) = \cos(x)\cosh(h) - i\sin(x)\sinh(h),$$

Applying equation (1.2)

$$\frac{\partial f(x)}{\partial x} = -\frac{\sin(x)\sinh(h)}{h},$$

Since h is arbitrary we desire it to be as small as possible or

$$\frac{\partial f(x)}{\partial x} = \lim_{h\to 0} -\frac{\sin(x)\sinh(h)}{h},$$

using L'Hôpital's rule

$$\frac{\partial f(x)}{\partial x} = \lim_{h\to 0} -\frac{\sin(x)\cosh(h)}{1},$$

of course, as $h \to 0$ then $\cosh(h) \to 1$ thereby recovering the exact derivative

$$\frac{\partial f(x)}{\partial x} = -\sin(x).$$

Appendix B: AD EXAMPLE

1. Take an example composition function $f(x) = (sin(cos(x)))^2$

2. Suppose we want to know $\frac{df}{dx} \mid_{x_0}$ where x_0 is a particular value of x.

   (a) Given that:

   i. $S \in R^1$

   ii. $X \in S$

   iii. $g, h, k \in H : S \to S$

   iv. $\forall s \in S : g(s) = s^2$, $h(s) = sin(s)$, $k(s) = cos(s)$

   v. $\forall x \in X : f(x) = g(h(k(x)))$

   (b) Given that the computer is programmed with some mathematical definitions:

   i. A. $u, v, w \in H : R^3 \to R^1$

   B. $s, a, b, x \in R^1$

   ii. particular values can be identified:

   $s = s_0, .., s_n, ...s_\infty \mid n = [0, \infty)$

   and similarly for the other variables a, b, x

iii. functions u, v, w and their partial derivatives w.r.t s are defined such that:

for a, b, s equal to a_n, b_n, s_n:

$$u \mid_{a_n,b_n,s_n} = a_n \cdot s_n^{b_n}$$
$$\frac{\partial u}{\partial s} \mid_{a_n,b_n,s_n} = a_n \cdot b_n \cdot s_n^{b_n-1}$$

$$v \mid_{a_n,b_n,s_n} = a_n \cdot sin(b_n \cdot s_n)$$
$$\frac{\partial v}{\partial s} \mid_{a_n,b_n,s_n} = a_n \cdot b_n \cdot cos(b_n \cdot s_n)$$

$$w \mid_{a_n,b_n,s_n} = a_n \cdot cos(b_n \cdot s_n)$$
$$\frac{\partial w}{\partial s} \mid_{a_n,b_n,s_n} = -a_n \cdot b_n \cdot sin(b_n \cdot s_n)$$

(c) Given that it is possible to describe $f(x) = (sin(cos(x)))^2$ in terms the computer understands by inputting $f(x)$ such that the computer stores an equivalent statement $f(x) = u(a, b, s) \mid_{arguments}$, iff the arguments of u, v, w are chosen such that u, v, w approximate g, h, k as follows:

Function	a	b	s	Approximates Function
u	1	2	v	g
v	1	1	w	h
w	1	1	x	k

3. It follows from 2(a)v that we can evaluate $\frac{d}{dx} f(x) \mid_{x_0}$ with the chain rule:

$$\frac{d}{dx} f(x) = \frac{d}{dx} \cdot g(h(k(x))) \mid_{x_0}$$
$$\frac{d}{dx} f(x) = \frac{dg}{dh} \cdot \frac{dh}{dk} \cdot \frac{dk}{dx} \mid_{x_0}$$

From the rest of 2a it follows that we can approximate $\frac{d}{dx} f(x) \mid_{x_0}$ by specializing the computer's general forms of u, v, w according to 2c, with parameters a, b chosen for each function and held as constant, and evaluating, such that the total derivative of our original function w.r.t x where $x = x_0$ is approximated by the partial derivative of our equivalent

statement, $f(x) = u(a, b, s) \mid_{arguments}$, w.r.t s where $s = x_0$.

For completeness we write out the computer's steps to evaluate 3 under the conditions of 2c with a particular value of $s = x_0$, in equation format:

$$\frac{\partial}{\partial x} f(x) \mid_{x_0} = \frac{\partial u}{\partial v} \mid_{1,2,v|_{1,1,w|_{1,1,x_0}}} \cdot \frac{\partial v}{\partial w} \mid_{1,1,w|_{1,1,x_0}} \cdot \frac{\partial w}{\partial s} \mid_{1,1,x_0} \cdot \frac{ds}{dx}$$

Because the computer knew the analytical forms of the partial derivatives of each of u, v, w beforehand, all it needed to do was:

1. to evaluate each of u, v, w according to 2c, in order from $w \rightarrow v \rightarrow u$,
2. to remember the values for x_0 and the output of each function evaluation besides u,
3. then to use x_0 and the output of the function evaluations as input for the corresponding partial derivative function evaluations, and
4. store the individual partials.

Lastly, to compute the partial derivative of the entire composition function, the computer multiplies the individual partials together in observance of the chain-rule.

BIBLIOGRAPHY

[1] Al-Mohy, A.H., Higham, N.J.: The complex step approximation to the fréchet derivative of a matrix function. Numerical Algorithms **53**(1), 133–148 (2010)

[2] Belytschko, T., Moran, B., Liu, W.K.: Nonlinear finite element analysis for continua and structures, vol. 1. Wiley (1999)

[3] Bobaru, F., Duangpanya, M.: A peridynamic formulation for transient heat conduction in bodies with evolving discontinuities. Journal of Computational Physics (2011)

[4] Burke, J.: Chapter 2. basic convergence theory. University Lecture (1998)

[5] Chapra, S., Canale, R.: Numerical Methods for Engineers, sixth edn. McGraw-Hill, Inc., NY (2010)

[6] Dennis, J.: On the convergence of broydens method for nonlinear systems of equations. Mathematics of Computation **25**(115), 559–567 (1971)

[7] Gay, D.: Semiautomatic differentiation for efficient gradient computations. Tech. Rep. SAND2004-4688P, Sandia National Laboratories: Optimization and Uncertainty Estimation Department (2004)

[8] Griewank, A.: On automatic differentiation. Tech. Rep. CRPC-TR89003, Rice University: Center for Research on Parallel Computation (1989)

[9] Ha, Y., Bobaru, F.: Studies of dynamic crack propagation and crack branching with peridynamics. International Journal of Fracture **162**(1–2), 229–224 (2010)

[10] Heroux, M., Bartlett, R., Howle, V., Hoekstra, R., Hu, J., Kolda, T., Lehoucq, R., Long, K., Pawlowski, R., Phipps, E., Salinger, A., Thornquist, H., Tuminaro, R., Willenbring, J., Williams, A., Stanley, K.: An overview of the trilinos project. ACM Trans. Math. Softw. **31**(3), 397–423 (2005). DOI http://doi.acm.org/10.1145/1089014.1089021

[11] Heroux, M., Willenbring, J.: Trilinos users guide. Tech. Rep. SAND2003-2952, Sandia National Laboratories (2003)

[12] Hughes, T.J., Pister, K.S.: Consistent linearization in mechanics of solids and structures. Computers & Structures **8**(3), 391–397 (1978)

[13] Hughes, T.J., Taylor, R.L.: Unconditionally stable algorithms for quasi-static elasto/visco-plastic finite element analysis. Computers & Structures **8**(2), 169–173 (1978)

[14] Katiyar, A., Foster, J., Sharma, M.: A peridynamic formulation of pressure driven convctive fluid transport in porous media. Journal of Computational Physics **Submitted** (2013)

[15] Littlewood, D.J.: Simulation of dynamic fracture using peridynamics, finite element modeling, and contact. ASME Conference Proceedings **2010**(44465), 209–217 (2010). DOI 10.1115/IMECE2010-40621

[16] Liu, H., Ni, Q.: Incomplete jacobian newton method for nonlinear equations. Computers & Mathematics with Applications **56**(1), 218–227 (2008)

[17] Lyness, J.: Differentiation formulas for analytic functions. Math. Comp **22**(102), 352–362 (1968)

[18] Lyness, J., Moler, C.: Numerical differentiation of analytic functions. SIAM Journal on Numerical Analysis **4**(2), 202–210 (1967)

[19] Martins, J., Kroo, I., Alonso, J.: An automated method for sensitivity analysis using complex variables. AIAA paper **689**, 2000 (2000)

[20] Parks, M., Littlewood, D., Mitchell, J., Silling, S.: Peridigm users' guide. SANDIA REPORT 2012-7800, Sandia National Laboratories (2012)

[21] Perez-Foguet, A., Rodriguez-Ferran, A., Huerta, A.: Numerical differentiation for local and global tangent operators in computational plasticity. Computer methods in applied mechanics and engineering **189**(1), 277–296 (2000)

[22] Perez-Foguet, A., Rodriguez-Ferran, A., Huerta, A., et al.: Numerical differentiation for non-trivial consistent tangent matrices: an application to the mrs-lade model (2012)

[23] Phipps, E., Gay, D., Bartlett, R.: Sacado: Automatic differentiation tools for c++ codes. Tech. Rep. SAND2009-7540C, Sandia National Laboratories: Computer Science Research Institute (2009). Presented at Trilinos User's Group Meeting 2009 in Albuquerue, New Mexico.

[24] Rezaiee Pajand, M., ALAMATIAN, J.: The dynamic relaxation method using new formulation for fictitious mass and damping. Structural Engineering and Mechanics **34** (2010)

[25] Seleson, P., Parks, M.L., Gunzburger, M., Lehoucq, R.B.: Peridynamics as an upscaling of molecular dynamics. Multiscale Modeling & Simulation **8**(1), 204–227 (2009)

[26] Silling, S.: Reformulation of elasticity theory for discontinuities and long-range forces. Journal of the Mechanics and Physics of Solids **48**(1), 175–209 (2000). DOI 10.1016/S0022-5096(99)00029-0

[27] Silling, S., Epton, M., Weckner, O., Xu, J., Askari, E.: Peridynamic states and constitutive modeling. J Elasticity **88**, 151–184 (2007). DOI 10.1007/s10659-007-9125-1

[28] Silling, S., Lehoucq, R.: Peridynamic theory of solid mechanics. Advances in Applied Mechanics **44**, 73–168 (2010). DOI 10.1016/S0065-2156(10)44002-8

[29] Simo, J., Hughes, T.: Computational Inelasticity. Springer, New York (1998)

[30] Squire, W., Trapp, G.: Using complex variables to estimate derivatives of real functions. SIAM Review pp. 110–112 (1998)

[31] Team, C.D.: Cubit 14.0 user documentation. Tech. rep., Sandia National Laboratories (2013)

[32] Team, S.S.M.: Adagio 4.18 user's guide. Tech. Rep. SAND2010-6313, Sandia National Laboratories (2010)

[33] Voorhees, A., Millwater, H., Bagley, R.: Complex variable methods for shape sensitivity of finite element models. Finite elements in analysis and design **47**(10), 1146–1156 (2011). DOI 10.1016/j.finel.2011.05.003

[34] Young, T., Mohlenkamp, M.: Introduction to Numerical Methods and Matlab Programming for Engineers. Ohio University Department of Mathematics, OH (2009)

VITA

Michael Brothers studied at Trinity University in San Antonio, Texas where he earned a B.S. in engineering science. After a period of work in industry at the United Stated Patent and Trademark Office as a patent examiner, he returned to San Antonio to resume his studies in mechanical engineering at the University of Texas at San Antonio, where he pursued a M.S. in mechanical engineering, for which this thesis partially fulfilled requirements of graduation. During his masters studies, as a graduate research assistant, he researched complex number based numerical differentiation techniques for the application of computing tangent-stiffness matrices within a massively parallel computational peridynamics code. Following his graduation he continued his education, pursuing a Ph.D in mechanical engineering at UTSA.