COMPDYN 2015
5th ECCOMAS Thematic Conference on
Computational Methods in Structural Dynamics and Earthquake Engineering
M. Papadrakakis, V. Papadopoulos, V. Plevris (eds.)
Crete Island, Greece, 25–27 May 2015

EFFICIENT NONLINEAR DYNAMIC ANALYSIS OF FRAME STRUCTURES

Johannes Burtscher

Unit of Applied Mechanics, University of Innsbruck
Technikerstr. 13, 6020 Innsbruck, Austria
e-mail: johannes.burtscher@uibk.ac.at

Keywords: nonlinear dynamic analysis; sparse linear systems; matrix factor modification.

Abstract. *This paper presents the application of multiple-rank modifications to the sparse Cholesky factorization* $\mathbf{LDL}^T$ *for faster nonlinear finite element analysis of moment resisting frames. Such frames resist lateral forces primarily by bending moments at the joints and are often modeled by the concept of concentrated plasticity. That is, the joints are represented by rotational springs with nonlinear material behavior, whereas the beams and columns remain linear elastic. This implies that within a materially-nonlinear-only finite element analysis, large parts of the system matrix remain unchanged. The Cholesky factors, however, are completely recomputed amongst solution steps and no reuse is made of the previous factorization. I address this drawback with a multiple-rank update routine implemented in the publicly available library CHOLMOD. The routine allows an update or downdate of a sparse* $\mathbf{LDL}^T$ *factorization with a runtime proportional to the number of nonzero entries in* $\mathbf{L}$ *and* $\mathbf{D}$ *that change. Exemplary simulations show that the proposed method can be substantially faster if changes to the simulation's system matrix can efficiently be expressed as a multiple-rank update problem. It also becomes apparent that the method is not restricted to frame problems but rather can be applied to any physical phenomenon where nonlinearities are locally bounded.*

1 INTRODUCTION

Realistic analysis of engineering structures often requires nonlinear methods. The occurring nonlinearities stem from various reasons such as inelastic material behavior, large displacements and rotations, or complex boundary conditions. For many practical applications only nonlinear material behavior is of concern. Often, these nonlinearities are further restricted to local domains whereas large parts of the structure remain linear elastic. Standard finite element codes used for structural dynamic analysis do not take advantage of this fact, which leads to simulations that are ultimately slower than necessary.

Finite element methods discretize the governing partial differential equations in space, leading, in case of transient problems, to a set of ordinary differential equations. These can be solved by direct integration, which ultimately means to solve a large sparse linear system in the form of

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{1}$$

in each time step. If the underlying physical behavior is nonlinear and an implicit integration scheme is used in combination with a direct solver, the solution of Eq. (1) requires the repeated computation of some decomposition of $\mathbf{A}$. The computation of this factorization is a rather expensive operation and the costs cannot be amortized across time steps. This fact has motivated the search for methods, which allow for modifications of an existing factorization in order to obtain the new factors more efficiently.

In this article I present the application of such an update routine in the context of nonlinear structural dynamic analysis of frame problems. The method is substantially faster than full refactorization and requires only minor extensions to the finite element code.

2 FINITE ELEMENT METHOD

The semi-discrete equations of motion of a structural system read

$$\mathbf{M}\ddot{\mathbf{x}}(t) + \mathbf{C}\dot{\mathbf{x}}(t) + \mathbf{f}_{\text{int}}(\mathbf{x}(t), \boldsymbol{\kappa}) = \mathbf{f}_{\text{ext}}(t) \tag{2}$$

with $\mathbf{M}$ being the mass matrix, $\mathbf{C}$ the damping matrix, $\mathbf{f}_{\text{int}}$ the internal force vector, and $\mathbf{f}_{\text{ext}}$ the external force vector. Both $\mathbf{M}$ and $\mathbf{C}$ are herein assumed to be constant. The global unknowns are contained in the displacement vector $\mathbf{x}$ with $\dot{\mathbf{x}}$ and $\ddot{\mathbf{x}}$ being the first and second derivative with respect to time, that is, velocities and accelerations. $\boldsymbol{\kappa}$ denotes a set of internal variables, needed for the treatment of inelastic, path-dependent materials. The exact derivation of the equilibrium equation is given, together with a thorough introduction to the finite element method, e. g., by Bathe [1].

Eq. (2) is a system of ordinary differential equations of second order and can be solved by direct integration. For this purpose, equilibrium is only satisfied at discrete time intervals Δt apart, and a variation of displacements, velocities, and accelerations is assumed in each interval. Widely used is thereby Newmark's assumption [2] that accelerations vary linearly from time t to time $t + \Delta t$, so that

$$\dot{\mathbf{x}}^{t+\Delta t} = \dot{\mathbf{x}}^{t} + \Delta t \left[(1-\gamma)\ddot{\mathbf{x}}^{t} + \gamma\ddot{\mathbf{x}}^{t+\Delta t}\right] \tag{3}$$

$$\mathbf{x}^{t+\Delta t} = \mathbf{x}^{t} + \Delta t\dot{\mathbf{x}}^{t} + \tfrac{1}{2}\Delta t^2 \left[(1-2\beta)\ddot{\mathbf{x}}^{t} + 2\beta\ddot{\mathbf{x}}^{t+\Delta t}\right] \tag{4}$$

where the parameters β and γ determine integration accuracy and stability. Solving for $\ddot{\mathbf{x}}^{t+\Delta t}$ from Eq. (4) and substituting for $\ddot{\mathbf{x}}^{t+\Delta t}$ into Eq. (3) yields equations for accelerations and velocities at time $t + \Delta t$ in terms of the unknown displacements $\mathbf{x}^{t+\Delta t}$. Inserting these two equations into Eq. (2) yields now the fully discretized equations of motion as a system of nonlinear algebraic equations

$$\mathbf{M}\Big[\tfrac{1}{\beta\Delta t^2}(\mathbf{x}^{t+\Delta t} - \mathbf{x}^t) - \tfrac{1}{\beta\Delta t}\dot{\mathbf{x}}^t - \tfrac{1-2\beta}{2\beta}\ddot{\mathbf{x}}^t\Big] + \mathbf{C}\Big[\tfrac{\gamma}{\beta\Delta t}(\mathbf{x}^{t+\Delta t} - \mathbf{x}^t) + (1 - \tfrac{\gamma}{\beta})\dot{\mathbf{x}}^t + (1 - \tfrac{\gamma}{2\beta})\Delta t\,\ddot{\mathbf{x}}^t\Big] + \mathbf{f}_{\text{int}}(\mathbf{x}^{t+\Delta t}) - \mathbf{f}_{\text{ext}}^{t+\Delta t} = \mathbf{0} \quad (5)$$

The solution for each time step can then be found with iterative procedures.

One commonly applied iterative procedure to solve Eq. (5) is the Newton–Raphson method. It successively finds better approximations to the roots of a vector function $\mathbf{F}(\mathbf{x})$ with its Jacobian $\mathbf{J_F} = \frac{d\mathbf{F}}{d\mathbf{x}}$ and an already known state $\mathbf{x}_j$, according to

$$\mathbf{J_F}(\mathbf{x}_j)\,\Delta\mathbf{x}_{j+1} = -\mathbf{F}(\mathbf{x}_j) \quad \text{with} \quad \Delta\mathbf{x}_{j+1} = \mathbf{x}_{j+1} - \mathbf{x}_j \quad (6)$$

The resulting linear system is solved in each iteration j until some error measure lies within a given tolerance. Using Eq. (6) with the discretized equations of motion as $\mathbf{F}(\mathbf{x})$, leads with

$$\left.\frac{\partial \mathbf{f}_{\text{int}}}{\partial \mathbf{x}^{t+\Delta t}}\right|_{\mathbf{x}_j^{t+\Delta t}} = \mathbf{K}(\mathbf{x}_j^{t+\Delta t})$$

finally to a relation in the form of Eq. (1)

$$\Big[\tfrac{1}{\beta\Delta t^2}\mathbf{M} + \tfrac{\gamma}{\beta\Delta t}\mathbf{C} + \mathbf{K}(\mathbf{x}_j^{t+\Delta t})\Big]\,\Delta\mathbf{x}_{j+1} = \mathbf{f}_{\text{ext}}^{t+\Delta t} - \mathbf{f}_{\text{int}}(\mathbf{x}_j^{t+\Delta t}) - \mathbf{M}\ddot{\mathbf{x}}_j^{t+\Delta t} - \mathbf{C}\dot{\mathbf{x}}_j^{t+\Delta t} \quad (7)$$

for finding “dynamic equilibrium” within a time step.

3 DIRECT SOLUTION AND UPDATE PROCEDURE

For the solution of Eq. (7) either direct or iterative methods can be used. Direct methods find an exact solution within a predetermined number of operations, whereas iterative methods produce a sequence of approximate solutions that should ultimately converge to the exact one. As long as memory is of no concern, that is, problems with low number of unknowns, direct solvers are preferable because of their accuracy and robustness. I thus use CHOLMOD [3], a sparse direct solver for symmetric, positive-definite matrices authored by Tim Davis.

Direct solution techniques solve a linear system by some decomposition of $\mathbf{A}$, being basically a variation of Gauss elimination [4]. If $\mathbf{A}$ is a real, symmetric matrix, the general form of this decomposition is

$$\mathbf{A} = \mathbf{LDL}^{\mathrm{T}}$$

where $\mathbf{L}$ is a unit lower triangular matrix and $\mathbf{D}$ a diagonal matrix. With this factors at hand, a solution for $\mathbf{x}$ can be obtained by forward reduction, diagonal scaling, and backward substitution, which are simple and efficient methods for computing $\mathbf{L}^{-1}\mathbf{b}$, $\mathbf{D}^{-1}(\mathbf{L}^{-1}\mathbf{b})$, and $\mathbf{L}^{-\mathrm{T}}(\mathbf{D}^{-1}\mathbf{L}^{-1}\mathbf{b})$ in succession. The computation of $\mathbf{L}$ and $\mathbf{D}$, however, can be quite time consuming and has obviously to be repeated each time the linearized system's matrix $\mathbf{A}$ changes.

A change to $\mathbf{A}$ can most generally be expressed as $\mathbf{A}_{j+1} = \mathbf{A}_j + \Delta\mathbf{A}_{j+1}$. A special case and one of particular interest is a so-called *rank-one modification*

$$\mathbf{A}_{j+1} = \mathbf{A}_j + \alpha\mathbf{v}\mathbf{v}^{\mathrm{T}}$$

where α is a scalar and $\mathbf{v}$ a vector of appropriate size. Finding the factorization of $\mathbf{A}_{j+1}$ after such a modification is usually described as a problem of *updating* the factors of $\mathbf{A}_j$ and its importance has long been recognized. An overview of several update schemes can be found in the classic paper by Gill et al. [5] published back in 1974. More recently, Davis and Hager [6] took up some ideas given therein and developed an algorithm in regard of modern sparse methods and extended it in further consequence to the multiple rank case [7]. The resulting algorithm is implemented in CHOLMOD as routine `cholmod_updown` and incorporated in the finite element code used for this study.

The update/downdate routine `cholmod_updown` takes an $\mathbf{LDL}^\mathrm{T}$ factorization and computes a new factorization according to

$$\overline{\mathbf{LDL}}^\mathrm{T} = \mathbf{LDL}^\mathrm{T} \pm \mathbf{CC}^\mathrm{T}$$

where $\mathbf{A} + \mathbf{CC}^\mathrm{T}$ corresponds to an update and, conversely, $\mathbf{A} - \mathbf{CC}^\mathrm{T}$ to a downdate. Apparently, it is necessary to express $\Delta\mathbf{A}$ as $\mathbf{CC}^\mathrm{T}$ to employ this routine. In so doing, I make use of the idea stated at the beginning that in a materially-nonlinear-only analysis large parts of a structure usually remain linear elastic. This implies that $\Delta\mathbf{A}$ should be extremely sparse and should hold a rank much smaller than the dimension of $\mathbf{A}$. The $n \times n$ matrix $\Delta\mathbf{A}$ can then conveniently be expressed as

$$\Delta\mathbf{A} = \mathbf{PWP}^\mathrm{T}$$

where $\mathbf{P}$ is an $n \times m$ sparse incidence matrix consisting solely of zeros and ones, and $\mathbf{W}$ is the the change to the linearized system's matrix in compressed (and dense) form of size $m \times m$ with $m \leq n$. Computing the eigendecomposition

$$\mathbf{W} = \mathbf{Q\Lambda Q}^\mathrm{T}$$

and separating the eigenvalues together with the corresponding eigenvectors in positive and negative ones, yields $\mathbf{C}$ in case of an update as

$$\mathbf{C}^+ = \mathbf{PQ}^+(\mathbf{\Lambda}^+)^{1/2}$$

and in case of a downdate as

$$\mathbf{C}^- = \mathbf{PQ}^-|\mathbf{\Lambda}^-|^{1/2}$$

It should be obvious by now that the initial assumption of $\Delta\mathbf{A}$ being extremely sparse is crucial, as the eigendecomposition of the dense matrix $\mathbf{W}$ is an expensive operation where the costs can easily outweigh a refactorization from scratch.

4 EXEMPLARY STRUCTURE AND NUMERICAL RESULTS

I implemented the `cholmod_updown` routine in my finite element code and tested its performance against the standard methods from SuperLU [8] and CHOLMOD. The moment resisting frame depicted in Fig. 1 thereby serves as a benchmark example. Such frames are amongst the most common forms in modern engineering and basically are steel assemblages of beams and columns rigidly connected to each other. They withstand lateral forces primarily by bending and should by design avoid potentially brittle shear failure modes. To achieve a ductile behavior during strong earthquake shaking, the beam ends and column bases are designed to sustain large inelastic rotations and thus provide an energy dissipating mechanism.

For the numerical model, the concept of concentrated plasticity is used. Although these models date back to the 1960s, they are the still amongst the preferred modeling approaches for seismic

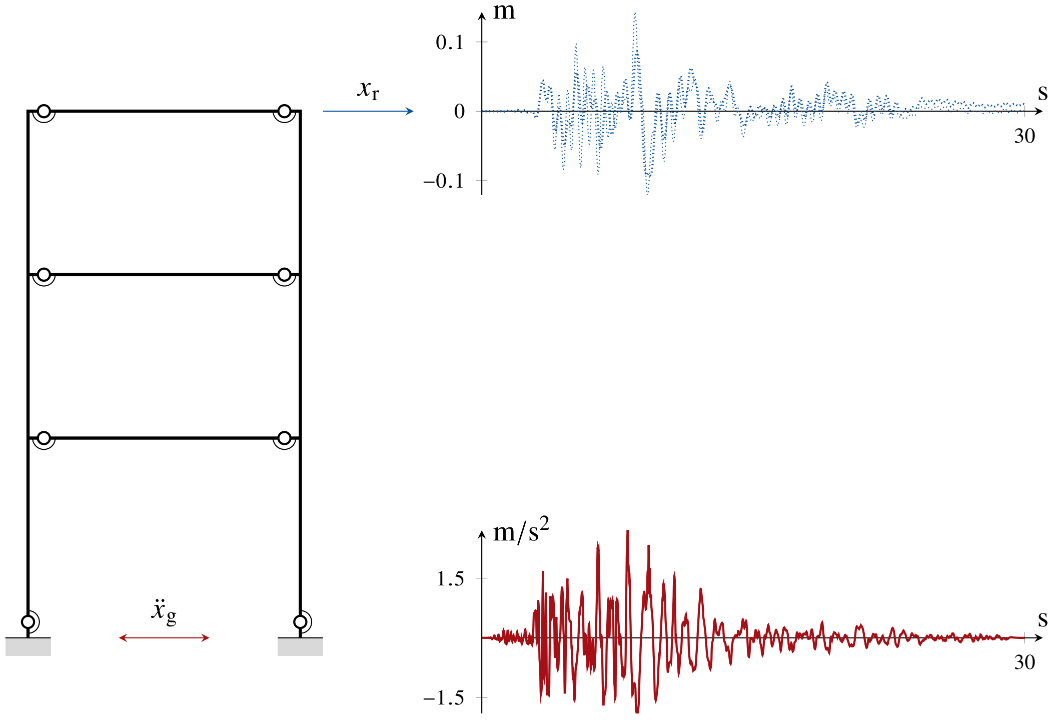

Figure 1: A three story moment resisting frame with nonlinear rotational springs at the beam ends and the column bases, subjected to the ground motion record from the 1994 Northridge earthquake (Beverly Hills, 14145 Mulholland station). The top right time history plot shows the horizontal roof displacement for linear-elastic springs (dotted line) and for perfectly plastic springs (solid line).

performance assessment in earthquake engineering. Within this concept, beams and columns are modeled as two-node linear elastic beam elements, whereas the parts which are designed to experience inelastic deformations, are modeled with rotational springs of zero length. The material behavior of these springs is for the case at hand elastic-perfectly plastic.

The exemplary structure is subjected to ground motions from the 1994 Northridge earthquake (Beverly Hills, 14145 Mulholland station) and the horizontal roof displacement is chosen as engineering demand parameter. The frame is 6 m wide and 11.6 m high, resulting from the first story being 4.4 m and the remaining ones being each 3.6 m high. The flexural rigidity of the beams and columns is assumed to be $1 \cdot 10^6\,\mathrm{kN/m^2}$ and $2 \cdot 10^6\,\mathrm{kN/m^2}$, respectively, and all springs have an elastic stiffness of $210 \cdot 10^6\,\mathrm{kN/m^2}$. The results of a linear elastic simulation is shown in the top right time history plot of Fig. 1 as dotted line. For the nonlinear response, the material behavior of the springs was modified such that they start yielding at a bending moment of 2400 kN m (base), 1300 kN m (first floor), 1000 kN m (second floor), and 800 kN m (roof). The nonlinear results are depicted in the same plot as solid line.

The nonlinear simulation was run three times: with the standard solution procedure, that is, a complete refactorization of $\mathbf{A}$ in each solution step, using first SuperLU and second CHOLMOD. In the third run I used the update routine `cholmod_updown` and applied incremental changes $\Delta\mathbf{A}$ to the system's linearized matrix. The runtime of all three approaches is shown in Fig. 2. The top graph shows the time needed for each individual solve, whereas the bottom graph shows the same data in accumulated form. Of all three solvers, SuperLU is slowest which is apparent as the library is intended to solve non-symmetric linear systems. These timings reflect no weakness of SuperLU, but rather emphasize that only specially tailored routines achieve really high performance. The standard solve from CHOLMOD, designed for symmetric matrices, thus outperforms SuperLU

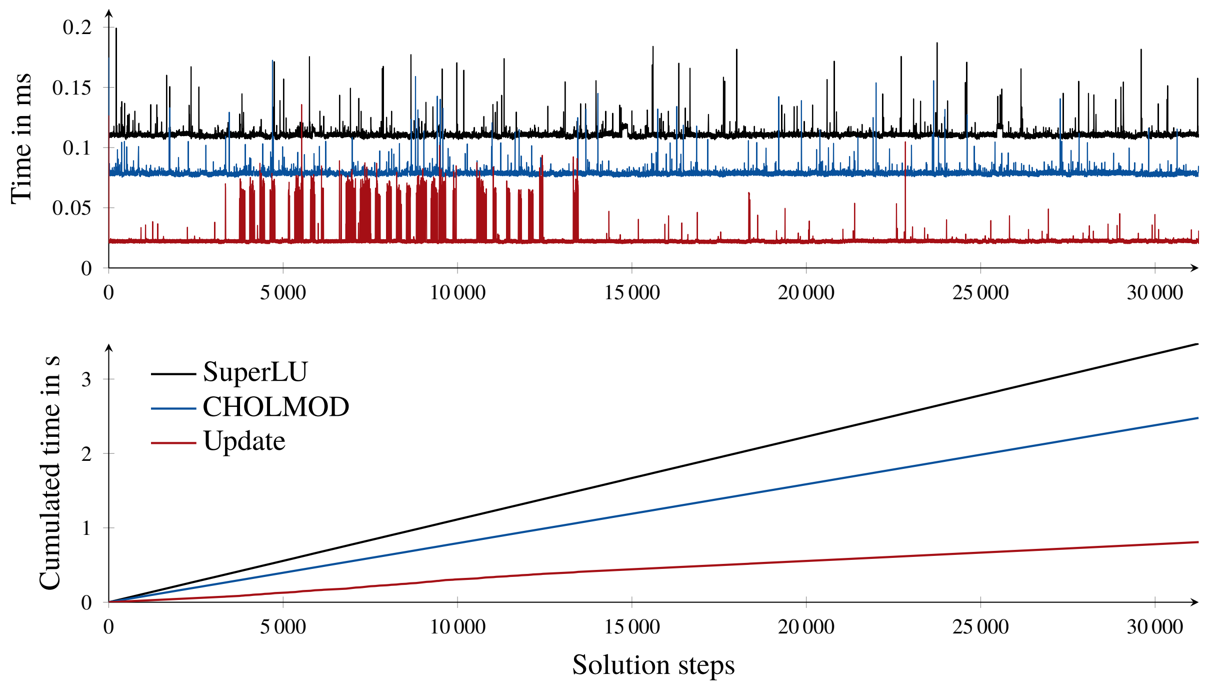

Figure 2: Runtime over solution steps for solving Eq. (7) of the exemplary structure individually (top graph) and in accumulated form (bottom graph) using three different methods. SuperLU is slowest due to its general applicability to non-symmetric linear systems, whereas CHOLMOD exploits this fact and is faster by a factor of 1.4. The update routine logically has erratic performance and is in the best case around 3.8 times faster than CHOLMODs standard solve. If the rank of the update matrix $\Delta\mathbf{A}$ increases, the performance hit is evident.

by a factor of 1.4. The update procedure compares badly with the two previous methods because it obviously has an erratic performance. In the best case where no update is needed, it is 3.8 times faster than the reference solve from CHOLMOD. The time needed therein results only from the solution phase (forward reduction to backward substitution) and eventually poses a lower bound. In the worst-case scenario, the update routine is on par with CHOLMOD.

5 CONCLUSION

The solution of Eq. (7) is one the most time consuming parts in a finite element solution. If a direct solver is used, this can be attributed to the computation of the $\mathbf{LDL}^{\mathrm{T}}$ factorization. For nonlinear problems, this process has to be repeated in each iteration and can be problematic regarding the overall performance. I address this problem by an update procedure which allows for the modification of an existing factorization.

The runtime of the `cholmod_updown` routine is proportional to the numbers of nonzero entries in $\mathbf{L}$ and $\mathbf{D}$ that change. The problem, however, is the form of the update matrix. The finite element code can easily be modified to return $\Delta\mathbf{A}$, but it can be rather expensive to rewrite it in the form $\mathbf{CC}^{\mathrm{T}}$. If the compressed update matrix $\mathbf{W}$ is very small, the additional costs are negligible, but with increasing size, they inevitably get higher and will ultimately exceed the costs of a full refactorization. The crucial point is in consequence the central premise of local plastic regions which result in local changes to $\mathbf{A}$. This is, however, only true in the context of materially-nonlinear-only analysis. If geometric nonlinearities have to be considered, the rank of $\Delta\mathbf{A}$ becomes far too big for the update method to still be performant. The exemplary frame structure meets all requirements and thus allows for a considerably faster analysis using the update approach.

In future work, I will address the limiting factor of $\Delta\mathbf{A}$ being of small rank. By implication, the `cholmod_updown` routine has to be ruled out, but I believe that a different updating procedure can be devised which has, in the worst-case scenario, the same runtime as a complete refactorization from scratch.

REFERENCES

[1] K.-J. Bathe. *Finite Element Procedures*. Prentice Hall, 1996.

[2] N. M. Newmark. A Method of Computation for Structural Dynamics. *ASCE Journal of the Engineering Mechanics Division*, **85**, 67–94, 1959.

[3] Y. Chen, T. A. Davis, W. W. Hager, S. Rajamanickam. Algorithm 887: CHOLMOD, Supernodal Sparse Cholesky Factorization and Update/Downdate. *ACM Transactions on Mathematical Software*, **35**, 22:1–22:14, 2008.

[4] G. H. Golub, C. F. Van Loan. *Matrix Computations*, 4th Edition. John Hopkins University Press, 2013.

[5] P. E. Gill, G. H. Golub, W. Murray, M. A. Saunders. Methods for Modifying Matrix Factorizations. *Mathematics of Computation*, **28**, 505–535, 1974.

[6] T. A. Davis, W. W. Hager. Modifying a Sparse Cholesky Factorization. *SIAM Journal on Matrix Analysis and Applications*, **20**, 606–627, 1999.

[7] T. A. Davis, W. W. Hager. Multiple-rank Modifications of a Sparse Cholesky Factorization. *SIAM Journal on Matrix Analysis and Applications*, **22**, 997–1013, 2001.

[8] X. S. Li. An Overview of SuperLU: Algorithms, Implementation and User Interface. *ACM Transactions on Mathematical Software*, **31**, 302–325, 2005.