ARGONNE NATIONAL LABORATORY
9700 South Cass Avenue
Argonne, Illinois 60439

FAST SOLUTION OF NONLINEAR POISSON-TYPE EQUATIONS

Brett M. Averick* **and James M. Ortega†**

Mathematics and Computer Science Division

Preprint MCS-P262-0991

August 1991

*Army High Performance Computing Research Center, University of Minnesota, 1100 Washington Avenue South, Minneapolis, MN 55415. The research of this author was supported in part by the Army Research Office under grant DAALO3-89-C-0038.

†Department of Computer Science, University of Virginia, Charlottesville, VA 22903. The research of this author was supported in part by the National Aeronautics and Space Administration under grants NAG-1-1112-FDP and NAG-1-1050.

Fast Solution of Nonlinear Poisson-type Equations

by

Brett M. Averick and James M. Ortega

ABSTRACT

This paper is concerned with the solution of nonlinear Poisson-type equations. A change of variable is made that effectively reduces the solution of such equations to solving a linear Poisson equation followed by a series of one-dimensional nonlinear equations. Comparison with other methods shows the new method to be competitive.

1 Introduction

In [3], we considered methods for the numerical solution of nonlinear Poisson-type equations of the form

$$\nabla \cdot [K(u)\nabla u] = f, \tag{1.1}$$

where K is a positive differentiable function. If (1.1) is discretized by finite difference methods, we obtain a discrete system of the form

$$F(\mathbf{u}) = A(\mathbf{u})\mathbf{u} - \mathbf{b}(\mathbf{u}), \tag{1.2}$$

where the matrix $A(\mathbf{u})$ is symmetric positive definite for all $\mathbf{u}$. Newton's method applied to (1.2) is

$$F'(\mathbf{u}^k)\delta^{k+1} = -F(\mathbf{u}^k), \quad k = 0, 1, \ldots, \tag{1.3}$$

where the Jacobian matrix is of the form

$$F'(\mathbf{u}) = A(\mathbf{u}) + B(\mathbf{u}). \tag{1.4}$$

The Jacobian matrix has the usual sparsity of Poisson problems, and we would like to solve the Newton systems (1.3) by a conjugate gradient method. The matrix $B(\mathbf{u})$ in (1.4) is not symmetric, however, which would require using methods such as GMRES [8] for nonsymmetric systems.

An alternative approach considered in [3] is based on the fact that $\|B(\mathbf{u})\|$ is small compared with $\|A(\mathbf{u})\|$ in many cases. This motivated the use of the approximate Newton method

$$A(\mathbf{u}^k)\delta^{k+1} = -F(\mathbf{u}^{\mathbf{k}}), \quad k = 0, 1, \ldots, \tag{1.5}$$

in which the systems of (1.5) are solved by the Incomplete Cholesky Conjugate Gradient (ICCG) method.

In the present paper, we consider an entirely different approach based on the formulation [4] of (1.1) as

$$\nabla^2 \phi(u) = f. \tag{1.6}$$

If ϕ is a function such that

$$\phi'(u) = K(u), \tag{1.7}$$

then

$$\nabla^2 \phi(u) = \nabla \cdot (\phi'(u)\nabla u) = \nabla \cdot (K(u)\nabla u), \tag{1.8}$$

and (1.6) is equivalent to (1.1). Thus, we can obtain the solution of (1.1), in principle, by a two-stage process:

I. Solve the Poisson equation

$$\nabla^2 w = f. \tag{1.9}$$

II. Solve the one-dimensional nonlinear equations

$$\phi(u_P) = w_P, \tag{1.10}$$

where w_P denotes the solution of (1.9) at a point P in the domain.

In the actual algorithm we consider, the domain is first discretized so that (1.9) becomes a discrete Poisson equation, and w_P is the solution of this discrete problem at grid point P. Under the assumption that $K(u)$ is positive, solutions of (1.10) will be unique. If we assume further that $K(u)$ is bounded away from zero, then a solution of (1.10) will exist for any w_P.

Even on conventional machines this approach may have considerable merit since it allows the use of fast Poisson solvers whenever the domain is suitable. Results given in the next section show that for a small two-dimensional problem run on a Sun-3/60, this new method is considerably faster than the best method considered in [3]. On parallel and vector computers, the advantage is even greater since the solution of the nonlinear equations (1.10) has excellent parallelism (or vectorization) across the grid points. In the next section, we will give numerical results for a fairly large (250,000+ unknowns) three-dimensional problem considered in [3]. Comparison to the results in [3] on a CRAY 2 shows considerable superiority of the new method. The method does have limitations, however. These are given in Section 3, along with further discussion of parallelism and other properties.

2 Numerical Results

We first consider results on a serial computer - a Sun-3/60 - for a two-dimensional problem of the form (1.1) on the unit square and with $K(u) = \frac{10}{3} + \frac{9}{10}u$. We use the Dirichlet boundary conditions $u(x, y) = x^2 + y^2$ and choose the forcing function f of (1.1) so that $x^2 + y^2$ is the exact solution of (1.1). The boundary conditions for the Poisson equation (1.9) are given by

$$w_\Gamma = \phi(u_\Gamma), \tag{2.1}$$

where w_Γ and u_Γ are values of w and u on the boundary Γ, respectively.

We discretize the Poisson equation (1.9) by usual five-point finite differences on an $N \times N$ mesh of interior grid points. The discrete Poisson problem is then solved by a fast Poisson solver (FPS). Since K is linear, ϕ is quadratic, and the solutions of (1.10) are easily evaluated. However, for consistency with more general equations, we used a Newton iteration for these one-dimensional equations. Timing results using double-precision arithmetic are given in Table 1 for $N = 31$ and $N = 63$. The second column gives times for the FPS. NL is the time for solving the nonlinear equations (1.10). For comparison, Table 1 also lists times for the same problem using the TANICCG method from [3]. This method solves the approximate Newton systems (1.5) by ICCG(0), using truncation in the sense of [5] to determine the

Table 1: Times (seconds) for Two-Dimensional Problem on a Sun 3/60

N	FPS	NL	Total	TANICCG	Speedup
31	4.2	1.7	5.9	79.8	13.6
63	23.5	7.2	30.7	669.3	21.8

number of inner iterations; further details may be found in [3]. The last column of Table 1 gives the ratios of the TANICCG times to those of the new method.

We next consider a three-dimensional problem treated in [3] in which

$$K(u) = \frac{(100 + 27u)^{\frac{3}{2}}}{300 + 27u}, \tag{2.2}$$

so that

$$\phi(u) = 209.6 \tan^{-1}(.135u + .5)^{\frac{1}{2}} + (3u + 11.1)^{\frac{1}{2}}(2u - 37). \tag{2.3}$$

The domain is the unit cube, and the forcing function f of (1.1) was chosen so that $x^2+y^2+z^2$ is the exact solution of the differential equation. Thus, the boundary values are also given by $x^2 + y^2 + z^2$, and then the boundary values for the Poisson equation (1.9) are obtained from (2.1).

Table 2 gives timings for single-precision arithmetic on a single processor of a CRAY 2 for $N = 31$ (29,791 equations) and $N = 63$ (250,047 equations). As in Table 1, the second column is the time for the FPS, and NL is the time for solving the nonlinear equations (1.10). These one-dimensional nonlinear equations were solved by Newton's method, vectorized across all the grid points.

In [3], we used for (1.5) the initial approximation

$$u^0_{i,j,k} \;=\; x_i \,+\, y_j^2 \,+\, z_k^2 \;, \tag{2.4}$$

which is a linear interpolation of the boundary values in the x-direction. For the Newton iterations for (1.10), we also used (2.4) as well as choosing $u^0_{i,j,k}$ as the solution of the discrete Poisson problem at the i, j, k grid point. Of these two ways to choose $u^0_{i,j,k}$, (2.4) was the best and only these times are reported in Table 2. The next-to-last column gives times for the best version of the TANICGG method in [3], and the last column again shows the ratios of the TANICCG times to those of the new method. (The times for TANICCG given in [3] were obtained on a CRAY 2 at the NASA-Langley Research Center and do not agree with those of Table 2, which were obtained on a CRAY 2 at the University of Minnesota Army High Performance Computing Research Center.) In [3] we used the convergence test

$$\|F(\mathbf{u}^k)\|_2 \;\leq\; \|F(\hat{\mathbf{u}})\|_2, \tag{2.5}$$

Table 2: Times (seconds) for Three-Dimensional Problem on a CRAY 2

N	FPS	NL	Total	TANICCG	Speedup
31	.06	.05	.11	1.1	10
63	.31	.35	.66	17.4	25

where F is the function (1.2), $\mathbf{u}^k$ is the iterate of (1.5), and $\hat{\mathbf{u}}$ is the exact solution of the differential equation evaluated at the grid points. This test, although impractical in practice, ensures that the convergence error is commensurate with the discretization error. A similar test was used for the Newton iterates $u_P^{(k)}$ for (1.10):

$$| \phi(u_P^{(k)}) - w_P | \leq | \phi(\hat{u}_P) - w_P | . \tag{2.6}$$

Here, w_P is the solution of the discrete Poisson problem, and $\hat{u}_p$ is the exact solution of the differential equation. The test (2.6) was implemented as a vector compare across all the grid points. This may cause some additional work for equations that have already converged, but it has been our experience that this vectorization outweighs the extra computation. (No more than three Newton iterations were required for each equation.) Although (2.5) and (2.6) are different tests, each suited to their particular methods, we also used (2.5) for the vector of one-dimensional Newton iterates of the new method and found that the same number of iterations were required as using (2.6). The actual errors in the final iterates produced by the two methods differ by about 15%.

The times in Table 2 for the TANICCG method are significantly worse than those of the new method, but some caveats are in order. The ICCG method in [3] was implemented by using the red/black ordering to obtain long vector lengths for the CRAY 2. It is known (see, e.g., [6] and, for a recent review, [7]) that this ordering may seriously degrade the rate of convergence of ICCG, and it is quite likely that better results would be obtained by using the diagonal ordering, as, for example, in [1]. Also, more sophisticated versions of ICCG or the use of other methods might give better results. However, on the basis of the times in Table 2, it is reasonable to conjecture that no iterative method for solving the Newton systems (1.3), or some approximation of them such as (1.5), will be competitive with the new method. Moreover, our fast Poisson solver was a triple Fourier analysis method that used the vectorized one-dimensional fast sine transformations of VFFTPACK obtained from *netlib*. It is known that other fast Poisson solvers that use, for example, cyclic reduction, are faster so that this would give further advantage to the new method.

Table 3: Solution of Poisson Equation by ICCG on a CRAY 2.

N	ICCG	NL	Total	Previous Total
31	.527	.046	.57	.11
63	9.87	.38	10.25	.70

3 Further Discussion of the Method

Although the results in the preceding section indicate significant potential for the new method, we must point out limitations. First, it is necessary to integrate $K(u)$ to obtain $\phi(u)$. Even for the relatively simple function (2.2), the corresponding ϕ of (2.3) is rather complicated and was found only by using MACSYMA. For other K it may be impossible or impractical to obtain the integral explicitly. Recall that ϕ is needed in (2.1) to obtain the boundary values of the Poisson equation and to solve the nonlinear equations (1.10). In both cases, ϕ could be approximated by numerical integration, but this would introduce additional error as well as computing time. Moreover, it seems impossible to apply this approach if K is also an explicit function of the spatial variables, $K = K(u, x, y, z)$, or if K is a vector, as in the equation $(K_1(u)u_x)_x + (K_2(u)u_y)_y = f$.

One reason the times presented in the preceding section were so good is that a fast Poisson solver (FPS) could be used. If the domain is such that this is not possible, then solution of the Poisson equation may be considerably more expensive. This is illustrated in Table 3 where the discrete Poisson equation of the three-dimensional problem of the preceding section is solved by ICCG. The time for this is givin in the column labeled ICCG; NL is, as before, the time to solve the nonlinear equations (1.10). The last column in Table 3 reproduces the times from Table 2 using the FPS. Note that even without the FPS, the new method is still almost twice as fast as the TANICCG method.

Clearly, the new method is potentially very good for parallel computation since the solutions of the one-dimensional equations are completely independent. However, the load balancing could be degraded by an unequal number of Newton iterations on different equations. For example, if the initial approximation at grid point P_1 is much better than that at grid point P_2, the Newton iteration at P_2 could take longer to converge. On the other hand, on distributed memory machines, no communication is needed in the Newton iterations except for the convergence test. Most important, if the domain is such that a fast Poisson solver can be used, then a good FPS routine must be available for the particular parallel machine.

Finally, another possible benefit of the new method is discretization error. The best results we have been able to obtain mathematically for the discretization (1.2) of (1.1) is that the local discretization error is $0(h)$, although experimental results indicate it is

probably $0(h^2)$. For the new method, however, the only discretization error occurs in the Poisson equation and is $0(h^2)$.

Acknowledgments

We are indebted to Professor Irena Lasiecka of the University of Virginia for reference [4] and to Barry F. Smith of Argonne National Laboratory for his assistance in using the VFFTPACK package.

References

[1] C. Ashcraft and R. Grimes, *On vectorizing incomplete factorization and SSOR preconditioners*, SIAM J. Sci. Stat. Comput., 9 (1988), pp. 122–151.

[2] B. Averick, *Solution of Nonlinear Poisson-type Equations*, Ph.D. Dissertation. Dept. Applied Math., University of Virginia, Charlottesville, VA., January 1991.

[3] B. Averick and J. Ortega, *Solution of nonlinear Poisson-type equations*, Applied Numerical Math, to appear.

[4] M. Crandall, *Semigroups of nonlinear transformations in banach spaces*, in Contributions to Nonlinear Functional Analysis, E. Zarantonello, ed., Academic Press, 1971, pp. 157–179.

[5] R. Dembo, S. Eisenstat, and T. Steihaug, *Inexact Newton methods*, SIAM J. Numer. Anal., 19 (1982), pp. 400–408.

[6] I. Duff and G. Meurant, *The effect of ordering on preconditioned conjugate gradients*, BIT, 29 (1989), pp. 635–657.

[7] J. Ortega, *Orderings for conjugate gradient preconditionings*, SIAM J. Optimization, to appear.

[8] Y. Saad and M. Schultz, *GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM J. Sci. Stat. Comput., 7 (1986), pp. 856–869.