AN EVOLUTIONARY ALGORITHM FOR DIAGONAL SCALED PRECONDITIONED CONJUGATE GRADIENT

PETER PENNEY

An Evolutionary Algorithm for Diagonal Scaled Preconditioned Conjugate Gradient

by

© Peter Penney

a Project Report submitted

to the School of Graduate Studies

in partial fulfillment of the

requirements for the

degree of Master of Science

Computational Science

Memorial University of Newfoundland

April 2008

St. John's, Newfoundland and Labrador

Abstract

The Preconditioned Conjugate Gradient method (PCG) is an iterative method used to solve linear systems of equations Ax=b, were A is often a large and sparse matrix. In the PCG method, diagonal scaling (Jacobi scaling) may be used to precondition the Matrix A so that the method converges in fewer iteration steps than the conventional conjugate gradient method. Diagonal scaling is carried out by using the diagonal of A as a pre-conditioner at each conjugate gradient iteration step. This project proposes a novel approach using an evolutionary algorithm to evolve different diagonal matrices to precondition the Matrix A at each iteration step. The evolutionary algorithm proceeds by applying computational crossover and mutation operators to generate a small population of matrices. The diagonal matrix resulting in the lowest relative residual in the population (higher fitness) is selected to pre-condition the Matrix A for the next iteration. Subsequently, a new generation of diagonal matrices will compete to become the preconditioner for the following iteration. This process continues for the number of iteration steps defined by the PCG Method. Results from conventional diagonal scaled PCG method will be compared with the evolutionary algorithm based diagonal scaled PCG method.

Acknowledgments

The author would like to thank his supervisor, Dr. George Miminis for his guidance and direction throughout his research and graduate studies.

The author would also like to thank Dr. Manrique Mata-Montero and Dr. Paul Gillard for their examination, review and critique of this project. Thanks also to Dr. Tina Yu for her assistance during the author's research preparation.

Thank you also to Gail Kenny for her assistance finalizing the project.

Thanks to my loving wife Marion for her patience and support throughout the author's research and preparation for this project.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

AMD	Advanced Micro Devices
CG	Conjugate Gradient
DSPCG	Diagonally Scaled Preconditioned Conjugate Gradient
EA	Evolutionary Algorithm
FDM	Finite Difference Method
FEM	Finite Element Method
GHz	Gigahertz
LU	Lower Upper (refers to Matrix decomposition into Lower and Upper Triangular Matrices)
PCG	Preconditioned Conjugate Gradient
RAM	Random Access Memory
SOR	Successive Over Relaxation
SPD	Symmetric Positive Definite

Introduction

This project outlines an Evolutionary Algorithm (EA) for diagonal scaled preconditioned conjugate gradient method (DSPCG). The DSPCG Method is an iterative method for solving a set of linear equations of the form **Ax=b**, where A is an n by n Matrix (usually sparse) and x and b are both n vectors, the vector x is the unknown and b is referred to as the right hand side vector.

First we discuss a related paper by Jun He et. al., then we proceed with a discussion of the Conjugate Gradient, the PCG and DSPCG Methods. Next, we give an overview of an EA for DSPCG. Finally, we will compare and discuss the results from the conventional DSPCG tests with the EA DSPCG.

Systems of equations obtained from the University of Florida's Matrix Market[10] were solved using both DSPCG and EA DSPCG. Timing results for both the conventional and EA DSPCG algorithms were compared and contrasted for the various matrices obtained from the Matrix Market. The table below lists the properties of the six matrices used during testing.

Table 1: Test Matrices Properties

Name	Size	Number Non-Zero Elements	% Non-Zero	Condition Number x 10^9 (Est.)	Notes
BCSSTK14	1,806	63,454	1.945	1.3	Roof of the Omni Coliseum, Atlanta
BCSSTK15	3,948	117,816	0.756	6.5	Module of an offshore platform
BCSSTK16	4,884	290,378	1.217	4.9	U.S. Army Corps of Engineers dam
BCSSTK17	10,974	428,650	0.356	13.0	Elevated pressure vessel
BCSSTK18	11,948	149,090	0.104	43.0	R.E. Ginna Nuclear Power Station
S3DKQ4M2	90,449	2,455,670	0.030	190.0	Finite element analysis of cylindrical shells

Related Work

To the best of the author's knowledge, the paper by Jun He et. al.[5] is the closest reference available which relates to the topic of this project. Their work is only similar in that it used an EA to solve a system of linear equations. Jun He et. al. used an EA to arrive at an optimal over-relaxation factor for which they used the Successive Over Relaxation Method (SOR) to solve the system of linear equations. In this project we investigate the idea of an EA in relation to the PCG method.

The Diagonal Scaled Preconditioned Conjugate Gradient (DSPCG) and the Successive Over Relaxation Method are iterative methods used to solve linear systems of equations. The DSPCG method that this project deals with is mathematically very different from the SOR Method. An exhaustive literary search by this project's author failed to find any application of EAs together with the PCG method.

The Preconditioned Conjugate Gradient Method

Systems of linear equations arise very often in practice, for example, when Finite Element Methods (FEM) or Finite Difference Methods (FDM) are employed to solve physical problems (e.g. determining the forces in a structural member such as an aircraft wing). The matrices arising from the solution of the partial differential equation associated with FEM and FDM are very large (10,000 x 10,000 and larger) and very sparse (less than 1% of the elements are non-zero).

There are two basic methods for solving systems of equations: direct and iterative. Direct methods, which include Gaussian Elimination and Cholesky Decomposition, solve the system in a predetermined number of steps by transforming the original matrix A to a triangular matrix. During this process however, many of the zero elements of the matrix A are changed to non-zero values (fill-in). This could be very inefficient since only the non-zero elements of a sparse matrix are stored. Furthermore, the special data structure of sparse matrices are generally more difficult to update (add or delete elements).

Therefore, when working with sparse matrices every attempt should be made to maintain their sparsity and avoid transformation of the matrix. Iterative methods in general work with the original matrix throughout the solution process. Since there is no fill-in of the zero elements in the matrix, iterative methods are well suited for use with large sparse matrices.

Matrices arising from both the FEM and FDM are often Symmetric Positive Definite (SPD). A matrix A is Symmetric Positive Definite if:

1) $A=A^T$ (i.e., A is symmetric) and

2) $A \in \Re^{n \times n}$ is positive definite if $x^T Ax > 0$ for all nonzero $x \in \Re^n$ (for example, Golub and Van Loan[4] p. 140)

SPD matrices have several desirable properties. For example, since they are symmetric, only the upper (or lower) triangle of the matrix needs to be stored. Pivoting, which ensures that algorithms have desirable numerical properties but could be computationally expensive, is not needed when matrices are positive definite. Positive definite matrices appear very often in practice. For example, the global stiffness matrix resulting from the FEM solution to a partial differential equation is SPD.

So, Gaussian elimination can be applied to SPD matrices without pivoting. The symmetric property of A means that the overall storage requirement of any algorithm can be reduced by half. A matrix being SPD also means that the decomposition step proceeds quickly since memory movement of matrix rows and/or columns is reduced (i.e., no pivoting is required in the decomposition).

Both the PCG and the Cholesky Decomposition methods may be applied to SPD matrices. Furthermore, for SPD matrices, the PCG method will converge for any initial guess of the solution x (i.e., x_0) (Golub and Van Loan[4] p. 522).

Method	Solution Type	Operation Count	Matrix Type
Cholesky	Direct	$n^3/3$	SPD
LU (Gaussian Elimination)	Direct	$2n^3/3$	General
PCG	Iterative	varies up to n^3	SPD

There are many iterative methods available for solving a system of equations, but by far, the most commonly used is the PCG Method. Memory requirements for PCG are generally much lower than the Cholesky or Gaussian Elimination direct methods. Other iterative methods such as SOR (Successive Over-Relaxation) may be used for full or non-SPD matrices but these methods are generally much slower.

Iterative methods in general, including the PCG method, proceed as follows:

1. Start with an initial guess x_0 for the solution of $Ax = b$
2. Calculate a sequence of approximate solutions $x_1, x_2, \ldots, x_m$. Each x_i is calculated as a linear function of A, b, x_{i-1} for i=1,2,…,m-1. Under specific criteria, possibly different for each method, the sequence converges to the correct solution x_m within a specified tolerance bounded by the computer precision.

Method of Steepest Descent

The Conjugate Gradient Method is derived from the Method of Steepest Descent. The Method of Steepest Descent is an iterative method that computes the value of the next approximate solution x_i by selecting the direction of the largest *gradient* of the *Quadratic Form*[8] of Ax=b. The *Quadratic Form* of Ax=b is defined as:

$$f(x) = \frac{1}{2}x^T Ax - b^T x$$

Its *gradient* is defined as:

$$f'(x) = \begin{bmatrix} \frac{\partial}{\partial x_1} f(x) \\ \frac{\partial}{\partial x_2} f(x) \\ \vdots \\ \frac{\partial}{\partial x_n} f(x) \end{bmatrix} = \frac{1}{2}A^T x + \frac{1}{2}Ax - b$$

For symmetric matrices where $A=A^T$ we have $f'(x) = Ax-b$. Therefore, solving Ax=b is equivalent to minimizing the above *Quadratic Form*.

As an example consider the following set of equations:

$3x_1 + 2x_2 = 2$

$2x_1 + 6x_2 = -8$

The exact solution to this system of equations is:

$x_1 = 2; x_2 = -2$

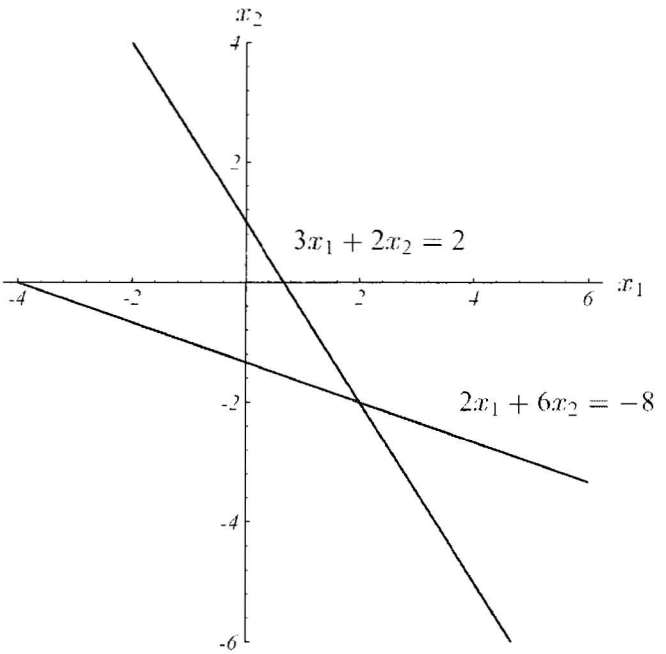

Figure 1) Sample System of Equations with solution at [2,-2] (Shewchuk, J.[8])

with *Quadratic Form* looking as:

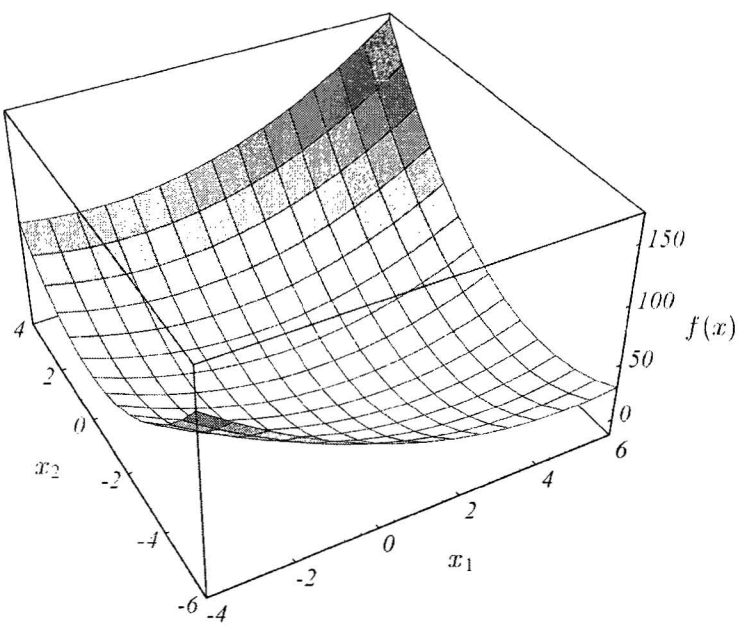

Figure 2) Quadratic form of Sample System of Equations (Shewchuk, J.[8])

The *gradient* and the solution by *steepest decent* will look as:

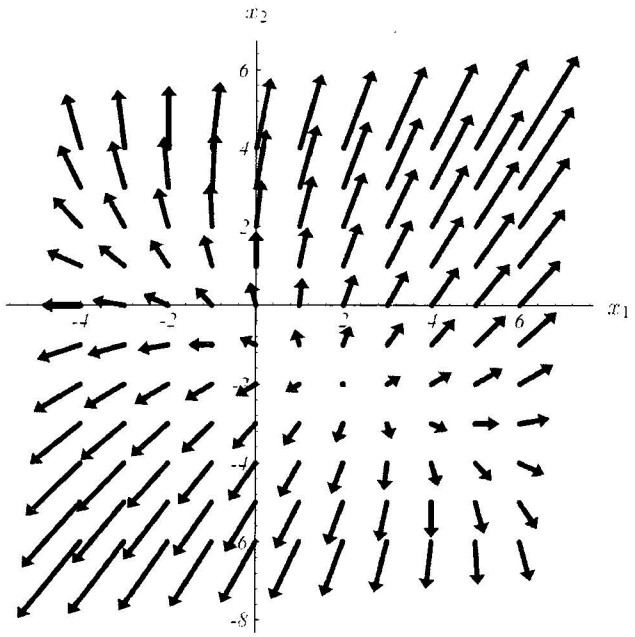

Figure 3) Gradient f'(x) of the Quadratic Form (Shewchuk, J.[8]). Note that at [2,-2] the gradient is perpendicular to the x_1, x_2 plane.

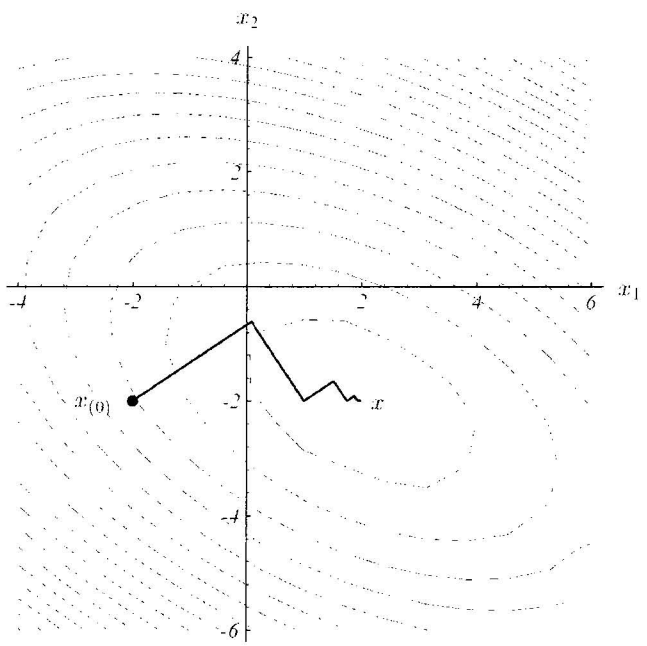

Figure 4) Solution by Steepest Descent from starting point [-2,-2] (Shewchuk, J.[8]).

Conjugate Gradient

Selecting A-orthogonal search directions d_0, d_1, … d_n will force the solution to converge to a solution for each dimension x_i. Two vectors u and v are said to be A-orthogonal, or conjugate, if $u^T Av = 0$

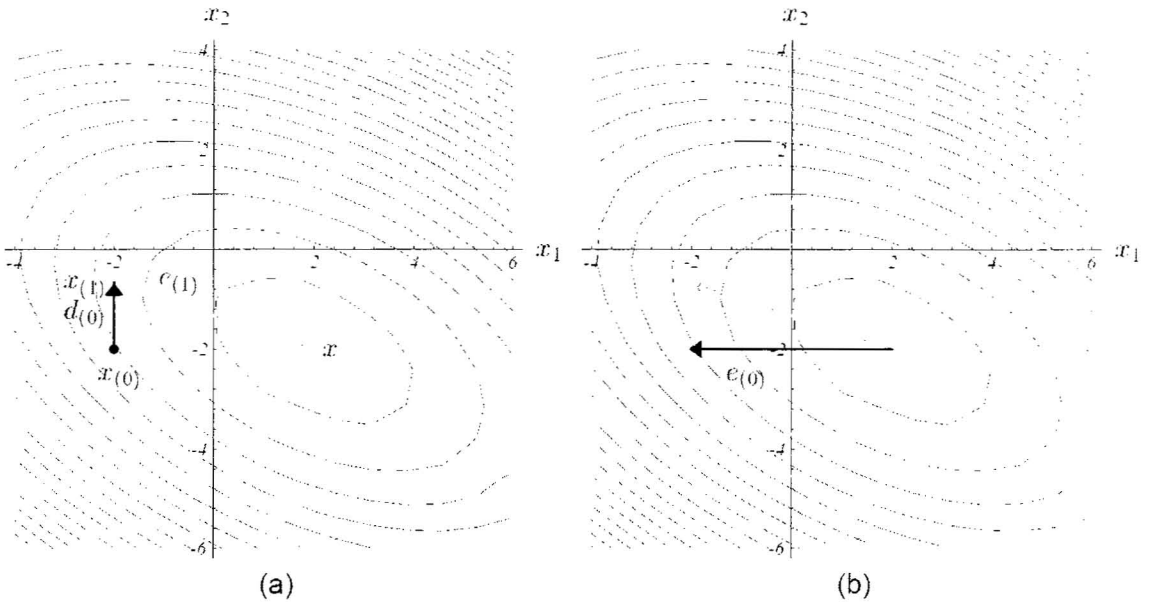

Figure 5) The method of Conjugate Directions converges in n steps
(a) The first step is taken along some direction d_0. The minimum point x_1 is chosen by the constraint that e_1 must be A-orthogonal to d_0. (b) The initial error e_0 can be expressed as a sum of A-orthogonal components (gray arrows). Each step of Conjugate Directions eliminates one of these components. (Shewchuk, J.[8])

Golub and Van Loan[4] (p. 493) derives the Conjugate Gradient Method. Later in this paper, we will examine Golub and Van Loan's algorithm for the PCG method.

Conjugate Gradient Algorithm

Algorithm 9.3.1 *If* $A \in \Re^{n \times n}$ *is symmetric positive definite,* $b \in \Re^n$, *and* $x_0 \in \Re^n$ *is an initial guess* ($Ax_0 \approx b$), *then this algorithm computes the solution to* $Ax=b$ (Golub and Van Loan[4] p. 493).

$r_0 = b - Ax_0$
$\beta_0 = \|r_o\|_2$
$q_0 = 0$
$k = 0$
while $\beta_k \neq 0$

$q_{k+1} = r_k / \beta_k$
$k = k + 1$
$\alpha_k - q_k^T A q_k$
$r_k = (A - \alpha_k I)\, q_k - \beta_{k-1} q_{k-1}$
$\beta_k = \|r_k\|_2$
if $k=1$

$d_1 = \alpha_1$
$c_1 = q_1$
$\rho_1 = \beta_0 / \alpha_1$
$x_1 = p_1 q_1$

else

$\mu_{k-1} = \beta_{k-1} / d_{k-1}$
$d_k = \alpha_k - \beta_{k-1} \mu_{k-1}$
$c_k = q_k - \mu_{k-1} c_{k-1}$
$\rho_k = -\mu_{k-1} d_{k-1} \rho_{k-1} / d_k$
$x_k = x_{k-1} + \rho_k c_k$

end

end

$x = x_k$

Golub and Van Loan[4] (p. 490 ff) shows that Algorithm 9.3.1 converges quickly if the condition number is small (i.e., of the same order as the condition number of the identity

matrix). In most situations, the condition number is rather large and algorithm 9.3.1 will not perform efficiently.

Preconditioned Conjugate Gradient Method

If we can solve a equivalent problem, $\widetilde{A}x = \widetilde{b}$, with the same solution x, such that the condition number of $\widetilde{A}$ is smaller than the condition number of A, then we can accelerate the convergence rate of the conjugate gradient method.

A well know method of transforming A such that $\widetilde{A}$ has a smaller condition number is by preconditioning the matrix A by another Matrix M^{-1} so that the condition number of $M^{-1}A = \widetilde{A}$ is smaller than the condition number of A.

If we left multiply Ax = b by a Matrix M^{-1} then: Ax = b => M^{-1} (Ax) = M^{-1} (b). For this method to be efficient, M^{-1} (or M) must be chosen so that 1) the matrix multiplication $M^{-1}A$ is also efficient, and 2) the condition number of $M^{-1}A$ ($= \widetilde{A}$) is smaller than the condition number of A. Additionally, Matrix M^{-1} must be chosen so that $M^{-1}A$ ($= \widetilde{A}$) is also SPD, a requirement of both the CG and PCG methods.

The following algorithm by Golub and Van Loan[4] (p. 532 ff) shows the method of **Preconditioned Conjugate Gradients.** This is the algorithm used in this project to obtain the results presented.

> ***Algorithm 10.3.1 [Preconditioned Conjugate Gradients]*** *Given a symmetric positive definite Matrix* $A \in \Re^{n \times n}$, $b \in \Re^n$, *a symmetric positive definite preconditioner M, and an initial guess* ($Ax_0 \approx b$), *the following algorithm solves the linear system Ax=b* (Golub and Van Loan[4] p. 534).

$k = 0$
$r_0 = b - Ax_0$
while $(r_k \neq 0)$

Solve $Mz_k = r_k$
$k = k + 1$
if $k = 1$

$p_1 = z_0$

else

$\beta_k = r_{k-1}^T z_{k-1} / r_{k-2}^T z_{k-2}$
$p_k = z_{k-1} + \beta_k p_{k-1}$

end

$\alpha_k = r_{k-1}^T z_{k-1} / p_k^T A p_k$
$x_k = x_{k-1} + \alpha_k p_k$
$r_k = r_{k-1} - \alpha_k A p_k$

end

$x = x_k$

The PCG Method converges with a maximum complexity of $O(n^3)$ (Golub and Van Loan[4] p. 532) where n is the size of the matrix A. The convergence rate is also dependent upon the convergence tolerance criteria. For this project, we used a tolerance of 10^{-12} for 5 of the Matrices and 10^{-6} for the large matrix S3DKQ4M2. This resulted in a calculated complexity estimate of approximately $O(n^3/5)$ (based on total iteration count divided by the matrix size n^3).

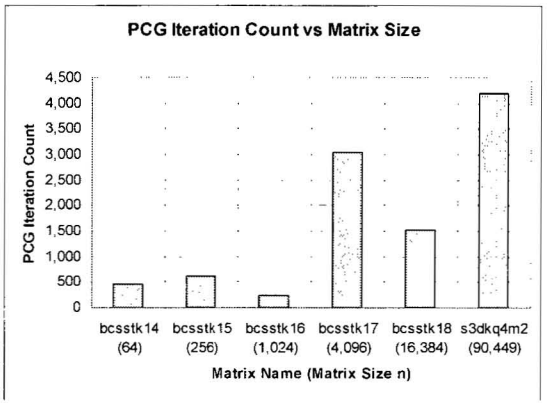

Figure 6) P.C.G. Iteration Count Vs Matrix Size

Preconditioning of the Conjugate Gradient method is a method whereby the convergence of the solution is accelerated by pre (or post) multiplying the matrix by another matrix so that the resulting systems of equations are computationally less expensive to solve. There are many efficient (and sometimes complicated) methods to precondition a matrix for PCG including pre-multiplying by a diagonal Matrix (inexpensive computationally); incomplete LU (Gaussian Elimination) or Cholesky decomposition; and, multi-grid methods. This project examines the diagonal matrix preconditioning method known as diagonal scaling.

Diagonal Scaled Preconditioned Conjugate Gradient

Preconditioning the Matrix A by a diagonal Matrix prior to the conjugate gradient method accelerates the convergence of the method to the final solution. For example, Greenbaum[3] (pp 165-168) shows that the diagonal of the original Matrix A is close to the optimal diagonal preconditioner with respect to the condition number of $M^{-1}A$.

The original system of equations: $\mathbf{Ax} = \mathbf{b}$ is transformed into the equivalent system: $\mathbf{M^{-1}Ax} = \mathbf{M^{-1}b}$ which has the same solution ($\mathbf{x}$) as the original equation.

Selecting M = diag(A) leads to a *nearly* optimal diagonal preconditioner (Greenbaum[3] pp. 165-166).

Returning to our original sample problem, we can see the contours of the gradient for the unconditioned and preconditioned system of equations.

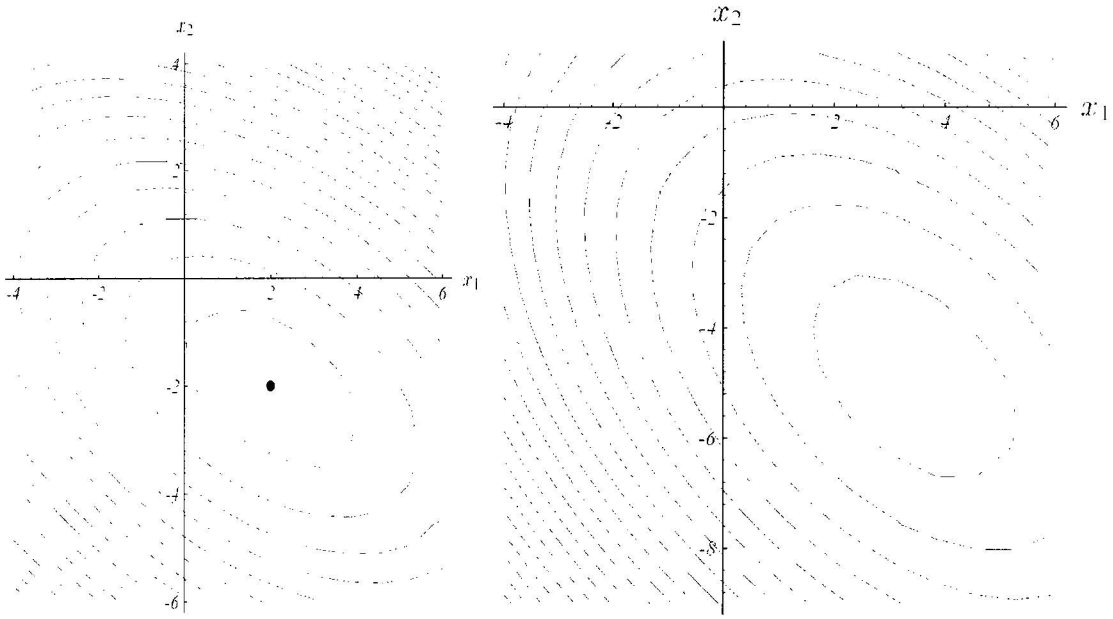

Figure 7) Unconditioned Contour (left) and Preconditioned (right).
Note the more "roundness" of the contours in the preconditioned gradients (Shewchuk, J.[8]).

The factors that need to be evaluated when choosing a preconditioner for the PCG method are:

1. Computationally efficient to compute M^{-1}
2. Computationally efficient to solve systems involving the preconditioning Matrix M^{-1}

Both items above are important to maintain the efficiency of the overall algorithm.

For more detailed and extensive explanations of the Conjugate Gradient method and the PCG method in particular, see Shewchuk, J.[8] and Golub and Van Loan[4] (pp 520-528).

Condition Number of the Matrix

Obtaining an accurate solution to a system of equations with any method, either direct or iterative, depends upon the Condition Number of a matrix with respect to Matrix Inversion.

The condition number of the matrix, with respect to Inversion, is defined as:

$$\kappa = \|A\| \cdot \|A^{-1}\|$$

where ||A|| denotes a norm of A. Estimating the condition number can be very important since it can tell us how accurate a solution to the system of equations we may compute. For example, if $\kappa = 10^7$ then we potentially lose 7 significant digits in the solution x. Computers, which have finite precision, have approximately 15 decimal digits of precision for double precision variables. Therefore, if the condition number, $\kappa = 10^7$ then the best we may expect in a solution is about one half of the 15 significant digits available with double precision. Golub and Van Loan[4], describe an efficient algorithm to estimate the condition number κ of a Matrix (pp 128-130).

Figure 8 shows that the condition number tends to increase with matrix size. Therefore, we must be more careful when dealing with large matrices since the condition number may become large as the matrix size increases.

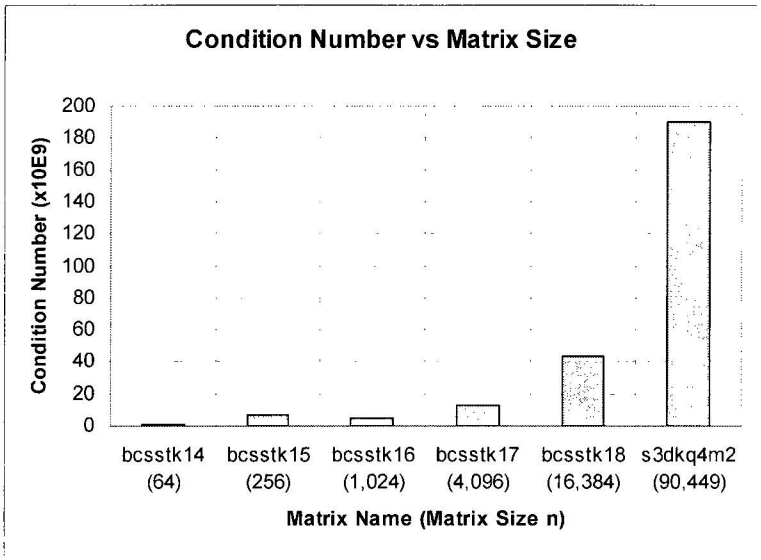

Figure 8) Condition Number vs Matrix Size

An example of a sparse matrix is shown below for the case of a 4 node by 4 node FEM discretization. Note that most of the elements are zero with the non-zero elements clustered about the diagonal. Also note the symmetry of the matrix and that the largest element for each row and column is also on the diagonal.

17	-0.5	0	0	-0.5	0	0	0	0	0	0	0	0	0	0	0
-0.5	**18**	-0.5	0	0	-1	0	0	0	0	0	0	0	0	0	0
0	-0.5	**18**	-0.5	0	0	-1	0	0	0	0	0	0	0	0	0
0	0	-0.5	**17**	0	0	0	-0.5	0	0	0	0	0	0	0	0
-0.5	0	0	0	**18**	-1	0	0	-0.5	0	0	0	0	0	0	0
0	-1	0	0	-1	**4**	-1	0	0	-1	0	0	0	0	0	0
0	0	-1	0	0	-1	**4**	-1	0	0	-1	0	0	0	0	0
0	0	0	-0.5	0	0	-1	**18**	0	0	0	-0.5	0	0	0	0
0	0	0	0	-0.5	0	0	0	**18**	-1	0	0	-0.5	0	0	0
0	0	0	0	0	-1	0	0	-1	**4**	-1	0	0	-1	0	0
0	0	0	0	0	0	-1	0	0	-1	**4**	-1	0	0	-1	0
0	0	0	0	0	0	0	-0.5	0	0	-1	**18**	0	0	0	-0.5
0	0	0	0	0	0	0	0	-0.5	0	0	0	**17**	-0.5	0	0
0	0	0	0	0	0	0	0	0	-1	0	0	-0.5	**18**	-0.5	0
0	0	0	0	0	0	0	0	0	0	-1	0	0	-0.5	**18**	-0.5
0	0	0	0	0	0	0	0	0	0	0	-0.5	0	0	-0.5	**17**

Figure 9) Sample Matrix for n_x=n_y=16

Evolutionary Algorithm for Diagonally Scaled Preconditioned Conjugate Gradient

Integrating the EA into the DSPCG Algorithm is outlined in figure 10 below. The reproduction, crossover and mutation operators are employed to generate a new population. The new population will include individuals with fitness within the top 50%. In the case of PCG, the fitness will be the relative error of the current individual, with smaller values indicating better fitness. The relative error for iteration i is $||b - Ax_i||_2/||Ax_i||_2$ where b is the right hand side vector (constant), A is the Matrix (constant) and x_i is the current calculated solution (varies at each iteration). At convergence, the relative error is less than the specified tolerance.

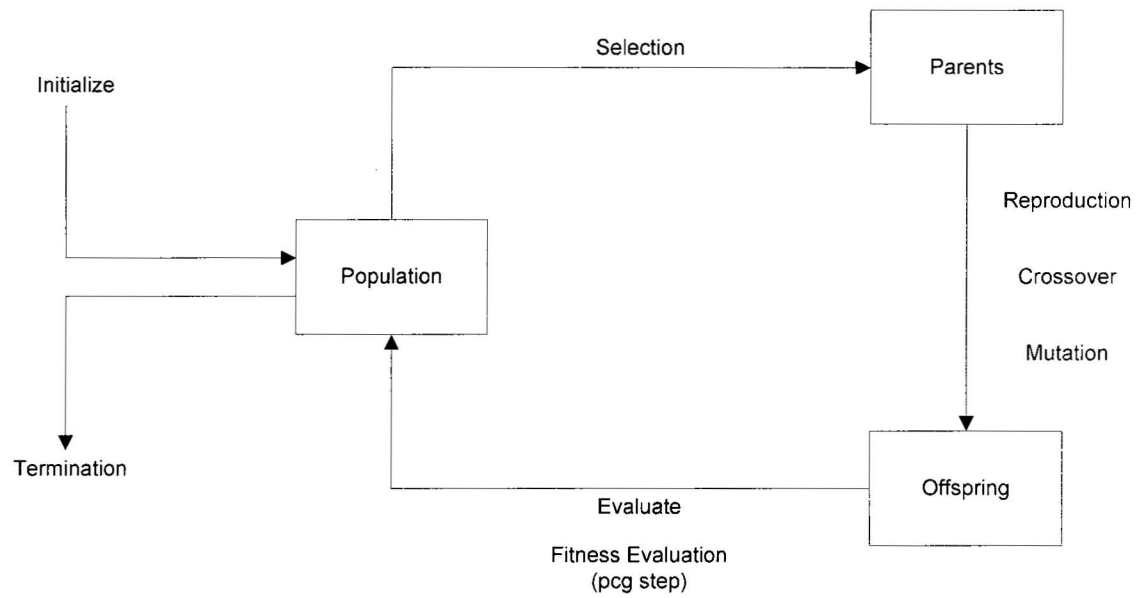

Figure 10) Evolutionary Algorithm
Start with initialization, compute one or more loops and then terminate under the specified termination condition(s).

EAs mimic biological evolution to find computer solutions to physical problems (generally optimization problems). Figure 10 shows the pertinent steps in an EA as applied to the PCG Method.

Individuals: A single item in a population. In biological evolution, this is the specimen being studied (fruit fly, human, etc.). In this project, an individual is a vector of real numbers representing the preconditioning diagonal Matrix M.

Initialize: Here we set up the Matrices and vectors, as well as the various program parameters (stopping tolerance, population size, crossover and mutation operators, etc.).

Population: Indicates the set of individuals for the next step of the EA.

Selection: Selects the top n individuals based on the fitness criteria (Evaluation) from the population.

Parents: The top n individuals selected during the Selection process are the bases for the Reproduction stage.

Reproduction: The individuals in the population reproduce n offspring using crossover and mutation operators.

Crossover: An evolutionary operator that takes portions of the parents' chromosomes to produce an offspring individual. For this PCG method, portions of the solution vector x are selected from each of the two parents.

Mutation: An evolutionary operator whereby an individual is randomly changed. For this project, components of the solution vector x are selected randomly and changed by a random amount.

Offspring: Individuals that are added to the population as a result of reproduction. The offspring individuals are created by crossover and (possibly) mutation.

Evaluation: Individuals are "tested" or evaluated based on fitness criteria and given a score. For this project, the score is the relative residual ($||b - Ax_i||_2/||Ax_i||_2$) with lower values indicating higher fitness.

Termination: Stopping criteria for the Evolution Algorithm. This is usually a maximum number of cycles in the process. For this project, two criteria were used: 1) a maximum number of iterations and 2) a tolerance value (10^{-12} or 10^{-6}) for the relative residual.

Since the overall complexity of the algorithm is critical for Matrix equation solvers, the number of computations in each step is critical. Furthermore, the complexity of each step of the EA is therefore critical for linear systems of equation solvers. If we assume that the conventional PCG method converges in k steps, where k < size of the Matrix (Golub and Van Loan[4] p. 522), and we also assume also that the Fitness Evaluation is of the same complexity as one iteration step (exactly the same complexity in this instance), then, in order to "break even", the EA must determine a solution in one half the number of iterations. This would mean a huge time-saving with respect to Matrix equation solvers. The fitness evaluation is the most critical step for this EA.

Evolutionary Algorithm Representation

The Matrix M=diag(A) is the variable in this algorithm that was evolved. The EA representation that was used was a real number vector representing the diagonal of the preconditioning matrix M (stored as a one dimensional array):

$$M = D = \begin{bmatrix} \delta_1 \\ \delta_2 \\ \vdots \\ \delta_n \end{bmatrix}$$

Crossover Operator

A single point crossover operator was used. Two parents combine to generate two offspring during each generation.

Two parents d_1 and d_2

$$d_1 : \alpha_1, \alpha_2, \alpha_3, \ldots \alpha_n$$
$$d_2 : \xi_1, \xi_2, \xi_3, \ldots \xi_n$$

reproduce two offspring d_3 and d_4 through crossover operation

$$d_3 : \alpha_1, \alpha_2, \xi_3, \ldots \xi_n$$
$$d_4 : \xi_1, \xi_2, \alpha_3, \ldots \alpha_n$$

Mutation Operator

A Normally Distributed (Gaussian) operator was used as follows:

$$d_{i+1}^{(k+M)} = d_i^{(k+C)} + g_i^{(k+M)}, \qquad i = 1 \ldots k$$

Where d and g are n vectors, and g is normally distributed with mean 0 and standard deviation ranging from 0.01% to 10%. The superscript M above represents the Mutation operation while the C represents the Crossover operation.

Fitness Evaluation

The choice for fitness evaluation was the relative residual of the solution vector. The relative residual r is a measure of how far the current individual is away from the actual solution.

$$r = ||b-Ax||/||Ax||$$

Individuals with lower relative residuals have a higher fitness than those individuals with higher relative residuals. The relative residual for each individual in the population is determined during each generation.

Selection and Reproduction

Parents and offspring compete and the best m/2 individuals of $d^{(k+m)}$ reproduce. The individuals are then ranked based on their relative residuals, with the top half surviving through to the next generation.

Testing Setup

Six Matrices from the Matrix Market[10] were tested ranging from a 1,806 x 1,806 matrix BCSSTK14 up to the 90,449 x 90,449 matrix S3DKQ4M2. Table 1 (repeated below) shows the basic properties of each of the matrix tested. PCG is very dependant upon the condition number with higher condition numbers indicating slower convergence.

Table 2: Test Matrices Properties (repeated)

Name	Size	Number Non-Zero Elements	% Non-Zero	Condition Number x 10^9 (Est.)	Notes
BCSSTK14	1,806	63,454	1.945	1.3	Roof of the Omni Coliseum, Atlanta
BCSSTK15	3,948	117,816	0.756	6.5	Module of an offshore platform
BCSSTK16	4,884	290,378	1.217	4.9	U.S. Army Corps of Engineers dam
BCSSTK17	10,974	428,650	0.356	13.0	Elevated pressure vessel
BCSSTK18	11,948	149,090	0.104	43.0	R.E. Ginna Nuclear Power Station
S3DKQ4M2	90,449	2,455,670	0.030	190.0	Finite element analysis of cylindrical shells

In order to test the applicability of EA for this problem and to eliminate the possibility of random solutions occurring during the evolution, the fastest converging matrix from the above list (BCSSTK16) was tested with random preconditioning matrices. The test consisted of running the program 100,000 times with a different random preconditioning matrix. To allow for quicker calculation, the standard PCG was first executed and the iteration count recorded (243 for this instance). For the test with 100,000 random vectors, if the iteration count exceeded the count for standard DSPCG (243), the test was stopped for that individual. The test completed for 100,000 random vectors without any individuals performing better than the conventional PCG. Therefore, we can conclude

that this problem is in fact applicable to EA, and a random test does not perform better than we can expect an EA to perform.

Program Parameters

To facilitate easy changes to the EA test program, a parameter file was used. Following is a brief discussion of each of the program parameters.

Sample parameter file (Filename = eadspcg.parameters)

```
# filename max_iters tol xoverrate mutationrate maxgenerations matrixsamplesize
s3dkq4m2        filename
99999           max_iters
10d-12          tol
95              xoverrate percentage
5               mutationrate percentage
100             mutationrate mean as a percentage of Xi (should be set to 100)
0.1             mutationrate standard deviation as a percentage of Xi
100             maxgenerations (set to 0 for regular DSPCG)
1               matrixsamplesize percentage
```

The first line is a header line and is not used within the program. Each of the lines following consists of a value and a text description of the parameter for the user. The portion of the line after the first value is discarded by the program. Additionally, the order of the parameters is set and cannot be changed without changing the code.

The second line is the filename for the input Matrix (s3dkq4m2 filename). This file must be located in a subdirectory below the current directory named **data**.

The third parameter (99999 max_iters) is the maximum number of PCG iterations before the program stops. This should be at MOST the dimension of the Matrix.

The fourth parameter (10d-12 tol) is the PCG tolerance that will indicate that the PCG iteration has converged.

The fifth parameter (95 xoverrate percentage) is the EA crossover rate expressed as a percentage. A 95 would indicate that that 95 percent of the time, crossover will occur. For this test, the two values used for this parameter were **95** and **50**.

The sixth parameter (5 mutationrate percentage) is the EA mutation rate expressed as a percentage. A 5 would indicate that 5 percent of the time, mutation will occur. For this test, the two values used for this parameter were **5** and **10.**

The seventh parameter (100 mutation rate mean as a percentage of X_i) is the Gaussian mean of the Diagonal Matrix elements that will be used within the mutation operator. (e.g. if M(9) = 123, and the mutationrate mean is 100, then the Gaussian mean would be 123. The value should normally be set to 100.

The eighth parameter (0.1 mutationrate standard deviation as a percentage of X_i) is the Gaussian standard deviation of the Diagonal Matrix that will be used within the mutation operator. (e.g. if M(9) = 40, and the mutationrate mean is 0.1, then the mutated M(9) element would be within the range [39.94 and 40.04] for one standard deviation. For this test, the two values used for this parameter were **1** and **0.1.**

The ninth parameter (100 maxgenerations (set to 0 for regular DSPCG)) is the maximum number of generations that the EA will execute. After that number is reached, the algorithm continues using conventional DSPCG with no EA. Setting this value to 0 will make the program perform conventional DSPCG.

The tenth parameter (1 matrixsamplesize percentage) is not yet implemented. The intention of this parameter is to indicate the percentage of the Matrix to use during each generation. If the matrix has 1000 rows, and the matrixsamplesize is 1, then 10 rows would be used to perform the fitness evaluation.

Finally, the population size for this test was set at 4 with the top 2 individuals surviving each generation.

Results and Conclusions

Timings and results were conducted on a Dual Core AMD64 computer running at 2.0 GHz and with 3GB of RAM. The processors each have 512MB of L2 cache. The tolerance used for these test was 10^{-12}, except for Matrix S3DKQ4M2 which used 10^{-6}.

Table 3: Results

Matrix Name	Matrix Size	Toler-ance	DSPCG Iteration Count	EA Iteration Counts	DSPCG time (sec)	EA time (sec)
BCSSTK14	1,806	10^{-12}	456	473 – 540	0.243	0.45 – 0.52
BCSSTK15	3,948	10^{-12}	626	749 – 946	0.728	1.31 – 1.61
BCSSTK16	4,884	10^{-12}	243	309 – 342	0.586	1.55 - 1.70
BCSSTK17	10,974	10^{-12}	3,044	3,419 - 3,807	7.85	9.61 - 11.0
BCSSTK18	11,948	10^{-12}	1,522	1,602 - 1,642	2.80	3.44 - 4.41
S3DKQ4M2	90,449	10^{-6}	4,201	4,246 – 4,273	109	120 – 122

Overhead associated with the EA will account for some of differences for the EA Iteration Time. However, the DSPCG method accounts for the differences in the EA Iteration Counts and the DSPCG Iteration counts. For the large Matrix S3DKQ4M2, a tolerance of 10^{-6} was used to keep the iteration count low so that the effects of the EA would be more visible.

The condition number of the original Matrix A is critical to the convergence of the PCG method. In general, the higher the condition number, the longer PCG will take to converge to a solution. Additionally, the condition number of a Matrix generally increases with the size of the matrix. Both these points will explain the time taken for S3DKQ4M2 as well as explaining the lower convergence rates and elapsed times for all other Matrices.

Unfortunately, the DSPCG did not accelerate the convergence rate or convergence time for time any of the matrices in questions. Table 3 shows that the iteration count increased in each instance and sometimes by a noticeable amount. Additionally, the convergence time increased with each of the matrices for the DSPCG.

The figures on the following pages (Figures 11 – 16) show the convergence of the EA for the DSPCG. Each of the different traces on the figures indicates different values of the EA Parameters. The legend on the figures indicates the values of the Crossover Rate, the Mutation Rate as well as the Mutation standard Deviation. For example, 50-10-1 indicates a 50% crossover rate, a 10% mutation rate and a 1% mutation standard deviation.

Eight tests were conducted for each of the six matrices listed in table 3. The following table lists the values of the EA Parameters that were used in each test.

Table 4:Evolutionary Algorithm Parameter Values used During Testing

Test Number	Crossover Rate	Mutation Rate	Mutation Standard Deviation
1	95%	5%	0.1%
2	95%	5%	1%
3	95%	10%	0.1%
4	95%	10%	1%
5	50%	5%	0.1%
6	50%	5%	1%
7	50%	10%	0.1%
8	50%	10%	1%

The graphs in Figures 17 through 26 show the iteration convergence rate as well as the time for convergence for each of the matrices. In each case, both the convergence rate and convergence time was worse for the EA than for conventional DSPCG.

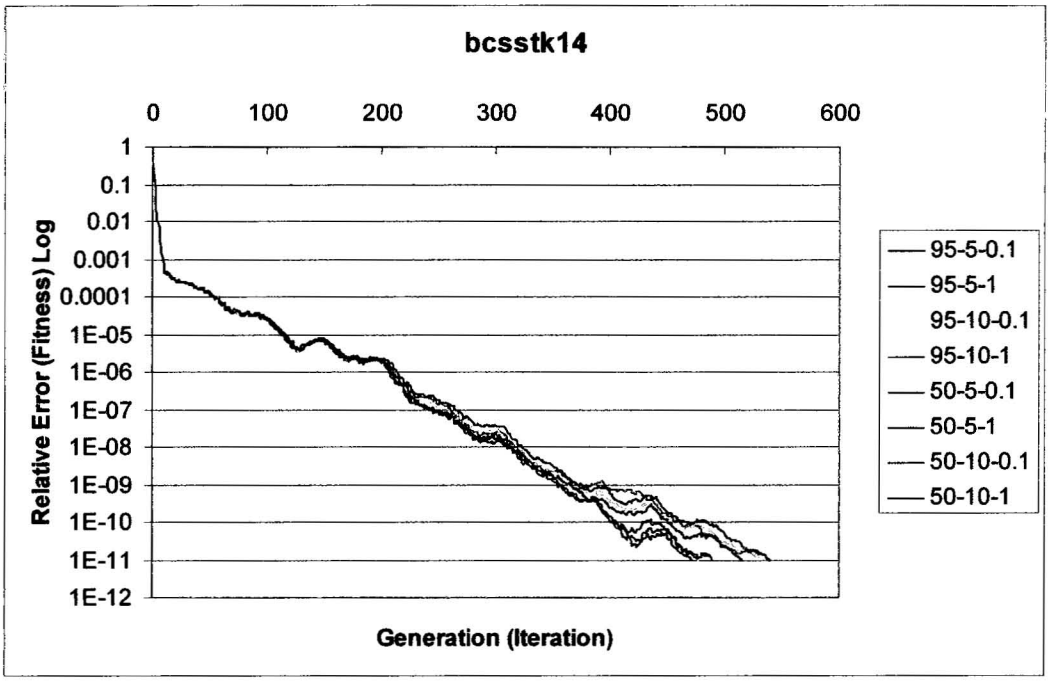

Figure 11) Matrix BCSSTK14* Convergence Rate, Tolerance = 10^{-12}

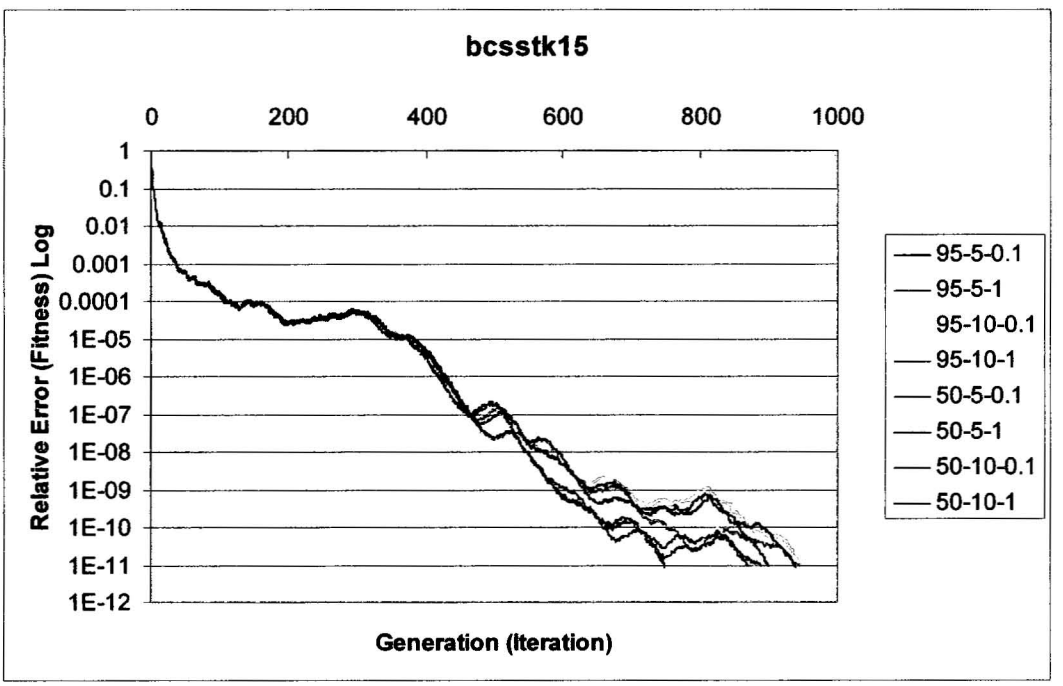

Figure 12) Matrix BCSSTK15* Convergence Rate, Tolerance = $10e^{-12}$

* The legend indicates the crossover rate, mutation rate and mutation rate standard deviation EA parameters. For example, 95-05-0.1 indicates a 95% crossover rate, a 5% mutation rate and a 0.1% mutation standard deviation.

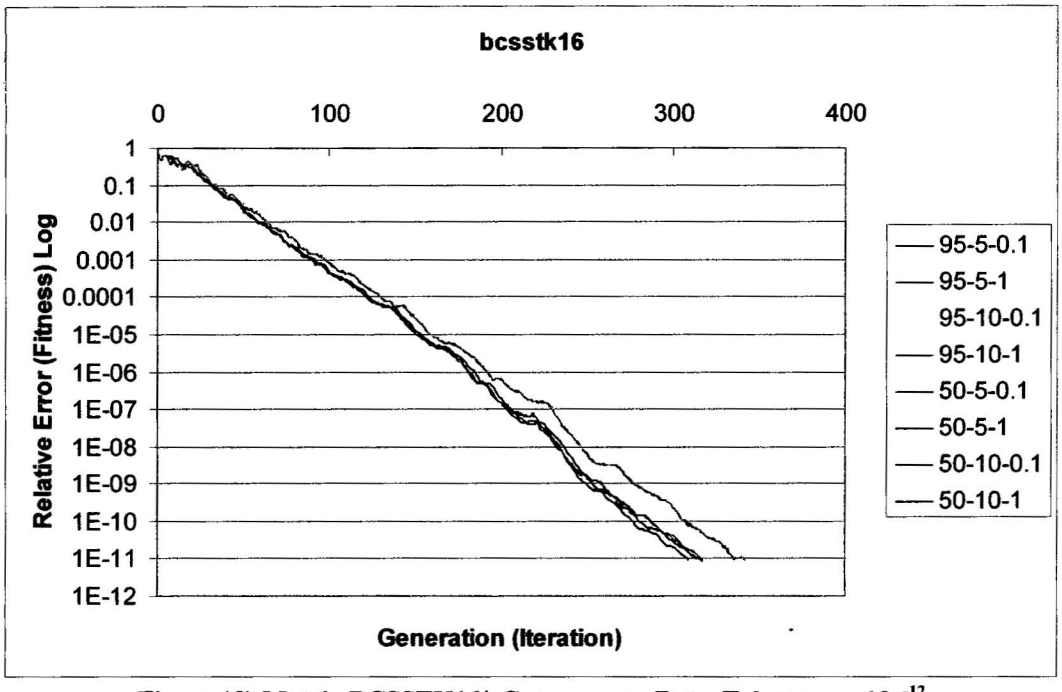

Figure 13) Matrix BCSSTK16* Convergence Rate, Tolerance = $10e^{-12}$

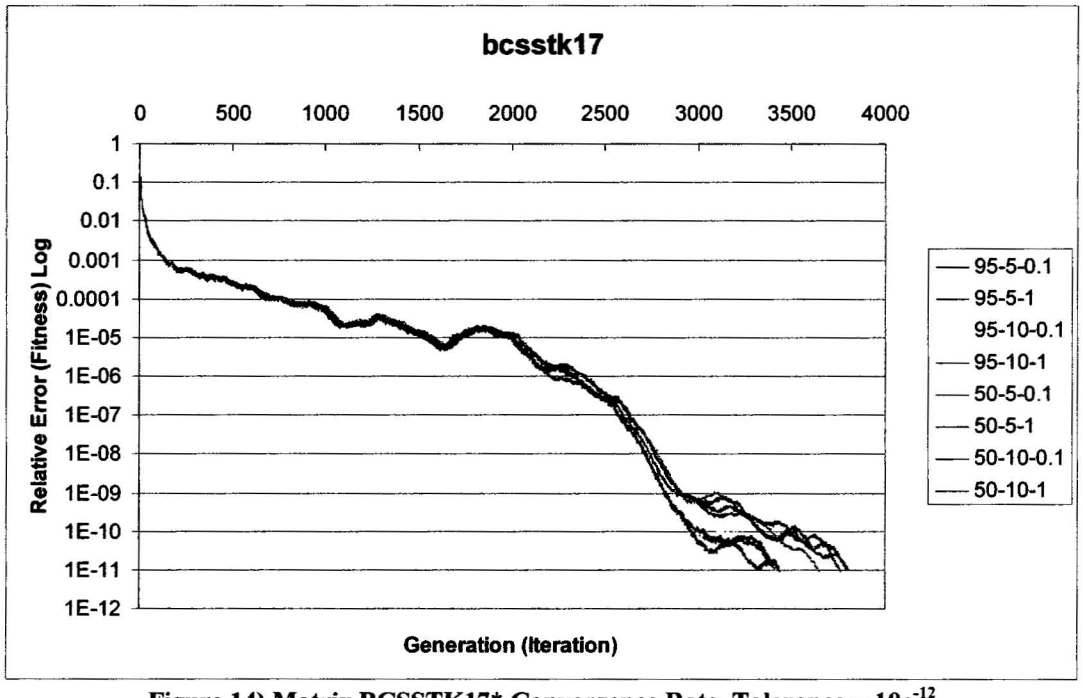

Figure 14) Matrix BCSSTK17* Convergence Rate, Tolerance = $10e^{-12}$

* The legend indicates the crossover rate, mutation rate and mutation rate standard deviation EA parameters. For example, 95-05-0.1 indicates a 95% crossover rate, a 5% mutation rate and a 0.1% mutation standard deviation.

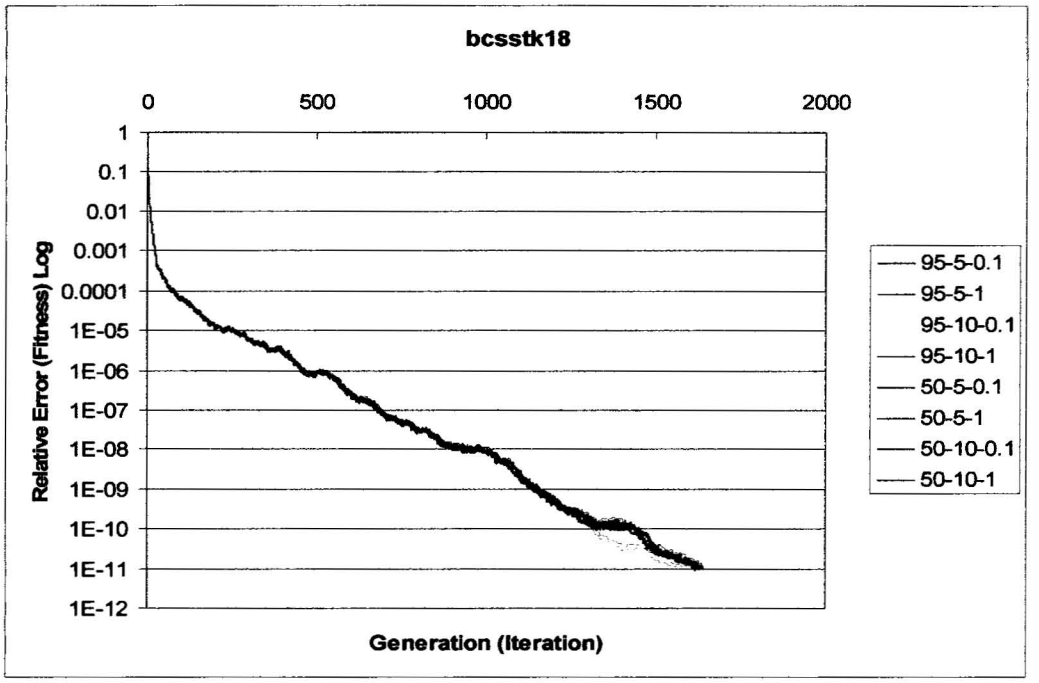

Figure 15) Matrix BCSSTK18* Convergence Rate, Tolerance = $10e^{-12}$

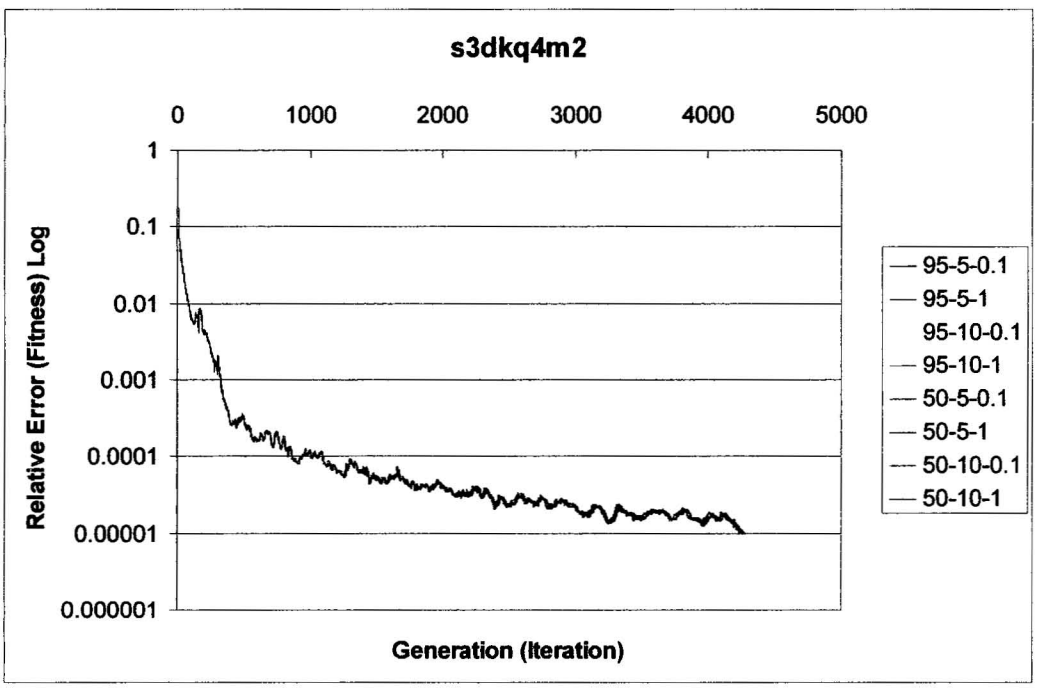

Figure 16) Matrix S3DKQ4M2* Convergence Rate, Tolerance = $10e^{-6}$

* The legend indicates the crossover rate, mutation rate and mutation rate standard deviation EA parameters. For example, 95-05-0.1 indicates a 95% crossover rate, a 5% mutation rate and a 0.1% mutation standard deviation.

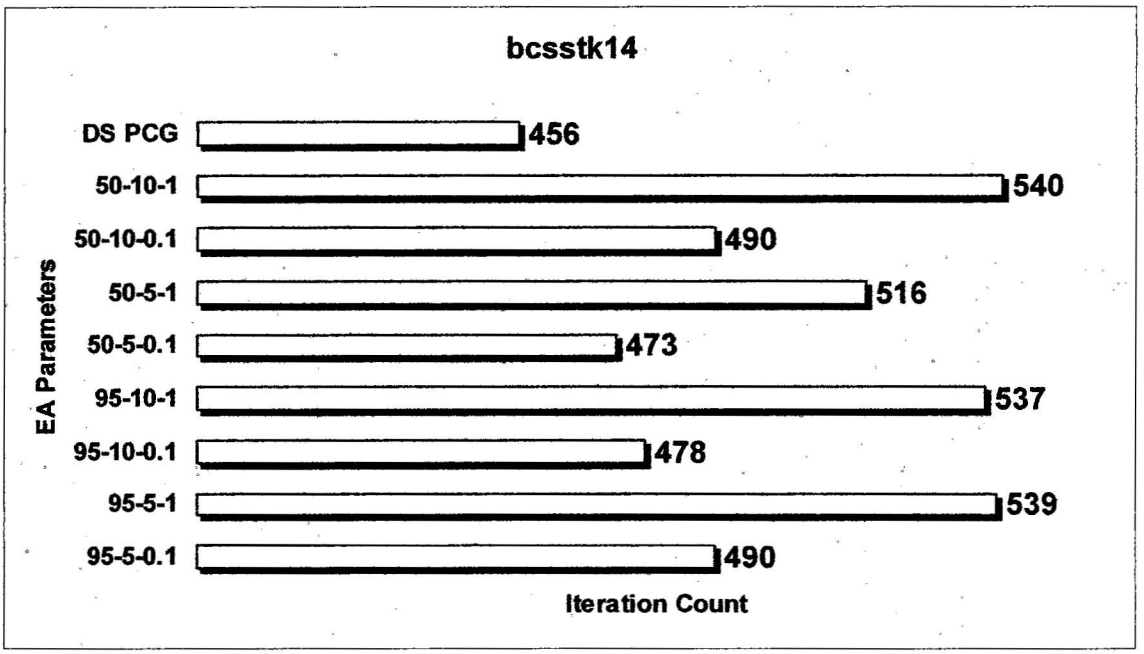

Figure 17) Matrix BCSSTK14*, Iteration Count, Tolerance = $10e^{-12}$

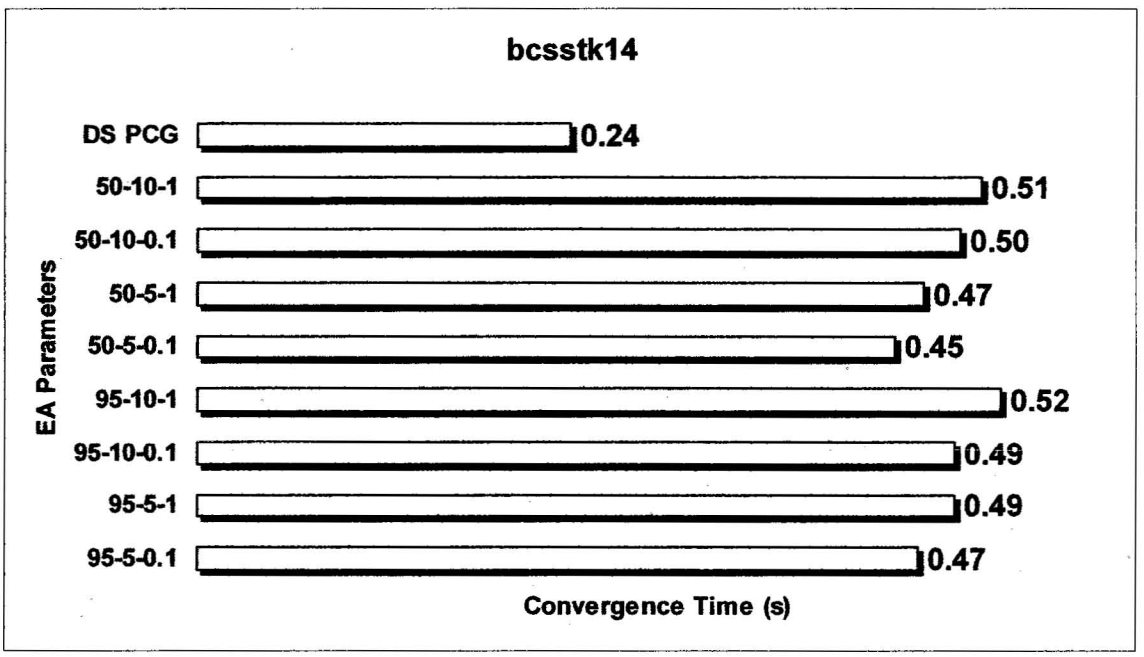

Figure 18) Matrix BCSSTK14*, Convergence Time (sec.), Tolerance = $10e^{-12}$

* The legend indicates the crossover rate, mutation rate and mutation rate standard deviation EA parameters. For example, 95-05-0.1 indicates a 95% crossover rate, a 5% mutation rate and a 0.1% mutation standard deviation.

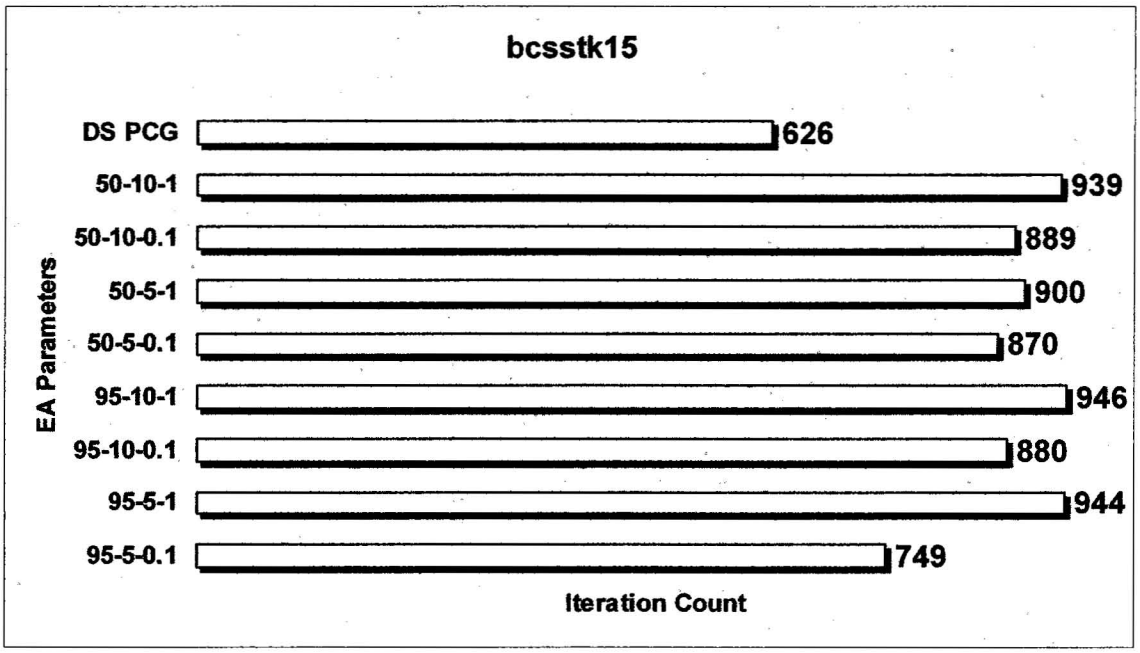

Figure 19) Matrix BCSSTK15*, Iteration Count, Tolerance = $10e^{-12}$

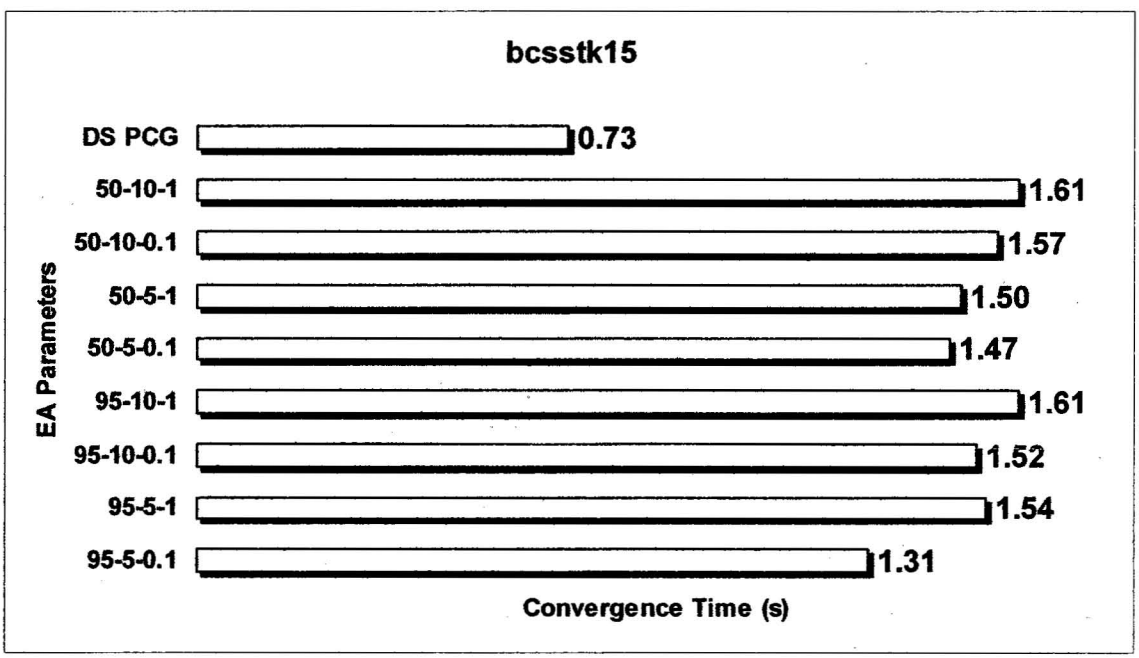

Figure 20) Matrix BCSSTK15*, Convergence Time (sec.), Tolerance = $10e^{-12}$

* The legend indicates the crossover rate, mutation rate and mutation rate standard deviation EA parameters. For example, 95-05-0.1 indicates a 95% crossover rate, a 5% mutation rate and a 0.1% mutation standard deviation.

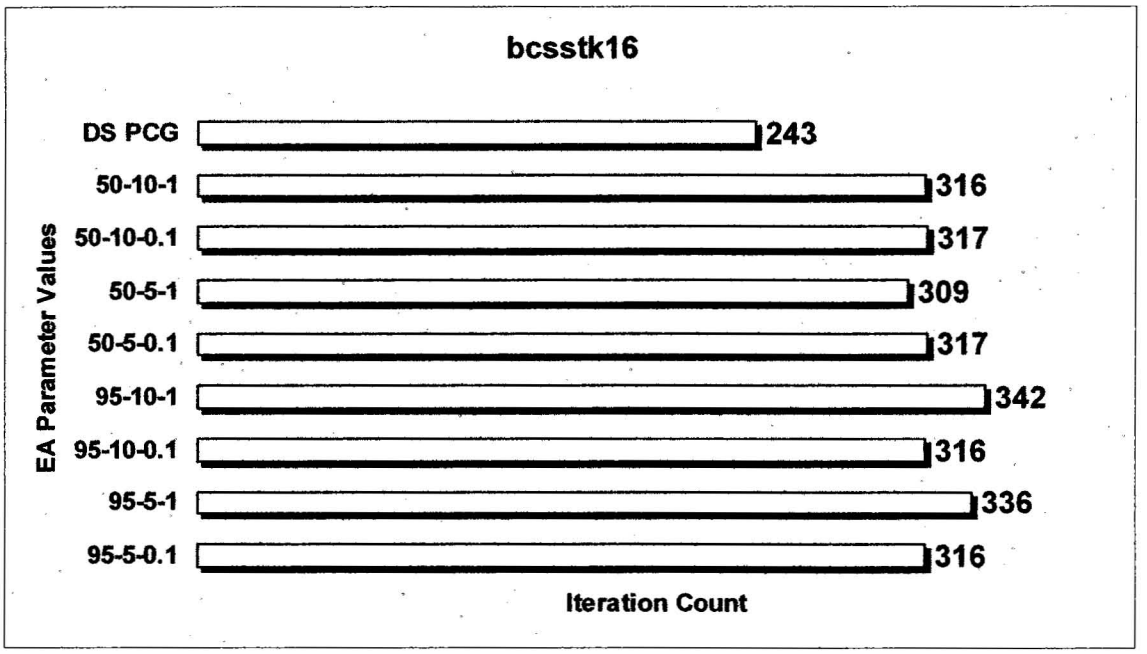

Figure 21) Matrix BCSSTK16*, Iteration Count, Tolerance = $10e^{-12}$

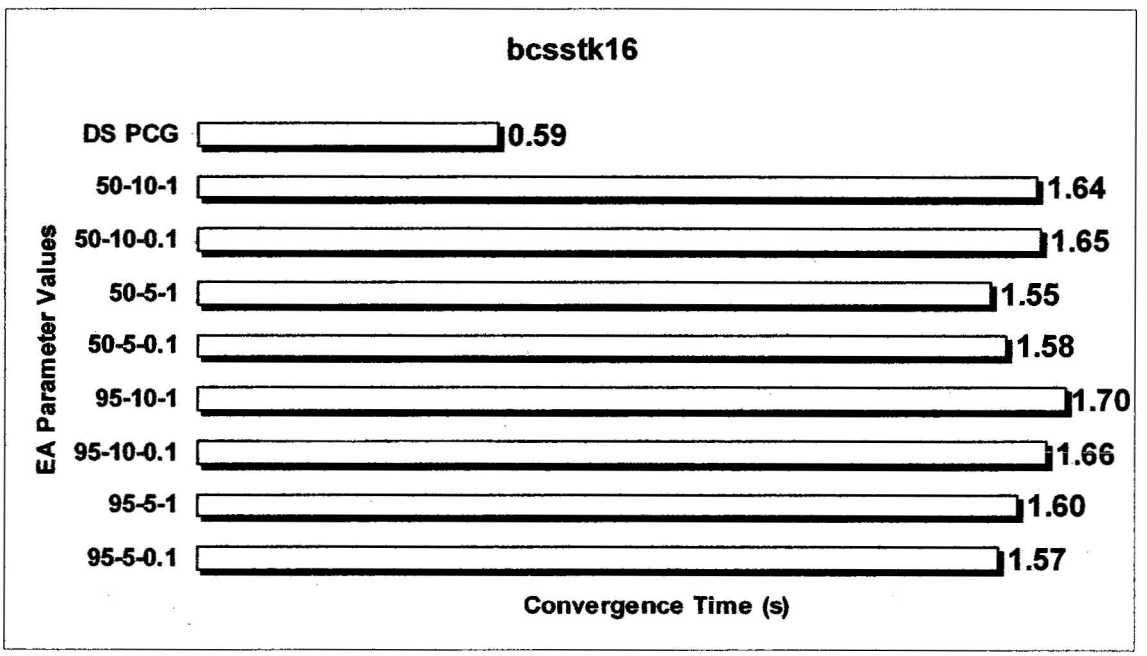

Figure 22) Matrix BCSSTK16*, Convergence Time (sec.), Tolerance = $10e^{-12}$

* The legend indicates the crossover rate, mutation rate and mutation rate standard deviation EA parameters. For example, 95-05-0.1 indicates a 95% crossover rate, a 5% mutation rate and a 0.1% mutation standard deviation.

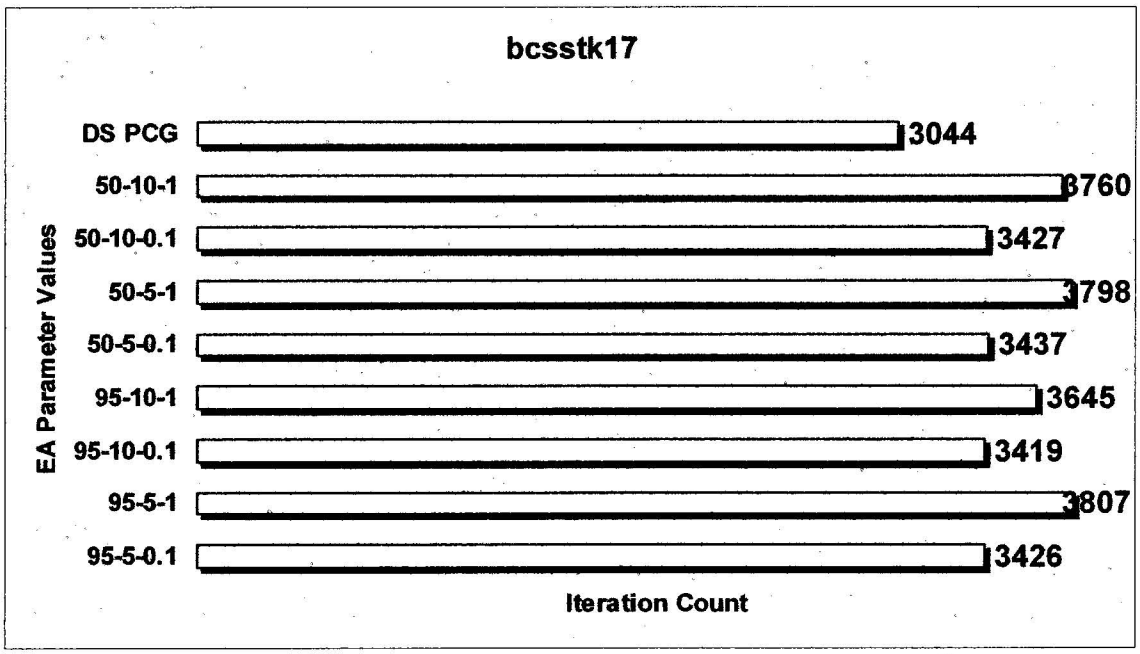

Figure 23) Matrix BCSSTK17*, Iteration Count, Tolerance = $10e^{-12}$

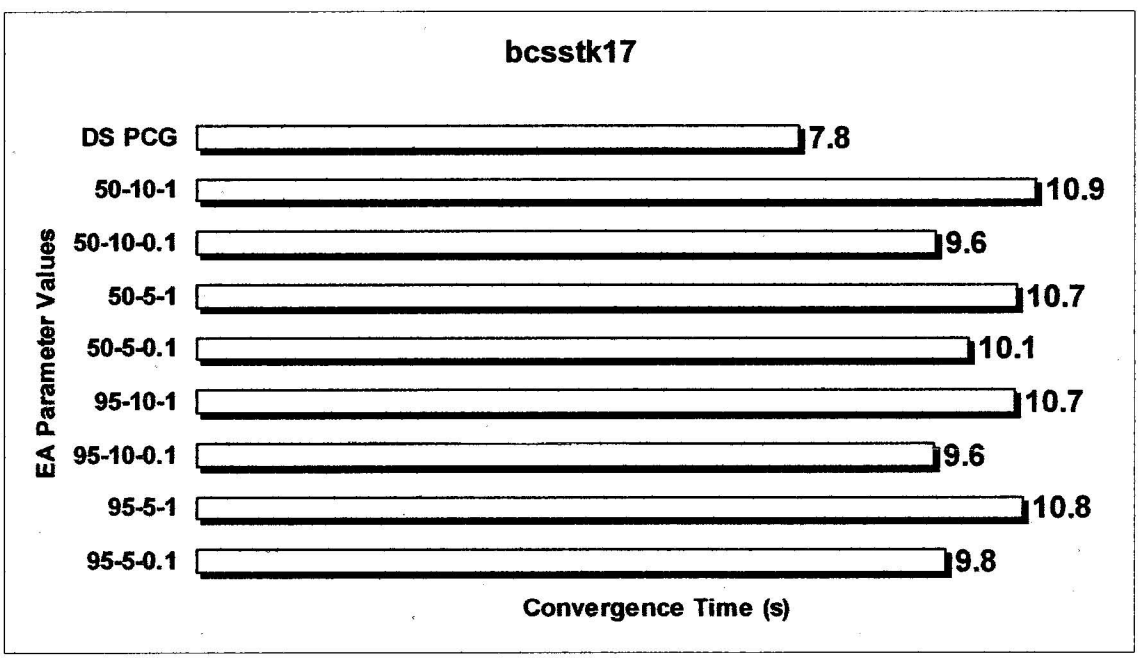

Figure 24) Matrix BCSSTK17*, Convergence Time (sec.), Tolerance = $10e^{-12}$

* The legend indicates the crossover rate, mutation rate and mutation rate standard deviation EA parameters. For example, 95-05-0.1 indicates a 95% crossover rate, a 5% mutation rate and a 0.1% mutation standard deviation.

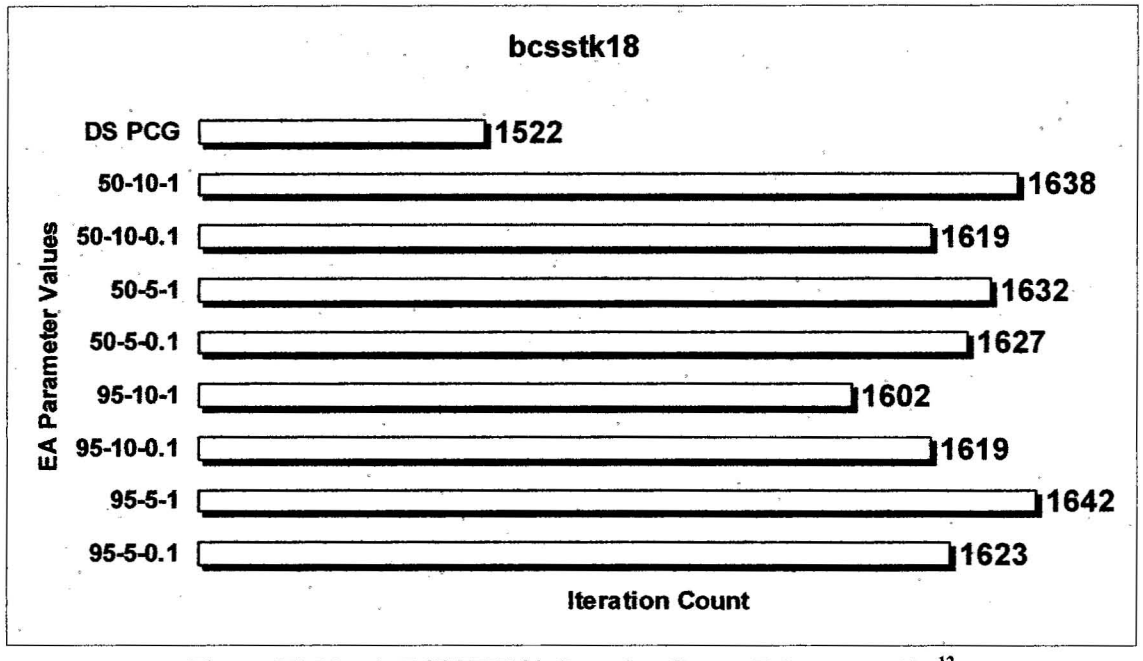

Figure 25) Matrix BCSSTK18*, Iteration Count, Tolerance = $10e^{-12}$

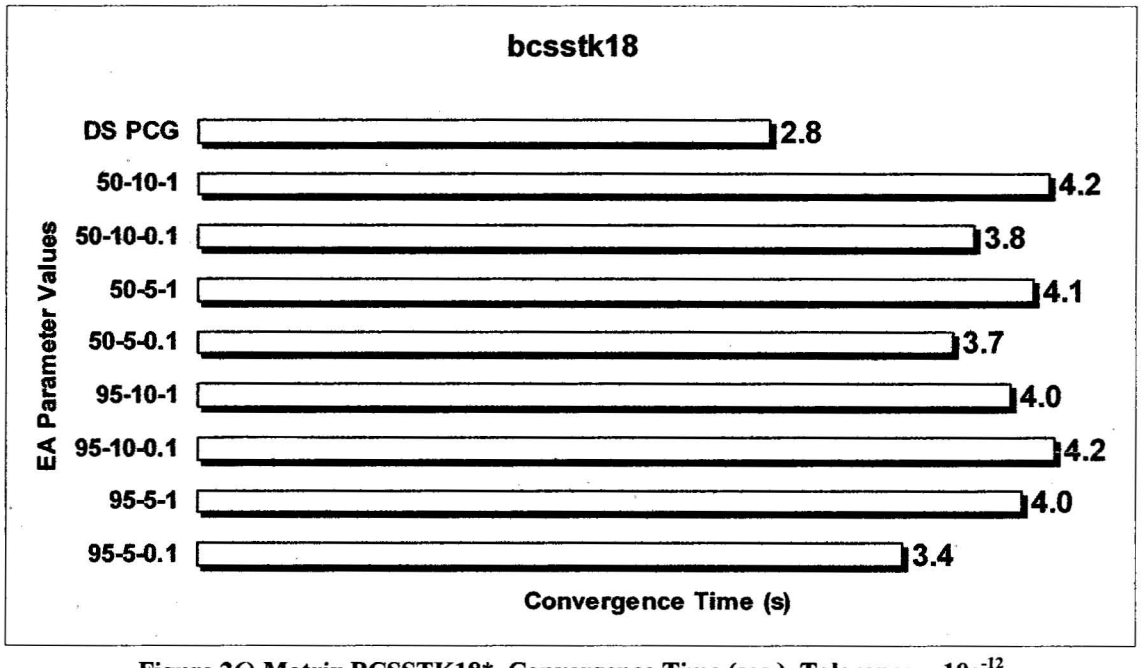

Figure 26) Matrix BCSSTK18*, Convergence Time (sec.), Tolerance = $10e^{-12}$

* The legend indicates the crossover rate, mutation rate and mutation rate standard deviation EA parameters. For example, 95-05-0.1 indicates a 95% crossover rate, a 5% mutation rate and a 0.1% mutation standard deviation.

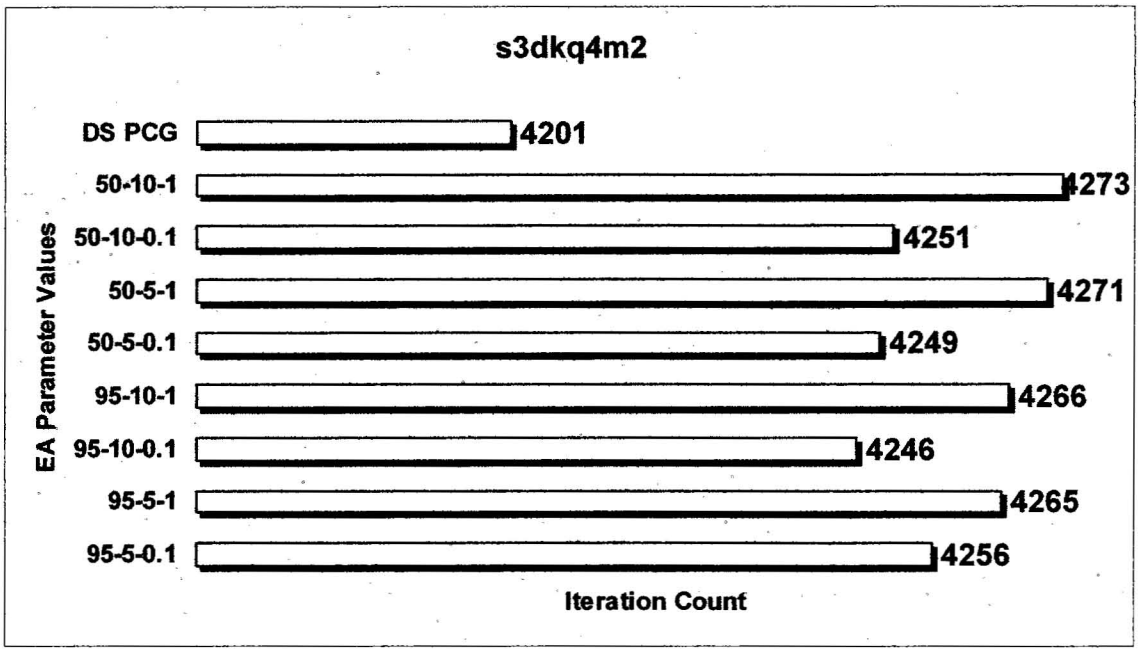

Figure 27) Matrix S3DKQ4M2*, Iteration Count, Tolerance = $10e^{-6}$

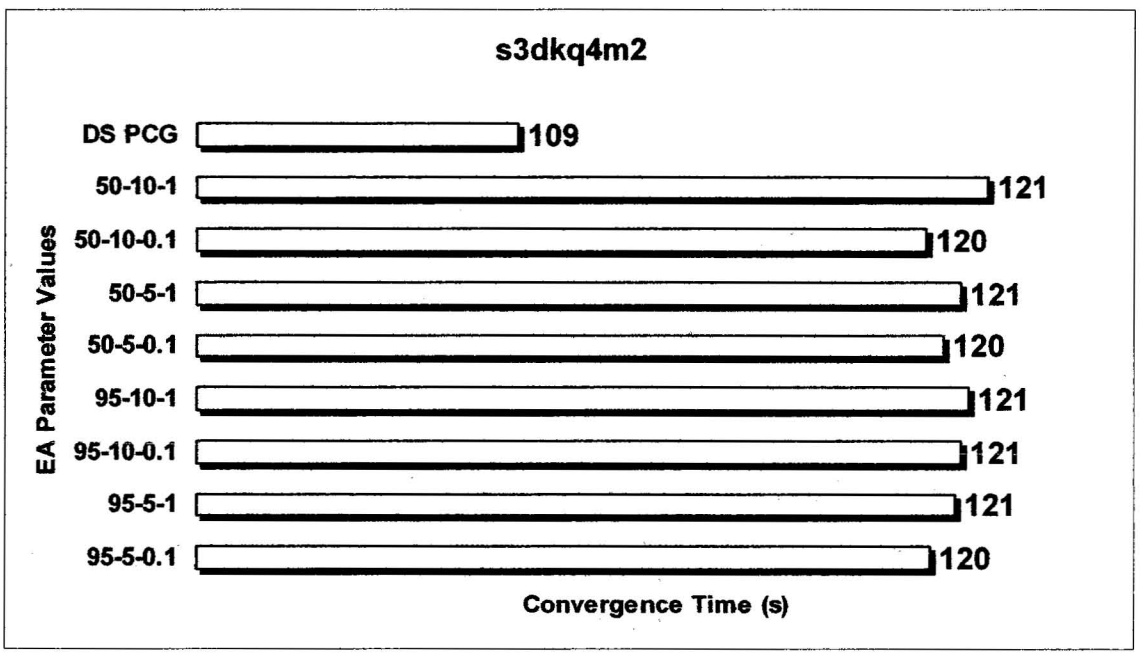

Figure 28) Matrix S3DKQ4M2*, Convergence Time (sec.), Tolerance = $10e^{-6}$

* The legend indicates the crossover rate, mutation rate and mutation rate standard deviation EA parameters. For example, 95-05-0.1 indicates a 95% crossover rate, a 5% mutation rate and a 0.1% mutation standard deviation.

Future Work

Our tests showed no significant improvement by EA over the standard DSPCG method. However, different results may be obtained with other matrices, or with other EA methods. The results obtained from this set of tests, along with other testing that was conducted but not included in this project, leads the author to believe that an EA would not increase the convergence rate of DSPCG.

Greenbaum[3] indicated that the diagonal of the original Matrix is close to the optimal diagonal matrix. The tests conducted agree with this statement and indicate that the EA did not find another diagonal matrix that would outperform the diagonal of the original Matrix.

However, there are other Preconditioning Matrices that can be applied for which the optimal is not known. In the same paper, Greenbaum[3] (pp. 166-167) also stated that there is no known optimal Tridiagonal Preconditioning Matrix. There is room here for an EA to accelerate the PCG in finding an optimal Tridiagonal Matrix that would accelerate either convergence iteration rates or convergence time, or both.

The idea of Matrix Sampling, which was not covered in this project, would assist with any EA that deals with an iterative solution to a set of linear equations. Both Matrix Multiplication and Matrix Equation Solving are areas where sampling the matrix at a much lower frequency than that of a complete matrix multiplication (or solve) could benefit greatly. Matrix sampling for an EA might entail selecting every 100 or 200 rows and columns of the matrix and individual instead of using the full matrix and vector. This would greatly reduce the cost of the EA portion of the algorithm. At each iteration step, the sampled fitness (relative error) of each of the individuals could be used to rank each individual. For EA, it is the ***relative*** fitness of the individual that is important and not the absolute fitness. If the DSPCG method requires the actual fitness, then the whole matrix

and complete individual may be used in the DSPCG step. This method could potentially reduce most of the EA overhead in each generation.

List of References

1. Baeck, T., Fogel D. and Michalewicz, Z., Evolutionary Computation 1: Basic Algorithms and Operators, Institute of Physics Publishing, Bristol and Philadelphia. 2000.

2. Fogol, D. B., Evolutionary Algorithms in Engineering and Computer Science, K. Miettinen et al., eds., Wiley, New York, 1999

3. Greenbaum, Anne, Iterative methods for solving linear systems. Philadelphia, PA: SIAM, 1997.

4. Golub, Gene and Van Loan, Charles: Matrix Computations 3rd Edition, The John Hopkins University Press, Baltimore and London, 1996

5. He, Jun, Xu, Jiyou and Yao. Xin, Solving Equations using Hybrid Evolutionary Computation Techniques, IEEE Transactions on Evolutionary Computation September 2000

6. Meurant, G., Computer Solution of Large Linear Systems. Amsterdam: North Holland, Elsevier, 1999.

7. Mitchell, M., An Introduction to Genetic Algorithms. MIT Press, Cambridge, MA, 1996.

8. Shewchuk, Jonathan Richard, An Introduction to the Conjugate Gradient Method Without the Agonizing Pain, School of Computer Science Carnegie Mellon University, 1994.

9. Weiss, Rüdiger, Parameter Free Iterative Solvers, Mathematical Research (Vol 97) Akademie Verlag, 1996

Web Sites

10. T. Davis, University of Florida Sparse Matrix Collection, retrieved Feb 28, 2006 http://www.cise.ufl.edu/research/sparse/matrices, NA Digest, vol. 92, no. 42, October 16, 1994, NA Digest, vol. 96, no. 28, July 23, 1996, and NA Digest, vol. 97, no. 23, June 7, 1997.

11. EALib, a Java Evolutionary Computation Toolkit, retrieved Feb 24, 2005 http://www.evolvica.org/ealib/index.html

12. Mathcom Solutions Inc., Scientific Computing FAQ: Partial Differential Equations (PDEs) and Finite Element Modeling (FEM), retrieved February 23, 2005 http://www.mathcom.com/corpdir/techinfo.mdir/scifaq/q260.html

13. Mathworld, retrieved September 27, 2005, http://mathworld.wolfram.com/ OpenFEM An Open-Source Finite Element Toolbox, retrieved February 24, 2005 http://www.openfem.net/

14. Netlib, Netlib Repository at UTK and ORNL, retrieved February 24, 2005, http://www.netlib.org

15. OpenFEM An Open-Source Finite Element Toolbox, retrieved February 24, 2006 http://www.openfem.net/

16. University of Saskatchewan, College of Engineering, INTERNET FINITE ELEMENT RESOURCES, retrieved February 23, 2006 http://www.engr.usask.ca/~macphed/finite/fe_resources/fe_resources.html

17. Wikipedia, the free encyclopedia that anyone can edit, Genetic Algorithm, retrieved July 28, 2007, http://en.wikipedia.org/wiki/Genetic_algorithms