Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods[1]

Richard Barrett[2], Michael Berry[3], Tony F. Chan[4],
James Demmel[5], June M. Donato[6], Jack Dongarra[3,6],
Victor Eijkhout[4], Roldan Pozo[7] Charles Romine[6],
and Henk Van der Vorst[8]

[1]This work was supported in part by DARPA and ARO under contract number DAAL03-91-C-0047, the National Science Foundation Science and Technology Center Cooperative Agreement No. CCR-8809615, the Applied Mathematical Sciences subprogram of the Office of Energy Research, U.S. Department of Energy, under Contract DE-AC05-84OR21400, and the Stichting Nationale Computer Faciliteit (NCF) by Grant CRG 92.03.

[2]Los Alamos National Laboratory, Los Alamos, NM 87544.

[3]Department of Computer Science, University of Tennessee, Knoxville, TN 37996-1301.

[4]Applied Mathematics Department, University of California, Los Angeles, CA 90024-1555.

[5]Computer Science Division and Mathematics Department, University of California, Berkeley, CA 94720.

[6]Mathematical Sciences Section, Oak Ridge National Laboratory, Oak Ridge, TN 37831-6367.

[7]National Institute of Standards and Technology, Gaithersburg, MD, 20899

[8]Department of Mathematics, Utrecht University, Utrecht, the Netherlands.

This book is also available in Postscript from over the Internet.
To retrieve the postscript file you can use one of the following methods:

1. anonymous ftp to www.netlib.org
   cd templates
   get templates.ps
   quit

2. from any machine on the Internet type:
   rcp anon@www.netlib.org:templates/templates.ps templates.ps

3. send email to netlib@ornl.gov and in the message type:
   send templates.ps from templates

The url for this book is *http://www.netlib.org/templates/Templates.html* .

A bibtex reference for this book follows:

@BOOK{templates, AUTHOR = {R. Barrett and M. Berry and T. F. Chan and J. Demmel and J. Donato and J. Dongarra and V. Eijkhout and R. Pozo and C. Romine, and H. Van der Vorst }, TITLE = {Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods}, PUBLISHER = {SIAM}, YEAR = {1994}, ADDRESS = {Philadelphia, PA} }

How to Use This Book

We have divided this book into five main chapters. Chapter 1 gives the motivation for this book and the use of templates.

Chapter 2 describes stationary and nonstationary iterative methods. In this chapter we present both historical development and state-of-the-art methods for solving some of the most challenging computational problems facing researchers.

Chapter 3 focuses on preconditioners. Many iterative methods depend in part on preconditioners to improve performance and ensure fast convergence.

Chapter 4 provides a glimpse of issues related to the use of iterative methods. This chapter, like the preceding, is especially recommended for the experienced user who wishes to have further guidelines for tailoring a specific code to a particular machine. It includes information on complex systems, stopping criteria, data storage formats, and parallelism.

Chapter 5 includes overviews of related topics such as the close connection between the Lanczos algorithm and the Conjugate Gradient algorithm, block iterative methods, red/black orderings, domain decomposition methods, multigrid-like methods, and row-projection schemes.

The Appendices contain information on how the templates and `BLAS` software can be obtained. A glossary of important terms used in the book is also provided.

The field of iterative methods for solving systems of linear equations is in constant flux, with new methods and approaches continually being created, modified, tuned, and some eventually discarded. We expect the material in this book to undergo changes from time to time as some of these new approaches mature and become the state-of-the-art. Therefore, we plan to update the material included in this book periodically for future editions. We welcome your comments and criticisms of this work to help us in that updating process. Please send your comments and questions by email to `templates@cs.utk.edu`.

List of Symbols

$A, \ldots, Z$	matrices
$a, \ldots, z$	vectors
$\alpha, \beta, \ldots, \omega$	scalars
A^T	matrix transpose
A^H	conjugate transpose (Hermitian) of A
A^{-1}	matrix inverse
A^{-T}	the inverse of A^T
$a_{i,j}$	matrix element
$a_{\cdot,j}$	jth matrix column
$A_{i,j}$	matrix subblock
a_i	vector element
u_x, u_{xx}	first, second derivative with respect to x
(x, y), $x^T y$	vector dot product (inner product)
$x_j^{(i)}$	jth component of vector x in the ith iteration
$\mathrm{diag}(A)$	diagonal of matrix A
$\mathrm{diag}(\alpha, \beta \ldots)$	diagonal matrix constructed from scalars $\alpha, \beta \ldots$
$\mathrm{span}(a, b \ldots)$	spanning space of vectors $a, b \ldots$
$\mathcal{R}$	set of real numbers
$\mathcal{R}^n$	real n-space
$\|\|x\|\|_2$	2-norm
$\|\|x\|\|_p$	p-norm
$\|\|x\|\|_A$	the "A-norm", defined as $(Ax, x)^{1/2}$
$\lambda_{\max}(A), \lambda_{\min}(A)$	eigenvalues of A with maximum (resp. minimum) modulus
$\sigma_{\max}(A), \sigma_{\min}(A)$	largest and smallest singular values of A
$\kappa_2(A)$	spectral condition number of matrix A
$\mathcal{L}$	linear operator
$\overline{\alpha}$	complex conjugate of the scalar α
$\max\{S\}$	maximum value in set S
$\min\{S\}$	minimum value in set S
$\sum$	summation
$O(\cdot)$	"big-oh" asymptotic bound

Conventions Used in this Book

D	diagonal matrix
L	lower triangular matrix
U	upper triangular matrix
Q	orthogonal matrix
M	preconditioner
$I,\ I^{n\times n}$	$n \times n$ identity matrix
$\hat{x}$	typically, the exact solution to $Ax = b$
h	discretization mesh width

Author's Affiliations

Richard Barrett
Los Alamos National Laboratory

Michael Berry
University of Tennessee, Knoxville

Tony Chan
University of California, Los Angeles

James Demmel
University of California, Berkeley

June Donato
Oak Ridge National Laboratory

Jack Dongarra
University of Tennessee, Knoxville
and Oak Ridge National Laboratory

Victor Eijkhout
University of California, Los Angeles

Roldan Pozo
National Institute of Standards and Technology

Charles Romine
Oak Ridge National Laboratory

Henk van der Vorst
Utrecht University, the Netherlands

Acknowledgments

The authors gratefully acknowledge the valuable assistance of many people who commented on preliminary drafts of this book. In particular, we thank Loyce Adams, Bill Coughran, Matthew Fishler, Peter Forsyth, Roland Freund, Gene Golub, Eric Grosse, Mark Jones, David Kincaid, Steve Lee, Tarek Mathew, Noël Nachtigal, Jim Ortega, and David Young for their insightful comments. We also thank Geoffrey Fox for initial discussions on the concept of templates, and Karin Remington for designing the front cover.

This work was supported in part by DARPA and ARO under contract number DAAL03-91-C-0047, the National Science Foundation Science and Technology Center Cooperative Agreement No. CCR-8809615, the Applied Mathematical Sciences subprogram of the Office of Energy Research, U.S. Department of Energy, under Contract DE-AC05-84OR21400, and the Stichting Nationale Computer Faciliteit (NCF) by Grant CRG 92.03.

Contents

List of Figures

Chapter 1

Introduction

Which of the following statements is true?

- Users want "black box" software that they can use with complete confidence for general problem classes without having to understand the fine algorithmic details.

- Users want to be able to tune data structures for a particular application, even if the software is not as reliable as that provided for general methods.

It turns out both are true, for different groups of users.

Traditionally, users have asked for and been provided with black box software in the form of mathematical libraries such as `LAPACK`, `LINPACK`, `NAG`, and `IMSL`. More recently, the high-performance community has discovered that they must write custom software for their problem. Their reasons include inadequate functionality of existing software libraries, data structures that are not natural or convenient for a particular problem, and overly general software that sacrifices too much performance when applied to a special case of interest.

Can we meet the needs of both groups of users? We believe we can. Accordingly, in this book, we introduce the use of *templates*. A template is a description of a general algorithm rather than the executable object code or the source code more commonly found in a conventional software library. Nevertheless, although templates are general descriptions of key algorithms, they offer whatever degree of customization the user may desire. For example, they can be configured for the specific data structure of a problem or for the specific computing system on which the problem is to run.

We focus on the use of iterative methods for solving large sparse systems of linear equations.

Many methods exist for solving such problems. The trick is to find the most effective method for the problem at hand. Unfortunately, a method that works well for one problem type may not work as well for another. Indeed, it may not work at all.

Thus, besides providing templates, we suggest how to choose and implement an effective method, and how to specialize a method to specific matrix types. We restrict ourselves to *iterative methods*, which work by repeatedly improving an approximate solution until it is accurate enough. These methods access the coefficient matrix A of the linear system only via the matrix-vector product $y = A \cdot x$ (and perhaps $z = A^T \cdot x$). Thus the user need only supply a subroutine for computing y (and perhaps z) given x, which permits full exploitation of the sparsity or other special structure of A.

We believe that after reading this book, applications developers will be able to use templates to get their program running on a parallel machine quickly. Nonspecialists will know how to choose and implement an approach to solve a particular problem. Specialists will be able to assemble and modify their codes—without having to make the huge investment that has, up to now, been required to tune large-scale applications for each particular machine. Finally, we hope that all users will gain a better understanding of the algorithms employed. While education has not been one of the traditional goals of mathematical software, we believe that our approach will go a long way in providing such a valuable service.

1.1 Why Use Templates?

Templates offer three significant advantages. First, templates are general and reusable. Thus, they can simplify ports to diverse machines. This feature is important given the diversity of parallel architectures.

Second, templates exploit the expertise of two distinct groups. The expert numerical analyst creates a template reflecting in-depth knowledge of a specific numerical technique. The computational scientist then provides "value-added" capability to the general template description, customizing it for specific contexts or applications needs.

And third, templates are not language specific. Rather, they are displayed in an Algol-like structure, which is readily translatable into the target language such as `FORTRAN` (with the use of the Basic Linear Algebra Subprograms, or `BLAS`, whenever possible) and `C`. By using these familiar styles, we believe that the users will have trust in the algorithms. We also hope that users will gain a better understanding of numerical techniques and parallel programming.

For each template, we provide some or all of the following:

- a mathematical description of the flow of the iteration;
- discussion of convergence and stopping criteria;
- suggestions for applying a method to special matrix types (*e.g.*, banded systems);
- advice for tuning (for example, which preconditioners are applicable and which are not);
- tips on parallel implementations; and
- hints as to when to use a method, and why.

For each of the templates, the following can be obtained via electronic mail.

- a `MATLAB` implementation based on dense matrices;
- a `FORTRAN-77` program with calls to `BLAS`[1];

[1] For a discussion of `BLAS` as building blocks, see [69, 70, 71, 144] and `LAPACK` routines [3]. Also, see Appendix B.

- a C++ template implementation for matrix/vector classes.

See Appendix A for details.

1.2 What Methods Are Covered?

Many iterative methods have been developed and it is impossible to cover them all. We chose the methods below either because they illustrate the historical development of iterative methods, or because they represent the current state-of-the-art for solving large sparse linear systems. The methods we discuss are:

1. Jacobi
2. Gauss-Seidel
3. Successive Over-Relaxation (SOR)
4. Symmetric Successive Over-Relaxation (SSOR)
5. Conjugate Gradient (CG)
6. Minimal Residual (MINRES) and Symmetric LQ (SYMMLQ)
7. Conjugate Gradients on the Normal Equations (CGNE and CGNR)
8. Generalized Minimal Residual (GMRES)
9. Biconjugate Gradient (BiCG)
10. Quasi-Minimal Residual (QMR)
11. Conjugate Gradient Squared (CGS)
12. Biconjugate Gradient Stabilized (Bi-CGSTAB)
13. Chebyshev Iteration

For each method we present a general description, including a discussion of the history of the method and numerous references to the literature. We also give the mathematical conditions for selecting a given method.

We do not intend to write a "cookbook", and have deliberately avoided the words "numerical recipes", because these phrases imply that our algorithms can be used blindly without knowledge of the system of equations. The state of the art in iterative methods does not permit this: some knowledge about the linear system is needed to guarantee convergence of these algorithms, and generally the more that is known the more the algorithm can be tuned. Thus, we have chosen to present an algorithmic outline, with guidelines for choosing a method and implementing it on particular kinds of high-performance machines. We also discuss the use of preconditioners and relevant data storage issues.

Chapter 2

Iterative Methods

The term "iterative method" refers to a wide range of techniques that use successive approximations to obtain more accurate solutions to a linear system at each step. In this book we will cover two types of iterative methods. Stationary methods are older, simpler to understand and implement, but usually not as effective. Nonstationary methods are a relatively recent development; their analysis is usually harder to understand, but they can be highly effective. The nonstationary methods we present are based on the idea of sequences of orthogonal vectors. (An exception is the Chebyshev iteration method, which is based on orthogonal polynomials.)

The rate at which an iterative method converges depends greatly on the spectrum of the coefficient matrix. Hence, iterative methods usually involve a second matrix that transforms the coefficient matrix into one with a more favorable spectrum. The transformation matrix is called a *preconditioner*. A good preconditioner improves the convergence of the iterative method, sufficiently to overcome the extra cost of constructing and applying the preconditioner. Indeed, without a preconditioner the iterative method may even fail to converge.

2.1 Overview of the Methods

Below are short descriptions of each of the methods to be discussed, along with brief notes on the classification of the methods in terms of the class of matrices for which they are most appropriate. In later sections of this chapter more detailed descriptions of these methods are given.

- Stationary Methods
  - Jacobi.
    The Jacobi method is based on solving for every variable locally with respect to the other variables; one iteration of the method corresponds to solving for every variable once. The resulting method is easy to understand and implement, but convergence is slow.
  - Gauss-Seidel.
    The Gauss-Seidel method is like the Jacobi method, except that it uses updated values as soon as they are available. In general, if the Jacobi

method converges, the Gauss-Seidel method will converge faster than the Jacobi method, though still relatively slowly.

- SOR.

  Successive Overrelaxation (SOR) can be derived from the Gauss-Seidel method by introducing an extrapolation parameter ω. For the optimal choice of ω, SOR may converge faster than Gauss-Seidel by an order of magnitude.

- SSOR.

  Symmetric Successive Overrelaxation (SSOR) has no advantage over SOR as a stand-alone iterative method; however, it is useful as a preconditioner for nonstationary methods.

• Nonstationary Methods

  - Conjugate Gradient (CG).

    The conjugate gradient method derives its name from the fact that it generates a sequence of conjugate (or orthogonal) vectors. These vectors are the residuals of the iterates. They are also the gradients of a quadratic functional, the minimization of which is equivalent to solving the linear system. CG is an extremely effective method when the coefficient matrix is symmetric positive definite, since storage for only a limited number of vectors is required.

  - Minimum Residual (MINRES) and Symmetric LQ (SYMMLQ).

    These methods are computational alternatives for CG for coefficient matrices that are symmetric but possibly indefinite. SYMMLQ will generate the same solution iterates as CG if the coefficient matrix is symmetric positive definite.

  - Conjugate Gradient on the Normal Equations: CGNE and CGNR.

    These methods are based on the application of the CG method to one of two forms of the *normal equations* for $Ax = b$. CGNE solves the system $(AA^T)y = b$ for y and then computes the solution $x = A^T y$. CGNR solves $(A^T A)x = \tilde{b}$ for the solution vector x where $\tilde{b} = A^T b$. When the coefficient matrix A is nonsymmetric and nonsingular, the normal equations matrices AA^T and $A^T A$ will be symmetric and positive definite, and hence CG can be applied. The convergence may be slow, since the spectrum of the normal equations matrices will be less favorable than the spectrum of A.

  - Generalized Minimal Residual (GMRES).

    The Generalized Minimal Residual method computes a sequence of orthogonal vectors (like MINRES), and combines these through a least-squares solve and update. However, unlike MINRES (and CG) it requires storing the whole sequence, so that a large amount of storage is needed. For this reason, restarted versions of this method are used. In restarted versions, computation and storage costs are limited by specifying a fixed number of vectors to be generated. This method is useful for general nonsymmetric matrices.

- BiConjugate Gradient (BiCG).

  The Biconjugate Gradient method generates two CG-like sequences of vectors, one based on a system with the original coefficient matrix A, and one on A^T. Instead of orthogonalizing each sequence, they are made mutually orthogonal, or "bi-orthogonal". This method, like CG, uses limited storage. It is useful when the matrix is nonsymmetric and nonsingular; however, convergence may be irregular, and there is a possibility that the method will break down. BiCG requires a multiplication with the coefficient matrix and with its transpose at each iteration.

- Quasi-Minimal Residual (QMR).

  The Quasi-Minimal Residual method applies a least-squares solve and update to the BiCG residuals, thereby smoothing out the irregular convergence behavior of BiCG, which may lead to more reliable approximations. In full glory, it has a look ahead strategy built in that avoids the BiCG breakdown. Even without look ahead, QMR largely avoids the breakdown that can occur in BiCG. On the other hand, it does not effect a true minimization of either the error or the residual, and while it converges smoothly, it often does not improve on the BiCG in terms of the number of iteration steps.

- Conjugate Gradient Squared (CGS).

  The Conjugate Gradient Squared method is a variant of BiCG that applies the updating operations for the A-sequence and the A^T-sequences both to the same vectors. Ideally, this would double the convergence rate, but in practice convergence may be much more irregular than for BiCG, which may sometimes lead to unreliable results. A practical advantage is that the method does not need the multiplications with the transpose of the coefficient matrix.

- Biconjugate Gradient Stabilized (Bi-CGSTAB).

  The Biconjugate Gradient Stabilized method is a variant of BiCG, like CGS, but using different updates for the A^T-sequence in order to obtain smoother convergence than CGS.

- Chebyshev Iteration.

  The Chebyshev Iteration recursively determines polynomials with coefficients chosen to minimize the norm of the residual in a min-max sense. The coefficient matrix must be positive definite and knowledge of the extremal eigenvalues is required. This method has the advantage of requiring no inner products.

2.2 Stationary Iterative Methods

Iterative methods that can be expressed in the simple form

$$x^{(k)} = Bx^{(k-1)} + c \tag{2.1}$$

(where neither B nor c depend upon the iteration count k) are called *stationary* iterative methods. In this section, we present the four main stationary iterative methods: the *Jacobi method*, the *Gauss-Seidel method*, the *Successive Overrelaxation (SOR)* *method* and the *Symmetric Successive Overrelaxation (SSOR) method*. In each case, we summarize their convergence behavior and their effectiveness, and discuss how and when they should be used. Finally, in §2.2.5, we give some historical background and further notes and references.

2.2.1 The Jacobi Method

The Jacobi method is easily derived by examining each of the n equations in the linear system $Ax = b$ in isolation. If in the ith equation

$$\sum_{j=1}^{n} a_{i,j} x_j = b_i,$$

we solve for the value of x_i while assuming the other entries of x remain fixed, we obtain

$$x_i = (b_i - \sum_{j \neq i} a_{i,j} x_j)/a_{i,i}. \tag{2.2}$$

This suggests an iterative method defined by

$$x_i^{(k)} = (b_i - \sum_{j \neq i} a_{i,j} x_j^{(k-1)})/a_{i,i}, \tag{2.3}$$

which is the Jacobi method. Note that the order in which the equations are examined is irrelevant, since the Jacobi method treats them independently. For this reason, the Jacobi method is also known as the *method of simultaneous displacements*, since the updates could in principle be done simultaneously.

In matrix terms, the definition of the Jacobi method in (2.3) can be expressed as

$$x^{(k)} = D^{-1}(L + U)x^{(k-1)} + D^{-1}b, \tag{2.4}$$

where the matrices D, $-L$ and $-U$ represent the diagonal, the strictly lower-triangular, and the strictly upper-triangular parts of A, respectively.

The pseudocode for the Jacobi method is given in Figure 2.1. Note that an auxiliary storage vector, $\bar{x}$ is used in the algorithm. It is not possible to update the vector x in place, since values from $x^{(k-1)}$ are needed throughout the computation of $x^{(k)}$.

Convergence of the Jacobi method

Iterative methods are often used for solving discretized partial differential equations. In that context a rigorous analysis of the convergence of simple methods such as the Jacobi method can be given.

As an example, consider the boundary value problem

$$\mathcal{L}u = -u_{xx} = f \qquad \text{on } (0,1), \qquad u(0) = u_0, \quad u(1) = u_1,$$

discretized by

$$Lu(x_i) = 2u(x_i) - u(x_{i-1}) - u(x_{i+1}) = f(x_i)/N^2 \qquad \text{for } x_i = i/N,\ i = 1 \ldots N-1.$$

The eigenfunctions of the $\mathcal{L}$ and L operator are the same: for $n = 1 \ldots N-1$ the function $u_n(x) = \sin n\pi x$ is an eigenfunction corresponding to $\lambda = 4\sin^2 n\pi/(2N)$.

Choose an initial guess $x^{(0)}$ to the solution x.
for $k = 1, 2, \ldots$
  for $i = 1, 2, \ldots, n$
    $\bar{x}_i = 0$
    for $j = 1, 2, \ldots, i-1, i+1, \ldots, n$
      $\bar{x}_i = \bar{x}_i + a_{i,j} x_j^{(k-1)}$
    end
    $\bar{x}_i = (b_i - \bar{x}_i)/a_{i,i}$
  end
  $x^{(k)} = \bar{x}$
  check convergence; continue if necessary
end

Figure 2.1: The Jacobi Method

The eigenvalues of the Jacobi iteration matrix B are then $\lambda(B) = 1 - \lambda(L)/2 = 1 - 2\sin^2 n\pi/(2N)$.

From this it is easy to see that the high frequency modes (*i.e.*, eigenfunction u_n with n large) are damped quickly, whereas the damping factor for modes with n small is close to 1. The spectral radius of the Jacobi iteration matrix is $\approx 1 - 10/N^2$, and it is attained for the eigenfunction $u(x) = \sin \pi x$.

The type of analysis applied to this example can be generalized to higher dimensions and other stationary iterative methods. For both the Jacobi and Gauss-Seidel method (below) the spectral radius is found to be $1 - O(h^2)$ where h is the discretization mesh width, *i.e.*, $h = N^{-d}$ where N is the number of variables and d is the number of space dimensions.

2.2.2 The Gauss-Seidel Method

Consider again the linear equations in (2.2). If we proceed as with the Jacobi method, but now assume that the equations are examined one at a time in sequence, and that previously computed results are used as soon as they are available, we obtain the Gauss-Seidel method:

$$x_i^{(k)} = (b_i - \sum_{j<i} a_{i,j} x_j^{(k)} - \sum_{j>i} a_{i,j} x_j^{(k-1)})/a_{i,i}. \tag{2.5}$$

Two important facts about the Gauss-Seidel method should be noted. First, the computations in (2.5) appear to be serial. Since each component of the new iterate depends upon all previously computed components, the updates cannot be done simultaneously as in the Jacobi method. Second, the new iterate $x^{(k)}$ depends upon the order in which the equations are examined. The Gauss-Seidel method is sometimes called the *method of successive displacements* to indicate the dependence of the iterates on the ordering. If this ordering is changed, the *components* of the new iterate (and not just their order) will also change.

```
Choose an initial guess x^(0) to the solution x.
for k = 1, 2, ...
    for i = 1, 2, ..., n
        σ = 0
        for j = 1, 2, ..., i − 1
            σ = σ + a_{i,j} x_j^(k)
        end
        for j = i + 1, ..., n
            σ = σ + a_{i,j} x_j^(k−1)
        end
        x_i^(k) = (b_i − σ)/a_{i,i}
    end
    check convergence; continue if necessary
end
```

Figure 2.2: The Gauss-Seidel Method

These two points are important because if A is sparse, the dependency of each component of the new iterate on previous components is not absolute. The presence of zeros in the matrix may remove the influence of some of the previous components. Using a judicious ordering of the equations, it may be possible to reduce such dependence, thus restoring the ability to make updates to groups of components in parallel. However, reordering the equations can affect the rate at which the Gauss-Seidel method converges. A poor choice of ordering can degrade the rate of convergence; a good choice can enhance the rate of convergence. For a practical discussion of this tradeoff (parallelism versus convergence rate) and some standard reorderings, the reader is referred to Chapter 3 and §4.4.

In matrix terms, the definition of the Gauss-Seidel method in (2.5) can be expressed as

$$x^{(k)} = (D - L)^{-1}(Ux^{(k-1)} + b). \tag{2.6}$$

As before, D, $-L$ and $-U$ represent the diagonal, lower-triangular, and upper-triangular parts of A, respectively.

The pseudocode for the Gauss-Seidel algorithm is given in Figure 2.2.

2.2.3 The Successive Overrelaxation Method

The Successive Overrelaxation Method, or SOR, is devised by applying extrapolation to the Gauss-Seidel method. This extrapolation takes the form of a weighted average between the previous iterate and the computed Gauss-Seidel iterate successively for each component:

$$x_i^{(k)} = \omega \bar{x}_i^{(k)} + (1 - \omega)x_i^{(k-1)}$$

```
Choose an initial guess x^(0) to the solution x.
for k = 1, 2, ...
    for i = 1, 2, ..., n
        σ = 0
        for j = 1, 2, ..., i − 1
            σ = σ + a_{i,j} x_j^(k)
        end
        for j = i + 1, ..., n
            σ = σ + a_{i,j} x_j^(k−1)
        end
        σ = (b_i − σ)/a_{i,i}
        x_i^(k) = x_i^(k−1) + ω(σ − x_i^(k−1))
    end
    check convergence; continue if necessary
end
```

Figure 2.3: The SOR Method

(where $\bar{x}$ denotes a Gauss-Seidel iterate, and ω is the extrapolation factor). The idea is to choose a value for ω that will accelerate the rate of convergence of the iterates to the solution.

In matrix terms, the SOR algorithm can be written as follows:

$$x^{(k)} = (D - \omega L)^{-1}(\omega U + (1-\omega)D)x^{(k-1)} + \omega(D - \omega L)^{-1}b. \tag{2.7}$$

The pseudocode for the SOR algorithm is given in Figure 2.3.

Choosing the Value of ω

If $\omega = 1$, the SOR method simplifies to the Gauss-Seidel method. A theorem due to Kahan [130] shows that SOR fails to converge if ω is outside the interval $(0, 2)$. Though technically the term *underrelaxation* should be used when $0 < \omega < 1$, for convenience the term overrelaxation is now used for any value of $\omega \in (0, 2)$.

In general, it is not possible to compute in advance the value of ω that is optimal with respect to the rate of convergence of SOR. Even when it is possible to compute the optimal value for ω, the expense of such computation is usually prohibitive. Frequently, some heuristic estimate is used, such as $\omega = 2 - O(h)$ where h is the mesh spacing of the discretization of the underlying physical domain.

If the coefficient matrix A is symmetric and positive definite, the SOR iteration is guaranteed to converge for *any* value of ω between 0 and 2, though the choice of ω can significantly affect the rate at which the SOR iteration converges. Sophisticated implementations of the SOR algorithm (such as that found in `ITPACK` [140]) employ adaptive parameter estimation schemes to try to home in on the appropriate value of ω by estimating the rate at which the iteration is converging.

For coefficient matrices of a special class called *consistently ordered with property A* (see Young [217]), which includes certain orderings of matrices arising from the discretization of elliptic PDEs, there is a direct relationship between the spectra of the Jacobi and SOR iteration matrices. In principle, given the spectral radius ρ of the Jacobi iteration matrix, one can determine *a priori* the theoretically optimal value of ω for SOR:

$$\omega_{\text{opt}} = \frac{2}{1+\sqrt{1-\rho^2}}. \tag{2.8}$$

This is seldom done, since calculating the spectral radius of the Jacobi matrix requires an impractical amount of computation. However, relatively inexpensive rough estimates of ρ (for example, from the power method, see Golub and Van Loan [109, p. 351]) can yield reasonable estimates for the optimal value of ω.

2.2.4 The Symmetric Successive Overrelaxation Method

If we assume that the coefficient matrix A is symmetric, then the Symmetric Successive Overrelaxation method, or SSOR, combines two SOR sweeps together in such a way that the resulting iteration matrix is similar to a symmetric matrix. Specifically, the first SOR sweep is carried out as in (2.7), but in the second sweep the unknowns are updated in the reverse order. That is, SSOR is a forward SOR sweep followed by a backward SOR sweep. The similarity of the SSOR iteration matrix to a symmetric matrix permits the application of SSOR as a preconditioner for other iterative schemes for symmetric matrices. Indeed, this is the primary motivation for SSOR since its convergence rate, with an optimal value of ω, is usually *slower* than the convergence rate of SOR with optimal ω (see Young [217, page 462]). For details on using SSOR as a preconditioner, see Chapter 3.

In matrix terms, the SSOR iteration can be expressed as follows:

$$x^{(k)} = B_1 B_2 x^{(k-1)} + \omega(2-\omega)(D-\omega U)^{-1} D (D-\omega L)^{-1} b, \tag{2.9}$$

where

$$B_1 = (D-\omega U)^{-1}(\omega L + (1-\omega)D),$$

and

$$B_2 = (D-\omega L)^{-1}(\omega U + (1-\omega)D).$$

Note that B_2 is simply the iteration matrix for SOR from (2.7), and that B_1 is the same, but with the roles of L and U reversed.

The pseudocode for the SSOR algorithm is given in Figure 2.4.

2.2.5 Notes and References

The modern treatment of iterative methods dates back to the relaxation method of Southwell [193]. This was the precursor to the SOR method, though the order in which approximations to the unknowns were relaxed varied during the computation. Specifically, the next unknown was chosen based upon estimates of the location of the largest error in the current approximation. Because of this, Southwell's relaxation method

Choose an initial guess $x^{(0)}$ to the solution x.
Let $x^{(\frac{1}{2})} = x^{(0)}$.
for $k = 1, 2, \ldots$
  for $i = 1, 2, \ldots, n$
    $\sigma = 0$
    for $j = 1, 2, \ldots, i-1$
      $\sigma = \sigma + a_{i,j} x_j^{(k-\frac{1}{2})}$
    end
    for $j = i+1, \ldots, n$
      $\sigma = \sigma + a_{i,j} x_j^{(k-1)}$
    end
    $\sigma = (b_i - \sigma)/a_{i,i}$
    $x_i^{(k-\frac{1}{2})} = x_i^{(k-1)} + \omega(\sigma - x_i^{(k-1)})$
  end
  for $i = n, n-1, \ldots, 1$
    $\sigma = 0$
    for $j = 1, 2, \ldots, i-1$
      $\sigma = \sigma + a_{i,j} x_j^{(k-\frac{1}{2})}$
    end
    for $j = i+1, \ldots, n$
      $\sigma = \sigma + a_{i,j} x_j^{(k)}$
    end
    $x_i^{(k)} = x_i^{(k-\frac{1}{2})} + \omega(\sigma - x_i^{(k-\frac{1}{2})})$
  end
  check convergence; continue if necessary
end

Figure 2.4: The SSOR Method

was considered impractical for automated computing. It is interesting to note that the introduction of multiple-instruction, multiple data-stream (MIMD) parallel computers has rekindled an interest in so-called *asynchronous*, or *chaotic* iterative methods (see Chazan and Miranker [54], Baudet [30], and Elkin [92]), which are closely related to Southwell's original relaxation method. In chaotic methods, the order of relaxation is unconstrained, thereby eliminating costly synchronization of the processors, though the effect on convergence is difficult to predict.

The notion of accelerating the convergence of an iterative method by extrapolation predates the development of SOR. Indeed, Southwell used overrelaxation to accelerate the convergence of his original relaxation method. More recently, the *ad hoc SOR* method, in which a different relaxation factor ω is used for updating each variable, has given impressive results for some classes of problems (see Ehrlich [83]).

The three main references for the theory of stationary iterative methods are