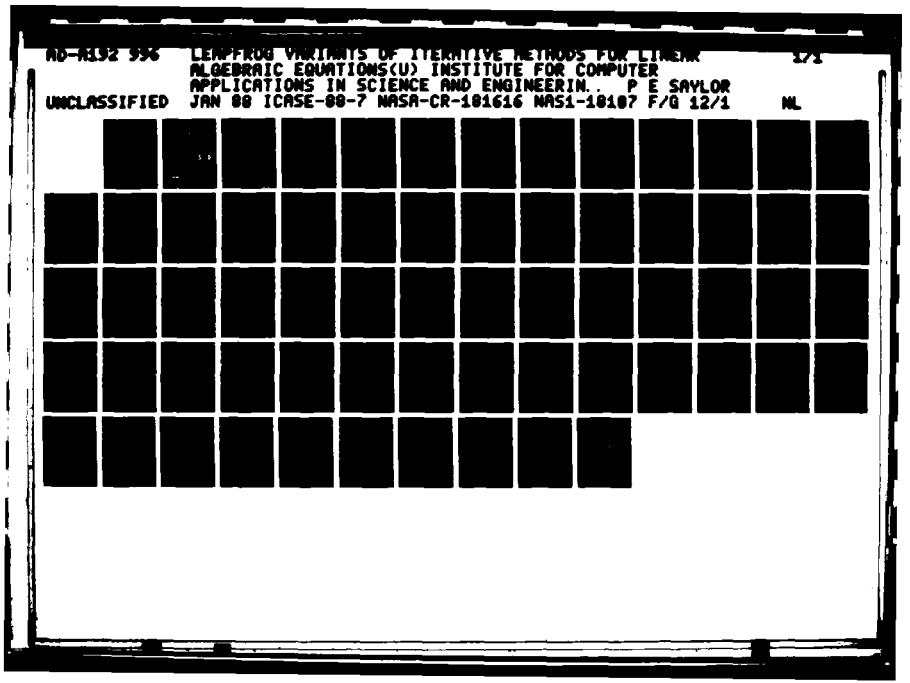
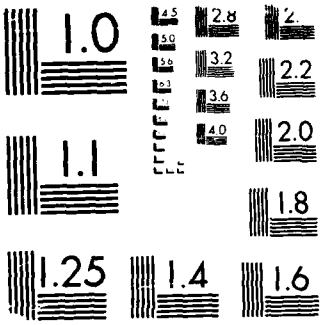

AD-A192 996 LEAPFROG VARIANTS OF ITERATIVE METHODS FOR LINEAR 1/1
ALGEBRAIC EQUATIONS(U) INSTITUTE FOR COMPUTER
APPLICATIONS IN SCIENCE AND ENGINEERIN.. P E SAYLOR
UNCLASSIFIED JAN 88 ICASE-88-7 NASA-CR-181616 NAS1-18107 F/G 12/1 NL



MICROCOPY RESOLUTION TEST CHART

NASA Contractor Report 181616

ICASE REPORT NO. 88-7

ICASE

LEAPFROG VARIANTS OF ITERATIVE METHODS
FOR LINEAR ALGEBRAIC EQUATIONS

Paul E. Saylor

Contract No. NAS1-18107 and AFOSR 85-0189
January 1988

INSTITUTE FOR COMPUTER APPLICATIONS IN SCIENCE AND ENGINEERING
NASA Langley Research Center, Hampton, Virginia 23665

Operated by the Universities Space Research Association

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665

TABLE OF CONTENTS

LEAPFROG VARIANTS OF ITERATIVE METHODS FOR LINEAR ALGEBRAIC EQUATIONS

Paul E. Saylor

ABSTRACT

Two iterative methods are considered, Richardson's method and a general second order method. For both methods, a variant of the method is derived for which only even numbered iterates are computed. The variant is called a *leapfrog* method. Comparisons between the conventional form of the methods and the leapfrog form are made under the assumption that the number of unknowns is large. In the case of Richardson's method, it is possible to express the final iterate in terms of only the initial approximation, a variant of the iteration called the *grand-leap* method. In the case of the grand-leap variant, a set of parameters is required. An algorithm is presented to compute these parameters that is related to algorithms to compute the weights and abscissas for Gaussian quadrature. General algorithms to implement the leapfrog and grand-leap methods are presented. Algorithms for the important special case of the Chebyshev method are also given.

This research was supported by the National Aeronautics and Space Administration under NASA Contract No. NAS1-18107 and by the U.S. Office of Scientific Research under Contract No. AFOSR 85-0189 while the author was in residence at the Institute for Computer Applications in Science and Engineering (ICASE), NASA Langley Research Center, Hampton, VA 23665.

1. Introduction.

The subject of this paper is a set of techniques to improve efficiency in the iterative solution of a real or complex linear system $\mathbf{Ax}=\mathbf{b}$, especially for the solution of large problems on supercomputers.

An iterative method generates a sequence ..., $\mathbf{x}^{(i-2)}$, $\mathbf{x}^{(i-1)}$, $\mathbf{x}^{(i)}$, ... For the methods of this paper, a variant such that $\mathbf{x}^{(i)}$ can be expressed directly in terms of $\mathbf{x}^{(i-2)}$ with no dependence on $\mathbf{x}^{(i-1)}$ will be called a *leapfrog* method. A variant of Richardson's method is also presented for which the final iterate is computed from the intial approximation with no computation of intermediate iterates. This will be called the *grand–leap* method. The advantages of the leapfrog and grand–leap methods are: (i) a slight reduction in some cases in the total number of arithmetic operations; (ii) an increase in the number of terms in vector sums, an advantage on supercomputers that "chain", i.e., transmit results from one arithmetic unit directly to another; and (iii) a reduction in I/O operations for large problems. In his Ph. D. thesis [Chro86] studied methods to omit intermediate successive iterates for the conjugate gradient method as a way to allow parallel computation of matrix vector products. His goals overlapped somewhat with those of this paper but the approach is not the same.

Two iterative methods are considered: Richardson's method [FoWa60, HaYo81] and a general second order iterative method. For Richardson's method the leapfrog method was used in [Smol81, SmSa85] as a technique to avoid complex arithmetic. In this paper its other properties are explored.

Richardson's method is an old method the advantages of which have generally been ignored; however, see [AnGo72]. In the symmetric positive definite case, Richardson iteration parameters do not yield an optimum iterate at each step whereas a second order method does. This is one reason for the neglect of Richardson's method. However, in a paper of Tal–Ezer [Tal87], a novel approach is described in which Richardson iterates are almost optimum at each step.

The Chebyshev iteration is an example of a second order method [Mnt77, Mnt78], used for the solution of nonsymmetric systems. The Chebyshev iteration is not applicable, however, unless the eigenvalues of $\mathbf{A}$ lie in a half plane. Furthermore, the Manteuffel adaptive algorithm [Mnt78] assumes the eigenvalues appear in complex conjugate pairs, which holds if the matrix is real. This is a brief argument for the use of Richardson's method if the matrix is either a general real nonsymmetric matrix with eigenvalues in both half planes or is a complex matrix the eigenvalues of which do not appear in complex conjugate pairs. It should be stressed that large complex matrices arise in signal processing, and constitute an important class of problems.

1.1. Summary. In §2, the leapfrog version of Richardson's method is derived. Iteration parameters are assumed given, with the exception of the Chebyshev case for which explicit formulas are given as well as an algorithm. With properly chosen parameters, the method applies to any real or complex matrix.

In §3, the grand-leap method is presented for computing the final Richardson's method iterate in terms of the initial iterate. An algorithm is also given.

Comparisons among the conventional, leapfrog and grand-leap versions of Richardson's method are made in §4.

In §5, the general formula for a second order method is stated, and a leapfrog version derived. An algorithm (Algorithm 3) is stated in which the parameters are assumed given. Algorithms for these parameters are presented in §7.

Optimum L_∞-iteration parameters are defined in the Chebyshev case in §2. In §6, L_2-optimum parameters are defined. Optimum L_2-Richardson's parameters, in the case of real eigenvalues, are the roots of an orthogonal polynomial. An algorithm to compute roots of orthogonal polynomials is developed, which is an implementation of the Stieltjes algorithm [Gaut82] and related to an algorithm presented in [GoWe69] for the weights and nodes for Gaussian quadrature. This algorithm is modified to yield the quantities required to execute the grand-leap method. The L_2-approach is only one approach to optimum parameters. A non-L_2 treatment is given in [ElSt85, Tal87]. Each of these references is more general than the L_2-methods in §6. A completely general L_2-approach may be based on [SaSm88] but is beyond the scope of this paper.

Algorithms based on the methods in §6 are gathered and presented in §7. Algorithms for the special and important Chebyshev case are also given in §7.

1.2. Conventions and Notation. Although an l_2-inner product is a special case of the L_2-inner product if the measure is chosen correctly, for clarity and convenience, the two are used separately.

The solution of a linear set of equations $\mathbf{Ax} = \mathbf{b}$ generally requires that the set be preconditioned by transforming it into a set such as, for example, $\mathbf{CAx} = \mathbf{Cb}$ for which the iterative method converges more rapidly. (Other preconditionings yield systems such as $\mathbf{CAQQ}^{-1}\mathbf{x} = \mathbf{Cb}$, but the same remarks hold for these other cases.) There is no change in the techniques or algorithms presented in this paper if they are applied to $\mathbf{CAx} = \mathbf{Cb}$ rather than $\mathbf{Ax} = \mathbf{b}$ other than the change in the matrix from $\mathbf{A}$ to $\mathbf{CA}$. It will therefore be assumed that $\mathbf{A}$ is the preconditioned matrix.

It will be convenient from time to time to state that an algorithm converges with no restriction on the input data and if certain conditions on the eigenvalues are met. In a practical sense, of course, there are restrictions on the data such as those needed to prevent overflow, which may be infeasible to analyze and formulate.

Matrices and vectors are denoted by boldface type.

The number of unknowns is denoted by N.

2. Richardson's Method: Conventional Form and Leapfrog Form.

This section begins with a statement of Richardson's method, from which the leapfrog form is easily derived. The Chebyshev case is outlined and an algorithm given.

2.1. Conventional Richardson's Method. Let τ_0 , ..., τ_{k-1} be a *cycle* of *iteration parameters* where k is called the *period*. The purpose of iteration parameters is to reduce the error, but discussion of this is postponed until later. For now, attention is directed to the iteration, and the reader is asked to accept the parameters as given.

Let $\mathbf{x}^{(0)}$ be an initial guess. Richardson's method is defined as follows. For $i = 1, \ldots, k$,

$$\mathbf{r}^{(i-1)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(i-1)}$$

$$\mathbf{x}^{(i)} = \mathbf{x}^{(i-1)} + \tau_{i-1}\mathbf{r}^{(i-1)} . \qquad (2.1.1)$$

2.2. Leapfrog Form. The recursion

$$\mathbf{r}^{(i)} = \mathbf{r}^{(i-1)} - \tau_{i-1}\mathbf{A}\mathbf{r}^{(i-1)} \qquad (2.2.1)$$

will be used, which may be derived by first subtracting (2.1.1) from $\mathbf{x} = \mathbf{x}$ to obtain

$$\mathbf{e}^{(i)} = \mathbf{e}^{(i-1)} - \tau_{i-1}\mathbf{r}^{(i-1)} \tag{2.2.2}$$

where $\mathbf{e}^{(i)} = \mathbf{x} - \mathbf{x}^{(i)}$, and then multiplying (2.2.2) by $\mathbf{A}$. Since $\mathbf{r}^{(i)} = \mathbf{A}\mathbf{e}^{(i)}$, (2.2.1) follows. Vectors $\mathbf{e}^{(i)}$ and $\mathbf{r}^{(i)}$ are called the *(true) error* and the *residual error* respectively.

The leapfrog step from $\mathbf{x}^{(i-2)}$ to $\mathbf{x}^{(i)}$ results from using (2.2.1) in (2.1.1) to give [SmSa85], for $i = 2, 4, \ldots, k$ (under the assumption that k is even)

$$\mathbf{x}^{(i)} = \mathbf{x}^{(i-2)} + \tau_{i-2}\mathbf{r}^{(i-2)} + \tau_{i-1}\left[\mathbf{I} - \tau_{i-2}\mathbf{A}\right]\mathbf{r}^{(i-2)}$$

$$= \mathbf{x}^{(i-2)} + (\tau_{i-1} + \tau_{i-2})\mathbf{r}^{(i-2)} - \tau_{i-2}\tau_{i-1}\mathbf{A}\mathbf{r}^{(i-2)} .$$

2.3. Optimum Chebyshev Parameters. It is easy to show that the error and residual error satisfy

$$\mathbf{e}^{(k)} = R_k(\mathbf{A})\mathbf{e}^{(0)} ,$$

$$\mathbf{r}^{(k)} = R_k(\mathbf{A})\mathbf{r}^{(0)} \tag{2.3.1}$$

where R_k is the polynomial

$$R_k(\varsigma) = \prod_{i=1}^{k}(1 - \tau_{i-1}\varsigma).$$

Any polynomial such that $R_k(0) = 1$ is called a *residual polynomial* [Stie58]. Parameters are chosen to minimize $R_k(\mathbf{A})$ in some sense, to be discussed next.

Let Ω be a set containing the spectrum of $\mathbf{A}$. Usually one thinks of Ω as an interval or union of intervals on the real line. However if $\mathbf{A}$ is nonsymmetric, then both Ω and the spectrum may lie off the real axis in the complex plane. Two commonly used methods to minimize $R_k(\mathbf{A})$ are either to minimize the L_∞-norm of polynomial $R_k(\varsigma)$ over Ω or to minimize a weighted L_2-norm over Ω. In this part, only the L_∞-norm will be discussed. How Chebyshev polynomials are used to minimize this norm will now be outlined. The papers of Manteuffel give more details [Mnt77, Mnt78].

The *Chebyshev residual polynomial* is defined by

$$R_k(\varsigma) = \frac{T_k\left(\dfrac{\varsigma - d}{c}\right)}{T_k\left(-\dfrac{d}{c}\right)},$$

where T_k is the Chebyshev polynomial of degree k, and d and c are parameters defining a confocal family of ellipses: d is the center and the foci are $d \pm c$. Henceforth, in the Chebyshev case, the set Ω containing the nonzero spectrum will be an ellipse. (An ellipse, as the term is used here, means the union of the curve and its interior.) Parameter c is assumed either real or purely imaginary; in either case, c^2 is real. (If $c = 0$, $R_k(\varsigma)$ reduces to $(\varsigma - d)^k / d^k$.) Assume that d is real and one member of the confocal family with center d and foci $d \pm c$ is in the interior of the right half plane, i. e., $d > 0$. (If there is one in the left half plane, then we consider $-\mathbf{A}$ instead of $\mathbf{A}$.) These assumptions mean respectively

that the major axis of each ellipse of the family is either on the real line or that the major axis of each ellipse is perpendicular to the real line. If the major axis is on the real line, then the assumption that at least one member of the family lie in the interior of the right half plane means that $d - |c| > 0$. Finally, note that if the eigenvalues are real, then ellipse Ω may be assumed to be the interval $[d - |c|, d + |c|]$, which is the degenerate ellipse of the confocal family.

Among all residual polynomials, it may be shown [Mnt78] that the Chebyshev residual polynomial has the minimum L_∞-norm over the interval $\Omega = [d - |c|, d + |c|]$, and closely approximates the residual polynomial with minimum L_∞-norm over any ellipse, Ω, with center d and foci $d \pm c$.

It is not necessary that d and c^2 be real in order that the Chebyshev iteration converge. The reason for assuming above that these are real quantities is connected with the Manteuffel algorithm [Mnt78]. The Manteuffel algorithm is valid only when the eigenvalues of $\mathbf{A}$ appear in complex conjugate pairs, and it is this that leads to the assumption that d and c^2 are real. In general for any d and c^2, convergence results if there is one ellipse with center d and foci $d \pm c$ containing the eigenvalues that does not also contain the origin. As a practical matter, however, the Chebyshev iteration is useful only when d and c can be obtained by some technique such as the Manteuffel algorithm.

In the Chebyshev case the Richardson iteration parameters are derived from the roots, $\{\rho_i\}$, of T_k which are

$$\rho_i = \cos\left\{\pi\frac{1+2i}{2k}\right\},\ i = 0,\ \ldots,\ k-1.$$

The roots of R_k are therefore $d + c\rho_i$, for $i = 0, \ldots, k-1$ and the parameters for Richardson's method are

$$\tau_i = \frac{1}{c\rho_i + d}.$$

2.4. An Algorithm in the Chebyshev Case. First a technical note on avoiding complex arithmetic: If the major axis is vertical, i.e., if c is pure imaginary, then the roots of R_k occur in conjugate pairs. It is an advantage to order the parameters $\{\tau_i\}$ in such a way that, in this case, τ_{i-1} and τ_{i-2} are conjugate pairs (in order that $\tau_{i-1} + \tau_{i-2}$ and $\tau_{i-1}\tau_{i-2}$, required by the algorithm, are real), a task equivalent to ordering the roots, $\{\rho_i\}$, of T_k in such a way that $\rho_{i-2} = -\rho_{i-1}$.

Let $h = \pi/2k$. The roots of T_k are the cosine's of $\theta_1 = h$, $\theta_2 = \pi - h$, $\theta_3 = 3h$, $\theta_4 = \pi - 3h$, ... If k is even, which it is in the leapfrog case, the last two roots in this ordering are the cosine's of $\theta_{k-1} = \frac{\pi}{2} - h$ and $\theta_k = \frac{\pi}{2} + h$. Moreover, $\cos\theta_1 = -\cos\theta_2$, etc. In the algorithm, the formula

$$\theta_i = \left\{2\left\lfloor\frac{i+1}{2}\right\rfloor - 1\right\}(-1)^{i-1}h + \pi\,\mathrm{mod}(i+1,\ 2)$$

for $i = 1, ..., k$, will be used. If $\rho_i = \cos\theta_i$, then in the case when c is pure imaginary, $\{d + c\rho_1, d + c\rho_2\}$, $\{d + c\rho_3, d + c\rho_4\}$, ... is a sequence of conjugate pairs.

If the major axis is real, the error is reduced after a cycle of exactly k parameters, but the algorithm may not have converged. If the major axis is vertical, the error is reduced only if k is sufficiently large, a requirement that in practical applications, however, is observed to be reasonable. If the algorithm has not converged, the cycle of parameters is repeated. After the parameters have been recycled l times, the error, $\mathbf{e}^{(kl)} = \mathbf{x} - \mathbf{x}^{(kl)}$, satisfies $\mathbf{e}^{(kl)} = \left(R_k(\mathbf{A})\right)^l \mathbf{e}^{(0)}$. But $R_{kl}(\zeta) \neq \left(R_k(\zeta)\right)^l$ where R_{kl} is the optimum Chebyshev residual polynomial of degree kl. This is a basic problem with Richardson's method: It is optimum only at the end of one cycle of parameters. For an alternative approach, not based on Chebyshev parameters, see [Tal87] in which a method is proposed to increase the number of parameters in an almost optimum way (thus the period is not fixed) until convergence is achieved.

Algorithm 1. (Leapfrog Richardson's method in the Chebyshev case.)

Purpose. Execute Richardson's method with Chebyshev parameters and omit alternate steps.

Input. Matrix $\mathbf{A}$, right side $\mathbf{b}$, initial guess $\mathbf{x}^{(0)}$, cycle k, and ellipse parameters d and c. The ellipse parameters are assumed known, for example, as output from the Manteuffel algorithm [Mnt78, Ashb85]. The user must also

provide a maximum number of cycles of iterations and an error criterion to halt the iteration.

Output. Iterate $\mathbf{x}^{(k)}$, the iterate reached after the last cycle of k parameters in the standard execution of Richardson's method.

Restrictions. If d and c^2 are real, then d is assumed either positive or negative and will be assumed positive without loss of generality. Also, for real c, $0 < d - |c|$. In general for any d and c^2, convergence results if there is one ellipse with center d and foci $d \pm c$ containing the eigenvalues that does not also contain the origin. If the matrix is singular, the algorithm converges to a solution if the system is consistent. Period k is even.

Notes. (1) Quantities θ_i, ρ_i, and τ_i need not be array variables since only three values are used during execution, but subscripts make the algorithm more convenient to state. (2) A slight modification of the algorithm would allow $\{\tau_i\}$ to be an input array, for example, from Algorithm 4.

1) Set $h := \dfrac{\pi}{2k}$.

2) Do either until convergence or a limit on the number of loops is exceeded.

2.1) For $i = 2$ to k by 2 do:

2.1.1) Set $\theta_{i-1} := \left\{2\left\lfloor\dfrac{i}{2}\right\rfloor - 1\right\}(-1)^{i-2}h + \pi \mathrm{mod}(i, 2)$.

2.1.2) Set $\theta_i := \left\{2\left\lfloor\dfrac{i+1}{2}\right\rfloor - 1\right\}(-1)^{i-1}h + \pi \mathrm{mod}(i+1, 2)$.

2.1.3) Set $\rho_{i-1} := \cos\theta_{i-1}$.

2.1.4) Set $\rho_i := \cos\theta_i$.

2.1.5) Set $\tau_{i-2} := \dfrac{1}{d + c\,\rho_{i-1}}$.

2.1.6) Set $\tau_{i-1} := \dfrac{1}{d + c\,\rho_i}$.

2.1.7) Set $\alpha := \tau_{i-2} + \tau_{i-1}$.

2.1.8) Set $\nu := \tau_{i-2}\tau_{i-1}$.

2.1.9) Set $\mathbf{r}^{(i-2)} := \mathbf{b} - \mathbf{A}\mathbf{x}^{(i-2)}$.

2.1.10) Set $\mathbf{t} := \mathbf{A}\mathbf{r}^{(i-2)}$.

2.1.11) Set $\mathbf{x}^{(i)} := \mathbf{x}^{(i-2)} + \alpha\mathbf{r}^{(i-2)} - \nu\mathbf{t}$.

2.1.12) Endfor.

2.2) If not converged, set $\mathbf{x}^{(0)} := \mathbf{x}^{(k)}$.

2.3) Enddo.

3. Richardson's Method in One Step.

It is easy to see that leapfrog could be continued further to allow computing $\mathbf{x}^{(i)}$ from $\mathbf{x}^{(i-4)}$. Ultimately, one arrives at an expression for $\mathbf{x}^{(k)}$ in terms of $\mathbf{x}^{(0)}$ with no intermediate approximations, the form of which is, as will be seen momentarily,

$$\mathbf{x}^{(k)} = \mathbf{x}^{(0)} + C_{k-1}(\mathbf{A})\mathbf{r}^{(0)} , \tag{3.1}$$

where C_{k-1} is a polynomial of degree $k-1$. Computing $\mathbf{x}^{(k)}$ only from $\mathbf{x}^{(0)}$ while omitting the computation of any intermediate approximation will be called the *grand-leap*.

3.1. Krylov Subspace Methods. Richardson's method is an example of a Krylov subspace method, i. e., for $1 \leq i$,

$$\mathbf{x}^{(i)} - \mathbf{x}^{(0)} \in V_i , \tag{3.1.1}$$

where V_i is the *Krylov subspace* defined by

$$V_i = \text{span}\left\{\mathbf{r}^{(0)}, \ldots, \mathbf{A}^{i-1}\mathbf{r}^{(0)}\right\}.$$

The proof of (3.1.1) is an easy induction based on $\mathbf{x}^{(i)} = \mathbf{x}^{(i-1)} + \tau_{i-1}\mathbf{r}^{(i-1)}$. Membership relation (3.1.1) implies (3.1).

3.2. An Expression for $\mathbf{C}_{k-1}$. Multiply

$$\mathbf{x}^{(k)}-\mathbf{x}^{(0)}=\mathbf{x}^{(k)}-\mathbf{x}+\mathbf{x}-\mathbf{x}^{(0)}=C_{k-1}(\mathbf{A})\mathbf{r}^{(0)}$$

by $\mathbf{A}$ to obtain

$$\mathbf{r}^{(0)}-\mathbf{r}^{(k)}=\mathbf{A}C_{k-1}(\mathbf{A})\mathbf{r}^{(0)}.$$

Since

$$\mathbf{r}^{(k)}=R_k(\mathbf{A})\mathbf{r}^{(0)}\ ,$$

it follows that

$$C_{k-1}(\varsigma)=\frac{1-R_k(\varsigma)}{\varsigma}\ ,$$

which is a polynomial since $R_k(0)=1$.

Therefore if

$$R_k(\varsigma)=\theta_k\varsigma^k+\cdots+\theta_1\varsigma+1$$

then

$$C_{k-1}(\varsigma)=\theta_k\varsigma^{k-1}+\cdots+\theta_1. \tag{3.2.1}$$

The representation of any polynomial in ς in terms of powers of ς is called the *power form.*

3.3. A Remark on Polynomial Preconditioning. Assume that residual polynomial R_k is small on set Ω (containing the spectrum of **A**.) Therefore, on Ω, $C_{k-1}(\varsigma)$ is an approximation to ς^{-1}, and $C_{k-1}(\mathbf{A})$ is an approximation to $\mathbf{A}^{-1}$. Polynomial C_{k-1} arises in so-called polynomial preconditioning [Adm82, AMS87, Chen82, JMP83, Sayl83, Tal87].

3.4. Methods to Compute $\mathbf{C}_{k-1}(\mathbf{A})\mathbf{r}^{(0)}$. In the important Chebyshev case,

$$R_k(\varsigma) = \frac{T_k\left(\frac{d-\varsigma}{c}\right)}{T_k\left(\frac{d}{c}\right)}.$$

The coefficients, θ_i, could be easily determined by expanding $T_k\left(\frac{d-\varsigma}{c}\right)$ in terms of powers of ς. In principle, the coefficients of any residual polynomial could be determined in the same way, although no residual polynomial is as well documented as the Chebyshev case.

Despite the simplicity of this approach, it has the unfavorable feature that even when the coefficients, θ_i, are known explicitly, it is numerically difficult to compute the vector $\mathbf{d} = \theta_k \mathbf{A}^{k-1}\mathbf{r}^{(0)} + \cdots + \theta_1 \mathbf{r}^{(0)}$ due to the ill conditioning of the basis $\{\mathbf{r}^{(0)}, \cdots, \mathbf{A}^{k-1}\mathbf{r}^{(0)}\}$, if k is large. However, to avoid instability it is often sufficient to take a small value of k, say $k = 5$. If the power form coefficients are known then nested' polynomial evaluation (Horner's rule) could be used to

compute $C_{k-1}(\mathbf{A})\mathbf{r}^{(0)}$.

An algorithm is presented later, Algorithm 2, in which the roots, $\{\sigma_i : i = 1, ..., k-1\}$, and the leading coefficient, g_{k-1}, of C_{k-1} will be assumed given. The roots of C_{k-1} may be computed from the power form (for example by computing the eigenvalues of the companion matrix.) A method and algorithm (Algorithm 5) are presented in §6 that do not require the power form coefficients. Also see [Tal87] for a non–L_2 approach.

3.4.1. Avoiding Complex Arithmetic. If $\mathbf{A}$ is real, then it is reasonable to assume that the coefficients of C_{k-1} are real. If so, then the roots of C_{k-1} occur in complex conjugate pairs. Let σ and $\overline{\sigma}$ be a conjugate pair. Since

$$(\mathbf{A}-\sigma)(\mathbf{A}-\overline{\sigma})\mathbf{u} = \left(\mathbf{A}^2 - (\sigma+\overline{\sigma})\mathbf{A} + |\sigma|^2\right)\mathbf{u}\,,$$

no complex arithmetic is required to evaluate $C_{k-1}(\mathbf{A})\mathbf{r}^{(0)}$ when the factored form is used. (In the general case when $\mathbf{A}$ is complex, the roots do not occur in conjugate pairs.)

3.5. Algorithm for the Grand–Leap.

Algorithm 2. (Compute $\mathbf{x}^{(k)} = \mathbf{x}^{(0)} + C_{k-1}(\mathbf{A})\mathbf{r}^{(0)}$.*)*

Purpose. Compute the final iterate $\mathbf{x}^{(k)}$ from the initial guess with no intermediate iterates computed, except those at the end of each cycle.

Input. Matrix **A**, right side **b**, the intitial guess $\mathbf{x}^{(0)}$, period k, the leading coefficient, g_{k-1} and the roots σ_1, ..., σ_{k-1} of C_{k-1}, i.e., $C_{k-1}(\varsigma) = g_{k-1}\prod_{j=1}^{k-1}(\varsigma - \sigma_j)$. Parameter τ_0, which is the reciprocal of the root of R_1, is required if $k = 1$. These parameters are generated from Algorithm 5, and, in the Chebyshev case, from Algorithm 6. However, there are other sources for the parameters such as [Tal87]. The user must also provide a maximum number of cycles of iterations, and an error criterion to halt the iteration.

Output. Iterate $\mathbf{x}^{(k)}$, the last iterate reached after a cycle of k parameters in the standard execution of Richardson's method.

Restrictions. The algorithm executes with no restrictions on the input data. However in order for the algorithm to converge to the solution of $\mathbf{Ax} = \mathbf{b}$, for all **b**, it is necessary that $|\ R_k(\lambda_i)| = |\ 1 - \lambda_i C_{k-1}(\lambda_i)| < 1$ for each nonzero eigenvalue, λ_i, of **A**. If this holds and **A** is singular, the algorithm converges to a solution when the system is consistent.

1) Set $\mathbf{r}^{(0)} := \mathbf{b} - \mathbf{A}\mathbf{x}^{(0)}$.

2) If $k = 1$ then

2.1) Set $\mathbf{x}^{(1)} := \mathbf{x}^{(0)} + \tau_1 \mathbf{r}^{(0)}$.

2.2) Return.

3) Separate the roots $\{\sigma_i\}$ of C_{k-1} into real roots and conjugate pairs of (nonreal) roots: σ_1, ..., σ_m are real; σ_{m+1}, ..., σ_{k-1} are nonreal and $\sigma_{j+1}=\overline{\sigma}_j, m+1\leq j\leq k-2$.

4) Do until convergence or a limit on the number of loops is exceeded:

4.1) Set $\mathbf{r}^{(0)}:=\mathbf{b}-\mathbf{A}\mathbf{x}^{(0)}$.

4.2) Set

$$\mathbf{x}^{(k)} := \mathbf{x}^{(0)}+g_{k-1}\prod_{j=m+1}^{k-1}\left[\mathbf{A}\left(\mathbf{A}-(\sigma_j+\sigma_{j+1})\right)+\sigma_j\sigma_{j+1}\right]\times\prod_{j=1}^{m}(\mathbf{A}-\sigma_j)\mathbf{r}^{(0)} .$$

4.3) If not converged set $\mathbf{x}^{(0)}:=\mathbf{x}^{(k)}$

Enddo.

4. Comparisons.

To display some of the advantages in the leapfrog and grand-leap approach a side-by-side comparison of algorithms is made in Table 1. The conventional Richardson's method is compared to the leapfrog version, in which alternate steps are omitted, and to the grand-leap version in which all intermediate steps have been omitted. The period, k, is assumed even.

The operations shown in Table 1 form the kernel of a loop, the commands for which have been omitted. The details for a complete algorithm have been given already and would be distracting here.

On any computer, reducing the number of arithmetic operations, the traditional goal of algorithm design, is an advantage. In the case of the leapfrog and grand-leap versions, it is a thin advantage but an advantage nevertheless and one that is unexpected. (Since Richardson's method is a Krylov subspace method, the number of matrix-vector multiplications cannot be reduced.) It is a further advantage on supercomputers that do chaining that there are more terms in the leapfrog expression for $\mathbf{x}^{(i)}$ than in the conventional expression.

Some additional comment is needed on how arithmetic operations are counted. The number of arithmetic operations given in the table is based on the assumption that there is no mixed real and complex arithmetic. Let us consider when mixed arithmetic occurs. The Table 1 parameters are $\{\tau_i, \sigma_i, \tau_i + \overline{\tau}_i, \tau_i\overline{\tau}_i, \sigma_i + \overline{\sigma}_i, \sigma_i\overline{\sigma}_i\}$. In the Hermitian symmetric positive definite

case, the roots of R_k are real, the Table 1 parameters are real, and there is no mixed arithmetic. (It is reasonable to assume this. A Hermitian positive definite system could be solved with nonreal parameters, however.) If **A** is a general complex nonsymmetric matrix, all Table 1 parameters are general complex quantities and since the matrix is complex, there is again no mixed arithmetic.

Mixed arithmetic occurs in Richardson's method if **A** is real and nonsymmetric, for then the Table 1 parameters are general, complex quantities, whereas the other quantities are real. If **A** is real, it is reasonable to assume that polynomials R_k and C_{k-1} are real. The roots may then be grouped in conjugate pairs, and the leapfrog and grand-leap methods performed in real arithmetic. Richardson's method, however, requires complex arithmetic, and the number of arithmetic operations is effectively larger than shown in Table 1. In this case, one would not want to consider Richardson's method, which was the motive for using the leapfrog method in [SmSa85].

Now we come to an aspect of these comparisons, namely the effect on I/O due to the solution of large systems, that is important to take into account but is necessarily limited due to the range of the subject.

The limitation made here is to consider only programmer-controlled storage, from among a list of topics required for a more complete discussion that includes architectures, specific application problems, and implementation details. The reader may object that although it is reasonable to dismiss architectures, it is still

not reasonable to restrict discussion in quite this way. For, the typical user is running problems on a virtual memory machine and is beset with multiple worries that deserve attention, such as memory "touches", or the loading of vector registers, or the losses due to flushing a cache. Unfortunately, such transfers between memory levels are hardware dependent and simply cannot be analyzed within the scope of this paper; the conclusions reached here below do not necessarily hold in these cases. It should be noted, however, that even for virtual memory systems, there exist limits [Eccl83] that compel the use of explicit I/O commands similar to those in Table 1.

One final comment to justify the narrow focus that is taken: It is characteristic of many supercomputers that only programmer-controlled peripheral storage is available for large problems and when needed is usually responsible for languid performance. This dismal fact often attracts comment. For example, Ortega and Voigt observe, "The [programmer-controlled] I/O problem produced by very large problems [is]... known to be potentially devastating on high performance systems ... ." [OrVo85]. Programmer-controlled storage includes system commands, custom utilities, and the less efficient choice, depending on circumstances, of Fortran commands. Only Fortran commands are given in Table 1.

In order to weigh the effect of transfers from peripheral storage, a large set of linear equations is assumed. This vague statement will be sharpened in order to

arrive at a rather specific assumption. The discretization of coupled partial differential equations in three dimensions yields, in some applications, leviathan systems of order ten million complex unknowns. Such problems lead to the assumption that a matrix multiplication, which may involve a preconditioning, absorbs the primary memory and that processing after a matrix multiplication requires reading in a vector from disk. This assumption is seen in Table 1 when, for example, in the leapfrog algorithm, $\mathbf{r}^{(i-2)}$ must be read from disk after computing $\mathbf{t} = \mathbf{A}\mathbf{r}^{(i-2)}$.

Under these conditions, a third advantage of the leapfrog and grand-leap algorithms is seen: there are fewer READ's and WRITE's.

In an actual implementation, it may well happen, for example, that two matrix READ's are not necessary in any algorithm and that $\mathbf{x}^{(i-1)}$ in the conventional algorithm need not be written on disk. Conditions will vary, and the results in the table are only representative. If the assumption on matrix vector multiplications is not valid, the comparisons would change, but it is plausible that for large problems there would remain an I/O advantage to the leapfrog and the grand-leap formulations.

Table 1

Conventional	Leapfrog	Grand–Leap
	$\alpha=\tau_{i-2}+\tau_{i-1}$ $\nu=\tau_{i-2}\tau_{i-1}$	$\alpha=\sigma+\bar{\sigma}$ $\nu=\sigma\bar{\sigma}$
READ **A** $\mathbf{v}=\mathbf{A}\mathbf{x}^{(i-2)}$	READ **A** $\mathbf{v}=\mathbf{A}\mathbf{x}^{(i-2)}$	READ **A** $\mathbf{v}_1=\mathbf{A}\mathbf{u}^{(i-2)}-\alpha\mathbf{u}^{(i-2)}$
READ **b** $\mathbf{r}^{(i-2)}=\mathbf{b}-\mathbf{v}$	READ **b** $\mathbf{r}^{(i-2)}=\mathbf{b}-\mathbf{v}$	READ **A** $\mathbf{v}_2=\mathbf{A}\mathbf{v}_1$
READ $\mathbf{x}^{(i-2)}$ $\mathbf{x}^{(i-1)}=\mathbf{x}^{(i-2)}+\tau_{i-2}\mathbf{r}^{(i-2)}$ WRITE $\mathbf{x}^{(i-1)}$	READ **A** $\mathbf{t}=\mathbf{A}\mathbf{r}^{(i-2)}$	
READ **A** $\mathbf{v}=\mathbf{A}\mathbf{x}^{(i-1)}$		
READ **b** $\mathbf{r}^{(i-1)}=\mathbf{b}-\mathbf{v}$	READ $\mathbf{r}^{(i-2)}$	
READ $\mathbf{x}^{(i-1)}$ $\mathbf{x}^{(i)}=\mathbf{x}^{(i-1)}+\tau_{i-1}\mathbf{r}^{(i-1)}$ WRITE $\mathbf{x}^{(i)}$	READ $\mathbf{x}^{(i-2)}$ $\mathbf{x}^{(i)}=\mathbf{x}^{(i-2)}+\alpha\mathbf{r}^{(i-2)}-\nu\mathbf{t}$ WRITE $\mathbf{x}^{(i)}$	READ $\mathbf{u}^{(i-2)}$ $\mathbf{u}^{(i)}=\mathbf{v}_2+\nu\mathbf{u}^{(i-2)}$ WRITE $\mathbf{u}^{(i)}$
2 matrix mults.	2 matrix mults.	2 matrix mults.
4 vector READ's 2 matrix READ's 2 vector WRITE's	3 vector READ's 2 matrix READ's 1 vector WRITE	1 vector READ 2 matrix READ's 1 vector WRITE
4N adds 2N mults.	3N adds 2N mults.	2N adds 2N mults.

5. Second Order Iterations.

In the real eigenvalue case, residual polynomials of practical value are orthogonal polynomials, and satisfy a three term recursion. This elegant property yields *second order methods*, which have an extra term in the expression for the new iterate as compared to Richardson's method. Richardson's method is also called a *first order* method and a second order method sometimes called Richardson's second order method. In a second order method, each new iterate is optimum in the sense that the residual polynomial satisfies an L_2–optimality property, to be be discussed in §6. The Chebyshev iteration, employed by Manteuffel [Mnt77], is an example. In the case of a first order method, $\mathbf{x}^{(k)}$ is optimum if the residual polynomial, R_k, is optimum, but $\mathbf{x}^{(i)}$ is not optimum for $i \neq k$, a fact commented on previously in the Notes for *Algorithm* 1. There is a cost in the second order method for optimality at each step: a larger number of arithmetic operations and a greater use of storage compared to Richardson's method.

The objective is a second order method for which only even numbered iterates are computed using information only at even numbered steps, a method hereafter called a *second order leapfrog* method. In the Chebyshev case, an algorithm will be given.

5.1. The Second Order Iteration. Some preliminaries are needed. Let $\mathbf{x}^{(0)}$ be the given initial guess. Define $\Delta\mathbf{x}^{(-1)}$ to be zero and for $0 \leq k$,

$$\Delta\mathbf{x}^{(k)} = \mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}.$$

The second order iteration requires a set of parameters $\{\alpha_k, \gamma_k : 1 \leq k\}$ that are given explicitly in the Chebyshev case in Algorithm 3, and derived in a general way in Algorithm 4. Assume these parameters are given. The iteration may now be stated. Let $\mathbf{r}^{(0)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(0)}$. For $k \geq 1$,

$$\Delta\mathbf{x}^{(k-1)} = \gamma_k \Delta\mathbf{x}^{(k-2)} + \alpha_k \mathbf{r}^{(k-1)}, \tag{5.1.1}$$

$$\mathbf{x}^{(k)} = \mathbf{x}^{(k-1)} + \Delta\mathbf{x}^{(k-1)},$$

and

$$\mathbf{r}^{(k)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(k)}.$$

5.2. Second Order Leapfrog. The derivation is somewhat lengthier than in the case of Richardson's method due to: the need to express $\Delta\mathbf{x}^{(k)}$ in terms of information at step $k-2$; and a complication involving the residual vector.

First, an expression for $\mathbf{x}^{(k)}$, $k \geq 2$, is obtained in terms of information at step $k-2$. Since

$$\mathbf{x}^{(k)} = \mathbf{x}^{(k-1)} + \Delta\mathbf{x}^{(k-1)}$$

it follows that

$$\mathbf{x}^{(k)} = \mathbf{x}^{(k-2)} + \Delta\mathbf{x}^{(k-2)} + \Delta\mathbf{x}^{(k-1)} .$$

Now use (5.1.1) in the last equation to obtain

$$\mathbf{x}^{(k)} = \mathbf{x}^{(k-2)} + \Delta\mathbf{x}^{(k-2)} + \gamma_k \Delta\mathbf{x}^{(k-2)} + \alpha_k \mathbf{r}^{(k-1)} .$$

Define

$$\Delta\mathbf{r}^{(k-2)} = \mathbf{r}^{(k-1)} - \mathbf{r}^{(k-2)} .$$

Then

$$\mathbf{x}^{(k)} = \mathbf{x}^{(k-2)} + \Delta\mathbf{x}^{(k-2)} + \gamma_k \Delta\mathbf{x}^{(k-2)} + \alpha_k \left(\mathbf{r}^{(k-2)} + \Delta\mathbf{r}^{(k-2)} \right) .$$

It remains to express $\Delta\mathbf{r}^{(k)}$ and $\Delta\mathbf{x}^{(k)}$ in terms of information at step $k-2$. The expression for $\Delta\mathbf{r}^{(k)}$ is simply

$$\Delta\mathbf{r}^{(k)} = -\mathbf{A}\Delta\mathbf{x}^{(k)} .$$

Finally, to obtain $\Delta\mathbf{x}^{(k)}$,

$$\Delta\mathbf{x}^{(k)} = \alpha_{k+1}\mathbf{r}^{(k)} + \gamma_{k+1}\Delta\mathbf{x}^{(k-1)}$$

$$= \alpha_{k+1}\mathbf{r}^{(k)} + \gamma_{k+1}\left[\alpha_k \left(\mathbf{r}^{(k-2)} + \Delta\mathbf{r}^{(k-2)} \right) + \gamma_k \Delta\mathbf{x}^{(k-2)} \right] .$$

To summarize, the formulas to go from step $k-2$ to step k are

$$\mathbf{x}^{(k)} = \mathbf{x}^{(k-2)} + \left[\alpha_k \left(\mathbf{r}^{(k-2)} + \Delta\mathbf{r}^{(k-2)}\right) + \gamma_k \Delta\mathbf{x}^{(k-2)}\right] + \Delta\mathbf{x}^{(k-2)},$$

$$\mathbf{r}^{(k)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(k)},$$

$$\Delta\mathbf{x}^{(k)} = \alpha_{k+1}\mathbf{r}^{(k)} + \gamma_{k+1}\left[\alpha_k \left(\mathbf{r}^{(k-2)} + \Delta\mathbf{r}^{(k-2)}\right) + \gamma_k \Delta\mathbf{x}^{(k-2)}\right],$$

and

$$\Delta\mathbf{r}^{(k)} = -\mathbf{A}\Delta\mathbf{x}^{(k)}.$$

Initially,

$$\mathbf{x}^{(0)} = \text{given},$$

$$\mathbf{r}^{(0)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(0)},$$

$$\Delta\mathbf{x}^{(0)} = \alpha_1 \mathbf{r}^{(0)},$$

and

$$\Delta\mathbf{r}^{(0)} = -\mathbf{A}\Delta\mathbf{x}^{(0)}.$$

Table 2

Conventional Second Order	Leapfrog Second Order
	READ $\Delta\mathbf{x}^{(k-2)}$
	READ $\mathbf{r}^{(k-2)}$
READ $\mathbf{x}^{(k-2)}$	READ $\mathbf{x}^{(k-2)}$
$\Delta\mathbf{x}^{(k-2)}=\gamma_{k-1}\Delta\mathbf{x}^{(k-3)}+\alpha_{k-1}\mathbf{r}^{(k-2)}$	$\mathbf{w} = \alpha_k \left(\mathbf{r}^{(k-2)} + \Delta\mathbf{r}^{(k-2)}\right) + \gamma_k \Delta\mathbf{x}^{(k-2)}$
WRITE $\Delta\mathbf{x}^{(k-2)}$	WRITE $\mathbf{w}$
$\mathbf{x}^{(k-1)}=\mathbf{x}^{(k-2)}+\Delta\mathbf{x}^{(k-2)}$ WRITE $\mathbf{x}^{(k-1)}$	$\mathbf{x}^{(k)} = \mathbf{x}^{(k-2)} + \mathbf{w} + \Delta\mathbf{x}^{(k-2)}$ WRITE $\mathbf{x}^{(k)}$
READ $\mathbf{A}$ $\mathbf{v}=\mathbf{A}\mathbf{x}^{(k-1)}$ READ $\mathbf{b}$ $\mathbf{r}^{(k-1)}=\mathbf{b}-\mathbf{v}$	READ $\mathbf{A}$ $\mathbf{v}=\mathbf{A}\mathbf{x}^{(k)}$ READ $\mathbf{b}$ $\mathbf{r}^{(k)}=\mathbf{b}-\mathbf{v}$
READ $\Delta\mathbf{x}^{(k-2)}$ $\Delta\mathbf{x}^{(k-1)}=\gamma_k\Delta\mathbf{x}^{(k-2)}+\alpha_k\mathbf{r}^{(k-1)}$ WRITE $\Delta\mathbf{x}^{(k-1)}$	READ $\mathbf{w}$ $\Delta\mathbf{x}^{(k)} = \alpha_{k+1}\mathbf{r}^{(k)} + \gamma_{k+1}\mathbf{w}$ WRITE $\Delta\mathbf{x}^{(k)}$
READ $\mathbf{x}^{(k-1)}$ $\mathbf{x}^{(k)} = \mathbf{x}^{(k-1)} + \Delta\mathbf{x}^{(k-1)}$ WRITE $\mathbf{x}^{(k)}$	WRITE $\mathbf{r}^{(k)}$
READ $\mathbf{A}$ $\mathbf{v} = \mathbf{A}\mathbf{x}^{(k)}$	READ $\mathbf{A}$ $\Delta\mathbf{r}^{(k)} = -\mathbf{A}\Delta\mathbf{x}^{(k)}$
READ $\mathbf{b}$ $\mathbf{r}^{(k)} = \mathbf{b} - \mathbf{v}$	
2 matrix mults.	2 matrix mults.
6 vector READ's 2 matrix READ's 4 vector WRITE's	5 vector READ's 2 matrix READ's 4 vector WRITE's
6N adds 4N mults.	6N adds 4N mults.

5.3. Comparisons. Under the same assumptions as for the previous set of comparisons, the two versions of the second order iteration are compared in Table 2. There are fewer advantages of the leapfrog algorithm in this case since the number of arithmetic operations and the number of WRITE's is the same. The advantages are that there are fewer READ's and a greater number of terms in the sum defining $\mathbf{x}^{(k)}$ in the leapfrog version. Note that variations are possible, for example, in recomputing $\mathbf{w}$ in the leapfrog version, and that the arrangement of terms used here is not necessarily suitable for a particular problem or architecture.

5.4. Algorithm for the Second Order Leapfrog Method in the Chebyshev Case. For the convenience of the reader, an algorithm is given below for the case of Chebyshev parameters. As before with Algorithms 1 and 2, no attempt is made to incorporate I/O statements.

Algorithm 3. (Second order leapfrog iteration with Chebyshev parameters.)

Purpose. Execute the leapfrog form of the second order iteration for the Chebyshev case. The parameters are the same as for the standard second order Chebyshev iteration as used for example in the Manteuffel algorithm [Mnt78, Ashb85].

Input. Matrix $\mathbf{A}$, right side $\mathbf{b}$, and initial guess $\mathbf{x}^{(0)}$; also a pair d and c such that d is the center and $d \pm c$ are the foci of a family of ellipses over which the

Chebyshev residual polynomial is (nearly) minimum with respect to the uniform norm. The ellipse parameters are assumed known, for example, as output from the Manteuffel algorithm [Ashb85]. In the general non–Chebyshev case, this algorithm could be easily modified to allow $\{\alpha_k\}$ and $\{\gamma_k\}$ to be input parameters, say, from Algorithm 4.

Output. The algorithm generates a set of optimum iterates converging to the solution of $\mathbf{Ax} = \mathbf{b}$ if the restrictions are satisfied.

Restrictions. The restrictions are the same as for Algorithm 1.

Notes. The α_k and γ_k parameters need not be array variables; the subscripts aid clarity.

1) Set $\mathbf{r}^{(0)} := \mathbf{b} - \mathbf{A}\mathbf{x}^{(0)}$.

2) Set $\alpha_1 := 1/d$.

3) Set $\Delta\mathbf{x}^{(0)} := \alpha_1\mathbf{r}^{(0)}$.

4) Set $\Delta\mathbf{r}^{(0)} := -\mathbf{A}\Delta\mathbf{x}^{(0)}$.

5) Set $\alpha_2 := \dfrac{2d}{2d^2 - c^2}$.

6) Set $\gamma_2 = d\,\alpha_2 - 1$.

7) Do $k = 2$ by 2 until either convergence or a limit is exceeded:

7.1) Set $\mathbf{w} := \alpha_k \left(\mathbf{r}^{(k-2)} + \Delta \mathbf{r}^{(k-2)} \right) + \gamma_k \Delta \mathbf{x}^{(k-2)}$.

7.2) Set $\mathbf{x}^{(k)} := \mathbf{x}^{(k-2)} + \mathbf{w} + \Delta \mathbf{x}^{(k-2)}$.

7.3) Set $\mathbf{r}^{(k)} := \mathbf{b} - \mathbf{A}\mathbf{x}^{(k)}$.

7.4) Set $\alpha_{k+1} := \dfrac{1}{d - \left(c^2 \alpha_k / 4 \right)}$.

7.5) Set $\gamma_{k+1} := d\,\alpha_{k+1} - 1$.

7.6) Set $\alpha_{k+2} := \dfrac{1}{d - \left(c^2 \alpha_{k+1} / 4 \right)}$.

7.7) Set $\gamma_{k+2} := d\,\alpha_{k+2} - 1$.

7.8) Set $\Delta \mathbf{x}^{(k)} := \alpha_{k+1} \mathbf{r}^{(k)} + \gamma_{k+1} \mathbf{w}$.

Enddo

6. L_2-Optimum Parameters.

If either the L_2- or l_2-norm is used to define optimum residual polynomials, then it turns out that optimum residual polynomials form a family of orthogonal polynomials if the inner product (either integral or sum) is defined over a real set. From this fact, algorithms follow for the computation of the τ-parameters for Richardson's method, the σ-parameters for the grand-leap method, and the parameters for the second order method, which are presented in this section. The assumption that the inner product is defined over a real set usually means that the eigenvalues of the system matrix are real. The Chebyshev case is an exception for which the eigenvalues need not be real. (Since Chebyshev polynomials form an orthogonal family, Chebyshev residual polynomials are L_2-optimum as well as L_∞-optimum.)

The algorithms in this section generalize to the case of an inner product defined over a contour in the complex plane. (A generalization may be based on [SaSm88].)

6.1. L_2-Optimality. Some notation is necessary. Let Γ be an interval or a union of intervals on the real line (generally, Γ could be any measurable subset) and let w be a positive weight function on Γ. Define

$$(f\,,\,g)_w = \frac{1}{L}\int_\Gamma f(\xi)g(\xi)w(\xi)d\xi$$

where $L = \int_{\Gamma} w(\xi) d\xi$. The set Γ may be assumed to be real by a linear change of variables if necessary. In practice, rather than the continuous inner product, one would use a discrete inner product of the form

$$(f, g)_w = \frac{1}{M} \sum_{i=1}^{M} f(\xi_i) g(\xi_i) w(\xi_i) m(\xi_i),$$

where $m(\xi)$ is a measure, such as $|\xi_i - \xi_{i-1}|$. A norm is defined by

$$\| f \|_w^2 = (f, f)_w .$$

An (L_2-) *optimum residual polynomial* of degree k is defined to be that residual polynomial, R_k, with the smallest norm,

$$\| R_k \|_w^2 \leq \| P_k \|_w^2 ,$$

where P_k is any residual polynomial of degree k. Ideally, Γ should contain the spectrum of **A**, and conform to the spectrum as closely as possible. Thus if the spectrum were contained in the union of two intervals, Γ should also be the union of, if possible, the same two intervals. How to find the interval or union of intervals containing the spectrum is a difficult problem, and is not considered here; the reader is referred to the Manteuffel algorithm [Mnt78] (which, however, computes only one interval containing the spectrum.) If $\{R_i\}$ is a set of optimum residual polynomials, then [Stie58] they form an orthogonal set with respect to the modified weight function, $\xi w(\xi)$:

$$(R_i, R_j)_{\xi w} = 0 \qquad (6.1.1)$$

if and only if $i \neq j$.

6.2. The Recursive Property of Orthogonal Polynomials. The well-known three term recursion for orthogonal polynomials is recalled, a property that yields not only a second order iteration, which is derived here, but, also in §7.1, an algorithm for computing, among other things, the roots of the optimum residual polynomial, needed in order to execute Richardson's method.

Define ϕ_{-1} to be zero, and let ϕ_0 be a nonzero constant. A family, $\{\phi_k : 0 \leq k\}$, of orthogonal polynomials satisfies a three term recursion: for $1 \leq k$,

$$\phi_k(\xi) = (\alpha_k \xi + \beta_k)\phi_{k-1}(\xi) - \gamma_k \phi_{k-2}(\xi) , \tag{6.2.1}$$

where $\alpha_k, \beta_k, \gamma_k$ are *recursion coefficients* given by

$$\frac{\beta_k}{\alpha_k} = \frac{-(\xi\phi_{k-1}, \phi_{k-1})_w}{\| \phi_{k-1} \|_w^2} ,$$

$$\frac{\gamma_k}{\alpha_k} = \frac{(\xi\phi_{k-1}, \phi_{k-2})_w}{\| \phi_{k-2} \|_w^2} .$$

One coefficient is a parameter that allows a normalization, such as, for example, $\| \phi_k \|_w = 1$. If $\{\phi_k : 0 \leq k\}$ is a family of residual polynomials, the desired normalization is $\phi_k(0) = 1$, which yields $\beta_k = \gamma_k + 1$ and, with $R_k(\xi) = \phi_k(\xi)$,

$$R_k(\xi)=(\alpha_k\xi+\gamma_k+1)R_{k-1}(\xi)-\gamma_k R_{k-2}(\xi)\ . \tag{6.2.2}$$

6.3. Second Order Iteration. The recursion for the residual polynomials yields an iteration for which $\mathbf{x}^{(k)}$ is L_2–optimum in the following sense: The error, $\mathbf{e}^{(k)}=\mathbf{x}-\mathbf{x}^{(k)}$, satisfies $\mathbf{e}^{(k)}=R_k(\mathbf{A})\mathbf{e}^{(0)}$, where R_k is an L_2–optimum residual polynomial.

To derive the iteration, replace ξ with $\mathbf{A}$ in (6.2.2) and multiply on the right by $\mathbf{r}^{(0)}$ to get[Stie58], for $1\leq k$ and $\mathbf{r}^{(-1)}$ defined to be zero,

$$\mathbf{r}^{(k)}=(1+\gamma_k)\mathbf{r}^{(k-1)}+\alpha_k\mathbf{A}\mathbf{r}^{(k-1)}-\gamma_k\mathbf{r}^{(k-2)}.$$

Replace $\mathbf{r}^{(j)}$ by $\mathbf{b}-\mathbf{A}\mathbf{x}^{(j)}$, $j=k-2,k-1,k$, and multiply on the left by $\mathbf{A}^{-1}$ to obtain

$$\mathbf{x}^{(k)}=(1+\gamma_k)\mathbf{x}^{(k-1)}+\alpha_k\mathbf{r}^{(k-1)}-\gamma_k\mathbf{x}^{(k-2)}.$$

Initially, $\mathbf{x}^{(0)}$ is given and $\mathbf{r}^{(0)}=\mathbf{b}-\mathbf{A}\mathbf{x}^{(0)}$.

The iteration is usually expressed in terms of the iterant difference, $\mathbf{x}^{(k)}-\mathbf{x}^{(k-1)}$, as in (5.1.1).

6.4. A Method for the Roots of C_{k-1}. A matrix will be derived, the eigenvalues of which are the roots of C_{k-1}.

Recall from §6.1 that an optimum residual polynomial of degree k is defined to be that residual polynomial, R_k, that solves the weighted least squares problem

$$\int_{\Gamma} |\, R_k(\xi)|^{\,2} w(\xi) d\,\xi \leq \int_{\Gamma} |\, P_k(\xi)|^{\,2} w(\xi) d\,\xi\ ,$$

where P_k is any residual polynomial of degree k. Also if $\{R_i\}$ is a set of optimum residual polynomials, then they form an orthogonal set with respect to the modified weight function, $\xi w(\xi)$; see (6.1.1).

The roots of orthogonal polynomials may be computed by a stable algorithm based on the fact that the roots are the eigenvalues of a symmetric tridiagonal matrix, $\mathbf{S}_k$. The algorithm is called the *Stieltjes algorithm* and matrix $\mathbf{S}_k$ is called the *Jacobi matrix*. The Stieltjes algorithm is recommended for computing the optimum Richardson's method parameters, which are the reciprocals of the roots of the optimum residual polynomial. Matrix $\mathbf{S}_k$ may be modified by one element to obtain a matrix the nonzero eigenvalues of which are roots of C_{k-1}.

If only the roots of R_k and C_{k-1} are desired, it is not important that $R_k(0) = 1$. It is preferable to work with the normalized family

$$\left\{\phi_i(\xi) = \frac{R_i(\xi)}{\| R_i \|_{\xi w}}\right\}.$$

6.4.1. The Roots of $\mathbf{R}_k(\xi)$. The elements of matrix $\mathbf{S}_k$ are the coefficients of the three term recursion satisfied by $\{\phi_i\}$.

It is convenient to write the recursion (6.2.1) in the slightly different form

$$\xi\phi_{k-1}(\xi)=s_{k,k-1}\phi_{k-2}(\xi)+s_{kk}\phi_{k-1}(\xi)+s_{k,k+1}\phi_k(\xi).$$

The first three terms of the recursion are

$$\begin{aligned}
\xi\phi_0(\xi) &= s_{11}\phi_0(\xi) + s_{12}\phi_1(\xi) \\
\xi\phi_1(\xi) &= s_{21}\phi_0(\xi) + s_{22}\phi_1(\xi) + s_{23}\phi_2(\xi) \\
\xi\phi_2(\xi) &= s_{32}\phi_1(\xi) + s_{33}\phi_2(\xi) + s_{34}\phi_3(\xi),
\end{aligned}$$

which may be written in matrix form as

$$\xi\begin{bmatrix}\phi_0(\xi)\\ \phi_1(\xi)\\ \phi_2(\xi)\end{bmatrix}=\begin{bmatrix}s_{11} & s_{12} & 0\\ s_{21} & s_{22} & s_{23}\\ 0 & s_{32} & s_{33}\end{bmatrix}\begin{bmatrix}\phi_0(\xi)\\ \phi_1(\xi)\\ \phi_2(\xi)\end{bmatrix}+s_{34}\phi_3(\xi)\begin{bmatrix}0\\ 0\\ 1\end{bmatrix}.$$

In general, k terms yield the matrix–vector equation

$$\xi\boldsymbol{\phi}(\xi)=\mathbf{S}_k\boldsymbol{\phi}(\xi)+s_{k,k+1}\phi_k(\xi)\boldsymbol{\delta}_k\ ,$$

where

$$\boldsymbol{\phi}(\xi)=(\phi_0(\xi),\phi_1(\xi),\ \ldots,\ \phi_{k-1}(\xi))^T,$$

$\boldsymbol{\delta}_k = (0,\ \ldots,\ 0,\ 1,\ 0,\ \ldots,\ 0)^T$ is the k^{th} unit vector and $\mathbf{S}_k=(s_{ij})$ is the tridiagonal Jacobi matrix [Wilf62, GoWe69]. The eigenvalues of the Jacobi matrix coincide with the roots of ϕ_k.

Next, a nested procedure in which the eigenvalues of $\mathbf{S}_1,\ldots,\mathbf{S}_k$ are successively computed will be described for computing the roots of ϕ_k. Let $\{\rho_{ij}:1\le j\le i\}$ be the roots of ϕ_i; these are the eigenvalues of $\mathbf{S}_i$.

The procedure begins with the initial polynomial ϕ_0. Since ϕ_0 is a constant such that $\| \phi_0 \|_{\xi w} = 1$, it follows that

$$\phi_0(\xi) = \frac{1}{\| 1 \|_{\xi w}} .$$

Next, to compute the root of ϕ_1, it follows from

$$\xi\phi_0(\xi) = s_{11}\phi_0(\xi) + s_{12}\phi_1(\xi),$$

that, if ϕ_1 is to be orthogonal to ϕ_0,

$$\mathbf{S}_1 = (s_{11}) = ((\xi\phi_0, \phi_0)_{\xi w}) .$$

Of course, $\phi_1(s_{11}) = 0$.

Now assume $\mathbf{S}_{k-1}$ has been computed, $2 \leq k$. Since it is the $(k-1)\times(k-1)$ principal submatrix of $\mathbf{S}_k$, only the last row and column of $\mathbf{S}_k$ need be computed, a total of three nonzero elements. Since the polynomials are normalized, $\mathbf{S}_k$ may be proved to be symmetric. Hence only $s_{k-1,k}$ and s_{kk} are required.

Matrix $\mathbf{S}_{k-1}$ yields the roots $\rho_{1,k-1}, \ldots, \rho_{k-1,k-1}$ of ϕ_{k-1}. Let

$$\pi_{k-1}(\xi) = (\xi - \rho_{1,k-1}) \cdots (\xi - \rho_{k-1,k-1}).$$

Then

$$\phi_{k-1}(\xi) = \frac{\pi_{k-1}(\xi)}{\| \pi_{k-1} \|_{\xi w}} .$$

The elements to be computed are $s_{k,k-1}$, and s_{kk}. These are unknowns in the relations (with, of course, $s_{k-1,k} = s_{k,k-1}$)

$$\xi\phi_{k-2}(\xi) = s_{k-1,k-2}\phi_{k-3}(\xi) + s_{k-1,k-1}\phi_{k-2}(\xi) + s_{k-1,k}\phi_{k-1}(\xi)$$

and

$$\xi\phi_{k-1}(\xi) = s_{k,k-1}\phi_{k-2}(\xi) + s_{kk}\phi_{k-1}(\xi) + s_{k,k+1}\phi_k(\xi). \tag{6.4.1}$$

Orthogonality yields

$$s_{k-1,k} = (\xi\phi_{k-2},\ \phi_{k-1})_{\xi w} \tag{6.4.2}$$

and

$$s_{kk} = (\xi\phi_{k-1},\ \phi_{k-1})_{\xi w}. \tag{6.4.3}$$

This completes the computation of $\mathbf{S}_k$. An algorithm (Algorithm 4) is given in §7.

6.4.2. The Roots of $\mathbf{C}_{k-1}$. We have

$$\xi C_{k-1}(\xi) = 1 - R_k(\xi).$$

Since $R_k(\xi) = \dfrac{\phi_k(\xi)}{\phi_k(0)}$, it follows that

$$\phi_k(\xi)=\phi_0(\xi)[\phi_k(0)/\phi_0(\xi)]-\phi_k(0)\left(\xi C_{k-1}(\xi)\right).$$

Equation (6.4.1) therefore gives

$$\xi\phi_{k-1}(\xi)=\tilde{s}_{k1}\phi_0(\xi)+s_{k,k-1}\phi_{k-2}(\xi)+s_{kk}\phi_{k-1}(\xi)-s_{k,k+1}\phi_k(0)\left(\xi C_{k-1}(\xi)\right), \tag{6.4.4}$$

where $\tilde{s}_{k1}:=s_{k,k+1}\phi_k(0)/\phi_0(\xi)$. (Of course, $\phi_0(\xi)$ is a constant.) Define a lower Hessenburg matrix $\tilde{\mathbf{S}}_R=(\tilde{\mathbf{s}}_{ij})$ by setting $\tilde{s}_{ij}:=s_{ij}$ unless $i=k, j=1$, in which case $\tilde{s}_{k1}$ has been defined.

The equation

$$\xi\boldsymbol{\phi}(\xi)=\mathbf{S}_k\boldsymbol{\phi}(\xi)+s_{k,k+1}\phi_k(\xi)\boldsymbol{\delta}_k$$

now becomes

$$\xi\boldsymbol{\phi}(\xi)=\tilde{\mathbf{S}}_k\boldsymbol{\phi}(\xi)-s_{k,k+1}\phi_k(0)\left(\xi C_{k-1}(\xi)\right)\boldsymbol{\delta}_k\ .$$

The roots of $\xi C_{k-1}(\xi)$ are therefore the eigenvalues of $\tilde{\mathbf{S}}_k$.

6.5. Leading Coefficient $\mathbf{g}_{k-1}$. Preparations are nearly complete for the computation of

$$\mathbf{x}^{(k)}=\mathbf{x}^{(0)}+C_{k-1}(\mathbf{A})\mathbf{r}^{(0)}\ .$$

There is one remaining detail, an expression for the leading coefficient g_{k-1} such that

$$C_{k-1}(\xi)=g_{k-1}(\xi-\sigma_1)\cdots(\xi-\sigma_{k-1}).$$

Recall that

$$\xi C_{k-1}(\xi)=1-R_k(\xi),$$

$$R_k(\xi)=\phi_k(\xi)/\phi_k(0),$$

$$\phi_k(\xi)=\frac{\pi_k(\xi)}{\|\pi_k\|_{\xi w}},$$

and

$$\pi_k(\xi)=(\xi-\rho_{1k})\cdots(\xi-\rho_{kk}).$$

Therefore,

$$g_{k-1}=\frac{-1}{\|\pi_k\|_{\xi w}\phi_k(0)}.$$

In the Chebyshev case, an alternative formula is given in the next section.

7. Algorithms.

In this section, algorithms are given for the root finding algorithms for the general l_2-case and also for the Chebyshev case.

7.1. Algorithm for Normalized Residual Polynomials.

Algorithm 4. (Compute the recursion coefficients of a specified orthogonal family, and the roots and leading coefficients of the degree k orthogonal polynomial ϕ_k of the family.)

Purpose: Generate the factored form of successive normalized (real) orthogonal polynomials, ϕ_i, $i = 0, ..., k$; and the residual polynomial recursion parameters, $\{\alpha_k, \gamma_k\}$. Additional output is described below. If polynomials are optimum with respect to a weight function w, they are then orthogonal with respect to the (real) weight function $\xi w(\xi)$, and this will be the weight function used below. Note that $R_k(\xi) = \phi_k(\xi)/\phi_k(0)$ is a residual polynomial.

Input. A subprogram must be provided to compute an inner product $(f,g) = \frac{1}{L}\int_\Gamma f(\xi)g(\xi)\xi w(\xi)d\xi$ where $L = \int_\Gamma d\xi$. In practice, this subprogram would compute a discrete inner product $(f,g) = \frac{1}{M}\sum_{i=1}^{M} f(\xi_i)g(\xi_i)\xi_i w(\xi_i)$. Input to the subprogram would be the number, M, of nodes, the nodes $\xi_i, i = 1, ..., M$, and the weight, $w(\xi_i)$ (an array or external function).

Input to Algorithm 4 then consists of input to the subprogram, the subprogram itself, and the degree, k, of the highest degree normalized orthogonal polynomial.

Output. The algorithm generates (1) a two dimensional array of roots, $\{\rho_{ji}\}$, of ϕ_i, $0 < i \leq k$, required for (a modification of) Algorithm 1 in the non-Chebyshev case; (2) parameter $\tau_1 = \dfrac{1}{\rho_{11}}$, needed for Algorithm 2 in the special case $k = 1$; (3) the Jacobi matrix $\mathbf{S}_{\phi_k}$; (4) ϕ_0, and $\phi_k(0)$, needed for Algorithm 5; (5) the recursion coefficients $\{\alpha_k, \gamma_k\}$ for the residual polynomials, needed for (a modification of) Algorithm 3; and (6) the array of leading coefficients, ν_i, of ϕ_i, needed for Algorithm 5.

Restrictions. Degree k must satisfy $k \leq M$. The restriction on Γ is that the nodes, $\{\xi_i\}$, lie on the real line, which holds if $\mathbf{A}$ is Hermitian symmetric positive definite. However, it is not necessary that $\mathbf{A}$ be Hermitian symmetric positive definite. For example, if $\mathbf{A}$ is real nonsymmetric then Γ may be taken to be the major axis of an ellipse enclosing the nonzero spectrum and the inner product taken to be the inner product defining Chebyshev polynomials; see §7.3.

Notes. The reciprocals of the roots of ϕ_k are the τ-parameters needed for Richardson's method, and are general input for a modification of Algorithm 1.

This algorithm directly computes the recursion coefficients, $\{s_{ij}\}$, for normalized polynomials; see (6.4.1). These are not the recursion coefficients, $\{\alpha_k, \gamma_k\}$, for residual polynomials. An expression for the α_k, γ_k coefficients in terms of the s_{ij} coefficients is given as follows.

Since $R_i(\xi) = \phi_i(\xi)/\phi_i(0)$, (6.4.1) is equivalent to

$$R_k(\xi) = \frac{\phi_{k-1}(0)}{s_{k,k+1}\phi_k(0)}\xi R_{k-1}(\xi) - \frac{s_{kk}\phi_{k-1}(0)}{s_{k,k+1}\phi_k(0)}R_{k-1}(\xi) - \frac{s_{k,k-1}\phi_{k-2}(0)}{s_{k,k+1}\phi_k(0)}R_{k-2}(\xi) .$$

From (6.2.2), it follows that

$$\alpha_k = \frac{\phi_{k-1}(0)}{s_{k,k+1}\phi_k(0)} ,$$

$$\gamma_k = \frac{s_{k,k-1}\phi_{k-2}(0)}{s_{k,k+1}\phi_k(0)} .$$

For step 6.4) below, the monic polynomial, π_i, is used to calculate the leading coefficient of ϕ_i. Let the i roots of ϕ_i be $\{\rho_{ji}\}$. Define $\pi_i(\xi) = \prod_{j=1}^{i}(\xi - \rho_{ji})$. Let

$$\nu_i = \frac{1}{\| \pi_i \|_{\xi w}} .$$

Then $\phi_i(\xi) = \nu_i \pi_i(\xi)$.

The factored form $\phi_i(\xi) = \nu_i \prod_{j=1}^{i}(\xi - \rho_{ji})$ is recommended for evaluating $\phi_i(\xi)$.

1) Set $\nu_0 := \frac{1}{\| 1 \|_{\xi w}}$. Set $\phi_0 := \nu_0$.

2) Set $s_{11} := (\xi\phi_0, \phi_0)_{\xi w}$, and set $\rho_{11} := s_{11}$, the root of ϕ_1 .

3) If $k = 1$, return.

4) Set $\nu_1 := \dfrac{1}{\| \pi_1 \|_{\xi w}}$.

5) Set $\phi_1(0) := \nu_1(-\rho_{11})$.

6) For $2 \leq i \leq k$, do:

6.1) Set the first $i-2$ elements of the last column of $\mathbf{S}_{\phi_i}$, column i, equal to 0.

6.2) Formulas (6.4.2) and (6.4.3) give the remaining two elements. Set

$$s_{i-1,i} := (\xi\phi_{i-2}, \phi_{i-1})_{\xi w} ,$$

$$s_{ii} := (\xi\phi_{i-1}, \phi_{i-1})_{\xi w} .$$

6.3) Compute the eigenvalues of (the symmetric matrix) $\mathbf{S}_{\phi_i}$. Set the roots, $\{\rho_{ji}\}$, of ϕ_i to these eigenvalues.

6.4) Set $\nu_i := \dfrac{1}{\| \pi_i \|_{\xi w}}$.

6.5) Set $\phi_i(0) := (-1)^i \nu_i \rho_{1i} \cdots \rho_{ii}$.

6.6) Set $\alpha_i := \dfrac{\phi_{i-1}(0)}{s_{i,i+1}\phi_i(0)}$.

6.7) Set $\gamma_i := \dfrac{s_{i,i-1}\phi_{i-2}(0)}{s_{i,i+1}\phi_i(0)}$.

Enddo.

7.2. Algorithm for the Grand–Leap Parameters.

Algorithm 5. (Compute $\tilde{\mathbf{S}}_k$, and the roots and leading coefficient of C_{k-1}.)

Purpose. Compute the σ–parameters and parameter g_{k-1} needed for the grand–leap algorithm; these parameters are the roots and leading coefficient respectively of C_{k-1}.

Input. A matrix $(s_{ij})_{k+1\times k+1}$, such as $\mathbf{S}_{\phi_{k-1}}$ from Algorithm 4, nonzero parameters ν_k, ϕ_0, and $\phi_k(0)$, such as, also, from Algorithm 4, and period k.

Output. The algorithm generates the $k-1$ roots, $\{\sigma_i\}$, and leading coefficient g_{k-1} of the "polynomial preconditioner" C_{k-1}. These quantities become input for Algorithm 2.

Restrictions. There are no restrictions other than those imposed on the input parameters.

1) Initialize $k \geq 2$.

2) Set $\tilde{s}_{ij} := s_{ij}$ for $1 \leq i,\ j \leq k$ ($\tilde{s}_{k1}$ will be reset in step 3).)

3) Set $\tilde{s}_{k1} := s_{k,k+1}\phi_k(0)/\phi_0$.

4) Set $\tilde{\mathbf{S}}_k := (\tilde{s}_{ij})$.

5) Compute the eigenvalues of $\tilde{\mathbf{S}}_k$. Set the roots of C_{k-1} equal to the nonzero eigenvalues of $\tilde{\mathbf{S}}_k$.

6) Set $g_{k-1} := -\dfrac{\nu_k}{\phi_k(0)}$.

7.3. The Chebyshev Case. For this special case, the grand-leap parameters may be determined with explicit inner products.

7.3.1. Chebyshev Orthogonality. Assume $d, c \neq 0$, and 0 is not in the interval $[d-c, d+c]$. If d and c are real, this is equivalent to assuming $d > 0$, and $d - |c| > 0$. Let $T_i(\mu)$ be the Chebyshev polynomial of degree i defined by the familiar recursion $T_0 = 1, T_1(\mu) = \mu$, and for $1 \leq i$, $T_{i+1}(\mu) = 2\mu T_i(\mu) - T_{i-1}(\mu)$.

Let $\psi_i(\xi) = T_i[(\xi - d)/c]$ be the shifted and translated Chebyshev polynomial. The Chebyshev residual polynomial is therefore $\dfrac{\psi_i(\xi)}{\psi_i(0)}$. The family $\{\psi_i : 0 \leq i\}$ satisfies the orthogonality relations (where $c > 0$ if real)

$$\int_{d-c}^{d+c} \psi_i(\xi)\overline{\psi_j(\xi)}\xi w(\xi)\frac{d\xi}{c} = \begin{cases} \pi, i = j = 0 \\ \dfrac{\pi}{2}, i = j \neq 0 \\ 0, i \neq j \end{cases}$$

where

$$\xi w(\xi) = \left[\frac{c^2 - (\xi - d)^2}{c^2}\right]^{-1/2}$$

7.3.2. Recursions for the Shifted Chebyshev Polynomials. The recursion for T_i yields a recursion for $\{\psi_i\}$:

$$\xi\psi_0 = d\,\psi_0 + c\,\psi_1(\xi)$$

and for $1 \leq i$,

$$\xi\psi_i(\xi) = \frac{c}{2}\,\psi_{i-1}(\xi) + d\,\psi_i(\xi) + \frac{c}{2}\,\psi_{i+1}.$$

7.3.3. Roots of $\mathbf{C}_{k-1}$. The roots of C_{k-1} are among the eigenvalues of $\tilde{\mathbf{S}}_k$. An explicit expression for $\tilde{\mathbf{S}}_k$ requires $\mathbf{S}_k$ and $\phi_k(0)$. To determine these quantities, the three term recursion for the normalized polynomials is needed. The normalized polynomials are:

$$\left\{\phi_i(\xi) = \frac{\psi_i(\xi)}{\| \psi_i \|_{\xi w}}\right\}.$$

The recursion is

$$\phi_0 = \frac{1}{\sqrt{\pi}},$$

$$\xi\phi_0(\xi) = d\,\phi_0(\xi) + \frac{c}{\sqrt{2}}\,\phi_1(\xi),$$

$$\xi\phi_1(\xi) = \frac{c}{\sqrt{2}}\,\phi_0(\xi) + d\,\phi_1(\xi) + \frac{c}{2}\,\phi_2(\xi),$$

and for $2 \leq i$,

$$\xi\phi_i(\xi) = \frac{c}{2}\,\phi_{i-1}(\xi) + d\,\phi_i(\xi) + \frac{c}{2}\,\phi_{i+1}(\xi).$$

7.3.3.1. Matrix $\mathbf{S}_k$. From the recursion for $\{\phi_i\}$, it follows that

$$\mathbf{S}_k = \begin{bmatrix} d & \frac{c}{\sqrt{2}} & & & \\ \frac{c}{\sqrt{2}} & d & \frac{c}{2} & & \\ & \frac{c}{2} & d & \frac{c}{2} & \\ & & \cdot & \cdot & \cdot \\ & & & \cdot & \cdot \\ & & & & \cdot \end{bmatrix}.$$

To modify $\mathbf{S}_k$ to obtain $\tilde{\mathbf{S}}_k$, $\phi_k(0)$ is needed. For $1 \leq k$,

$$\phi_k(0) = \frac{\psi_k(0)}{\| \psi_k \|_{\xi w}} = \frac{\cosh k \cosh^{-1}\left(-\frac{d}{c}\right)}{\sqrt{\pi/2}}$$

which follows from $T_k(\mu) = \cosh k \cosh^{-1}(\mu)$.

7.3.3.2. Matrix $\tilde{\mathbf{S}}_{\mathbf{k-1}}$. Therefore,

$$\tilde{\mathbf{S}}_k = \begin{bmatrix} d & \frac{c}{\sqrt{2}} & & & & & & & \\ \frac{c}{\sqrt{2}} & d & \frac{c}{2} & & & & & & \\ 0 & \frac{c}{2} & d & \frac{c}{2} & & & & & \\ \cdot & & \cdot & \cdot & \cdot & & & & \\ \cdot & & & \cdot & \cdot & \cdot & & & \\ \cdot & & & & \cdot & \cdot & \cdot & & \\ 0 & & & & & \frac{c}{2} & d & \frac{c}{2} & \\ \frac{\phi_k(0)}{\phi_0} & \frac{c}{2} & 0 & \cdot & \cdot & \cdot & 0 & \frac{c}{2} & d \end{bmatrix}. \quad (7.3.1)$$

7.3.4. Leading Coefficient $\mathbf{g}_{k-1}$. Since

$$C_{k-1}(\xi) = \frac{1 - R_k(\xi)}{\xi},$$

it suffices to find the leading coefficient of $R_k = \dfrac{\psi_k(\xi)}{\psi_k(0)}$.

The leading coefficient of $\psi_k(\xi)$ is easily obtained from the recursion

$$\psi_0 = 1,$$

$$\psi_1(\xi) = \frac{\xi - d}{c}\,\psi_0,$$

and, for $2 \leq i$,

$$\psi_i(\xi) = 2\frac{\xi - d}{c}\,\psi_{i-1}(\xi) - \psi_{i-2}(\xi)\,.$$

The leading coefficient of ψ_k is therefore $\frac{1}{2}\left(\frac{2}{c}\right)^k$. This combined with the expression for $\psi_k(0)$ gives

$$g_{k-1} = -\frac{1}{2}\,\frac{\left(\frac{2}{c}\right)^k}{\cosh k \cosh^{-1}\left(-\frac{d}{c}\right)}\,.$$

7.3.5. Algorithm for the Grand–Leap Parameters in the Chebyshev Case.

Algorithm 6. (Compute the parameters for the Grand–Leap Algorithm in the Chebyshev Case.)

Purpose. Compute g_{k-1}, τ_0, and σ_1, ..., σ_{k-1} in the Chebyshev case as required for Algorithm 2.

Input. Ellipse parameters d and c and period k.

Output. The $k-1$ roots, $\{\sigma_i\}$, and leading coefficient g_{k-1} of the polynomial preconditioner C_{k-1}.

Restrictions. If the grand–leap algorithm is to converge then the ellipse parameters must satisfy the same restrictions as in Algorithm 1.

Notes. The algorithm uses a matrix $\tilde{\mathbf{S}}_k$ that is not defined if $c = 0$. In this case, the set of Chebyshev residual polynomials reduces to the family $\{R_k = (\varsigma - d)^k / d^k : 0 \leq k$, which is not an orthogonal family for any weight function. The case $c = 0$ is often used in the Manteuffel algorithm in order to compute improved ellipse parameters adaptively. If one believed that $c = 0$ then Richardson's method would converge in a single step if the matrix were normal, in which case no need exists for the grand-leap formulation. If one were computing ellipse parameters adaptively, then the parameter computation technique reduces to a sequence of matrix vector multiplications, and again the grand-leap formulation is not desired. For these reasons, if c is small relative to d the algorithm halts. The halting criterion is a comparison of $| c/d |^2$ to the machine epsilon, denoted in the algorithm by "mach eps" and defined to be the largest machine number, ϵ, such that the floating point sum $1 + \epsilon$ equals the machine number 1.

As a final note, there does exist an analog of $\tilde{\mathbf{S}}_k$ in the degenerate case ($c = 0$), and the algorithm control could branch to the computation of the eigenvalues of the analog of $\tilde{\mathbf{S}}_k$, but the lack of a practical need obviates this version.

1) Set $\tau_1 := \dfrac{1}{-c\,\pi/2+d}$.

2) If $k = 1$ or $| c/d | \leq \sqrt{\text{mach eps}}$ then return.

3) Set

$$\phi_k(0) := \frac{\psi_k(0)}{\| \psi_k \|_{\xi w}} = \frac{\cosh k \cosh^{-1}\left| -\frac{d}{c} \right|}{\sqrt{\frac{\pi}{2}}}.$$

4) Set the roots $\{ \sigma_1, \ldots, \sigma_{k-1} \}$ of C_{k-1} equal to the nonzero eigenvalues of matrix (7.3.1).

5) Set the leading coefficient g_{k-1}' of C_{k-1} equal to

$$g_{k-1} = - \frac{\frac{1}{2}\left(\frac{2}{c}\right)^k}{\cosh k \cosh^{-1}\left| -\frac{d}{c} \right|}.$$

8. Summary.

The leapfrog and grand-leap variants of Richardson's method and a general second order method have been described. A comparison among the methods and variants shows that there are advantages either to omitting every other iterant or to omitting all iterants (except the last).

The leapfrog and grand-leap variants require sets of parameters that may be computed from the eigenvalues of a matrix. In the leapfrog case, the matrix is the same as that which expresses the roots of a member of a family of orthogonal polynomials as the eigenvalues of a symmetric tridiagonal matrix. This matrix may be modified slightly to yield the parameters needed for the grand-leap variant.

Algorithms for the leapfrog and grand-leap methods are given in the Chebyshev case. In the Chebyshev case, explicit values for the elements of the tridiagonal matrix are well known and need not be computed.

9. Acknowledgements.

I am indebted to David Gottlieb and Ahmed Sameh for their interest and questions, to Paul Concus for many corrections, and to Jerry Minerbo for comments on the requirements for solving large problems. Using normalized polynomials is a suggestion of Bill Gragg. Partial support was provided by NSF DMS 8 7 03226.

References

[Adm82] L. Adams. "Iterative Algorithms for Large Sparse Linear Systems on Parallel Computers", NASA CR-166027, NASA Langley Research Center, Hampton, Va. 23665, 1982.

[AnGo72] R. S. Anderssen and G. H. Golub. "Richardson's Non-Stationary Matrix Iterative Procedure", Research Report 304, Stanford University, Dept. of Computer Science, 1972.

[Ashb85] S. F. Ashby. "CHEBYCODE: A FORTRAN Implementation of Manteuffel's Adaptive Chebyshev Algorithm", Research Report No. UIUCDCS-R-85-1203, Univ. of Illinois at Urbana-Champaign, Dept. of Computer Science, 1985.

[AMS87] S. F. Ashby, T. A. Manteuffel and P. E. Saylor. "A Taxonomy for Conjugate Gradient Methods", Report No. UIUCDCS-R-87-1355, 1987.

[Chen82] So Cheng Chen. "Polynomial Scaling in the Conjugate Gradient Method and Related Topics in Matrix Scaling", Report CS-82-23, Computer Science Dept., Penn. State Univ., University Park, Pa. 16802, 1982.

[Chro86] A. Chronopoulos. "A Class of Parallel Iterative Methods Implemented on Multiprocessors", Report No. UIUCDCS-R-86-1267, Univ. of Illinois at Urbana-Champaign, Dept. of Computer Science, November 1986.

[Eccl83] T. K. Eccles. *A Method of Data Management in a Simlulator Used for Large*

Reservoir Models. In: **Proceedings of the Seventh SPE Symposium on Reservoir Simulation**. SPE, Dallas, TX, 1983.

[ElSt85] H. C. Elman and R. L. Streit. "Polynomial Iteration for Nonsymmetric Indefinite Linear Systems", Research Report 380, Yale University, Dept. of Computer Science, 1985.

[FoWa60] G. E. Forsythe and W. W. Wasow. **Finite Difference Methods for Partial Differential Equations**. John Wiley and Sons, New York, 1960.

[Gaut82] W. Gautschi. *On Generating Orthogonal Polynomials.* **SIAM J. Stat. Sci. Comp.**, Vol. 3, No. 3, pp. 289–317, 1982.

[GoWe69] G. H. Golub and J. H. Welsch. *Calculation of Gaussian Quadrature Rules.* **Mathematics of Computation**, Vol. 23, pp. 221–230, 1969.

[HaYo81] L. A. Hageman and D. M. Young. **Applied Iterative Methods**. Academic Press, New York, 1981.

[JMP83] O. G. Johnson, C. A. Micchelli and G. Paul. *Polynomial Preconditioning for Conjugate Gradient Calculations.* **SIAM J. Numer. Anal.**, Vol. 20, No. 2, pp. 362–376, 1983.

[Mnt77] T. A. Manteuffel. *The Tchebyshev Iteration for Nonsymmetric Linear Systems.* **Numer. Math.**, Vol. 28, pp. 307–327, 1977.

[Mnt78] T. A. Manteuffel. *Adaptive procedure for estimating parameters for the nonsymmetric Tchebyshev iteration.* **Numer. Math.**, Vol. 31, pp. 183–208,

1978.

[OrVo85] J. Ortega and R. G. Voigt. "Solution of Partial Differential Equations on Vector and Parallel Computers", SIAM J. Stat. Sci. Comp., pp. 149–240, 1985.

[Sayl83] P. E. Saylor. *Preconditioning Symmetric Indefinite Matrices.* In: **Preconditioning Methods: Analysis and Applications**, David J. Evans, ed. Gordon and Breach Science Publishers, New York, pp. 295–319, 1983.

[SaSm88] P. E. Saylor and D. C. Smolarski S.J. *Computing the Roots of Complex Orthogonal and Kernel Polynomials.* **SIAM J. Sci. Stat. Comput.**, Vol. 9, 1988.

[Smol81] D. C. Smolarski S.J. "Optimum Semi–Iterative Methods of the Solution of Any Linear Algebraic System with a Square Matrix", Report No. UIUCDCS–81–1077, Univ. of Illinois at Urbana–Champaign, Dept. of Computer Science, December 1981.

[SmSa85] D. C. Smolarski S.J. and P. E. Saylor. "An Optimum Semi–Iterative Method for Solving Any Linear Set with a Square Matrix", Report No. UIUCDCS–R–85–1218, Univ. of Illinois at Urbana–Champaign, Dept. of Computer Science, July 1985.

[Stie58] E. Stiefel. *Kernel Polynomials in Linear Algebra and Their Numerical*

Applications. **National Bureau of Standards Math. Series**, Vol. 49, pp. 1–22, 1958.

[Tal87] H. Tal-Ezer. "Polynomial Approximation of Functions of Matrices and Applications", NASA Contractor Report 178376, ICASE Report No. 87–63, Institute for Computer Applications in Science and Engineering, NASA Langley Research Center, Hampton, Va. 23665, 1987.

[Wilf62] H. S. Wilf. **Mathematics for the Physical Sciences**. John Wiley and Sons, New York, 1962.

Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA CR-181616 ICASE Report No. 88-7		

4. Title and Subtitle	5. Report Date
LEAPFROG VARIANTS OF ITERATIVE METHODS FOR LINEAR ALGEBRA EQUATIONS	January 1988
	6. Performing Organization Code

7. Author(s)	8. Performing Organization Report No.
Paul E. Saylor	88-7
	10. Work Unit No. 505-90-21-01

9. Performing Organization Name and Address	11. Contract or Grant No.
Institute for Computer Applications in Science and Engineering Mail Stop 132C, NASA Langley Research Center Hampton, VA 23665-5225	NAS1-18107, AFOSR 85-0189
	13. Type of Report and Period Covered Contractor Report
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Langley Research Center Hampton, VA 23665-5225	14. Sponsoring Agency Code

15. Supplementary Notes

Langley Technical Monitor:
Richard W. Barnwell

Final Report

Submitted to SIAM Journal of Scientific and Statistical Computing

16. Abstract

Two iterative methods are considered, Richardson's method and a general second order method. For both methods, a variant of the method is derived for which only even numbered iterates are computed. The variant is called a leapfrog method. Comparisons between the conventional form of the methods and the leapfrog form are made under the assumption that the number of unknowns is large. In the case of Richardson's method, it is possible to express the final iterate in terms of only the initial approximation, a variant of the iteration called the grand-leap method. In the case of the grand-leap variant, a set of parameters is required. An algorithm is presented to compute these parameters that is related to algorithms to compute the weights and abscissas for Gaussian quadrature. General algorithms to implement the leapfrog and grand-leap methods are presented. Algorithms for the important special case of the Chebyshev method are also given.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
iterative methods, Richardson's method, Chebyshev method	64 - Numerical Analysis Unclassified - unlimited

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of pages	22. Price
Unclassified	Unclassified	63	A04

NASA FORM 1626 OCT 86

NASA-Langley, 1988

END

DATE

FILMED

DTIC

JULY 88