Draft Table of Contents and Comments on
"Computer Solution of Nonlinear Equations"

Preface

Part 1: Methods for a single nonlinear equation

1. Basic concepts
   1.1 Introduction
   1.2 Notation
   1.3 Simple and multiple zeros
   1.4 Order of convergence and error constants
   1.5 Computational efficiency
   1.6 Divided differences and polynomial interpolation
   1.7 Rootfinding by direct and inverse interpolation
   1.8 Difference equations
   1.9 Floating-point arithmetic
   1.10 Examples and problems
   1.11 Notes and references

2. One-point methods without memory
   2.1 One-point methods and functional iteration
   2.2 Newton's method
   2.3 Taylor series methods
   2.4 Other one-point methods without memory
   2.5 Numerical examples
   2.6 Problems
   2.7 Notes and references
3. Multipoint methods and methods with memory
   3.1 Definitions and examples
   3.2 Methods based on Hermite interpolation
   3.3 Matrix representations of Hermite methods
   3.4 Composite Hermite methods
   3.5 Methods using only function evaluations
   3.6 Comparison of efficiencies
   3.7 Methods for finding zeros of derivatives
   3.8 Accelerating convergence
   3.9 Numerical examples
   3.10 Problems
   3.11 Notes and references

4. Globally convergent methods and practical considerations
   4.1 Bisection and regula falsi
   4.2 The Illinois and Pegasus methods
   4.3 Fast hybrid methods
   4.4 Special methods for multiple zeros
   4.5 The effect of rounding errors
   4.6 Deflation
   4.7 Special methods for polynomials
   4.8 Numerical examples
   4.9 Problems
   4.10 Notes and references
5. Methods using mainly derivative evaluations
   5.1 Introduction
   5.2 Two classes of methods
   5.3 Some results on orthogonal polynomials
   5.4 The order of convergence
   5.5 Comparison of efficiencies
   5.6 Methods of practical interest
   5.7 Computation of error constants
   5.8 Some nonlinear Runge-Kutta methods
   5.9 Numerical examples
   5.10 Problems and open questions
   5.11 Notes and references

6. Variable precision methods
   6.1 Introduction and definitions
   6.2 VP discrete Newton methods
   6.3 VP secant methods
   6.4 VP methods using inverse interpolation
   6.5 Some bounds and conjectures
   6.6 Applications
   6.7 Summary of VP methods
   6.8 Problems
   6.9 Notes and references
Part 2: Methods for systems of nonlinear equations

7. Methods using derivatives
   7.1 Introduction
   7.2 Newton's method
   7.3 Conjugate gradient and quasi-Newton methods
   7.4 Equivalence of certain quasi-Newton methods
   7.5 Convergence results for quasi-Newton methods
   7.6 Numerical examples
   7.7 Problems
   7.8 Notes and references

8. Efficient methods using function evaluations
   8.1 Motivation
   8.2 The discrete Newton and Shamanskii-Traub methods
   8.3 Various generalizations of the secant method
   8.4 Triangular and orthogonal factorization methods
   8.5 Retaining approximations to the Jacobian
   8.6 Order of convergence theorems
   8.7 Methods with optimal efficiency
   8.8 Numerical examples
   8.9 Problems
   8.10 Notes and references
9. Nonlinear least squares problems
   9.1 Introduction
   9.2 The Gauss-Newton method
   9.3 The Levenberg-Marquardt method
   9.4 Discrete analogues
   9.5 Other methods for sums of squares
   9.6 Problems with separable variables
   9.7 Numerical examples
   9.8 Problems
   9.9 Notes and references

10. Widely convergent methods
    10.1 Methods based on minimization
    10.2 Points of attraction and repulsion
    10.3 Continuation and parameter-variation methods
    10.4 Davidenko's method
    10.5 Other differential equation methods
    10.6 Difficulties near Jacobian singularities
    10.7 Numerical examples
    10.8 Problems
    10.9 Notes and references
11. Practical considerations for systems
   11.1 Scaling
   11.2 Multiple zeros
   11.3 Choice of stepsize and effect of rounding errors
   11.4 Stopping criteria
   11.5 Deflation
   11.6 Systems with a large, sparse Jacobian
   11.7 Problems
   11.8 Notes and references
Part 2: Optimal and parallel methods

12. Complexity results for one equation
   12.1 Introduction
   12.2 One-point methods without memory
   12.3 Optimal order of methods with unbounded memory
   12.4 Results on methods with bounded memory
   12.5 Optimal methods using mainly derivative evaluations
   12.6 Conjectures and open problems
   12.7 Problems
   12.8 Notes and references

13. Complexity results for systems
   13.1 Introduction
   13.2 Results for linear systems
   13.3 Wozniakowski's maximal methods
   13.4 Some conjectures
   13.5 Problems
   13.6 Notes and references

14. Methods for parallel computers
   14.1 The use of parallelism at different levels
   14.2 Parallel methods for one equation
   14.3 Parallel methods for systems of equations
   14.4 Complexity results for parallel methods
   14.5 Problems
   14.6 Notes and references

Bibliography

Appendix: Fortran subroutines

Index
General comments

1. Not intended to compete with the books by Traub, Ostrowski, Householder, or Ortega & Rheinboldt, but to be complementary. In order to keep the book self-contained, some overlap with Traub and Ostrowski is unavoidable in Chs. 1-3.

2. Intended to be suitable for graduate textbook and reference. A considerable number of problems are included. These are intended both to illustrate and extend the ideas of the text. To avoid cluttering the text, each chapter concludes with notes (historical and technical) and references. The size is expected to be 300-400 pages.

3. The book is intended to be useful to someone who needs to solve nonlinear equations. Thus, numerical examples and well-tested Fortran programs are included.

4. Much material which is new or only available in technical journals is included (e.g. parts of Chs. 3, 4 & 10, and most of Chs. 5, 6, 8 and 12-14).

5. As of Jan. '75, drafts of Chs. 1 & 2 are written, and Chs. 5, 6, 8 and parts of 10, 12 & 14 will be based on papers which are already written. An annotated bibliography of about 400 papers has been prepared (this may be pruned later).
Part 1: Methods for a single nonlinear equation

Chapter 1: Basic concepts

1.1 Introduction
1.2 Notation
1.3 Simple and multiple zeros
1.4 Order of convergence and error constants
1.5 Computational efficiency
1.6 Divided differences and polynomial interpolation
1.7 Rootfinding by direct and inverse interpolation
1.8 Difference equations
1.9 Floating-point arithmetic
1.10 Examples and problems
1.11 Notes and references
1. Introduction

These notes describe some practical methods for the numerical solution of nonlinear equations and systems of equations. The emphasis is on methods which are efficient, reliable, and convenient to use on a digital computer.

Nonlinear equations often arise when physical problems are subjected to mathematical analysis. A relatively simple example is the problem of determining the level at which a light sphere floats in a liquid. Archimedes (287-212BC) reduced this problem to the solution of a cubic equation which he was able to solve graphically. A more complex example is the solution of boundary-value problems by "shooting" or "multiple shooting" methods, which lead to systems of nonlinear equations. These and several other examples are described in more detail in Section 10.

It might be thought that finding an accurate approximation to a root of a single nonlinear equation

$$f(x) = 0 \quad (1.1)$$

on a modern digital computer is a trivial problem. However, this is not true if the function $f$ is difficult to evaluate. For example, problems arise in which $f$ is an integral of the form

$$f(x) = \int \cdots \int g(x, t_1, \ldots, t_k) dt_1 \cdots dt_k, \quad (1.2)$$

or in which the evaluation of $f(x)$ involves solving a system of differential equations, evaluating a determinant, or minimizing a function of several variables. Also, it may be necessary to solve equations of the form (1.1) thousands
or even millions of times in the course of solving some more difficult problem. Thus, there is a real need for highly reliable and efficient methods for solving single algebraic or transcendental equations.

A system of \( n \) equations in \( n \) unknowns, say

\[
\begin{align*}
  f_1(x_1, \ldots, x_n) &= 0, \\
  & \vdots \\
  f_n(x_1, \ldots, x_n) &= 0,
\end{align*}
\]

may be written as

\[
\mathbf{f}(\mathbf{x}) = \mathbf{0},
\]

where \( \mathbf{f}, \mathbf{x} \) and \( \mathbf{0} \) are \( n \)-dimensional vectors. Using this notation, (1.1) and (1.4) are formally identical, but systems of equations actually present many more practical and theoretical difficulties than a single equation. Thus, we deal with methods for a single equation in Part 1 (Chapters 1 - 6) and defer consideration of systems until Part 2 (Chapters 7 - 11).

Several important topics are omitted from these notes. We do not discuss methods which are designed specifically for polynomial equations, except very briefly in Section 6.6. For such methods we recommend the excellent book by Householder (70), and the references given there. The methods which we consider require only that \( f(x) \), and sometimes certain of its derivatives, can be evaluated numerically for any given argument \( x \) in a certain domain. Thus, our methods apply equally well to polynomial and transcendental equations, but do not take
advantage of the special properties of polynomials. We assume that the reader is familiar with some direct numerical methods for solving systems of linear equations (see Forsythe & Moler (67) or Stewart (73a)). The methods of Part 2 are intended for nonlinear systems of moderate size with a dense Jacobian matrix. Thus, we deal only briefly (in Section 11.6) with linearly convergent methods which are suited to mildly nonlinear systems with a large, sparse, Jacobian matrix. These methods are discussed in Rheinboldt (74) and the references given there.

Although some of the results of Chapters 7 and 10 may be generalized to methods for solving equations in function spaces, we restrict ourselves entirely to equations in finite-dimensional Euclidean spaces, i.e., a finite number of equations in a finite number of (usually real) variables. This is because continuous problems usually have to be discretized (i.e., reduced to a finite-dimensional approximation) before they can be solved numerically. Some of the possible generalizations are given in the third edition of the classic book by Ostrowski (73).
2. Notation

To avoid repetition, certain notational conventions will be followed throughout. The letters i, j, k, m, n, p, q, and r denote integers (usually nonnegative). Other lower-case Roman and Greek letters (a, b, ..., x, y, ..., \( \alpha \), \( \beta \), ...) denote real scalars or functions (f and g are reserved for functions). All functions are real-valued and have real arguments. The \( n \)-th derivative of \( f \) is written as \( f^{(n)} \), with the customary abbreviations \( f^{(0)} = f \), \( f^{(1)} = f' \), \( f^{(2)} = f'' \), etc.

Boldface letters \( \mathbf{a}, \mathbf{b}, \ldots \) denote real column vectors or vector-valued functions. The superscript \( \text{"T"} \) denotes vector or matrix transpose. Capital letters \( A, B, \ldots \) usually denote real matrices, constants, functions, or methods. Unless the dimensions are stated explicitly, all matrices are \( n \) by \( n \) and vectors have dimension \( n \). \( U \) denotes a lower triangular matrix, \( U \) an upper triangular matrix, \( I \) the identity matrix, \( D \) a diagonal matrix, and \( Q \) an orthogonal matrix (i.e., \( Q^TQ = QQ^T = I \)). Elements of the vector \( \mathbf{a} \) are denoted by \( a_i \), and elements of \( A \) by \( a_{ij} \) (the element in the \( i \)-th row and \( j \)-th column). An exception is made for elements of \( L \), which are denoted by \( l_{ij} \). If there is any risk of ambiguity, we write \( a_{i,j} \) instead of \( a_{ij} \).

Norms of vectors and matrices are written as \( \| \mathbf{a} \| \) and \( \| A \| \). Unless otherwise specified, \( \| \mathbf{a} \| \) is the Euclidean norm (2-norm), defined by

\[
\| \mathbf{a} \| = (a_1^2 + \cdots + a_n^2)^{1/2},
\]

(2.1)

and \( \| A \| \) is the induced matrix norm. (If these definitions are unfamiliar, the excellent book by Stewart (73a) is
The columns of the identity matrix are \( e_1, \ldots, e_n \), so the \( j \)-th column of \( A \) is \( Ae_j \).

If an interval \([a, b]\) or \((a, b)\) is mentioned, it is assumed that \( a < b \). If \( f \) has \( n \) continuous derivatives on \([a, b]\), we write \( f \in C^n [a, b] \). If, in addition, \( f^{(n)} \) is Lipschitz continuous on \([a, b]\), we write \( f \in LC^n [a, b] \).

(A function \( g \) is Lipschitz continuous on \([a, b]\) if
\[
|g(x) - g(y)| \leq c|x - y|
\]
for some constant \( c \) and all \( x, y \in [a, b] \).

If \( f \) is a function of one variable, and \( \xi \) a real number such that
\[
f(\xi) = 0,
\]
we say that \( \xi \) is a zero of \( f \), or a root of the equation (1.1). Similarly for zeros of vector functions and roots of systems of equations.

Positive constants whose values do not need to be specified further are denoted by \( c_1, c_2, \ldots, c_i \) and the same \( c_1 \) may denote a different constant in different sections. The notation
\[
f = o(g)
\]
(2.4)
means that
\[
|f(x)| \leq c_1 |g(x)|
\]
(2.5)
for all \( x \) in a neighbourhood of some point which will be clear from the context (usually \( \xi \) or \( \infty \)).

Other notation will be defined as necessary (e.g., our notation for divided differences is defined in Section 6). Section \( n \) means the \( n \)-th section of the current chapter, section \( m.n \) means Section \( n \) of Chapter \( m \), and similarly for equation numbers.
Chapter 2: One-point methods without memory

2.1 One-point methods and functional iteration
2.2 Newton's method
2.3 Taylor series methods
2.4 Other one-point methods without memory
2.5 Numerical examples
2.6 Problems
2.7 Notes and references
Chapter 4: Practical considerations and programs

This is fairly well covered by Ch. 4 of Brent (1973a), which is reproduced in the following pages.

Additional reading

Gives a modified Illinois or Pegasus method which is usually fast, but could be much slower than bisection.

Gives some interesting methods which can never be much slower than bisection, and compares them and several other methods numerically. Includes Algol 60 procedures.

Describes a combination of bisection and linear interpolation which is usually fast, but occasionally very slow. See Bus & Dekker (74) and Brent (73a) for improvement.


Additional reading cont.


Wilkes, M. V., Wheeler, D. J. & Gill, S., 1951, The preparation of programs for an electronic digital computer, Addison-Wesley, Reading, Massachusetts. One of the earliest references on the subject (the problem of guaranteeing convergence did not arise when computations were done by hand, because the human computers usually had some common sense).

Wilkinson, J. H., 1967, Two algorithms based on successive linear interpolation, Report CS 60, Comp. Sci. Dept., Stanford Univ. The first algorithm has the same order as the Pegasus method (1.44...) and is very similar to it. The second algorithm is an early version of Dekker's (see Dekker (69)).
AN ALGORITHM
WITH GUARANTEED
CONVERGENCE FOR FINDING
A ZERO OF A FUNCTION

Section 1
INTRODUCTION

Let \( f \) be a real-valued function, defined on the interval \([a, b]\), with \( f(a)f(b) \leq 0 \). \( f \) need not be continuous on \([a, b]\); for example, \( f \) might be a limited-precision approximation to some continuous function (see Forsythe (1969)). We want to find an approximation \( \zeta \) to a zero \( \xi \) of \( f \), to within a given positive tolerance \( 2\delta \), by evaluating \( f \) at a small number of points. Of course, there may be no zero in \([a, b]\) if \( f \) is discontinuous, so we shall be satisfied if \( f \) takes both nonnegative and nonpositive values in \([\zeta - 2\delta, \zeta + 2\delta] \cap [a, b] \).

Clearly, such a \( \zeta \) may always be found by bisection in about \( \log_2(|(b - a)/\delta|) \) steps, and this is the best that we can do for arbitrary \( f \). In this chapter we describe an algorithm which is never much slower than bisection (see Section 3), but which has the advantage of superlinear convergence to a simple zero of a continuously differentiable function, if the effect of rounding errors is negligible. This means that, in practice, convergence is often much faster than for bisection (see Section 4). There is no contradiction here: bisection is the optimal algorithm (in a minimax sense) for the class of all functions which change sign on \([a, b]\), but it is not optimal for other classes of functions: e.g., \( C^1 \) functions with simple zeros, or convex functions. (See Gross and Johnson (1959), Bellman and Dreyfus (1962), and Chernousko (1970).)
Chapter 5: Methods using mainly derivative evaluations

5.1 Introduction
5.2 Two classes of methods
5.3 Some results on orthogonal polynomials
5.4 Theorems on the order of convergence
5.5 Comparison of efficiencies
5.6 Some methods of practical interest
5.7 Other possibilities
5.8 Some nonlinear Runge-Kutta methods
5.9 Numerical results
5.10 References

This Chapter appeared as "Efficient methods for finding zeros of functions whose derivatives are easy to evaluate", Tech. Report, Carnegie-Mellon University (Dept. of Comp. Sci.), Dec. 1974. (The report includes Fortran programs.)
Chapter 8: Efficient methods using function evaluations

This is covered by Brent (73b), which is reproduced in the following pages.
SOME EFFICIENT ALGORITHMS FOR SOLVING SYSTEMS
NONLINEAR EQUATIONS*

RICHARD P. BRENT†

Abstract. We compare the Ostrowski efficiency of some methods for solving systems of nonlinear equations without explicitly using derivatives. The methods considered include the discrete Newton method, Shamanskiǐ's method, the two-point secant method, and Brown's methods. We introduce a class of secant methods and a class of methods related to Brown's methods, but using orthogonal rather than stabilized elementary transformations. The idea of these methods is to avoid finding a new approximation to the Jacobian matrix of the system at each step, and thus increase the efficiency. Local convergence theorems are proved, and the efficiencies of the methods are calculated. Numerical results are given, and some possible extensions are mentioned.

1. Introduction. We are interested in comparing iterative processes for approximating a solution \( x^* \) of a system \( f(x) = 0 \) of nonlinear equations. If \( x_0, x_1, \ldots \) is a convergent sequence of vectors with limit \( x^* \in \mathbb{R}^n \), then the order of convergence \( \rho \) is defined by

\[
\rho = \lim_{i \to \infty} \inf (-\log \| x_i - x^* \|)^{1/i}.
\]

(1)

It does not matter which of the usual vector norms is used in (1). Other definitions of order may be given (see Ortega and Rheinboldt (1970, Chap. 9), Voigt (1971), and Brent (1972b, § 3.2)), but (1) is adequate for our purposes. We only consider processes for which \( \rho > 1 \), and in this case \( \rho \) is the same as the R-order of Ortega and Rheinboldt (1970).

If \( w_i \) is the amount of work required to compute \( x_i \) from \( x_{i-1} \) and other results which may have been saved from previous iterations, then the efficiency \( E \) of the process is defined by

\[
E = \lim_{i \to \infty} \inf \left( \frac{\log (-\log \| x_i - x^* \|)}{\sum_{j=1}^{w_i} w_j} \right).
\]

(2)

In particular, if there exists \( w = \lim_{i \to \infty} w_i > 0 \), then \( E = (\log \rho)/w \) is the logarithm of the "efficiency index" of Ostrowski (1960, § 3.11). The \( w_i \) may be measured in any appropriate units: we mainly use function evaluations, i.e., evaluations of \( f \).

Consider iterative methods \( M \) and \( M' \) with orders \( \rho \) and \( \rho' \) and efficiencies \( E , E' \). For simplicity, suppose that the \( w_i \) are bounded and the lower limits in (1) and (2) may be replaced by limits. Our justification for the term "efficiency" is that method \( M \) requires \( E'/E \) times as much work as method \( M' \) to reduce \( \| x_i - x^* \| \) to a very small positive tolerance. Thus, if factors such as the domains of convergence, ease of implementation, and storage space required are comparable, the method with the higher efficiency is to be preferred, and this is not always the method with the higher order. (As a trivial illustration, consider taking every second iterate of \( M \) as an iterate of \( M' \), so \( x'_i = x_{2i} \) and \( w'_i = w_{2i-1} + w_{2i} \). Then \( \rho' = \rho^2 > \rho \), but

* Received by the editors January 13, 1972.
† This author received his Ph.D. in Computer Science in 1971 from Stanford University under the direction of Professors Forsythe and G. Golub. He is now a Research Fellow in the Computer Centre at the Australian National University, Canberra, Australia. Most of the work in this paper was performed while the author was visiting the IBM Thomas J. Watson Research Center.
Chapter 12: Complexity results for one equation

This is partly covered by Brent, Winograd & Wolfe (73), which is reproduced in the following pages.

Additional reading


Kung, H. T. & Traub, J. F., 1973c, Optimal order and efficiency for iterations with two evaluations, Comp. Sci. Dept., Carnegie-Mellon Univ. This paper shows that the optimal order for a method without memory which uses one evaluation of $f$ and one of $f'$ (or two of $f$) per iteration is 2. See also Kung & Traub (73a, b).


Optimal Iterative Processes for Root-Finding

Richard Brent, Shmuel Winograd, and Philip Wolfe
Mathematical Sciences Department, IBM Watson Research Center,
Yorktown Heights, New York

Received August 3, 1972

Abstract. Let \( f_a(x) \) be a function of one variable with a simple zero at \( r_a \). An iteration scheme is said to be locally convergent if, for some initial approximations \( x_1, \ldots, x_d \) near \( r_a \) and all functions \( f \) which are sufficiently close (in a certain sense) to \( f_a \), the scheme generates a sequence \( \{x_n\} \) which lies near \( r_a \) and converges to a zero \( r \) of \( f \). The order of convergence of the scheme is the infimum of the order of convergence of \( \{x_n\} \) for all such functions \( f \). We study iteration schemes which are locally convergent and use only evaluations of \( f, f', \ldots, f^{[d]} \) at \( x_1, \ldots, x_{d+1} \) to determine \( x_d \), and we show that no such scheme has order greater than \( d + 2 \). This bound is the best possible, for it is attained by certain schemes based on polynomial interpolation.

I. Introduction

Many "iterative" methods are known for the numerical solution of the problem of finding a zero \( r \) of a function \( f(x) \) of a single real variable. The iterative process generates a sequence \( \{x_n\} \) of approximations to \( r \), where \( x_n \) is determined by the values of \( f \) and possibly of some of its derivatives at previous members of the sequence. (The term "iterative" is widely and loosely used; the preceding description seems to cover its use in our subject.) If the process starts at points which are close enough to \( r \), then the sequence \( \{x_n\} \) should converge to \( r \). The various methods differ in the amount of information used, the particular way the information is used to generate the next approximation, and consequently the rate at which the sequence \( \{x_n\} \) converges to \( r \). The secant method and Newton's method are examples of iterative methods which are much used in practice. Traub's book [1] describes a wide variety of such processes, all fitting the general outline: Given the points \( x_{k-1}, \ldots, x_{k-m} \) as well as the values of the function and its first \( d \) derivatives at these points, construct the minimal degree interpolating polynomial fitting these data, and choose \( x_k \) as a root of this polynomial (or as its value at zero, if it is a polynomial in the dependent variable). The secant method and Newton's method are in this class of iteration methods.

An iterative method does not, however, have to use the root of such a polynomial. For example, the iteration defined by

\[
x_k = \frac{f(x_{k-1}) (x_{k-1} + f(x_{k-1})) - f(x_{k-2}) (x_{k-1} + 1) + f(x_{k-2})}{f(x_{k-1}) - f(x_{k-2})},
\]

* This work was supported (in part) by the Office of Naval Research under contract numbers N0014-69-C-0023 and N0014-71-C-0112.