Numerical Solution of Nonlinear Equations

R. Brent
27 Jan 75

Draft Table of Contents and Comments on

"Computer Solution of Nonlinear Equations"

Part 2: Methods for systems of nonlinear equations

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. 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

$$f(x) = 0 \tag{1.1}$$

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

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

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

$$\left.\begin{aligned} f_1(x_1, \ldots, x_n) &= 0, \\ &\vdots \\ f_n(x_1, \ldots, x_n) &= 0, \end{aligned}\right\} \tag{1.3}$$

may be written as

$$\underset{\sim}{f}(\underset{\sim}{x}) = \underset{\sim}{0}, \tag{1.4}$$

where $\underset{\sim}{f}$, $\underset{\sim}{x}$ and $\underset{\sim}{0}$ 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, ... , α , β , ...) 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 $\underset{\sim}{a}$, $\underset{\sim}{b}$, ... denote real column vectors or vector-valued functions. The superscript "T" denotes vector or matrix transpose. Capital letters A, B, ... 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. L denotes a lower triangular matrix, U an upper triangular matrix, I the identity matrix, D á diagonal matrix, and Q an orthogonal matrix (i.e., $Q^TQ = QQ^T = I$). Elements of the vector $\underset{\sim}{a}$ 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 m_{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 $\|\underset{\sim}{a}\|$ and $\|A\|$. Unless otherwise specified, $\|\underset{\sim}{a}\|$ is the Euclidean norm (2-norm), defined by

$$\|\underset{\sim}{a}\| = (a_1^2 + \ldots + a_n^2)^{1/2}, \tag{2.1}$$

and $\|A\|$ is the induced matrix norm. (If these definitions are unfamiliar, the excellent book by Stewart (73a) is

recommended.) The columns of the identity matrix are $\underset{\sim}{e}_1, \ldots, \underset{\sim}{e}_n$, so the j-th column of A is $A\underset{\sim}{e}_j$.

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

$$|g(x) - g(y)| \leq c|x - y| \tag{2.2}$$

for some constant c and all $x, y \in [a, b]$.)

If f is a function of one variable, and ζ a real number such that

$$f(\zeta) = 0, \tag{2.3}$$

we say that ζ is a <u>zero</u> of f, or a <u>root</u> 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 $c_1, c_2, \ldots$, and the same c_i may denote a different constant in different sections. The notation

$$f = O(g) \tag{2.4}$$

means that

$$|f(x)| \leq c_1|g(x)| \tag{2.5}$$

for all x in a neighbourhood of some point which will be clear from the context (usually ζ or ∞).

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

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

Anderson, N. & Bjorck, A., 1973, A new high order method of regula falsi type for computing a root of an equation, BIT 13, 253-264. CR 15#26760 Gives a modified Illinois or Pegasus method which is usually fast, but could be much slower than bisection.

Bus, J. C. P. & Dekker, T. J., 1974, Two efficient algorithms with guaranteed convergence for finding a zero of a function, Tech. Report NW 13/74, Stichting Mathematisch Centrum, Amsterdam. Gives some interesting methods which can never be much slower than bisection, and compares them and several other methods numerically. Includes Algol 60 procedures.

Dekker, T. J., 1969, Finding a zero by means of successive linear interpolation, in Constructive Aspects of the Fundamental Theorem of Algebra (ed. by Dejon & Henrici), Wiley, New York. Describes a combination of bisection and linear interpolation which is usually fast, but occas onally very slow. See Bus & Dekker (74) and Brent (73a) for improvement.

Dowell, M. & Jarratt, P., 1971, A modified regula falsi method for computing the root of an equation, BIT 11, 168-174. Describes the "Illinois" method (regula falsi except the retained f value is halved).

Dowell, M. & Jarratt, P., 1972, The "Pegasus" method for computing the root of an equation, BIT 12, 503-508. Gives an improvement of the Illinois method, but may still be very slow in exceptional cases. See King (1973b).

Additional reading cont.

Jarratt, P. & Nudds, D., 1965, The use of rational functions in the iterative solution of equations on a digital computer, Comp. J. 8, 62-65. Describes the use of rational rather than polynomial approximation. See also problem 1.42 and Bus & Dekker (74).

King, R. F., 1973b, An improved Pegasus method for root finding, BIT 13, 423-427. CR 15#26849
Improves Dowell & Jarratt (72), but convergence may still be slow sometimes.

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)).

4

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 $\hat{\zeta}$ to a zero ζ of f, to within a given positive tolerance 2δ, 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 $[\hat{\zeta} - 2\delta, \hat{\zeta} + 2\delta] \cap [a, b]$.

Clearly, such a $\hat{\zeta}$ 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

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.

SIAM J. NUMER. ANAL.
Vol. 10, No. 2, April 1973

8/2

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, Shamanskii'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 $\mathbf{x}^*$ of a system $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ of nonlinear equations. If $\mathbf{x}_0, \mathbf{x}_1, \cdots$ is a convergent sequence of vectors with limit $\mathbf{x}^* \in R^n$, then the *order of convergence* ρ is defined by

$$\rho = \liminf_{i \to \infty} (-\log \|\mathbf{x}_i - \mathbf{x}^*\|)^{1/i}. \tag{1}$$

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 ρ is the same as the R-order of Ortega and Rheinboldt (1970).

If w_i is the amount of work required to compute $\mathbf{x}_i$ from $\mathbf{x}_{i-1}$ and other results which may have been saved from previous iterations, then the *efficiency* E of the process is defined by

$$E = \liminf_{i \to \infty} \left(\frac{\log(-\log \|\mathbf{x}_i - \mathbf{x}^*\|)}{\sum_{j=1}^{i} w_j} \right). \tag{2}$$

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 $\mathbf{f}$.

Consider iterative methods M and M' with orders ρ, ρ' 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 $\|\mathbf{x}_i - \mathbf{x}^*\|$ 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 $\mathbf{x}_i' = \mathbf{x}_{2i}$ 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., 1973a, Optimal order of one-point and multipoint iteration, Tech. Report, Comp. Sci. Dept., Carnegie-Mellon Univ. (to appear in J. ACM).

Kung, H. T. & Traub, J. F., 1973b, Computational complexity of one-point and multipoint iteration, in Complexity of Real Computation (ed. by R. Karp), American Math. Soc., New York.

Kung, H. T. & Traub, J. F., 1973c, Optimal order and efficiency for iterations with tow 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).

Rissanen, J., 1971, On optimum root-finding algorithms, J. Math. Anal. Appl. 36, 220-225. Shows that the secant method has optimal order amongst a class of methods using function evaluations and having only limited memory.

Wozniakowski, H., 1974a, Maximal stationary iterative methods for the solution of operator equations, SIAM J. Numer. Anal. 11, 934-949. Mainly relevant to systems rather than a single equation.

Wozniakowski, H., 1974b, Generalized information and maximal order of iteration for operator equations, Tech. Report, Comp. Sci. Dept., Carnegie-Mellon Univ.

Hindmarsh, A. C., 1972, Optimality in a class of rootfinding algorithms, SIAM J. Numer. Anal. 9, 205-214. Restricted to composite Hermite interpolatory methods.

Traub, J. F., 1964, Iterative methods for the solution of equations, Prentice-Hall. See esp. Thm. 5-3 for one-point methods without memory.

Numer. Math. 20, 327—341 (1973)

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 $f_0(x)$ be a function of one variable with a simple zero at r_0. An iteration scheme is said to be locally convergent if, for some initial approximations $x_1, \ldots, x_s$ near r_0 and all functions f which are sufficiently close (in a certain sense) to f_0, the scheme generates a sequence $\{x_k\}$ which lies near r_0 and converges to a zero r of f. The order of convergence of the scheme is the infimum of the order of convergence of $\{x_k\}$ for all such functions f. We study iteration schemes which are locally convergent and use only evaluations of $f, f', \ldots, f^{[d]}$ at $x_1, \ldots, x_{k-1}$ to determine x_k, 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 $\{x_k\}$ of approximations to r, where x_k 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 $\{x_k\}$ 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 $\{x_k\}$ 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

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

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