Numerical Differentiation of Analytic Functions

BENGT FORNBERG
California Institute of Technology

It is well known that the classical difference formulas for evaluating high derivatives of a real function $f(\zeta)$ are very ill-conditioned. However, if the function $f(\zeta)$ is analytic and can be evaluated for complex values of ζ, the problem can be shown to be perfectly well-conditioned. An algorithm that performs this evaluation for an arbitrary analytic function $f(\zeta)$ is described. A short FORTRAN program for generating up to 50 leading derivatives is to be found in the algorithm section of this issue. To use this program, no knowledge is required either of the method or of the analytical nature (e.g., position of nearest singularity, its type) of the function.

Key Words and Phrases: numerical differentiation, Taylor series coefficients, analytic functions
CR Categories: 5.16
The Algorithm: CPSC: Complex Power Series Coefficients, *ACM Trans. Math. Softw. 7*, 4 (Dec. 1981), 542–547.

1. INTRODUCTION

The purpose of the present algorithm is to provide the numerical FORTRAN user with an automatic method for finding derivatives of analytic functions. A typical usage may be to find a moderate number (up to 50 is allowed) of derivatives of functions whose analytic properties (position and nature of singularities in the complex plane, etc.) are not easily available. For reference, we briefly describe below two earlier numerical methods. Knowledge of these methods is not required to understand our present method, which is also summarized. This introductory section is concluded with some remarks on formula manipulation systems.

The fact that differentiation (or equivalently, evaluation of Taylor coefficients) of analytic functions can be efficiently performed numerically was observed about 15 years ago. The earliest algorithm seems to be the one described by Abate and Dubner in 1968 [1]. Their method is based on a numerical scheme for the inversion of Laplace transforms. Let us denote the function as $f(\zeta)$ and consider

This work was supported by Control Data Corporation and by the Department of Energy (Office of Basic Energy Sciences).
Author's address: Department of Applied Mathematics, 217-50 Caltech, Pasadena, CA 91125.

an expansion around $\zeta = z$:

$$f(\zeta) = \sum_{\nu=0}^{\infty} a_\nu (\zeta - z)^\nu. \tag{1}$$

The coefficients a_ν are to be determined.

We can define $g(t)$ as a piecewise linear function satisfying $g(n) = \sum_{\nu=0}^{n-1} a_\nu$ when n is a positive integer, and zero for nonpositive n. The Laplace transform of $g(t)$ can be shown to be

$$Lg(t) = \frac{1 - e^{-s}}{s^2} f(z + e^{-s}).$$

This transform can be numerically inverted by repeated complex numerical quadrature to give $g(n)$ for different n. The coefficients a_ν can then be evaluated by $a_\nu = g(\nu + 1) - g(\nu)$.

A different approach was taken by Lyness and co-workers [6]–[8]. By Cauchy's theorem, (1) implies

$$a_\nu = \frac{1}{2\pi i} \int_{c_r} \frac{f(\zeta)}{(\zeta - z)^{\nu+1}} d\zeta. \tag{2}$$

This periodic contour integral over a circle with radius r can be evaluated efficiently by trapezoidal approximations. If this is to be performed simultaneously for a large number of values of ν, the fast Fourier transform algorithm [2, 3] gives the different trapezoidal approximations directly from the equispaced function values on c_r. Extrapolation methods have been developed to suppress truncation errors, which are very dependent on singularities in $f(\zeta)$. Lyness and Moler discuss in [7] cancellation of error terms with the use of the Möbius transform. An algorithm was later published by Lyness and Sande [8]. That algorithm requires the user to choose a radius and then attempts to reach a required accuracy with the use of an increasing number of sample points on this fixed circle. A comparison in performance between this method and the present one is given in the last section.

The aim of this study is to develop an alternative algorithm leading to both a shorter and a more automatic code.[1] Its use does not require knowledge about the method or the character of the function.

As before, we denote the function as $f(\zeta)$. We want to evaluate a number of leading coefficients a_ν in (1). The calculation in this routine proceeds in the following steps.

(1) A radius r is chosen.
(2) We put $\zeta = z + r \cdot e^{i\theta}$ and evaluate $f(\zeta)$ for n (a power of 2) equidistant values of θ in the interval $(0, 2\pi)$.
(3) Since

$$f(z + r \cdot e^{i\theta}) = \sum_{\nu=0}^{\infty} a_\nu r^\nu e^{i\nu\theta} \tag{3}$$

[1] An early version of this description and code was given by the present author in [4].

a discrete (fast) Fourier transform (FFT) on these function values can be used to obtain the numbers b_ν, where

$$b_\nu = r^\nu \times \{a_\nu + r^n a_{n+\nu} + r^{2n} a_{2n+\nu} + r^{3n} a_{3n+\nu} + \cdots\},$$
$$\nu = 0, 1, 2, \ldots, n-1. \quad (4)$$

(The terms after the leading one a_ν are aliasing terms: They correspond to the higher modes that cannot be distinguished from the νth mode using only function values at the discrete points.)

(4) On the basis of two tests using the b_ν, the radius r is changed in order to optimize the final accuracy. If a stop criterion is not satisfied, we return to step 2.

(5) Using the results from the last three radii, repeated Richardson extrapolation gives the desired coefficients.

In the special case of an expansion around a real point z for a function that is real on the real axis, symmetries can be used to reduce all work by a factor of 2. (The code in the FORTRAN algorithm is for the general case. It does not test for or exploit this possibility.)

In some cases, in particular when analytical rather than numerical results are desired, formula manipulation systems can be considered. For the typical numerical FORTRAN user, disadvantages with this approach include limited availability of powerful systems, uncertain efficiency, difficulties in automatic interfacing with FORTRAN code, and a very limited function repertoire. Most special functions (e.g., the gamma function) are easily available numerically, but often not differentiable in closed form.

2. IMPLEMENTATION

To explain the details of this algorithm and the basic structure of a code for it, we consider the following very simple example: $f(\zeta) = 1/(1-\zeta)$ expanded around $z = 0$. Assume that we require the first 15 coefficients and that all computations are performed with 14 significant digits. Increased accuracy is gained by changes in the radii and by extrapolations, not by increasing the number of sample points. Therefore, we can decide to use 32 points. (This power of 2 is chosen in the code if we want between 13 and 25 coefficients.) The radius of the first circle can be chosen at random (within some four orders of magnitude of an "optimal" choice). Assume we pick $r_0 = 0.6580924658$. The first FFT will return the 32 values

$$b_{0,\nu} = \begin{bmatrix} 1.0000015317 \\ 0.6580934738 \\ \vdots \\ 0.0000035368 \\ 0.0000023275 \end{bmatrix}, \qquad \nu = 0, 1, \ldots, 31. \quad (5)$$

After division by r_0^{ν} we get

$$b_{0,\nu}r_0^{-\nu} = \begin{bmatrix} 1.0000015317 \\ 1.0000015317 \\ \vdots \\ 1.0000015322 \\ 1.0000015342 \end{bmatrix}. \tag{6}$$

From formula (4) we know that these numbers are equal to

$$\begin{bmatrix} a_0 + r_0^{32}a_{32} + r_0^{64}a_{64} + \cdots \\ a_1 + r_0^{32}a_{33} + r_0^{64}a_{65} + \cdots \\ \vdots \\ a_{30} + r_0^{32}a_{62} + r_0^{64}a_{94} + \cdots \\ a_{31} + r_0^{32}a_{63} + r_0^{64}a_{95} + \cdots \end{bmatrix}. \tag{7}$$

If this calculation is repeated with another radius r_1, the linear combination

$$b_{1,\nu}r_1^{-\nu} - (b_{1,\nu}r_1^{-\nu} - b_{0,\nu}r_0^{-\nu}) \cdot \frac{1}{1 - (r_0/r_1)^{32}}$$

of the two resulting vectors will give the vector

$$\begin{bmatrix} a_0 - (r_0 \cdot r_1)^{32}a_{64} + \cdots \\ a_1 - (r_0 \cdot r_1)^{32}a_{65} + \cdots \\ \vdots \\ a_{30} - (r_0 \cdot r_1)^{32}a_{94} + \cdots \\ a_{31} - (r_0 \cdot r_1)^{32}a_{95} + \cdots \end{bmatrix}, \tag{8}$$

where the first truncation error term is eliminated. This is a standard Richardson extrapolation (see, for example, [5]). The repeated version of it with a third radius r_2 gives as the final result

$$\begin{bmatrix} a_0 + (r_0 \cdot r_1 \cdot r_2)^{32}a_{96} + \cdots \\ a_1 + (r_0 \cdot r_1 \cdot r_2)^{32}a_{97} + \cdots \\ \vdots \\ a_{30} + (r_0 \cdot r_1 \cdot r_2)^{32}a_{126} + \cdots \\ a_{31} + (r_0 \cdot r_1 \cdot r_2)^{32}a_{127} + \cdots \end{bmatrix}. \tag{9}$$

By using small radii, we can decrease the truncation errors $(r_0 \cdot r_1 \cdot r_2)^{32}a_{96+\nu}$. However, the use of small radii will lead to few significant digits in the last b_ν, since an FFT produces an output with approximately the same absolute error for all coefficients. It is reasonable to choose radii such that the two types of errors approach the same level. This would happen, in this example, if r is chosen such that the last b_ν are four powers of 10 less than the first ones. Figures 1–4 illustrate this. Figure 1 shows schematically the b_ν in fixed-point format. The number b_{n-1} has four more leading zeros than b_0. Coefficients beyond b_{n-1} appear superposed

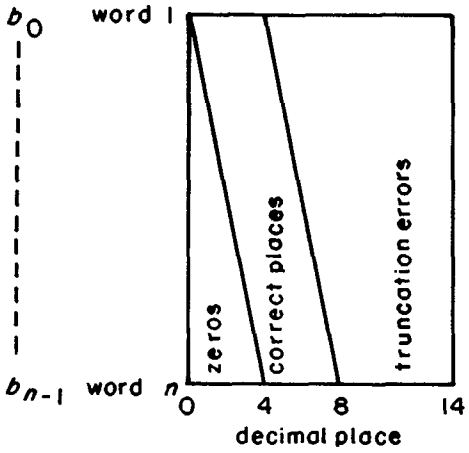

Fig. 1. Typical form of the b_ν in fixed point.

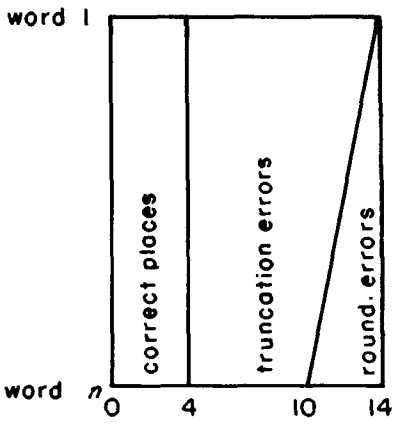

Fig. 2. Corresponding mantissas for the b_ν in floating point.

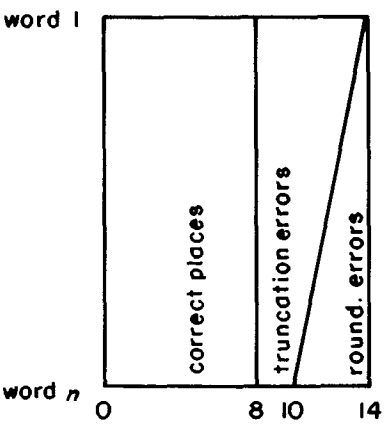

Fig. 3. The b_ν after one Richardson extrapolation.

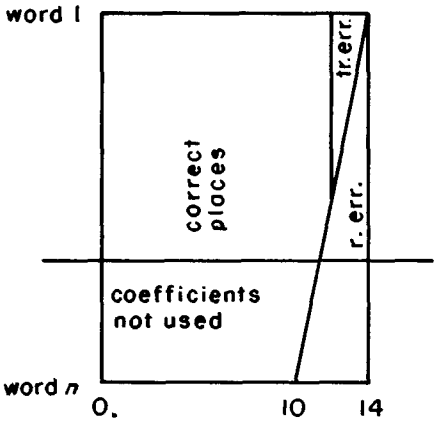

Fig. 4. The b_ν after two Richardson extrapolations.

on b_0, b_1, etc., as truncation errors. The rounding errors have the same absolute size for all coefficients. In a computer with floating-point word format, mantissas usually have no leading zeros. Figure 2 shows what these mantissas would look like. This figure can be compared with (6). Comparing (7) and (8) we see that one step of Richardson extrapolation will remove the leading truncation errors (corresponding to coefficients $r_0^{32}a_{32}$ to $r_0^{32}a_{63}$ in (7)). This is illustrated in Figure 3 where the number of correct places has doubled from 4 to 8. After one more extrapolation step, represented by (9) and Figure 4, we have reached our final result. With a word length of 14 places, about 12 are correct.

The initial radius $r_0 = 0.658092\ldots$ was not the optimal choice. Therefore, a search for a radius such that the b_ν look like those in Figure 1 is conducted. If a radius is too small, it causes the b_ν to decrease too rapidly, and the opposite holds

true if the radius is too big. We compare $|b_\nu|$ with a geometric progression c_ν such that $c_0 = 1$ and $c_{n-1} = 10^{-4}$ (in this case $n = 32$) and check if $\max_\nu |b_\nu|/c_\nu$ is attained for a ν in the upper or lower half of the range $[0, n-1]$ for ν.

In this example, the maximum is attained for $\nu = 1$. However, if r is bigger than the radius of convergence, the b_ν normally correspond to a Laurent expansion and not to a Taylor expansion. If our test should suggest an increase in r (as in the case in our example), a further test is made by reevaluating three function values, assuming that the series is a Taylor series, and comparing them with the true function values. The arguments used are $z + r\cdot(-0.4 + 0.3\cdot i)$, $z + r\cdot(0.7 + 0.2\cdot i)$, and $z + r\cdot(0.02 - 0.06\cdot i)$, where z is the center of the expansion. The arguments are randomly chosen. The values are considered to agree if the maximal difference in the three cases is less than 10^{-3} times the maximal function value in the three points, according to the evaluation by the series. If the values do not agree, r is decreased; otherwise r is increased. In the latter case, this test is not performed again because the first test will be sufficient. The reason for this can be seen in our example.

The first radius was $r_0 = 0.658092\ldots$. After this, the radius is repeatedly changed, either by a factor of 2 or of 0.5, until it becomes too large or too small. Since the first test suggests an increase in our example, the second test is performed, and it shows agreement. The next radius used is thus $r_1 = 1.316185\ldots$. We obtain

$$b_{1,\nu} = \begin{bmatrix} -0.0048649092 \\ -0.0064031201 \\ \vdots \\ -18.4749031128 \\ -24.3163890891 \end{bmatrix}. \tag{10}$$

These numbers correspond to a Laurent series. There is now a singularity inside our circle. The form assumed in eqs. (3) and (4) must be changed to also include negative values of ν. These additional terms in (4) will decay very slowly (if at all), since we are only very closely outside the singularity (we were too far inside it before the radius was doubled). If the decay rate is too slow, we can safely decrease the radius without having to decide if we ever had a Laurent series or not. From this point on, the factor by which we change r is either the square root of the previous factor or one divided by the square root. We do this six times (in this example) and obtain

$$r_2 = 0.9306832904$$

$$b_{2,\nu} = \begin{bmatrix} 1.1115844835 \\ 1.0345331046 \\ \vdots \\ 0.1288249475 \\ 0.1198952260 \end{bmatrix}$$

$$r_3 = 0.7826082426$$

$$b_{3,\nu} = \begin{bmatrix} 1.0003922760 \\ 0.7829152410 \\ \vdots \\ 0.0006404760 \\ 0.0005012418 \end{bmatrix}$$

$$r_4 = 0.7176549227$$

$$b_{4,\nu} = \begin{bmatrix} 1.0000245082 \\ 0.7176725112 \\ \vdots \\ 0.0000475862 \\ 0.0000341504 \end{bmatrix}$$

$$r_5 = 0.7494282207$$

$$b_{5,\nu} = \begin{bmatrix} 1.0000980401 \\ 0.7495016947 \\ \vdots \\ 0.0001745597 \\ 0.0001308199 \end{bmatrix} \qquad b_{5,\nu} r_5^{-\nu} = \begin{bmatrix} 1.0000980401 \\ 1.0000980401 \\ \vdots \\ 1.0000980401 \\ 1.0000980401 \end{bmatrix} \tag{11}$$

$$r_6 = 0.7658385618$$

$$b_{6,\nu} = \begin{bmatrix} 1.0001960995 \\ 0.7659887424 \\ \vdots \\ 0.0003343506 \\ 0.0002560586 \end{bmatrix} \qquad b_{6,\nu} r_6^{-\nu} = \begin{bmatrix} 1.0001960995 \\ 1.0001960995 \\ \vdots \\ 1.0001960995 \\ 1.0001960995 \end{bmatrix}$$

$$r_7 = 0.7575889589$$

$$b_{7,\nu} = \begin{bmatrix} 1.0001386553 \\ 0.7576940027 \\ \vdots \\ 0.0002415846 \\ 0.0001830219 \end{bmatrix} \qquad b_{7,\nu} r_7^{-\nu} = \begin{bmatrix} 1.0001386553 \\ 1.0001386553 \\ \vdots \\ 1.0001386553 \\ 1.0001386554 \end{bmatrix}.$$

For the last three radii, we perform the repeated Richardson extrapolation on the 15 first coefficients. This gives us our final answer.

$$a_\nu = \begin{bmatrix} 1.000000000003 \\ 1.000000000002 \\ 1.000000000002 \\ 1.000000000002 \\ 1.000000000002 \\ 1.000000000002 \\ 1.000000000002 \\ 1.000000000002 \\ 1.000000000001 \\ 1.000000000001 \\ 1.000000000001 \\ 1.000000000001 \\ 0.999999999999 \\ 0.999999999999 \\ 0.999999999999 \end{bmatrix}. \tag{12}$$

To obtain a practical code for automated general use, some fixed choices have to be made, even in cases where manual fine tuning for each individual test function could have been used. This is particularly the case if the potential advantages by case-dependent tuning, in either software safety or efficiency, are minimal. One such case is to relate the number of coefficients generated internally (n) to the number of coefficients asked for (N). The adapted rule was

$$\begin{aligned} 1 \le N \le 6, &\quad n = 8 \\ 7 \le N \le 12, &\quad n = 16 \\ 13 \le N \le 25, &\quad n = 32 \\ 26 \le N \le 51, &\quad n = 64 \end{aligned} \tag{13}$$

(called in the following ranges 1, 2, 3, and 4, respectively). Use of $n = 32$ or 64 is wasteful if use of $n = 8$ can perform the same job at a fraction of the cost. The other extreme, use of $n = 16$ if $N = 16$, on the whole worked satisfactorily but did fail in a few test cases. The choices given above represent a safe compromise, avoiding the two disadvantages mentioned. The choices of performing two Richardson extrapolations rather than, for example, one or three, represents a similar compromise. The two steps performed clearly increased the accuracy. Performing four or more extrapolations would offer only insignificant additional improvements. A further choice is, after the first time the changes in the radii altered direction, to let the number of circles used be three plus the range number. Use of fewer circles would cause the final accuracy to fluctuate more as a function of the initially supplied radius. This rule also makes the constants in the final extrapolations independent of the range number.

Table I. Performance of the Method in Six Test Cases

Function	Center of expansion	nth term in series	Maximal relative error in range				|actual error|/estimated error for ranges								Time on CDC 7600 (in ms) for ranges			
							1		2		3		4					
			1	2	3	4	max	min	max	min	max	min	max	min	1	2	3	4
$\frac{1}{1-\zeta}$	0	1	1×10^{-12}	6×10^{-12}	5×10^{-11}	1×10^{-10}	0.06	0.04	0.17	0.03	0.31	0.07	0.42	0.15	2	3	8	18
$\frac{10^{-20}}{1-10^3 \cdot \zeta}$	0	10^{3n-23}	4×10^{-12}	4×10^{-12}	2×10^{-11}	1×10^{-10}	0.12	0.05	0.12	0.04	0.14	0.02	0.16	0.06	3	8	16	35
e^{ζ}	0	$\frac{1}{(n-1)!}$	2×10^{-13}	1×10^{-12}	4×10^{-9}	4×10^{-2}	0.03	0.00	0.02	0.01	0.08	0.01	0.07	0.01	2	6	16	37
$e^{100\zeta}$	0	$\frac{100^{n-1}}{(n-1)!}$	8×10^{-13}	9×10^{-12}	5×10^{-9}	9×10^{-3}	0.11	0.01	0.13	0.04	0.03	0.02	0.02	0.00	4	7	13	27
$\frac{1+\zeta}{(1-\zeta)^3}$	0	n^2	4×10^{-12}	1×10^{-12}	1×10^{-11}	3×10^{-11}	0.25	0.02	0.17	0.00	0.07	0.01	0.05	0.01	2	3	8	18
$\frac{1}{\zeta}$	$0.4 + 0.3i$	$-(-1.6 + 1.2i)^n$	1×10^{-12}	6×10^{-12}	2×10^{-11}	7×10^{-11}	0.06	0.02	0.17	0.05	0.32	0.02	0.35	0.02	2	4	7	18

3. RESTRICTIONS

Figures 1–4 and the discussions about error levels become exact if the power series coefficients satisfy a geometric progression. A case in point is $f(z) = 1/(1 - z)$, which was used to illustrate the description. The errors in the method depend on how close to being true this geometric progression behavior is for the range of coefficients we consider. When we have increases in error, they will normally be revealed by the error estimates provided. The largest deviation from this geometric behavior (for functions not explicitly constructed to have isolated very large terms in an expansion like, say $f(z) = 10^5 + (1/(1 - z))$) exists among entire functions. (The coefficients for a function with a radius of convergence limited by either a branch point, essential singularity, or multiple pole will asymptotically decay by a rate lying between the two geometric progressions given by functions with a simple pole just inside and just outside the original circle. The rate can then be well estimated by these geometric progressions.) Two examples in Table I are used to illustrate the case e^z. The first 25 terms give little or no problem but, if 50 terms are required, the very lowest ones lose accuracy. However, the supplied error estimate predicts that accurately.

It is necessary that the function have at least one nonzero coefficient among the first $n/2$ and also one among the next $n/2$ coefficients, n being the number of points in the Fourier transforms. The routine should therefore not be used on a polynomial of low order.

If the radius of convergence is limited only by a branch point at which the function remains continuous, the irregularity may be difficult to detect numerically. An example is $f(z) = (1 + z)^{10}\log(1 + z)$ expanded around $z = 0$. The radius of convergence is 1 but, owing to the factor $(1 + z)^{10}$, the function appears to be numerically very regular around the critical point $z = -1$. The method will still work satisfactorily in ranges 1 and 2 (i.e., up to 12 coefficients), but for more coefficients both values and error estimates fail. Should this unusual problem be encountered, a reliable strategy would be to accept only coefficients obtained by using two different ranges.

If the initial guess for a computational radius is wrong either way by a factor of more than about 30,000, the routine makes an error exit leaving all elements of the result vector zero and of an error estimate vector 10^{10}. The reason for this restriction is that we want to avoid any risk of infinite loops. This could otherwise happen, for example, in cases of complex functions that lack a Taylor expansion ($f(z) = \bar{z}$, $f(z) = \sqrt{z}$ at $z = 0$, for a z on a branch cut, etc.).

4. ERROR ESTIMATE

The error estimate produced by the routine is based on a direct calculation following the principle shown in Figures 1–4. If the machine accuracy is ϵ (in our example, 10^{-14}), the truncation error is estimated by $\epsilon^{3/14}$ times the last Richardson correction, and the rounding error by

$$\epsilon \cdot \frac{1}{r^{\nu}} \cdot \max_{\nu} \frac{|b_{\nu}|}{c_{\nu}}.$$

The use of $\epsilon^{3/14}$ instead of $\epsilon^{4/14}$ includes a safety factor of $\epsilon^{1/14}$. This error estimate

Table II

k	$N = 6$ Range 1	$N \geq 12$ Ranges 2, 3, and 4 all agree to this accuracy for first 12 numbers
0	1.000000000	1.000000000
1	1.000000000	1.000000000
2	4.000000000	4.000000000
3	4.000000000	3.999999999
4	28.00000000	28.00000000
5	−164.0000000	−164.0000000
6		64.000000
7		−13376.00000
8		47248.0000
9		−858224.000
10		13829824.00
11		−112705856.0
⋮		—
⋮		—

aims to be realistic. If a still larger safety margin is desired, the estimate can be increased further.

5. TEST EXAMPLES

Table I shows the results that were obtained when the routine was applied to six different simple test functions. These calculations were performed on a CDC 7600 computer.

Three further examples follow.

Example 1. Consider

$$f(\zeta) = \frac{e^{\zeta}}{(\sin \zeta)^3 + (\cos \zeta)^3}.$$

All derivatives of this function at $\zeta = 0$ are integers. We obtain the values for $f^{(k)}(0)$ shown in Table II. The value $N = 51$ gives $[f^{(50)}(\zeta)]_{\zeta=0} = 1.46483674605 \times 10^{70}$, estimated by the routine to be correct to 11.4 decimal places. (Execution in quadruple precision on an IBM 3032 computer gives the value $0.14648367459103292022510999 56 \times 10^{70}$ with the last one or two digits estimated as uncertain. I thank one of the referees for quoting the value $0.1464836745910329 \times 10^{70}$ obtained by the formula manipulation system MACSYMA.)

A driver program for this example is included with the algorithm.

Example 2. Calculation of the first 15 Bernoulli numbers directly from the generating function

$$f(\zeta) = \zeta\left(\frac{1}{2} + \frac{1}{e^{\zeta} - 1}\right) = 1 + B_1 \frac{\zeta^2}{2!} - B_2 \frac{\zeta^4}{4!} + B_3 \frac{\zeta^6}{6!} - B_4 \frac{\zeta^8}{8!} + - \cdots$$

$$B_k = (-1)^{k+1} f^{(2k)}(0), \qquad k = 1, 2, 3, \ldots.$$

Table III

Exact	Computed	Relative error
$B_1 = 1/6$	$0.166666666667 \times 10^{0}$	0.27×10^{-12}
$B_2 = 1/30$	$0.333333333333 \times 10^{-1}$	0.42×10^{-12}
$B_3 = 1/42$	$0.238095238095 \times 10^{-1}$	0.34×10^{-12}
$B_4 = 1/30$	$0.333333333333 \times 10^{-1}$	0.18×10^{-12}
$B_5 = 5/66$	$0.757575757576 \times 10^{-1}$	0.64×10^{-13}
$B_6 = 691/2730$	$0.253113553114 \times 10^{0}$	-0.21×10^{-13}
$B_7 = 7/6$	$0.116666666667 \times 10^{1}$	-0.18×10^{-12}
$B_8 = 3617/510$	$0.709215686274 \times 10^{1}$	-0.23×10^{-12}
$B_9 = 43867/798$	$0.549711779449 \times 10^{2}$	-0.21×10^{-12}
$B_{10} = 174611/330$	$0.529124242424 \times 10^{4}$	-0.43×10^{-12}
$B_{11} = 854513/138$	$0.619212318840 \times 10^{4}$	-0.81×10^{-12}
$B_{12} = 236364091/2730$	$0.865802531135 \times 10^{5}$	-0.51×10^{-12}
$B_{13} = 8553103/6$	$0.142551716667 \times 10^{7}$	-0.12×10^{-11}
$B_{14} = 23749461029/870$	$0.272982310678 \times 10^{8}$	-0.11×10^{-11}
$B_{15} = 8615841276005/14322$	$0.601580873900 \times 10^{9}$	-0.17×10^{-11}

Evaluating 31 terms, the Bernoulli numbers are obtained immediately from every second coefficient (see Table III).

Example 3. Comparison between the present routine CPSC and ACM Algorithm 413 (ENTCAF) [8].

The previous example (evaluating the first 15 Bernoulli numbers) was run with both routines on a CDC CYBER 203. Execution times and the largest relative errors were observed for different choices of user-supplied radii. There is a version of ENTCAF called ENTCRE for the case of a function that is real on the real axis and is expanded at a real point. It uses symmetries to save almost a factor of 2 in execution time. A real version of CPSC could be similarly written. Although the example chosen is real, we use the complex versions of both algorithms to obtain a comparison between the methods. Several parameters have to be supplied to ENTCAF to guide its calculations. We specified that it should attempt to reach an absolute accuracy of 10^{-10} in the "normalized" Taylor coefficients (relative accuracy cannot be specified), and that if this level was not possible, it should do the best it could. Further, most 1024 function evaluations were permitted for each case. Figures 5 and 6 show how computer time and the accuracy obtained varied with the supplied radius for CPSC and ENTCAF. We see that the accuracy reached by CPSC is virtually independent of the supplied radius r_0 and that the computer time has a minimum at about $r_c = 2\pi$, the radius of convergence for the Taylor expansion. ENTCAF gives a similar accuracy for r_0 between 4.6 and 6.1, with computer times between 2 and 3.5 times faster than CPSC. We note that the best r_0 now are slightly smaller than r_c, but if r_c is not known (or is infinite for entire functions), this observation is of little use in finding a good computational radius. As soon as a too large value of r_0 is used, the time needed merely to recognize a failure exceeds the time of CPSC. (A special option in ENTCAF to try to stop early if failure seemed likely increased the time when a too large r_0 was used.)

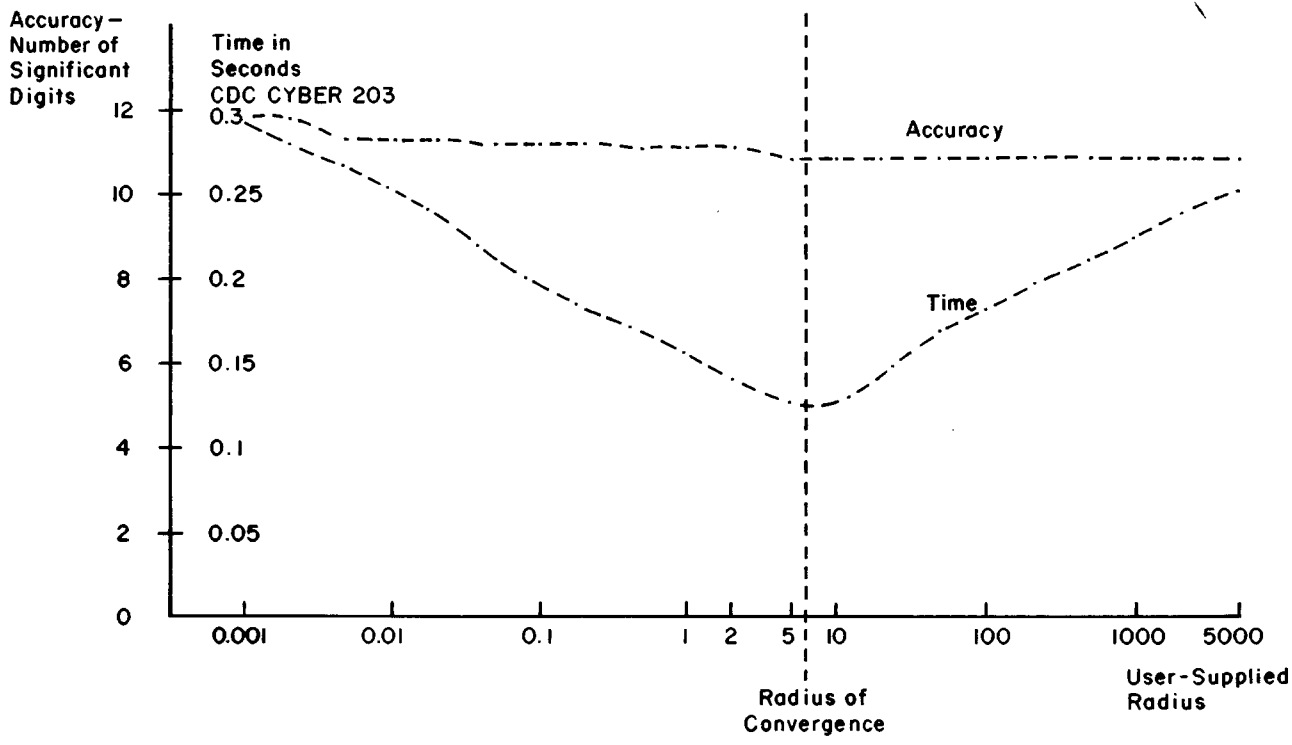

Fig. 5. Computer time and reached accuracy are shown as functions of the user-supplied radius for CPSC in the test case of finding the first 15 Bernoulli numbers.

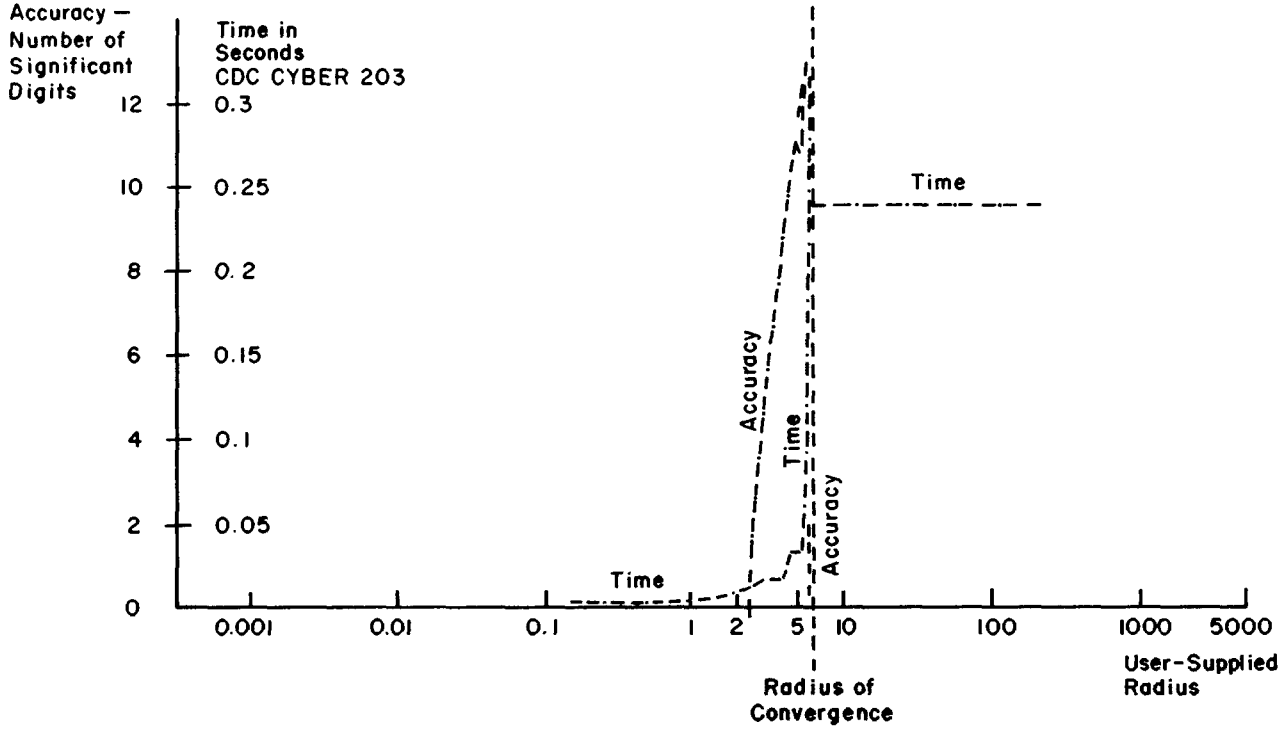

Fig. 6. Computer time and reached accuracy are shown as functions of the user-supplied radius for ENTCAF in the test case of finding the first 15 Bernoulli numbers.

REFERENCES

1. ABATE, J., AND DUBNER, H. A new method for generating power series expansion of functions. *SIAM J. Numer. Anal. 5* (1968), 102–112.
2. COOLEY, J.W., LEWIS, P.A.W., AND WELCH, P.D. The fast Fourier transform and its applications. *IEEE Trans. Educ. E-12,* 1 (1969), 27–34.
3. COOLEY, J.W., AND TUKEY, J.W. An algorithm for the machine calculation of complex Fourier series. *Math. Comput. 19* (1965), 297–301.
4. FORNBERG, B. CPSC: Complex power series coefficients. CERN-Data Handling Division Rep. DD/73/29, 1973.
5. KOPAL, Z. *Numerical Analysis.* Chapman & Hall, Ltd., London, 1955, pp. 250–251.
6. LYNESS, J.N. Differentiation formulas for analytic functions. *Math. Comput. 22* (1968), 352–362.
7. LYNESS, J.N., AND MOLER, C.B. Numerical differentiation of analytic functions. *SIAM J. Numer. Anal. 4* (1967) 202–210.
8. LYNESS, J.N., AND SANDE, G. Algorithm 413—ENTCAF and ENTCRE: Evaluation of normalized Taylor coefficients of an analytic function. *Commun. ACM 14,* 10 (Oct. 1971), 669–675.

Received August 1978; revised February 1980; accepted June 1981