A1

Algorithms and Computational Concerns

Overview

Appendix 1 presents some of the basic algorithms for computing with polynomials and rational functions and discusses some of the complexity issues. Included is a discussion of root finding methods. It requires very little background and can essentially be read independently.

Algorithms and Computational Concerns

Polynomials lend themselves to computation perhaps more than any other object of analysis. Algorithms that involve special functions, differential equations, series, and the like usually must reduce at some point to a finite polynomial or rational approximation or truncation. This often allows analytic problems to be reduced to algebraic ones. This appendix will present, as a series of exercises, some of the principal algorithmic concerns. The reader is encouraged to experiment with the algorithms. With current technology this is most comfortably done in any of the large symbolic manipulation packages available. Code, actual or schematic, is not presented. Indeed, methods rather than algorithms are presented. Current "practical" best methods date quickly in this rapidly evolving area. It is also the authors' belief that today's theoretical curiosities may be vital for tomorrow's

algorithms as larger instances are calculated on faster machines. Historically we have already seen this happen repeatedly with algorithms such as the fast Fourier transform algorithm.

One of the most interesting lessons to be learned from the last few decades of revitalized interest in computational mathematics is that many of the most familiar mathematical algorithms, such as how to multiply large numbers, were very poorly understood, and indeed, still are incompletely analyzed. Very many of the familiar processes of mathematics, such as multiplication of large numbers or computation of determinants, can be computed far more expeditiously than allowed by the usual "school" algorithms. See, for example, Aho, Hopcroft, and Ullman [74]; Bini and Pan [92]; Borodin and Munro [75]; Borwein and Borwein [87]; Brent [74]; Knuth [81]; Pan [92]; Smale [85]; and Wilf [86] for the complexity side of the following exercises.

E.1 Complexity and Recursion. We are concerned with measuring the size of an algorithm given an input of a certain length. Unfortunately, there are many different ways of measuring this. (One could, for example, use the length of the tapes of some well-defined instantiation of a Turing machine.) We will settle for less. The measure of input size will usually be chosen to be a natural one, so for polynomials of degree n, the measure will often be n. The complexity measure then depends a bit on the problem. For example, it might count the number of additions of coefficients (we do not distinguish subtraction from addition) and multiplications of coefficients required to evaluate the polynomial at a point. (So, for example, by Horner's rule $O(n)$ operations suffice.) Care is already required to count naturally. Note that we have not specified the size of the coefficients (this may or may not be reasonable) and so we could cheat on addition of coefficients by doing two additions as one addition of twice the length. (Since $a+b$ and $c+d$ can be decoded from $(10^m a + c) + (10^m b + d)$, where m is larger than the number of digits in any of a, b, c, or d.) It is more reasonable in this context to fix a precision (or to think of working to infinite precision or over some polynomial ring). Our cases are fairly simple, and the measures should be clear in context.

We adopt the following notations:

$$f(n) = O(g(n)) \quad \text{means} \quad \limsup_{n\to\infty} \frac{f(n)}{g(n)} < \infty$$

and

$$f(n) = \Omega(g(n)) \quad \text{means} \quad \limsup_{n\to\infty} \frac{g(n)}{f(n)} < \infty\,.$$

Parts of these exercises are reprinted from Borwein and Borwein [87], with permission from Wiley.

So the first measure gives an upper bound, while the second gives a lower bound.

Many algorithms are analyzed recursively. For example, addition of two polynomials of degree at most $2n$ reduces to two additions of polynomials of degree at most n, plus perhaps an "overhead" for reassembling the pieces. In other words, for the complexity of addition, we have

$$A(2n) \leq 2A(n) + c\,,$$

from which one can deduce that

$$A(n) = O(n)\,.$$

This general recursive strategy of breaking a problem in half is often called "divide and conquer."

We introduce the following functions. Here n is the maximum degree of the polynomials p and q. Additions, multiplications, and so on are performed in the underlying field of coefficients (in our case $\mathbb{C}$ or $\mathbb{R}$) and are all performed to some predetermined fixed precision (possibly infinite).

$$\begin{aligned}
A(n) &:= \text{the maximum number of } \pm\,,\ \times\,,\ \div\,, \text{ to compute } p \pm q\,;\\
M(n) &:= \text{the maximum number of } \pm\,,\ \times\,,\ \div\,, \text{ to compute } pq\,;\\
e(n) &:= \text{the maximum number of } \pm\,,\ \times\,,\ \div\,, \text{to evaluate}\\
&\qquad p(\alpha) \text{ for an arbitrary fixed } \alpha \in \mathbb{C};\\
E(n) &:= \text{the maximum number of } \pm\,,\ \times\,,\ \div\,, \text{to evaluate}\\
&\qquad p(\alpha_1), \ldots, p(\alpha_n) \text{ for arbitrary fixed } \alpha_1, \ldots, \alpha_n \in \mathbb{C}.
\end{aligned}$$

These are the complexity functions for polynomial addition, polynomial multiplication, polynomial evaluation at a single point, and polynomial evaluation at n points, respectively. The input for the computation is the coefficients (and the evaluation points for $e(n)$ and $E(n)$). So the input may be considered to be in $\mathbb{C}^{n+1}$ (or more generally an $(n+1)$-dimensional vector space over an infinite field). In the first two cases the output is the sequence of coefficients. In the last two cases, respectively, the output is the evaluation and the sequence of evaluations.

a] Show that usual algorithms give

$$\begin{aligned}
A(n) &\leq 2n + 2 = O(n)\,,\\
M(n) &= O(n^2)\,,\\
e(n) &= O(n) \qquad \text{(Horner's rule)}\,,\\
E(n) &= O(n^2)\,.
\end{aligned}$$

E.2 and E.3 of this appendix provide better upper bounds for the last three functions above.

b] Show that

$$\begin{aligned} A(n) &\geq n+1\,, \\ M(n) &\geq n+1\,, \\ e(n) &\geq n+1\,, \\ E(n) &\geq n+1\,. \end{aligned}$$

Hint: In all cases this is a uniqueness argument. At least one operation must be performed for each coefficient, otherwise the algorithm will not distinguish various different sequences of input. □

c] Some Recursive Bounds. Let $a, b > 0$ and $c, d > 1$. Suppose that f is monotone on $(0, \infty)$.

If $f(n) \leq af(n/c) + bn$, then

$$\begin{aligned} f(n) &= O(n) && \text{if } a < c\,, \\ f(n) &= O(n \log n) && \text{if } a = c\,, \\ f(n) &= O(n^{\log_c a}) && \text{if } a > c\,. \end{aligned}$$

If $f(n) \leq df(n/d) + bn(\log n)^{c-1}$, then

$$f(n) = O(n(\log n)^c)\,.$$

Hint: Analyze the equality case. Then establish the general principle that the equality solution is the maximal solution. □

E.2 The (Finite) Fast Fourier Transform (FFT). This is undoubtedly one of the most widely used algorithms. It has, in its various forms, tremendous practical and theoretical applications.

Let w be a primitive $(n+1)$th root of unity in either $\mathbb{C}$ or a finite field $\mathbb{F}_m$, that is, $w^{n+1} = 1$ and $w^k \neq 1$ for $k = 1, 2, \ldots, n$. In the complex case we may take $w := e^{2\pi i/(n+1)}$. Consider the following two problems.

Interpolation Problem. Given $n+1$ numbers, $\alpha_0, \alpha_1, \ldots, \alpha_n$, find the coefficients of the unique polynomial

$$p(z) := a_0 + a_1 z + \cdots + a_n z^n$$

of degree n that satisfies

$$p(w^k) = \alpha_k\,, \qquad k = 0, 1, \ldots, n\,.$$

Evaluation Problem. Given the coefficients of a polynomial p_n of degree at most n, calculate the $n+1$ values

$$p(w^k)\,, \qquad k = 0, 1, \ldots, n\,.$$

These are the two directions of the *(finite) Fourier transform*. The classical approaches to either part of the Fourier transform problem have complexity at least cn^2. This is the complexity, for example, of evaluating p_n at $n+1$ points using Horner's rule. Both directions can, however, be solved with complexity $O(n \log n)$.

a] If $n+1 = 2^m$, then both the interpolation and the evaluation problem have complexity $O(n \log n)$. (Here we are counting the number of additions and multiplications in the underlying coefficient field, which for most of our purposes is $\mathbb{C}$.)

Outline. We treat the evaluation first. Suppose

$$p(x) := a_0 + a_1 x + \cdots + a_n x^n \,.$$

Let

$$q(x^2) := a_0 + a_2 x^2 + a_4 x^4 + \cdots + a_{n-1} x^{n-1}$$

and

$$x\, r(x^2) := x(a_1 + a_3 x^2 + \cdots + a_n x^{n-1}) \,.$$

Then, with $y := x^2$,

$$p(x) = x\, r(y) + q(y) \,,$$

where r and q are both polynomials of degree at most $2^{m-1} - 1$. The observation that makes the proof work is that for w an $(n+1)$th root of unity,

$$(w^k)^2 = \left(w^{(n+1)/2 + k}\right)^2 .$$

Hence, evaluating $p(x)$ at the $n+1$ roots of unity reduces to evaluating r and q each at the $\frac{1}{2}(n+1)$ points $(w^2)^1, (w^2)^2, \ldots, (w^2)^{(n+1)/2}$ and amalgamating the results. Observe that w^2 is a primitive (2^{m-1})th root of unity, so we can iterate this process. Let $F(2^m)$ be the number of additions and multiplications required to evaluate a polynomial of degree at most $2^m - 1$ at the 2^m points w^k, $k = 1, 2, \ldots, 2^m$, where w is a primitive (2^m)th root of unity. Then

$$F(1) = 1 \qquad \text{and} \qquad F(2^m) = 2F(2^{m-1}) + 2 \cdot 2^m \,, \qquad m = 1, 2, \ldots .$$

The second term comes from the single addition and multiplication required to calculate each $p(w^k)$ from $r(w^{2k})$ and $q(w^{2k})$. This recursion solves as

$$F(2^m) = 2^{m+1} m \,,$$

and the bound for the evaluation problem is established.

The interpolation problem is equivalent to evaluation. This can be seen as follows. Let w be a primitive $(n+1)$th root of unity, and let

$$W := \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & w & w^2 & \dots & w^n \\ 1 & w^2 & w^4 & \dots & w^{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & w^n & w^{2n} & \dots & w^{n^2} \end{pmatrix} .$$

Then

$$W^{-1} = \frac{1}{n+1} \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & w^{-1} & w^{-2} & \dots & w^{-n} \\ 1 & w^{-2} & w^{-4} & \dots & w^{-2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & w^{-n} & w^{-2n} & \dots & w^{-n^2} \end{pmatrix}$$

and w^{-1} is also a primitive $(n+1)$th root of unity. The interpolation problem can be formulated as follows. Find $(a_0, \dots, a_n)$ so that

$$W(a_0, a_1, \dots, a_n) = (\alpha_0, \alpha_1, \dots, \alpha_n).$$

However, this can be solved by

$$W^{-1}(\alpha_0, \alpha_1, \dots, \alpha_n) = (a_0, a_1, \dots, a_n)\,,$$

which is exactly the evaluation problem. □

See Borodin and Munro [75] and Borwein and Borwein [87]. Versions of FFT exist in a plethora of shapes and sizes. We have just exposed the tip of the iceberg.

As an application we construct a fast polynomial multiplication.

b] Fast Polynomial Multiplication. Suppose the polynomials p, q of degree at most $n-1$ are given. Compute the coefficients of the product pq as follows:

b1] Use an FFT to evaluate p and q at the primitive $(2n)$th roots of unity $w^1, w^2, \dots, w^{2n}$.

b2] Form

$$p(w^k)q(w^k)\,, \qquad k = 1, 2, \dots, 2n\,.$$

b3] Find the coefficients of the product pq by solving the interpolation problem once again by using the FFT. Show that this algorithm requires

$$O(n \log n)$$

additions, multiplications, and divisions (of complex numbers) and therefore $M(n) = O(n \log n)$.

This is the best-known upper bound on the serial complexity of polynomial multiplication. The only known lower bound is the trivial one $O(n)$. In parallel (on a PRAM) polynomial multiplication can be done in $O(\log n)$ time on $O(N)$ processors; see Pan [92]. The same bounds apply for the FFT in a].

E.3 Other Elementary Operations.

a] Fast Polynomial Division. For polynomials p of degree n and q of degree $m \le n$, it is possible to find polynomials u and r with $\deg(r) < \deg(q)$ such that

$$p(x) = u(x)q(x) + r(x)$$

in $O(n \log n)$ additions and multiplications (of complex numbers).

Outline. Simplify by observing that it suffices to calculate u since r may then be computed by E.2 b]. If we replace x by $1/x$ then

$$\frac{p(x^{-1})}{q(x^{-1})} = u(x^{-1}) + \frac{r(x^{-1})}{q(x^{-1})}$$

and so, for some $h \ge 1$,

$$\frac{p^*(x)}{q^*(x)} = u^*(x) + x^{n-m+h}\frac{r^*(x)}{q^*(x)}, \qquad \text{where} \qquad v^*(x) := x^{\deg(v)}v(x^{-1})\,.$$

To calculate u^* (and hence to calculate u) it suffices to calculate the first $n-m$ $(= \deg(u))$ Taylor coefficients of $1/q^*$. This can be done by Newton's method (see the next exercise) as follows: Suppose $\deg s_i = j-1$ and

$$\frac{1}{q^*(x)} - s_i(x) = O(x^j)\,.$$

Establish that

$$\frac{1}{q^*(x)} - \left[2s_i(x) - s_i^2(x)q^*(x)\right] = \frac{1}{q^*(x)}\left[1 - s_i(x)q^*(x)\right]^2 = O(x^{2j})\,.$$

Note that we may assume $q^*(0) \neq 0$. Now the computation of

$$s_{i+1} := 2s_i - s_i^2 q^*$$

can be performed by using an FFT-based polynomial multiplication and it needs only be performed by using the first $2j-1$ coefficients of q^* and s_i. By starting with an appropriate first estimate of s_0 (say, $s_0(x) := 1/q^*(0)$), and proceeding inductively as above (doubling the number of coefficients used at each stage) we can show that the required number of terms of the expansion can be calculated in $O(n \log n)$ additions and multiplications (of complex numbers).

b] Fast Reversion of Power Series. Let

$$f(x) = \sum_{k=0}^{\infty} a_k x^k\,, \qquad a_0 \neq 0$$

be a formal power series with given coefficients. Show that the first n coefficients of the formal Taylor expansion of $1/f$ can be computed in $O(n \log n)$ additions, multiplications, and divisions (of complex numbers).

c] Fast Polynomial Expansion. Given $\alpha_1, \alpha_2, \ldots, \alpha_n$ show that the coefficients of $\prod_{i=1}^{n}(x - \alpha_i)$ can all be calculated in $O(n(\log n)^2)$ additions, multiplications, and divisions (of complex numbers).

Hint: Proceed recursively by dividing the problem into two parts of roughly half the size. Recombine the pieces using part b] of the previous exercise. □

d] Fast Polynomial Expansion at Arbitrary Points. Given a polynomial p of degree at most n, and $n+1$ distinct points $x_0, x_1, \ldots, x_n$, show that $p(x_0), p(x_1), \ldots, p(x_n)$ can all be evaluated in $O(n(\log n)^2)$ multiplications and additions.

Hint: Let

$$q_1(x) := \prod_{i=0}^{\lfloor n/2 \rfloor - 1} (x - x_i)$$

and let r_1 be the remainder on dividing p by q_1. Note that $r_1(x_i) = p(x_i)$ for each $i < n/2$. Similarly, use

$$q_2(x) := \prod_{\lfloor n/2 \rfloor}^{n} (x - x_i).$$

Thus two divisions reduce the problem to two problems of half the size. □

e] Extend d] to rational functions.

f] Evaluation of x^n. The S-and-X binary method for calculating x^n is the following algorithm. Suppose n has binary representation $\delta_0\delta_1\delta_2\ldots\delta_k$ with $\delta_0 = 1$. Given symbols S and X, define

$$S_i := \begin{cases} SX & \text{if } \delta_i = 1 \\ S & \text{if } \delta_i = 0 \end{cases}$$

and construct the rule

$$S_1 S_2 \cdots S_k \,.$$

Now let S be the operation of squaring, and let X be the operation of multiplying by x. Let $S_1 S_2 \cdots S_k$ operate from left to right beginning with x. For example, for $n = 27$,

$$\delta_0\delta_1\delta_2\delta_3\delta_4 = 11011$$

and

$$S_1 S_2 S_3 S_4 = (SX)(S)(SX)(SX)\,.$$

The sequence of calculations of x^{27} is then

$$x \to x^2 \to x^3 \to x^6 \to x^{12} \to x^{13} \to x^{26} \to x^{27}.$$

f1] Prove that the above method computes x^n and observe that it only requires storing x, n, and one partial product.

f2] Show that the number of multiplications in the S-and-X method is less than $2\lfloor \log_2 n \rfloor$.

f3] Show that the S-and-X method is optimal for computation of x^{2m} (considering only multiplications).

f4] Show that the S-and-X method is not optimal for computing x^{15}.

An extended discussion of this interesting and old problem is presented in Knuth [81].

Much further material on the complexity of polynomial operations and complexity generally may be found in Aho, Hopcroft, and Ulman [74]; Borodin and Munro [75]; Pan [92]; and Wilf [86].

E.4 Newton's Method. One of the very useful algorithms, both in theory and practice, for zero finding is Newton's method.

a] Suppose f is analytic in a (complex) neighborhood of z_0, and suppose $f(z_0) = 0$ and $f'(z_0) \neq 0$. Show that the iteration

$$x_{n+1} := x_n - \frac{f(x_n)}{f'(x_n)}$$

converges locally uniformly quadratically, that is, with a constant c independent of n,

$$|x_{n+1} - z_0| < c\,|x_n - z_0|^2$$

for initial values x_0 in some neighborhood of z_0. As before, we call this *locally quadratic convergence.*

Hint: Note that

$$f(x_n) = f(z_0) + (x_n - z_0)f'(z_0) + O((x_n - z_0)^2)\,,$$

which implies

$$x_{n+1} - z_0 = (x_n - z_0)\left[\frac{f'(x_n) - f'(z_0)}{f'(x_n)}\right] + O((x_n - z_0)^2)\,.$$

□

b] Show that the iteration

$$x_{n+1} := x_n + \frac{f(x_n)}{f'(x_n)}$$

converges locally quadratically to the simple poles of a meromorphic function f. Note that this is Newton's method with the sign changed.

c] The iteration

$$x_{n+1} := x_n - \frac{f(x_n)f'(x_n)}{f'(x_n)^2 - f(x_n)f''(x_n)}$$

converges locally quadratically to a zero of an analytic f independent of its multiplicity; see Henrici [74].

d] Let $g := f^{-1}$. The iteration

$$x_{n+1} := x_n + (n+1)\,\frac{g^{(n)}(x_n)}{g^{(n+1)}(x_n)}$$

converges locally uniformly to a zero of an analytic function f with $(n+2)$th order. Newton's method is $n := 0$; Halley's method is $n := 1$ (see Householder [70]).

Newton's method and its variants work tremendously well provided that a good starting value can be found. This is a problem. On a real interval a bisection method can be used initially to localize the zeros. In the plane, life is more complicated as is seen in the next exercise. Another drawback to Newton's method is the need to compute the derivative. Of course, this is not a problem for polynomials, but in a general setting it is usually replaced by an approximation such as

$$\frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}$$

(which yields the so-called secant method).

e] Consider Newton's method for computing $\sqrt{x}$ starting with $x_0 := 1$. This gives

$$x_{n+1} := \frac{1}{2}\left(x_n + \frac{x}{x_n}\right).$$

Show that $r_n(x) := x_{n+1}$ is a rational function in x with numerator of degree 2^n and denominator of degree $2^n - 1$. Show that $r_n(x) - \sqrt{x}$ has a zero of order 2^{n+1} at 1. So r_n is in fact the $(2^n, 2^n - 1)$ Padé approximant to $\sqrt{x}$. (This implies, though not obviously, that r_n has all real negative roots and poles.)

An attractive feature of Newton's method is that it is "self-correcting." So, for example, to compute a root to a large precision, one can start at a small precision and double the precision at each iteration. This is a substantial savings both practically and theoretically. The same feature applies to Newton's method solutions over formal power series, as in E.3 a]. This allows for doubling the number of terms used at each stage. Much additional material on Newton's method is available in Borwein and Borwein [87], Henrici [74], Householder [70], and Traub [82].

E.5 Newton's Method in Many Variables.

a] Let $f : \mathbb{C}^n \to \mathbb{C}^n$, and suppose f has Jacobian

$$J := \left| \frac{\partial f_i}{\partial z_j} \right| ,$$

where $f := (f_1, f_2, \ldots , f_n)$ with $f_i : \mathbb{C}^n \to \mathbb{C}$. Let

$$\mathbf{x} := (x_1, x_2, \ldots , x_n) \in \mathbb{C}^n .$$

The function $J(\mathbf{x})$ is the Jacobian evaluated at $\mathbf{x}$. Then Newton's method becomes

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{s}_k$$

where $\mathbf{s}_k$ solves $J(\mathbf{x}_k)\mathbf{s}_k = f(\mathbf{x}_k)$. This iteration converges locally uniformly quadratically to a zero $\mathbf{z_0}$ of f, that is, with a constant c independent of n,

$$|\mathbf{x}_{n+1} - \mathbf{z}_0| < c\,|\mathbf{x}_n - \mathbf{z}_0|^2$$

for initial values $\mathbf{x_0}$ in some neighborhood of $\mathbf{z_0}$, provided in a neighborhood of $\mathbf{z}_0$, f is continuously differentiable, J^{-1} exists and is bounded in norm, and J satisfies a Lipschitz condition. We call this *locally quadratic convergence*. For a polynomial f, we require only that J^{-1} exists in a neighborhood of the zero $\mathbf{z}_0$. (For examples and detail, see Dennis and Schnabel [83].)

b] Finding All Zeros of a Polynomial. Let

$$p(z) := a_0 + a_1 z + \cdots + a_n z^n , \qquad a_n := 1 .$$

Let $f_i(x_1, \ldots , x_n)$ be the ith coefficient of

$$g(x) := \prod_{i=1}^{n} (x - x_i) - \sum_{i=0}^{n} a_i x^i$$

and let

$$f(\mathbf{x}) := (f_1, f_2, \ldots , f_n) .$$

Then the iteration of part a] applied to f converges locally quadratically to $\mathbf{z}_0 := (z_1, z_2, \ldots, z_n)$, where $z_1, z_2, \ldots, z_n$ are the zeros of p, provided the zeros of p are distinct.

c] Another Approach to Finding All Zeros. Let $x_j(k)$ denote the kth approximation to the jth root of p, where p is a polynomial of degree n with n distinct roots. Let

$$x_j(k+1) = x_j(k) - \frac{p(x_j(k))}{\prod_{i \neq j}^{k} (x_j(k) - x_i(x))}, \qquad j = 1, 2, \ldots, n.$$

Show that for sufficiently good choice of $(x_0(1), \ldots, x_0(k))$, the above iteration converges locally quadratically to a sequence of the n distinct zeros of p.

Hint: This is really just the single variable version of Newton's method for each root, where the derivative is approximated by the derivative of the kth estimate. □

d] Observe that the iteration of part c] fails to converge for $p(x) := x^n - 1$, $n > 2$, if the starting values are all taken to be real.

In practice, the iteration of part c] works rather well for reasonably chosen starting values. One can use the techniques of the next exercise to localize the zeros first. With care, an algorithm can be given that computes a zero of a polynomial with an error $< 2^{-b}$ in $O(n \log b \log n)$ time and all zeros in $O(n^2 \log b \log n)$ time; see Pan [92]. On a parallel machine (PRAM) an algorithm requiring $O(\log^3(nb))$ time and $O(nb)^{O(1)}$ processors can be given for computing all zeros.

e] Modify the iteration of E.4 c] as given in the previous exercise to get a method that computes all roots even in the presence of repeated roots.

For further discussion, see Aberth [73], Durand [60], Kerner [66], and Werner [82].

E.6 Localizing Zeros.

a] Cauchy Indices. Let r be a real rational function with a real pole α. The *Cauchy index* of r at α is

$$\begin{cases} 1 & \text{if} \quad \lim_{x \to \alpha -} r(x) = -\infty \quad \text{and} \quad \lim_{x \to \alpha +} r(x) = \infty \\ -1 & \text{if} \quad \lim_{x \to \alpha -} r(x) = \infty \quad \text{and} \quad \lim_{x \to \alpha +} r(x) = -\infty \\ 0 & \text{otherwise.} \end{cases}$$

The Cauchy index of r on an interval $[a, b]$ is the sum of the Cauchy indices of the poles of r in (a, b). (We demand that neither a nor b be poles of r.) We denote this by $I_a^b(r)$.

b] The Euclidean Algorithm. Let p_0 and p_1 be nonzero polynomials. Define polynomials $p_0, p_1, \ldots, p_m$ and $q_1, q_2, \ldots, q_m$ (by the usual division algorithm) so that

$$\begin{aligned} p_0(z) &= p_1(z)q_1(z) - p_2(z)\,, \qquad \deg(p_2) < \deg(p_1)\,; \\ p_1(z) &= p_2(z)q_2(z) - p_3(z)\,, \qquad \deg(p_3) < \deg(p_2)\,; \\ &\vdots \\ p_{m-1}(z) &= p_m(z)q_m(z) - 0\,; \end{aligned}$$

the algorithm stops the first time that the remainder is zero. This is called the *Euclidean algorithm.* Show that p_m is the greatest common factor of p_0 and p_1.

c] Let $p_0, p_1, \ldots, p_m$ be the polynomials generated by the Euclidean algorithm in part b]. Let $P_i := p_i/p_m$, $i = 0, 1, \ldots, m$. Suppose $p_0(a)p_0(b) \neq 0$. Show that if $P_k(\gamma) = 0$ for some k and $a \leq \gamma \leq \beta$, then

$$P_1(\gamma) \neq 0 \quad \text{if} \quad k = 0$$

and

$$P_{k-1}(\gamma)P_{k+1}(\gamma) < 0 \quad \text{if} \quad 1 \leq k \leq m-1\,.$$

d] Show that for real polynomials p_0 and p_1 with $p_0(a)p_0(b) \neq 0$,

$$I_a^b(p_1/p_0) = v(a) - v(b)\,, \qquad a < b\,,$$

where $v(\alpha)$ is the number of sign changes in the sequence

$$(p_0(\alpha), p_1(\alpha), \ldots, p_m(\alpha))$$

and where the polynomials p_i are generated by the Euclidean algorithm as in part b]. (As before, by a sign change we mean that $p_i(\alpha)p_{i+k}(\alpha) < 0$ and $p_{i+1}(\alpha) = p_{i+2}(\alpha) = \cdots = p_{i+k-1}(\alpha) = 0$.)

Hint: Without loss of generality, we may assume that $p_m = 1$ (why?), so $P_k = p_k$ for each k. First, note that $v(x)$ may change magnitude only if $p_i(x) = 0$ for some i. By continuity of the polynomials p_i, and by part c], $v(x)$ is constant on any subinterval of $[a, b]$ that does not contain a zero of p_0.

We are now reduced to considering the behavior of $v(x)$ at the zeros of p_0. Consider the various possibilities for the behavior of p_1/p_0 at the zeros of p_0 by considering the four possible changes of signs of p_0 and p_1 at the zeros of p_0 and the effect this has on the Cauchy index and the increase and decrease of $v(x)$. (Note that $v(x)$ decreases by 1 if the Cauchy index is 1 while $v(x)$ increases by 1 if the Cauchy index is -1.) □

e] Zeros on an Interval. Suppose p is a real polynomial and $p(a)p(b) \neq 0$. Then $I_a^b(p'/p)$ equals the number of distinct zeros of p in $[a, b]$. This also equals $v(a) - v(b)$, where $v(x)$ is as in d] with $p_0 := p$ and $p_1 := p'$.

Hint: Use

$$\frac{p'(z)}{p(z)} = \sum \frac{m_k}{z - \zeta_k}$$

and note that each distinct pole in (a, b) contributes $+1$ to the Cauchy index. □

f] Show that the number of zeros of a real polynomial on the real line equals

$$\lim_{a \to \infty} \left(v(-a) - v(a) \right),$$

where $v(x)$ is computed as in part e]. (That is, $v(x)$ is the number of sign changes in the sequence $(p_0, p_1, \ldots, p_m)$ generated by the Euclidean algorithm with $p_0 := p$ and $p_1 := p'$.)

g] The Number of Zeros in $H := \{z \in \mathbb{C} : \mathrm{Im}(z) > 0\}$. Suppose that the monic polynomial $p \in \mathcal{P}_n^c$ has (exactly) k real zeros counting multiplicities. Write

$$p(x) = r(x) + i\, s(x), \qquad r, s \in \mathcal{P}_n .$$

Then the number of zeros of p in H is

$$\frac{1}{2} \left[n - k - I_{-\infty}^{\infty}(s/r) \right],$$

and this can be computed as in part d] by using

$$I_{-\infty}^{\infty}(s/r) = \lim_{\alpha \to \infty} I_{-\alpha}^{\alpha}(s/r) .$$

Hint: First consider the case where $k = 0$. Consider p on a counterclockwise semicircular contour with base $[-\alpha, \alpha]$ and radius α. Consider the argument of p as the contour is traversed. On the half-circle, for large α, the argument increases by $n\pi$ asymptotically; while on the axis the change is $-\pi I_{-\alpha}^{\alpha}(s/r)$, from which the result follows. □

h] Budan-Fourier Theorem. *Let $p \in \mathcal{P}_n$. Let $V(x)$ be the number of sign changes in the sequence*

$$\left(p(x), p'(x), \ldots, p^{(n)}(x) \right) .$$

Then the number of zeros of p in the interval $[a, b]$, counting multiplicities, is $V(a) - V(b) - 2m$ for some nonnegative integer m.

We have followed Henrici [74] in this discussion.

E.7 Zeros in a Disk.

a] The transform $\dfrac{z + i}{1 + iz}$ maps the unit circle to the real axis and maps the open unit disk D to the upper half-plane $\{z \in \mathbb{C} : \mathrm{Im}(z) > 0\}$. So the algorithms of the previous exercise apply after transformation.

b] Zero Counting by Winding Number. Let $\omega_m := e^{2\pi i/m}$, where m is a positive integer. If $p \in \mathcal{P}_n^c$ then

$$z_p := \frac{1}{2\pi i}\int_{\partial D} \frac{p'(z)}{p(z)}\,dz \sim \sum_{k=1}^{m} \omega_m^k \frac{p'(\omega_m^k)}{p(\omega_m^k)}$$

counts the number of zeros of p in the open unit disk D (assuming no zeros on the boundary ∂D). More precisely, we have

$$z_p = \lim_{m\to\infty} \sum_{k=1}^{m} \omega_m^k \frac{p'(\omega_m^k)}{p(\omega_m^k)}\,.$$

(Note that this lends itself to rapid evaluation by FFT methods.)

c] Show that

$$\frac{1}{2\pi i}\int_{\partial D} \frac{1}{z-\alpha} = \sum_{k=1}^{m} \frac{\omega_m^k}{\omega_m^k - \alpha} + \epsilon_m(\alpha)\,,$$

where

$$\epsilon_m(\alpha) \le \frac{|\alpha|^{m-1}}{1-|\alpha|} \qquad \text{if} \quad |\alpha| < 1$$

and

$$\epsilon_m(\alpha) \le \frac{|\alpha|^{-m-1}}{1-|\alpha|^{-1}} \qquad \text{if} \quad |\alpha| > 1\,.$$

Hint: Write

$$\frac{1}{z-\alpha} = -\frac{1}{\alpha}\sum_{k=0}^{\infty} \frac{z^k}{\alpha^k}$$

and use the fact that

$$0 = \int_{\partial D} p(z)\,dz = \sum_{k=1}^{m} \omega_m^k p(\omega_m^k)$$

for every $p \in \mathcal{P}_{m-1}^c$. □

d] If $p \in \mathcal{P}_n^c$ has no zeros in the annulus $\zeta \le z \le 1/\zeta$ and if m in part b] is greater than

$$\left\lfloor \frac{\log \frac{1-\zeta}{n}}{\log \zeta} \right\rfloor + 1\,,$$

then the error in estimating z_p by the sum is less than 1. So this provides an algorithm.

E.8 Computing General Chebyshev Polynomials. Given a Chebyshev system $\mathcal{M} := (f_0, \dots, f_n)$ of C^1 functions on, say, $[0,1]$ how does one compute the associated Chebyshev polynomial T_n? That is, how does one find the unique equioscillating form $T := \sum_{i=0}^{n} a_i f_i$ of Section 3.3?

a] The Remez Algorithm

Step 1. Choose

$$\mathbf{x}_0 := (0 =: x_0^{(0)} < x_1^{(0)} < \cdots < x_n^{(0)} := 1)$$

and find $P_0 \in \operatorname{span} \mathcal{M}$ such that

$$P_0(x_i^{(0)}) = (-1)^i\,, \qquad i = 0, 1, \dots, n\,.$$

Step 2. Inductively set $\mathbf{x}_m := (0 =: x_0^{(m)} < x_1^{(m)} < \cdots < x_n^{(m)} := 1)$, where

$$P'_{m-1}(x_i^{(m)}) = 0\,, \qquad i = 1, 2, \dots, n-1\,.$$

(That is, find the extrema of P_{m-1}.)

Step 3. Find $P_m \in \operatorname{span} \mathcal{M}$ with

$$P_m(x_i^{(m)}) = (-1)^i\,, \quad i = 0, 1, \dots, n\,.$$

Then, provided the initial estimate $\mathbf{x}_0$ is sufficiently good, $P_m \to T_n$ quadratically (see Veidinger [60]).

This is reasonably easy to code. It involves solving an interpolation problem in Steps 1 and 3. The zero finding at Step 2 can be done quite easily since one can find very good starting values for Newton's method, namely $(x_i^{(m-1)})_{i=0}^{n}$.

b] This algorithm modifies to solve the best approximation problem

$$\min_{p \in \operatorname{span} \mathcal{M}} \|\omega p - f\|_{[a,b]}$$

for $\omega, f \in C[a,b]$, where ω is positive on $[a,b]$. One uses the Remez agorithm to find an equioscillating form

$$P(x) = f(x) - \omega(x) \sum_{i=0}^{n} c_i f_i(x)$$

at $n+2$ points. To do this, one solves the system

$$f(x_k) - \omega(x_k) \sum_{i=0}^{n} c_i f_i(x_k) + (-1)^k \lambda\,, \qquad k = 0, 1, \dots, n+1$$

for both the c_i and λ. (This works reasonably well, provided that at each stage $\|P\|_{[a,b]}$ occurs at one of the x_k. If not, an extra point must be inserted where the maximum norm occurs and one of the original points must be dropped. This is effected in such a way as to maintain the alternations in sign of the error.) For details see Cody, Fraser, and Hart [68] and Veidinger [60].