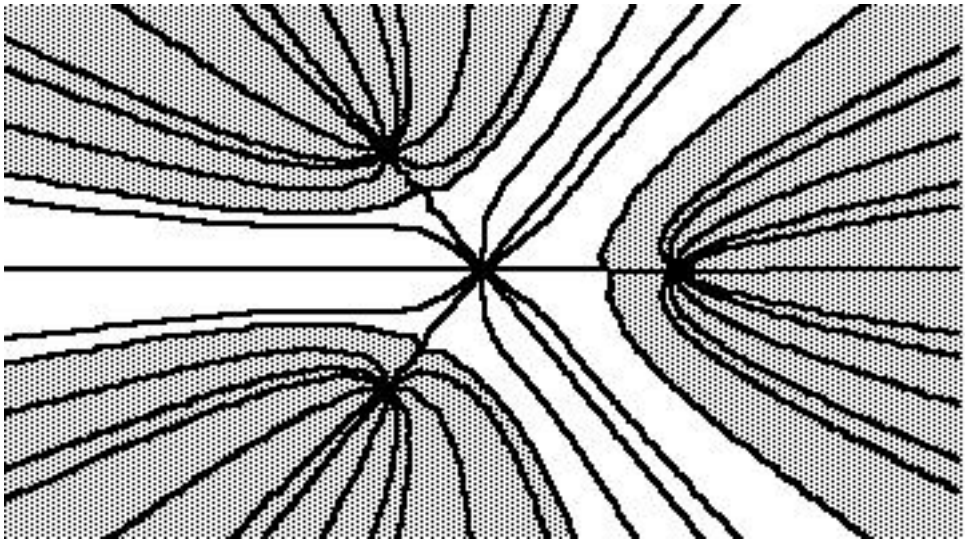
Theory of Equations

Lesson 4

by

Barry H. Dayton
Northeastern Illinois University
Chicago, IL 60625, USA

www.neiu.edu/~bhdayton/theq/



Chapter 2

Numerical Analysis

NUMERICAL SOLUTION OF POLYNOMIAL EQUATIONS

In this chapter we will discuss numerical algorithms for finding the roots of polynomials and solving other related problems.

2.1 Exact and Approximate Numbers

In modern Mathematics we use numbers in two quite different ways, exact numbers and decimal approximations.

EXACT NUMBERS

Exact numbers derived from the ancient Euclidean ideal of Mathematics as an exact science, they are still useful for theoretical considerations. Exact numbers include the integers, rational numbers and the specific numbers π and e, the base of the natural logarithms. Moreover, given exact numbers one can apply the algebraic operations of addition, subtraction, multiplication and divison, one can also apply (exact) mathematical functions.

Thus exact numbers include

$$3, \quad \frac{2}{3}, \quad \sqrt{7}, \quad \frac{3}{\sqrt{7}}, \quad 3\sqrt{\frac{1}{4}+\sqrt{3}}, \quad \sqrt[3]{5}, \quad \ln(2), \quad \cos(\pi/10)$$

Exact numbers include real and complex numbers, in some cases it may not be

evident whether an exact number is real or complex, for example:

$$\sqrt{-5+\sqrt{-11}}+\sqrt{-5-\sqrt{-11}}$$

is a real number, can you prove that?

Given two exact numbers the key question is: are they equal? It is not always possible to tell, but there are ways that often work. For example, Maple has a built in proceedure named radnormal which can usually tell if a exact number made up of radicals of rational numbers is zero. And $x = y$ if and only if $x - y = 0$. Recall that the equality relation is reflexive (always $x = x$), symmetric (if $x = y$ then $y = x$) and transitive (if $x = y$ and $y = z$ then $x = z$).

On the other hand, it is very hard to decide which of two real exact numbers is larger. And exact numbers need not be real but we saw in Chapter 1 that there is no ordering of complex numbers. So order is not a useful concept with exact numbers.

We will be using exact numbers in chapters 4,5 and 6.

DECIMAL APPROXIMATIONS

The real numbers can be represented as *infinite decimals*, that is, numbers such as $3.14159265358979323847\ldots$. While this is useful for some theoretical considerations it is not possible to represent real numbers this way for computational purposes as it would take an infinitely long amount of time to write down most numbers.

Fractions can be represented as infinite *repeating* decimals, for example $\frac{1}{3} = 0.3333333\ldots = 0.\overline{3}$ or $\frac{24}{7} = 3.428571428571\ldots = 3.\overline{428571}$. The sequence of repeating digits is called the *repetend*. Sometimes the repetend is $\overline{0}$ such as in $\frac{3}{16} = 0.1875\overline{0}$. Since there is only a finite amount of information that must be given, a few digits and the repetend, in principle we could represent fractions exactly using decimals. The following Example/Exercise shows why this is not practical:

Exercise 2.1.1 [20 points] Do the indicated operations on repeating decimals giving the answers as repeating decimals. In particular, find the repetends. You may use Maple.

1. Add $3.\overline{91} + 2.\overline{911} + 1.\overline{91111}$
2. Multiply $2.\overline{07} * 3.\overline{13}$

For these reasons we will never, in this text, use decimals as exact numbers, only as *approximations* to exact real numbers. Thus we can approximate $\pi = 3.1415927$ or $\frac{24}{7} = 3.4285714285$ or $\frac{3}{16} = 0.1875$

We will use the usual convention in rounding, for positive numbers if the $n+1^{st}$ digit to the right of the decimal point is 0,1,2,3 or 4 we round to n digits by dropping all digits to the right of the n^{th} digit. However if the $n+1^{st}$ digit is 5 or higher we drop the digits to the right of the n^{th} digit but raise the n^{th} digit by 1. In this case if the n^{th} digit is 9 we have to adjust digits further to the left. For negative numbers we apply this rounding convention to the absolute value and then put the negative sign back in.

Thus, rounding to 5 digits we see

2.53422468	rounds to 2.53422
-3.43414682	rounds to -3.43415
1.9999999	rounds to 2.00000
0.000004	rounds to 0.00000

In this text we will assume that a decimal number represents any number that rounds to it. Thus the decimal number 2.53422 represents not a specific real number but rather the half open interval $[2.53415, 2.534225) = \{x \in \boldsymbol{R} | 2.53415 \leq x < 2.534225\}$.

In our usage the numbers $\frac{1}{2}, 0.5, 0.50, 0.500$ are all different: $\frac{1}{2}$ is an exact number, 0.5 represents all numbers in the interval $[\frac{45}{100}, \frac{55}{100}) = [.45, .55)$, 0.50 represents all numbers in the interval $[0.495, 0.505)$ and 0.500 represents all numbers in $[0.4995, 0.5005)$.

Figure 2.1 shows this pictorially.

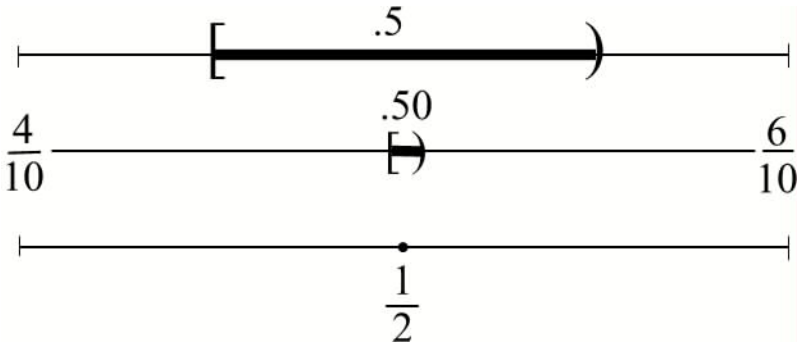


Figure 2.1: $\frac{1}{2}, .5, .50$

Note also that -0.5 represents $(-.55, -.45]$ and $0.$ represents the open interval $(-.5, .5)$ while 0.0 represents $(-.05, .05)$ and 0.00 represents $(-.005, .005)$.

With our usage, equality of decimal numbers is not a useful concept, for example we don't have the symmetry property since 0.5 represents 0.467 but 0.467 does not

represent 0.5 Rather, the key relation is order, that is given decimal approximations x, y is $x \leq y$ or is $y \leq x$? This can be easily decided in the usual way, or by seeing if $x - y \geq 0$. So when we write, for example, $\pi = 3.14159$ what we mean is that pi is *represented* by 3.14159.

Calculators and computers use a special kind of decimal representation called *floating point notation*. Here each number can be represented as a number (the *mantissa*) greater than or equal to .1 and less than 1.0 multiplied by an integral power of 10 (the power is called the characteristic). A sign, plus or minus, is also stored with the number. For example, several floating point numbers and their ordinary decimal equivalents are

$$\begin{aligned} .3451123\texttt{E}09 &= 345112300 \\ .3451123\texttt{E}02 &= 34.51123 \\ .3451123\texttt{E}00 &= .3451123 \\ .3451123\texttt{E}{-}04 &= .00003451123 \end{aligned}$$

The computer or calculator will treat these as exact numbers but we should not, when we use the results of a calculation, treat the answer as an exact number. Rather we should treat the mantissa as a decimal approximation representing an interval of numbers but multiply this interval by 10 raised to the characteristic. For example, consider the floating point numbers above. The mantissa .3451123 represents any number in the interval $[.34511225, .34511235)$ So

$$\begin{aligned} .3451123\texttt{E}09 \quad &\text{represents} \quad [345112250, 345112350) \\ .3451123\texttt{E}02 \quad &\text{represents} \quad [34.511225, 34.511235) \\ .3451123\texttt{E}00 \quad &\text{represents} \quad [.34511225, 34511235) \\ .3451123\texttt{E}{-}04 \quad &\text{represents} \quad [.000034511225, .0000345112350) \end{aligned}$$

In particular the integer 345112276 could be represented by .3451123E09 We will see, however, in the next section that when the accuracy of a floating point representation is discussed, it is the accuracy of the mantissa that is important, the characteristic should be ignored.

Finally we note that for decimal or floating point approximation to complex numbers use two real decimal (resp. floating point) approximations, one for the real part and the other for the complex part. This will represent an entire half open square of actual complex numbers. Of course, there is no ordering here.

2.2 Numerical Algorithms

Numerical algorithms use floating point approximations to real numbers to do calculations. One then gets approximate, rather than exact, solutions to the problem. Traditionally, when working with numerical algorithms, there are two main concerns: accuracy and efficiency (speed). Using modern calculation equiptment these will not be an important consideration in our working with polynomials, but we discuss these factors briefly in this section, for more details consult a Numerical Analysis textbook. Other concerns regarding numerical methods are stability/reliability and simplicity of the method. We will address these issues when discussing particular algorithms.

ACCURACY

As mentioned above it is the mantissa of a floating point number that determines its accuracy. Precisely we use the term *significant digits*.

To define the concept of significant digits we make a very brief excursion into error analysis. Let p be a real number and p^* an approximation of p, eg. p^* is a floating point number. The *absolute* error is $|p^* - p|$ whereas the *relative error* is $\frac{|p^*-p|}{|p|}$. The relative error is considered the more important measurement by numerical analysists. Thus, p^* is said to approximate p to *t significant digits* if t is the largest non-negative integer so that the relative error is less than $5 * 10^{-t}$. Note that if $p = 2.43333333333$ and $p^* = 2.433$ then the relative error is $1.37 * 10{-}4$ so p^* approximates p to 4 significant digits. On the other hand if $p = 9.23333333333$ and $p^* = 9.233$ while the absolute error is the same, the relative error is $3.61 * 10^{-5}$ so p^* approximates p to 5 significant digits. At first this might seem a bit strange but if one considers that the number 9.233 is about as accurate as the number 10.233 then 5 significant digits seems more reasonable.

For practical purposes, the number of significant digits accuracy one can get from a computer or calculator is at best 1 less than the number of digits of the mantissa in the floating point representation of the number. One way to find out how many digits a given machine has is to do the following: enter 1/3 as a division problem. Multiply by 10 and subtract 3. Repeat this last step until the leading digit is no longer 3. Your *machine precision* is 1/10 raised to the power of the number of times you can do this. A quicker way that might work is by the calculation $4 - 4/3 - 4/3 - 4/3$ (in Maple let `x := evalf(4/3)` and do `4 - x - x - x`) . Some calculators or programs make assumptions about rational numbers so you may need to calculate something like $3 * \tan(\pi/6)^2 - 1$ using an internal value for π.

Calculators may have a machine precision of between 7 and 12 digits, the machine precision of a computer program may depend on the program. Maple's machine precision is determined by the variable `Digits` which has a default value of 10 but can be

arbitrarily set by the user.

For the purposes of the exercises in this Chapter you should try for 5 significant digit accuracy, which should be attainable on any calculator or computer, with sufficient care.

One very important consideration for the computer or calculator user is the representation of the number 0. In principle the number 0 is not representable as a floating point number. Some computers have special ways of denoting exact 0, however due to round off error numbers which should be 0 are often expressed as floating point numbers with large negative exponents. If you are dealing with, say, 5 significant digit accuracy, any number given in exponential form $m\texttt{E}{-}t$ where $t > 6$ should be considered as, and treated as, 0. For example $-0.382342\texttt{E}{-}8$ or .789543E-10 are different ways of representing the number zero. In any case, any number less than your machine precision in absolute value must be consisdered to be zero.

Floating point multiplications or additions of numbers with the same sign give results with about the same relative error as the numbers started with. Exponentiation or use of math functions such as sine or cosine use many floating point additions and multiplications and so results will have more relative error.

One place where floating point arithmetic can kill you is when adding numbers of about the same magnitude but of opposite signs. For example, let $p = 243.2222222222222$ and $q = 243.2211111111111$ $p^* = 243.222$ and $q^* = 243.221$ are approximations to p and q good to 6 significant digits. $p - q = .0011111111111$ but $p^* - q^* = .001$, the absolute error is .0001111111111 but the relative error is 0.1 and thus the appoximation of $p - q$ by $p^* - q^*$ is good to only one significant digit. Thus whenever using a calculator or computer great care must be used to avoid this situation, i.e. make sure you never get small numbers as a result of addition or subtraction of large ones.

The most serious disaster with floating point is that the commutative law of addition may not hold. For example, consider $A = 10^{20}, B = -10^{20}$ and $C = 1$. Then $A + B + C = 1$ but $A + C + B = 0$.

EFFICIENCY

With computers and calculators arithmetic can be done with great speed and little human effort. Yet the study of efficiency of algorithms is more important today than any time in the past. The reason for this is that we now tackle harder problems than were previously deemed possible and we often repeat calculations hundreds or thousands of times. For example, instead of going through many cumbersome steps in a process called "separation of roots" which was formerly used to get the approximate location of real roots of a polynomial, it is now easy to simply use the computer to graph the polynomial by plotting points. But a plotting program may evaluate the polynomial between 280 or 640 times, depending on the machine. Such programs can be

aggravatingly slow for polynomials of large degree unless care is taken to use a fast algorithm for polynomial evaluation.

There is no one measure of the efficiency of an algorithm, the measure to be used depends on the type of algorithm and the personal tastes of the evaluator. One simple measure to compare algorithms which do the same problem on the same computer is the actual time of execution, which can be easily measured by a real or computer stopwatch.

Often the interest in efficiency centers on how algorithms handle increasingly larger or more complex versions of the same problem. In this case some sort of numerical measure of the complexity of the algorithm is needed. One measure is the number of lines of a standard computer language (such as BASIC) needed to implement the algorithm, with lines repeated in loops counted according to the number of times repeated. On the other hand one may wish to count the number of arithmetic operations needed. Or one may want to assign a degree of difficulty to each operation and multiply the number of each kind of operation by its difficulty.

For example, on a TI-85 such operations as addition, multiplication or an IF decision take about the same amount of time but exponentiation or evaluation of some other math function such as sine or logarithm takes about 5 times as long. For example on a calculator the exponentiation operation x^r does not distinguish whether is an integer or a decimal number (provided $x > 0$) and essentially does the exponentiation by converting to logarithms, multiplying and converting back so implementing the operation x^2 by exponentiation takes 5 times as long as by using multiplication $x * x$. It is interesting to note than in the 17th century when exponential notation was just beginning to be used that mathematicians such as Descartes and Newton always wrote xx rather than x^2. In retrospect this notion was not a bad idea and probably should be readopted today!

2.3 Maple Programming and Evaluation of Polynomials

Although the nice thing about Maple is that most operations you need to do are built in, occasionally you will want to program something new. In fact Maple is a very good programming environment for the mathmatician. We discuss Maple procedures briefly, partly because these procedures are a good way to illustrate algorithms.

Maple Implementation

Programming in Maple is structured and based on *procedures*. A procedure is essentially a set of Maple commands gathered together under one name. The syntax for defining a procedure will be something like

```
F := proc(a,b,c)
      .
      .
      .
end;
```

where the dots indicate Maple commands. The variables a, b, c are the *parameters* of the procedure. A Maple procedure may have zero or many arguments. To use the procedure defined above simply use the syntax (for example)

```
F(2,7,sqrt(3));
```

One technicality that Maple forces us to deal with is the idea of *local* and *global* variables and parameters. In the above example, a, b, c are parameters. These variables retain the values given to them in the calling statement, thus above $a = 2, b = 7$ and $c = \sqrt{3}$. Parameters can not be assigned new values within a procedure. Local variables are variables which exist only within the procedure. Thus if the procedure F above uses the local variable X then when the procedure is done running Maple no longer recognizes X as an assigned variable unless X existed beforehand. In this latter case the value of X was not changed by the procedure. If X is declared as a global variable then the value of X before the procedure was run becomes available to the procedure and the value that X has at the end of the procedure is retained by X when the procedure is done. The latest versions of Maple require the user to declare all variables used as local or global or Maple will give a warning and declare all undeclared variables as local. Note that parameters are automatically local and do not get declared.

Two important Maple statements for programing are the repetition statement and the selection statement. The repetition statement is of the form

```
for i from 2 to 10 do
     .
     .
od;
```

and the selection statement is of the form

```
if
  .
  .
elseif
  .
  .
else
  .
fi;
```

In both cases the dots indicate one or more Maple statements. Note the use of `od;` and `fi;` to end `for` and `if` statements respectively. The `else` and `elseif` are optional in the `if` statement. For more options, including a `while` statement, see the help pages for `for` and `if`.

Note that the `for` statement can also be used alone outside a procedure as well, for example if you wish to make a table of values of the polynomial `f` you could simply enter on the command line:

```
for t from 0 to 5 by 0.1 do print(t,subs(x=t, f)); od;
```

Be very careful to not forget the `od` because Maple will not give any output until it receives the `od`.

When defining a procedure or a stand-alone `for` loop use SHIFT-ENTER rather than just ENTER, until the last line of the procedure. This tells Maple to wait for the next line of the procedure and to not skip lines. You will not be able to leave the procedure defining mode until you put in a last line consisting of `end;` and hit ENTER. You can later scroll back and edit a procedure, then hit ENTER anywhere in the procedure when you are done. The easiest way to save procedures is to save the worksheet on which they are defined.

For examples of Maple procedures see the various programs in the remainder of this section.

EVALUATION OF POLYNOMIALS

Given a polynomial of degree n,

$$p(x) = a_0 + a_1 x + a_2 x^2 + \cdots a_n x^n$$

we may wish to evaluate $p(x)$ at some value $x = c$. Of course with Maple if $p(x)$ is entered as a polynomial, named p, then we just use the statement `subs(x=c,p);` But to understand the mechanics of this, suppose we are given the sequence of coefficients and did not have computer algebra programs already available, how would we evaluate the polynomial?

The most obvious method is to follow the algebra of the description of the polynomial, i.e. multiply a_1 by x, then multiply a_2 by x^2 and so on, then add a_0 and all these terms. This can be implemented as a Maple program as follows:

Maple Implementation

The most convenient way to deal with the list of coefficients is to use an array. The difference between an array and a list is that one can choose the indices of an array, which is useful here since we wish to start at 0, also individual entries of arrays can be accessed. Thus the statement

```
P:= array(0..n, [a_0, a_1,...,a_n]);
```

defines the appropriate array. The entry a_i is then obtained by `P[j]` for $j = 0, 1 \ldots n$.

The procedure to use is

```
EVP := proc(c) local j,y; global P,n;
    y := 0;
    for j from 0 to n do
y := y + P[j]*c^j;
    od;
end;
```

To run the program first enter the degree and the coefficients, then run the program for example

```
n := 4;
P := array(0..n,[-4,-4,-2,-11,1]);
EVP(11.213);
```

One interesting feature of Maple is that c could be an indeterminant. Thus executing `EVP('x')` gives the algebraic version of the polynomial!

On the whole, this program will work quite well, but from the point of view of numerical analysis it has two flaws. First of all we have speed. We note that in each step of our `for` loop we have one addition, one multiplication and one exponentiation. If we think of exponentiation as equivalent to 5 additions or multiplications, we have the equivalent of seven operations each step. For a polynomial of degree N we have $N+1$ steps so the complexity is $7*(N+1)$ operations. We will see that this is over 3 times more than necessary. Secondly there is accuracy. Suppose we wish to evaluate the polynomial

$$p(x) = -4 - 4x - 2x^2 - 11x^3 + 4x^4$$

at $c = 11.213$. The following chart shows the beginning value of y and the value of $a_j * c^j$ each time the calculation $y \leftarrow y + a_j * c^j$ is executed by a calculator with machine accuracy 10^{-7}.

j	y	$a_j c^j$	$y + a_j c^j$
0	0.0	-4.0	-4.0
1	-4.0	-44.852	-48.852
2	-48.852	-251.4628	-300.3148
3	-300.3148	-15508.09	-15808.40
4	-15808.40	15808.38	$-.02050781$
			(Answer)

We note that all the digits of the final value of y except the first two cannot be justified from the sum $-15808.40 + 15808.38$ and in fact are put there quite arbitrarily by the calculator just to bring the number up to the correct length. Thus all we can really conclude is that the answer is somewhere about $-.02$, i.e. at best one significant digit.

Experts in numerical analysis prefer a different algorithm, the *nested algorithm*. Here we write our polynomial as

$$x^4 - 11x^3 - 2x^2 - 4x - 4 = (((1 * x - 11) * x - 2) * x - 4) * x - 4$$

A Maple Procedure to evaluate the polynomial in this form might be

Maple Implementation

```
EVH := proc(c) local j,y; global n,P;
```

```
    y := 0;
    for j from n by -1 to 0 do
y := y*c +P[j];
    od;
end;
```

Which is executed by the statements

```
n := 4;
P := array(0..n,[-4,-4,-2,-11,1]);
EVH(11.213);
```

First we note that the calculation $y = y * c + a_j$ contains only two operations, neither of them exponentiations. Thus the complexity of this algorithm to evaluate a polynomial of degree N in terms of additions and multiplications is $2 * (N + 1)$, 3.5 times faster than the exponential algorithm. As for accuracy here is the chart of additions performed by this algorithm:

j	$y * c$	a_j	$y * c + a_j$
4	0.0	1.	1.0
3	11.213	$-11.$	.213
2	2.388369	$-2.$	.3883692
1	4.354782	-4	.3547820
0	3.978166	-4	$-.02183445$
			(Answer)

Note that unlike the chart for the exponential form all the numbers tend to be in a much smaller range of order of magnitude. It is true that we are constantly adding numbers of like magnitude but opposite sign, but the difference is only smaller by a factor of 10 – thus in the last column only the last digit is unwarrented. Thus we may have much more confidence in this answer - probably to 4 significant digits (the last actually computed digit is somewhat suspect always).

It is instructive to compare the chart above to long division of the polynomial $p(x)$ by $x - c$ where $c = 11.213$ For convenience we round off to 5 significant digits.

$$
\begin{array}{rl|rcrcrcrcr}
 & & x^3 & + & .213x^2 & + & .38836x & + & .35478 & & \\
\hline
x - 11.213 & | & x^4 & - & 11x^3 & - & 2x^2 & - & 4x & - & 4 \\
 & & x^4 & - & 11.213x^3 & & & & & & \\
\cline{3-5}
 & & & & .213x^3 & - & 2x^2 & & & & \\
 & & & & .213x^3 & - & 2.3883x^2 & & & & \\
\cline{5-7}
 & & & & & & .3883x^2 & - & 4x & & \\
 & & & & & & .3883x^2 & - & 4.3547x & & \\
\cline{7-9}
 & & & & & & & & -.3547x & - & 4 \\
 & & & & & & & & -.3547x & - & 3.9782 \\
\hline
 & & & & & & & & & & -.0218
\end{array}
$$

What we notice is that the first four numbers in the right hand column of the chart are exactly the coefficients of the successive remainders, and hence the coefficients of the quotient. The last entry is the last remainder, and hence, by the remainder theorem, the value of the polynomial at 11.213. Thus the process of nested evaluation is exactly the remainder theorem and the nested algorithm is nothing more nor less than what is usually called synthetic division. Further we note that if we need to know the quotient the nested algorithm provides these quotients with no extra work! We will see the usefulness of this in the next section.

While the nested algorithm seems to be more accurate in the example above, in general numerical analysts view the nested and exponential methods of evaluating polynomials as having about the same accuracy. Their preference for the nested method is based on the fact that the nested method is much faster. This can be explored in Maple as follows.

Maple Implementation

To check on the relative speed of different methods of evaluating polynomials on Maple one can use the `time()` procedure. This measures CPU time used, but only to the nearest second. Thus one may need to evaluate a procedure many times to get an accurate time count. The following procedure will help compare polynomial evaluation schemes.

```
Timeproc := proc(F,k) local st,j;
    st := time();
```

```
    for j from 1 to k do F(2.00 + .01*j): od;
    time()-st;
end;
```

To use this we replace the parameter F with the name of our evaluation procedure. The parameter k is the number of times the polynomial is to be evaluated. k=1000 is a good choice, once you have the procedure debuged. Before trying F as EVP or EVH you should figure out the overhead by trying a simple procedure such as the *arrow function*

```
F := c -> c;
```

then subtract the time Timeproc takes to evaluate this from your subsequent times. You can also compare Maple's internal evaluation procedure with EVP and EVH. Create the arrow function

```
F := c -> subs(x=c,p);
```

Here p will be a polynomial in standard algebraic Maple syntax.

One final comment on Maple is in order. Unlike calculators or standard programming languages, when Maple does a numerical calculation it estimates the accuracy and only prints out what it feels are signifigant digits. Thus if you run the examples above on Maple your results will be different from those reported above.

Exercise 2.3.1 [20 points] Explore the comparative evaluation time of EVP and EVH on your Maple system using `k = 1000`. Also compare these procedures to `subs(x=c,p);` as indicated above where p is a polynomial in exponential form and again when p is in nested form. Try this for several polynomials of various coefficients and degrees. Write up and try to explain your findings.

Exercise 2.3.2 [20 points] Let $p(x) = -13 - 7x - 6x^2 - 3x^3 - 16x^4 + 2x^5$. How accurately can you calculate $p(8.2341)$? Try different calculator and computer environments, and try both the nested and exponential method on each environment. Try the different methods (`EVP, EVH, subs(x=c,p)` on Maple and try different setting of `Digits`. Write up your findings and try to explain based on the discussion in this section.

2.4 Taylor's Series and Horner's Process

The exponential method for evaluation lacks good precision in the example of the last section primarily because for x of large absolute value x^j gets very large, often much larger in magnitude than the value of the polynomial. One way to try and avoid this problem is using the nested method, (or synthetic division) but this doesn't always work out as well as in the example above. A more certain method to deal with this problem is to use a change of variable to replace x by a number of absolute value less than 1, the successive terms in the exponential evaluation get smaller, rather than larger.

To accomplish this, pick a convenient number c near the number at which you want to evaluate the polynomial. If your original polynomial has integer coefficients, choosing c to be an integer will allow exact arithmetic. Let $p(x)$ be your polynomial, by the Remainder Theorem

$$p(x) = (x-c)g_1(x) + p(c)$$

Applying the remainder theorem to $g_1(x)$ gives

$$g_1(x) = (x-c)g_2(x) + g_1(c)$$

and substituting this into the first equation gives

$$p(x) = (x-c)^2 g_2(x) + (x-c)g_1(c) + p(c)$$

Now applying the Remainder Theorem to $g_2(x)$ and substituting

$$p(x) = (x-c)^3 g_3(x) + (x-c)^2 g_2(c) + (x-c)g_1(c) + p(c)$$

We can continue in this manner noting that the degree of the quotient $g_k(x)$ goes down 1 each step. Thus if $\deg p(x) = n$, $g_n(x)$ is a non-zero constant (in fact it is the leading coefficient of $p(x)$) and thus

$$p(x) = b_0 + b_1(x-c) + b_2(x-c)^2 + \cdots + b_n(x-c)^n \tag{2.1}$$

where $b_0 = p(c)$ and $b_k = g_k(c)$ and $g_k(x) = (x-c)g_{k+1}(x) + g_k(c)$ is the k^{th} quotient, $g_0(x) = p(x)$. This polynomial in $(x-c)$ is the *Taylor Series* of $p(x)$ about c.

This argument shows, as one also learns in calculus, that for a polynomial, the Taylor series is also a polynomial in $(x-c)$ and so convergence does not become a problem. As an historical aside, it should be noted that Taylor did not invent Taylor's series, in fact Taylor's series were discovered by the Scottish mathematican Gregory

even before Newton invented calculus. The derivation using calculus was first given by the Swiss mathematician Jean Bernoulli and Brook Taylor at about the same time.

We note that at each step we needed to calculate both the next quotient and the value of the last quotient at c. The otherwise obscure mathematician W.G. Horner is given credit for noticing that both can be accomplished at the same time by synthetic division. Thus the method of calculating the Taylor coefficients by synthetic division is known as Horner's Process. In implementing this algorithm one stores the coefficients in an array, but in fact the same array can be used repeatedly. We now describe a Maple procdure to calculate the Taylor coefficients using Horner's method and a given `array` of coefficients.

Maple Implementation

Before giving the procedure we revise our procedure `Evp` to make the array `P` a parameter so we can deal with several different arrays.

```
Evp := proc(c,P) local n,j,y;
   n := nops([indices(P)]) - 1;
   y := 0;
   for j from 0 to n do
y := y + P[j]*c^j;
   od;
end;
```

Note that the first line of the procedure gives a method for extracting the number of elements in a one-dimensional array. If an existing array variable is assigned a new name, say `Q := P`, then any change in `Q` changes `P` also. Maple's `copy` command is used to make a different copy of the array. Then here is our Horner's Process procedure:

```
Horners := proc(c,P) local n,Q,k,j;
    n := nops([indices(P)])-1;
    Q := copy(P);
    for k from 0 to n-1 do
 for j from n-1 by -1 to k do
    Q[j] := Q[j+1]*c + Q[j];
 od;
```

```
    od;
    RETURN(Q);
end;
```

An example of running this procedure is

```
Q := Horners(11,P);
entries(Q);
Evp(x-11,Q);
```

where `P` is the array giving the polynomial $p(x) = -4 - 4x - 2x^2 - 11x^3 + x^4$ of the previous section. Note that `entries(Q)` gives the coefficients of `Q` (warning: not guaranteed to be in the correct order!) whereas `Evp(x-11,Q)` almost writes $p(x)$ as a polynomial in $x - 11$. I say "almost" since Maple cannot resist the temptation to simplify the linear term.

While the above procedure is given to explain how Horner's process is implemented, Maple has a built in procedure which does this, called `taylor`. The syntax would be

```
p := -4 - 4*x - 2*x^2 - 11*x^3 + x^4;
taylor(p, x=11, 5);
```

The last argument, 5, gives the number of terms of the Taylor polynomial that you want displayed. For a polynomial of degre n this should be $n + 1$. You are cautioned that the result of the `taylor` procedure is of the `SERIES` data type and cannot be manipulated like a polynomial, even though it prints out on the screen like one. A `series` named `f` can be converted back to a polynomial by

```
convert(f,polynom);
```

The example above has the data for the polynomial $p(x)$ used as an example in the previous section. Running the procedure we obtain for $c = 11$ (the integer closest to 11.213)

$$p(x) = -290 + 1283(x - 11) + 361(x - 11)^2 + 33(x - 11)^3 + (x - 11)^4$$

Thus evaluating the polynomial $p(x)$ at $x = 11.213$ is equivalent to evaluating

$$f(x) = -290 + 1283z + 361z^2 + 33z^3 + z^4 \text{ at } z = 0.213$$

This can be done quite accurately by the exponential method, especially by calculator, noting that the first 3 terms, although relatively large can be calculated exactly. Thus the successive terms are

$$\begin{array}{rr} & -290.000000 \\ & 273.279000 \\ & 16.378209 \\ & .318899 \\ & .002058 \\ \hline \text{Answer} & -.021833 \end{array}$$

where all the digits indicated are accurate except in the sum where the last digit is probably inaccurate. This sum can be calculated by calculator doing the integer parts and fractional parts separately. Here we are sure that we have 4 significant digit accuracy.

Differentiating the Taylor's series (Equation 2.1 above) n times we see that the Taylor's coefficients are connected to the higher derivatives of $p(x)$ by the formula

$$b_k = \frac{p^{(k)}(c)}{k!}$$

Here $p^{(k)}$ denotes the kth derivative of $p(x)$, the zero-th derivative being the function itself. This formula is known from calculus but it should be noted that we have only used algebraic methods. Thus Horner's process can be used to calculate successive derivatives, a fact which we will use several times later in the chapter.

2.5 Synthetic Division

In precomputer days the nested algorithm and Horner's process was done by *synthetic division.* While I don't generally teach synthetic division to freshman algebra students, it is informative to see how this used to work and have the technique available for hand calculations.

In Section 2.3 the long division example was used to show that the nested algorithm was equivalent to using the remainder theorem. One can see that there is a lot of redundant writing and poor use of space. In particular, the "trial remainder" and the next coefficient of the quotient are the same. These numbers actually end up being written 3 times. Thus to save time and space we write these only once, as well as not repeating the coefficients of the dividend. Finally we turn our work upside down in anticipation of Horner's process. Thus, to divide $f(x) = 3x^3 + 8x^2 + 5x - 7$ by $x + 2$ we set up our work as follows:

$$\begin{array}{r|rrrr} -2 & 3 & 8 & 5 & -7 \\ & & -6 & -4 & -2 \\ \hline & 3 & 2 & 1 & -9 \end{array}$$

Note here that the -2 on the upper left is the number c where we are evaluating the polynomial, or, in other words, we are dividing $f(x)$ by $x - c$, in this case $x + 2 = x - (-2)$ so $c = -2$. Then the coefficients of $f(x)$ are listed in the first row, leading coefficient first. We then bring this leading coefficient directly down to the third row (our one needless repetition). Then we multiply this leading coefficient by c and put the result in the second row under the second coefficient, this is the -6. Now we **add**. Here we see an additional advantage to synthetic division over long division: adding rather than subtracting requires less thought and is less likely to lead to an arithmetic error. The sum, here 2, goes in the third row. This is multiplied by $c = -2$ and brought to the second row as -4, added to 5 and the sum 1 is recorded in the third row. Finally, this is multiplied by $c = -2$, added to -7 and recorded as -9. One sees that this last number is the remainder, or value $f(c)$, and the numbers in the third row are the coefficients of the quotient: $q(x) = 3x^2 + 2x + 1$. Since the work done is exactly the nested evaluation this method gives us the quotient for free!

Here is what the example in §2.3 would look like using synthetic division. Recall we are evaluating $f(x) = x^4 - 11x^3 - 2x^2 - 4x - 4$ at $c = 11.213$ to get $f(c) = -0.218$

$$\begin{array}{r|rrrrr} 11.213 & 1 & -11 & -2 & -4 & -4 \\ & & 11.213 & 2.3883 & 4.3547 & 3.9782 \\ \hline & 1 & .213 & .38836 & .35478 & -.0218 \end{array}$$

The further advantage of the nested method is that the quotient is in a good position to use it as the dividend to find the derivative or next coefficient in the Taylor series. As an example we find the Taylor expansion of the polynomial $f(x) = 2x^4 + 7x^3 - 4x^2 - 27x - 18$ around $x = 2$.

$$\begin{array}{c|ccccc}
2 & 1 & 0 & 20 & -32 & -4 \\
 & & 2 & 4 & 48 & 32 \\
\hline
2 & 1 & 2 & 24 & 16 & 28 \\
 & & 2 & 8 & 64 & \\
\hline
2 & 1 & 4 & 32 & 80 & \\
 & & 2 & 12 & & \\
\hline
2 & 1 & 6 & 44 & & \\
 & & 2 & & & \\
\hline
2 & 1 & 8 & & & \\
 & & & & & \\
\hline
 & 1 & & & &
\end{array}$$

We can now read off the Taylor expansion as

$$f(x) = 28 + 80(x-2) + 44(x-2)^2 + 8(x-2)^3 + (x-4)^4$$

noticing that the last division did not give new information and could have been eliminated.

Note that for the example above in the even rows the leading entry was 2, which was our value of c. This will always happen for a monic polynomial. We note that our final value for the coefficient of $(x-2)^3$ was the coefficient of x^3 in the original polynomial $f(x)$ (here 0) plus $n * c$ where $n = 4$ is our degree. Again this is typical of monic polynomials in general so we get the following theorem which is a higher degree analog of *completing the square*.

Theorem 2.5.1 *Let $f(x) = x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$ and $c = -\dfrac{a_{n-1}}{n}$. Then the coefficient of $(x-c)^{n-1}$ in the Taylor expansion of $f(x)$ about $x = c$ is 0.*

To see how this is related to completing the square consider $f(x) = x^2 + 8x - 5$. We would like to find the roots, that is, solve $f(x) = 0$. Using Horner's method we have

$$\begin{array}{c|ccc}
-4 & 1 & 8 & 5 \\
 & & -4 & -16 \\
\hline
-4 & 4 & 0 & -21 \\
 & & -4 & \\
\hline
 & 1 & 0 &
\end{array}$$

which says that $f(x) = (x+4)^2 - 21$. If we change variables by substituting $y = x+4$ the right hand side becomes $y^2 - 21$. Setting $y^2 - 21 = 0$ we get $y^2 = 21$ or $y = \pm\sqrt{21}$. Substituting $y = x + 4$ back gives $(x + 4) = \pm\sqrt{21}$ or $x = -4 \pm \sqrt{21}$ as our roots.

In higher degrees this trick is not sufficient to solve the equation, in general, but we will see in Chapter 4 that the standard methods start by making this change of variable to eliminate one coefficient and make the computations easier.

2.6 Newton's Upper Bound on the Modulus of Roots

In the precomputer days it was advisable to get estimates of the location of the roots of a polynomial before using a numerical method. In Uspensky's Theory of Equations he devotes approximately 50 pages out of 180 pages on polynomial equations to the topic of "separation of roots." This is no longer very necessary and we will discuss only a few such methods.

In this section we discuss a method, one of Newton's early applications of calculus, for the upper bound of the modulus of roots. We first deal with real polynomials.

Theorem 2.6.1 *Let $p(x)$ be a real polynomial of degree n and c a real number. Suppose $p^{(0)}(c) = p(c), p^{(1)}(c) = p'(c), p^{(2)}(c), \ldots p^{(n)}(c)$ are all positive. Then every real root α satisfies $\alpha < c$.*

Proof: From the standpoint of calculus one can prove by induction on n that the condition insures that the function $p(x)$ is strictly increasing for $x > c$ and thus since $p(c) > 0$, $p(x) > 0$ for $x > c$.

An alternate argument can be obtained from the results of the previous section. Since the Taylor's coefficients for the expansion about $(x-c)$ differ from the k^{th} derivatives by a factor of $k!$, the condition of the Theorem says that all of Taylor's coefficients are positive. One then easily sees that if $x > c$ then $(x - c)$ is positive and then $p(x)$ is positive.

In practice, Horner's method should be used to calculate the derivatives or Taylor's coefficients. Trying various values of c by trial and error quickly will get an upper bound on the absolute value of real roots.

For complex roots or complex polynomials we can appeal to the inequality $|z+w| \geq |z| - |w|$ which can be derived easily from the triangle inequality. Applying this, and the triangle inequality gives for $p(x) = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n$ the inequality $|p(z)| \geq |a_n||z|^n - |a_{n-1}||z|^{n-1} - \cdots - |a_1||z| - |a_0|$. Replacing $|z|$ by the real variable x we see that $p(z)$ can be 0 only when $x = |z|$ and $f(x) < 0$ for the real polynomial $f(x) = |a_n|x^n - |a_{n-}1|x^{n-1} - \cdots - |a_0|$. Thus we get

Theorem 2.6.2 *Let $p(z) = a_0+a_1z+a_2z^2+\cdots+a_nz^n$ be a real or complex polynomial of degree n and $f(x) = |a_n|x^n - |a_{n-1}|x^{n-1} - \cdots - |a_0|$. If c is a positive real number for which $f(c), f'(c), \ldots, f^{(n)}(c)$ are all positive then $|\alpha| < c$ for all complex roots α of $p(z)$.*

Exercise 2.6.1 [5 points] Without finding roots, prove that all the complex roots α of $p(x) = 2 - 3x + x^4 + 8x^7$ satisfy $|\alpha| < 1$.

Exercise 2.6.2 [10 points] Without solving, find an upper bound for the modulus of complex roots for $p(x) = 3 - 2x + 4x^2 - 5x^3 + 3x^4 - x^5$. Find a small interval of the real line containing all the real roots [*Hint: a negative root of $p(x)$ is a positive root of $p(-x)$*].