Numerical Analysis:

Calculus and Fundamentals:

Big "O" Truncation Error:

The 0^{th} Order of Approximation

Clearly, the sequences $\left\{\frac{1}{n^2}\right\}_{n=1}^{\infty}$ and $\left\{\frac{1}{n}\right\}_{n=1}^{\infty}$ are both converging to zero.In addition, it should be observed that the first sequence is converging to zero more rapidly than the second sequence.In the coming modules some special terminology and notation will be used to describe how rapidly a sequence is converging.

Definition 1. The function $f(h)$ is said to be big Oh of $g(h)$,denoted $f(h) = \mathbf{O}(g(h))$,if there exist constants $\hat{C} > 0$ and $\delta > 0$ such that

$|f(h)| \leq \hat{C}\,|g(h)|$ whenever $0 \leq |h| \leq \delta$.

The big Oh notation provides a useful way of describing the rate of growth of a function in terms of well-known elementary functions (x^n, $x^{\frac{1}{n}}$, a^x, $\log_a x$, etc.).The rate of convergence of sequences can be described in a similar manner.

Definition 2. Let $\{x_n\}_{n=1}^{\infty}$ and $\{y_n\}_{n=1}^{\infty}$ be two sequences.The sequence $\{x_n\}$ is said to be of order big Oh of $\{y_n\}$, denoted $\{x_n\} = \mathbf{O}(\{y_n\})$,if there exist $\hat{C} > 0$ andNsuch that

$|x_n| \leq \hat{C}\,|y_n|$ whenever $n \geq N$.

Often a function $f(h)$ is approximated by a function $p(h)$ and the error bound is known to be $M|h^n|$ .This leads to the following definition.

Definition 3. Assume that $f(h)$ is approximated by the function $p(h)$ and that there exist a real constant $M > 0$ and a positive integer n so that

$\frac{|f(h) - p(h)|}{|h^n|} \leq M$ for sufficiently small h.

We say thatapproximateswith order of approximationandwrite

$f(h) = p(h) + \mathbf{O}(h^n)$.

When this relation is rewritten in the form $|f(h) - p(h)| \leq M|h^n|$,we see that the notation $\mathbf{O}(h^n)$ stands in place of the error bound $M|h^n|$ .The following results show how to apply the definition to simple combinations of two functions.

Theorem (Big "O" Remainders for Series Approximations).

Assume that $f(h) = p(h) + \mathbf{O}(h^n)$ and $g(h) = q(h) + \mathbf{O}(h^m)$,and $r = \min\{m, n\}$ .Then

(i) $f(h) + g(h) = p(h) + q(h) + \mathbf{O}(h^r)$,

(ii) $f(h)\,g(h) = p(h)\,q(h) + \mathbf{O}(h^r)$,

(iii) $\dfrac{f(h)}{g(h)} = \dfrac{p(h)}{q(h)} + \mathbf{O}(h^r)$,

provided that $g(h) \neq 0$ and $q(h) \neq 0$.

It is instructive to consider $p(x)$ to be the n^{th} degree Taylor polynomial approximation of $f(x)$;then the remainder term is simply designated $\mathbf{O}(h^{n+1})$,which stands for the presence of omitted terms starting with the power h^{n+1} .The remainder term converges to zero with the same rapidity that h^{n+1} converges to zero as h approaches zero, as expressed in the relationship

$$\mathbf{O}(h^{n+1}) \approx M h^{n+1} = \frac{f^{(n+1)}(c)}{(n+1)!} h^{n+1}$$

for sufficiently smallh.Hence the notation $\mathbf{O}(h^{n+1})$ stands in place of the quantity $M h^{n+1}$, whereMis a constant or behaves like a constant.

Theorem (Taylor polynomial).

Assume that the function $f(x)$ and its derivatives $f'(x), f''(x), \ldots, f^{(n+1)}(x)$ are all continuous on $[a, b]$ .Ifboth x_0 and $x = x_0 + h$ lie in the interval $[a, b]$,and $h = x - x_0$ then

$$f(x_0 + h) = \sum_{k=0}^{n} \frac{f^{(k)}(x_0)}{k!} h^k + \mathbf{O}(h^{n+1})$$

,

is the n-th degree Taylor polynomial expansion of $f(x)$ about x_0 .The Taylor polynomial of degree nis

$$P_n(x) = \sum_{k=0}^{n} \frac{f^{(k)}(x_0)}{k!} (x - x_0)^k$$

and

$$f(x) = P_n(x) + R_n(x) = P_n(x) + \mathbf{O}(h^{n+1}).$$

The integral form of the remainder is

$$R_n(x) = \frac{1}{n!}\int_{x_0}^{x}(x-t)^n f^{(n+1)}(t)\,dt,$$

and Lagrange's formula for the remainder is

$$R_n(x) = \frac{f^{(n+1)}(c)}{(n+1)!}(x-x_0)^{n+1} = \frac{f^{(n+1)}(c)}{(n+1)!}h^{n+1} = \mathbf{O}(h^{n+1})$$

where c depends on x and lies somewhere between x_0 and x.

The following example illustrates the theorems above.The computations use the addition properties

(i) $\mathbf{O}(h^p) + \mathbf{O}(h^p) = \mathbf{O}(h^p)$,

(ii) $\mathbf{O}(h^p) + \mathbf{O}(h^q) = \mathbf{O}(h^r)$ where $r = \min\{p, q\}$,

(iii) $\mathbf{O}(h^p)\,\mathbf{O}(h^q) = \mathbf{O}(h^s)$ where $s = p + q$.

Order of Convergence of a Sequence

Numerical approximations are often arrived at by computing a sequence of approximations that get closer and closer to the answer desired. The definition of big Oh for sequences was given in definition 2, and the definition of order of convergence for a sequence is analogous to that given for functions in Definition 3.

Definition 4. Suppose that $\lim_{n\to\infty} x_n = x$ and $\{r_n\}_{n=1}^{\infty}$ is a sequence with $\lim_{n\to\infty} r_n = 0$.We say that $\{x_n\}_{n=1}^{\infty}$ converges to xwith the order of convergence $\mathbf{O}(r_n)$,if there exists a constant $K > 0$ such that

$$\frac{|x_n - x|}{|r_n|} \le K$$ for n sufficiently large.

This is indicated by writing

$$x_n = x + \mathbf{O}(r_n)$$

or

$x_n \to x$ with order of convergence $O(r_n)$.

Example. Let $x_n = \frac{Cos[n]}{n^2}$ and $Y_n = \frac{1}{n^2}$;then $\lim_{n\to\infty} x_n = 0$ with a rate of convergence $O\left(\frac{1}{n^2}\right)$.
Solution.

The Origin of Complex Numbers:

Chapter 1 Complex Numbers:

Overview:

Get ready for a treat. You're about to begin studying some of the most beautiful ideas in mathematics. They are ideas with surprises. They evolved over several centuries, yet they greatly simplify extremely difficult computations, making some as easy as sliding a hot knife through butter. They also have applications in a variety of areas, ranging from fluid flow, to electric circuits, to the mysterious quantum world. Generally, they are described as belonging to the area of mathematicsknown as complex analysis.

Section 1.1The Origin of Complex Numbers

Complex analysis can roughly be thought of as the subject that applies the theory of calculus to imaginary numbers. But what exactly are imaginary numbers? Usually, students learn about them in high school with introductory remarks from their teachers along the following lines: "We can't take the square root of a negative number. But let's pretend we can and begin by using the symbol $i = \sqrt{-1}$." Rules are then learned for doing arithmetic with these numbers. At some level the rules make sense. If $i = \sqrt{-1}$, it stands to reason that $i^2 = -1$. However, it is not uncommon for students to wonder whether they are really doing magic rather than mathematics.

If you ever felt that way, congratulate yourself! You're in the company of some of the great mathematicians from the sixteenth through the nineteenth centuries. They, too, were perplexed by the notion of roots of negative numbers. Our purpose in this section is to highlight some of the episodes in the very colorful history of how thinking about imaginarynumbers developed. We intend to show you that, contrary to popular belief, there is really nothing imaginary about "imaginary numbers." They are just as real as "real numbers."

Our story begins in 1545. In that year the Italian mathematician Girolamo Cardano published Ars Magna (The Great Art), a 40-chapter masterpiece in which he gave for the first time an algebraic solution to the general cubic equation

$$z^3 + a_2 z^2 + a_1 z + a_0 = 0 .$$

Cardano did not have at his disposal the power of today's algebraic notation, and he tended to think of cubes or squares as geometric objects rather than algebraic quantities.Essentially, however, his solution began with the substiution $z = x - \frac{1}{3} a_2$ .This move transforms $z^3 + a_2 z^2 + a_1 z + a_0 = 0$ into the cubic equation $x^3 + b x + c = 0$ without a squared term, which is called a depressed cubic and can be written as

$x^3 + b x + c = 0$.

You need not worry about the computational details, but the coefficients are $b = a_1 - \frac{1}{3} a_2^2$ and

$$c = a_0 - \frac{1}{3} a_1 a_2 + \frac{1}{27} 2 a_2^3 .$$

Exploration.

```
eqn0 = z^3 + a2 z^2 + a1 z + a0;
eqn1 = ReplaceAll[eqn0, z → x - a2 / 3];
eqn2 = ExpandAll[eqn1];
eqn3 = Collect[eqn2, x];
Print[eqn0 == 0];
Print["Substitute  ", z → x - a2 / 3];
Print["Get"];
Print[eqn1 == 0];
Print[eqn2 == 0];
Print[eqn3 == 0];
A = CoefficientList[eqn3, x];
Print[""];
Print[x^3, " + (", A[[2]], ")", x, " + (", A[[1]], ") = 0"];
```

$$z^3 + a_0 + z\, a_1 + z^2 a_2 == 0$$

Substitute $z \to x - \frac{a_2}{3}$

Get

$$a_0 + a_1 \left(x - \frac{a_2}{3}\right) + \left(x - \frac{a_2}{3}\right)^3 + \left(x - \frac{a_2}{3}\right)^2 a_2 == 0$$

$$x^3 + a_0 + x\, a_1 - \frac{a_1 a_2}{3} - \frac{x\, a_2^2}{3} + \frac{2 a_2^3}{27} == 0$$

$$x^3 + a_0 - \frac{a_1 a_2}{3} + \frac{2 a_2^3}{27} + x \left(a_1 - \frac{a_2^2}{3}\right) == 0$$

$$x^3 + \left(a_1 - \frac{a_2^2}{3}\right) x + \left(a_0 - \frac{a_1 a_2}{3} + \frac{2 a_2^3}{27}\right) = 0$$

To illustrate, begin with $z^3 + 9z^2 + 24z + 20 = 0$ and substitute $z = x - \frac{1}{3}a_2 = x - \frac{9}{3} = x - 3$ .The equation then becomes

$(x-3)^3 + 9(x-3)^2 + 24(x-3) + 20 = 0$, which simplifies to $x^3 - 3x + 2 = 0$.

Exploration.

```
eqn0 = z^3 + 9 z^2 + 24 z + 20;
eqn1 = ReplaceAll[eqn0, z → x - 9 / 3];
eqn2 = ExpandAll[eqn1];
Print[eqn0 == 0];
Print["Substitute  ", z → x - 9 / 3];
Print["Get"];
Print[eqn1 == 0];
Print[eqn2 == 0];
```

```
20 + 24 z + 9 z^2 + z^3 == 0
Substitute  z → -3 + x
Get
20 + 24 (-3 + x) + 9 (-3 + x)^2 + (-3 + x)^3 == 0
2 - 3 x + x^3 == 0
```

If Cardano could get any value of x that solved a depressed cubic, he could easily get a corresponding solution to $z^3 + a_2 z^2 + a_1 z + a_0 = 0$ from the identity $z = x - \frac{1}{3}a_2$. Happily, Cardano knew how to solve a depressed cubic. The technique had been communicated to him by Niccolo Fontana who, unfortunately, came to be known as Tartaglia(the stammerer) due to a speaking disorder. The procedure was also independently discovered some 30 years earlier by Scipione del Ferro of Bologna. Ferro and Tartaglia showed that one of the solutions to the depressed cubic equation is

$$x = \sqrt[3]{\frac{-c}{2} + \sqrt{\frac{c^2}{4} + \frac{b^3}{27}}} + \sqrt[3]{-\frac{c}{2} - \sqrt{\frac{c^2}{4} + \frac{b^3}{27}}}$$.

Although Cardano would not have reasoned in the following way, today we can take this value for x and use it to factor the depressed cubic into a linear and quadratic term. The remaining roots can then be found with the quadratic formula.

For example, to solve $z^3 + 9z^2 + 24z + 20 = 0$,use thesubstitution $z = x - 3$ to get $x^3 - 3x + 2 = 0$,which is a depressed cubic equation.Next, apply the "Ferro-Tartaglia" formula with $b = -3$ and $c = 2$ to get

$\sqrt[3]{\frac{-2}{2}+\sqrt{\frac{2^2}{4}+\frac{(-3)^3}{27}}}+\sqrt[3]{-\frac{2}{2}-\sqrt{\frac{2^2}{4}+\frac{(-3)^3}{27}}}=(-1)^{1/3}+(-1)^{1/3}=-2$.Since $x=-2$ is a root, $x+2$ must be a factor of x^3-3x+2.Dividing $x+2$ into x^3-3x+2 gives x^2-2x+1 ,which yields the remaining (duplicate) roots of $x=1$.The solutions to $z^3+9z^2+24z+20=0$ are obtained by recalling $z=x-3$, which yields the three roots $z_1=-2-3=-5$ and $z_2=z_3=1-3=-2$.

Exploration.

```
eqn0 = z^3 + 9 z^2 + 24 z + 20;
eqn1 = ReplaceAll[eqn0, z → x - 9 / 3];
eqn2 = ExpandAll[eqn1];
Print[eqn0 == 0];
Print["Substitute  ", z → x - 9 / 3];
Print["Get"];
Print[eqn1 == 0];
Print[eqn2 == 0];
```

```
20 + 24 z + 9 z^2 + z^3 == 0
Substitute  z → -3 + x
Get
20 + 24 (-3 + x) + 9 (-3 + x)^2 + (-3 + x)^3 == 0
2 - 3 x + x^3 == 0
```

Complex Functions and Linear Mappings:

Chapter 2Complex Functions:

Overview:

The last chapter developed a basic theory of complex numbers. For the next few chapters we turn our attention to functions of complex numbers. They are defined in a similar way to functions of real numbers that you studied in calculus; the only difference is that they operate on complex numbers rather than realnumbers. This chapter focuses primarily on very basic functions, their representations, and properties associated with functions such as limits and continuity. You will learn some interesting applications as well as some exciting new ideas.

2.1 Functions and Linear Mappings

A complex-valued function f of the complex variable z is a rule that assigns to each complex number z in a setD one and only one complex number w.We write $w = f(z)$ and call w the image of z under f.A simple example of a complex-valued function is given by the formula $w = f(z) = z^4$.The set D is called the domain of f, and the set of all images $\{w = f(z) : z \in D\}$ is called the range of f.When the context is obvious, we omit the phrase complex-valued, and simply refer to a functionf, or to a complex function f.

We can define the domain to be any set that makes sense for a given rule, so for $w = f(z) = z^4$,we could have the entire complex plane for the domain D, or we might artificially restrict the domain to some set such as $D = D_1(0) = \{z : |z| < 1\}$.Determining the range for a function defined by a formula is not always easy, but we will see plenty of examples later on.In some contexts functions are referred to as mappings or transformations.

In Section 1.6, we used the term domain to indicate a connected open set.When speaking about the domain ofa function, however, we mean only the set of points on which the function is defined.This distinction is worth noting, and context will make clear the use intended.

Just as z can be expressed by its real and imaginary parts, $z = x + iy$,we write $f(z) = w = v + iv$,where u and v are the real and imaginary parts of w, respectively.Doing so gives us the representation

$$w = f(z) = f(x, y) = f(x + iy) = u + iv .$$

Because u and v depend on x and y, they can be considered to be real-valued functions of the real variables x and y; that is,

$u = u(x, y)$ and $v = v(x, y)$.

Combining these ideas, we often write a complex function f in the form

$$f(z) = f(x + iy) = u(x, y) + iv(x, y) .$$

Figure 2.1 illustrates the notion of a function(mapping) using these symbols.

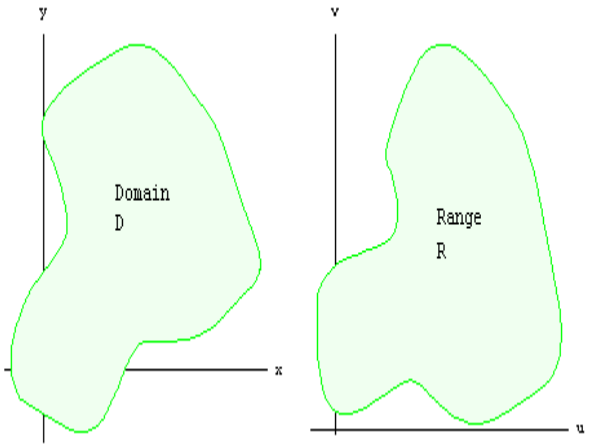


Figure 2.1 The mapping $w = f(z) = u(x, y) + i v(x, y)$.

There are two methods for defining a complex function in *Mathematica*.

We now give several examples that illustrate how to express a complex function.

Example 2.1. Write $f(z) = z^4$ in the for $f(z) = u(x, y) + i v(x, y)$.

Solution.Using the binomial formula, we obtain

$$f(z) = f(x + i y) = (x + i y)^4$$

$$= x^4 + 4 x^3 (i y) + 6 x^2 (i y)^2 + 4 x (i y)^3 + (i y)^4$$

$$= x^4 + 4 i x^3 y - 6 x^2 y^2 - 4 i x y^3 + y^4$$

$$= x^4 - 6 x^2 y^2 + y^4 + i (4 x^3 y - 4 x y^3)$$

$$= u(x, y) + i v(x, y)$$

so that $u(x, y) = x^4 - 6 x^2 y^2 + y^4$ and $v(x, y) = 4 x^3 y - 4 x y^3$.

Example 2.2. Express the function $f(z) = \overline{z}\,\mathrm{Re}[z] + z^2 + \mathrm{Im}[z]$ in the form $f(z) = u(x, y) + i v(x, y)$.

Solution.Using the elementary properties of complexnumbers, it follows that

$$f(z) = (x - i y) x + (x + i y)^2 + y = (2 x^2 - y^2 + y) + i (x y)$$

so that $u(x, y) = 2x^2 - y^2 + y$ and $v(x, y) = xy$.

Examples 2.1 and 2.2 show how to find u(x,y) andv(x,y) when a rule for computing f is given. Conversely, if u(x,y) and v(x,y) are two real-valued functions of the real variables x and y, they determine a complex-valued function $f(z) = u(x, y) + iv(x, y)$,and we can use the formulas

$$x = \frac{z + \bar{z}}{2} \text{ and } y = \frac{z - \bar{z}}{2i}$$

to find a formula for f involving the variables z and $\bar{z}$.

Example 2.3. Express $f(z) = 4x^2 + i4y^2$ by a formula involving the variables z and $\bar{z}$.

Solution.Calculation reveals that

$$f(z) = 4\left(\frac{z + \bar{z}}{2}\right)^2 + i4\left(\frac{z - \bar{z}}{2i}\right)^2$$

$$= z^2 + 2z\bar{z} + \bar{z}^2 - i(z^2 - 2z\bar{z} + \bar{z}^2)$$

$$= (1 - i)z^2 + (2 + 2i)z\bar{z} + (1 - i)\bar{z}^2$$

Using $z = re^{i\theta}$ in the expression of a complex function f may be convenient.It gives us the polar representation

$$f(z) = f\left(re^{i\theta}\right) = U(r, \theta) + iV(r, \theta),$$

where U and V are real functions of the real variables r and θ.

Remark. For a given function f, the functions u and v defined above are different from those used previously in $f(z) = f(x + iy) = u(x, y) + iv(x, y)$ which used Cartesian coordinates instead of polarcoordinates.

Example 2.4. Express $f(z) = z^2$ in both Cartesian and polar form.

Solution.For the Cartesian form, a simple calculationgives

$$f(z) = f(x+iy) = (x+iy)^2$$

$$= x^2 - y^2 + 2ixy$$

$$= u(x, y) + iv(x, y)$$

so that $u(x, y) = x^2 - y^2$ and $v(x, y) = xy$.

For the polar form, we get v

$$f(z) = f\left(re^{i\theta}\right) = \left(re^{i\theta}\right)^2 = r^2 e^{i2\theta}$$

$$= r^2(\cos 2\theta + i \sin 2\theta)$$

$$= r^2 \cos 2\theta + ir^2 \sin 2\theta$$

$$= U(r, \theta) + iV(r, \theta)$$

so that $U(r, \theta) = r^2 \cos 2\theta$ and $V(r, \theta) = r^2 \sin 2\theta$.

Remark. Once we have defined u and v for a function f in Cartesian form, we must use different symbols if we want to express f in polar form. As is clear here, the functions u and U are quite different, as are v and V.Of course, if we are working only in one context, we can use any symbols we choose.

For a given function f, the functions u and v defined here are different from those defined by equation (2-1), because equation (2-1) involves Cartesiancoordinates and equation (2-2) involves polarcoordinates.

Example 2.5. Express $f(z) = z^5 + 4z^2 - 6$ in polar form.

Solution. We obtain

$$f(z) = f\left(re^{i\theta}\right) = \left(re^{i\theta}\right)^5 + 4\left(re^{i\theta}\right)^2 - 6$$

$$= r^5 e^{i5\theta} + 4r^2 e^{i2\theta} - 6$$

$$= r^5 \cos 5\theta + 4r^2 \cos 2\theta - 6 + i(r^5 \sin 5\theta + 4r^2 \sin 2\theta)$$

$$= U(r, \theta) + iV(r, \theta)$$

so that $U(r,\theta) = r^5 \cos 5\theta + 4r^2 \cos 2\theta - 6$ and $V(r,\theta) = r^5 \sin 5\theta + 4r^2 \sin 2\theta$.

We now look at the geometric interpretation of a complex function.If D is the domain of real-valued functions u(x,y) and v(x,y), the equations

$u = u(x, y)$ and $v = v(x, y)$

describe a transformation (or mapping) from D in the xy plane into the uv plane, also called the w plane. Therefore, we can also consider the function

$w = f(z) = u(x, y) + i\,v(x, y)$

to be a transformation (or mapping) from the set D in the z plane onto the range R in the w plane.This idea was illustrated in Figure 2.1. In the following paragraphs we present some additional key ideas. They are staples for any kind of function, and you should memorize all the terms in bold.

If A is a subset of the domain D of f, the set $B = \{w = f(z) : z \in A\}$ is called the image of the set A, and f is said to map A onto B.The image of a single point is a single point, and the image of the entire domain, D, is the range, R.The mapping $w = f(z)$ is said to be from A into S if the image of A is contained inS.Mathematicians use the notation $f : A \to S$ to indicate that a function maps A into S.Figure 2.2 illustratesa function f whose domain is D and whose range isR.The shaded areas depict that the function maps A onto B.The function also maps A into R, and, of course, it maps D onto R.

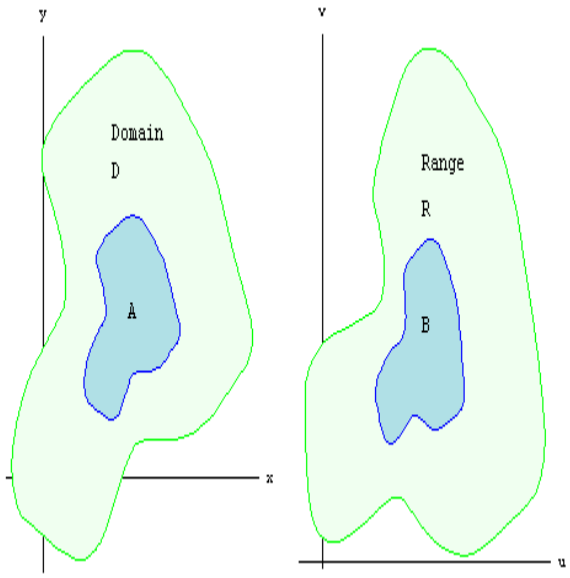


Figure 2.2 $w = f(z)$ maps A ontoB; $w = f(z)$ maps A into R.

The inverse image of a point w is the set of all points z in D such that $w = f(z)$.The inverse image ofa point may be one point, several points, or nothing at all.If the last case occurs then the point w is not in the range of f.For example, if $w = f(z) = iz$,the inverse image of the point -1 is the single point i, because $w = f(i) = i(i) = -1$,and i is the only point that maps to -1.In the case of $w = f(z) = z^2$,the inverse image of the point -1 is the set $\{i, -i\}$.You will learn in Chapter 5 that, if $w = f(z) = e^z$,the inverse image of the point 0 is the empty set---there is no complex number z such that $e^z = 0$.

The inverse image of a set of points, S, is the collection of all points in the domain that map into S.If f maps D onto R it is possible for the inverseimage of R to be function as well, but the original function must have a special property: a function f is said to be one-to-one if it maps distinct points $z_1 \neq z_2$ onto distinct points $f(z_1) \neq f(z_2)$.Many times an easy way to prove that a function f is one-to-one is to suppose $f(z_1) = f(z_2)$,and from this assumption deduce that z_1 must equal z_2.Thus, $f(z) = iz$ is one-to-one because if $f(z_1) = f(z_2)$,then $iz_1 = iz_2$.Dividing both sides of the last equation by i gives $z_1 = z_2$.Figure 2.3 illustrates the idea of a one-to-one function: distinct points get mapped to distinct points.

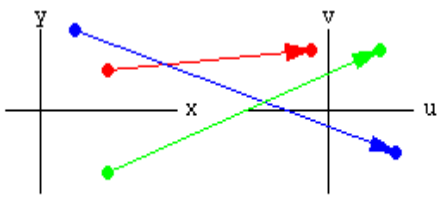


Figure 2.3A functionw = f(z)that is one-to-one.

The function $f(z) = z^2$ is not one-to-one because $-i \neq i$,but $f(i) = f(-i) = -1$.Figure 2.4 depicts this situation: at least two different points get mapped to the same point.

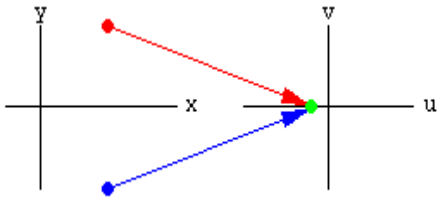


Figure 2.4A function that is not one-to-one.

In the exercises we ask you to demonstrate that one-to-one functions give rise to inverses that arefunctions.Loosely speaking, if $w = f(z)$ maps the set A one-to-one and onto the set B, then for each w in B there exists exactly one point z in AA such that $w = f(z)$.For any such value of z we can take the equation $w = f(z)$ and "solve" for z as a function ofw.Doing so produces an inverse function $z = g(w)$ where the following equations hold:

$g(f(z)) = z$ for all $z \in A$, and

$f(g(w)) = w$ for all $z \in B$

Conversely, if $w = f(z)$ and $z = g(w)$ are functions that map A into B and B into A, respectively, and the above hold, then f maps the set A one-to-one and onto the set B.

Further, if f is a one-to-one mapping from D onto T and if A is a subset of D, then f is a one-to-one mapping from A onto its image B.We can also show that, if $\xi = f(z)$ is a one-to-one mapping from A onto B and $w = g(\xi)$ is a one-to-one mapping from B onto S, then the composite mapping $w = g(f(z))$ is a one-to-one mapping from A onto S.

We usually indicate the inverse of f by the symbol f^{-1}.If the domains of f and f^{-1} are A and B respectively, then we write

$f^{-1}(f(z)) = z$ for all $z \in A$,and

$f(f^{-1}(w)) = w$ for all $w \in B$.

Also, for $z_0 \in A$ and $w_0 \in B$.

$w_0 = f(z_0)$ iff $f^{-1}(w_0) = z_0$,and

$z_0 = f^{-1}(w_0)$ iff $f(z_0) = w_0$.

The Solution of Nonlinear Equations f(x) = 0:

Fixed Point Iteration:

A fundamental principle in computer science is *iteration*.As the name suggests, a process is repeated until an answer is achieved. Iterative techniques are used to find roots ofequations, solutions of linear and nonlinear systems ofequations, and solutions of differential equations.

A rule or function $g(x)$ for computing successive terms is needed, together with a starting value p_0 .Then a sequence of values $\{p_k\}$ is obtained using the iterative rule $p_{k+1} = g(p_k)$ .The sequence has the pattern

p_0 **(starting value)**

$p_1 = g(p_0)$

$p_2 = g(p_1)$

$\vdots$

$p_k = g(p_{k-1})$

$p_{k+1} = g(p_k)$

$\vdots$

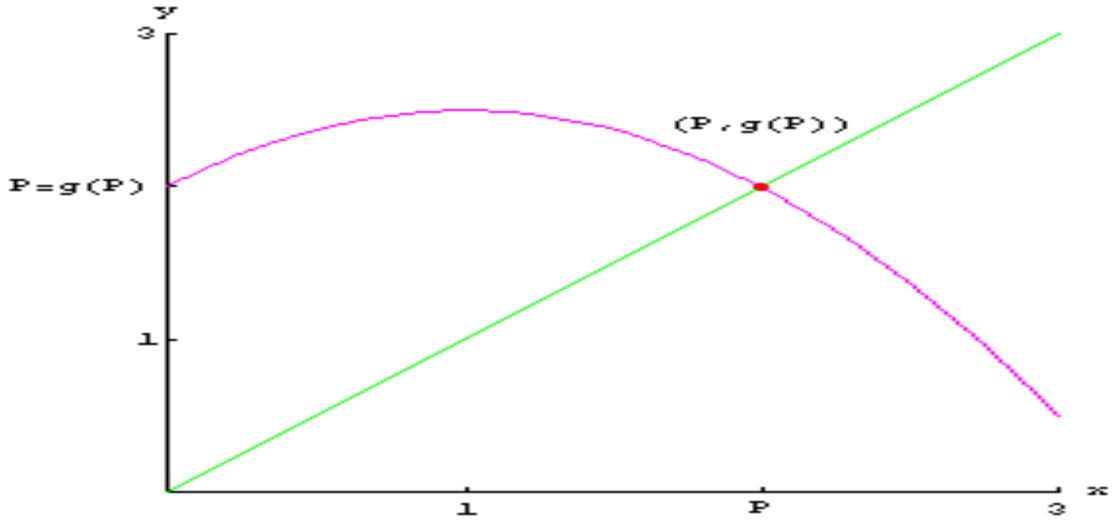
What can we learn from an unending sequence of numbers?If the numbers tend to a limit, we suspect that it is the answer.

Finding Fixed Points

Definition (FixedPoint). A fixed point of a function $g(x)$ is a number P such that $P = g(P)$.

Caution.A fixed point is not a root of the equation $0 = g(x)$,it is a solution of the equation $x = g(x)$.

Geometrically, the fixed points of a function $g(x)$ are the point(s) of intersection of the curve $y = g(x)$ and the line $y = x$.



Definition (Fixed Point Iteration). The iteration $p_n = g(p_{n-1})$ for $n = 0, 1, \cdots$ is called fixed point iteration.

Theorem (For a converging sequence). Assume that $g(x)$ is a continuous function and that $\{p_n\}_{n=0}^{\infty}$ is a sequence generated by fixed point iteration.
If $\lim_{n\to\infty} p_n = P$,then P is a fixed point of $g(x)$.

The following two theorems establish conditions for the existence of a fixed point and the convergence of the fixed-point iteration process to a fixed point.

Theorem (First Fixed Point Theorem). Assume that $g \in C[a, b]$, i. e. $g(x)$ is continuous on $[a, b]$.
Then we have the following conclusions.
(i).If the range of the mapping $y = g(x)$ satisfies $y \in [a, b]$ for all $x \in [a, b]$, then g has a fixed point in $[a, b]$.
(ii).Furthermore, suppose that $g'(x)$ is defined over (a, b) and that a positive constant $K < 1$ exists with
$|g'(x)| \le K$ for all $x \in (a, b)$,then g has a unique fixed point P in $[a, b]$.

Theorem (Second Fixed Point Theorem). Assume that the following hypothesis hold true.
(a) P is a fixed point of a function g,
(b) $g, g' \in C[a, b]$,
(c) K is a positive constant,
(d) $p_0 \in (a, b)$, and
(e) $g(x) \in [a, b]$ for all $x \in [a, b]$.
Then we have the following conclusions.

(i).If $|g'(x)| \le K < 1$ for all $x \in [a, b]$,then the iteration $p_n = g(p_{n-1})$ will converge to the unique fixed point $P \in (a, b)$ .In this case, P is said to be an attractive fixed point.
(ii).If $|g'(x)| > 1$ for all $x \in [a, b]$,then the iteration $p_n = g(p_{n-1})$ will not converge to P .
In this case, P is said to be a repelling fixed point and the iteration exhibits local divergence.

Remark 1. It is assumed that $p_0 \neq P$ in statement (ii).

Remark 2. Because g is continuous on an interval containing P , it is permissible to use the simpler criterion $|g'(P)| \le K < 1$ and $|g'(P)| > 1$ in (i) and (ii), respectively**.**

Corollary. Assume that g satisfies hypothesis (a)-(e)of the previous theorem.Bounds for the error involved when using p_n to approximate P are given by
$|P - p_n| \le K^n |P - p_0|$ **for** $n \ge 1$**,**
and
$$\left|P - p_n\right| \le \frac{K^n}{1-K} \left|p_1 - p_0\right| \text{ for } n \ge 1.$$

Graphical Interpretation of Fixed-point Iteration

Since we seek a fixed point P to $g(x)$,it is necessary that the graph of the curve $y = g(x)$ and the line $y = x$ intersect at the point (P, P) .
The following animations illustrate two types iteration: monotone and oscillating.

Algorithm (Fixed Point Iteration).To find a solution to the equation $x = g(x)$ by starting with p_0 and iterating $p_n = g(p_{n-1})$.

Mathematical Subroutine (Fixed Point Iteration).

The Bisection Method:

Background. The bisection method is one of the bracketing methods for finding roots of equations.
Implementation. Given a function f(x) and an interval which might contain a root, perform a predetermined number of iterations using the bisection method.
Limitations. Investigate the result of applying the bisection method over an interval where there is a discontinuity.Applythe bisection method for a function using an interval where there are distinct roots.Apply the bisection method over a "large" interval.

Theorem (Bisection Theorem). Assume that $f \in C[a, b]$ and that there exists a number $r \in [a, b]$ such that $f(r) = 0$.
If $f(a)$ and $f(b)$ have opposite signs, and $\{c_n\}$ represents the sequence of midpoints generated by the bisection process, then

$$\left| r - c_n \right| \leq \frac{b - a}{2^{n+1}} \text{ for } n = 0, 1, \ldots,$$

and the sequence $\{c_n\}$ converges to the zero $x = r$.

That is, $\lim_{k \to \infty} c_n = r$.

Mathematical Subroutine (Bisection Method).

```
Bisection[a0_, b0_, m_] :=
 Module[{},
  a = N[a0];
  b = N[b0];
  c = (a + b)/2;
  k = 0;
  output = {{k, a, c, b, f[c]}};
  While[k < m,
   If[Sign[f[b]] == Sign[f[c]],
    b = c, a = c;];
   c = (a + b)/2;
   k = k + 1;
   output = Append[output, {k, a, c, b, f[c]}];];
  Print[NumberForm[TableForm[output,
     TableHeadings → {None, {"k", "a_k", "c_k", "b_k", "f[c_k]"}}], 16]];
  Print["   c   = ", NumberForm[c, 16]];
  Print[" Δc   = ±", (b - a)/2];
  Print["f[c] = ", NumberForm[f[c], 16]];]
```

Example . Find all the real solutions to the cubic equation $x^3 + 4x^2 - 10 = 0$.

Solution.

Reduce the volume of printout.
After you have debugged you program and it is working properly, delete the unnecessary print statements.

Concise Program for the Bisection Method

```
Bisection[a0_, b0_, m_] :=
 Module[{a = N[a0], b = N[b0]},
  c = (a + b)/2;
  k = 0;
  While[k < m,
   If[Sign[f[b]] == Sign[f[c]],
    b = c, a = c; ];
   c = (a + b)/2;
   k = k + 1; ];
  Print["   c   = ", NumberForm[c, 16] ];
  Print[" Δc   = ±", (b - a)/2 ];
  Print["f[c] = ", NumberForm[f[c], 16] ]; ];
```

Now test the example to see if it still works. Use the last case in Example 1 given above and compare with the previous results.

```
Bisection[1, 2, 30];    c  = 1.365230013150722

f[c]  =  -4.349217874732858 × 10^-9
Δc  = ±4.656612 × 10^-10
```

Reducing the Computational Load for the Bisection Method

The following program uses fewer computations in the bisection method and is the traditional way to do it.Ca

```
Bisection[a0_, b0_, m_] :=
Module[{a = N[a0], b = N[b0]},
   c = (a + b)/2;
   Yb = f[b];
   Yc = f[c];
   k = 0;
   While[k < m,
    If[Sign[Yb] == Sign[Yc],
     b = c;
     Yb = Yc,
     a = c; ];
    c = (a + b)/2;
    Yc = f[c];
    k = k + 1; ];
 Print["   c   = ", NumberForm[c, 16] ];
 Print[" Ac    = ±", (b - a)/2 ];
 Print["f[c] = ", NumberForm[Yc, 16] ]; ]
```

n you determine how many fewer functional evaluations are used ?

Various Scenarios and Animations for the Bisection Method.

```
Bisection[a0_, b0_, m_] :=
 Module[{},
  a = N[a0];
  b = N[b0];
  c = (a + b)/2;
  k = 0;
  output = {{k, a, c, b, f[c]}};
  While[k < m,
   If[Sign[f[b]] == Sign[f[c]],
    b = c, a = c; ];
   c = (a + b)/2;
   k = k + 1;
   output = Append[output, {k, a, c, b, f[c]}]; ];
  Print[NumberForm[TableForm[output,
     TableHeadings → {None, {"k", "a_k", "c_k", "b_k", "f[c_k]"}}], 16]];
  Print["   c   = ", NumberForm[c, 16] ];
  Print[" Ac    = ±", (b - a)/2 ];
  Print["f[c] = ", NumberForm[f[c], 16] ]; ]
```

```
FixedPointIteration[x0_, max_] :=
 Module[{},
  p0 = N[x0];
  k = 0;
  Print["   p", 0, "  = ", PaddedForm[p0, {15, 15}]];
  While[k < max,
   Module[{},
    p1 = g[p0];
    k = k + 1;
    Print["   p", k, "  = ", PaddedForm[p1, {15, 15}] ];
    p0 = p1; ]; ];
  p = p0;
  Print[" "];
  Print["The function is g[x] = ", g[x]];
  Print["   p   = ", PaddedForm[p, {15, 15}]];
  Print["g[p] = ", PaddedForm[g[p], {15, 15}]]; ];
```

Example. Use fixed point iteration to find the fixed point(s) for the function $g(x) = 1 + x - \frac{x^2}{3}$.
Solution.

The Regula Falsi Method:

Background.

The Regula Falsi method is one of the bracketing methods for finding roots of equations.

Implementation. Given a function f(x) and an interval which might contain a root, perform a predetermined number of iterations using the Regula Falsi method.

Limitations. Investigate the result of applying the Regula Falsi method over an interval where there is a discontinuity.Applythe Regula Falsi method for a function using an interval where there are distinct roots.Apply the Regula Falsi method over a "large" interval.

Theorem (Regula Falsi Theorem). Assume that $f \in C[a, b]$ and that there exists a number $r \in [a, b]$ such that $f(r) = 0$.
If $f(a)$ and $f(b)$ have opposite signs, and

$$c_n = \frac{a_n f(b_n) - b_n f(a_n)}{f(b_n) - f(a_n)}$$

represents the sequence of points generated by the Regula Falsi process, then the sequence $\{c_n\}$

converges to the zero $x = r$.

That is, $\lim_{k \to \infty} c_n = r$.

Mathematica Subroutine (Regula Falsi Method).

```
RegulaFalsi[a0_, b0_, m_] :=
 Module[{},
  a = N[a0];
  b = N[b0];
  c = (a f[b] - b f[a])/(f[b] - f[a]);
  k = 0;
  output = {{k, a, c, b, f[c]}};
  While[k < m,
   If[ Sign[f[b]] == Sign[f[c]],
    b = c, a = c; ];
   c = (a f[b] - b f[a])/(f[b] - f[a]);
   k = k + 1;
   output = Append[output, {k, a, c, b, f[c]}]; ];
  Print[NumberForm[TableForm[output,
     TableHeadings → {None, {"k", "a_k", "c_k", "b_k", "f[c_k]"}}], 16] ];
  Print["  c  = ", NumberForm[c, 16] ];
  Print["f[c] = ", NumberForm[f[c], 16] ; ]
```

Example. Find all the real solutions to the cubic equation $x^3 + 4x^2 - 10 = 0$.
Solution.

Remember. The Regula Falsi method can only be used to find a real root in an interval [a,b] in which f[x] changes sign.

Reduce the volume of printout.

After you have debugged you program and it is working properly, delete the unnecessary print statements.

Concise Program for the Regula Falsi

```
RegulaFalsi[a0_, b0_, m_] :=
 Module[{},
  a = N[a0];
  b = N[b0];
  c = (a f[b] - b f[a])/(f[b] - f[a]);
  k = 0;
  While[k < m,
   If[Sign[f[b]] == Sign[f[c]],
    b = c, a = c; ];
   c = (a f[b] - b f[a])/(f[b] - f[a]);
   k = k + 1; ];
  Print["   c   = ", NumberForm[c, 16] ];
  Print["f[c] = ", NumberForm[f[c], 16] ]; ]
```

Now test the example to see if it still works. Use the last case in Example 1 given above and compare with the previous results.

```
RegulaFalsi[1, 2, 30];    c   = 1.365230013414096
 f[c]   =   -6.217248937900877 × 10^-15
```

Reducing the Computational Load for the Regula Falsi Method

The following program uses fewer computations in the Regula Falsi method and is the traditional way to do it.Can you determine how many fewer functional evaluations are used ?

Newton's Method:

If $f(x)$, $f'(x)$, and $f'(x)$ are continuous near a root p, then this extra information regarding the nature of $f(x)$ can be used to develop algorithms that will produce sequences $\{p_k\}$ that converge faster to p than either the bisection or false position method. The Newton-Raphson (or simply Newton's) method is one of the most useful and best knownalgorithms that relies on the continuity of $f'(x)$ and $f'(x)$ .The method is attributed to Sir Isaac Newton (1643-1727) andJoseph Raphson (1648-1715).

Theorem (Newton-Raphson Theorem). Assume that $f \in C^2[a, b]$ and there exists a number $p \in [a, b]$, where $f(p) = 0$.If $f'(p) \neq 0$, then there exists a $\delta > 0$ such that the sequence $\{p_k\}_{k=0}^{\infty}$ defined by the iteration

$$p_{k+1} = g(p_k) = p_k - \frac{f(p_k)}{f'(p_k)} \text{ for } k = 0, 1, \ldots$$

will converge to p for any initial approximation $p_0 \in [p - \delta, p + \delta]$.

Algorithm (Newton-Raphson Iteration). To find a root of $f(x) = 0$ given an initial approximation p_0 using the iteration

$$p_{k+1} = p_k - \frac{f(p_k)}{f'(p_k)} \text{ for } k = 0, 1, \cdots, m .$$

Mathematica Subroutine (Newton-Raphson Iteration).

```
NewtonRaphson[x0_, max_] :=
 Module[{},
  k = 0;
  p0 = N[x0];
  Print["p0 = ", PaddedForm[p0, {16, 16}], ",   f[p0] = ", NumberForm[f[p0], 16]];
  p1 = p0;
  While[k < max,
   p0 = p1;
   p1 = p0 - f[p0]/f'[p0];
   k = k + 1;
   Print["p", k, " = ", PaddedForm[p1, {16, 16}], ",
                    f[", "p", k, "] = ", NumberForm[f[p1], 16]]];

 Print["  p  = ", NumberForm[p1, 16]];
 Print[" Δp  = ±", Abs[p1 - p0]];
 Print["f[p] = ", NumberForm[f[p1], 16]]]
```

Example. Use Newton's method to find the three roots of the cubic polynomial $f[x] = 4x^3 - 15x^2 + 17x - 6$.

Determine the Newton-Raphson iteration formula $g[x] = x - \frac{f[x]}{f'[x]}$ that is used.Show details of the computations for the starting value $p_0 = 3$.

Solution.

Definition (Order of a Root) Assume thatf(x)and its derivatives $f'(x), \ldots, f^{(m)}(x)$ are defined and continuous on an interval aboutx = p.We say thatf(x) = 0has a root of ordermatx = pif and only if

$$f(p) = 0, f'(p) = 0, f''(p) = 0, \ldots, f^{(m-1)}(p) = 0, f^{(m)}(p) \neq 0.$$

A root of orderm = 1is often called a **simple root**, and ifm > 1it is called a**multiple root**.A root of orderm = 2is sometimes called a **double root**, and so on.The next result will illuminate these concepts.

Definition (Order of Convergence) Assume that p_n converges top,and set $E_n = p - p_n \text{ for } n \geq 0$ .If two positive constants $A \neq 0 \text{ and } R > 0$ exist, and

$$\lim_{n \to \infty} \frac{|p - p_{n+1}|}{|p - p_n|^R} = \lim_{n \to \infty} \frac{|E_{n+1}|}{|E_n|^R} = A,$$

then the sequence is said to converge topwith **order of convergence R**.The numberAis called the **asymptotic error constant**.The cases $R = 1, 2$ are given specialconsideration.

(i) If $R = 1$, the convergence of $\{p_k\}_{k=0}^{\infty}$ is called **linear**.

(ii) If $R = 2$, the convergence of $\{p_k\}_{k=0}^{\infty}$ is called **quadratic**.

Theorem (Convergence Rate for Newton-Raphson Iteration) Assume that Newton-Raphson iteration produces a sequence $\{p_k\}_{k=0}^{\infty}$ that converges to the rootpof the function $f(x)$.

Ifpis a simple root, then convergence is quadratic and

$$|E_{k+1}| \approx \frac{|f''(p)|}{2|f'(p)|}(|E_k|)^2$$

forksufficiently large.

Ifpis a multiple root of orderm,then convergence is linear and $|E_{k+1}| \approx \frac{m-1}{m}|E_k|$ forksufficiently large.

Reduce the volume of printout.

After you have debugged you program and it is working properly, delete the unnecessary print statements

```
Print[p0 = , PaddedForm[N[p0], {11, 11}], ,   f[p0] = , f[p0] ];
```
and
```
Print[p, k, = , PaddedForm[N[p1], {11, 11}], ,   f[p, k, ] = , f[p1] ];
```

Concise Program for the Newton-Raphson Method

```
NewtonRaphson[x0_, max_] :=
 Module[{},
  k = 0;
  p0 = N[x0];
  p1 = p0;
  While[ k < max,
   p0 = p1;
   p1 = p0 - f[p0]/f'[p0];
   k = k + 1; ];
  Print["   p   = ", NumberForm[p1, 16] ];
  Print["  Δp   = ±", Abs[p1 - p0] ];
  Print["f[p] = ", NumberForm[f[p1], 16] ]; ];
```

Now test this subroutine using the function in Example 1.

```
f[x_] = 4 x^3 - 15 x^2 + 17 x - 6;
Print["f[x] = ", f[x] ];    f[x] = -6 + 17 x - 15 x^2 + 4 x^3 NewtonRaphson[3.0, 7];    p  = 2.
 Δp  = ±2.85424×10^-10  f[p] = 0. NewtonRaphson[0.0, 8];    p  = 0.7500000000000001
 Δp  = ±3.58021×10^-11
f[p]  =  -1.1102230246251157×10^-15
NewtonRaphson[1.4, 5];    p  = 1.

f[p]  =  8.881784197001125×10^-16
Δp  = ±2.27026×10^-10
```

Error Checking in the Newton-Raphson Method

Error checking can be added to the Newton-Raphson method.Here we have added a third parameter δ to the subroutine which estimate

```
NewtonRaphson[x0_, δ_, max_] :=
 Module[{},
  k = 0;
  p0 = N[x0];
  Δp = 1;
  While[ And[ k < max, δ < Abs[Δp] ],
   Δp = f[p0] / f'[p0];
   p1 = p0 - Δp;
   k = k + 1;
   p0 = p1; ];
  Print["   p   = ", NumberForm[p1, 11] ];
  Print["  Δp   = ±", Abs[Δp] ];
  Print["f[p] = ", NumberForm[f[p1], 11] ]; ];
```

e the accuracy of the numerical solution.

The following subroutine call uses a maximum of 20 iterations, just to make sure enough iterations are performed.However, it will terminate when the difference between consecutive iterations is less than 10^{-10} .By interrogatingkafterward we can see how many iterations were actually performed.

```
f[x_] = 4 x^3 - 15 x^2 + 17 x - 6;
Print["f[x] = ", f[x] ];
```

```
NewtonRaphson[ 0.0, 10^-10, 20 ];
f[x] = -6 + 17 x - 15 x^2 + 4 x^3   p  = 0.75   Δp  = ±3.58021×10^-11  f[p] = -1.1102230246×10^-15
```

Various Scenarios

The Secant Method:

The Newton-Raphson algorithm requires two functionsevaluations per iteration, $f(p_k)$ and $f'(p_k)$ .Historically, thecalculation of a derivative could involve considerable effort.But, with modern computer algebra software packages such as*Mathematica*, this has become less of an issue.Moreover, many functions have non-elementary forms (integrals, sums, discrete solution to an I.V.P.), and it is desirable to have a method for finding a root that does not depend on the computation of aderivative. The secant method does not need a formula for thederivative and it can be coded so that only one new function evaluation is required per iteration.

The formula for the secant method is the same one that was used in the regula falsi method, except that the logical decisions regarding how to define each succeeding term are different.

Theorem (Secant Method).

Assume that $f \in C^2[a, b]$ and there exists a number $p \in [a, b]$, where $f(p) = 0$. If $f'(p) \neq 0$, then there exists a $\delta > 0$ such that the sequence $\{p_k\}_{k=0}^{\infty}$ defined by the iteration

$$p_{k+1} = g(p_{k-1}, p_k) = p_k - \frac{f(p_k)(p_k - p_{k-1})}{f(p_k) - f(p_{k-1})}$$

for $k = 0, 1, \ldots$

will converge to p for certain initial approximations $p_0, p_1 \in [p - \delta, p + \delta]$.

Algorithm (Secant Method). Find a root of $f(x) = 0$ given two initial approximations p_0 and p_1 using the iteration

$$p_{k+1} = p_k - \frac{f(p_k)(p_k - p_{k-1})}{f(p_k) - f(p_{k-1})}$$

for $k = 1, 2, \cdots, m$.

```
SecantMethod[x0_, x1_, max_] :=
 Module[{},
  k = 1;
  p0 = N[x0];
  p1 = N[x1];
  Print["p0 = ", PaddedForm[p0, {16, 16}], ",   f[p0] = ", NumberForm[f[p0], 16]];
  Print["p1 = ", PaddedForm[p1, {16, 16}], ",   f[p1] = ", NumberForm[f[p1], 16]];
  p2 = p1;
  p1 = p0;
  While[k < max,
   p0 = p1;
   p1 = p2;
   p2 = p1 - f[p1] (p1 - p0)/(f[p1] - f[p0]);
   k = k + 1;
   Print["p", k, " = ", PaddedForm[p2, {16, 16}], ",   f[", "p", k, "] = ", NumberForm[f[p2], 16]]; ];
  Print["  p  = ", NumberForm[p2, 16]];
  Print[" Ap  = ±", Abs[p2 - p1]];
  Print["f[p] = ", NumberForm[f[p2], 16]]; ]
```

Mathematica Subroutine (Secant Method).

Example. Use the secant method to find the three roots of the cubic polynomial $f[x] = 4x^3 - 16x^2 + 17x - 4$.

Determine the secant iteration formula $g[x] = x - \frac{f[x]}{f'[x]}$ that is used.

Show details of the computations for the starting value $p_0 = 3$ and $p_1 = 2.8$.

Solution.

Reduce the volume of printout.

After you have debugged you program and it is working properly, delete the unnecessary print statements

```
Print[p0 = , PaddedForm[N[p0], {11, 11}], ,    f[p0] = , f[p0] ];

Print[p1 = , PaddedForm[N[p1], {11, 11}], ,    f[p1] = , f[p1] ];
```

and

```
Print[p, k,  = , PaddedForm[N[p2], {11, 11}], ,    f[p, k, ] = , f[p2] ];
```

and

Concise Program for the Secant Method

```
SecantMethod[x0_, x1_, max_] :=
 Module[{},
  k = 1;
  p0 = N[x0];
  p1 = N[x1];
  p2 = p1;
  p1 = p0;
  While[ k < max,
   p0 = p1;
   p1 = p2;
   p2 = p1 - f[p1] (p1 - p0) / (f[p1] - f[p0]);
   k = k + 1; ];
  Print["  p  = ", NumberForm[p2, 16] ];
  Print[" Δp  = ±", Abs[p2 - p1] ];
  Print["f[p] = ", NumberForm[f[p2], 16] ]; ];
```

Now test this subroutine using the function in Example 1.

```
f[x_] = 4 x^3 - 16 x^2 + 17 x - 4;
Print["f[x] = ", f[x] ];     f[x] = -4 + 17 x - 16 x^2 + 4 x^3 SecantMethod[3.0, 2.8, 10];
   p  = 2.406803251324166  Δp  = ±0.  f[p] = 0.  SecantMethod[0.6, 0.5, 10];
   p  = 0.328538458611415  Δp  = ±0.  f[p] = 1.387778780781446×10^-16
SecantMethod[1.0, 1.1, 8];    p  = 1.26465829006442  Δp  = ±0.  f[p] = 0.
```

Error Checking in the Secant Method

Error checking can be added to the secant method.Here we have added a third parameter δ to the subroutine which estimate the accuracy of the numerical solution.

```
SecantMethod[x0_, x1_, δ_, max_] :=
 Module[{},
  k = 0;
  p0 = N[x0];
  p1 = N[x1];
  Δp = 1;
  While[ And[ k < max, δ < Abs[Δp] ],
   Δp = f[p1] (p1 - p0) / (f[p1] - f[p0]);
   p2 = p1 - Δp;
   k = k + 1;
   p0 = p1;
   p1 = p2; ];
  Print["  p  = ", NumberForm[p2, 11] ];
  Print[" Δp  = ±", Abs[Δp] ];
  Print["f[p] = ", NumberForm[f[p2], 11] ]; ];
```

The following subroutine call uses a maximum of 20 iterations, just to make sure enough iterations are performed.
However, it will terminate when the difference between consecutive iterations is less than 10^{-10}.
By interrogatingkafterward we can see how many iterations were actually performed.

```
f[x_] = 4 x^3 - 16 x^2 + 17 x - 4;
Print["f[x] = ", f[x] ];     f[x] = -4 + 17 x - 16 x^2 + 4 x^3
SecantMethod[3.0, 2.8, 10^-10, 20];
Print["The number of iterations used was ", k];    p  = 2.4068032513
  Δp  = ±6.43586×10^-14  f[p] = 0. The number of iterations used was 8
SecantMethod[0.6, 0.5, 10^-10, 20];
Print["The number of iterations used was ", k];    p  = 0.32853845861
  Δp  = ±1.11885×10^-13  f[p] = 1.3877787808×10^-16  The number of iterations used was 8
SecantMethod[1.0, 1.1, 10^-10, 20];
Print["The number of iterations used was ", k];    p  = 1.2646582901
  Δp  = ±6.2303×10^-15  f[p] = 0.  The number of iterations used was 6
```

Various Scenarios and Animations for the Secant Method.

```
SecantMethod[x0_, x1_, max_] :=
 Module[{},
  k = 1;
  p0 = N[x0];
  p1 = N[x1];
  Print["f[x] = ", f[x]];
  Print["p_{k+1} = g[p_k, p_{k-1}] = p_k - (f[p_k] (p_k - p_{k-1}))/(f[p_k] - f[p_{k-1}])"];
  Print["p_0 = ", PaddedForm[p0, {16, 16}], ",    f[p_0] = ", NumberForm[f[p0], 16]];
  Print["p_1 = ", PaddedForm[p1, {16, 16}], ",    f[p_1] = ", NumberForm[f[p1], 16]];
  p2 = p1;
  p1 = p0;
  While[k < max,
   p0 = p1;
   p1 = p2;
   p2 = p1 - (f[p1] (p1 - p0))/(f[p1] - f[p0]);
   k = k + 1;
   Print["p", k, " = ", PaddedForm[p2, {16, 16}], ",    f[", "p", k, "] = ", NumberForm[f[p2],
  Print[""];
  Print["f[x] = ", f[x]];
  Print["  p   = ", NumberForm[p2, 16]];
  Print[" Δp   = ±", Abs[p2 - p1]];
  Print["f[p] = ", NumberForm[f[p2], 16]]; ]
```

Muller's Method:

Background

Muller's method is a generalization of the secant method, in the sense that it does not require thederivative of the function. It is an iterative method that requires three starting points $(p_0, f(p_0))$, $(p_1, f(p_1))$, and $(p_2, f(p_2))$.A parabola is constructed that passes through the three points; then the quadratic formula is used to find a root of the quadratic for the nextapproximation.It has been proved that near a simple root Muller's method converges faster than the secant method and almost as fast as Newton's method.The method can be used to find real or complex zeros of a function and can be programmed to use complexarithmetic.

Mathematica Subroutine (Newton-Raphson Iteration).

```
NewtonRaphson[x0_, max_] :=
 Module[{},
  k = 0;
  p0 = N[x0];
  Print["p0 = ", PaddedForm[p0, {16, 16}], ",   f[p0] = ", NumberForm[f[p0], 16]];
  p1 = p0;
  While[k < max,
   p0 = p1;
   p1 = p0 - f[p0]/f'[p0];
   k = k + 1;
   Print["p", k, " = ", PaddedForm[p1, {16, 16}], ",   f[", "p", k, "] = ", NumberForm[f[p1], 16]]; ];
  Print["  p   = ", NumberForm[p1, 16]];
  Print[" Δp   = ±", Abs[p1 - p0]];
  Print["f[p] = ", NumberForm[f[p1], 16]]; ]
```

Mathematica Subroutine (Muller's Method).

Example. Use Newton's method and Muller's method to find numerical approximations to the multiple root $p = 1$ of the function $f[x] = x^3 - 3x + 2$.
Show details of the computations for the starting value $p_0 = 1$ .Compare the number of iterations for the two methods.

Solution.

```
NewtonRaphson[x0_, max_] :=
 Module[{},
  k = 0;
  p0 = N[x0];
  Print["f[x] = ", f[x]];
  g[x_] = x - f[x]/f'[x];
  Print["g[x] = x - f[x]/f'[x]"];
  Print["g[x] = ", g[x]];
  Print["g[x] = ", Simplify[g[x]]];
  Print["  p0 = ", PaddedForm[p0, {16, 16}], ",f[p0] = ",
                                  NumberForm[f[p0], 16]];
  p1 = p0;
  While[k < max,
   p0 = p1;
   p1 = p0 - f[p0]/f'[p0];
   k = k + 1;
   Print["  p", k, " = ", PaddedForm[p1, {16, 16}], ",
              f[", "p", k, "] = ", NumberForm[f[p1], 16]]; ]
  Print[""];
  Print["f[x] = ", f[x]];
  Print["  p  = ", NumberForm[p1, 16]];
  Print[" Ap  = ±", Abs[p1 - p0]];
  Print["f[p] = ", NumberForm[f[p1], 16]]; ]
```

```
RegulaFalsi[a0_, b0_, m_] :=
Module[{},
   a = N[a0];
   b = N[b0];
   Ya = f[a];
   Yb = f[b];
   c = (a Yb - b Ya)/(Yb - Ya);
   Yc = f[c];
   k = 0;
   While[k < m,
    If[ Sign[Yb] == Sign[Yc] ,
     b = c;
     Yb = Yc,
     a = c;
     Ya = Yc; ];
    c = (a Yb - b Ya)/(Yb - Ya);
    Yc = f[c];
    k = k + 1; ];
 Print["   c   = ", NumberForm[c, 11] ];
 Print["f[c] = ", NumberForm[Yc, 11] ]; ]
```

Various Scenarios and Animations for Regula Falsi Method.

```
RegulaFalsi[a0_, b0_, m_] :=
 Module[{},
  a = N[a0];
  b = N[b0];
  c = (a f[b] - b f[a])/(f[b] - f[a]);
  k = 0;
  output = {{k, a, c, b, f[c]}};
  While[k < m,
   If[ Sign[f[b]] == Sign[f[c]],
    b = c, a = c; ];
   c = (a f[b] - b f[a])/(f[b] - f[a]);
   k = k + 1;
   output = Append[output, {k, a, c, b, f[c]}]; ];
  Print[NumberForm[TableForm[output,
     TableHeadings → {None, {"k", "a_k", "c_k", "b_k", "f[c_k]"}}], 16] ];
  Print["   c   = ", NumberForm[c, 16] ];
  Print["f[c] = ", NumberForm[f[c], 16] ]; ]
```

Halley's Method:

Background

Definition (Order of a Root) Assume thatf(x)and its derivatives $f'(x), \ldots, f^{(m)}(x)$ are defined and continuous on an interval about $x = p$.We say that $f(x) = 0$has a root of ordermat $x = p$if and only if

$$f(p) = 0, f'(p) = 0, f''(p) = 0, \ldots, f^{(m-1)}(p) = 0, f^{(m)}(p) \neq 0.$$

A root of order $m = 1$is often called a simple root, and if $m > 1$it is called a multiple root.A root of order $m = 2$.is sometimes called adouble root, and so on.The next result will illuminate theseconcepts.

Definition (Order of Convergence) Assume that p_n converges top,and set $E_n = p - p_n$ for $n \geq 0$ .If two positive constants $A \neq 0$ and $R > 0$ exist, and

$$\lim_{n \to \infty} \frac{|p - p_{n+1}|}{|p - p_n|^R} = \lim_{n \to \infty} \frac{|E_{n+1}|}{|E_n|^R} = A,$$

then the sequence is said to converge topwith order of convergenceR.The numberAis called the asymptotic error constant.The cases $R = 1, 2, 3$ are given specialconsideration.

(i) If $R = 1$, the convergence of $\{p_k\}_{k=0}^{\infty}$ is called linear.

(ii) If $R = 2$, the convergence of $\{p_k\}_{k=0}^{\infty}$ is called quadratic.

(ii) If $R = 3$, the convergence of $\{p_k\}_{k=0}^{\infty}$ is called cubic.

Theorem (Newton-Raphson Iteration).

Assume that $f \in C^2[a, b]$ and there exists a number $p \in [a, b]$, where $f(p) = 0$.If $f'(p) \neq 0$, then there exists a $\delta > 0$ such that the sequence $\{p_k\}_{k=0}^{\infty}$ defined by the iteration

$$p_{k+1} = g(p_k) = p_k - \frac{f(p_k)}{f'(p_k)} \text{ for } k = 0, 1, \ldots$$

will converge to p for any initial approximation $p_0 \in [p - \delta, p + \delta]$.

Furthermore, if p is a simple root, then $\{p_{k+1}\}$ will have order ofconvergence $R = 2$,i.e.

$$\lim_{n \to \infty} \frac{|p - p_{n+1}|}{|p - p_n|^2} = \lim_{n \to \infty} \frac{|E_{n+1}|}{|E_n|^2} = A.$$

Theorem (Convergence Rate for Newton-Raphson Iteration)

Assume that Newton-Raphson iteration produces a sequence $\{p_k\}_{k=0}^{\infty}$ that converges to the rootpof the function $f(x)$.

Ifpis a simple root, then convergence is quadratic and $|E_{k+1}| \approx \frac{|f''(p)|}{2|f'(p)|}(|E_k|)^2$ forksufficiently large.

Ifpis a multiple root of orderm,then convergence is linear and $|E_{k+1}| \approx \frac{m-1}{m}|E_k|$ forksufficiently large.

Halley's Method

The Newton-Raphson iteration function is

(1) $g(x) = x - \frac{f(x)}{f'(x)}$.

It is possible to speed up convergence significantly when the root is simple.A popular method is attributed to Edmond Halley (1656-1742) and uses the iteration function:

(2) $g(x) = x - \frac{f(x)}{f'(x)}\left[1 - \frac{f(x)\,f''(x)}{2(f'(x))^2}\right]^{-1}$,

The term in brackets shows where Newton-Raphson iteration function is changed.

Theorem (Halley's Iteration). Assume that $f \in C^3[a, b]$ and there exists a number $p \in [a, b]$, where $f(p) = 0$.If $f'(p) \neq 0$, then there exists a $\delta > 0$ such that the sequence $\{p_k\}_{k=0}^{\infty}$ defined by the iteration

$$p_{k+1} = p_k - \frac{f[p_k]}{f'[p_k]}\left(1 - \frac{f[p_k]\,f''[p_k]}{2(f'[p_k])^2}\right)^{-1} \text{ for } k = 0, 1, \ldots$$

will converge to p for any initial approximation $p_0 \in [p - \delta, p + \delta]$.

Furthermore, if p is a simple root, then $\{p_{k+1}\}$ will have order ofconvergence $R = 3$,i.e.

$$\lim_{n\to\infty} \frac{|p - p_{n+1}|}{|p - p_n|^3} = \lim_{n\to\infty} \frac{|E_{n+1}|}{|E_n|^3} = A.$$

Square Roots

The function $f(x) = x^2 - a$ where $a > 0$ can be used with (1) and (2) to produce iteration formulas for finding $\sqrt{a}$ .If it is used in (1), the result is the familiar Newton-Raphson formula for finding square roots:

(3) $g(x) = x - \frac{x^2 - a}{2x}$.

When it is used in (2) the resulting Halley formula is:

$$g(x) = x - \left(\frac{x^2 - a}{2x}\right)\left(1 - \frac{(x^2 - a)\,2}{2\,(2x)^2}\right)^{-1}$$

(4)or

$$g(x) = \frac{x^3 + 3ax}{3x^2 + a}$$

This latter formula is a third-order method for computing $\sqrt{a}$ .Because of the rapid convergence of the sequences generated by (3) and (4), the iteration usually converges to machine accuracy in a few iterations.Multiple precision arithmetic is needed to demonstrate the distinction between second and third orderconvergence.The software *Mathematica* has extended precisionarithmetic which is useful for exploring these ideas.

Example. Consider the function $f(x) = x^2 - 2$, which has a root at $x = \sqrt{2}$.
(a). Use the Newton-Raphson formula to find the root.Use the starting value $p_0 = 2$
(b). Use Halley's formula to find the root.Use the starting value $p_0 = 2$

Solution.

Solution (a).
Solution (b).

Horner's Method:

Evaluation of a Polynomial

Let the polynomial $P(x)$ of degree n have coefficients $\{a_i\}_{i=0}^{n}$.Then $P(x)$ has the familiar form

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_k x^k + \ldots + a_3 x^3 + a_2 x^2 + a_1 x + a_0$$

Horner's method (or synthetic division) is a technique for evaluating polynomials.It can be thought of as nestedmultiplication.For example, the fifth-degree polynomial

$$P_5(x) = a_5 x^5 + a_4 x^4 + a_3 x^3 + a_2 x^2 + a_1 x + a_0$$

can be written in the "nested multiplication" form

$$P_5(x) = ((((a_5 x + a_4) x + a_3) x + a_2) x + a_1) x + a_0 .$$

Theorem (Horner's Method for Polynomial Evaluation) Assume that

(1) $P(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_k x^k + \ldots + a_3 x^3 + a_2 x^2 + a_1 x + a_0$

and $x = z$ is a number for which $P(z)$ is to be evaluated.Then $P(z)$ can be computed recursively as follows.

(2)Set $b_n = a_n$,
and
$b_k = a_k + z\, b_{k+1}$ for $k = n-1, n-2, \ldots, 2, 1, 0$.

Then $P(z) = b_0$.

Moreover, the coefficients $\{b_i\}_{i=0}^{n}$ can be used to construct $Q_0(x)$ and R_0

(3) $Q_0(x) = b_n x^{n-1} + b_{n-1} x^{n-2} + \ldots + b_3 x^2 + b_2 x + b_1$
and
(4) $P(x) = (x - z)\, Q_0(x) + R_0$,

where $Q_0(x)$ is the quotient polynomial of degreen-1and $R_0 = b_0 = P(z)$ is the remainder.

Example. Use synthetic division (Horner's method) to find $P(3)$ for the polynomial
$P(x) = x^5 - 6x^4 + 8x^3 + 8x^2 + 4x - 40$.

Solution.

Heuristics

In the days when "hand computations" were necessary, the Horner tableau (or table) was used.The coefficients $\{a_k\}_{k=0}^{n}$ of the polynomial are entered on the first row in descending order, the second row is reserved for the intermediate computation step($+z\,b_{k+1}$)and the bottom row contains the coefficients b_n and $\{b_k = a_k + z\,a_{k+1}\}_{k=0}^{n-1}$.

Input	a_n	a_{n-1}	a_{n-2}	$\cdots$	a_k	$\cdots$	a_2	a_1	a_0
$x = z$	$\downarrow$	$+z\,b_n$	$+z\,b_{n-1}$	$\cdots$	$+z\,b_{k+1}$	$\cdots$	$+z\,b_3$	$+z\,b_2$	$+z\,b_1$
	$\nearrow$	$\nearrow$	$\nearrow$		$\nearrow$		$\nearrow$	$\nearrow$	$b_0 = P(z)$
	b_n	b_{n-1}	b_{n-2}		b_k	$\cdots$	b_2	b_1	Output

Lemma (Horner's Method for Derivatives) Assume that

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_k x^k + \ldots + a_3 x^3 + a_2 x^2 + a_1 x + a_0$$

and $x = z$ is a number for which $P(z)$ and $P'(z)$ are to be evaluated.We have already seen that $P(z) = b_0$ can be computed recursively as follows.

$b_n = a_n$,and

$b_k = a_k + z\,b_{k+1}$ for $k = n-1, n-2, \ldots, 2, 1, 0$.

The quotient polynomial $Q_0(x) = b_n x^{n-1} + b_{n-1} x^{n-2} + \ldots + b_3 x^2 + b_2 x + b_1$ andremainder $R_0 = b_0 = P(z)$ form the relation

$$P(x) = (x - z)\,Q_0(x) + R_0 .$$

We can compute $P'(z) = c_1$ can be computed recursively asfollows.

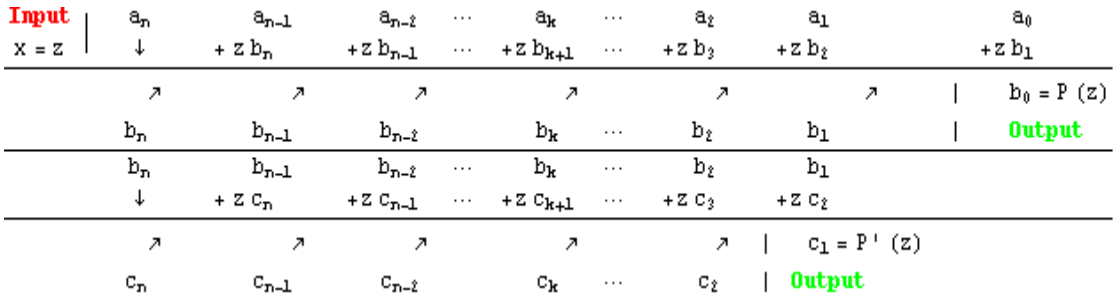
(i) $c_n = b_n$,and

$c_k = b_k + z\,c_{k+1}$ for $k = n-1, n-2, \ldots, 2, 1$.

The quotient polynomial $Q_1(x) = c_n x^{n-2} + c_{n-1} x^{n-3} + \ldots + c_4 x^2 + c_3 x + c_2$ and remainder $R_1 = c_1 = P'(z)$ form the relation

(ii) $Q_0(x)(x) = (x - z)\,Q_1(x) + R_1$.

The Horner tableau (or table) was used for computing the coefficients is given below.



Using vector coefficients

As mentioned above, it is efficient to store the coefficients $\{a_{[[k]]}\}_{k=1}^{n+1}$ of a polynomial $P(x)$ of degree n in the vector $a = \{a_{[[1]]}, a_{[[2]]}, \ldots, a_{[[n]]}, a_{[[n+1]]}\}$. Notice that this is a shift of the index for $a_{[[k]]}$ and the polynomial $P(x)$ is written in the form

$$P(x) = \sum_{k=0}^{n} a_{[[k+1]]}\, x^k .$$

Given the value $x = z$, the recursive formulas for computing the coefficients $\{b_{[[k]]}\}_{k=1}^{n+1}$ and $\{c_{[[k]]}\}_{k=2}^{n+1}$ of $Q_0(x)$ and $Q_1(x)$, have the new form

$$b_{[[n+1]]} = a_{[[n+1]]}$$
$$b_{[[k]]} = a_{[[k]]} + z\, b_{[[k+1]]} \text{ for } k = n, n-1, \ldots, 3, 2, 1 .$$

$$c_{[[n+1]]} = b_{[[n+1]]}$$
$$c_{[[k]]} = b_{[[k]]} + z\, c_{[[k+1]]} \text{ for } k = n, n-1, \ldots, 3, 2 .$$

$$P(z) = b_{[[1]]}$$

Then $P'(z) = c_{[[2]]}$

Newton-Horner method

Assume that $P(x)$ is a polynomial of degree $n \geq 2$ and there exists a number $r \in [a, b]$, where $p(r) = 0$.If $p'(r) \neq 0$, then there exists a $\delta > 0$ such that the sequence $\{r_k\}_{k=0}^{\infty}$ defined by the Newton-Raphson iteration formula

$$r_{k+1} = r_k - \frac{f(r_k)}{f'(r_k)} \text{ for } k = 0, 1, \ldots$$

will converge torfor any initial approximation $r_0 \in [r - \delta, r + \delta]$.The recursive formulas in the Lemma can be adapted to compute $P(r_k) = b_{k,0} = b_{[1]}$ and $P'(r_k) = c_{k,1} = c_{[2]}$ and the resulting Newton-Horner iteration formula looks like

$$r_{k+1} = r_k - \frac{b_{[1]}}{c_{[2]}} \text{ for } k = 0, 1, \ldots$$

Algorithm (Newton-Horner Iteration). To find a root of $f(x) = 0$ given an initial approximation r_0 using the iteration

$$r_{k+1} = r_k - \frac{b_{[1]}}{c_{[2]}} \text{ for } k = 0, 1, \cdots, \max .$$

Mathematica Subroutine (Newton-Horner Iteration).

```
NewtonHorner[x0_, max_] :=
 Module[{j = 0, k, n = Length[a] - 1, r0 = N[x0]},
  c = b = Table[0, {i, n + 1}];
  Print["r0 = ", PaddedForm[r0, {16, 16}], ",   P[r0] = ", NumberForm[P[r0], 16] ];
  r1 = r0;
  While[ j < max,
   r0 = r1;
   b[[n+1]] = a[[n+1]];
   For[ k = n, 1 ≤ k, k--, b[[k]] = a[[k]] + r0 b[[k+1]]; ];
   c[[n+1]] = b[[n+1]];
   For[ k = n, 2 ≤ k, k--, c[[k]] = b[[k]] + r0 c[[k+1]]; ];
   r1 = r0 - b[[1]] / c[[2]];
   j = j + 1;
   Print["r", j, " = ", PaddedForm[r1, {16, 16}], ",   P[", "r", j, "] = ", NumberForm[P[r1], 16] ]; ];
  Print["  r  = ", NumberForm[r1, 16] ];
  Print[" Δr  = ±", Abs[r1 - r0] ];
  Print["P[r] = ", NumberForm[P[r1], 16] ]; ]
```

Mathematica Subroutine (Newton-Raphson Iteration).

```
NewtonRaphson[x0_, max_] :=
 Module[{k = 0, r0 = N[x0]},
  Print["r0 = ", PaddedForm[r0, {16, 16}], ",   P[r0] = ", NumberForm[P[r0], 16] ];
  r1 = r0;
  While[ k < max,
   r0 = r1;
   r1 = r0 - P[r0]/P'[r0];
   k = k + 1;
   Print["r", k, " = ", PaddedForm[r1, {16, 16}], ",   f[", "r", k, "] = ", NumberForm[P[r1], 16] ]; ];
  Print["  r  = ", NumberForm[r1, 16] ];
  Print[" Δr  = ±", Abs[r1 - r0] ];
  Print["P[r] = ", NumberForm[P[r1], 16] ]; ]
```

Lemma (Horner's Method for Higher Derivatives) Assume that the coefficients $\{a_{[[1,k]]}\}_{k=1}^{n+1}$ of a polynomial $P[x]$ of degree n are stored in the first row of the matrix $[a_{i,j}]_{n+2\times n+1}$.Then the polynomial $P(x)$ can written in the form

$$P[x] = \sum_{k=0}^{n} a_{[[1,k+1]]}\, x^k .$$

Given the value $x = z$,the subroutine for computing all the derivatives $\{P^{(i)}[z]\}_{i=0}^{n}$ is

```
For[ i = 2, i ≤ n + 2, i++,
 a[[i,n+1]] = a[[i-1,n+1]];
 For[ k = n, i - 1 ≤ k, k--,
  a[[i,k]] = a[[i-1,k]] + z a[[i,k+1]]; ] ]
```

and

$P^{(i)}[z] == i!\, a_{[[i+2,i+1]]}$ for $i = 0, 1, \ldots, n$.

Polynomial Deflation

Given the polynomial $P(x) = x^5 - 6x^4 + 8x^3 + 8x^2 + 4x - 40$ in example 5,the iteration

$$r_0 = 3.0000000000000000$$

$$r_1 = r_0 - \frac{P[r_0]}{P'[r_0]} = 2.3200000000000000$$

$$r_2 = r_1 - \frac{P[r_1]}{P'[r_1]} = 1.9560094736807230$$

$$r_3 = r_2 - \frac{P[r_2]}{P'[r_2]} = 1.9992543389453250$$

$$r_4 = r_3 - \frac{P[r_3]}{P'[r_3]} = 1.9999997777286130$$

$$r_5 = r_4 - \frac{P[r_4]}{P'[r_4]} = 1.9999999999999800$$

$$r_6 = r_5 - \frac{P[r_5]}{P'[r_5]} = 2.0000000000000000$$

will converge to the root $r = 2$ of $P(x)$
.The *Mathematica*command NewtonHorner[3.0,6] produces the above sequence, then the quotient polynomial $Q(x)$ is constructed with the command $Q[x\_] = \sum_{k=0}^{n-1} b_{[[k+2]]}\, x^k$.

```
a = {-40, 4, 8, 8, -6, 1};
n = Length[a] - 1;
P[x_] = Sum[a[[k+1]] x^k, {k, 0, n}];
Print["P[x] = ", P[x]];
NewtonHorner[3.0, 6];
Q[x_] = Sum[b[[k+2]] x^k, {k, 0, n-1}];
Print["Q[x] = ", Q[x]];
```

$$P[x] = -40 + 4x + 8x^2 + 8x^3 - 6x^4 + x^5$$

$$r_0 = 3.0000000000000000, \quad P[r_0] = 17.$$

$$r_1 = 2.3200000000000000, \quad P[r_1] = 5.62609848320001$$

$$r_2 = 1.9560094736807230, \quad P[r_2] = -0.895277042894989$$

$$r_3 = 1.9992543389453250, \quad P[r_3] = -0.0149176691755244$$

$$r_4 = 1.9999997777286130, \quad P[r_4] = -4.445428135824159 \times 10^{-6}$$

$$r_5 = 1.9999999999999800, \quad P[r_5] = -3.943512183468556 \times 10^{-13}$$

$$r_6 = 2.0000000000000000, \quad P[r_6] = 0.$$

$$r = 2.$$

$$\Delta r = \pm 1.9762 \times 10^{-14}$$

$$P[r] = 0.$$

$$Q[x] = 20. + 8.x + 3.9968 \times 10^{-14} x^2 - 4.x^3 + x^4$$

The root stored in the computer is located in the variabler1.

```
Print["r1 = ", r1];  r1 = 2.
```

The coefficients ofQ[x]printed above have been rounded off.Actually there is a little bit of round off error in the coefficients forming $Q(x)$,we will have to dig them out to look at them.

```
NumberForm[ CoefficientList[Q[x], x], 17]
{20., 8.00000000000008, 3.996802888650564×10^-14, -4.00000000000002, 1}
Q[x] == x^4 - 4.00000000000002 x^3 + 3.996802888650564×10^-14 x^2 + 8.00000000000008 x + 20.0 True
```

Now we have a computer approximation for the factorization $P[x] = (x-2)\,Q[x]$.

```
P[x] == (x - 2) (x^4 - 4.00000000000002 x^3 + 3.996802888650564×10^-14 x^2 + 8.00000000000008 x + 20.0);
Print[ExpandAll[ P[x] == (x - 2) Q[x] ]]
-40 + 4 x + 8 x^2 + 8 x^3 - 6 x^4 + x^5 == -40. + 4. x + 8. x^2 + 8. x^3 - 6. x^4 + x^5
```

We should carry out one more step in the iteration using the commandNewtonHorner[3.0,7]and get a more accuratecalculation for the coefficients of $Q(x)$ .When this is done the result will be:

$$Q[x] = x^4 - 4.0\,x^3 + 0.0\,x^2 + 8.0\,x + 20.0$$

$$P[x] = (x-2)\,(x^4 - 4.0\,x^3 + 0.0\,x^2 + 8.0\,x + 20.0)$$

If the other roots of $P(x)$ are to be found, then they must be the roots of the quotient polynomial $Q(x)$ .The polynomial $Q(x)$ is referred to as the deflated polynomial, because its degree is one less than the degree of $P(x)$ .For this example it is possible to factor $Q(x)$ as the product of two quadratic polynomials $Q(x) = (x^2 - 6x + 10)\,(x^2 + 2x + 2)$ .Therefore, $P(x)$ has the factorization

$$P(x) = (x-2)\,(x^2 - 6x + 10)\,(x^2 + 2x + 2)\,,$$

and the five roots of $P(x)$ are

$$x = 2,\ x = -1 \pm i,\ x = 3 \pm i\,.$$

This can be determined by using *Mathematica* and the commandFactor.

```
Factor[x^5 - 6 x^4 + 8 x^3 + 8 x^2 + 4 x - 40]  (-2 + x) (10 - 6 x + x^2) (2 + 2 x + x^2)
```

This still leaves some unanswered questions that we will answer in other modules.The quadratic factors can be determined using the Lin-Bairstow method.Or if one prefers complex arithmetic, then Newton's method can be used.For example, starting with the imaginary number

$r_0 = -2.0 + 1.0\,i$ Newton's method will create a complex sequence converging to the complex root $r = -1 + i$ of $P(x)$.

```
NewtonHorner[-2.0 + 1.0 i, 7];
r0 = -2.0000000000000000 + 1.0000000000000000 i,   P[r0] = 40. + 245. i
r1 = -1.4902651627811200 + 0.8511748152554390 i,   P[r1] = 3.343802536319485 + 76.84233652265891 i
r2 = -1.1180519317795970 + 0.8407886612644430 i,   P[r2] = -7.561158842520449 + 20.60448964481001 i
r3 = -0.9698985209513260 + 0.9719658806497940 i,   P[r3] = -4.458338420040017 - 0.1580185345289356 i
r4 = -1.0013370794844870 + 1.0012335968029000 i,   P[r4] = 0.2060381592934801 + 0.00829626577382392 i
r5 = -1.0000025032418580 + 1.0000022836709940 i,   P[r5] = 0.0003829543586180151 + 0.0000175674534095279 i
r6 = -1.0000000000087910 + 1.0000000000078390 i,   P[r6] = 1.330442422897704×10^-9 + (7.618794484187674×10^-11) i
r7 = -1.0000000000000000 + 1.0000000000000000 i,   P[r7] = 0. + 0. i
  r  = -1. + 1. i
 Δr  = ±1.17788×10^-11
P[r] = 0. + 0. i
```

However, starting with purely imaginary number $r_0 = 1.0\,i$ will create a divergent sequence.

```
NewtonHorner[1.0 i, 10];
r0 =  0.0000000000000000 + 1.0000000000000000 i,   P[r0] = -54. - 3. i
r1 = -0.3780821917808219 - 0.2082191780821918 i,   P[r1] = -40.65317886887399 - 0.4123382877891374 i
r2 =  2.1607575536125880 - 15.8894199160829300 i,   P[r2] = 308063.3308675807 - (1.00176134613537×10^6) i
r3 =  1.9717364373463110 - 12.6790172153154300 i,   P[r3] = 101002.2372526074 - 328828.1754869839 i
r4 =  1.8217743397572500 - 10.1021340366957000 i,   P[r4] = 33118.98201473677 - 108054.4759072764 i
r5 =  1.7037243827569750 - 8.0295102563098700 i,   P[r5] = 10859.46466311313 - 35572.09331145697 i
r6 =  1.6123410300834480 - 6.3567114471065750 i,   P[r6] = 3558.518713431996 - 11747.70164317031 i
r7 =  1.5443353184076870 - 4.9985659583630250 i,   P[r7] = 1163.209829649385 - 3901.676893738399 i
r8 =  1.4989195401959420 - 3.8843067461211420 i,   P[r8] = 377.1911839395179 - 1309.17892284011 i
r9 =  1.4794329632793190 - 2.9531157033987400 i,   P[r9] = 119.2871452998118 - 447.3890976140948 i
r10 =  1.4975401054337750 - 2.1504161125202740 i,   P[r10] = 34.80806905605333 - 157.4405784778136 i
  r  = 1.497540105433775 - 2.150416112520274 i
 Δr  = ±0.802904
P[r] = 34.80806905605333 - 157.4405784778136 i
```

For cases involving complex numbers the reader should look at the Lin-Bairstow and the Fundamental Theorem of Algebra modules.

Getting Real Roots

The following example illustrates polynomial deflation and shows that the order in which the roots are located could be important.In light of example 6 we know that better calculations are made for evaluating $P(x)$ whenxis small.The Newton-Horner subroutine is modified to terminate early if $P(x)$ evaluates close to zero (when a root is located).

Mathematica Subroutine (Newton-Horner Iteration).

```
NewtonHorner[x0_, max_, ϵ0_] :=
 Module[{ϵ = ϵ0, j = 0, k, n = Length[a] - 1, r0 = N[x0]},
  c = b = Table[0, {i, n + 1}];
  b[[1]] = 1;
  r1 = r0;
  While[ And[j < max, Abs[b[[1]]] > ϵ],
   r0 = r1;
   b[[n+1]] = a[[n+1]];
   For[ k = n, 1 ≤ k, k--, b[[k]] = a[[k]] + r0 b[[k+1]]; ];
   c[[n+1]] = b[[n+1]];
   For[ k = n, 2 ≤ k, k--, c[[k]] = b[[k]] + r0 c[[k+1]]; ];
   r1 = r0 - b[[1]] / c[[2]];
   j = j + 1; ];
  Return[r1]; ]
```

The Solution of Linear Systems AX = B:

Forward Substitution and Back Substitution:

Background

We will now develop the back-substitution algorithm , which is useful for solving a linear system of equations that has an upper-triangular coefficient matrix.

Definition (Upper-Triangular Matrix). An $n \times n$ matrix $A = [a_{i,j}]$ is called *upper-triangular* provided that the elements satisfy $a_{i,j} = 0$ whenever $i > j$.

If A is an upper-triangular matrix, then $AX = B$ is said to be an upper-triangular system of linear equations.

$$\begin{aligned}
a_{1,1}x_1 + a_{1,2}x_2 + a_{1,3}x_3 + \cdots + a_{1,n-1}x_{n-1} + a_{1,n}x_n &= b_1 \\
a_{2,2}x_2 + a_{2,3}x_3 + \cdots + a_{2,n-1}x_{n-1} + a_{2,n}x_n &= b_2 \\
a_{3,3}x_3 + \cdots + a_{3,n-1}x_{n-1} + a_{3,n}x_n &= b_3 \\
\vdots \qquad\qquad & \quad \vdots \quad \vdots \\
a_{n-1,n-1}x_{n-1} + a_{n-1,n}x_n &= b_{n-1} \\
a_{n,n}x_n &= b_n
\end{aligned} \tag{1}$$

Theorem (Back Substitution). Suppose that $AX = B$ is an upper-triangular system with the form given above in (1). If $a_{i,i} \neq 0$ for $i = 1, 2, \ldots, n$ then there exists a unique solution.

The back substitution algorithm. To solve the upper-triangular system $AX = B$ by the method of back-substitution. Proceed with the method only if all the diagonal elements are nonzero.
First compute

$$x_1 = \frac{b_1}{a_{1,1}}$$

and then use the rule

$$x_i = \frac{b_i - \sum_{j=i+1}^{n} a_{i,j} x_j}{a_{i,i}} \quad \textbf{for} \quad i = n-1, n-2, \ldots, 1$$

Or, use the "generalized rule"

$$x_i = \frac{b_i - \sum_{j=i+1}^{n} a_{i,j} x_j}{a_{i,i}} \quad \textbf{for} \quad i = n, n-1, \ldots, 1$$

where the "understood convention" is that $\sum_{j=n+1}^{n} a_{n+1,j} x_j$ is an "empty summation" because the lower index of summation is greater than the upper index of summation.

Remark. The loop control structure will permit us to use one formula**.**

Mathematica Subroutine (Back Substitution).

```
BackSub[ a0_, b0_, n_ ] :=
  Module[ { a = a0,  b = b0,  i, j, x },
    x = Table[0, {n}];
    For[ i = n,  1 ≤ i,  i--,
      x[[i]] = (b[[i]] - Σ_{j=i+1}^{n} a[[i,j]] x[[j]]) / a[[i,i]] ; ];
    Return[x] ; ];
```

Pedagogical version for "printing all the details."

```
BackSubDetails[ a0_, b0_, n_ ] :=
 Module[ { a = a0, b = b0, i, j, x, X },
  x = Table["x"i, {i, 1, n}];
  X = Table["x"i, {i, 1, n}];
  For[ i = n, 1 ≤ i, i--,
   Print[Flatten[Chop[a[[i]].X] == b[[i,1]];
   If[i < n, Print[Flatten[Chop[a[[i]]].Flatten[x] == b[[i,1]];
    Print[a[[i,i]] x[[i]] == (b[[i]] - Sum[a[[i,j]] x[[j]], {j, i+1, n}])[[1]] ];];
   x[[i]] = (b[[i]] - Sum[a[[i,j]] x[[j]], {j, i+1, n}]) / a[[i,i]];
   Print["x"i, "  = ", x[[i,1]], "\n"];
  Return[x]; ];
```

Lower-triangular systems.

We will now develop the *lower-substitution algorithm*, which is useful for solving a linear system of equations that has a lower-triangular coefficient matrix.

Definition (Lower-Triangular Matrix). An $n \times n$ matrix $A = [a_{i,j}]$ is called *lower-triangular* provided that the elements satisfy $a_{i,j} = 0$ whenever $i < j$.

If A is an lower-triangular matrix, then $AX = B$ is said to be a lower-triangular system of linear equations.

$$
\begin{array}{rl}
 & a_{1,1}x_1 = b_1 \\
 & a_{2,1}x_1 + a_{2,2}x_2 = b_2 \\
 & a_{3,1}x_1 + a_{3,2}x_2 + a_{3,3}x_3 = b_3 \\
 & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \vdots \qquad\quad \vdots \qquad\quad \vdots \\
 & a_{n-1,1}x_1 + a_{n-1,2}x_2 + a_{n-1,3}x_3 + \cdots + a_{n-1,n-1}x_{n-1} = b_{n-1} \\
\textbf{(2)} \quad & a_{n,1}x_1 + a_{n,2}x_2 + a_{n,3}x_3 + \cdots + a_{n,n-1}x_{n-1} + a_{n,n}x_n = b_n
\end{array}
$$

Theorem (Forward Substitution). Suppose that $AX = B$ is an lower-triangular system with the form given above in (2). If $a_{i,i} \neq 0$ for $i = 1, 2, \ldots, n$ then there exists a unique solution.

The forward substitution algorithm. To solve the lower-triangular system $AX = B$ by the method offorward-substitution. Proceed with the method only if all the diagonal elements are nonzero. First compute

$$x_1 = \frac{b_1}{a_{1,1}}$$

and then use the rule

$$x_i = \frac{b_i - \sum_{j=1}^{i-1} a_{i,j}\, x_j}{a_{i,i}} \quad \textbf{for} \quad i = 2, 3, \ldots, n.$$

Remark. The loop control structure will permit us to use one formula

$$x_i = \frac{b_i - \sum_{j=1}^{i-1} a_{i,j}\, x_j}{a_{i,i}} \quad \textbf{for} \quad i = 1, 2, \ldots, n.$$

Mathematical Subroutine (Forward Substitution).

```
ForeSub[ a0_, b0_, n_ ] :=
  Module[ { a = a0, b = b0, i, j , x},
    x = Table[0, {n}];
    For[ i = 1, i ≤ n, i++,
      x[[i]] = (b[[i]] - Sum[a[[i,j]] x[[j]], {j, 1, i-1}]) / a[[i,i]] ];
    Return[x]; ];
```

The Newton Interpolation Polynomial.

The following result is an alternate representation for a polynomial which is useful in the area of interpolation.

Definition (Newton Polynomial). The following expression is called a Newton polynomial of degree n.

$$\begin{aligned} P(t) = {} & c_1 + c_2 (t - x_1) + c_3 (t - x_1)(t - x_2) + c_4 (t - x_1)(t - x_2)(t - x_3) \\ & + \ldots \\ & + c_{n+1} (t - x_1)(t - x_2)(t - x_3) \ldots (t - x_n) \end{aligned}$$

or

$$P(t) = \sum_{i=1}^{n+1} \left(c_i \prod_{j=1}^{i-1} (t - x_j) \right)$$

If n+1 points $\{(x_k, y_k)\}_{k=1}^{n+1}$ are given, then the followingequations can be used to solve for the n+1 coefficients $\{c_i\}_{i=1}^{n+1}$.

$$P(x_k) = y_k$$

or

$$\sum_{i=1}^{n+1} \left(c_i \prod_{j=1}^{i-1} (x_k - x_j) \right) = y_k \quad \textbf{for} \quad k=1, 2,..., n+1.$$

This system of equations is lower-triangular.

$$c_1 = y_1$$

$$c_1 + c_2 (x_2 - x_1) = y_2$$

$$c_1 + c_2 (x_3 - x_1) + c_3 (x_3 - x_1) (x_3 - x_2) = y_3$$

$$c_1 + c_2 (x_4 - x_1) + c_3 (x_4 - x_1) (x_4 - x_2) + c_4 (x_4 - x_1) (x_4 - x_2) (x_4 - x_3) = y_4$$

...

$$c_1 + c_2 (x_{n+1} - x_1) + c_3 (x_{n+1} - x_1) (x_{n+1} - x_2) + \ldots + c_{n+1} (x_{n+1} - x_1) (x_{n+1} - x_2) (x_{n+1} - x_3) = y_{n+1}$$

Gauss-Jordan Elimination:

In this module we develop a algorithm for solving a general linear system of equations $\mathbf{AX = B}$ consisting of nequations and n unknowns where it is assumed that the system has a unique solution. The method is attributedJohann Carl Friedrich Gauss (1777-1855) and Wilhelm Jordan(1842 to 1899). The following theorem states the sufficient conditions for the existence and uniqueness of solutions of a linear system $\mathbf{AX = B}$.

Theorem (Unique Solutions) Assume that $\mathbf{A}$ is an $n \times n$ matrix. The following statements are equivalent.

(i) Given any $n \times 1$ matrix $\mathbf{B}$, the linear system $\mathbf{AX = B}$ has a unique solution.

(ii) The matrix $\mathbf{A}$ is nonsingular (i.e., $\mathbf{A^{-1}}$ exists).

(iii) The system of equations $\mathbf{AX = 0}$ has the unique solution $\mathbf{X = 0}$.

(iv) The determinant of $\mathbf{A}$ is nonzero, i.e. $\det(\mathbf{A}) \neq 0$.

It is convenient to store all the coefficients of the linear system $\mathbf{AX = B}$ in one array of dimension $n \times n+1$. The coefficients of $\mathbf{B}$ are stored in column $n+1$ of the array

(i.e. $a_{i,n+1} = b_i$). Row k contains all the coefficients necessary to represent the i^{th} equation in the linear system. The augmented matrix is denoted $\mathbf{M} = [\mathbf{A} \mid \mathbf{B}]$ and the linear system is represented as follows:

$$\mathbf{M} = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,n} & b_1 \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,n} & b_2 \\ a_{3,1} & a_{3,2} & a_{3,3} & \cdots & a_{3,n} & b_3 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ a_{n,1} & a_{n,2} & a_{n,3} & \cdots & a_{n,n} & b_n \end{pmatrix}$$

The system $\mathbf{AX} = \mathbf{B}$, with augmented matrix $\mathbf{M}$, can be solved by performing row operations on $\mathbf{M}$. The variables are placeholders for the coefficients and cam be omitted until the end of the computation.

Theorem (Elementary Row Operations). The following operations applied to the augmented matrix $\mathbf{M}$ yield an equivalent linear system.

(i) Interchanges: The order of two rows can be interchanged.

(ii) Scaling: Multiplying a row by a nonzero constant.

(iii) Replacement: Row r can be replaced by the sum of that tow and a nonzero multiple of any other row;

that is: $\text{row}_r = \text{row}_r - c\,\text{row}_p$.

It is common practice to implement (iii) by replacing a row with the difference of that row and a multiple of another row.

Definition (Pivot Element). The number $a_{p,p}$ in the coefficient matrix $\mathbf{A}$ that is used to eliminate $a_{i,p}$ where $i = p+1, p+2, \ldots, n$, is called the p^{th} pivot element, and the p^{th} row is called the pivot row.

Theorem (Gaussian Elimination with Back Substitution). Assume that $\mathbf{A}$ is an $n \times n$ nonsingular matrix. There exists a unique system $\mathbf{UX} = \mathbf{Y}$ that is equivalent to the given system $\mathbf{AX} = \mathbf{B}$, where $\mathbf{U}$ is an upper-triangular matrix with $u_{i,i} \neq 0$ for $i = 1, 2, \ldots, n$. After $\mathbf{U}$ and $\mathbf{Y}$ are constructed, back substitution can be used to solve $\mathbf{UX} = \mathbf{Y}$ for $\mathbf{X}$.

Algorithm I. (Limited Gauss-JordanElimination). Construct the solution to the linear system $\mathbf{AX} = \mathbf{B}$ by using Gauss-Jordan elimination under the assumption that row interchanges are not needed. The following subroutine uses row operations to eliminate x_p in column p for $p = 1, 2, \ldots, n$.

Mathematical Subroutine (Limited Gauss-JordanElimination).

```
GaussJordan[A0_, n_] :=
 Module[ {A = A0, i, p},
  Print[ MatrixForm[A] ];
  For[ p = 1, p ≤ n, p++,
   A[[p]] = A[[p]] / A[[p,p]];
   For[ i = 1, i ≤ n, i++,
    If[ i ≠ p,
     A[[i]] = A[[i]] - A[[i,p]] A[[p]] ]; ];
   Print[MatrixForm[A]]; ]; ]
```

Provide for row interchanges in the Gauss-Jordan subroutine.

Add the following loop that will interchange rows and perform partial pivoting.

```
For[ k = p + 1, k ≤ n, k++,
 If[ Abs[A[[k,p]]] > Abs[A[[p,p]]],
  A[[{p,k}]] = A[[{k,p}]]; ]; ];
```

To make these changes, copy your subroutine GaussJordan and place a copy below. Then you can copy the above lines by selecting them and then use the "Edit" and "Copy" menus. The improved Gauss-Jordan subroutine should look like this (blue is for placement information only). Or just use the active *Mathematica* code given below.

Algorithm II. (Complete Gauss-JordanElimination). Construct the solution to the linear system $AX = B$ by using Gauss-Jordan elimination. Provision is made for row interchanges if they are needed.

Mathematical Subroutine (Complete Gauss-JordanElimination).

```
GaussJordan[A0_, n_] :=
 Module[ {A = A0, i, k, p},
  Print[ MatrixForm[A] ];
  For[ p = 1, p ≤ n, p++,
   For[ k = p + 1, k ≤ n, k++,
    If[ Abs[A[[k,p]]] > Abs[A[[p,p]]],
     A[[{p,k}]] = A[[{k,p}]]; ]; ];
   A[[p]] = A[[p]] / A[[p,p]];
   For[ i = 1, i ≤ n, i++,
    If[ i ≠ p,
     A[[i]] = A[[i]] - A[[i,p]] A[[p]] ]; ];
   Print[MatrixForm[A]]; ];
  Return[A]; ]
```

Use the subroutine "GaussJordan" to find the inverseof a matrix.

Theorem (Inverse Matrix) Assume that A is an $n \times n$ nonsingular matrix. Form the augmented matrix $M = [A \mid I_{n,n}]$ of dimension $n \times 2n$. Use Gauss-Jordanelimination to reduce the matrix M so that the identity $I_{n,n}$ is in the first n columns. Then the inverse A^{-1} is located in columns $n, n+1, \ldots, 2n$.

Algorithm III. (Concise Gauss-JordanElimination).nbsp; Construct the solution to the linear system $AX = B$ by using Gauss-Jordan elimination. The print statements are for pedagogical purposes and are not needed.

Mathematical Subroutine (Concise Gauss-JordanElimination).

```
GaussJordan[A0_, n_] :=
 Module[ {A = A0, i, k, p},
  For[ p = 1, p ≤ n, p++,
   For[ k = p + 1, k ≤ n, k++,
    If[ Abs[A[[k,p]]] > Abs[A[[p,p]]],
     A[[{p,k}]] = A[[{k,p}]]; ]; ];
   A[[p]] = A[[p]] / A[[p,p]];
   For[ i = 1, i ≤ n, i++,
    If[ i ≠ p,
     A[[i]] = A[[i]] - A[[i,p]] A[[p]] ]; ]; ];
  Return[ A ]; ]
```

Remark. The Gauss-Jordan elimination method is the "heuristic" scheme found in most linear algebratextbooks. The line of code

$$A_{[[p]]} = \frac{A_{[[p]]}}{A_{[[p,p]]}};$$

divides each entry in the pivot row by its leading coefficient $A_{[[p,p]]}$. Is this step necessary? A more computationally efficient algorithm will be studied which uses upper-triangularization followed by back substitution. The partial pivoting strategy will also be employed, which reduces propagated error and instability.

Application to Polynomial Interpolation

Consider a polynomial of degree n=5 that passes through the six points (x_i, y_i) for $i = 1, 2, 3, 4, 5, 6$;

$$p[x] = c_1 + c_2 x + c_3 x^2 + c_4 x^3 + c_5 x^4 + c_6 x^5 .$$

For each point (x_i, y_i) is used to an equation $p[x_i] = y_i$, which in turn are used to write a system of six equations in six unknowns $\{c_i\}_{i=1}^{6}$

$$c_1 + c_2 x_1 + c_3 x_1^2 + c_4 x_1^3 + c_5 x_1^4 + c_6 x_1^5 = y_1$$

$$c_1 + c_2 x_2 + c_3 x_2^2 + c_4 x_2^3 + c_5 x_2^4 + c_6 x_2^5 = y_2$$

$$c_1 + c_2 x_3 + c_3 x_3^2 + c_4 x_3^3 + c_5 x_3^4 + c_6 x_3^5 = y_3$$

$$c_1 + c_2 x_4 + c_3 x_4^2 + c_4 x_4^3 + c_5 x_4^4 + c_6 x_4^5 = y_4$$

$$c_1 + c_2 x_5 + c_3 x_5^2 + c_4 x_5^3 + c_5 x_5^4 + c_6 x_5^5 = y_5$$

$$c_1 + c_2 x_6 + c_3 x_6^2 + c_4 x_6^3 + c_5 x_6^4 + c_6 x_6^5 = y_6$$

The above system can be written in matrix form **MC** = **B**

$$\begin{pmatrix} 1 & x_1 & x_1^2 & x_1^3 & x_1^4 & x_1^5 \\ 1 & x_2 & x_2^2 & x_2^3 & x_2^4 & x_2^5 \\ 1 & x_3 & x_3^2 & x_3^3 & x_3^4 & x_3^5 \\ 1 & x_4 & x_4^2 & x_4^3 & x_4^4 & x_4^5 \\ 1 & x_5 & x_5^2 & x_5^3 & x_5^4 & x_5^5 \\ 1 & x_6 & x_6^2 & x_6^3 & x_6^4 & x_6^5 \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \end{pmatrix}$$

Solve this linear system for the coefficients $\{c_i\}_{i=1}^{6}$ and then construct the interpolating polynomial

$$p[x] = c_1 + c_2 x + c_3 x^2 + c_4 x^3 + c_5 x^4 + c_6 x^5 .$$

The Matrix Inverse:

Background

Theorem (Inverse Matrix) Assume that $\mathbf{A}$ is an $n \times n$ nonsingular matrix. Form the augmented matrix $\mathbf{M} = [\mathbf{A} \mid \mathbf{I}_{n,n}]$ of dimension $n \times 2n$. Use Gauss-Jordanelimination to reduce the matrix $\mathbf{M}$ so that the identity $\mathbf{I}_{n,n}$ is in the first n columns. Then the inverse $\mathbf{A}^{-1}$ is located in columns $n, n+1, \ldots, 2n$. The augmented matrix $\mathbf{M} = [\mathbf{A} \mid \mathbf{I}_{n,n}]$ looks like:

$$\mathbf{M} = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,n} & 1 & 0 & 0 & \cdots & 0 \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,n} & 0 & 1 & 0 & \cdots & 0 \\ a_{3,1} & a_{3,2} & a_{3,3} & \cdots & a_{3,n} & 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & & \vdots & & & \vdots & & \\ a_{n,1} & a_{n,2} & a_{n,3} & \cdots & a_{n,n} & 0 & 0 & 0 & \cdots & 1 \end{pmatrix}$$

We can use the previously developed Gauss-Jordan subroutine to find the inverse of a matrix.

Algorithm (Complete Gauss-JordanElimination). Construct the solution to the linear system $AX = B$ by using Gauss-Jordan elimination. Provision is made for row interchanges if they are needed.

Mathematica Subroutine (Complete Gauss-JordanElimination).

```
GaussJordan[A0_, n_] :=
 Module[ {A = A0, i, k, p},
  Print[ MatrixForm[A] ];
  For[ p = 1, p ≤ n, p++,
   For[ k = p + 1, k ≤ n, k++,
    If[ Abs[A[[k,p]]] > Abs[A[[p,p]]],
     A[[{p,k}]] = A[[{k,p}]]; ]; ];
   A[[p]] = A[[p]] / A[[p,p]];
   For[ i = 1, i ≤ n, i++,
    If[ i ≠ p,
     A[[i]] = A[[i]] - A[[i,p]] A[[p]] ]; ];
   Print[MatrixForm[A]]; ];
  Return[A]; ]
```

Definition (Hilbert Matrix). The elements of the Hilbert matrix H_n of order n are $h_{i,j} = \frac{1}{i+j-1}$ for $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, n$.

$$H_n = \begin{pmatrix} 1 & \frac{1}{2} & \cdots & \frac{1}{n-1} & \frac{1}{n} \\ \frac{1}{2} & \frac{1}{3} & \cdots & \frac{1}{n} & \frac{1}{n+1} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \frac{1}{n-1} & \frac{1}{n} & \cdots & \frac{1}{2n-3} & \frac{1}{2n-2} \\ \frac{1}{n} & \frac{1}{n+1} & \cdots & \frac{1}{2n-2} & \frac{1}{2n-1} \end{pmatrix}$$

The Inverse Hilbert Matrix

The formula for the elements of the inverse Hilbert matrix H_n of order n is known to be

$$(H^{-1})_{i,j} = \frac{(-1)^{i+j}}{i+j-1} \frac{(n+i-1)!\,(n+j-1)!}{((i-1)!\,(j-1)!)^2\,(n-i)!\,(n-j)!}$$

which can be expressed using binomial coefficients

$$(H^{-1})_{i,j} = (-1)^{i+j}\,(i+j-1)\binom{n+i-1}{n-j}\binom{n+j-1}{n-i}\binom{i+j-2}{i-1}^2.$$

When exact computations are needed these formulas should be used instead of using a subroutine or built in procedure for computing the inverse of H_n.

Application to Continuous Least Squares Approximation

The continuous least squares approximation to a function $f(x)$ on the interval [0,1] for the set of functions $\{u_i(x)\}_{i=1}^{n}$ can solved by using the normal equations

(1) $$\sum_{j=1}^{n} c_i \langle u_i, u_j \rangle = \langle f, u_i \rangle \quad \text{for} \quad 1 \le i \le n.$$

Where the inner product is $\langle f, g \rangle = \int_0^1 f(x)\, g(x)\, dx$. Solve the linear system (1) for the coefficients $\{c_i\}_{i=1}^{n}$ and construct the approximation function

$$F[x] = \sum_{i=1}^{n} c_i u_i(x).$$

Definition (Gram Matrix). The Gram matrix **G** is a matrix of inner products where the elements are $G_{i,j} = \langle u_i, u_j \rangle$.

The case when the set of functions is $\{u_i(x)\}_{i=1}^{n} = \{1, x, x^2, \ldots, x^{n-2}, x^{n-1}\}$ will produce the Hilbert matrix. Since we require the computation to be as exact as possible and an exact formula is known for the inverse of the Hilbert matrix, this is an example where an inverse matrix comes in handy.

Cholesky, Doolittle and Crout Factorization

Background

Definition (LU-Factorization). The nonsingular matrix A has an LU-factorization if it can be expressed as the product of a lower-triangular matrix L and an upper triangular matrix U:

$$\mathbf{A} = \mathbf{LU}.$$

When this is possible we say that A has an LU-decomposition. It turns out that this factorization (when it exists) is not unique. If L has 1's on it's diagonal, then it is called a Doolittle factorization. If U has 1's on its diagonal, then it is called a Crout factorization. When $\mathbf{U} = \mathbf{L}^T$

(or $L = U^T$), it is called a Cholesky decomposition.

Theorem (A = LU; Factorization with NO Pivoting). Assume thatA has a Doolittle, Crout or Cholesky factorization. The solution X to the linear system $AX = B$, is found in three steps:

1. Construct the matrices L and U, if possible.
2. Solve $LY = B$ for Y using forward substitution.
3. Solve $UX = Y$ for X using back substitution.

Theorem (A = LU; Doolittle Factorization). Assume that A has aDoolittle factorization A = LU.

$$\begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} & \cdots & a_{2,n} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} & \cdots & a_{3,n} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} & \cdots & a_{4,n} \\ \\ a_{n,1} & a_{n,2} & a_{n,3} & a_{n,4} & \cdots & a_{n,n} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ l_{2,1} & 1 & 0 & 0 & \cdots & 0 \\ l_{3,1} & l_{3,2} & 1 & 0 & \cdots & 0 \\ l_{4,1} & l_{4,2} & l_{4,3} & 1 & \cdots & 0 \\ \\ l_{n,1} & l_{n,2} & l_{n,3} & l_{n,4} & \cdots & 1 \end{pmatrix}$$

$$\begin{pmatrix} u_{1,1} & u_{1,2} & u_{1,3} & u_{1,4} & - - - & u_{1,n} \\ 0 & u_{2,2} & u_{2,3} & u_{2,4} & - - - & u_{2,n} \\ 0 & 0 & u_{3,3} & u_{3,4} & - - - & u_{3,n} \\ 0 & 0 & 0 & u_{4,4} & - - - & u_{4,n} \\ \\ 0 & 0 & 0 & 0 & - - - & u_{n,n} \end{pmatrix}$$

The solution X to the linear system $AX = B$, is found in three steps:

1. **Construct the matrices L and U, if possible.**
2. **Solve $LY = B$ for Y using forward substitution.**
3. **Solve $UX = Y$ for X using back substitution.**

For curiosity, the reader might be interested in other methods ofcomputing L and U**.**

Theorem (A = LU; Crout Factorization). Assume that A has a Crout factorization A = LU.

$$\begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & - - - & a_{1,n} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} & - - - & a_{2,n} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} & - - - & a_{3,n} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} & - - - & a_{4,n} \\ \\ a_{n,1} & a_{n,2} & a_{n,3} & a_{n,4} & - - - & a_{n,n} \end{pmatrix} =$$

$$\begin{pmatrix} l_{1,1} & 0 & 0 & 0 & - - - & 0 \\ l_{2,1} & l_{2,2} & 0 & 0 & - - - & 0 \\ l_{3,1} & l_{3,2} & l_{3,3} & 0 & - - - & 0 \\ l_{4,1} & l_{4,2} & l_{4,3} & l_{4,4} & - - - & 0 \\ \\ l_{n,1} & l_{n,2} & l_{n,3} & l_{n,4} & - - - & l_{n,n} \end{pmatrix}$$

$$\begin{pmatrix} 1 & u_{1,2} & u_{1,3} & u_{1,4} & \cdots & u_{1,n} \\ 0 & 1 & u_{2,3} & u_{2,4} & \cdots & u_{2,n} \\ 0 & 0 & 1 & u_{3,4} & \cdots & u_{3,n} \\ 0 & 0 & 0 & 1 & \cdots & u_{4,n} \\ 0 & 0 & 0 & 0 & \cdots & 1 \end{pmatrix}$$

The solution X to the linear system $AX = B$, is found in three steps:

1. **Construct the matrices L and U, if possible.**
2. **Solve $LY = B$ for Y using forward substitution.**
3. **Solve $UX = Y$ for X using back substitution.**

Mathematical Subroutine (Doolittle).

```
Doolittle[n_] :=
 Module[{i, j, k, m},
  L = Table[0, {n}, {n}];
  U = L;
  For[k = 1, k ≤ n, k++,
   L[[k,k]] = 1;
   For[j = k, j ≤ n, j++,
    U[[k,j]] = A[[k,j]] - Sum[L[[k,m]] U[[m,j]], {m, 1, k-1}]];
   For[i = k + 1, i ≤ n, i++,
    L[[i,k]] = (A[[i,k]] - Sum[L[[i,m]] U[[m,k]], {m, 1, k-1}]) / U[[k,k]]]]]
```

Mathematical Subroutine (Crout).

```
Crout[n_] :=
 Module[{i, j, k, m},
  L = Table[0, {n}, {n}];
  U = L;
  For[k = 1, k ≤ n, k++,
   L[[k,k]] = A[[k,k]] - Sum[L[[k,m]] U[[m,k]], {m, 1, k-1}];
   For[j = k, j ≤ n, j++,
    U[[k,j]] = (A[[k,j]] - Sum[L[[k,m]] U[[m,j]], {m, 1, k-1}]) / L[[k,k]];
   For[i = k + 1, i ≤ n, i++,
    L[[i,k]] = (A[[i,k]] - Sum[L[[i,m]] U[[m,k]], {m, 1, k-1}]) / U[[k,k]]]]]
```

Mathematical Subroutine (Forward Elimination).

```
ForeSub[n_] :=
 Module[{i, j},
  Y = Table[0, {n}];
  Y[[n]] = B[[n]] / L[[n, n]];
  For[i = 1, i ≤ n, i++,
   Y[[i]] = (B[[i]] - Sum[L[[i, j]] Y[[j]], {j, 1, i - 1}]) / L[[i, i]]]]
```

Mathematical Subroutine (Back Substitution).

```
BackSub[n_] :=
 Module[{i, j},
  X = Table[0, {n}];
  X[[n]] = Y[[n]] / U[[n, n]];
  For[i = n - 1, 1 ≤ i, i--,
   X[[i]] = (Y[[i]] - Sum[U[[i, j]] X[[j]], {j, i + 1, n}]) / U[[i, i]]]]
```

Theorem (Cholesky Factorization). If A is real, symmetric and positive definite matrix, then it has a Cholesky factorization

$$A = U^T U$$

where U an upper triangular matrix.

Remark. Observe that $L = U^T$ is a lower triangular matrix, so that A= LU. Hence we could also write Cholesky factorization

$$A = L L^T$$

where L a lower triangular matrix.

Theorem (A = LU; Cholesky Factorization). Assume that A has a Cholesky factorization $A = U^T U$, where $L = U^T$.

$$\begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & - - - & a_{1,n} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} & - - - & a_{2,n} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} & - - - & a_{3,n} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} & - - - & a_{4,n} \\ & & & & & \\ a_{n,1} & a_{n,2} & a_{n,3} & a_{n,4} & - - - & a_{n,n} \end{pmatrix} =$$

$$\begin{pmatrix} u_{1,1} & 0 & 0 & 0 & \cdots & 0 \\ u_{1,2} & u_{2,2} & 0 & 0 & \cdots & 0 \\ u_{1,3} & u_{2,3} & u_{3,3} & 0 & \cdots & 0 \\ u_{1,4} & u_{2,4} & u_{3,4} & u_{4,4} & \cdots & 0 \\ & & & & & \\ u_{1,n} & u_{2,n} & u_{3,n} & u_{4,n} & \cdots & u_{n,n} \end{pmatrix} \begin{pmatrix} u_{1,1} & u_{1,2} & u_{1,3} & u_{1,4} & \cdots & u_{1,n} \\ 0 & u_{2,2} & u_{2,3} & u_{2,4} & \cdots & u_{2,n} \\ 0 & 0 & u_{3,3} & u_{3,4} & \cdots & u_{3,n} \\ 0 & 0 & 0 & u_{4,4} & \cdots & u_{4,n} \\ & & & & & \\ 0 & 0 & 0 & 0 & \cdots & u_{n,n} \end{pmatrix}$$

Or if you prefer to write the Cholesky factorization as $A = L L^T$, where $U = L^T$.

$$\begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} & \cdots & a_{2,n} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} & \cdots & a_{3,n} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} & \cdots & a_{4,n} \\ & & & & & \\ a_{n,1} & a_{n,2} & a_{n,3} & a_{n,4} & \cdots & a_{n,n} \end{pmatrix} = \begin{pmatrix} l_{1,1} & 0 & 0 & 0 & \cdots & 0 \\ l_{2,1} & l_{2,2} & 0 & 0 & \cdots & 0 \\ l_{3,1} & l_{3,2} & l_{3,3} & 0 & \cdots & 0 \\ l_{4,1} & l_{4,2} & l_{4,3} & l_{4,4} & \cdots & 0 \\ & & & & & \\ l_{n,1} & l_{n,2} & l_{n,3} & l_{n,4} & \cdots & l_{n,n} \end{pmatrix} \begin{pmatrix} l_{1,1} & l_{2,1} & l_{3,1} & l_{4,1} & \cdots & l_{n,1} \\ 0 & l_{2,2} & l_{3,2} & l_{4,2} & \cdots & l_{n,2} \\ 0 & 0 & l_{3,3} & l_{4,3} & \cdots & l_{n,3} \\ 0 & 0 & 0 & l_{4,4} & \cdots & l_{n,4} \\ & & & & & \\ 0 & 0 & 0 & 0 & \cdots & l_{n,n} \end{pmatrix}$$

The solution X to the linear system $AX = B$, is found in three steps:

1. Construct the matrices L and $U = L^T$, if possible.
2. Solve $LY = B$ for Y using forward substitution.
3. Solve $UX = Y$ for X using back substitution**.**

The following Cholesky subroutine can be used when the matrix A is real, symmetric and positive definite.

Observe that the loop starting with For[j=k,j<=n,j++, is notnecessary and that U is computed by forming the transpose of L.

Mathematical Subroutine (Cholesky factorization).

```
Cholesky[n_] :=
 Module[{i, k, m},
  L = Table[0, {n}, {n}];
  For[k = 1, k ≤ n, k++,
   L[[k,k]] = Sqrt[A[[k,k]] - Sum[(L[[k,m]])^2, {m, 1, k-1}]];
   For[i = k + 1, i ≤ n, i++,
    L[[i,k]] = (A[[i,k]] - Sum[L[[i,m]] L[[k,m]], {m, 1, k-1}]) / L[[k,k]]]];
  U = Transpose[L]]
```

Application to Polynomial Curve Fitting

Theorem (Least-Squares Polynomial Curve Fitting). Given the n data points $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, the least squares polynomial of degree m of the form

$$P_m(x) = c_1 + c_2 x + c_3 x^2 + \ldots + c_m x^{m-1} + c_{m+1} x^m$$

that fits the n data points is obtained by solving the following linear system

$$\begin{pmatrix} n & \sum_{i=1}^n x_i & \sum_{i=1}^n x_i^2 & \cdots & \sum_{i=1}^n x_i^m \\ \sum_{i=1}^n x_i & \sum_{i=1}^n x_i^2 & \sum_{i=1}^n x_i^3 & \cdots & \sum_{i=1}^n x_i^{m+1} \\ \sum_{i=1}^n x_i^2 & \sum_{i=1}^n x_i^3 & \sum_{i=1}^n x_i^4 & \cdots & \sum_{i=1}^n x_i^{m+2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \sum_{i=1}^n x_i^m & \sum_{i=1}^n x_i^{m+1} & \sum_{i=1}^n x_i^{m+2} & \cdots & \sum_{i=1}^n x_i^{2m} \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_{m+1} \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^n y_i \\ \sum_{i=1}^n x_i y_i \\ \sum_{i=1}^n x_i^2 y_i \\ \vdots \\ \sum_{i=1}^n x_i^m y_i \end{pmatrix}$$

for the m+1 coefficients $\{c_1, c_2, \ldots, c_m, c_{m+1}\}$. These equationsare referred to as the "normal equations".

Application to Continuous Least Squares Approximation

The continuous least squares approximation to a function $f(x)$ on the interval [0,1] for the set of functions $\{u_i(x)\}_{i=1}^n$ can solved by using the normal equations

(1) $$\sum_{j=1}^n c_i \langle u_i, u_j \rangle = \langle f, u_i \rangle$$ **for** $1 \le i \le n$.

Where the inner product is $\langle f, g\rangle = \int_0^1 f(x)\, g(x)\, dx$. Solve the linear system (1) for the coefficients $\{c_i\}_{i=1}^n$ and construct the approximation function

$$F[x] = \sum_{i=1}^{n} c_i u_i(x)$$

.

Definition (Gram Matrix). The Gram matrix G is a matrix of inner products where the elements are $G_{i,j} = \langle u_i, u_j\rangle$ **.**

The case when the set of functions is $\{u_i(x)\}_{i=1}^n = \{1, x, x^2, \ldots, x^{n-2}, x^{n-1}\}$ will produce the Hilbert matrix. Since we require the computation to be as exact as possibleand an exact formula is known for the inverse of the Hilbert matrix, this is an example where an inverse matrix comes in handy**.**

Jacobi and Gauss-Seidel Iteration

Background

Iterative schemes require time to achieve sufficient accuracy and are reserved for large systems of equationswhere there are a majority of zero elements in the matrix. Often times the algorithms are taylor-made to takeadvantage of the special structure such as band matrices. Practical uses include applications in circuit analysis, boundary value problems and partial differentialequations.

Iteration is a popular technique finding roots ofequations. Generalization of fixed point iteration can be applied to systems of linear equations to produce accurateresults. The method Jacobi iteration is attributed to Carl Jacobi (1804-1851) and Gauss-Seidel iteration is attributed to Johann Carl Friedrich Gauss (1777-1855) and Philipp Ludwig von Seidel (1821-1896).

Consider that the n×n square matrix **A** is split into three parts, the main diagonal **D**, below diagonal **L** and above diagonal **U**. We have **A** = **D** - **L** - **U**.

$$\begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,n-2} & a_{1,n-1} & a_{1,n} \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,n-2} & a_{2,n-1} & a_{2,n} \\ a_{3,1} & a_{3,2} & a_{3,3} & \cdots & a_{3,n-2} & a_{3,n-1} & a_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ a_{n-2,1} & a_{n-2,2} & a_{n-2,3} & \cdots & a_{n-2,n-2} & a_{n-2,n-1} & a_{n-2,n} \\ a_{n-1,1} & a_{n-1,2} & a_{n-1,3} & \cdots & a_{n-1,n-2} & a_{n-1,n-1} & a_{n-1,n} \\ a_{n,1} & a_{n,2} & a_{n,3} & \cdots & a_{n,n-2} & a_{n,n-1} & a_{n,n} \end{pmatrix} =$$

$$\begin{pmatrix} a_{1,1} & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & a_{2,2} & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & a_{3,3} & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & a_{n-2,n-2} & 0 & 0 \\ 0 & 0 & 0 & \cdots & 0 & a_{n-1,n-1} & 0 \\ 0 & 0 & 0 & \cdots & 0 & 0 & a_{n,n} \end{pmatrix} -$$

$$\begin{pmatrix} 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ -a_{2,1} & 0 & 0 & \cdots & 0 & 0 & 0 \\ -a_{3,1} & -a_{3,2} & 0 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ -a_{n-2,1} & -a_{n-2,2} & -a_{n-2,3} & \cdots & 0 & 0 & 0 \\ -a_{n-1,1} & -a_{n-1,2} & -a_{n-1,3} & \cdots & -a_{n-1,n-2} & 0 & 0 \\ -a_{n,1} & -a_{n,2} & -a_{n,3} & \cdots & -a_{n,n-2} & -a_{n,n-1} & 0 \end{pmatrix} -$$

$$\begin{pmatrix} 0 & -a_{1,2} & -a_{1,3} & \cdots & -a_{1,n-2} & -a_{1,n-1} & -a_{1,n} \\ 0 & 0 & -a_{2,3} & \cdots & -a_{2,n-2} & -a_{2,n-1} & -a_{2,n} \\ 0 & 0 & 0 & \cdots & -a_{3,n-2} & -a_{3,n-1} & -a_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & -a_{n-2,n-1} & -a_{n-2,n} \\ 0 & 0 & 0 & \cdots & 0 & 0 & -a_{n-1,n} \\ 0 & 0 & 0 & \cdots & 0 & 0 & 0 \end{pmatrix}$$

Definition (Diagonally Dominant). The matrix A is strictly diagonally dominant if

$$|a_{i,i}| > \sum_{j=1}^{i-1} |a_{i,j}| + \sum_{j=i+1}^{n} |a_{i,j}| \quad \text{for} \quad i = 1, 2, \ldots, n.$$

Theorem (Jacobi Iteration). The solution to the linear system $AX = B$ can be obtained starting with P_0 , and using iteration scheme

$$P_{k+1} = M_J P_k + C_J$$

where

$M_J = D^{-1}(L + U)$ and $C_J = D^{-1} B$. If P_0 is carefully chosen a sequence $\{P_k\}$ is generated which converges to the solution P, i.e. $AP = B$.
A sufficient condition for the method to be applicable is that **A**is strictly diagonally dominant or diagonally dominant and irreducible.

Theorem (Gauss-Seidel Iteration). The solution to the linear system $AX = B$ can be obtained starting with P_0 , and using iteration scheme

$$P_{k+1} = M_S P_k + C_S$$

where

$$M_S = (D - L)^{-1} U \quad \text{and} \quad C_S = (D - L)^{-1} B .$$

If P_0 is carefully chosen a sequence $\{P_k\}$ is generated which converges to the solution $\mathbf{P}$, i.e. $AP = B$.
A sufficient condition for the method to be applicable is that **A**is strictly diagonally dominant or diagonally dominant and irreducible.

Mathematical Subroutine (Jacobi Iteration).

```
Jacobi[A0_, B0_, P0_, max_] :=
 Module[{A = N[A0], B = N[B0], i, j, k = 0, n = Length[P0], P = P0, Pold = P0},
  Print["P"0, " = ", P];
  While[ k < max,
   For[ i = 1, i ≤ n, i++,
    P[[i]] = 1/A[[i,i]] (B[[i]] + A[[i,i]] Pold[[i]] - Sum[A[[i,j]] Pold[[j]], {j,1,n}]) ];
   Print["P"k+1, " = ", P];
   Pold = P;
   k = k + 1; ];
  Return[P]; ];
```

Mathematical Subroutine (Gauss-Seidel Iteration).

```
GaussSeidel[A0_, B0_, P0_, max_] :=
 Module[{A = N[A0], B = N[B0], i, j, k = 0, n = Length[P0], P = P0},
  Print["P"0, " = ", P];
  While[ k < max,
   For[ i = 1, i ≤ n, i++,
    P[[i]] = 1/A[[i,i]] (B[[i]] + A[[i,i]] P[[i]] - Sum[A[[i,j]] P[[j]], {j,1,n}]) ];
   Print["P"k+1, " = ", P];
   k = k + 1; ];
  Return[P]; ];
```

Warning.

Iteration does not always converge. A sufficient condition for iteration to Jacobi iteration to converge is that **A** is strictly diagonally dominant. The following subroutine will check to see if a matrix is strictly diagonally dominant. It should be used before any call to Jacobi iteration or Gauss-Seidel iteration is made. There exists a counter-example for which Jacobi iteration converges and Gauss-Seidel iteration does not converge. The "true" sufficient condition for Jacobi iteration to converge is that the "spectral radius" of $M = D^{-1}(A - D)$ is less than 1, where D is the diagonal of A. That is, the magnitude of the largest eigenvalue of **M**must be less than 1. This condition seems harsh because numerical computation of eigenvalues is an advanced topic compared to solution of a linear system.

```
Dominant[A_] :=
 Module[{cond = 1, n = Length[A]},
  For[ i = 1, i ≤ n, i++,
   If[ Sum[Abs[A[[i,j]]], {j,1,n}] > 2 Abs[A[[i,i]]], cond = 0; ]; ];
  If[ cond == 1,
   Print["The matrix IS strictly diagonally dominant."];
   Print["Jacobi iteration should converge."]; ,
   Print["The matrix is NOT strictly diagonally dominant."];
   Print["Jacobi iteration will FAIL !"]; ]; ];
```

More efficient subroutines

A tolerance can be supplied to either the Jacobi or Gauss-Seidel method which will permit it to exit the loop if convergence has been achieved.

Mathematical Subroutine (Jacobi Iteration).

```
Jacobi[A0_, B0_, P0_, ϵ_, max_] :=
 Module[{A = N[A0], B = N[B0], e, i, j, k = 0, n = Length[P0], P = P0, Pold = P0},
  Print["P"0, " = ", P];
  e = 1;
  While[ And[ k < max, e > ϵ ],
   For[ i = 1, i ≤ n, i++,
    P[[i]] = 1/A[[i,i]] (B[[i]] + A[[i,i]] Pold[[i]] - Sum[A[[i,j]] Pold[[j]], {j,1,n}]) ];
   e = Sqrt[(P - Pold).(P - Pold)];
   Print["P"k+1, " = ", P];
   Pold = P;
   k = k + 1];
  Return[P]; ];
```

Mathematical Subroutine (Gauss-Seidel Iteration).

```
GaussSeidel[A0_, B0_, P0_, ϵ_, max_] :=
 Module[{A = N[A0], B = N[B0], e, i, j, k = 0, n = Length[P0], P = P0, Pold = P0},
  Print["P"0, " = ", P];
  e = 1;
  While[ And[ k < max, e > ϵ ],
   Pold = P;
   For[ i = 1, i ≤ n, i++,
    P[[i]] = 1/A[[i,i]] (B[[i]] + A[[i,i]] P[[i]] - Sum[A[[i,j]] P[[j]], {j,1,n}]) ];
   e = Sqrt[(P - Pold).(P - Pold)];
   Print["P"k+1, " = ", P];
   k = k + 1];
  Return[P]; ];
```

Subroutines using matrix commands

In the Jacobi subroutine we can use fix point iteration as suggested by the theory.

Mathematical Subroutine (Jacobi Iteration).

```
JacobiFP[A0_, B0_, P0_, max_] :=
 Module[{A = N[A0], B = N[B0], i, k = 0, n = Length[P0], P = P0},
  Print["P"0, "  = ", P];
  d = DiagonalMatrix[Table[A[[i,i]], {i, Length[A]}]];
  M = -Inverse[d].(A - d);
  c = Inverse[d].(B);
  For[k = 1, k ≤ max, k++,
   P = c + M.P;
   Print["P"k+1, "  = ", P]; ];
  Return[P]; ];
```

Successive Over Relaxation - SOR Method:

Background

Suppose that iteration is used to solve the linear system $AX = B$, and that P_k is an approximate solution. We call $R_k = B - AP_k$ the residual vector, and if P_k is a good approximation then $R_k \approx 0$. A method based on reducing the norm of the residual will produce a sequence $\{P_k\}$ that converges faster. The successive over relaxation - SOR method introduces a parameter ω which speeds up convergence. The SOR method can be used in the numerical solution of certain partial differential equations.

Consider that the n×n square matrix **A** is split into three parts, the main diagonal **D**, below diagonal **L** and above diagonal **U**. We have **A** = **D** - **L** - **U**.

$$\begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,n-2} & a_{1,n-1} & a_{1,n} \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,n-2} & a_{2,n-1} & a_{2,n} \\ a_{3,1} & a_{3,2} & a_{3,3} & \cdots & a_{3,n-2} & a_{3,n-1} & a_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ a_{n-2,1} & a_{n-2,2} & a_{n-2,3} & \cdots & a_{n-2,n-2} & a_{n-2,n-1} & a_{n-2,n} \\ a_{n-1,1} & a_{n-1,2} & a_{n-1,3} & \cdots & a_{n-1,n-2} & a_{n-1,n-1} & a_{n-1,n} \\ a_{n,1} & a_{n,2} & a_{n,3} & \cdots & a_{n,n-2} & a_{n,n-1} & a_{n,n} \end{pmatrix} =$$

$$\begin{pmatrix} a_{1,1} & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & a_{2,2} & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & a_{3,3} & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & a_{n-2,n-2} & 0 & 0 \\ 0 & 0 & 0 & \cdots & 0 & a_{n-1,n-1} & 0 \\ 0 & 0 & 0 & \cdots & 0 & 0 & a_{n,n} \end{pmatrix} -$$

$$\begin{pmatrix} 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ -a_{2,1} & 0 & 0 & \cdots & 0 & 0 & 0 \\ -a_{3,1} & -a_{3,2} & 0 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ -a_{n-2,1} & -a_{n-2,2} & -a_{n-2,3} & \cdots & 0 & 0 & 0 \\ -a_{n-1,1} & -a_{n-1,2} & -a_{n-1,3} & \cdots & -a_{n-1,n-2} & 0 & 0 \\ -a_{n,1} & -a_{n,2} & -a_{n,3} & \cdots & -a_{n,n-2} & -a_{n,n-1} & 0 \end{pmatrix} -$$

$$\begin{pmatrix} 0 & -a_{1,2} & -a_{1,3} & \cdots & -a_{1,n-2} & -a_{1,n-1} & -a_{1,n} \\ 0 & 0 & -a_{2,3} & \cdots & -a_{2,n-2} & -a_{2,n-1} & -a_{2,n} \\ 0 & 0 & 0 & \cdots & -a_{3,n-2} & -a_{3,n-1} & -a_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & -a_{n-2,n-1} & -a_{n-2,n} \\ 0 & 0 & 0 & \cdots & 0 & 0 & -a_{n-1,n} \\ 0 & 0 & 0 & \cdots & 0 & 0 & 0 \end{pmatrix}$$

Definition (Diagonally Dominant). The matrix A is strictly diagonally dominant if

$$|a_{i,i}| > \sum_{j=1}^{i-1} |a_{i,j}| + \sum_{j=i+1}^{n} |a_{i,j}| \quad \text{for} \quad i = 1, 2, \ldots, n.$$

Theorem (Jacobi Iteration). The solution to the linear system $AX = B$ can be obtained starting with P_0, and using iteration scheme

$$P_{k+1} = M_J P_k + C_J$$

where

$$M_J = D^{-1}(L + U) \quad \text{and} \quad C_J = D^{-1} B.$$

If P_0 is carefully chosen a sequence $\{P_k\}$ is generated which converges to the solution P, i.e. $AP = B$.
A sufficient condition for the method to be applicable is that **A** is strictly diagonally dominant or diagonally dominant and irreducible.

Theorem (Gauss-Seidel Iteration). The solution to the linear system $AX = B$ can be obtained starting with P_0, and using iteration scheme

$$P_{k+1} = M_S P_k + C_S$$

where

$$M_S = (D - L)^{-1} U \quad \text{and} \quad C_S = (D - L)^{-1} B.$$

If P_0 is carefully chosen a sequence $\{P_k\}$ is generated which converges to the solution P, i.e. $AP = B$.

A sufficient condition for the method to be applicable is that **A** is strictly diagonally dominant or diagonally dominant and irreducible.

Theorem (SOR Iteration). Given a value of the parameter ω (chosen in the interval $0 < \omega < 2$), the solution to the linear system $\mathbf{AX} = \mathbf{B}$ can be obtained starting with $\mathbf{P}_0$, and using iteration scheme

$$\mathbf{P}_{k+1} = \mathbf{M}_\omega \mathbf{P}_k + \mathbf{C}_\omega$$

where

$$\mathbf{M}_\omega = (\mathbf{D} - \omega\ \mathbf{L})^{-1}\ ((1 - \omega)\ \mathbf{D} + \omega\ \mathbf{U})$$

and $\mathbf{C}_\omega = \omega\ (\mathbf{D} - \omega\ \mathbf{L})^{-1}\mathbf{B}$.

If $\mathbf{P}_0$ is carefully chosen a sequence $\{\mathbf{P}_k\}$ is generated which converges to the solution **P**, i.e. $\mathbf{AP} = \mathbf{B}$.

Remark. A theorem of Kahan states that the SOR method will converge only if ω is chosen in the interval $0 < \omega < 2$.

Remark. When we choose $\omega = 1$ the SOR method reduces to the Gauss-Seidel method.

Mathematical Subroutine (Jacobi Iteration).

```
Jacobi[A0_, B0_, P0_, max_] :=
 Module[{A = N[A0], B = N[B0], i, j, k = 0, n = Length[P0], P = P0, oldP = P0},
  Print["P"0, " = ", P];
  While[k < max,
   For[i = 1, i ≤ n, i++,
    P[[i]] = 1/A[[i,i]] (B[[i]] - Sum[A[[i,j]] oldP[[j]], {j, 1, i-1}] - Sum[A[[i,j]] oldP[[j]], {j, i+1, n}]) ];
   Print["P"k+1, " = ", P];
   oldP = P;
   k = k + 1; ];
  Return[P]; ];
```

Mathematical Subroutine (Gauss-Seidel Iteration).

```
GaussSeidel[A0_, B0_, P0_, max_] :=
 Module[{A = N[A0], B = N[B0], i, j, k = 0, n = Length[P0], P = P0},
  Print["P"0, " = ", P];
  While[ k < max,
   For[ i = 1, i ≤ n, i++,
    P[[i]] = 1/A[[i,i]] (B[[i]] - Sum[A[[i,j]] P[[j]], {j, 1, i-1}] - Sum[A[[i,j]] P[[j]], {j, i+1, n}]) ];
   Print["P"k+1, " = ", P];
   k = k + 1; ];
  Return[P]; ];
```

Mathematical Subroutine (Successive Over Relaxation).

```
SORmethod[A0_, B0_, P0_, ω_, max_] :=
 Module[{A = N[A0], B = N[B0], i, j, k = 0, n = Length[P0], P = P0, oldP = P0},
  Print["P"0, " = ", P];
  While[ k < max,
   For[ i = 1, i ≤ n, i++,
    P[[i]] = (1 - ω) oldP[[i]] + ω/A[[i,i]] (B[[i]] - Sum[A[[i,j]] P[[j]], {j, 1, i-1}] - Sum[A[[i,j]] oldP[[j]], {j, i+1, n}]) ];
   Print["P"k+1, " = ", P];
   oldP = P;
   k = k + 1; ];
  Return[P]; ];
```

Pivoting Methods:

Background

In the Gauss-Jordan module we saw an algorithm for solving a general linear system of equations $AX = B$ consisting of nequations and n unknowns where it is assumed that the system has a unique solution. The method is attributed

Johann Carl Friedrich Gauss (1777-1855) and

Wilhelm Jordan (1842 to 1899). The following theorem states the sufficient conditions for the existence and uniqueness of solutions of a linear system $AX = B$. **Theorem (Unique Solutions)** Assume that A is an $n \times n$ matrix. The following statements are equivalent.

(i) Given any $n \times 1$ matrix B, the linear system $AX = B$ has a unique solution.

(ii) The matrix A is

nonsingular (i.e., A^{-1} exists).
(iii) The system of equations $AX = 0$ has the unique solution $X = 0$.
(iv) The determinant of A is nonzero, i.e. $\det(A) \neq 0$.

It is convenient to store all the coefficients of the linear system $AX = B$ in one array of dimension $n \times n+1$. The coefficients of B are stored in column $n+1$ of the array (i.e. $a_{i,n+1} = b_i$). Row k contains all the coefficients necessary torepresent the i^{th} equation in the linear system. The augmented matrix is denoted $M = [A \mid B]$ and the linear system is represented as follows:

$$M = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,n} & b_1 \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,n} & b_2 \\ a_{3,1} & a_{3,2} & a_{3,3} & \cdots & a_{3,n} & b_3 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ a_{n,1} & a_{n,2} & a_{n,3} & \cdots & a_{n,n} & b_n \end{pmatrix}$$

The system $AX = B$, with augmented matrix M, can be solved by performing row operations on M. The variables are placeholders for the coefficients and cam be omitted until the end of the computation.

Theorem (Elementary Row Operations). The following operations applied to the augmented matrix M yield an equivalent linear system.

(i) Interchanges: The order of two rows can be interchanged.

(ii)Scaling: Multiplying a row by a nonzero constant.

(iii) Replacement: Row r can be replaced by the sum of that tow and a nonzero multiple of any other row;

that is: $row_r = row_r - c\, row_p$.

It is common practice to implement

(iii) by replacing a row with the difference of that row and a multiple of another row.

Definition (Pivot Element). The number $a_{p,p}$ in the coefficient matrix A that is used to eliminate $a_{i,p}$ where $i = p+1, p+2, \ldots, n$, is called the p^{th} pivot element, and the p^{th} row is called the pivot row.

Theorem (Gaussian Elimination with Back Substitution). Assume that A is an $n \times n$ nonsingular matrix. There exists a unique system $UX = Y$ that is equivalent to the given

system $AX = B$, where U is an upper-triangular matrix with $u_{i,i} \neq 0$ for $i = 1, 2, \ldots, n$. After U and Y are constructed, back substitution can be used to solve $UX = Y$ for X. **ng**

Pivoting Strategies

There are numerous pivoting strategies discussed in theliterature. We mention only a few to give an indication of the possibilities.

(i) No Pivoting. No pivoting means no row interchanges. It can be done only if Gaussian elimination never run into zeros on the diagonal. Since division by zero is a fatal error we usually avoid this pivoting strategy.

Pivoting to Avoid

$a_{p,p} = 0$

If $a_{p,p} = 0$, then row p cannot be used to eliminate the elements in column p below the main diagonal. It isnecessary to find row k, where $a_{k,p} \neq 0$ and k > p, and then interchange row p and row k so that a nonzero pivot element is obtained. This process is called pivoting, and the criterion for deciding which row to choose is called a pivoting strategy. The first idea that comes to mind is the following one.

(ii) Trivial Pivoting. The trivial pivoting strategy is as follows. If $a_{p,p} \neq 0$, do not switch rows. If $a_{p,p} = 0$, locate the first row below p in which $a_{k,p} \neq 0$ and then switch rows k and p. This will result in a new element $a_{p,p} \neq 0$, which is a nonzero pivot element.

Pivoting to Reduce Error

Because the computer uses fixed-precision arithmetic, it is possible that a small error will be introduced each time that an arithmetic operation is performed. The following example illustrates how use of the trivial pivoting strategy in Gaussian elimination can lead to significant error in the solution of a linear system of equations.

(iii) Partial Pivoting. The partial pivoting strategy is as follows. If $a_{p,p} \neq 0$, do not switch rows. If $a_{p,p} = 0$, locate row u below p in which $|a_{u,p}| = \max_{p+1 \le i \le n} |a_{i,p}|$ and $a_{u,p} \neq 0$ and then switch rows u and p. This will result in a new element $a_{p,p} \neq 0$, which is a nonzero pivot element.

Remark. Only row permutations are permitted. The strategy is to switch the largest entry in the pivot column to the diagonal.

(iv) Scaled Partial Pivoting. At the start of the procedurewe compute scale factors for each row of the matrix A as follows:

$$s_i = \max_{1 \le j \le n} |a_{i,j}| \quad \text{for} \quad i = 1, 2, \ldots, n.$$

The scale factors are interchanged with their corresponding row in the elimination steps. The

scaled partial pivoting strategy is as follows. If $a_{p,p} \neq 0$, do not switch rows. If $a_{p,p} = 0$, locate row u below p in which $\frac{|a_{u,p}|}{s_u} = \max_{p+1 \le i \le n} \frac{|a_{i,p}|}{s_i}$ and $a_{u,p} \neq 0$ and then switch rows u and p. This will result in a new element $a_{p,p} \neq 0$, which is a nonzero pivot element.
Remark. Only row permutations are permitted. The strategy is to switch the largest scaled entry in the pivot column to the diagonal.

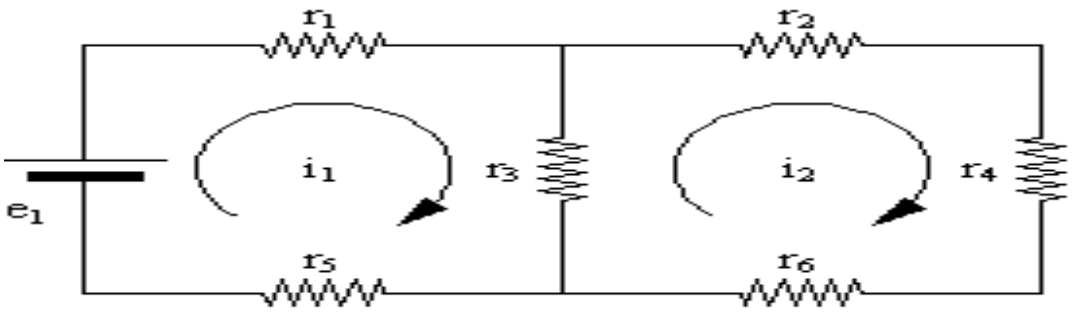
(v) Total Pivoting. The total pivoting strategy is as follows. If $a_{p,p} \neq 0$, do not switch rows. If $a_{p,p} = 0$, locate row u below p and column v to the right of p in which $|a_{u,v}| = \max_{p+1 \le i,j \le n} |a_{i,j}|$ and $a_{u,v} \neq 0$ and then: first switch rows u and p and second switch column v and p. This will result in a new element $a_{p,p} \neq 0$, which is a nonzero pivot element. This is also called "complete pivoting" or "maximal pivoting."
Remark. Both row and column permutations are permitted. The strategy is to switch the largest entry in the part of the matrix that we have not yet processed to the diagonal.

Kirchoff's Law:

Background
Solution of linear systems can be applied to resistornet work circuits. Kirchoff's voltage law says that the sum of the voltage drops around any closed loop in the network must equal zero. A closed loop has the obvious definition: starting at a node, trace a path through the circuit that returns you to the original starting node.

Network #1

Consider the network consisting of six resistors and two battery, shown in the figure below.



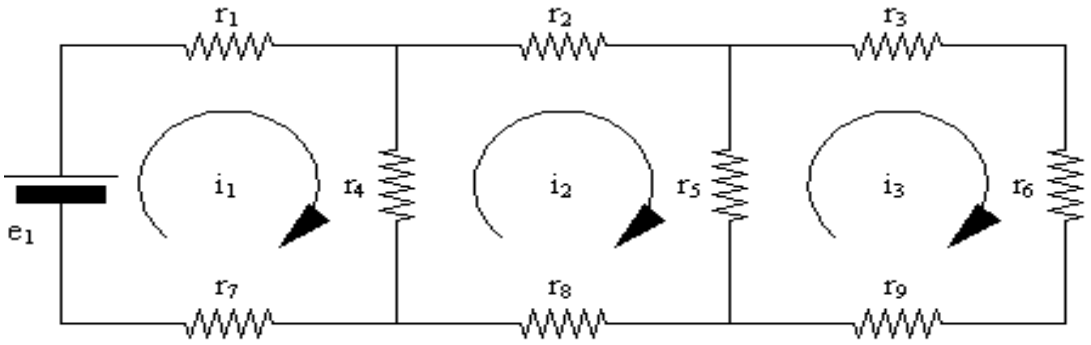
There are two closed loops. When Kirchoff's voltage law is applied, we obtain the following linear system of equations.

$$(r_1 + r_3 + r_5)\, i_1 - r_3\, i_2 = e_1$$

$$-r_3\, i_1 + (r_2 + r_3 + r_4 + r_6)\, i_2 = 0$$

Network #2

Consider the network consisting of nine resistors and one battery, shown in the figure below.



There are three loops. When Kirchoff's voltage law is applied, we obtain the following linear system of equations.

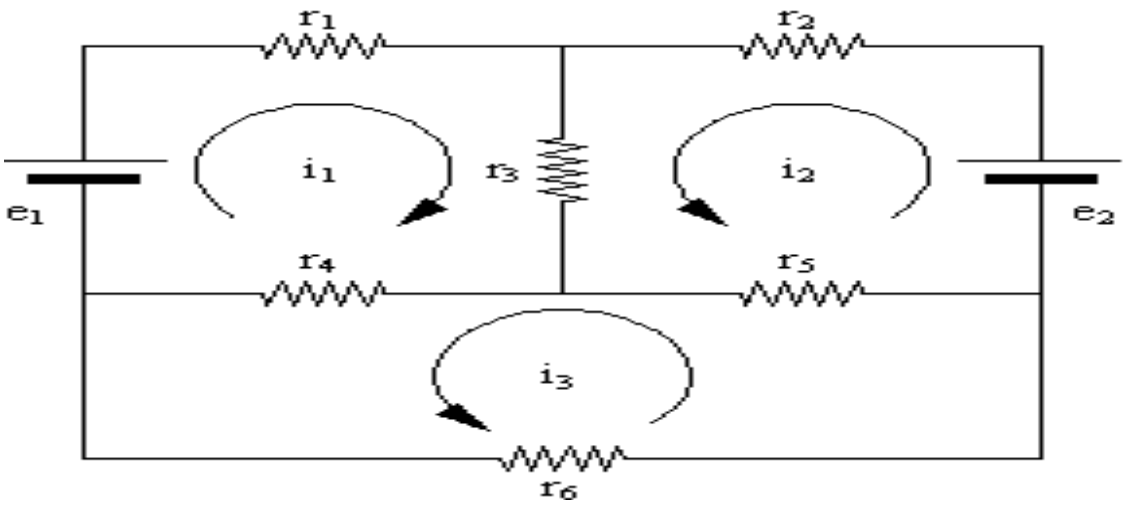
$$(r_1 + r_4 + r_7)\, i_1 - r_4\, i_2 = e_1$$

$$-r_4\, i_1 + (r_2 + r_4 + r_5 + r_8)\, i_2 - r_5\, i_3 = 0$$

$$-r_5\, i_2 + (r_3 + r_5 + r_6 + r_9)\, i_3 = 0$$

Network #3

Consider the network consisting of six resistors and two batteries, shown in the figure below.



There are three loops. When Kirchoff's voltage law is applied, we obtain the following linear system of equations.

$$\begin{aligned}
(r_1 + r_3 + r_4)\, i_1 &+ r_3\, i_2 &+ r_4\, i_3 &= e_1 \\
r_3\, i_1 &+ (r_2 + r_3 + r_5)\, i_2 &- r_5\, i_3 &= e_2 \\
r_4\, i_1 &- r_5\, i_2 &+ (r_4 + r_5 + r_6)\, i_3 &= 0
\end{aligned}$$

Interpolation and Polynomial Approximation:

Lagrange Polynomials:

Background.

We have seen how to expand a function $f(x)$ in a Maclaurin polynomial about $x_0 = 0$ involving the powers x^k and a Taylor polynomial about $x_0 \neq 0$ involving the powers $(x - x_0)^k$. The Lagrange polynomial of degree n passes through the $n+1$ points (x_k, y_k) for $k = 0, 1, \ldots, n$ and were investigated by the mathematician Joseph-Louis Lagrange (1736-1813).

Theorem (Lagrange Polynomial). Assume that $f \in C^{n+1}[a, b]$ and $x_k \in [a, b]$ for $k = 0, 1, \ldots, n$ are distinct values. Then

$$f(x) = P_n(x) + R_n(x),$$

where $P_n(x)$ is a polynomial that can be used toapproximate $f(x)$,

$$P_n(x) = \sum_{k=0}^{n} y_k \frac{(x - x_0) \ldots (x - x_{k-1})(x - x_{k+1}) \ldots (x - x_n)}{(x_k - x_0) \ldots (x_k - x_{k-1})(x_k - x_{k+1}) \ldots (x_k - x_n)}$$

and we write

$$f(x) \approx P_n(x).$$

The Lagrange polynomial goes through the $n+1$ points $\{(x_k, y_k)\}_{k=0}^{n}$, i.e.

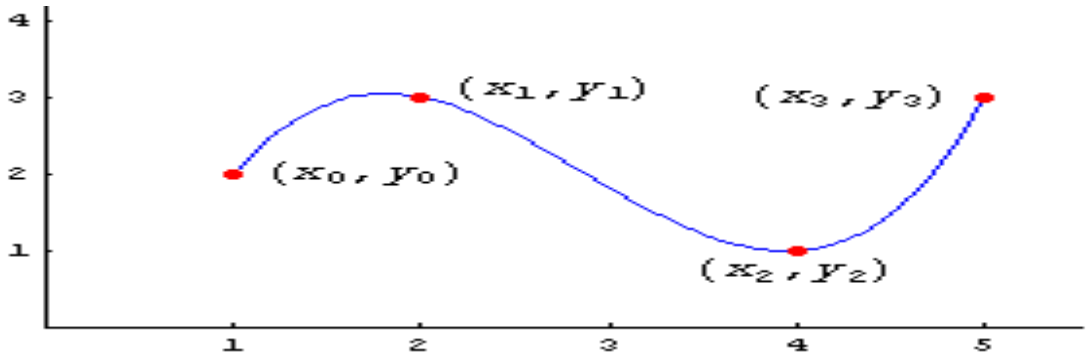
$$P_n(x_k) = f(x_k) \quad \text{for} \quad k = 0, 1, \ldots, n.$$

The remainder term $R_n(x)$ has the form

$$R_n(x) = \frac{f^{(n+1)}(c)}{(n+1)!}(x - x_0)(x - x_1)(x - x_2) \ldots (x - x_{n-1})(x - x_n),$$

for some value $c = c(x)$ that lies in the interval $[a, b]$.

The cubic curve in the figure below illustrates a Lagrange polynomial of degree n = 3, which passes through the four points (x_k, y_k) for $k = 0, 1, 2, 3$.



```
x = 0; y = 0; Clear[p, t, x, y];
p[t_] := y0 (t - x1) (t - x2) (t - x3) / ((x0 - x1) (x0 - x2) (x0 - x3)) + y1 (t - x0) (t - x2) (t - x3) / ((x1 - x0) (x1 - x2) (x1 - x3)) +
    y2 (t - x0) (t - x1) (t - x3) / ((x2 - x0) (x2 - x1) (x2 - x3)) + y3 (t - x0) (t - x1) (t - x2) / ((x3 - x0) (x3 - x1) (x3 - x2));
Print["p[x] = ", p[x]];
Print["p[x0] = ", p[x0], "\n", "\n", "p[x1] = ", p[x1], "\n", "\n", "p[x2] = ", p[x2], "\n", "\n", "p[x3] = ", p[x3]];
```

$$p[x] = \frac{(x-x_1)(x-x_2)(x-x_3)\,y_0}{(x_0-x_1)(x_0-x_2)(x_0-x_3)} + \frac{(x-x_0)(x-x_2)(x-x_3)\,y_1}{(-x_0+x_1)(x_1-x_2)(x_1-x_3)} + \frac{(x-x_0)(x-x_1)(x-x_3)\,y_2}{(-x_0+x_2)(-x_1+x_2)(x_2-x_3)} + \frac{(x-x_0)(x-x_1)(x-x_2)\,y_3}{(-x_0+x_3)(-x_1+x_3)(-x_2+x_3)}$$

$p[x_0] = y_0$

$p[x_1] = y_1$

$p[x_2] = y_2$

$p[x_3] = y_3$

Theorem. (Error Bounds for Lagrange Interpolation, Equally Spaced Nodes) Assume that $f(x)$ defined on $[a, b]$, which contains the equally spaced nodes $x_k = x_0 + kh$. Additionally, assume that $f(x)$ and the derivatives of $f(x)$ up to the order $n+1$ are continuous and bounded on the special subintervals $[x_0, x_1]$, $[x_0, x_2]$, $[x_0, x_3]$, $[x_0, x_4]$, and $[x_0, x_5]$, respectively; that is,

$$|f^{(n+1)}(x)| \le M_{n+1} \text{ for } x_0 < x < x_n,$$

for $n = 1, 2, 3, 4, 5$. The error terms corresponding to these three cases have the following useful bounds on their magnitude

(i). $|R_1(x)| \le \frac{M_2}{8} h^2$ is valid for $x \in [x_0, x_1]$,

(ii). $|R_2(x)| \le \frac{M_3}{9\sqrt{3}} h^3$ is valid for $x \in [x_0, x_2]$,

(iii). $|R_3(x)| \le \frac{M_4}{24} h^4$ is valid for $x \in [x_0, x_3]$,

(iv). $|R_4(x)| \le \frac{\sqrt{4750 + 290\sqrt{145}}}{3000} M_5 h^5$ is valid for $x \in [x_0, x_4]$,

(v). $|R_5(x)| \le \frac{10 + 7\sqrt{7}}{1215} M_6 h^6$ is valid for $x \in [x_0, x_5]$.

Algorithm (Lagrange Polynomial). To construct the Lagrange polynomial

$$P(x) = \sum_{k=0}^{n} y_k L_{n,k}(x)$$

of degree n, based on the $n+1$ points (x_k, y_k) for $k = 0, 1, \ldots, n$. The Lagrange coefficient polynomials $L_{n,k}(x)$ for degree n are:

$$L_{n,k}(x) = \frac{(x - x_0) \ldots (x - x_{k-1})(x - x_{k+1}) \ldots (x - x_n)}{(x_k - x_0) \ldots (x_k - x_{k-1})(x_k - x_{k+1}) \ldots (x_k - x_n)}$$

for $k = 0, 1, \ldots, n$.

You can use the first *MathematicaL* subroutine that does things in the "traditional way" or you are welcome to use the second subroutine that illustrates "Object Oriented Programming."

Mathematica Subroutine (Lagrange Polynomial).Traditional programming.

```
Lagrange[XY_] :=
 Module[{j, k, n, prod, sum, term, X, Y},
  n = Length[XY] - 1;
  X = Transpose[XY][[1]];
  Y = Transpose[XY][[2]];
  sum = 0;
  For[ k = 0, k ≤ n, k++,
   prod = 1;
   For[ j = 0, j ≤ n, j++,
    term = Which[ j == k, 1,
      j ≠ k, (x - X[[j+1]])/(X[[k+1]] - X[[j+1]]) ];
    prod = prod term; ];
   sum = sum + Y[[k+1]] prod; ];
  Return[sum]; ];
```

The above algorithm is sufficient for understanding and/orconstructing the Lagrange polynomial.

Object Oriented Programming. Welcome to the brave new world of "Object Oriented Programming." Use the following *Mathematica* subroutine which is "programmed" using the "mathematical objects" $x_{k\_}, y_{k\_}, \sum_{\square=\square}^{\square} \square$ and $\prod_{\square=\square}^{\square} \square$. Templates for the objects are located by going to "File" then select "Palettes", then select "BasicInput."

Mathematical Subroutine (Lagrange Polynomial).Object oriented programming.

```
Lagrange[XY_] :=
 Module[{j, k, n, X, Y},
  X_k_ := Transpose[XY][[1, k+1]];
  Y_k_ := Transpose[XY][[2, k+1]];
  n = Length[XY] - 1;
  For[ k = 0, k ≤ n, k++,
   L[n, k, x_] = (∏_{j=0}^{k-1} (x - X_j)/(X_k - X_j)) (∏_{j=k+1}^{n} (x - X_j)/(X_k - X_j)); ];
  Return[ ∑_{k=0}^{n} Y_k L[n, k, x] ]; ];
```

Mathematical Subroutine (Lagrange Polynomial).Compact object oriented programming.

```
Lagrange[XY_] :=
 Module[{j, k, n, X, Y},
  X_k_ := Transpose[XY][[1, k+1]];
  Y_k_ := Transpose[XY][[2, k+1]];
  n = Length[XY] - 1;
  Return[ ∑_{k=0}^{n} Y_k (∏_{j=0}^{k-1} (x - X_j)/(X_k - X_j)) (∏_{j=k+1}^{n} (x - X_j)/(X_k - X_j)) ]; ];
```

The Newton Polynomial:

Background.

We have seen how to expand a function $f(x)$ in a Maclaurin polynomial about $x_0 = 0$ involving the powers x^k and a Taylor polynomial about $x_0 \neq 0$ involving the powers $(x - x_0)^k$. These polynomials have a single "center" x_0. Polynomial interpolation can be used to construct the polynomial of degree $\leq n$ that passes through the n+1 points $(x_k, y_k) = (x_k, f(x_k))$, for $k = 0, 1, \cdots, n$. If multiple "centers" $x_0, x_1, \ldots, x_n$ are used, then the result is the so called Newton polynomial. We attribute much of the founding theory to Sir Isaac Newton (1643-1727).

Theorem (Newton Polynomial). Assume that $f \in C^{n+1}[a, b]$ and $x_k \in [a, b]$ for $k = 0, 1, \cdots, n$ are distinct values. Then

$$f(x) = P_n(x) + R_n(x),$$

where $P_n(x)$ is a polynomial that can be used to approximate $f(x)$,

$$P_n(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + a_3(x - x_0)(x - x_1)(x - x_2) + \ldots + a_n(x - x_0)(x - x_1)(x - x_2) \ldots (x - x_{n-1})$$

and we write

$$f(x) \approx P_n(x).$$

The Newton polynomial goes through the $n+1$ points $\{(x_k, y_k)\}_{k=0}^{n}$, i.e.

$$P_n(x_k) = f(x_k) \quad \text{for} \quad k = 0, 1, \ldots, n.$$

The remainder term $R_n(x)$ has the form

$$R_n(x) = \frac{f^{(n+1)}(c)}{(n+1)!}(x - x_0)(x - x_1)(x - x_2) \ldots (x - x_{n-1})(x - x_n),$$

for some value $c = c(x)$ that lies in the interval $[a, b]$. The coefficients a_i are constructed using divided differences.

Definition. Divided Differences.

The divided differences for a function $f[x]$ are defined as follows:

$$f[x_{i-1}, x_i] = \frac{f[x_i] - f[x_{i-1}]}{x_i - x_{i-1}}$$

$$f[x_{i-2}, x_{i-1}, x_i] = \frac{f[x_{i-1}, x_i] - f[x_{i-2}, x_{i-1}]}{x_i - x_{i-2}}$$

$$f[x_{i-3}, x_{i-2}, x_{i-1}, x_i] = \frac{f[x_{i-2}, x_{i-1}, x_i] - f[x_{i-3}, x_{i-2}, x_{i-1}]}{x_i - x_{i-3}}$$

$$f[x_{i-j}, x_{i-j+1}, \ldots, x_i] = \frac{f[x_{i-j+1}, \ldots, x_i] - f[x_{i-j}, \ldots, x_{i-1}]}{x_i - x_{i-j}}$$

The divided difference formulae are used to construct the divided difference table:

x_i $f[x_i]$ $f[x_{i-1}, x_i]$ $f[x_{i-2}, x_{i-1}, x_i]$ $f[x_{i-3}, x_{i-2}, x_{i-1}, x_i]$ $f[x_{i-4}, x_{i-3}, x_{i-2}, x_{i-1}, x_i]$ x_0 x_1
x_2 x_3 x_4 $f[x_0]$ $f[x_1]$ $f[x_2]$ $f[x_3]$ $f[x_4]$ $f[x_0, x_1]$ $f[x_1, x_2]$ $f[x_2, x_3]$ $f[x_3, x_4]$

$f[x_0, x_1, x_2]$

$f[x_1, x_2, x_3]$ $f[x_0, x_1, x_2, x_3]$

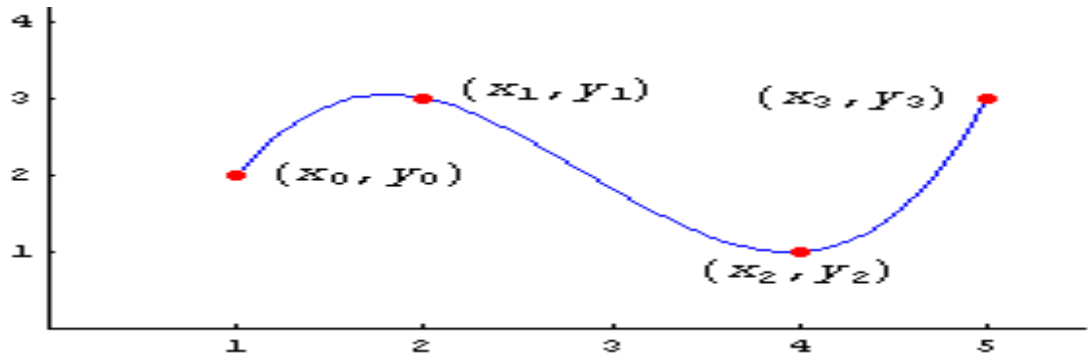
$f[x_2, x_3, x_4]$ $f[x_1, x_2, x_3, x_4]$ $f[x_0, x_1, x_2, x_3, x_4]$

The coefficient a_i of the Newton polynomial $P_n(x)$ is $a_i = f[x_0, x_1, \ldots, x_i]$ and it is the top element in the column of the i-th divided differences.

The Newton polynomial of degree $\leq n$ that passes through the $n+1$ points $(x_k, y_k) = (x_k, f(x_k))$, for $k = 0, 1, \cdots, n$ is

$$P_n(x) = a_0 + a_1 (x - x_0) + a_2 (x - x_0)(x - x_1) + a_3 (x - x_0)(x - x_1)(x - x_2) + \ldots + a_n (x - x_0)(x - x_1)(x - x_2) \ldots (x - x_{n-1})$$

The cubic curve in the figure below illustrates a Newton polynomial of degree n = 3.



```
Clear[f, p, t, x, y, z];
f[x_, y___, z_] := Module[{}, Return[(f[y, z] - f[x, y])/(z - x)]]
p[t_] = f[x0] + f[x0, x1] (t - x0) +
   f[x0, x1, x2] (t - x0) (t - x1) +
   f[x0, x1, x2, x3] (t - x0) (t - x1) (t - x2);
Print["p[x] = ", p[x]];
Print["p[x0] = ", p[x0]];
Print["p[x1] = ", p[x1]];
Print["p[x2] = ", p[x2]];
Print["p[x2] = ", Together[p[x2]]];
Print["p[x3] = ", p[x3]];
Print["p[x3] = ", Together[p[x3]] ];
```

$$p[x] = f[x_0] + \frac{(-f[x_0] + f[x_1])\,(x - x_0)}{-x_0 + x_1} + \frac{(x - x_0)\,(x - x_1)\left(-\frac{-f[x_0]+f[x_1]}{-x_0+x_1} + \frac{-f[x_1]+f[x_2]}{-x_1+x_2}\right)}{-x_0 + x_2} + \frac{(x - x_0)\,(x - x_1)\,(x - x_2)\left(-\frac{-\frac{-f[x_0]+f[x_1]}{-x_0+x_1} + \frac{-f[x_1]+f[x_2]}{-x_1+x_2}}{-x_0+x_2} + \frac{-\frac{-f[x_1]+f[x_2]}{-x_1+x_2} + \frac{-f[x_2]+f[x_3]}{-x_2+x_3}}{-x_1+x_3}\right)}{-x_0 + x_3}$$

$$p[x_0] = f[x_0] \quad p[x_1] = f[x_1]$$

$$p[x_2] = f[x_0] + \frac{(-f[x_0] + f[x_1])\,(-x_0 + x_2)}{-x_0 + x_1} + (-x_1 + x_2)\left(-\frac{-f[x_0] + f[x_1]}{-x_0 + x_1} + \frac{-f[x_1] + f[x_2]}{-x_1 + x_2}\right)$$

$$p[x_2] = f[x_2]$$

$$p[x_3] = f[x_0] + \frac{(-f[x_0] + f[x_1])\,(-x_0 + x_3)}{-x_0 + x_1} + \frac{\left(-\frac{-f[x_0]+f[x_1]}{-x_0+x_1} + \frac{-f[x_1]+f[x_2]}{-x_1+x_2}\right)(-x_0 + x_3)\,(-x_1 + x_3)}{-x_0 + x_2} + (-x_1 + x_3)\,(-x_2 + x_3)\left(-\frac{-\frac{-f[x_0]+f[x_1]}{-x_0+x_1} + \frac{-f[x_1]+f[x_2]}{-x_1+x_2}}{-x_0+x_2} + \frac{-\frac{-f[x_1]+f[x_2]}{-x_1+x_2} + \frac{-f[x_2]+f[x_3]}{-x_2+x_3}}{-x_1+x_3}\right)$$

$$p[x_3] = f[x_3]$$

Theorem. (Error Bounds for Newton Interpolation, Equally Spaced Nodes) Assume that $f(x)$ defined on $[a, b]$, which contains the equally spaced nodes $x_k = x_0 + kh$. Additionally, assume that $f(x)$ and the derivatives of $f(x)$ up to the order $n+1$ arecontinuous and bounded on the special subintervals $[x_0, x_1]$, $[x_0, x_2]$, $[x_0, x_3]$, $[x_0, x_4]$, and $[x_0, x_5]$, respectively; that is,

$$\left| f^{(n+1)}(x) \right| \leq M_{n+1} \quad \text{for} \quad x_0 \prec x \prec x_n,$$

for $n = 1, 2, 3, 4, 5$. The error terms corresponding to these three cases have the following useful bounds on their magnitude

(i). $|R_1(x)| \leq \frac{M_2}{8} h^2$ is valid for $x \in [x_0, x_1]$,

(ii). $|R_2(x)| \leq \frac{M_3}{9\sqrt{3}} h^3$ is valid for $x \in [x_0, x_2]$,

(iii). $|R_3(x)| \le \frac{M_4}{24} h^4$ is valid for $x \in [x_0, x_3]$,

(iv). $|R_4(x)| \le \frac{\sqrt{4750 + 290\sqrt{145}}}{3000} M_5 h^5$ is valid for $x \in [x_0, x_4]$,

(v). $|R_5(x)| \le \frac{10 + 7\sqrt{7}}{1215} M_6 h^6$ is valid for $x \in [x_0, x_5]$.

Algorithm (Newton Interpolation Polynomial). To constructand evaluate the Newton polynomial of degree $\le n$ that passes through the n+1 points $(x_i, y_i) = (x_i, f(x_i))$, for $i = 0, 1, \cdots, n$

$$P_n(x) = d_{0,0} + d_{1,1}(x - x_0) + d_{2,2}(x - x_0)(x - x_1) + d_{3,3}(x - x_0)(x - x_1)(x - x_2) + \ldots + d_{n,n}(x - x_0)(x - x_1)(x - x_2) \ldots (x - x_{n-1})$$

where

$d_{i,0} = y_i$ for $i = 0, 1, \ldots, n$

and

$$d_{i,j} = \frac{d_{i,j-1} - d_{i-1,j-1}}{x_i - x_{i-j}} \quad \text{for } i = 1, 2, \ldots, n \text{ and } j = 1, 2, \ldots, i$$

Remark 1. Newton polynomials are created "recursively."

$$P_n(x) = P_{n-1}(x) + d_{n,n}(x - x_0)(x - x_1)(x - x_2) \ldots (x - x_{n-1}).$$

Remark 2. *Mathematica*'s arrays are lists and the subscripts must start with 1 instead of 0.

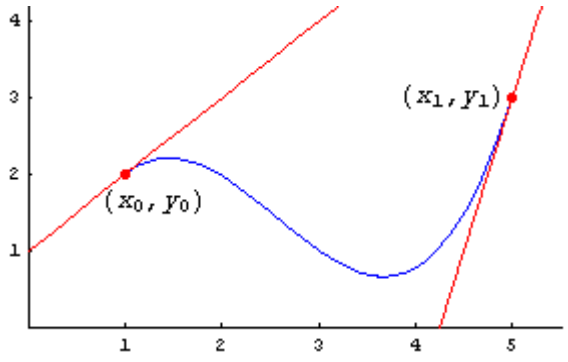
Mathematical Subroutine (Newton Polynomial).

```
NewtonPoly[XY_] :=
 Module[{j, i, n, X, Y},
  X = Transpose[XY][[1]];
  Y = Transpose[XY][[2]];
  n = Length[XY] - 1;
  d = Table["", {n + 1}, {n + 1}];
  d[[All,1]] = Y[[All]];
  For[ j = 1, j ≤ n, j++,
   For[ i = j, i ≤ n, i++,
    d[[i+1,j+1]] = (d[[i+1,j]] - d[[i,j]]) / (X[[i+1]] - X[[i+1-j]]) ]; ];
  For[ i = 0, i ≤ n, i++,
   p[i + 1, x_] = Product_{j=1}^{i} (x - X[[j]]) ];
  Return[ Sum_{i=0}^{n} d[[i+1,i+1]] p[i + 1, x] ]; ]
```

Hermite Polynomial Interpolation:

Background for the Hermite Polynomial

The cubic Hermite polynomial p(x) has the interpolative properties $p(x_0) = y_0$, $p(x_1) = y_1$, $p'(x_0) = d_0$, and $p'(x_1) = d_1$, both the function values and their derivativesare known at the endpoints of the interval $[x_0, x_1]$. Hermite polynomials were studied by the FrenchMathematician Charles Hermite (1822-1901), and are referred to as a "clamped cubic," where "clamped" refers to the slope at the endpoints being fixed. This situation isillustrated in the figure below.



Theorem (Cubic Hermite Polynomial). If $f[x]$ is continuous on the interval $[x_0, x_1]$, there exists a unique cubic polynomial $p[x] = ax^3 + bx^2 + cx + d$ such that

$$p[x_0] = f[x_0],$$
$$p[x_1] = f[x_1],$$
$$p'[x_0] = f'[x_0],$$
$$p'[x_2] = f'[x_2].$$

Remark. The cubic Hermite polynomial is a generalization of both the Taylor polynomial and Lagrange polynomial, and it is referred to as an "osculating polynomial." Hermite polynomials can be generalized to higher degrees by requiring that the use of more nodes $\{x_0, x_1, \ldots, x_n\}$ and the extension to agreement at higher derivatives $p^{(k)}[x_i] = f^{(k)}[x_i]$ for $i = 1, 2, \ldots, n$ and $k = 1, 2, \ldots, m_i$. The details are found in advanced texts on numerical analysis

More Background. The Clamped Cubic Spline

A clamped cubic spline is obtained by forming a piecewise cubic function which passes through the given set of knots $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$ with the condition the function values, their derivatives and second derivatives ofadjacent cubics agree at the interior nodes. The endpoint conditions are $S'(x_0) = d_0$ and $S'(x_n) = d_n$, where d_0 and d_n are given.

More Background. The Natural Cubic Spline

A natural cubic spline is obtained by forming a piecewise cubic function which passes through the given set of knots $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$ with the condition the function values, their derivatives and second derivatives ofadjacent cubics agree at the interior nodes. The endpoint conditions are $S''(x_0) = 0$ and $S''(x_n) = 0$. The natural cubic spline is said to be "a relaxed curve."

Cubic Splines:

Cubic Spline Interpolant

Definition (Cubic Spline). Suppose that $\{(x_k, y_k)\}_{k=0}^{n}$ are n+1 points, where $a = x_0 < x_0 < \ldots < x_n = b$. The function $S(x)$ is called a

cubic spline if there exists n cubic polynomials $S_k(x)$ with coefficients $s_{k,0}, s_{k,1}, s_{k,2}$, and $s_{k,3}$ that satisfy the properties:

I. $S(x) = S_k(x)$

$= s_{k,0} + s_{k,1}(x - x_k) + s_{k,2}(x - x_k)^2 + s_{k,3}(x - x_k)^3$

for $x \in [x_k, x_{k+1}]$ and $k = 0, 1, \ldots, n-1$.

II. $S(x_k) = y_k$ for $k = 0, 1, \ldots, n$.

The spline passes through each data point.

III. $S_k(x_{k+1}) = S_{k+1}(x_{k+1})$ for $k = 0, 1, \ldots, n-2$.

The spline forms a continuous function over [a,b].

IV. $S_k'(x_{k+1}) = S_{k+1}'(x_{k+1})$ for $k = 0, 1, \ldots, n-2$.

The spline forms a smooth function.

IV. $S_k''(x_{k+1}) = S_{k+1}''(x_{k+1})$ for $k = 0, 1, \ldots, n-2$.

The second derivative is continuous.

Lemma (Natural Spline). There exists a unique cubic spline with the free boundary conditions $S''(a) = 0$ and $S''(b) = 0$.

Remark. The natural spline is the curve obtained by forcing a flexible elastic rod through the points but letting the slope at the ends be free to equilibrate to the position that minimizes the oscillatory behavior of the curve. It is useful for fitting a curve to experimental data that is significant to several significant digits.

Program (Natural Cubic Spline). To construct and evaluate the cubic spline interpolant $S(x)$ for the $n+1$ data points $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$, using the freeboundary conditions $S''(a) = 0$ and $S''(b) = 0$.

Mathematical Subroutine (Natural Cubic Spline).

```
NaturalSpline[XY0_] := Module[{XY = XY0},
  Differences := Module[{k},
    n = Length[XY] - 1;
    X = Transpose[XY][[1]];
    Y = Transpose[XY][[2]];
    h = d = Table[0, {n}];
    m = Table[0, {n + 1}];
    a = b = c = v = Table[0, {n - 1}];
    s = Table[0, {n}, {4}];
    h[[1]] = X[[2]] - X[[1]];
    d[[1]] = (Y[[2]] - Y[[1]])/h[[1]];
    For[k = 2, k ≤ n, k++,
     h[[k]] = X[[k+1]] - X[[k]];
     d[[k]] = (Y[[k+1]] - Y[[k]])/h[[k]];
     a[[k-1]] = h[[k]];
     b[[k-1]] = 2 (h[[k-1]] + h[[k]]);
     c[[k-1]] = h[[k]];
     v[[k-1]] = 6 (d[[k]] - d[[k-1]])]; ];
TriDiagonal := Module[{k, t},
    m[[1]] = 0;
    m[[n+1]] = 0;
    For[k = 2, k ≤ n - 1, k++,
     t = a[[k-1]]/b[[k-1]];
     b[[k]] = b[[k]] - t c[[k-1]];
     v[[k]] = v[[k]] - t v[[k-1]]; ];
    m[[n]] = v[[n-1]]/b[[n-1]];
    For[k = n - 2, 1 ≤ k, k--,
     m[[k+1]] = (v[[k]] - c[[k]] m[[k+2]])/b[[k]]; ]; ];
ComputeCoeff := Module[{k},
    For[k = 1, k ≤ n, k++,
     s[[k,1]] = Y[[k]];
     s[[k,2]] = d[[k]] - 1/6 h[[k]] (2 m[[k]] + m[[k+1]]);
     s[[k,3]] = m[[k]]/2;
     s[[k,4]] = (m[[k+1]] - m[[k]])/(6 h[[k]]); ]; ];
CS[t_] := Module[{j},
    For[j = 1, j ≤ n, j++,
     If[X[[j]] ≤ t && t < X[[j+1]], k = j ]; ];
    If[t < X[[1]], k = 1 ];
    If[X[[n+1]] ≤ t, k = n ];
    w = t - X[[k]];
```

Remark. There are five popular types of splines: natural spline, clamped spline, extrapolated spline, parabolically terminated spline, endpoint curvature adjusted spline.
When *Mathematica* constructs a cubic spline it uses the "natural cubic spline."

Clamped Spline.

Lemma (Clamped Spline). There exists a unique cubic spline with the first derivative boundary conditions $S'(a) = d_0$ and $S'(b) = d_n$.

A property of clamped cubic splines.
A practical feature of splines is the minimum of the oscillatory behavior they possess. Consequently, among all functions f(x) which are twice continuously differentiable on [a,b] and interpolate a given set data points $\{(x_k, f(x_k))\}_{k=0}^{n}$, the cubic spline has "less wiggle." The next result explains this phenomenon.

Theorem (Minimum property of clamped cubic splines). Assume that $f \in C^2[a, b]$ and $S(x)$ is the unique clamped cubic spline interpolant for $f(x)$ which passes through $\{(x_k, f(x_k))\}_{k=0}^{n}$ and satisfies the clamped end conditions $S'(a) = f'(a)$ and $S'(b) = f'(b)$. Then

$$\int_a^b (S''(x))^2 dx \leq \int_a^b (f''(x))^2 dx .$$

Program (Clamped Cubic Spline). To construct and evaluate the cubic spline interpolant S(x) for the n+1 data points $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$, using thefirst derivative boundary conditions $S'(a) = d_0$ and $S'(b) = d_n$.

Curve Fitting:

Least Squares Lines:

Background
The formulas for linear least squares fitting were independently derived by German mathematician Johann Carl Friedrich Gauss (1777-1855) and the French mathematician Adrien-Marie Legendre (1752-1833).

Theorem (Least Squares Line Fitting). Given the n data points $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, the least squares line $y = ax + b$ that fits the points has coefficients a and b given by:

$$a = \frac{\left(\sum_{k=1}^{n} x_k\right)\left(\sum_{k=1}^{n} y_k\right) - n\sum_{k=1}^{n} x_k y_k}{\left(\sum_{k=1}^{n} x_k\right)^2 - n\sum_{k=1}^{n} x_k^2}$$

and

$$b = \frac{\left(\sum_{k=1}^{n} x_k\right)\left(\sum_{k=1}^{n} x_k y_k\right) - \sum_{k=1}^{n} x_k^2 \sum_{k=1}^{n} y_k}{\left(\sum_{k=1}^{n} x_k\right)^2 - n\sum_{k=1}^{n} x_k^2}$$

.

Remark. The least squares line is often times called the line of regression.

Mathematical Subroutine (Least Squares Line).

```
Regression[XY0_] := Module[ {k, n, XY = XY0 },
  n = Length[XY];
  X = Transpose[XY][[1]];
  Y = Transpose[XY][[2]];
  A = ( Sum[X[[k]]^2, {k, 1, n}]   Sum[X[[k]], {k, 1, n}] )
      ( Sum[X[[k]], {k, 1, n}]     n                      );
  B = ( Sum[X[[k]] Y[[k]], {k, 1, n}] )
      ( Sum[Y[[k]], {k, 1, n}]        );
  c = LinearSolve[A, B];
  a = c[[1,1]];
  b = c[[2,1]];
  E2 = Sum[(Y[[k]] - a X[[k]] - b)^2, {k, 1, n}];
  Return[ a x + b ]; ];
```

Philosophy. What comes first the chicken or the egg ? Which coordinate is more sacred, the abscissas or the ordinates. We are always free to choose which variable is independent when we graph a line; $y = ax + b$ or $x = cy + d$. When you realize that two different "least squares lines" can be produced we are amazed. What should we do ? Which line should we use ? You must decide a priori which variable is independent and which is dependent and then proceed. Exercise 3 asked you to think about the mathematics that is involved with this "paradox."

Another "Fit"

Theorem (Power Fit). Given the n data points $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, the power curve $y = ax^m$ that fits the points has coefficients a given by:

$$a = \sum_{k=1}^{n} x_k^m y_k \Big/ \sum_{k=1}^{n} x_k^{2m}$$

Remark. The case m = 1 is a line that passes through the origin.

Mathematical Subroutine (Power Curve).

```
PowerCurve[XY0_, m_] := Module[ {k, n, XY = XY0 },
  n = Length[XY];
  X = Transpose[XY][[1]];
  Y = Transpose[XY][[2]];
  a = Sum[(X[[k]])^m Y[[k]], {k, 1, n}] / Sum[(X[[k]])^(2 m), {k, 1, n}];
  Return[ a x^m]; ];
```

Least Squares Polynomials:

Theorem (Least-Squares Polynomial Curve Fitting). Given the n data points $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, the least squares polynomial of degree m of the form

$$P_m(x) = c_1 + c_2 x + c_3 x^2 + \ldots + c_m x^{m-1} + c_{m+1} x^m$$

that fits the n data points is obtained by solving the following linear system

$$\begin{pmatrix} n & \sum_{i=1}^n x_i & \sum_{i=1}^n x_i^2 & \cdots & \sum_{i=1}^n x_i^m \\ \sum_{i=1}^n x_i & \sum_{i=1}^n x_i^2 & \sum_{i=1}^n x_i^3 & \cdots & \sum_{i=1}^n x_i^{m+1} \\ \sum_{i=1}^n x_i^2 & \sum_{i=1}^n x_i^3 & \sum_{i=1}^n x_i^4 & \cdots & \sum_{i=1}^n x_i^{m+2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \sum_{i=1}^n x_i^m & \sum_{i=1}^n x_i^{m+1} & \sum_{i=1}^n x_i^{m+2} & \cdots & \sum_{i=1}^n x_i^{2m} \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_{m+1} \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^n y_i \\ \sum_{i=1}^n x_i y_i \\ \sum_{i=1}^n x_i^2 y_i \\ \vdots \\ \sum_{i=1}^n x_i^m y_i \end{pmatrix}$$

for the m+1 coefficients $\{c_1, c_2, \ldots, c_m, c_{m+1}\}$. These equationsare referred to as the "normal equations".

One thing is certain, to find the least squares polynomial the above linear system must be solved. There are various linear system solvers that could be used for this task. However, since this is such an important computation, most mathematical software programs have a built-in subroutine for this purpose. In *Mathematica* it is called the "**Fit**" procedure. **Fit[data, funs,**

vars] finds a leastsquares fit to a list of data as a linear combination of the functions **funs** of variablesvars.

We will check the "closeness of fit" with the **Root Mean Square or RMS** measure for the "error in the fit."

```
RMS[XY0_] := Module[ {k, n, X, Y, XY = XY0 },
  n = Length[XY];
  X = Transpose[XY][[1]];
  Y = Transpose[XY][[2]];
  Return[ Sqrt[ 1.0/n Sum[(Y[[k]] - f[X[[k]]])^2, {k, 1, n}] ]; ];
```

Mathematical Subroutine (Least Squares Parabola).

```
LSParabola[XY0_] := Module[ {k, n, XY = XY0 },
  n = Length[XY];
  X = Transpose[XY][[1]];
  Y = Transpose[XY][[2]];
  A = {{n, Sum[X[[k]], {k, 1, n}], Sum[X[[k]]^2, {k, 1, n}]},
       {Sum[X[[k]], {k, 1, n}], Sum[X[[k]]^2, {k, 1, n}], Sum[X[[k]]^3, {k, 1, n}]},
       {Sum[X[[k]]^2, {k, 1, n}], Sum[X[[k]]^3, {k, 1, n}], Sum[X[[k]]^4, {k, 1, n}]}};
  B = {{Sum[Y[[k]], {k, 1, n}]},
       {Sum[X[[k]] Y[[k]], {k, 1, n}]},
       {Sum[(X[[k]])^2 Y[[k]], {k, 1, n}]}};
  Z = LinearSolve[A, B];
  a = Z[[1,1]];
  b = Z[[2,1]];
  c = Z[[3,1]];
  E2 = Sqrt[ 1.0/n Sum[(Y[[k]] - a - b X[[k]] - c (X[[k]])^2)^2, {k, 1, n}] ];
  Return[ a + b x + c x^2 ]; ];
```

Caution for polynomial curve fitting.
Something goes radically wrong if the data is radically "NOT polynomial." This phenomenon is called "polynomial wiggle." The next example illustrates this concept.

Linear Least Squares
The linear least-squares problem is stated as follows. Suppose that n data points $\{(x_i, y_i)\}_{i=1}^{n}$

and a set of m linearly independent functions $\{f_j[x]\}_{j=1}^{m}$ are given. We want to fine m coefficients $\{c_j\}_{j=1}^{m}$ so that the function $f[x]$ given by the linear combination

$$f[x] = \sum_{j=1}^{m} c_j f_j[x]$$

will minimize the sum of the squares of the errors

$$E[c_1, c_2, \ldots, c_m] = \sum_{i=1}^{n} (f[x_i] - y_i)^2 = \sum_{i=1}^{n} \left(\sum_{j=1}^{m} c_j f_j[x_i] - y_i \right)^2 .$$

Theorem (Linear Least Squares). The solution to the linear least squares problem is found by creating the matrix $\mathbf{F}$ whose elements are $F_{i,j} = f_j[x_i]$

$$\mathbf{F} = \{F_{i,j}\}$$

The coefficients $\{c_j\}_{j=1}^{m}$ are found by solving the linear system

$$\mathbf{F}^T \mathbf{F}\, \mathbf{C} = \mathbf{F}^T \mathbf{Y}$$

where $\mathbf{C} = \text{Transpose}[\{c_1, c_2, \ldots, c_m\}]$ and $\mathbf{Y} = \text{Transpose}[\{y_1, y_2, \ldots, y_n\}]$

Nonlinear Curve Fitting:

Data Linearization Method for Exponential CurveFitting.

Fit the curve $y = c e^{ax}$ **to the data points** $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$.

Taking the logarithm of both sides we obtain $\ln(y) = \ln(c e^{ax}) = \ln(c) + \ln(e^{ax}) = \ln(c) + ax$, **thus**

$$\ln(y) = ax + \ln(c) .$$

Introduce the change of variable $X = x$ and $Y = \ln(y)$. **Then the previous equation becomes**

$$Y = aX + \ln(c)$$

which is a linear equation in the variables X and Y.

Use the change of variables $X = x$ and $Y = \ln(y)$ **on all the data points and obtain**

$X_k = x_k$ and $Y_k = \ln(y_k)$ **for** $k = 1, 2, \ldots, n$.

Fit the points $(X_1, Y_1), (X_2, Y_2), \ldots, (X_n, Y_n)$ **with a "least squares line" of the form** $Y = AX + B$.

Comparing the equations $Y = AX + B$ and $Y = aX + \ln(c)$ **we see that** $A = a$ and $B = \ln(c)$ **. Thus**

$a = A$ and $c = e^B$

are used to construct the coefficients which are then used to "fit the curve"

$y = c e^{ax}$

to the given data points $(x_1, Y_1), (x_2, Y_2), \ldots, (x_n, Y_n)$ **in the xy-plane.**

Data Linearization Method for a Power Function CurveFitting.

Fit the curve $Y = c x^a$ **to the data points** $(x_1, Y_1), (x_2, Y_2), \ldots, (x_n, Y_n)$.

Taking the logarithm of both sides we obtain $\ln(y) = \ln(c x^a) = \ln(c) + \ln(x^a) = \ln(c) + a \ln(x)$ **, thus**

$\ln(y) = a \ln(x) + \ln(c)$.

Introduce the change of variable $X = \ln(x)$ and $Y = \ln(y)$. **Then the previous equation becomes**

$Y = aX + \ln(c)$

which is a linear equation in the variables X and Y.

Use the change of variables $X = \ln(x)$ and $Y = \ln(y)$ **on all the data points and obtain**

$X_k = \ln(x_k)$ and $Y_k = \ln(y_k)$ **for** $k = 1, 2, \ldots, n$.

Fit the points $(X_1, Y_1), (X_2, Y_2), \ldots, (X_n, Y_n)$ **with a "least squares line" of the form** $Y = AX + B$.

Comparing the equations $Y = AX + B$ and $Y = aX + \ln(c)$ **we see that** $A = a$ and $B = \ln(c)$ **. Thus**

$a = A$ and $c = e^B$

are used to construct the coefficients which are then used to "fit the curve"

$y = c\,x^a$

to the given data points $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ **in the xy-plane.**

Logistic Curve Fitting:

Background for the Logistic Curve Fitting.

Fit the curve $y = f_1[x] = \frac{L}{1 + c\,e^{a\,x}}$ to the data points $(x_1, y_1), (x_1, y_1), \ldots, (x_n, y_n)$.

Rearrange the terms $\frac{L}{y} - 1 = c\,e^{a\,x}$. Then take thelogarithm of both sides:

$$\ln\left(\frac{L}{y} - 1\right) = \ln\left(c\,e^{a\,x}\right) = \ln(c) + a\,x$$

.

Introduce the change of variables: $X = x$ and $Y = \ln\left(\frac{L}{y} - 1\right)$. The previous equationbecomes

$Y = \ln(c) + a\,X$ which is now "linearized."

Use this change of variables on the data points $X_k = x_k$ and $Y_k = \ln\left(\frac{L}{y_k} - 1\right)$, i.e. same abscissa's but transformed ordinates.

Now you have transformed data points: $(X_1, Y_1), (X_1, Y_1), \ldots, (X_n, Y_n)$.
Use the "Fit" procedure get Y = A X + B, which must match the form $Y = \ln(c) + a\,X$, hence we must have $c = e^B$ and a = A.

Remark. For the method of "data linearization" we must know the constant L in advance. Since L is the "limiting population" for the "S" shaped logistic curve, a value of L that is appropriate to the problem at hand can usually be obtained by guessing.

Example. Use the method of "data linearization" to find thelogistic curve that fits the data for the population of the U.S. for the years 1900-1990. Fit the curve $y = f_1[x] = \frac{L}{1 + c\,e^{a\,x}}$ to the census data for the population of the U.S.

Date	Populatlion
1900 July	76094000
1910 July	92407000
1920 July	106461000
1930 July	123076741
1940 July	132122446
1950 July	152271417
1960 July	180671158
1970 July	205052174
1980 July	227224681
1990 July	249464396

Fast Fourier Transform (FFT):

Definition (Piecewise Continuous). The function $f(x)$ is piecewise continuous on the closed interval $[a, b]$, if there exists values $x_0, x_1, \ldots, x_n$ with $a = x_0 < x_1 < \ldots < x_n = b$ such that f is continuous in each of the open intervals $x_{k-1} < x < x_k$, for $k = 1, 2, \ldots, n$ and has left-hand and right-hand limits at each of the values x_k, for $k = 0, 1, 2, \ldots, n$.

Definition (Fourier Series). If $f(x)$ is periodic with period $2L$ and is piecewise continuous on $[0, 2L]$, then the Fourier Series $S(x)$ for $f(x)$ is

$$S(x) = \frac{a_0}{2} + \sum_{j=1}^{\infty} \left(a_j \cos\left(j \frac{\pi}{L} x\right) + b_j \sin\left(j \frac{\pi}{L} x\right) \right),$$

where the coefficients a_j and b_j are given by the so-calledEuler's formulae:

$$a_j = \frac{1}{L} \int_0^{2L} f(x) \cos\left(j \frac{\pi}{L} x\right) dx \quad \text{for } j = 0, 1, \ldots,$$

and

$$b_j = \frac{1}{L} \int_0^{2L} f(x) \sin\left(j \frac{\pi}{L} x\right) dx \quad \text{for } j = 1, 2, \ldots.$$

Theorem (Fourier Expansion). Assume that $S(x)$ is the Fourier Series for $f(x)$. If $f(x)$ and $f'(x)$ are piecewise continuous on $[0, 2L]$, then $S(x)$ is convergent for all $x \in [0, 2L]$.

The relation $f(x) = S(x)$ holds for all $x \in [0, 2L]$ where $f(x)$ is continuous. If $x = a$ is a point ofdiscontinuity of $f(x)$, then

$$S(a) = \frac{f(a^-) + f(a^+)}{2},$$

where $f(a^-)$ and $f(a^+)$ denote the left-hand and right-hand limits, respectively. With this understanding, we have the Fourier Series expansion:

$$f(x) = \frac{a_0}{2} + \sum_{j=1}^{\infty} \left(a_j \cos\left(j \frac{\pi}{L} x\right) + b_j \sin\left(j \frac{\pi}{L} x\right) \right).$$

Definition (Fourier Polynomial). If $f(x)$ is periodic with period $2L$ and is piecewise continuous on $[0, 2L]$, then the Fourier Polynomial $S(x)$ for $f(x)$ of degree m is

$$S(x) = \frac{a_0}{2} + \sum_{j=1}^{m} \left(a_j \cos\left(j \frac{\pi}{L} x\right) + b_j \sin\left(j \frac{\pi}{L} x\right) \right),$$

where the coefficients a_j and b_j are given by the so-calledEuler's formulae:

$$a_j = \frac{1}{L} \int_0^{2L} f(x) \cos\left(j \frac{\pi}{L} x\right) dx \quad \text{for } j = 0, 1, \ldots, m,$$

and

$$b_j = \frac{1}{L} \int_0^{2L} f(x) \sin\left(j \frac{\pi}{L} x\right) dx \quad \text{for } j = 1, 2, \ldots, m.$$

Numerical Integration calculation for the Fourier trigonometric polynomial**.**

Assume that $f(x)$ is periodic with period $2L$ and is piecewise continuous on $[0, 2L]$, we shall construct the Fourier trigonometric polynomial over [0,2L] of degree m.

$$P(x) = \frac{a_0}{2} + \sum_{j=1}^{m} \left(a_j \cos\left(j \frac{\pi}{L} x\right) + b_j \sin\left(j \frac{\pi}{L} x\right) \right),$$

There are n subintervals of equal width $\Delta x = \frac{2L}{n}$ based on $x_k = k\frac{2L}{n}$. The coefficients are

$$a_j = \frac{1}{L}\sum_{k=0}^{n-1} \text{Cos}\left[j\frac{\pi}{L}x_k\right] f(x_k)\,\Delta x \quad \textbf{for} \quad j = 0, 1, 2, \ldots, m.$$

and

$$b_j = \frac{1}{L}\sum_{k=0}^{n-1} \text{Sin}\left[j\frac{\pi}{L}x_k\right] f(x_k)\,\Delta x \quad \textbf{for} \quad j = 1, 2, \ldots, m.$$

The construction is possible provided that $2m+1 \le n$.

Remark. The sums can be viewed as numerical integrationof Euler's formulae when [0,2L] is divided into nsubintervals. The trapezoidal rule uses the weights $\frac{1}{2}$ for both the $f(x_0)$ and $f(x_n)$. Since f(x) has period $2L$, it follows that $f(x_n) = f(x_0)$. This permits us to use 1 for all the weights and the summation index from $k = 0$ to $k = n-1$.

The Fast Fourier Transform for data.

The FFT is used to find the trigonometric polynomial when only data points are given. We will demonstrate three ways to calculate the FFT. The first method involves computingsums, similar to "numerical integration," the second method involves "curve fitting," the third method involves "complex numbers."

Computing the FFT with sums.

Given data points $\{(x_n, y_n)\}$ where $x_0 = 0$ and $x_n = 2L$ over [0,2L] where $x_k = k\frac{2L}{n}$ for $k = 0, 1, 2, \ldots, n$. Also given that $y_n = y_0$, to that the data is periodic with period $2L$. We shall construct the FFT polynomial over [0,2L] of degree m.

$$P(x) = \frac{a_0}{2} + \sum_{j=1}^{m}\left(a_j \cos\left(j\frac{\pi}{L}x\right) + b_j \sin\left(j\frac{\pi}{L}x\right)\right)$$

The abscissa's form n subintervals of equal width $\Delta x = \frac{2L}{n}$, based on $x_k = k\frac{2L}{n}$. The coefficients are

$$a_j = \frac{2}{n}\sum_{k=0}^{n-1} \text{Cos}\left[j\frac{\pi}{L}x_k\right] y_k \quad \textbf{for} \quad j = 0, 1, 2, \ldots, m$$

and

$$b_j = \frac{2}{n}\sum_{k=0}^{n-1} \text{Sin}\left[j\frac{\pi}{L}x_k\right] y_k \quad \textbf{for} \quad j = 1, 2, \ldots, m.$$

The construction is possible provided that $2m+1 \le n$.

Determinants and Conic Section Curves:

Background.

Five points in the plane uniquely determine an equation for a conic section. The implicit formula for a conic section is often mentioned in textbooks, and the special cases for an ellipse, hyperbola, parabola, circle are obtained by either setting some coefficients equal to zero or making them the same value.

Implicit Equation for a Line.

The equation $c_1 x + c_2 y + c_3 = 0$ of the line through the two points $(x_1, y_1), (x_2, y_2)$ can be computed with the determinant

$$\begin{vmatrix} x & y & 1 \\ x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \end{vmatrix} = 0$$

Numerical Differentiation:

Background.

Numerical differentiation formulas formulas can be derived by first constructingthe Lagrange interpolating polynomial $P_2(x)$ through three points,differentiating the Lagrange polynomial, and finally evaluating $P_2'(x)$ at the desired point. In this module the truncation error will be investigated, but round off error from computer arithmetic using computer numbers will be studied in another module.

Theorem (Three point rule for $f'[x]$). The central difference formula for the first derivative, based on three points is

$$\frac{f[x+h]-f[x-h]}{2h},$$

and the remainder term is

$$R1[x, h] = \frac{-f^{(3)}[c]}{6} h^2.$$

Together they make the equation $f'[x] = D1[x, h] + R1[x, h]$, and the truncation error bound is

$$EB1[h] = \left| \frac{-f^{(3)}[c]}{6} h^2 \right| \leq \frac{M_3}{6} h^2$$

where $M_3 = \max_{a \leq x \leq b} | f^{(3)}[x] |$. This gives rise to the Big "O" notationfor the error term for $f'[x]$:

$$f'[x] = \frac{f[x+h] - f[x-h]}{2h} + O(h^2).$$

Theorem (Three point rule for $f''[x]$). The central difference formula for the second derivative, based on three points is

$$f''[x] \approx D2[x, h] = \frac{f[x-h] - 2f[x] + f[x+h]}{h^2},$$

and the remainder term is

$$R2[x, h] = \frac{-f^{(4)}[x]}{12} h^2.$$

Together they make the equation $f''[x] = D2[x, h] + R2[x, h]$, and the truncation error bound is

$$EB2[h] = \left| \frac{-f^{(4)}[x]}{12} h^2 \right| \leq \frac{M_4}{12} h^2$$

where $M_4 = \max_{a \leq x \leq b} | f^{(4)}[x] |$. This gives rise to the Big "O" notation or the error term for $f''[x]$:

$$f''[x] = \frac{f[x-h] - 2f[x] + f[x+h]}{h^2} + O(h^2).$$

Project I.

Investigate the numerical differentiation formula $f'[x] \approx \frac{f[x+h] - f[x-h]}{2h}$ and truncation error bound $EB1[h] = \left| \frac{-f^{(3)}[c]}{6} h^2 \right| \leq \frac{M_3}{6} h^2$ where $M_3 = \max_{a \leq x \leq b} | f^{(3)}[x] |$. The truncation error is

investigated. The round off error from computer arithmetic using computer numbers will be studied in another module.

Enter the three point formula for numerical differentiation.

```
Clear[f, h, x];
D1[x_, h_] = (f[x + h] - f[x - h])/(2 h);
Print["f'[x] ≈ ", D1[x, h]];
```

$$f'[x] \approx \frac{-f[-h+x] + f[h+x]}{2h}$$

Aside. From a mathematical standpoint, we expect that the limit of the difference quotient is the derivative. Such is the case, check it out.

```
Limit[(f[x + h] - f[x - h])/(2 h), h → 0, Analytic → True]
```

```
f'[x]
```

Example Consider the function $f[x] = e^{-x}\,Sin[x]$. Find the formula for the third derivative $f^{(3)}[x]$, it will be used in our explorations for the remainder term and the truncation error bound. Graph $f^{(3)}[x]$. Find the bound $M_3 = \max_{0 \le x \le \pi} |f^{(3)}[x]|$. Look at it's graph and estimate the value M_3, be sure to take the absolute value if necessary.

Solution

Project II.

Investigate the numerical differentiation formulae $f''[x] \approx \frac{f[x-h] - 2f[x] + f[x+h]}{h^2}$ and truncation error bound $EB1[h] = \left| \frac{-f^{(4)}[x]}{12} h^2 \right| \le \frac{M_4}{12} h^2$ where $M_4 = \max_{a \le x \le b} |f^{(4)}[x]|$. The truncation error is investigated. The round off error from computer arithmetic using computer numbers will be studied in another module.

Enter the formula for numerical differentiation.

```
Clear[d, f];
D2[x_, h_] = (f[x - h] - 2 f[x] + f[x + h])/h^2;
Print["f''[x] ≈ ", D2[x, h]];
```

$$f''[x] \approx \frac{-2f[x] + f[-h+x] + f[h+x]}{h^2}$$

Aside. It looks like the formula is a second divided difference, i.e. the difference quotient of two difference quotients. Such is the case.

```
Together[(f[x+h]-f[x])/h - (f[x]-f[x-h])/h) / h]
```

$$\frac{-2\,f[x]+f[-h+x]+f[h+x]}{h^2}$$

Aside. From a mathematical standpoint, we expect that the limit of the second divided difference is the second derivative. Such is the case.

```
Limit[(f[x-h]-2 f[x]+f[x+h])/h^2, h->0, Analytic->True]
```

```
f''[x]
```

Example. Consider the function $f[x] = e^{-x}\,Sin[x]$. Find the formula for the fourth derivative $f^{(4)}[x]$, it will be used in our explorations for theremainder term and the truncation error bound. Graph $f^{(4)}[x]$. Find the bound $M_4 = \max_{0 \le x \le \pi} | f^{(4)}[x] |$. Look at it's graph and estimate the value M_4, be sure to take the absolute value if necessary.

Solution