MATH/CSC 381
Numerical Methods

Dr. John Travis
MCC 206
925-3817 (leave voice mail if no answer)
Email: travis@mc.edu

Problems:

- *Finite Precision Machines* -- round-off errors. (Computer arithmetic is not exact.) A finite mantissa and exponent give rise to only a small subset of rational numbers are expressible on a computer.
- *Approximate mathematical models* -- truncation errors. (Finite means of approximating continuous or unknown quantities.) When attempting to solve a given mathematical problem, one may make some assumptions to make the mathematical problem easier to solve.
- *Algorithm Iteration* - convergence errors – (Stopping an iterative process before completed.) One can not practically let a process complete an infinite number of steps. Models may asymptotically converge. On a machine, one may only implement a finite number of steps in such a procedure. Hence, stopping before convergence yields an error.

Good Algorithms are

- Reliable – Give the desired answers
- Robust – Works in a wide variety of situations
- Portable – Work in a variety of computing environments
- Maintainable – Are easy to modify if needs change

Goal: To produce reasonably accurate results in the presence of (1), (2) and (3) above.

Roundoff Errors -

- *underflow* $0 < |y| < 0.1 \cdot \beta^{\text{minimum}}$, *overflow* $|y| \geq 1 \cdot \beta^{\text{maximum}}$.
- *machine numbers* $y = \pm 0.d_1d_2...d_k \cdot \beta^n$, where
  - β = *base* of machine,
  - n=#of digits in *exponent*,
  - k=#of digits in *mantissa*,
  - d_j = jth digit, such that d_1 is nonzero.
- The set of machine numbers is a very small subset of the rational numbers. For an actual real number $x = \pm 0.d_1d_2...d_kd_{k+1} ... \cdot \beta^n$, the computer can only represent the "closest" machine number, denoted by fl(x).
- *Chopping* yields fl(x) = first k digits of x, regardless of the $(k+1)^{st}$ digit; *rounding* yields fl(x) =chopped $(x + 0.5 \cdot \beta^k)$.
- *absolute error* - p = exact, p* = approx. yields absolute error = $| p - p^* |$. For computer calculations, $|x - fl(x)| = 0.d_{k+1} ... \cdot \beta^{n-k}$.
- *relative error* - generally a better estimate of how bad an approximation really is. $| p - p^* | / | p |$
- *significant digits* - number of decimal places which are known to be accurate

Numerical Crimes: Things which lead to loss of significant digits...
(1) Differencing nearly equal quantities - significant digits may cancel leaving an answer with fewer significant digits.
(2) Multiplying by large quantities - significant digits might double so that machine number can't maintain all of them.
(3) Dividing by small quantities - similar to (2).

HOMEWORK PROBLEMS, page 14, #3, 4, 5, 8

Ex. Consider quadratic formula in various forms. Notice, mathematically each method is equivalent.

Remark: Rounding errors introduce errors in repeated calculations in the same way that incorrect data does.

Stability: Small changes in the "initial" data yields correspondingly small changes in the final result.

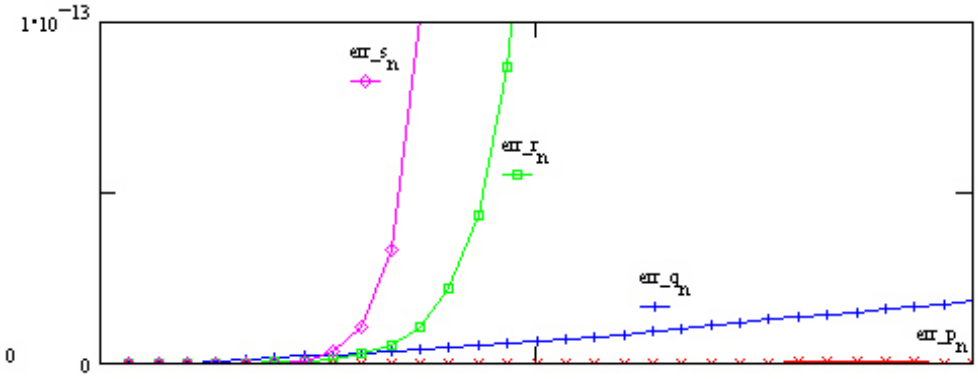
Ex: $p_n = \left(\frac{2}{3}\right)^n$, n≥0 found recursively by one of the four methods:

(a) $p_n = \left(\frac{2}{3}\right)p_{n-1}, p_0 = 1$, which is a stable algorithm, with the relative error nearest zero below

(b) $p_n = \left(\frac{7}{6}\right)p_{n-1} - \left(\frac{1}{3}\right)p_{n-2}, p_0 = 1, p_1 = \frac{2}{3}$, which is stable, with the relative error the next best, almost zero below

(c) $p_n = \left(\frac{17}{12}\right)p_{n-1} - \left(\frac{1}{2}\right)p_{n-2}, p_0 = 1, p_1 = \frac{2}{3}$, which is "relatively" stable, with the relative error going off the bottom below

(d) $p_n = \left(\frac{8}{3}\right)p_{n-1} - \left(\frac{4}{3}\right)p_{n-2}, p_0 = 1, p_1 = \frac{2}{3}$, which is unstable, with the relative error immediately going off the top below



This illustrates the effect of roundoff creeping into a calculation. As successive terms are calculated, roundoff error allows the "other solution" to enter the mix and the computed solution has some of the characteristics of this other solution as well. Dependent upon how the "other solution" compares to the desired one shows how the relative error will grow.

Another example of roundoff: Given x>1, consider the following procedure:

* Repeat $x \leftarrow \sqrt{x}$, n times
* Then, repeat $x \leftarrow x^2$, n times

In exact arithmetic, one should get what they started with. Experiment with several various values of n. Notice, if $1<x<a^2$, then $1<\sqrt{x}<a$. So, each number in the longer interval $(1,a^2)$ is mapped non-uniquely to a number in (1,a).

HOMEWORK ASSIGNMENT – page 28 #10. Fibonacci sequence question.
COMPUTING ASSIGNMENT – Assignment 1.5
READING ASSIGNMENT – Arianne 5 and Patriot Missile articles from SIAM News

Mathematical Problems to Investigate:
1. Finding Roots of equations -- Numerical solution of non-linear equations.
2. Evaluating and Graphing Functions -- Approximation of functions.
3. Integrating Hard Integrals -- Numerical Integration.
4. Finding Hard Derivatives -- Numerical Differentiation
5. Solving Linear Systems of Equations -- Gauss Elimination, Cholesky Factorization, Iterative Methods.
6. Solving Differential Equations -- Numerical O.D.E.s and P.D.E.s.

Remark: T is an operator if it maps functions onto other functions. Suppose x and y are functions with T(x)=y. Then we can classify many mathematical problems into one of the following categories:
- Direct Problem: Given f and x, find y. (3),(4)
- Inversion Problem: Given f and y, find x. (5),(6)
- Identification Problem: Given x and y, find f. (2)

Types of errors in computational problems:
1. Roundoff Errors - errors from the use of computer generated approximations to non-machine numbers
2. Truncation Errors - errors from the use of discrete mathematical models to approximate continuous or "limiting" problems
3. Convergence Errors - errors from stopping a convergent sequence after a finite number of steps

Order of Convergence: Given a function F which converges to L as $h \to 0$, then F(h) converges with order of convergence $O(h^p)$ provided there exists constants p and K such that

$$\left|F(h) - L\right| \le Kh^p .$$

As a special case, for a sequence $x_n \to x$, we have order of convergence p provided

$$\left|x_n - x\right| \le Kn^{-p} .$$

SOLUTIONS OF EQUATIONS IN ONE VARIABLE

Remark: Polynomial Rootfinding is not a stable problem.

Ex: $(x-2)^2 = 0$ has solution x=2. But $(x-2)^2 = 0.000001$ has solutions x=2.001 and x=1.997 in exact arithmetic. So, a change in the data of 0.000001 leads to a change in the final answer of 0.001, a relatively large change.

Ex. (Wilkinson) $f(x)=(x-20)(x-19)(x-18)...(x-2)(x-1) = x^{20} + 210x^{19} + ...$ obviously has roots 1,2,3,...,19,20. However, it can be shown that if the coefficient 210 is changes by a small change on the order of machine roundoff that the resulting function has in exact arithmetic only 10 real roots and 10 complex roots and these roots are completely different from 1,2,3,...,19,20. Therefore, we need to be careful about how we interpret the output from the methods below and would like to have additional information gathered (mathematically) to support the final output of each method.

Remark: Many algorithms will start from some given value and refine this values to obtain a more accurate value by using the previously obtained values. Each step is called an *iteration* and the methods are called *iterative methods*. To find solutions of equations, we will start with an initial guess to the root or an interval in which we know a root exists and then, using these, attempt to generate successive guesses which (hopefully) converge to the exact root. By letting this continue for as long as possible, we minimize the convergence error.

Problem: Given a continuous function f(x), determine a value p such that f(p)=0 (approximately). Use IVT...

BISECTION METHOD

Assume you have real numbers a and b, with a<b such that f(a) f(b) < 0. Then, IVT imples a root of f(x) exists in the interval [a,b]. Guess this root to be the midpoint of the interval = (a+b)/2. Notice, this "bisects" the original interval into two intervals, each half the size of the first. Reapply this procedure on the piece which is guaranteed to contain a root (by IVT again).

In general, consider the sequences a_n, p_n, and b_n given by

$$a_0=a,\ \ b_0=b,\ p_{n+1}=(a_n+b_n)/2$$

with a_{n+1} and b_{n+1} chosen to be the endpoints of the half of the interval that is kept. That is

$$a_{n+1}= a_n \text{ and } b_{n+1}= p_{n+1},$$
$$\text{OR}$$
$$a_{n+1}=p_{n+1} \text{ and } b_{n+1}=b_n.$$

How do you determine when this recursive process should end? How can you determine if $f(p_{n+1})=0$ in the presence of roundoff?

Error bound after n steps is

$$| x^* - p_{n+1} | < (b-a)\, 2^{-n}.$$

Hence, if you can choose the number of steps n such that $(b - a)\, 2^{-n} = \varepsilon$, for some small ε, then p_{n+1} must be at least ε -close to x^*.

Notice, one also could perhaps stop the iteration when $| f(p_{n+1}) | <$ tolerance.

Better to just check sign(f(a)) sign($f(p_n)$), instead of actually multiplying f(a) $f(p_n)$.

Bisection is "Robust" in that it will always converge to an exact root but will do so rather slowly.

HOMEWORK: page 38, #8, 10 (by hand using the interval [2,3] as the starting point.)

NEWTON'S METHOD - pg 45
--Suppose p is the unknown actual root of f(x) = 0 and p_0 is an approximation to p.
From Taylor's Theorem,

$$f(p) = f(p_0) + (p-p_0)f'(a) + (p-p_0)^2 f''(c)/2!,$$

where c is unknown. If p_0 is close to p, $p - p_0$ is "small" and $(p-p_0)^2$ is even "smaller". Then

$$0 = f(p) \approx f(p_0) + (p-p_0)f'(p_0),$$

or by solving for p, we obtain

$$p \approx p_0 - f(p_0)/f'(p_0).$$

Denote by p_1 the value computed by using the (exact) equation

$$p_1 = p_0 - f(p_0)/f'(p_0).$$

In general,

$$p_{n+1} = p_n - f(p_n)/f'(p_n), \text{ where } p_0 \text{ is known.}$$

- Also, can be derived graphically using a triangle with hypotenuse given by the tangent line to (a,f(a)).
- Newton's method is not as robust as Bisection but converges much faster. See page 47-48.

HOMEWORK: page 50 #4, 10 (by hand starting with initial guess 2.5), 16

SECANT METHOD - pg 39
- Use Newton's by replacing the explicit evaluation of f'(a) with the approximation .
- Requires no derivative but is not as fast as Newton's and needs two starting values instead of one.

METHOD OF FALSE POSITION - pg 42

- Mixes the ideas of the Secant Method and the Bisection Method.
- Convergence of this method can be hard to determine and even slower than Bisection.

COMPUTING ASSIGNMENT #3

Algorithm Rate of Convergence: If p is the exact value being successively approximated by the values p_n, then p_n converges to p…

- Linearly provided $| p - p_{n+1} | < M | p - p_n |$, for some constant M.
- Quadratically $| p - p_{n+1} | < M | p - p_n |^2$, for some constant M
- with rate r provided $| p - p_{n+1} | < M | p - p_n |^r$, for some constant M and largest constant r

Bisection exhibits linear convergence, Newton's exhibits quadratic convergence and Secant exhibits something in between.

HORNER'S ALGORITHM: Each calculation introduces additional roundoff and computer time. To minimize these for a given x value, rearrange the computations using a judicious choice of nested parenthesis. (Equivalent to Synthetic Substitution)

MUELLER'S METHOD: Takes 3 initial guesses p_0, p_1 and p_2, passes a quadratic through them and determines the zero closest to p_2.

The quadratic: $q(x) = a(p- p_2)^2 + b(p- p_2) + c$.

Evaluating at the "interpolation" conditions

$$f(p_0) = q(p_0)$$
$$f(p_1) = q(p_1)$$
$$f(p_2) = q(p_2)$$

yields the formulas on page 58. Use the improved quadratic-formula formulas and choose sign to make bottom largest possible.

- Requires a square root at each step and so is possibly more expensive to implement
- Only requires original function, not derivative
- Can approximate complex roots easily by using complex arithmetic and complex square roots.
- Rate of convergence is somewhere between Secant and Newton's

INTERPOLATION AND POLYNOMIAL APPROXIMATION

Weierstrass Approximation Theorem: If feC[a,b], then there exists a polynomial P(x) s.t. P(x) is arbitrarily close to f(x) on [a,b].

Remark: This is an existence theorem. In practice, finding the polynomial might be very difficult and of very high degree.

Taylor Polynomials: Using Taylor's theorem by throwing away the remainder term. However, one must know the actual function value at x=c as well as its derivatives values at some given point c before this can be used. Often this is requiring too much and so this may not be useful. Also, the Taylor's polynomial is only valid for interpolation in a small interval about x=c. Look at error bound. Therefore, this is not such a great method to use for real-life examples where the derivative values would most likely not be given or even easily approximated. Generally, Taylor's polynomials are only used for theoretical purposes.

Ex: Approximate f(x)=1/x about c=1 using higher and higher Taylor's polynomials. The approximation gets worse and worse as the degree of the polynomial gets larger.

LaGrange Polynomials: Given n+1 values of the function $(x_0,f(x_0))$, $(x_1,f(x_1))$, ... , $(x_n,f(x_n))$, find the polynomial or degree n which passes through all these points.

Derivation:

(1) Set up a general nth degree polynomial and plug in the data values to get a system of equations in the coefficients.
(2) Use Lagrange cardinal formulas. Writing these using a Horner-type arrangement gives a faster implementation.
(3) To actually compute the polynomial, one should use Newton's Divided Differences.

Ex: Approximate f(x)=1/x using $x_0=1$, $x_1=2$, etc. for a few values. This gives a much better approximation than Taylors did.

Neville's Method: A recursive way to generate values of the Lagrange polynomial. It does not, however, give a formula but a value for each variable x.

Error bound: See page 57 for bound on the error of approximating f(x) with it's Lagrange polynomial.

Result: Any two polynomials of the same degree n which agree at n+1 points are equal.
Pf: Suppose f(x), g(x) are of degree n and agree at n+1 points.
Then, r(x)=f(x)-g(x) is of degree at most n with n+1 roots gives r(x)=0.

Ex: Construct a parabola passing through (1,4), (-1,0) and (2,9). Then $p(x)= ax^2 + bx + c$ and by plugging in the points,

$4 = a + b + c$
$0 = a - b + c$
$9 = 4a + 2b + c,$

which is a 3x3 system of equations to solve.

If the polynomial p(x) were of higher degree, then this system is likely very difficult to solve.

However if we write $p(x)=c + b(x-1) + a(x-1)(x+1)$, then plugging in the points yields

$4 = c$
$0 = c - 2b$, or $b=2$
$9 = c + b + 3a$, or $a=1$,

which is very easy to solve successively. This second form is called Newton's form.

Newton's Divided Differences: Successively build up the polynomial by finding the coefficients as above.

- (x_0,f_0) gives $p_0(x)=f_0$, passing through the one point.
- (x_1,f_1) gives $p_1(x)=p_0(x) + (x-x_0)b_1$, passing through the first two points if $b_1=(f_1-f_0)/(x_1-x_0)$.
- (x_2,f_2) gives $p_2(x)=p_1(x) + (x-x_0)(x-x_1)b_2$, passing through the first three points if $b_2=?$.

Continuing in this manner shows the coefficients b_i are given by the divided difference table in the text where the terms are defined recursively via

$$f[x_j,...,x_k] = (f[x_{j+1},...,x_k] - f[x_j,...,x_{k-1}]) / (x_k - x_j), \text{ for } j<k.$$

If $x_0 < x_1 < ... < x_n$, this is called Newton's Forward Divided Difference Polynomial.
If $x_0 > x_1 > ... > x_n$, this is called Newton's Backward Divided Difference Polynomial.

If Δx is constant, the Divided Difference Formulas can be written in a closed form involving the binomial coefficents.

Program Subroutine To Determine the Newton's Divided Differences:
Let $c_j=f(x_j)$ for j=0,1,...,n
for k= 1:n
    for j=n:-1:k
        $c_j:=(c_j-c_{j-1})/(x_j-x_{j-k})$
    end
end

Then, the Newton's Interpolation polynomial is given by (with the f_j from above):

$$p(x) = c_0 + c_1(x-x_0) + c_2(x-x_0)(x-x_1) + c_3(x-x_0)(x-x_1)(x-x_2) + \ldots + c_n(x-x_0)(x-x_1)\ldots(x-x_{n-1}).$$

Notice, to evaluation of this polynomial at any given x is very easy and fast when using Horner's Algorithm.

$$p(x) = c_0 + (x-x_0)\{c_1 + (x-x_1)\{c_2 + (x-x_2)\{c_3 + \ldots + (x-x_{n-2})\{c_{n-1} + c_n(x-x_{n-1})\}\ldots\}\}\}.$$

Storage: To create the coefficients, you only need one vector and keeping only the bold term below at each stage:

$\mathbf{f_0}$
f_1 $\mathbf{f[x_0,x_1]}$
f_2 $f[x_1,x_2]$ $\mathbf{f[x_0,\ldots,x_2]}$
...
f_{n-1} $f[x_{n-2},x_{n-1}]$ $f[x_{n-3},\ldots,x_{n-1}]$ $f[x_{n-4},\ldots,x_{n-1}]$
f_n $f[x_{n-1},x_n]$ $f[x_{n-2},\ldots,x_n]$ $f[x_{n-3},\ldots,x_n]$ ... $\mathbf{f[x_0,\ldots,x_m]}$

Hermite Interpolation: Not only are the function values specified but also several derivative values. We will only consider the case when we match function values and 1st derivative values. This is implemented using a set of basis functions similar to Lagrange.

Piecewise Interpolation: Instead of using one polynomial (possibly of very high degree) for the entire interval of interest, use several polynomials each defined on small intervals. Then, to obtain the y-value for a given x-value, one must first determine which formula to use. Then, evaluate that formula for the given x-value. If $x_0 < x_1 < \ldots < x_n$, and $x_0 < x < x_n$, then one can determine which interval x belongs to by considering the iteration:

k = 0
Repeat
    k = k+1
Until $x \le x_k$
(Use the kth formula...)

Piecewise Cubic Hermite Interpolation: Suppose 2 function values and 2 derivative values are specified. Then, the interpolant will be of the form (assuming the endpoints are u=0 and u=1)

$$p(u) = H_0(u)\, f(0) + H_1(u)\, f'(0) + H_2(u)\, f'(1) + H_3(u)\, f(1) ,$$

where

$$H_0(u) = (1-u)^2(1+2u),$$
$$H_1(u) = (1-u)^2u,$$
$$H_2(u) = (u-1)u^2,$$
$$H_3(u) = (3-2u)u^2.$$

Given (x_k, y_k) and slopes m_k, then the piecewise formula for the interval $[x_{k-1}, x_k]$ is given by

$$p_k(x) = y_{k-1}H_0((x-x_{k-1})/\Delta x_k) + m_{k-1}\Delta x_k\, H_1((x-x_{k-1})/\Delta x_k) + m_k\Delta x_k\, H_2((x-x_{k-1})/\Delta x_k) + y_k\, H_3((x-x_{k-1})/\Delta x_k).$$

Notice, if we desire to evaluate this piecewise approximation for a fixed number of equally spaced points in each sub-interval, we can evaluate the basis polynomials once and store the results in vectors. These then can be reused for each interval.

Cubic Splines: Piece together cubics of the form

$$S_k(x) = a_k + b_k(x-x_{k-1}) + c_k(x-x_{k-1})^2 + d_k(x-x_{k-1})^3, \text{ for } x_{k-1} \leq x \leq x_k,$$

in such a way that the resulting approximation is continuous and has 2 continuous derivatives (no visible kinks). To achieve these ends, we impose the conditions:

Interpolation at left endpoint: $S_k(x_{k-1}) = y_{k-1}$, which yields n equations, for k=1 to n,

$$a_k = y_{k-1},$$

Interpolation at right endpoint: $S_k(x_k) = y_k$, which yields n equations, for k=1 to n,

$$a_k + b_k\Delta x_k + c_k\Delta x_k^2 + d_k\Delta x_k^3 = y_k,$$

Derivative Continuity: $S_k'(x_k) = S_{k+1}'(x_k)$, which yields n-1 equations, for k=1 to n-1,

$$b_k + 2c_k\Delta x_k + 3d_k\Delta x_k^2 = b_{k+1}, \text{ and}$$

2nd Derivative Continuity: $S_k''(x_k) = S_{k+1}''(x_k)$, which yields n-1 equations, for k=1 to n-1,

$$2c_k + 6d_k\Delta x_k = 2c_{k+1}.$$

Hence, we have 4n unknowns (the vectors of coefficients a, b, c and d) and 4n-2 equations. By adding two additional equations, we can uniquely solve for the unknown vectors.

End Conditions:
(1) Free spline - Assume S''(a) = S''(b)=0. This means the curve "loses" its concavity as you approach the endpoints.
(2) Clamped spline - For m_k given, set $S'(a)=m_0$, $S'(b)=m_1$. This means the curve is forced to go in a given direction at each end.
(3) Bessel ends - Set m_k automatically by using the tangent to the parabola which passes through the first three data points.
(4) Quadratic ends - For $a=x_0$, set $S''(a)=S''(x_1)$ and similarly for the other end.
(5) Not-a-knot - Force the cubic polynomials on the first two and the last two intervals to actually be the same spline

Geometric Design: Instead of thinking of data points as values to interpolate, think of data points as "attractors" of the curve.
Introduce Bezier curves as recursively linearly interpolating....

Bezier Interpolants: Suppose that we are given "control points" $\mathbf{b}_k = (bx_k, by_k)$ such that we desire a polynomial interpolant which passes through the first control point $\mathbf{b}_0$ and the last one $\mathbf{b}_n$, while the other points serving as attractors of the curve. Then, the curve can be given parametrically as

$$x(t) = \Sigma\, bx_k\, B_k(t), \text{ and}$$
$$y(t) = \Sigma\, by_k\, B_k(t),$$

where $B_k(t)$ is the kth Bernstein polynomial.

Derivative Continuity of Bezier Interpolants: The derivative of a Bezier curve is a Bezier curve of one less degree by simply taking the derivative of the formulas above. By equating these derivatives at the points where two curves meet, we force the resulting piecewise interpolant to be smooth. We can continue to force successive derivative to be equal at the points where the curves meet to get an interpolant with very high degree of smoothness. Notice, the similarity to splines.

NUMERICAL DIFFERENTIATION AND INTEGRATION

Remark: In many real-life mathematical models, one must utilize the integral or the derivative of a given function. Sometimes, these functions are easily differentiable or integrable. Many times they are not or doing so would be difficult. The question becomes determining how we can find good approximations to derivatives evaluated at given points and to definite integrals.

One idea: For any given function (or set of data points), determine a set of n+1 data points (x,y) and use the concepts from Chapter 3 to find a polynomial approximation to the original function. Usually, this polynomial approximation is very easy to differentiate and integrate. (Actually, this is not quite as true when using the Newton-type forms of the polynomial.) Use the resulting values from the polynomial as approximate values for the exact values.

Classroom Computing Assignment: Determine the Lagrange interpolating polynomial to $f(x)=1/(1+x^2)$ on the interval [-3,3] of degree n for various n. (Use the matrix approach by setting up the Vandermonde matrix, etc.) Then, differentiate and integrate this polynomial exactly using formulas and compare with the exact values. Experiment with at least two other nontrivial functions for f(x) for which you can get an exact answer via calculus.

Remark: In the following, we will only consider determining derivatives at given points and definite integrals since each of these is an actual number. If an interval approximation to the derivative function is desired or the indefinite integral is needed, the ideas which follow will need to be implemented over and over, once for each new value desired.

Derivative over [a,b]:

(1) Partition the interval [a,b] into $x_0, x_1, \ldots, x_n$.

(2a) Use some methods to determine an approximation m_k to the derivative for each x_k.

(3a) Connect these points (x_k,m_k) with some interpolant from chapter 3.

or

(2b) Create the polynomial p(x) to interpolate (x_k,y_k) using some method from chapter 3.

(3b) Determine exactly the derivative of the interpolant and evaluate $p'(x_k)$.

Integral over [a,b]:

(1) Partition the interval [a,b] into $x_0, x_1, \ldots, x_n$. Generally, we will assume that the partition is uniform with common interval width $\Delta x_k = h$.

(2a) Use some method to determine an approximation I_k to the $\int_{x_0}^{x_k} f(x)dx$.

(3a) Connect these points (x_k,I_k) with some interpolant from chapter 3.

Or

(2b) Create the polynomial p(x) to interpolate (x_k,y_k) using some method from chapter 3.
(3b) Determine exactly the indefinite integral of the interpolant.

(4) Note that in each case the answer might be off by an arbitrary constant.

Note: From the FTOC, if $F'(x) = f(x)$,

$$I_k = \int_{x_0}^{x_k} f(x)dx = F(x_k) - F(x_0) = F(x_k) - F(x_{k-1}) + F(x_{k-1}) - F(x_0) = \int_{x_{k-1}}^{x_k} f(x)dx + I_{k-1}.$$

Similarly,

$$I_k = \int_{x_0}^{x_k} f(x)dx = F(x_k) - F(x_0) = F(x_k) - F(x_{k-2}) + F(x_{k-2}) - F(x_0) = \int_{x_{k-1}}^{x_k} f(x)dx + I_{k-2}.$$

Therefore, we can determine an approximation to the definite integral over an wide interval recursively by simply adding one smaller integral over the next subinterval (or two) to the previous accumulated total.

Defn: Suppose A is an approximation to E. We say A is an order $O(h^p)$ approximation to the exact value E if, by using Taylor's expansions, we have $E - A = O(h^p)$, where h=a constant difference in x values. That is, if you take the difference of E and A, you get terms which only have powers of h bigger than or equal to p.

Numerical Integration:
From the definition of integration, one obtains the definite integral using limits of Reimann sums

$$\int_a^b f(x)dx = \lim \sum f(x_k)\,\Delta x_k .$$

An approximation to the value of the definite integral may be found by replacing the infinite sum with a finite one. In general, a numerical quadrature rule approximates the definite integral with a finite sum of the form

$$\sum a_k f(x_k),$$

where the x_k are a reasonably fine finite partition of the interval [a,b].

Defn: Given a numerical quadrature rule, the order of the method is the degree of the highest polynomial for which the method will obtain the exact integral.

Methods: Approximate the function f(x) with polynomials of ever increasing degree.
(a) Midpoint Rule: Approximate f(x) with a constant polynomial evaluated at the midpoint of the interval. If the number of subintervals is even, on each pair of subintervals approximate the original function with a constant having the value of the function at the center of the subinterval (ie. a constant interpolant). Notice, this method does not require the function to be evaluated at either endpoint and so is a good choice to use if the integrand is singular at an endpoint.
(b) Trapezoidal Rule: Approximiate f(x) with a linear polynomial which interpolates both endpoints. On each subinterval, approximate the "area under the curve" by using trapezoids.
(c) Simpson's Rule: Approximate the original function with a quadratic which interpolates the original function at each endpoint and at the midpoint of the interval.
(d) Gaussian Quadrature: Most accurate. Allow both the coeffiecients a_k and x_k to be variable. Use table on page 141 for [a,b]=[-1,1].

Composite Methods: Since the interval [a,b] can be large, to get a better approximation we break the interval into smaller pieces and accumulate the value of the definite integral using

$$I_k = \int_{x_{k-1}}^{x_k} f(x)dx + I_{k-1}$$, for "one-step" methods,

or

$$I_k = \int_{x_{k-1}}^{x_k} f(x)dx + I_{k-2}$$, for "two-step" methods.

This yields the "composite rules" noted in the text.

Homework: page98, #7, 9, 11 (a,c;g only) page 105, #7, page 111, #1, 2 (a,c,g only)

Derivatives:
(a) Forward-Difference formula - page 140. (Difference quotient when h>0.) Has error O(h).
(b) Backward-Difference formula - page 140. (Difference quotient when h<0.) Has error O(h).
(c) Centered-Difference (Midpoint) formula - page 142. (Difference quotient centered around a.) Has error $O(h^2)$.
(d) See various formulas on page 142, etc.
(e) Second Derivative formula - page 146

Remark: If one wants the derivative at several points x_k over an interval, use a difference formula recursively. However, numerical differentiation should be avoided as much as possible since the process is not stable over large intervals.

Homework: page 146, #5, 6

Computing Assignments: In a MathCad or in a high level programming language, implement as separate functions the forward difference, backward difference and central difference methods for determining the derivative as well as the trapezoidal rule, Simpson's rule and the midpoint rule for determining the definite integral. Approximate the deriviative and integral of $f(x) = 1/(1+x^2)$ for several different values. Compare your answers with the known exact derivative $f'(x) = -2x/(1+x^2)^2$ and the (indefinite) integral $\tan^{-1}(x)$. Consider other functions also and compare your approximation with the exact value via calculus.

CHAPTER 5 - SOLVING INITIAL VALUE PROBLEMS

Ex: Solving y'=4y gives $y(t)=Ce^{4t}$, where C is any constant. To eliminate the arbitrary constant C, we would need an initial condition for y(a) given. For example, y(0)=3 implies $y(t)=3e^{4t}$.

General Problem: Solve for y(t) on the interval $a \le t \le b$,
(**IVP**) y'=f(t,y(t))
y(a)=a, given.

Remark: Elementary Ordinary Differential Equation (ODE) theory gives mathematical discussions on the existence and uniqueness of exact solutions to (IVP). (See Well-Posed Condition on page 151.) We will always assume that we are given problems for which solutions exist and are unique.

Remark: If (IVP) cannot be solved by analytical methods, we will need to find a discrete solution by

finding an approximation to the solution at a finite number of points. Once we have these points, we can use approximation theory from Chapter 3 to find an approximate continuous solution.

Notation: When solving an Initial Value Problem numerically, we will assume the following:

- f(t,y)
- Initial condition $y(a) = y_0$
- Interval endpoints a and b
- Number of points desired n+1

Let $h = (b-a)/n$, be the mesh size with equally spaced mesh points given by $t_j = a + j\,h$.
Denote w_j = algorithm generated approximation for $y(t_j)$.
Denote $f_j = f(t_j,w_j)$, for j=0,1,2,...,n.

Euler's Method:

$w_0 = a$,
$w_{j+1} = w_j + h*f_j$, for j=0,1,2,...n.

Derivations of Euler's Method:

- From the DE, $y'(t_j) = f(t_j,y(t_j))$ and replace $y(t_j)$ by a forward difference formula.
- Use Taylor's theorem to expand $y(t_{j+1})$ about t_j and throw away nonlinear terms. Easily gives local error.
- $y(t_{j+1}) - y(t_j) = \int_{t_j}^{t_{j+1}} f(t,y)dt \approx \int_{t_j}^{t_{j+1}} f(t_j,y(t_j))dt = h\, f(t_j,y(t_j))$, approximating the integral using a single rectangle evaluated at the left endpoint
- Graphically, using the equation of the tangent line at $t=t_j$, extrapolate to $t=t_{j+1}$. Slope is $m=y'=f(t_j,y(t_j))$ yields $y - w_j = f_j\,(t - t_j)$. Plugging in $t=t_{j+1}$ gives the method.

Classroom Computing Assignment: Program Euler's Method in Mathviews using an external function for f(t,y) for the following Differential Equations. Represent the numerical solutions graphically and compare it graphically with the known exact solutions. Give output corresponding to several different values for the step size h.

- $y' = y/2$, with $y(0) = 1$. Compare to exact solution of $y(t) = e^{0.5t}$.
- $y' = \cos(t)$, with $y(0) = 0$. Compare to exact solution of $y(t) = \sin(t)$.
- $y' = y^2 / t^3$, with $y(1) = 2$. Compare to exact solution of $y(t) = 2t^2$.

Remark: We would like to compare methods for solving (IVP) to see which are "better" to use. To do this, consider the difference equation written in the form

$$w_{k+1} = w_j + h*f(t_k,w_k),$$

where f is "all the rest" of the method.
Define the local truncation error of a method to be

$$t(h) = (y(t_{k+1}) - y(t_k))/h + f(t_k,y_k).$$

The order of the method is the smallest power of h remaining in t(h) after expanding all terms about $t=t_k$ and simplifying.

Result: Euler's method is order 1.

Taylor's Methods: page 186

To develop methods of order higher than one, use the Taylor's exansion for $y(t_k+h)$ about t_k and note that the derivatives of y can be obtained to arbitrarly high degree by using the D.E. y'=f(t,y), y''=f'(t,y), y'''=f''(t,y), etc, where f(t,y(t)) is a given function.

Homework: page 190, #3, 6.

Runge Kutta Methods:

Instead of evaluating f(t,y) only at "nodal" points, allow this function to be evaluated at points between nodes. The extra degrees of freedom give higher order methods which don't involve the derivative calculations needed in Taylor's methods.

- Midpoint Method - page 194
- Modified Euler Method - page 195
- Heun's Method - page 195
- Classical Runge-Kutta Method - page 196 (This is one of the methods most often used in practice.)

Homework: page 199, #1, 2, 3, 10 (Feel free to use Software.)

Multi-Step Methods:

Instead of only using information at time t to get the approximation at time t+h, use the computational values from previous time steps also. This enables one to get methods of high accuracy without the complexity of using the Runge-Kutta methods. The tradeoff is that this type of method needs more that one starting value and these must be generated by a single-step method.

-Adam's-Bashforth Methods - page 169, explicitly gives w_{k+1}.

-Adam's-Moulton Methods - page 170, implicitly gives w_{k+1}.

Often, these multi-step methods are used together with an Adam's-Bashforth method used to give a first approximation to w_{k+1} and then an Adam's-Moulton method used to improve this value.

SOLVING SYSTEMS OF LINEAR EQUATIONS

Discuss:
(1) System of equations
(2) Matrix notation and the operations on matrices.
(3) Solving Diagonal Systems and Triangular Systems.
    (a) Forward Solving - $O(n^2)$
    (b) Backward Solving - $O(n^2)$
(4) Gaussian Elimination.
(5) Factorization of Matrices and using this to obtain the determinant in $O(n^3)$ time (vs. n! time with cofactor expansion)
(6) Other factorizations: CROUT, DOOLITTLE, CHOLESKY
(7) Triangular Solvers.

Remark: It is generally better to consider solving systems of equations in the matrix factorization context since in many situations this leads to a significant computational savings.

(a) Determination of A^{-1} using the factorization. Notice, if z_k is the kth column of A^{-1} and e_k is the column vector with all zeros except for a one in the kth position, we can write the matrix equation $A*A^{-1} = I$ as n matrix equations $A*z_k = e_k$. Factoring the matrix A into A=LU once, yields $LUz_k=e_k$, and notice each of these can be solved in $O(n^2)$ time.

(b) Consider a tensor product interpolating surface $x(u,v) = S\ S\ c_{i,j}\ A_i(u)\ B_j(v)$, where $A_i(u)$ and $B_j(v)$ are basis functions. Suppose we are given data points to interpolate $x_{i,j}$. Plugging these points into the above tensor product yields a system of equations X=ACB, where X, A, C, B are matrices. The formal solution of this is $C=A^{-1}XB^{-1}$, which requires $O(n^6)$ operations. Using a factorization process carefully yields only $O(n^4)$ operations - a significant savings for even small systems.