This document is to remind myself of some of the numerical methods useful to the MAE 3403 class. Most of the content is a summary from “*Numerical Methods”*, by *Robert W. Hornbeck*.

# Ch. 2: The Taylor Series

If *f*(*x*) can be represented by a convergent power series within an interval centered at *b*, it is said to *analytic*.

Thus for *x* within a convergent interval, *f* is given by a convergent power series: 

Differentiating this power series *n* times and setting *x*=*b* gives: 

Thus, the Taylor series of *f*(*x*) in the region of *x* close to x=*a* is given by: 

Suppose we want to find the value of *f*(*x*=b):



*Example*: Prove *Euler*’*s* formula which states: , where *j*=(-1)1/2

If true, the Taylor’s series expansions of both sides of the equation should be equal.

For LHS (*ejx*):



For RHS (cos(x)): recall 



For RHS (sin(x)):



*Q.E.D.*

# Ch. 3: The Finite Difference Calculus

## 3.1 Forward and backward differences

*h*

*h*

*f*(*x*)

*fj-2 f­j-1 fj fj+1 fj+2*

x-2h x-h x x+h x+2h

*x*

Consider a function *f*(*x*) which is analytic in the neighborhood of a point *x*. We can find *f*(*x*+*h*) by:



Now, solving for *f ‘*(*x*):

→ 

If we let  and  and  , then:

 is *the first forward difference approximation* of *f’*(*x*) with error of order *h*.

We can do a similar process for *f*(*x*-*h*), such that:

, where  is *the first backward difference approximation* of *f’*(*x*).

*3.2 Higher order derivative approximations*:

Find *f*(*x*+2*h*) by: 

Subtracting 2*f*(*x*+*h*), I can find: 

Solving for *f’’*(*x*):



Using subscript notation: 

In a similar way using the backward expansion: 

These are the *second forward* & *backward difference* equations. We now note that:

 and 

For example: 

Therefore:  and 

In table form:

**Forward Difference Approximation Backward Difference Approximation**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | *fj* | *fj+1* | *fj+2* | *fj+3* | *fj+4* |
| *hf’*(*xj*)= | -1 | 1 |  |  |  |
| *h2f’’*(*xj*)= | 1 | -2 | 1 |  |  |
| *h3f’’’*(*xj*)= | -1 | 3 | -3 | 1 |  |
| *h4fiv*(*xj*)= | 1 | -4 | 6 | -4 | 1 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | *fj-4* | *fj-3* | *fj-2* | *fj-1* | *fj* |
| *hf’*(*xj*)= |  |  |  | -1 | 1 |
| *h2f’’*(*xj*)= |  |  | 1 | -2 | 1 |
| *h3f’’’*(*xj*)= |  | -1 | 3 | -3 | 1 |
| *h4fiv*(*xj*)= | 1 | -4 | 6 | -4 | 1 |

To get a more accurate estimate of the *first forward* and *backward differences*, we can keep an extra term in the Taylor series expansion: and substitute for *f*’’(*x*)





 Note: this estimate has less error

## 3.3 Central Differences

 and 

So,  → 

∴ *first central difference*: 

*second central difference*:  → 

In Table form:

**Central Difference**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | *fj-2* | *fj-1* | *fj* | *fj+1* | *fj+2* |
| *2hf’*(*xj*)= |  | -1 | 0 | 1 |  |
| *h2f’’*(*xj*)= |  | 1 | -2 | 1 |  |
| *2h3f’’’*(*xj*)= | -1 | 2 | 0 | -2 | 1 |
| *h4fiv*(*xj*)= | 1 | -4 | 6 | -4 | 1 |

# Ch. 4: Interpolation and Extrapolation

*f*(*x*)

*x*

*-4h -3h -2h -h 0 h 2h 3h 4h*

Consider we have some experimental data at discrete values of x, but would like to know the value of *f*(*x*) at some *x* where no data was collected.

## 4.0-4.3 Evenly spaced data

Some example data might be

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| *x* | 0 | 1 | 2 | 3 | 4 | 5 |
| *f*(*x*) | -7 | -3 | 6 | 25 | 62 | 129 |

|  |  |  |
| --- | --- | --- |
| forward difference table | backward difference table | central difference table |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | *x* | *f*(*x*) | Δ*f* | Δ2*f* | Δ3*f* | Δ4*f* | Δ5*f* | | 0 | -7 | 4 | 5 | 5 | 3 | 1 | | 1 | -3 | 9 | 10 | 8 | 4 |  | | 2 | 6 | 19 | 18 | 12 |  |  | | 3 | 25 | 37 | 30 |  |  |  | | 4 | 62 | 67 |  |  |  |  | | 5 | 129 |  |  |  |  |  |   Δ*f*0=*f*1-*f*0  Δ2*f*0=(*f*2-*f*1)-(*f*1-*f*0)=*f*2-2*f*1+*f*0  Δ3*f*0=[(*f*3-*f*2)-(*f*2-*f*1)]-[(*f*2-*f*1)-(*f*1-*f*0)]=*f*3-3*f*2+3*f*1-*f0* | |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | *x* | *f*(*x*) | ∇*f* | ∇2*f* | ∇3*f* | ∇4*f* | ∇5*f* | | 0 | -7 |  |  |  |  |  | | 1 | -3 | 4 |  |  |  |  | | 2 | 6 | 9 | 5 |  |  |  | | 3 | 25 | 19 | 10 | 5 |  |  | | 4 | 62 | 37 | 18 | 8 | 3 |  | | 5 | 129 | 67 | 30 | 12 | 4 | 1 | | |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | *x* | *f*(*x*) | δ*f* | δ2*f* | δ3*f* | δ4*f* | δ5*f* | | 0 | -7 |  |  |  |  |  | | 0.5 |  | 4 |  |  |  |  | | 1 | -3 |  | 5 |  |  |  | | 1.5 |  | 9 |  | 5 |  |  | | 2 | 6 |  | 10 |  | 3 |  | | 2.5 |  | 19 |  | 8 |  | 1 | | 3 | 25 |  | 18 |  | 4 |  | | 3.5 |  | 37 |  | 12 |  |  | | 4 | 62 |  | 30 |  |  |  | | 4.5 |  | 67 |  |  |  |  | | 5 | 129 |  |  |  |  |  | |

For Taylor series around x=0, we have:  and forward difference equation of:



After continuing this type of substitution…

*Gregory-Newton forward interpolation formula*



*Gregory-Newton backward interpolation formula*



If we rescale the *x* variable such that *h*=1, we have:

*Gregory-Newton forward interpolation formula*



*Gregory-Newton backward interpolation formula*



Note: The *x* axis in the difference table can be shifted so that any desired point corresponds to x=0 (i.e., where we developed the difference tables based on the Taylor expansion).

## 4.4 Unevenly spaced data: Lagrange Polynomials

Consider a set of data *f*(*xi*) where the *xi* are not evenly spaced and there are *i*=*n*data points. We can define a polynomial of degree *n* associated with each point *xj*  as:

 (Note: exclude *xj* term)

Where *Aj* is a constant. Thus, 

So, when *x* is equal to any of the *xi* values, *Pj*=0, but *Pj*≠0 when *x*=*xj*.

Then, if one of the data points in the set of data is marked as *xk*: 

And we define *Aj* as:  which means: 

The polynomials *Pj*(*x*) are defined such that each one passes through zero at each data point except for the one point *xk* and they are called *Lagrange polynomials.* We can form a linear combination of the *Pj*(*x*):

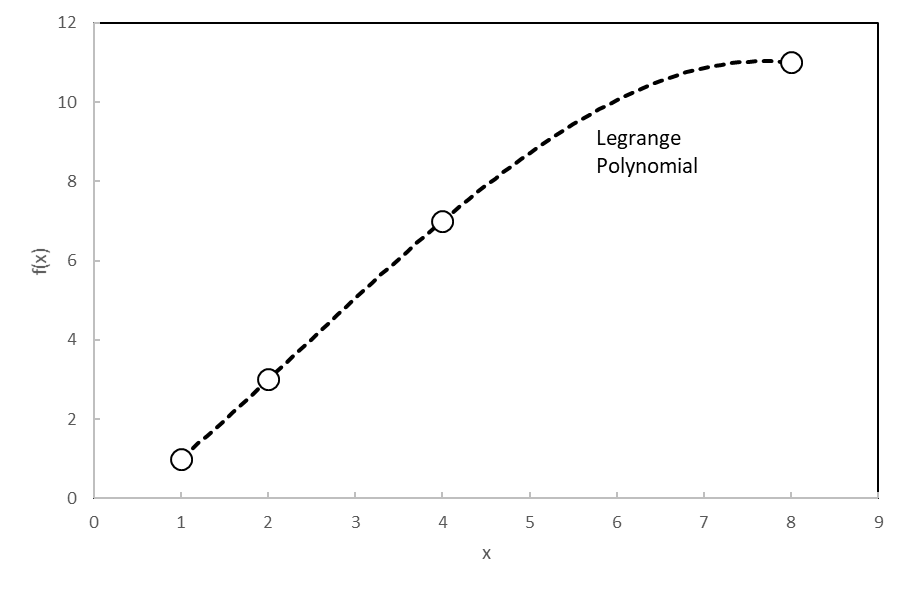


If we select one of the points (say *x2*) then: 

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *i* | 0 | 1 | 2 | 3 |
| *xi* | 1 | 2 | 4 | 8 |
| *f*(*xi*) | 1 | 3 | 7 | 11 |

*Example*: consider the following set of data

We wish to interpolate for *f*(7), then:



## 4.6 Interpolation with Cubic Spline Function

Given a set of points *xi* (i=0, 1, 3,…,n) which are not generally evenly spaced, and the corresponding *f*(*xi*), consider two adjacent points *xi* and *xi*+1. We wish to fit a cubic *Fi*(*x*) to these two points and use it as an interpolating function between them.

 for 

We have four unknowns and two end conditions: *Fi*(*xi*)=*f*(*xi*) and *Fi*(*xi+1*)=*f*(*xi+1*)

The next step is to also match the first and second derivatives of *Fi*(*x*) to those of the cubic *Fi-1*(*x*) in the adjacent interval (*xi*-*1*≤*x*≤*xi*). Carrying out this procedure for (*x0*≤*x*≤*xn*) with special treatment of the end points, an approximating function for the region will be constructed consisting of the set of cubics *Fi*(*x*) (*i*=0,1,…n-1). This function is denoted *g*(*x*) and called a cubic spline.

To construct *g*(*x*), it is convenient to note that, due to the matching of second derivatives of the cubics at each point *xi*, the second derivative of *g*(*x*) is continuous over the entire region *x0*≤*x*≤*xn*. Note that since we are dealing with a cubic, the second derivative is a straight line over each interval. At any point *x* in the interval *xi*≤*x*≤*xi+1*: And for interval *xi-1*≤*x*≤*xi*:

Integrating twice with Δ*xi*=(*xi+1*-*xi*) and Δ*xi-1*=(*xi*-*xi-1*) gives :

The expressions above are the 3rd order polynomials we need to fit the actual function in the intervals. But, it has lost the nice form of . Nevertheless, if we can find the values for *g’’*(*x*), we can use these expressions for interpolation. To do this, we must differentiate the above expressions and remember that , and. Thus:







At *x*=*xi*:

Since :





We could simplify the notation to:

 where , 

Thus, for all intervals *i*=1,2,…,n-1 we have this equation.

If the *xi* are evenly spaced then *λi*=2, *μi*=1 and:



We can specify that *g’’*(*x0*)=0 and *g’’*(*xn*)=0 to produce a *natural cubic spline*.

Or, we can specify ghat *g’*(*x0*)=α and *g’*(*xn*)=*β* to produce a *clamped cubic spline.*

*Natural cubic spline*:

The set of equations for *n* data points can be put in matrix form as follows:



Now, solve this set of linear equations and use expression for *F*(*x*) to find estimates of *f*(*x*).

*Clamped cubic spline*:

For point *x0*, we know that: 

and at *xn*, we have: 

The matrix equation is then:



Now, solve this set of linear equations and use expression for *F*(*x*) to find estimates of *f*(*x*).

*Natural cubic spline example*:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *i* | 0 | 1 | 2 | 3 | 4 |
| *xi* | 1 | 4 | 6 | 9 | 10 |
| *f*(*xi*) | 4 | 9 | 15 | 7 | 3 |

We want to approximate *f*(5).

First, set *g’’*(1)=*g’’*(10)=0.

Now, for *i*=1: 

For *i*=2: 

For *i*=3: 

Solving simultaneously: 

Substitution yields: *F1*(5)=12.56932

Note: once we have all the *g’’*(*xi*) values, we just pick the appropriate interval for the desired *x*.

# Ch. 5: Roots of Equations

The usual root solving problem involves an equation such as , which gets manipulated to  and we search for values of *x* that satisfy the equation. These values of *x* are the roots of the equation; that is, where *f*(*xroot*)=0.

*f*(*x*)

*x*

*0*

*a*

*b*

## 5.1 Bisection

Consider the simplest case where *f*(*x*) has only one real root in the interval *a*≤*x*≤*b*. Bisect the interval *xm*=(*a*+*b*)/2 and compute *f*(*xm*)\**f*(*b*). If the product is positive, then the root must be in the interval *a*<*x*<*xm* if negative *xm*<x<*b*. Select the interval which contains the root, biset it and repeat the entire procedure. This works for a single root, but if multiple roots exist, bisection gets very complicated.

## 5.2 Newton’s Method (Newton-Raphson)

Consider a point *x*0, which is not a root of *f*(*x*), but is “reasonably close” to a root. Expand *f*(*x*) as a Taylor series:



If *f*(*x*) =0, then *x* must be a root and the right-hand side constitutes an equation for the root *x*. Set rhs equal to zero and keep only two terms:



Now *x* represents an improved estimate of the root, and can replace *x0* to yield an even better estimate of the root on the next iteration:



*Newton’s method algorithm*:

*i*) input *x0*, *ϵ*, *ii*) *x←x0*, *iii*) *δ*←*f*(*x*)/*f’*(*x*), *iv*) *x*←*x*+*δ*, *v*) |*δ*|*<ϵ*?(root←x):(step *iii-v*)

## 5.3 Modified Newton’s Method

Let . Now, *u*(*x*) will have same roots as *f*(*x*). Next, apply *Newton’s Method* to *u*(*x*):

 where

Note: the *Modified Newton’s Method* is useful for functions with multiple roots.

## 5.4 The Secant Method

The secant method is a modification to the conventional *Newton’s method* with the derivative replaced by a difference expression:



*Secant method algorithm* *to find roots of f(x)=0*.

1. input values for *x0*, *x00*, and *ϵ*.
2. Calculate *δ*=*x0*-*x00*
3. Set *x*=*x0*
4. Compute *fold*=*f*(*x00*)
5. Compute *fnew*=*f*(*x*)
6. Compute *δ*=-*fnew*/[(*fnew*-*fold*)/*δ*]
7. Set *x*=*x*+*δ*
8. If |*δ*|<*ϵ*: return *x*
9. If |*δ*|>*ϵ*: *fold*=*fnew* and repeat steps ***v*)** to ***ix*)**

# Ch. 6 The Solution of Simultaneous Linear Algebraic Equations and Matrix Inversion

## Matrix terminology:

 is a *column matrix* (also a vector),  is a *row matrix*

 is a 4×4 square matrix with a *main diagonal*  of *c11, c22, c33, c44.*

The square matrix is symmetric if *cij*=*cji*.

 is an *upper triangular* matrix, is a *banded matrix*

 is the *identity matrix*

## Basic Matix operations:

*Addition*: ; A+B=B+A; A-B=-B+A

*Multiplication*:  (n×m ⋅ m× w yields n×w matrix); AI=A; AB≠BA in general

## Matrix Representation and Formal Solution of Simultaneous Linear Equations:

 can be expressed as:  or 

Ways to solve *CX*=*R* are:

## Cramer’s rule:

1.  where *Ck* is *C* with its *kth* column replaced by *R*. det(C) means determinant of C
2.  where *C-1* is the inverse of *C*

## Gaussian elimination

**Ax**=**b → A-1Ax=A-1b → x=A-1b**

To reduce a matrix **A** to echelon form, we perform the basic row operations of:

1. exchange rows
2. multiply row by a scalar
3. add scalar multiple of another row to replace a row

Goal is to get 1’s along the matrix diagonal

*Gauss-Jordan Method*

Use Gaussian elimination the put ***A*** in echelon form and then use rule iii) to make all other values in a column 0 except for along the body diagonal.

Observation: **A-1A = I**

If I use Gauss-Jordan method on matrix [**A**|**I**], I will find the result to be [**I**|**A**-1]

## Gauss-Sidel Iteration:

Consider the example of three equations solved for *x1*, *x2*, and *x3*,:

→ 

Algorithm for Gauss-Sidel for ***Ax***=***b***:

1. Create an augmented matrix ***M***=[***A***|***b***] such that ***M*** is *m*×*n* (generally *n*=*m+1*)
2. Exchange rows of ***M*** so that the largest numbers (absolute value) occur along body diagonal.
3. Guess initial values for *x1*, *x2*, *…xm.*
4. Calculate new *xi* by
5. With updated *xi*, calculate *xi+1* by **iv)**. After updating *xm*, one iteration is complete
6. Repeat steps **iv)** & **v)** for desired number of iterations or until change in *xi* is sufficiently small.
7. Return ***x***.

# Ch. 7 Least-Squares Curve Fitting

Consider a function *f*(*x*) that is to be approximated with a function *g*(*x*). The approximating function distance from the true function can be calculated as: 

If data is available at discrete *x* coordinates (*xi*) and there are *n* such coordinates, then *d*(*x*) is minimized in the least-squares sense if:  is minimized. Most commonly, we use a polynomial of degree *l* for the approximating function: . Now we have:



Note that *E* is minimized by varying the (*l*+1) coefficients. Note, setting the partial derivative of *E* w.r.t. each coefficient equal to zero, accomplishes the minimization.

etc.

The complete set of simultaneous linear equations in the coefficients of the polynomial is:



Which, can be solved by Gauss-Jordan elimination to find the a vector and hence *g*(*x*)

# Ch. 8: Numerical Integration

## 8.1 The Trapezoidal Rule

*f*(*x*)

*x*

*0*

*a*

*b*

Δ *x*

Consider an integrable function *f*(*x*) on the interval *a*≤*x*≤*b*. We wish to find: .

We divide the interval into *n* equal subintervals with width of *Δx*=(*b-a*)/*n* and estimate the area under each interval by:

 and 

*f*(*x*)

*x*

*0*

*xj*-1

*xn*

*fj-1*

*fj*

*fi+1*

*xj*

Thus in the interval (*xj-1*≤*x*≤*xj+1*):



So for the interval *a*≤*x*≤*b*:



Note: this method implicitly uses a linear interpolation between points to estimate *f*(*x*).

If the indefinite integral is defined as:  and *xj* is located at the dividing line between two panels, then *I*(*xj*) is the area under *f*(*x*) from *x*=*a* to this dividing line. The quantity *I*(*xj*+1) is then composed of this area plus the area of one more panel. Assuming that *I*(*x*) is analytic, then *I*(*xj*+1) can be obtained from the Taylor series expansion about *xj*:



Note: , etc.

Now, 

We can estimate the first derivative with a simple forward difference representation:



And substitute this into the integral expression and collect terms:



The single panel integral approximation is:



Thus:



The last term can be represented as:

, where *a*≤*x̅*≤*b* such that 

And the integral becomes:



We can estimate *f’’*(*x̅* ) as: 



This is the *trapezoidal rule with end correction*.

## 8.2 Simpson’s Rule

Instead of a linear interpolation, what if we use a parabolic arc interpolation function. Consider the following expansion of the integral as a function of *x*:



and



Subtracting:



Using a central difference for *f’’*(*xj*):



Substituting into the previous equation to get area between two panels *xj-1* and *xj+1*:



For the integral over the interval *a*≤*x*≤*b* if *Dj*=*I*(*xx*+1)-*I*(*xj*-1):

 Note: just odd so we don’t double count and this requires number of panels to be even.

If we sum for all pairs of panels then:





Or

