CS 257: Numerical Methods

Final Exam Study Guide
Version 1.00

Created by Charles Feng
http://www.fenguin.net

Contents

1 Introductory Matter

1.1 Calculus Review

Taylor polynomials:

$$f(x) \approx g(x) = \sum_{i=0}^{n} \frac{f^{(i)}(x_0)}{i!}(x - x_0)^i$$

$$\text{Degree 3: } g(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{f''(x_0)}{2}(x - x_0)^2 + \frac{f^{(3)}(x_0)}{6}(x - x_0)^3$$

Limit definition of derivative: $f'(x) = \lim_{\Delta x \to 0} \frac{f(x + \Delta x) - f(x)}{\Delta x}$

Mean Value Theorem: $f'(\theta) = \frac{f(b) - f(a)}{b - a}$

1.2 Linear Algebra Review

A matrix equation can be represented by several different ways:

$$\text{Matrix equation: } \begin{bmatrix} 3 & -1 \\ -3 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$\text{System of equations: } \begin{cases} 3x + y = 1 \\ -3x - y = -1 \end{cases}$$

$$\text{Linear combination of vectors: } x \begin{bmatrix} 3 \\ -3 \end{bmatrix} + y \begin{bmatrix} 1 \\ -1 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

If S is a set of n vectors, then S is *linearly independent* if there is no nonzero linear combination of elements of S that produces a zero vector. The *dimension* of a vector set S is the maximum number of linearly independent vectors in S.

A $n \times n$ matrix $\mathbf{A}$ is *non-singular* if and only if it has the following properties:

1. Its *rank* is n; in other words, it has n linearly independent columns.
2. It has a *multiplicative inverse* $\mathbf{A}^{-1}$ which satisfies the equation $\mathbf{A}\mathbf{A}^{-1} = \mathbf{A}^{-1}\mathbf{A} = \mathbf{I}$.
3. Its *determinant* is nonzero.

A *symmetric positive definite* matrix has the following properties:

1. $\mathbf{A}^T = \mathbf{A}$
2. $\mathbf{x}^T\mathbf{A}\mathbf{x} > 0$ for $\forall \mathbf{x} \in \mathbb{R}^n$ s.t. $\mathbf{x} \neq 0$

1.3 Floating-Point Numbers

A floating-point system is described using four parameters: t, β, L, and U. These are defined as follows:

1. t: the number of digits of precision.
2. β: the base the numbers are represented in.
3. L: the lower bound on the range of allowable exponents.
4. U: the upper bound on the range of allowable exponents.

A *normalized* floating-point system is one with only one digit before the decimal point.

Any floating-point number in our system is represented in the following form (where $0 \leq d_i \leq \beta - 1$ and $E \in [L, U]$):

$$\pm \left(d_0 + \frac{d_1}{\beta} + \frac{d_2}{\beta^2} + ... + \frac{d_{t-1}}{\beta^{t-1}} \right) \beta^E$$

There are several characteristics of our floating-point system:

1. The *overflow level* (OFL) is the largest number that can be represented, and is defined by $\beta^{U+1}(1 - \beta^{-t})$.
2. The *underflow level* (UFL) is the smallest number that can be represented, and is defined by β^L.
3. The *unit roundoff value* or *machine epsilon* is the smallest number that will not be lost by rounding when we add it to 1, and is defined by $\frac{1}{2}\beta^{1-t}$.
4. The total number of representable floating point numbers is $2(\beta - 1)(\beta)^{t-1}(U - L + 1) + 1$.

1.4 Error

Forward error: $\frac{|\hat{y} - y|}{|y|}$ where y is the actual answer and $\hat{y}$ is the one we got.

Backward error: $\frac{|\hat{x} - x|}{|x|}$ where x is the given input value and $\hat{x}$ is the input value obtained by backtracking from our answer.

1.5 Norms

A vector norm is a function $|| \cdot || : \mathbb{R}^n \rightarrow \mathbb{R}^+$ that satisfies the following properties for all $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$ and $a \in \mathbb{R}$:

1. $||\mathbf{u}|| \geq 0$
2. $||\mathbf{u}|| = 0 \iff \mathbf{u} = 0$
3. $||a\mathbf{u}|| = |a| \cdot ||\mathbf{u}||$
4. $||\mathbf{u} + \mathbf{v}|| \leq ||\mathbf{u}|| + ||\mathbf{v}||$

The 1-norm is defined as $||\mathbf{u}||_1 = \sum_{i=1}^{n} |u_i|$, or the sum of the absolute values of the elements. The 2-norm is the same as the magnitude of the vector and is defined as $||\mathbf{u}||_2 = \sqrt{\sum_{i=1}^{n} |u_i|^2}$. The infinity norm, $||\mathbf{u}||_\infty$, is the largest element in the vector.

A matrix norm is a function $|| \cdot || : \mathbb{R}^{m \times n} \rightarrow \mathbb{R}^+$ that satisfies the following properties for all $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{m \times n}$ and $c \in \mathbb{R}$:

1. $||\mathbf{A}|| \geq 0$
2. $||\mathbf{A}|| = 0 \iff \mathbf{A} = 0$
3. $||c\mathbf{A}|| = |c| \cdot ||\mathbf{A}||$
4. $||\mathbf{A} + \mathbf{B}|| \leq ||\mathbf{A}|| + ||\mathbf{B}||$
5. $||\mathbf{AB}|| \leq ||\mathbf{A}|| \cdot ||\mathbf{B}||$
6. $||\mathbf{Au}|| \leq ||\mathbf{A}|| \cdot ||\mathbf{u}||$

The Frobenius norm is defined for a $m \times n$ matrix $\mathbf{A}$ to be

$$||\mathbf{A}|| = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} |a_{ij}|^2}$$

The 1-norm is the maximum absolute column sum of the matrix. The ∞-norm is the maximum absolute row sum of the matrix.

1.6 Conditioning

The conditioning of a function measures its sensitivity to changes in input. Those that are highly sensitive to input are called *ill-conditioned*; those that are not that sensitive are called *well-conditioned*. It is better when the function is flat and worse when the function is steep.

The condition number is defined as the relative change in the solution over the relative change in input, or the relative forward error over the relative backward error.

The condition number of a matrix is can be found using any matrix norm, and is as follows:

$$\text{cond}_p(\mathbf{A}) = ||\mathbf{A}||_p \cdot ||\mathbf{A}^{-1}||_p$$

A matrix condition number satisfies several properties:

1. For any matrix $\mathbf{A}$, $\text{cond}(\mathbf{A}) \geq 1$.
2. For the identity matrix, $\text{cond}(\mathbf{I}) = 1$.
3. For any matrix $\mathbf{A}$ and scalar c, $\text{cond}(c\mathbf{A}) = \text{cond}(\mathbf{A})$.
4. For any diagonal matrix $\mathbf{D} = \text{diag}(d_i)$, $\text{cond}(\mathbf{D}) = (\max|d_i|)/(\min|d_i|)$.

2 Linear Systems

2.1 LU Factorization

We attempt to convert a matrix ($\mathbf{A}$) into the product of a lower triangular matrix ($\mathbf{L}$) and a upper triangular matrix ($\mathbf{U}$) which makes solving the system much easier.

How to solve a linear system $\mathbf{Ax} = \mathbf{LUx} = \mathbf{b}$: (forward-backward substitution)

1. Find $\mathbf{L}$ and $\mathbf{U}$.
2. Let $\mathbf{y} = \mathbf{Ux}$ and solve the equation $\mathbf{Ly} = \mathbf{b}$.
3. Then solve the equation $\mathbf{Ux} = \mathbf{y}$ for $\mathbf{x}$ which gives you the answer.

2.2 Gaussian Elimination

We will find the LU factorization using this method.

Say we have a $n \times n$ matrix $\mathbf{A}$. For each column i in the first $n-1$ columns, we will multiply the matrix by an *elementary elimination matrix* $\mathbf{M}_i$ which will turn all values below the diagonal of $\mathbf{A}$ into zero, yielding an intermediate matrix $\mathbf{A}_i$. This process will finally form an upper triangular matrix which is $\mathbf{U}$. Then to find $\mathbf{L}$ we will multiply all the $\mathbf{M}_i$ we used in reverse order, then find the inverse of the resulting matrix ($(\mathbf{M}_2\mathbf{M}_1)^{-1} = \mathbf{M}_1^{-1}\mathbf{M}_2^{-1}$). Therefore, for a 3×3 matrix, the LU factorization will be given by

$$\mathbf{A} = \mathbf{LU} = \mathbf{M}_1^{-1}\mathbf{M}_2^{-1}\mathbf{A}_2$$

On the elementary elimination matrix, the values below the diagonal on the jth column can be calculated simply by $-\mathbf{A}_{jj}/\mathbf{A}_{ij}$ where i is the row we want.

To get results with better conditioning, we can use *pivoting*. Its results differ from the standard results in that it permutes $\mathbf{L}$ so it isn't strictly triangular anymore (the diagonal is not all ones).

Partial pivoting involves multiplying the matrix by a *permutation matrix* $\mathbf{P}_i$—an identity matrix with the rows rearranged—in order to interchange rows. For each row i, we want the maximum value in the ith column at the location (m, m) so we switch around

rows to do this. Then we calculate and multiply the matrix by an elementary elimination matrix as before. Our LU factorization will then finally be

$$\mathbf{A} = \mathbf{LU} = \mathbf{P}_1^T \mathbf{M}_1^{-1} \mathbf{P}_2^T \mathbf{M}_2^{-1} \mathbf{A}_2$$

Complete pivoting involves multiplying the matrix by two permutation matrices—$\mathbf{P}_i$ and $\mathbf{R}_i$—the former on the left side to interchange rows and the latter on the right to interchange columns. For each column, we want the element with largest magnitude on or below or to the right of the diagonal element in the column; for example, for the first column we want the element with largest magnitude in the whole matrix on the $(1, 1)$ position. Then, we apply an elementary elimination matrix as before to eliminate lower triangular values. Complete pivoting has better conditioning than partial pivoting but it results in a permuted $\mathbf{U}$ which isn't upper triangular anymore. Our LU factorization will be

$$\mathbf{A} = \mathbf{L}\hat{\mathbf{U}} = \mathbf{P}_1^T \mathbf{M}_1^{-1} \mathbf{P}_2^T \mathbf{M}_2^{-1} \mathbf{A}_2 \mathbf{R}_2^T \mathbf{R}_1^T$$

The running time of Gaussian elimination is $\mathcal{O}(n^3/3)$.

2.3 Gauss-Jordan Method

This helps find the inverse of a square $n \times n$ matrix $\mathbf{A}$. We will append an identity matrix of the same size to the right side of $\mathbf{A}$ and then apply row operations to the $n \times 2n$ matrix to make the left side the identity matrix. After we are done, the right side will be $\mathbf{A}^{-1}$.

The Gauss-Jordan method's running time is $\mathcal{O}(n^3/2)$.

2.4 Cholesky Decomposition

For symmetric positive definite matrices, we can calculate an LU factorization that is also symmetric:

$$\mathbf{A} = \mathbf{L}\mathbf{L}^T$$

This is faster than the other LU factorizations; its running time is $\mathcal{O}(n^3/6)$.

2.5 Error in Linear Systems

We will derive lower and upper bounds on the error of linear systems.

To find a lower bound, we will manipulate two equations: $\mathbf{A}\Delta\mathbf{x} = \Delta\mathbf{b}$ and $\mathbf{A}\mathbf{x} = \mathbf{b}$.

$$\begin{aligned}
\mathbf{A}\Delta\mathbf{x} &= \Delta\mathbf{b} \\
||\mathbf{A}\Delta\mathbf{x}|| &= ||\Delta\mathbf{b}|| \\
||\mathbf{A}||\,||\Delta\mathbf{x}|| &\geq ||\Delta\mathbf{b}|| \\
||\Delta\mathbf{x}|| &\geq \frac{||\Delta\mathbf{b}||}{||\mathbf{A}||} \\
||\Delta\mathbf{x}|| &\geq \frac{||\Delta\mathbf{b}||}{||\mathbf{A}||}
\end{aligned}$$

We will then work on the equation $\mathbf{Ax} = \mathbf{b}$:

$$\begin{aligned} \mathbf{Ax} &= \mathbf{b} \\ \mathbf{x} &= \mathbf{A}^{-1}\mathbf{b} \\ ||\mathbf{x}|| &= ||\mathbf{A}^{-1}\mathbf{b}|| \\ ||\mathbf{x}|| &\leq ||\mathbf{A}^{-1}||\,||\mathbf{b}|| \\ \frac{1}{||\mathbf{x}||} &\geq \frac{1}{||\mathbf{A}^{-1}||\,||\mathbf{b}||} \end{aligned}$$

We put these two equations together and get

$$\frac{||\Delta\mathbf{x}||}{||\mathbf{x}||} \geq \frac{||\Delta\mathbf{b}||}{||\mathbf{b}||\,||\mathbf{A}||\,||\mathbf{A}^{-1}||} = \frac{||\Delta\mathbf{b}||}{||\mathbf{b}||\,\text{cond}(\mathbf{A})}$$

To derive an upper bound on the error, we do the following:

We will firstly manipulate the equation $\mathbf{A}\Delta\mathbf{x} = \Delta\mathbf{b}$:

$$\begin{aligned} \mathbf{A}\Delta\mathbf{x} &= \Delta\mathbf{b} \\ \Delta\mathbf{x} &= \mathbf{A}^{-1}\Delta\mathbf{b} \\ ||\Delta\mathbf{x}|| &= ||\mathbf{A}^{-1}\Delta\mathbf{b}|| \\ ||\Delta\mathbf{x}|| &\leq ||\mathbf{A}^{-1}||\,||\Delta\mathbf{b}|| \end{aligned}$$

We will then work on the equation $\mathbf{Ax} = \mathbf{b}$:

$$\begin{aligned} \mathbf{Ax} &= \mathbf{b} \\ ||\mathbf{Ax}|| &= ||\mathbf{b}|| \\ ||\mathbf{A}||\,||\mathbf{x}|| &\geq ||\mathbf{b}|| \\ ||\mathbf{x}|| &\geq \frac{||\mathbf{b}||}{||\mathbf{A}||} \\ \frac{1}{||\mathbf{x}||} &\leq \frac{||\mathbf{A}||}{||\mathbf{b}||} \end{aligned}$$

We then put the two equations together and get an upper bound of

$$\frac{||\Delta\mathbf{x}||}{||\mathbf{x}||} \leq \frac{||\mathbf{A}||\,||\mathbf{A}^{-1}||\,||\Delta\mathbf{b}||}{||\mathbf{b}||} = \text{cond}(\mathbf{A})\frac{||\Delta\mathbf{b}||}{||\mathbf{b}||}$$

We can also find a bound on the error based on the relative residual, defined as

$$\frac{||\mathbf{b} - \mathbf{A}\hat{\mathbf{x}}||}{||\mathbf{A}||\,||\hat{\mathbf{x}}||}$$

We begin with the numerator:

$$||\Delta\mathbf{x}|| = ||\hat{\mathbf{x}} - \mathbf{x}|| = ||\mathbf{A}^{-1}(\mathbf{A}\hat{\mathbf{x}} - \mathbf{b})|| = ||-\mathbf{A}^{-1}\mathbf{r}|| \leq ||\mathbf{A}^{-1}||\,||\mathbf{r}||$$

Then we add the denominator and manipulate further:

$$\frac{||\Delta\mathbf{x}||}{||\hat{\mathbf{x}}||} \leq \frac{||\mathbf{A}^{-1}||\,||\mathbf{r}||}{||\hat{\mathbf{x}}||} = \frac{||\mathbf{A}||\,||\mathbf{A}^{-1}||\,||\mathbf{r}||}{||\mathbf{A}||\,||\hat{\mathbf{x}}||} = \text{cond}(\mathbf{A})\frac{||\mathbf{r}||}{||\mathbf{A}||\,||\hat{\mathbf{x}}||}$$

2.6 Linear Least Squares

This method is based on the *normal equations*, or $\mathbf{A}^T\mathbf{A}\mathbf{x} = \mathbf{A}^T\mathbf{b}$. Because the normal equations can be solved using Cholesky factorization ($\mathbf{A}^T\mathbf{A}$ is symmetric positive definite) it results in a fast operation time.

The normal equations are most often used in polynomial approximation where a *Vandermonde matrix* in which columns are formed by increasing powers of input values (starting from the 0th power).

3 Nonlinear Systems

3.1 Interval Bisection Method

The Intermediate Value Theorem dictates that if we have a continuous function on a specified interval, then we are guaranteed to hit every value between the endpoints. This enables the interval bisection method to guarantee a solution.

Essentially for the interval bisection method, we find two points a and b with one function value above zero and one below zero. Our initial error will then be $|f(a) - f(b)|$. Then, for each step, we halve the error by comparing the function value at the midpoint $c = \dfrac{a+b}{2}$ with $f(b)$: if $f(b) \cdot f(c)$ is less than zero then we set the new range to be from c to b and if it is greater than zero we set the new range to be from a to c.

Although the interval bisection method has linear convergence for single roots, it is unable to find multiple roots at all.

3.2 Fixed-Point Methods

A point α is called a *fixed point* of $g(x)$ if it satisfies $g(\alpha) = \alpha$. For a function $f(x)$, we will try to manipulate it so that we get a function $g(x) = x$ (an example is adding x to both sides of $f(x) = 0$ and then setting $g(x) = f(x) + x$). Then, we iterate from an initial guess using the scheme $x_{k+1} = g(x_k)$.

If the magnitude of the derivative of the fixed-point function at the root is less than 1, and we have a starting guess close enough to the root, the fixed-point method will converge. Otherwise, the method will generally diverge.

This method enjoys quadratic convergence at points where $g'(\alpha) = 0$ and linear convergence everywhere else.

3.3 Newton's and Secant Methods

The iterative scheme of Newton's method is

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

Newton's method converges quadratically for simple roots and linearly for multiple roots.

The secant method is similar to Newton's method except instead of the derivative, it uses an approximation for the derivative and so its iterative scheme is

$$x_{k+1} = x_k - \frac{f(x_k)(x_k - x_{k-1})}{f(x_k) - f(x_{k-1})}$$

The secant method converges superlinearly ($n \approx 1.618$) and requires two close initial guesses to begin.

4 Interpolation

Interpolation is good for several things, including:

1. Finding a smooth curve through data points.
2. Reading between lines in a table.
3. Differentiating or integrating tabular data.
4. Replacing a complicated function with a simple one.

Basically, our interpolating function $f(t)$ can be described as a linear combination of a set of functions $\{\phi_j\}_{j=1\to n}$:

$$f(t_i) = \sum_{j=1}^{n} c_j \phi_j(t_i) = y_i \qquad i = 1 \to m$$

If we write out the above equations, we will see that we get a matrix equation $\mathbf{Ac} = \mathbf{y}$ where $a_{ij} = \phi_j(t_i)$ and $\mathbf{A}$ has dimensions $m \times n$. Usually we will use $m = n$ to find a unique solution.

4.1 Monomial Basis

We will set $\phi_i(t) = t^i$ with i ranging from 0 to n. We set up a Vandermonde matrix and use the normal equations with LU factorization to solve for the coefficients c_i, which will be on the order of $n^3/3$ multiplications.

Evaluating an interpolant would require $\dfrac{n(n+1)}{2}$ multiplications on worst case, $2n-1$ multiplications if we reuse known values of t^{k-1} to calculate t^k, and finally n multiplications if we use Horner's nested evaluation scheme:

$$f(t) = c_0 + t(c_1 + t(c_2 + t(...(c_{n-1} + c_n t)...)))$$

Adding new data points is very difficult as we will have to recalculate all of the coefficients.

4.2 Lagrange Basis

The formula for the jth Lagrange basis function is

$$\ell_j(t) = \frac{\Pi_{k=1,\ k\neq j}^{n}(t - t_k)}{\Pi_{k=1,\ k\neq j}^{n}(t_j - t_k)}$$

What's interesting is that when we plug in the basis values into the matrix, it results in an identity matrix. So the Lagrange basis functions are already our ϕs and so we can find the interpolating function easily. Moreover, the coefficients are merely the y-values of the given data poitns.

Therefore, finding the interpolating function is very easy with the Lagrange basis but evaluating takes n^2 multiplications and thus is a lot more than the monomial basis.

If we wish to add additional data points, we merely have to add a new term to the end and update each preceding term with the data point.

4.3 Newton Basis

The Newton basis functions are defined as

$$\eta_j(t) = \Pi_{k=1}^{j-1}(t - t_k)$$

The matrix for finding coefficients will be a lower triangular matrix and will therefore take n^2 multiplications to solve using forward substitution. We can use Horner's method for evaluating an interpolant so we only need n multiplications to do so.

To add a new data point to the Newton basis, we need to only add the term $c_{k+1}\eta_{k+1}(t)$ to the interpolating function, with $eta_{k+1}(t)$ defined as above and c_{k+1} being equal to $\frac{y_{k+1} - p_k(t_{k+1})}{\eta_{k+1}(t_{k+1})}$.

4.4 Error Bound

We can bound the error with the equation

$$|f(t) - p_n(t)| \leq \frac{Mh^{n+1}}{4n}$$

where $|f^{n+1}(\theta)| < M$ for $\theta \in [t_i, t_n]$ and $h = \max t_{i+1} - t_i \ : \ i = 1, ..., n$.

4.5 Piecewise Polynomial Interpolation

As the degree of a polynomial interpolant increases, the likelihood that the polynomial will oscillate does as well; this is bad for equally-spaced interpolating curves. Moreover, there are other functions such as Runge's function that cannot be approximated very well by said curves either.

A *linear piecewise interpolant* is formed by connecting the dots using straight lines. The derivative isn't continous at the knots which is unfortunate. Monotonicity is favored although it is very lacking in smoothness and only average for periodicity.

Hermite interpolation guarantees a continuous first derivative on the interval (t_1, t_n). Functions generated by this method are somewhat smooth and exhibit monotonicity.

A *cubic spline interpolant* guarantees continous first and second derivatives on the itnerval (t_1, t_n) and so are a subset of Hermite interpolants. They are made for smoothness but may not exhibit monotonicity in some cases due to more fluctuating function values.

The equations for a Hermite or cubic spline interpolant are as follows:

1. $q_i(t_i) = y_i$ for $i = 1 \to n - 1$
2. $q_i(t_{i+1}) = y_{i+1}$ for $i = 1 \to n - 1$
3. To make the first derivative continuous: $q_i'(t_{i+1}) = q_{i+1}'(t_{i+1})$ for $i = 1 \to n - 2$

Using the equations above, we have n degrees of freedom for a simple Hermite interpolant. A cubic spline interpolant also makes the second derivatives continuous:

$$q_i''(t_{i+1}) = q_{i+1}''(t_{i+1})$$

There are several types of cubic spline interpolation, each defining the last two equations so we can have a unique solution:

1. Natural cubic spline: set the second derivative values to be zero at the endpoints.
2. Force equality of the first and second derivatives at the endpoints; this would be used for a periodic spline.

5 Numerical Quadrature

In this section we will discuss how to approximate definite integrals. There are several definitions we need to know:

1. The *degree* of a quadrature rule is the maximum degree of polyonmials that the quadrature rule integrates exactly.
2. A *Newton-Cotes* quadrature rule has evenly-spaced nodes.
3. A *Gaussian* quadrature rule has the maximum possible degree a quadrature rule can have for the number of nodes used.

4. A *simple* quadrature rule is one rule that is applied over the entire interval of integration. A *composite* quadrature rule is formed by applying a simple quadrature rule on subintervals of the interval of integration.

5.1 Midpoint Rule

This rule has degree 1 and is defined by

$$I(f) = \int_a^b f(x)\,dx \approx f\left(\frac{a+b}{2}\right)(b-a) = M(f)$$

5.2 Trapezoid rule

This rule has degree 1 and is defined by

$$I(f) = \int_a^b f(x)\,dx \approx \frac{b-a}{2}(f(a)+f(b)) = T(f)$$

5.3 Simpson's rule

This rule has degree 3 and is defined by

$$I(f) = \int_a^b f(x)\,dx \approx \frac{b-a}{6}\left(f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right) = S(f)$$

5.4 Error analysis

Applying a Taylor polynomial of our function $f(x)$ around the midpoint $c = (a+b)/2$, we find that the error of the midpoint rule is about $E(f) + F(f)$ where $E(f)$ and $F(f)$ stand for the first and second error terms respectively.

Similarly, the error in the trapezoidal rule is about $2E(f) + 4F(f)$ and so it is less accurate than the midpoint rule.

Simpson's rule completely eliminates the $E(f)$ term (it's basically $\frac{2}{3}M(f) + \frac{1}{3}T(f)$) and so has much higher accuracy than the other two rules.

5.5 Gaussian Quadrature

For a n-point quadrature rule we select n points $x_1, x_2, ..., x_n$ with n weights $w_1, w_2, ..., w_n$ such that we can compute an integral $\int_a^b f(x)\,dx$ for every polynomial $f(x)$ of degree up to $2n-1$ using discrete series $\sum_{i=1}^{n} w_i f(x_i)$. We use a method of undetermined coefficients to solve for the $2n$ undetermined values that result from said integrals.

If we want to change the integration interval from (α, β) to (a, b) we can use the function

$$\int_a^b g(t)\,dt = \frac{b-a}{\beta-\alpha}\sum_{i=1}^{n} w_i g(t_i)$$

where

$$t_i = \frac{(b-a)x_i + a\beta - b\alpha}{\beta-\alpha}$$

5.6 Composite Quadrature

Basically we split the integration interval $[a, b]$ into k subintervals of the same size with step size $h = \frac{b-a}{k}$.

The midpoint method will then be defined as

$$h \sum_{j=1}^{k} f\left(\frac{x_{j+1} + x_j}{2}\right)$$

The trapezoidal method will become

$$h\left(\frac{1}{2}f(a) + f(x_1) + ... + \frac{1}{2}f(b)\right)$$

Simpson's method will become

$$\frac{h}{6}\left(f(a) + 4f\left(\frac{a+x_1}{2}\right) + 2f(x_1) + 4f\left(\frac{x_1+x_2}{2}\right) + ...\right)$$

If we know what error tolerance we need then we can use *adaptive quadrature* or *selective refinement* in which we continue subdivisions just on subintervals that don't meet the tolerance.

5.7 Higher-Order Quadrature

The *Monte Carlo* method is feasible for approximating integrals when other methods become infeasible. It works by sampling random points, averaging those function values, and the multiplying the mean value by the area or volume of the domain. The error of the Monte Carlo method goes to zero on the order of $1/\sqrt{n}$ where n is the number of points sampled.

5.8 Numerical Differentiation

We can come up with several approximations for derivatives by finding Taylor series and performing operations on them.

The Taylor series expansion of a function f around a point a is

$$f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)(x-a)^2}{2} + \cdots$$

Another way to write this (you derive this second expansion by substituting into the first expansion above with $a = y$ and $x = y+h$) is

$$f(y+h) = f(y) + f'(y)h + \frac{f''(y)h^2}{2} + \frac{f'''(y)h^3}{6} + \cdots$$

If we truncate this formula, we have

$$f(y+h) \approx f(y) + f'(y)h$$

which, when solved for $f'(y)$ gives

$$f'(y) \approx \frac{f(y+h) - f(y)}{h}.$$

This is the formula as we have already been using, just written with different variables. It is called the forward difference formula for the first derivative.

Now consider the expansion

$$f(y-h) = f(y) - f'(y)h + \frac{f''(y)h^2}{2} - \frac{f'''(y)h^3}{6} + \cdots$$

If we truncate this formula and solve for the first derivative, we find

$$f'(y) \approx \frac{f(y-h) - f(y)}{-h} = \frac{f(y) - f(y-h)}{h}.$$

This is called the backward difference formula for the first derivative.

When we subtract the two formulas we derived above, we obtain yet another approximation to the first derivative:

$$f'(y) \approx \frac{f(y+h) - f(y-h)}{2h}$$

This is called the centered difference formula for the first derivative. This approximation is generally better than the forward and backward differences.

If we add the two formulas above, we obtain

$$f(y+h) + f(y-h) = 2f(y) + h^2 f''(y) + \frac{h^4 f^{(4)}(y)}{12} + \cdots$$

We can truncate this and use it to approximate the second derivative

$$f''(y) \approx \frac{f(y+h) - 2f(y) + f(y-h)}{h^2}$$

This is the centered finite-difference formula for the second derivative.

6 Ordinary Differential Equations

Explicit ODEs can be written in the form $y^{(k)} = f(t, y, y', ..., y^{(k-1)})$. An ODE that contains a $y^{(k)}$ term on the right side is called an *implicit* ODE.

We don't want to work with ODEs with derivatives higher than first derivatives and so we will add extra variables (such as setting $w = y'$ and working from there) to produce systems of differential equations using only up to first derivatives. We solve these systems of equations using vectors of variables instead of just variables at each step,

We must consider several things to see whether we have an accurate numerical solution to our problem. These include:

1. The stability of the solution.
2. The stability of the numerical method.
3. The consistency of the numerical method.
4. The order of the numerical method.

A *stable* solution to an ODE means that solutions that start close together remain close together. An *unstable* solution diverges over time from the other solutions. *Asymptotically stable* solutions converge toward each other over time and tend to damp out error since slight approximations along the way will be negligible in the final solution.

6.1 Euler's Method

Euler's method is a recursive approximation defined by the formula

$$y_{k+1} = y_k + h_k f(t_k, y_k)$$

The h_k value is called the *step size* and will be a constant. Usually, when the step size is smaller, the approximation at each step is more accurate.

Euler's method has accuracy of order 1. The proof is as follows:

We begin by looking at a Taylor series expansion of $y(t+h)$.

$$y(t+h) = y(t) + hy'(t) + \frac{h^2 y''(t)}{2} + \frac{h^3 y'''(t)}{6} + \cdots$$

Next we assume our step size is relatively small ($h_k < 1$). Given this assumption, the terms with higher powers of h become negligible, and we write our equation as

$$y(t+h) = y(t) + hy'(t) + \mathcal{O}(h^2)$$

Then, substituting with $h = h_k$ and $t = t_k$, we find

$$y(t_k + 1) = y(t_k) + h_k y'(t_k) + \mathcal{O}(h^2)$$

Finally, invoking the ODE to substitute for y', we find

$$y(t_k + 1) = y(t_k) + h_k f(t_k, y_k(t_k)) + \mathcal{O}(h^2)$$

Now we are ready to involve Euler's method, and we subtract the equation above from Euler's method, which is

$$\hat{y}_{k+1} = \hat{y}_k + h_k f(t_k, \hat{y}_k)$$

The result of subtraction is

$$\hat{y}_{k+1} - y(t_k + 1) = \hat{y}_k - y(t_k) + h_k(f(t_k, \hat{y}_k) - f(t_k, y_k(t_k))) + \mathcal{O}(h^2)$$

Since we are looking for local error, we assume our approximate ($\hat{y}$) solutions are exact at time t_k. That means $\hat{y}_k = y(t_k)$, and most of the terms on the right-hand side above cancel out, leaving us

$$\ell_{k+1} = \mathcal{O}(h^2)$$

Thus, our local error is of order h^2 and Euler's method has accuracy of order $p - 1 = 2 - 1 = 1$.

For Euler's method to be stable for a differential equation $y' = \lambda y$, there must be a restriction on h_k:

$$h_k \leq \frac{-2}{\lambda}$$

6.2 Backwards Euler's Method

This method is similar to Euler's method except it is an implicit method. It is defined by the formula

$$y_{k+1} = y_k + h_k f(t_{k+1}, y_{k+1})$$

The backward Euler's method also has an order of accuracy of 1.

6.3 Taylor Series Methods

Consider the Taylor expansion

$$y(t+h) = y(t) + hy'(t) + \frac{h^2}{2}y''(t) + \frac{h^3}{6}y'''(t) + \cdots$$

If we take the first two terms on the right side, We can rewrite it as

$$y_{k+1} = y_k + hy'_k = y_k + hf(t_k, y_k)$$

which gives us Euler's method.

In order to get higher order methods we will need to take derivatives of our ODE using chain rule and then substituting the ODE back into said derivatives to obtain higher order derivatives written solely in terms of f. Then, we can substitute these back into our cropped Taylor expansion.

For example, the second-order Taylor series method is

$$y_{k+1} = y_k + hf(t_k, y_k) + \frac{h^2}{2}(f_t(t_k, y_k) + f_y(t_k, y_k)f(t_k, y_k))$$

6.4 Runge-Kutta Methods

Obviously, finding higher order Taylor series methods can get very complicated as one will have to take many derivatives. If we were to approximate said derivatives using one of the quadrature rules we presented earlier, it would make approximating solutions much easier.

We will now proceed to derive Heun's method, the Runge-Kutta method of order 2.

If we take the integral of y' and apply the Fundamental Theorem of Calculus, we find that

$$y_{k+1} - y_k = \int_{t_k}^{t_{k+1}} y'(t)\,dt$$

Substituting in with the ODE function and moving the y_k to the other side, we get:

$$y_{k+1} = y_k + \int_{t_k}^{t_{k+1}} f(t, y(t))dt$$

Now, in order to make finding the integral feasible, we apply a quadrature rule. Here we will apply the trapezoid rule. The result is:

$$y_{k+1} = y_k + \frac{h_k}{2}(f(t_k, y_k) + f(t_{k+1}, y_{k+1}))$$

Now, to make this method an explicit method, we find y_{k+1} using Euler's method. The final Heun's method, then, is given by:

$$\begin{aligned} y_{k+1} &= y_k + \frac{h_k}{2}(k_1 + k_2) \\ k_1 &= f(t_k, y_k) \\ k_2 &= f(t_{k+1}, y_k + h_k k_1) \end{aligned}$$