SFWR ENG 3X03

Teacher: Dr. Ned^2
Fall 2013

Numerical Methods Guy

Chapter 1

Types of errors

The focus in this course is determining the error margin in MATLAB. We will only focus on discretization errors.

Assuming u is an exact result and v is an approximation

Absolute error (A): $|u - v|$

Relative error (R): $\frac{|u-v|}{|u|}, u \neq 0$

Cancellation Error (R): when adding 2 numbers of similar absolute value, but opposite sign; this can be eliminated by rearranging your formula so only addition occurs

Overflow: when a number is too large (fatal)

Underflow: when a number is too small (usually rounded down to 0)

$\min_x f$: minimum value of f, with changing x

Other sources:

- **(A)** when y >> x, x + y
- **(A, R)** when |y| >> 1, xy
- **(A, R)** when |y| << 1, x/y

Condition Number

Forward error: difference between result and solution
Backward error: the difference between the value of x used to find the result and the value of x that would give the solution
Condition number:

Condition number = $\dfrac{\left|\dfrac{f(\hat{x})-f(x)}{f(x)}\right|}{\left|\dfrac{\hat{x}-x}{x}\right|} = \dfrac{\left|\dfrac{\hat{y}-y}{y}\right|}{\left|\dfrac{\Delta x}{x}\right|}$

Condition number = $\dfrac{\dfrac{|\Delta y|}{|y|} \leftarrow \text{relative forward error}}{\dfrac{|\Delta x|}{|x|} \leftarrow \text{relative backward error}}$

Summary: condition number = change in *solution*/change in *input*
Well-conditioned: low condition #
Ill-conditioned: high condition #

Horner's Method (or rule or scheme): nested polynomial form, i.e.

$$p_n(x) = c_0 + c_1 x + \cdots + c_n x^n \Rightarrow p_n(x) = \left(\cdots\left(\left(c_n x + c_{n-1}\right)x + c_{n-1}\right)x\cdots\right)x + c_0$$

Chapter 2 – Round-off Errors

Floating Point

Floating Point (FP) System (β, t, L, u)
β – base
t – number of digits
L – lower bound for exponent
u – upper bound for exponent

$$f(x) = x(1+\delta_x)$$
$$f(y) = y(1+\delta_y)$$
$$f(xy) = xy(1+\delta_{xy})$$

Rounding unit (a.k.a. machine epsilon): $\eta = \dfrac{1}{2}\beta^{1-t}$

flops: the number of elementary floating point operations necessary to compute something; elementary floating point operations are considered equal to each other in terms of time and can be any of addition, subtraction, multiplication, and division

Rounding

Types:

a) To nearest
b) Towards $+\infty$ (i.e. upper bound)
c) Towards $-\infty$ (i.e. lower bound)
d) Towards 0 / chopping

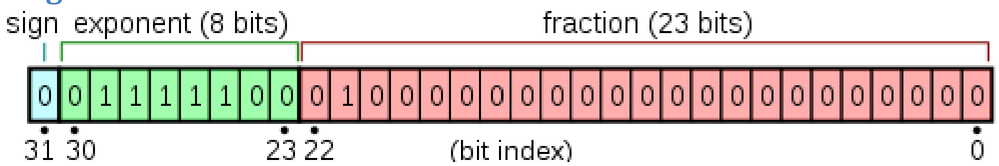
Ceiling function: $\lceil x \rceil$

Floor: $\lfloor x \rfloor$

IEEE Precision

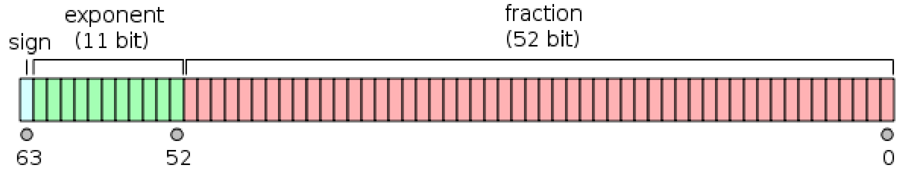
$+\infty$: from overflow of positives; maximum exponent, 0 fraction
$-\infty$: from overflow of negatives; same as $+\infty$, but with different sign
NaN: division by 0; maximum exponent; fraction is not 0

Single



A.K.A. short
Bias = 127

Double



A.K.A. long
Bias = 1023

Convergence

$$\left|x_{k+1} - x^*\right| \le C\left|x_k - x^*\right|^{\alpha}$$

$$\left|x_{k+1} - x^*\right| = \frac{m-1}{m}\left|x_k - x^*\right|, x^* = 0$$

Taylor Series

Rewrite a function, f(x) as an infinite sum of other functions.

$$f(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \ldots + \frac{f^{(n)}(a)}{n!}(x-a)^n$$

a is a constant that either you choose or it is given to you
It is easiest to do if you start off by finding the first couple derivatives and then plugging in *a*, so you can see how the function changes with each iteration.

e.g.)

$$f(x) = \ln x \qquad f(2) = \ln(2)$$
$$f'(x) = x^{-1} \qquad f'(2) = \frac{1}{2}$$
$$f''(x) = -1 \cdot x^{-2} \qquad f''(2) = -\frac{1}{4}$$
$$f'''(x) = 2 \cdot x^{-3} \qquad f'''(2) = 2\left(\frac{1}{8}\right)$$
$$f^{(4)}(x) = -3 \cdot 2x^{-4}$$

As you can see, it seems to have some sort of factorial coefficient

Taylor's Theorem

If you know the value of $f(x_0)$, but want the value of $f(x_0 + h)$:

$$f(x_0+h) = f(x_0) + hf'(x_0) + \frac{h^2}{2} f''(x_0) + \ldots + \frac{h^k}{k!} f^{(k)}(x_0) + \frac{h^{k+1}}{(k+1)!} f^{(k+1)}(\xi)$$

Where ξ is some point between x_0 and $x_0 + h$.

Chapter 3 – Solving Non-linear Equations

Find the roots.

Tolerance: a value that tells the computer what precision to stop computation
Absolute Tolerance: $|x_n - x_{n-1}| < \text{atol}$
Relative Tolerance: $|x_n - x_{n-1}| < \text{rtol} \cdot |x_n|$
$|f(x_n)| < \text{ftol} \leftarrow$ tolerance based on function

x^*: the root (or one of the roots)

Rate of Convergence: given $\rho = |g'(x^*)|$, where $0 < \rho < 1$, ?

Bisection Method

When given an upper and lower estimate for a number, find the midpoint between the two points. If the midpoint is negative, it becomes the new lower limit. If it is positive, it becomes the new upper limit. Keep doing this until the end of time…
This is used on graphing calculators.
It is linearly convergent.

Error

$$\underbrace{|c_n - c|}_{\text{error}} \le \frac{|b-a|}{2^n}$$

Newton's Method

It runs faster than bisection method. When given a single guess (x_0),

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

How it works:

1. Find the tangent line of the function at the point.
2. Find the point on the line that hits y = 0.
3. The point is your new x_0. Go to step 1.

One problem with this method is that sometimes the computation gets into an infinite loop, going back and forth endlessly between 2 points. Thus, you require another field, **maximum iterations**, to prevent infinite loops.

If you don't know the derivative of the function, you need to sub in the secant formula.
It is super-linearly convergent.

Secant Method

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

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

The order of convergence is the golden ratio, $\alpha = \frac{1+\sqrt{5}}{2} \approx 1.618$. It is super-linearly convergent

Solving Decimal Exponents

$$a^b = e^{b \ln a}$$

Chapter 4 – Linear Algebra Review

Review eigenvalue/vector/matrix stuff

Perturbation: a change in a vector

$\mathbf{x}^{\mathrm{T}}\mathbf{Ax}$ = scalar if **x** is a vector and **A** is a matrix

Algebraic Multiplicity: the number of <u>repeats</u> of a root
Geometric Multiplicity: the number of <u>different</u> roots

If $B = PAP^{-1}$, then A is similar to B

Vector norm: regular definition of norm; ℓ_p-norm $= \|\mathbf{x}\|_p = \left(\sum_{i=1}^{n} |x_i|^p\right)^{\frac{1}{p}}, 1 \le p \le \infty$

$$\|\mathbf{x}\|_\infty = \max_{1 \le i \le n} |x_i|$$

A norm must have the following properties:

1. $p(a\mathbf{v}) = |a|\, p(\mathbf{v})$

2. $p(\mathbf{u}+\mathbf{v}) \leq p(\mathbf{u}) + p(\mathbf{v})$
3. If $p(\mathbf{v}) = 0, \mathbf{v} = \mathbf{0}$

Matrix norm: same as vector norm, but for matrices $\|A\| = \max_{\mathbf{x}\neq\mathbf{0}} \frac{\|A\mathbf{x}\|}{\|\mathbf{x}\|} = \max_{\|\mathbf{x}\|=1} \|A\mathbf{x}\|$, i.e. the sum of all the elements on the row of the matrix with the highest sum
Euclidean norm: base 2 norm $\|\mathbf{x}\|_2$; magnitude of something in Euclidean space

Two matrices are considered **orthogonal** to each other if $\mathbf{u}^{\mathrm{T}}\mathbf{v} = 0$

Matrix multiplication review:
Recall $\mathbf{A} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}, \mathbf{B} = \begin{pmatrix} x \\ y \end{pmatrix}, \mathbf{AB} = \begin{pmatrix} ax+by \\ cx+dy \end{pmatrix}$
Positive definite: a matrix, **A**, is positive definite if the scalar, $\mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x}$, is positive, as long as $\mathbf{x} \neq \mathbf{0}$

Residual: $r = b - a\hat{x}$

Spectral radius: the eigenvalue of a matrix with the highest magnitude $\rho(A) = \max\{|\lambda|\}$

Chapter 5 – Solving a Matrix

When trying to solve for *x*, where **A***x* = **b**, where *A* is a real, non-singular, n×n matrix, and *b* is a real vector, there are two ways:

1. Direct methods (this chapter): exact solutions without round-off errors through variations of Gaussian Elimination
2. Iterative Methods: something like what is done for non-linear equations when direct methods fail

Backward Substitution

To obtain an **upper triangular** matrix (elements below the diagonal are 0), make your matrix an upper triangular matrix
Cost: $2\sum_{k=1}^{n-1}(n-k) = 2\frac{(n-1)n}{2} = O(n^2)$

Forward Substitution

To obtain a **lower triangular** matrix (elements above diagonal are 0), make your matrix a lower triangular matrix through Gauss-Jordan row operations.
Cost: same as backward substitution

Gaussian Naïve Elimination

Use this to solve for your upper and lower triangular matrices of your size *n* square matrix.
First, find the upper triangular matrix:

Start at the top left element

1. Go from left to right:
   a. Compare the current value to the value underneath it to find the value of $\frac{x_{i,j+1}}{x_{i,j}} = \ell_{i,j+1}$.
   b. Multiply each value of $x_{i,j\to n}$ by $\ell_{i,j+1}$.
   c. Repeat by comparing the current value to the value under the one you just compared it to, i.e. $\frac{x_{i,j+\text{repeats}}}{x_{i,j}}$.
2. Repeat starting at the next column, next row, i.e. start each at $x_{i=j}$

Make up your lower triangular matrix:

1. Your diagonal is all 1's
2. Everything above the diagonal is 0's
3. Use your *l* values from the U-matrix to fill in the last gaps

$$\begin{bmatrix} 1 & 0 & 0 \\ \ell_{12} & 1 & 0 \\ \ell_{13} & \ell_{23} & 1 \end{bmatrix}$$

Gaussian Elimination

A method of solving matrices that is useful for solving for *x*.

Cost: $2\sum_{k=1}^{n-1}(n-k) = 2\left((n-1)^2 + (n-2)^2 + \ldots + 1^1\right) = O\left(n^3\right)$

e.g.)

$$\left[\begin{array}{ccc|c} 1 & 1 & 1 & 3 \\ 0 & -1 & 0 & -4 \\ 0 & 0 & -5 & 4 \end{array}\right]\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 1 & 1 \\ 0 & -1 & 0 \\ 0 & 0 & -5 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 3 \\ -4 \\ 4 \end{bmatrix}$$

$$-5x_3 = 4$$

$$\boxed{x_3 = -\frac{4}{5}}$$

$$-x_2 = -4$$

$$\boxed{x_2 = 4}$$

$$x_1 + x_2 + x_3 = 3$$

$$x_1 + 4 + \tfrac{-4}{5} = 3$$

$$x_1 = \tfrac{4}{5} - 1$$

$$x_1 = \frac{4-5}{5}$$

$$\boxed{x_1 = -\frac{1}{5}}$$

$$\left(-\frac{1}{5}, 4, -\frac{4}{5}\right)$$

LU Decomposition

It is useful if you aren't given *m* number of **b** vectors.

Do not switch rows!

L: lower triangular
U: upper triangular
If L is a lower triangular, non-singular matrix, its inverse is also lower triangular.

1. Use Naïve Gaussian to find the upper and lower triangular matrices.
2. $A = LU \Rightarrow L(U\mathbf{x}) = \mathbf{b}$, which splits up into $\mathbf{y} = U\mathbf{x}$ & $L\mathbf{y} = \mathbf{b}$
3. Solve $L\mathbf{y} = \mathbf{b}$ for $\mathbf{y}$. ← $O(n^2)$
4. Use $\mathbf{y}$ to solve the equation $\mathbf{y} = U\mathbf{x}$ for $\mathbf{x}$ ← $O(n^2)$

$$\det(A) = \det(L)\det(U)$$

Cost: $O(n^3)$ for the back/forward substitution, but $O(n^2)$ for solving for **b**.

Finding the Inverse of a Matrix

Given A, find B

$$AB = I$$

GE With Pivoting (GEPP)

Due to floating point arithmetic errors, you sometimes get something where 1 + small# = 1. To avoid this we **pivot** by switching some of the rows.
Scaled GEPP: multiplying a row by a constant

Cholesky Decomposition

For symmetric matrices, you can represent the lower triangular as the transpose of the upper triangular, so we can let *G* represent *L*, where you try to find *G*. This halves the required memory and operations required.

$$A = GG^T$$

Matrix multiplication

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} = \begin{pmatrix} g_{11} & 0 \\ g_{21} & g_{22} \end{pmatrix} \begin{pmatrix} g_{11} & g_{21} \\ 0 & g_{22} \end{pmatrix}$$

$$a_{11} = g_{11}^2 + 0 \Rightarrow \boxed{g_{11} = \sqrt{a_{11}}}$$

$$a_{21} = g_{21}g_{11} + 0$$

$$g_{21} = \frac{a_{21}}{g_{11}} \Rightarrow \boxed{g_{21} = \frac{a_{21}}{\sqrt{a_{11}}}}$$

$$a_{22} = g_{21}^2 + g_{22}^2 \Rightarrow \boxed{g_{22} = \sqrt{a_{22} - g_{21}^2}}$$

$O(n^3)$

Chapter 6 – Linear Least Squares

Data fitting: forming a curve that fits between the points
Interpolation: drawing a curve along the points, guessing the shape

This method is a method of data-fitting
Linear least square: find a function that best fits the given data points. The function that you want is the one that minimizes the sum of the square of the distances from the point to the curve. Why squared?

Usually you'd be given the general shape of function.
Given *n* points, represent the distance between the points and the line by:

$$\phi(a,b) = \sum_{k=0}^{n-1} \left(f(x_k) - y_k \right)^2$$

Equate $\phi'(a,b) = 0$ to find the coefficients of *f* because that is when the function is at a minimum.

To do this find $\frac{\partial \phi(a,b)}{\partial a} = 0$ and $\frac{\partial \phi(a,b)}{\partial b} = 0$

Matrices

If your system is Ax = b, and your given y equation is $y = ax + be^x$, represent your y equation by the Ax. Given a set of *n* points

x	1	2	3
y	2	3	5

$$\overbrace{\begin{bmatrix} 1 & e \\ 2 & e^2 \\ 3 & e^3 \\ x & y_{\text{calculated}} \end{bmatrix}}^{A} \overbrace{\begin{bmatrix} a \\ b \end{bmatrix}}^{x} = \overbrace{\begin{bmatrix} y \begin{cases} 2 \\ 3 \\ 5 \end{cases} \end{bmatrix}}^{b}$$

Solve for x: x = b\A

Chapter 8–Eigenvalue

$A \in R^{n\times n}$

$Ax = \lambda x$

λ = eigenvalue
x = eigenvector

There are 3 methods to compute eigenvalues:

1. **Power method**: Finds the largest eigenvalue
2. **Inverse Power method**: Finds the smallest eigenvalue and accelerates
3. **QR Algorithm**: Finds all eigenvalues, but it's difficult (graduate level).

Power Method

Finds largest eigenvalue. Works well on large, sparse matrices.

$$x_{i+1} = \frac{Ax_i}{\|Ax_i\|}$$

$$\lambda_{i+1} = \|Ax_i\|$$

You could use either the infinity norm, the spectral norm, frobenius norm, or any norm. In fact, it isn't even necessary to normalize the eigenvector, but it is done so the magnitude is 1.

Deflation: once you have computed the largest eigenvalue, you can subtract it and repeat the method to find the other eigenvalues from highest to lowest

Inverse Power Method

Finds smallest eigenvalue, instead of largest eigenvalue
Plug in A^{-1} instead of A into the regular power method.

Chapter 10–Interpolation

Interpolation is determining the value of a point based on the values of the points around it. You do not need to use all given points.
Multiple methods:

1. Monomial basis functions: $\varphi_j(x) = X^j$
   a. Not useful: only for introducing the concept, yesterday
   b. Poorly conditioned
   c. $O(n^3)$ operations
   d. Mostly useful for theory
2. Lagrange (easier)
3. Newton Basis functions

Monomial Basis

Given $n-1$ points, each row represents a different equation

Vandermonde Matrix: $\begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix}$

e.g. (1,1),(2,3),(4,3)

$$p(1) = c_0 + c_1 + c_2 = 1$$

$$p(2) = c_0 + 2c_1 + 4c_2 = 3$$

$$p(3) = c_0 + 4c_1 + 16c_2 = 3$$

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 4 \\ 1 & 4 & 16 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \\ 3 \end{bmatrix}$$

Error

blah

Lagrange

Polynomial will be of order *n* for *n*+1 points

$$f_n(x) = \sum_{i=0}^{n} \underbrace{L_i(x)}_{\text{weighting function}} \underbrace{f(x_i)}_{y_i}$$

$$L_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j}$$

$$= \left(\frac{x - x_0}{x_i - x_0}\right)\left(\frac{x - x_1}{x_i - x_1}\right)\cdots\left(\frac{x - x_{i-1}}{x_i - x_{i-1}}\right)\left(\frac{x - x_{i+1}}{x_i - x_{i+1}}\right)\cdots\left(\frac{x - x_n}{x_i - x_n}\right)$$

Error

blah

Newton Basis Functions

Linear:

$$f_1(x) = c_0 + c_1(x - x_0)$$

$$f_1(x_0) = c_0 + c_1(x_0 - x_0)$$

$$\Rightarrow f_1(x_0) = c_0$$

$$f_1(x_1) = c_1 + c_1(x_1 - x_0)$$

$$\Rightarrow f_1(x_1) = f_1(x_0) + c_1(x_1 - x_0)$$

$$\Rightarrow \boxed{c_1 = \frac{f(x_1) - f(x_0)}{x_1 - x_0}}$$

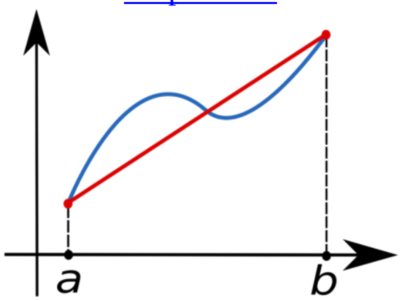
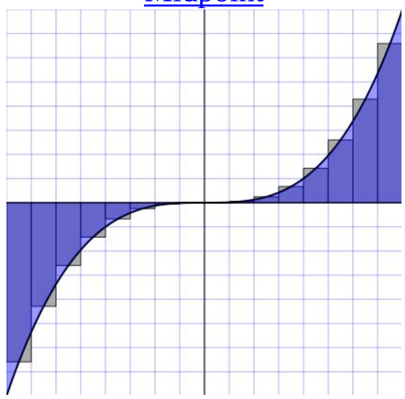
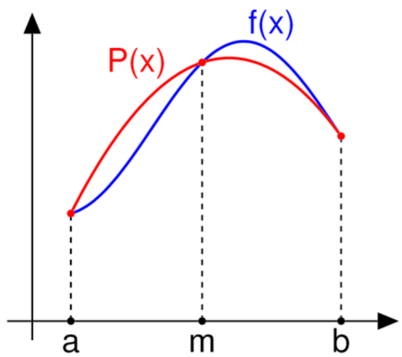
Polynomial: ?

Error

blah

Chapter 15 – Integration Methods

There are 3 quadrature integration methods:

Trapezoidal	Midpoint	Simpson's
		

Multiple Segment Trapezoidal Rule

Since this method finds the integration between a and b by finding the area assuming a straight line between f(a) and f(b), you have some error.

$$h = \frac{b-a}{n}$$

Error: $E_t = -\frac{1}{12}(b-a)^3 f''(\alpha), a < \alpha < b$

To reduce the area, break it up into *n* segments, such that you have segments: $a, a+h, \ldots, b-h, b$, where

Since each segment is now a different trapezoid, instead of having one overall error, you have multiple errors, such that the first error is:

$$E_1 = -\frac{1}{12}(a+h-a)^3 f''(\alpha_1), a < \alpha_1 < a+h$$

$$= -\frac{1}{12}(h)^3 f''(\alpha), f''(\alpha_1), a < \alpha_1 < a+h$$

The textbook summed it up to: $\int_a^b f(x)\mathrm{d}x = \frac{h}{2}\left[f(a) + f(b)\right] + h\sum_{i=1}^{r-1} f(t_i)$

Absolute Error

Notice how the *a* cancelled? Well that's going to happen all the time, so you can simply assume that the error for a given segment, *i*, the error will be:

$$E_i = -\frac{1}{12} h^3 f''(\alpha_i)$$

Therefore the total error:

$$\begin{aligned} E_T &= \sum_{i=1}^{n} -\frac{1}{12} h^3 f''(a_i) \\ &= -\frac{1}{12}\left(\frac{b-a}{n}\right)^3 \sum_{i=1}^{n} f''(\alpha_i) \\ &= -\frac{(b-a)^3}{12n^3} \sum_{i=1}^{n} f''(\alpha_i) \end{aligned}$$

However, the textbook sums it up to the following equation: $E(f) = -\frac{f''(\eta)}{12}(b-a)h^2$

Midpoint Rule

A.K.A. Rectangle Method or Riemann sum

$$\int_a^b f(x)\,\mathrm{d}x \approx h\sum_{i=1}^{r} f\left(a + \left(i - \tfrac{1}{2}\right)h\right)$$

Absolute Error

$$E(f) = \frac{f''(\xi)}{24}(b-a)h^2$$

Simpson's 1/3 Rule

A type of quadrature that also interpolates the line.

Choose 3 points, instead of 2, where the 3rd point is the midpoint between a and b. In order for the line to go through all 3 points, so it must be a curve => polynomial of order 2

$$f_2(x) = a_0 + a_1 x + a_2 x^2, a \le x \le b$$

$$f_2(a) = f(a) = a_0 + a_1 a + a_2 (a)^2$$

$$f_2\left(\tfrac{a+b}{2}\right) = f\left(\tfrac{a+b}{2}\right) = a_0 + a_1\left(\tfrac{a+b}{2}\right) + a_2\left(\tfrac{a+b}{2}\right)^2$$

$$f_2(b) = f(b) = a_0 + a_1(b) + a_2(b)^2$$

$$\begin{bmatrix} 1 & a & a^2 \\ 1 & \frac{a+b}{2} & \left(\frac{a+b}{2}\right)^2 \\ 1 & b & b^2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} f(a) \\ f\left(\frac{a+b}{2}\right) \\ f(b) \end{bmatrix}$$

Now solve the matrix for the coefficients a_0, a_1, and a_2.

Finally, solve the integral, where:

$$\int_a^b f(x)\,\mathrm{d}x \approx \int_a^b a_0 + a_1 x + a_2 x^2 \mathrm{d}x$$
$$= a_0(b-a) + a_1\left(\tfrac{b^2-a^2}{2}\right) + a_2\left(\tfrac{b^3-a^3}{3}\right)$$
$$= \frac{b-a}{6}\left[f(a) + 4f\left(\tfrac{a+b}{2}\right) + f(b)\right]$$

$\because h = \tfrac{b-a}{2}$

$$\therefore \int_a^b f(x)\,\mathrm{d}x = \frac{h}{3}\left[f(a) + 4f\left(\tfrac{a+b}{2}\right) + f(b)\right]$$

The textbook gives: $\int_a^b f(x)\,\mathrm{d}x \approx \frac{h}{3}\left[f(a) + 2\sum_{i=1}^{\frac{r}{2}-1} f(t_{2i}) + f(b)\right]$

Absolute Error

$$E(f) = -\frac{f^{(4)}(\zeta)}{180}(b-a)h^4$$

Composite Methods

These methods are a version of the quadrature methods that involves splitting up the interval between the points *a* and *b*, into intervals of size h_i. This *h* value is consistent between all the points, except in adaptive quadrature.

Errors

1. Trapezoidal

?

2. Midpoint

?

3. Simpson

?

Adaptive Quadrature

There is another version of the quadrature methods that have a changing *h* value, such that there is a smaller *h* for steeper lines, called **adaptive quadrature**. The less adaptive the method is, the **stiffer**.

These methods have the same formulas as the other quadrature methods. However, they have a changing *h* value.

Chapter 16–ODE's

To solve for $\frac{\mathrm{d}y}{\mathrm{d}x} = f(x, y), y(0) = y_0$, you could use Euler's formula: $y_{i+1} = y_i + f(x_i, y_i)h$, where $h = x_{i+1} - x_i$, but this is inaccurate, so use Runge

Runge-Kutta

Uses Taylor Series

$$y_{i+1} = y_i + \left.\frac{\mathrm{d}y}{\mathrm{d}x}\right|_{x_i, y_i} (x_{i+1} - x_i) + \frac{1}{2!}\left.\frac{\mathrm{d}^2 y}{\mathrm{d}x^2}\right|_{x_i, y_i} (x_{i+1} - x_i)^2 + \frac{1}{3!}\left.\frac{\mathrm{d}^3 y}{\mathrm{d}^3 x}\right|_{x_i, y_i} (x_{i+1} - x_i)^3 + \ldots$$

$$= y_i + f(x_i, y_i)h + \frac{1}{2!} f'(x_i, y_i)h^2 + \frac{1}{3!} f''(x_i, y_i)h^3 + \ldots$$

The second order Runge-Kutta only includes up to the part where you need to find the second derivative: $y_{i+1} = y_i + f(x_i, y_i)h + \frac{1}{2!} f'(x_i, y_i)h^2$

e.g.)

$$\frac{\mathrm{d}y}{\mathrm{d}x} = e^{-2x} - 3y, y(0) = 5$$

$$f(x, y) = e^{-2x} - 3y$$

$$f'(x, y) = \frac{\partial f}{\partial x} + \frac{\partial f}{\partial y}\frac{\mathrm{d}y}{\mathrm{d}x}$$

$$= (-2e^{-2x}) + (-3)(e^{-2x} - 3y)$$

$$= -5e^{-2x} + 9y$$

However, you want to avoid having to find the f', so they re-write it as:

$y_{i+1} = y_i = (a_1 k_1 + a_2 k_2)h$, where:

$$k_1 = f(x_i, y_i)$$

$$k_2 = f(x_i + p_1 h, y_i + q_{11} k_1 h)$$

With unknowns a_1, a_2, p_1, and q_{11}. However, there are 3 more equations determined that can help to determine the values.

$$a_1 + a_2 = 1$$

$$a_2 p_1 = \frac{1}{2}$$

$$a_2 q_{11} = \frac{1}{2}$$

Now the question is how much weighting you're going to apply between the two slopes to solve for the unknowns. How this is done is by different guesses of the value of a_2:

- Heun's Method: $a_2 = \frac{1}{2}$
- Midpoint method: $a_2 = 1$
- Ralston's method: $a_2 = \frac{2}{3}$

Heun's Method

If you assume: $a_2 = \frac{1}{2}$,

$$a_1 + a_2 = 1 \Rightarrow a_1 = \frac{1}{2}$$

$$a_2 p_1 = \frac{1}{2} \Rightarrow p_1 = 1$$

$$a_2 q_{11} = \frac{1}{2} \Rightarrow q_{11} = 1$$

$$\therefore y_{i+1} = y_i + \underbrace{\left(\frac{1}{2}k_1 + \frac{1}{2}k_2\right)}_{\text{average of 2 slopes}} h$$

$$k_1 = f\left(x_i, y_i\right)$$

$$k_2 = f\left(x_i + h, y_i + k_1 h\right)$$

Midpoint Method

Assume $a_2 = 1$

$$a_1 + a_2 = 1 \Rightarrow a_1 = 0$$

$$a_2 p_1 = \frac{1}{2} \Rightarrow p_1 = \frac{1}{2}$$

$$a_2 q_{11} = \frac{1}{2} \Rightarrow q_{11} = \frac{1}{2}$$

$$y_{i+1} = y_i + k_2 h$$

$$k_1 = f\left(x_i, y_i\right)$$

$$k_2 = f\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_1 h\right)$$

Ralston's Method

$$a_2 = \frac{2}{3}$$

$$a_1 + a_2 = 1 \Rightarrow a_1 = \frac{1}{3}$$

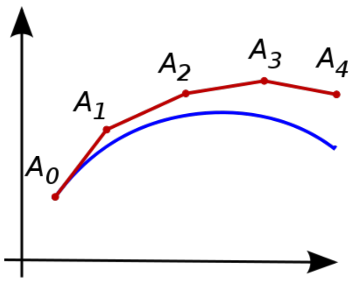
$$a_2 p_1 = \frac{1}{2} \Rightarrow p_1 = \frac{3}{4}$$

$$a_2 q_{11} = \frac{1}{2} \Rightarrow q_{11} = \frac{3}{4}$$

$$y_{i+1} = y_i + \left(\frac{1}{3}k_1 + \frac{2}{3}k_2\right)h$$
$$k_1 = f(x_i, y_i)$$
$$k_2 = f\left(x_i + \frac{3}{4}h, y_i + \frac{3}{4}k_1 h\right)$$

Forwards Euler

This is an **explicit** method to find the solution for an ODE. Compared to its **implicit** counterpart, Backwards Euler, it is more efficient for less stiff equations
Use the tangent line to determine the next point of the line. The higher you set the step size (the less stiff), the more accurate.



$$f(x, y(x)) = y'(x)$$
$$\begin{aligned} y_1 &= y(x_1) \\ &= y(x_0 + h) \\ &= y_0 + f(x_0, y_0)h \end{aligned}$$
$$y_i = y_{i-1} + f(x_{i-1}, y_{i-1})h$$
$$\boxed{y_i = y_{i-1} + y'(x_{i-1})h}$$

Error

blah

Stability

The quality of **stability** is mainly to describe how the method responds to different step sizes (*h* values).
To demonstrate stability, use the **Dahlquist Test Equation**, which is where we assume
$y'(x) = -ky(x)$
As an example, observe the analysis of the Dahlquist test equation using forward euler.

$$
\begin{aligned}
y(h) &= y(0) + y'(0)h \\
&= y(0) - ky(0)h \\
&= y(0)(1-kh) \\
y(2h) &= y(h) + y'(h)h \\
&= y(h) - ky(h)h \\
&= y(0)(1-kh) - ky(0)(1-kh)h \\
&= y(0)(1-kh)(1-kh) \\
&= y(0)(1-kh)^{2\leftarrow i}
\end{aligned}
$$

Notice how the power will appear to continuously increase? Stability is making sure the constant does not go to infinity as i approaches infinity. To do this, the initial value of the bracket must be between −1 and 1. To do this, set an h value such that:

$$
\begin{aligned}
1 - hk &> -1 \\
-hk &> -2 \\
hk &< 2 \\
\boxed{h < \frac{2}{k}}
\end{aligned}
$$

Backwards Euler

This implicit method for finding ODE's is more efficient for stiffer equations than its explicit counterpart, forwards Euler.

This is the equation: $y_{i+1} = y_i + f(x_{i+1}, y_{i+1})h$

Since y_{i+1} is on both sides of the equation, you need to bring it over to the left side and solve:

$$y_{i+1} - f(x_{i+1}, y_{i+1})h = y_i \Rightarrow \boxed{y_{i+1} - y'(x_{i+1})h = y_i}$$

and solve for y_{i+1}.

If $f(x, y) = -kxy$, you would have:

$$
\begin{aligned}
y_{i+1} + kx_{i+1}y_{i+1}h &= y_i \\
y_{i+1}(1 + kx_{i+1}h) &= y_i \\
y_{i+1} &= \frac{y_i}{(1 + kx_{i+1}h)} \\
y_{i+1} &= \left(\frac{1}{(1 + kx_{i+1}h)}\right)^i y_0
\end{aligned}
$$

If you take the limit as i → ∞, you can see that the fraction approaches 0, so this method is the more stable one.

Error

blah