CS 370 - Numerical Computation

Professor: Toshiya Hachisuka

Frank Jin

University of Waterloo

frank.jin@uwaterloo.ca

Contents

Floating Point Arithmetic

Section 1

Week 1 - May 2, 2022

Definition 1 (Properties of $\mathbb{R}$) Real numbers are:

- Infinite in extent (there exists x such that $|x|$ is arbitrarily large)
- Infinite in density ($a \leq x \leq b$ contains finitely many numbers)

There must be a way to approximate real numbers using finite amounts of data. Any real number can be expressed as an infinite expansion in base β. For example,

$$\frac{73}{3} = 24.333... = 2 \times 10^1 + 4 \times 10^0 + 3 \times 10^{-1} + ...$$

Definition 2 (Normalized Form of a Real Number) After expressing the real number in desired base β, we multiply by a power of β to shift it into normalized form:

$$0.d_1d_2d_3d_4... \times \beta^p$$

where

- d_i are digits in base β where $0 \leq d_i < \beta$
- normalized implies $d_1 \neq 0$
- p is an integer

Density is bounded by limiting the number of digits, t. Extent is bounded by limiting the range of values for p. So, our floating representation has the form has the form

$$\pm 0.d_1d_2...d_t \times \beta^p$$

for $L \leq p \leq U$ and $d_1 \neq 0$ where $\{\beta, t, L, U\}$ characterizes a specific floating point system F. The special case for floating point representation is 0. If $p > U$ or $p < L$, our system cannot represent the number. This is called overflow and underflow respectively. Examples:

- IEEE single precision (32 bits): $\{\beta = 2, t = 24, L = -126, U = 127\}$
- IEEE double precision (64 bits): $\{\beta = 2, t = 53, L = -1022, U = 1023\}$

Another system to be aware of is fixed point numbers where the number of digits after the decimal is fixed. This is essentially an integer representation scaled by a fixed factor, eg. 10234×10^{-3}.

Example Express 253.9 in a floating point system where $\beta = 10$, $t = 6$, $L = -5$, and $U = 5$.

Since 235.9 is already in base 10, we shift to get a leading 0 such that $p = 3$. This gives 0.2359×10^3. Since $t = 6$, the result is 0.235900×10^3.

Section 2

Week 1 - May 4, 2022

Floating point number systems are approximations.

$$E_{abs} = |x_{exact} - x_{approx}|$$

$$E_{rel} = \frac{|x_{exact} - x_{approx}|}{|x_{exact}|}$$

Floating point numbers are not evenly spaced. Due to some sparsity in representable numbers, absolute error can be quite different depending on the magnitude. Relative errors are useful:

- independent of magnitudes
- relate to the number of significant digits in the result; correct to approximately s digits if $E_{rel} \approx 10^{-s}$

Definition 3 (Floating Point Upper Bound) Given a floating point number F, relative error between any $x \in \mathbb{R}$ and its nearest approximation $fl(x)$, there is an upper bound E such that

$$\frac{|fl(x) - x|}{|x|} \leq E$$

where $fl(x)$ is in a neighbourhood of x between $(1 - E)x$ and $(1 + E)x$.

Rounding modes for floating numbers to consider:

- round-to-nearest (up from $\frac{1}{2}$)
- truncation/chopping (discard digits past t^{th})

Definition 4 (Machine Epsilon) Maximum relative error E for a FP system is called machine epsilon. Given

$$fl(x) = x(1 + \delta)$$

for some $|\delta| \leq E$, machine epsilon is defined as the smallest value such that $fl(1+E) > 1$. For $F = \{\beta, t, L, U\}$:

- round to nearest: $E = \frac{1}{2}\beta^{1-t}$
- truncation: $E = \beta^{1-t}$

Arithmetic with floating point numbers: for any $w, z \in F$.

$$w \oplus z = fl(w + z) = (w + z)(1 + \delta)$$

However, arithmetic is order dependent, so associativity of addition is broken. Analyzing the error bound for $(a \oplus b) \oplus c$:

For $|\delta_1| \leq E, |\delta_2| \leq E$:

$$\begin{aligned}|(a \oplus b) \oplus c - (a+b+c)| &= |fl(fl(a+b)+c) - (a+b+c)| \\ &= |((a+b)(1+\delta_1)+c)(1+\delta_2) - (a+b+c)| \\ &= |(a+b)\delta_1 + (a+b+c)\delta_2 + (a+b)\delta_1\delta_2| \\ &\leq |a+b|(|\delta_1| + |\delta_1\delta_2|) + |a+b+c||\delta_2| \\ &\leq |a+b|(E+E^2) + |a+b+c|E\end{aligned}$$

$$E_{rel} \leq \frac{|a+b|}{|a+b+c|}(E+E^2) + E$$

Slightly weakening the bound:

$$E_{rel} \leq \frac{|a|+|b|+|c|}{|a+b+c|}(2E+E^2)$$

A problem with input I and output O is well-conditioned if a change to the input ΔI gives a small change in the output. Otherwise, it is ill-conditioned. Therefore, for function f, f is well-conditioned if $f(x+\Delta x)$ for a small Δx is close to $f(x)$.

Definition 5 (Condition Number) Condition number for a function f is a number such that

$$E_{rel_{out}} \approx C_f E_{rel_{in}}$$

We can say that a small condition number equals a well-conditioned function.

Theorem 1 For a differentiable function f at x_0,

$$C_f \approx \frac{x_0 f'(x_0)}{f(x_0)}$$

Proof

$$C_f = \frac{E_{rel_{out}}}{E_{rel_{in}}} = \frac{\dfrac{f(x_0+\delta)-f(x_0)}{f(x_0)}}{\dfrac{(x_0+\delta)-x_0}{x_0}} = \frac{x_0}{f(x_0)} \times \frac{f(x_0+\delta)-f(x_0)}{\delta} \approx \frac{x_0 f'(x_0)}{f(x_0)}$$

□

Definition 6 (Stability Analysis) A stability analysis considers how an initial error propagates and magnifies through an algorithm.

Example Let $I_n = \displaystyle\int_0^1 \frac{x^n}{x+\alpha}dx$ for a given n and fixed α. For $n \geq 0$:

$$I_0 = \log\frac{1+\alpha}{\alpha} \qquad\qquad I_n = \frac{1}{n} - \alpha I_{n-1}$$

Here, we consider an initial error ϵ_0 in I_0. Let $\epsilon_0 = (I_0)_A - (I_0)_E$ where $(I_n)_E$ indicates

exact solution and $(I_n)_A$ indicates computed solution. Then,

$$\begin{aligned}\epsilon_n &= (I_n)_A - (I_n)_E \\ &= \frac{1}{n} - \alpha(I_{n-1})_A - \frac{1}{n} + \alpha(I_{n-1})_E \\ &= -\alpha((I_{n-1})_A - (I_{n-1})_E) \\ &= -\alpha\epsilon_{n-1} = (-\alpha)^2\epsilon_{n-2} = (-\alpha)^n\epsilon_0\end{aligned}$$

From this, we see that if $|\alpha| < 1$, initial error scales down over time. If $|\alpha| > 1$, initial error scales up over time.

Interpolation

Section 3

Week 2 - May 9, 2022

The problem that interpolation hopes to solve is:

Given a set of 2D points from function $y = p(x)$, can we estimate the value of p at other points?

The trivial case is when two points are given. Here, we can use linear interpolation to derive an equation of the form $y = mx + b$. Given three points, we can extend this quadratically, in a system of three equations:

$$\begin{bmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \\ x_3^2 & x_3 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

Eventually, we would like to generalize this process to n points.

Theorem 2 (Unisolvence Theorem) Given n data pairs (x_i, y_i) for $1 \leq i \leq n$ with distinct x_i, there is a unique polynomial

$$p(x) = c_1 + c_2 x^1 + ... + c_n x^{n-1}$$

where $\deg(p) \leq n - 1$ which interpolates the data.

We specify $\deg(p) \leq n - 1$ because multiple data points may align on a lower degree polynomial. For each point (x_i, y_i), we then have a linear equation $y_i = p(x_i)$ to form the system:

$$\begin{bmatrix} 1 & x_1 & x_1^2 & \dots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \dots & x_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \dots & x_n^{n-1} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

Definition 7 (Vandermonde Matrix) In the linear system of the form:

$$\begin{bmatrix} 1 & x_1 & x_1^2 & \dots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \dots & x_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \dots & x_n^{n-1} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

the Vandermonde matrix V is given by

$$V = \begin{bmatrix} 1 & x_1 & x_1^2 & \dots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \dots & x_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \dots & x_n^{n-1} \end{bmatrix}$$

We can prove the Unisolvence Theorem using Vandermonde matrices with the fact that

$$\det(V) = \prod_{i<j}(x_i - x_j) \neq 0$$

which implies invertability; an equivalent statement to the uniqueness of a solution. Solving systems with a large number of coefficients is ill-conditioned. An alternate method of solving these systems will be discussed.

Definition 8 (Monomial Form) The monomial form of a polynomial p with $\deg(p) \leq n-1$ is given by

$$p(x) = \sum_{i=1}^{n} c_i x^{i-1}$$

The monomial basis is the sequence

$$1, x, x^2, ...$$

In other words, the polynomial p is a weighted sum of the monomial basis with weights c_i.

Definition 9 (Lagrange Basis Function) Given n data points (x_i, y_i), we define

$$L_k(x) = \frac{(x-x_1)...(x-x_{k-1})(x-x_{k+1})...(x-x_n)}{(x_k-x_1)...(x_k-x_{k-1})(x_k-x_{k+1})...(x_k-x_n)}$$

In other words:

$$L_i(x_j) = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$$

With the Lagrange basis function, we can now construct our polynomial p as

$$p(x) = y_1 L_1(x) + y_2 L_2(x) + ... + y_n L_n(x) = \sum_{k=1}^{n} y_k L_k(x)$$

where $y_i = p(x_i)$.

Runge's phenomenon is when oscillating errors occur when fitting high degree polynomials to many points with polynomial interpolation.

Section 4

Week 2 - May 11, 2022

To minimize error, a helpful technique for interpolation is to use piecewise polynomials. We want to avoid "kinks" at data points with greater smoothness, achieved by enforcing continuous derivatives for our chosen function.

Hermite interpolation is the problem of fitting a polynomial given function values and derivatives. We choose to fit a cubic because we have 4 unknowns, and a quadratic only has three coefficients (not enough degree of freedom)

Example Fit a cubic function

$$p(x) = a + bx + cx^2 + dx^3$$

given Hermite data for two points x_1 and x_2:

$$p(x_1) = y_1 \qquad p(x_2) = y_2 \qquad p'(x_1) = s_1 \qquad p'(x_2) = s_2$$

To solve this, we simply take the derivative $p'(x) = b + 2cx + 3dx^2$, giving us two equations. We also get two equations from $p(x)$. This results in a linear system of

four equations for four unknowns (a, b, c, d).

When fitting many points, given values and first derivatives, we can take one cubic per pair of adjacent points. This shared data ensures that our result is of class C^1.

Theorem 3 (Closed Form Hermite Interpolation) Let $p_i(x)$ be defined as the polynomial p on the ith interval where

$$p_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3$$

The direct formulas for the polynomial coefficients are

$$a_i = y_i \qquad b_i = s_i \qquad c_i = \frac{3y_i' - 2s_i - s_{i+1}}{\Delta x_i} \qquad d_i = \frac{s_{i+1} + s_i - 2y_i'}{\Delta x_i^2}$$

where $\Delta x_i = x_{i+1} - x_i$ and $y_i' = \dfrac{y_{i+1} - y_i}{\Delta x_i}$.

Definition 10 (Knots) A knot is a point where the interpolant transitions from one polynomial/interval to another.

Definition 11 (Nodes) A node is a point where some control point is specified.

For Hermite interpolation, nodes and knots are equivalent.

Oftentimes, no derivative information is given. This introduces cubic splines, where we match a cubic $S_i(x)$ on each subinterval while being class C^2. This method requires interpolating conditions on each interval and derivative conditions at each interior point.

Theorem 4 (Cubic Spline) On each interval $[x_i, x_{i+1}]$, we fit a cubic $S_i(x)$ such that:

- Interval endpoints match:

$$S_i(x_i) = y_i \text{ and } S_i(x_{i+1}) = y_{i+1}$$

- Derivative conditions match at **interior** points:

$$S_i'(x_{i+1}) = S_{i+1}'(x_{i+1}) \text{ and } S_i''(x_{i+1}) = S_{i+1}''(x_{i+1})$$

Assuming n data points, there would be $n - 1$ intervals with 4 unknowns each, giving $4n - 4$ total unknowns. 2 interpolating conditions per interval $(2n - 2)$ and 2 derivative conditions at each interior point $(2n-4)$ gives a total of $4n-6$ equations. There are more unknowns than equations, thus this system is unsolvable. Hence, we include boundary conditions.

Definition 12 (Clamped Boundary) Assuming n data points, a clamped boundary specified at c is such that $S'(x_1) = c$ and $S'(x_n) = c$.

Definition 13 (Free Boundary) Assuming n data points, a free boundary is such that $S''(x_1) = S''(x_n) = 0$.

Definition 14 (Periodic Boundary) Assuming n data points, a periodic boundary is such that $S_1'(x) = S_n'(x)$ and $S_1''(x) = S_n''(x)$

Definition 15 (Not-A-Knot Boundary) Assuming n data points, a not-a-knot boundary is such that $S_1'''(x) = S_2'''(x)$ and $S_{n-1}'''(x) = S_n'''(x)$

The advantage of cubic splines over naive Hermite interpolation is that with cubic splines, given n points, we solve one system of size $4(n-1)$ at once. With Hermite interpolation, we solve $n-1$ separate systems of size 4×4.

Section 5

Week 3 - May 16, 2022

Recall that cubic splines of n points consist of a system of $4n-4$ unknowns. Given a naive Gaussian elimination algorithm, solving this system would be $O(n^3)$. We identify a strategy using Hermite interpolation:

1. Express unknown polynomials with closed form Hermite equations
2. Treat the s_i (slope at nodes) as unknowns
3. Solve for s_i that give continuous second derivatives
4. Given s_i, plug into closed form Hermite equations to retrieve polynomial coefficients

The derivations of the following equations are omitted from my notes as they are tedious.

Theorem 5 (Efficient Cubic Splines) For interior nodes $i = 2, ..., n-1$, we have

$$\Delta x_i s_{i-1} + 2(\Delta x_{i-1} + \Delta x_i)s_i + \Delta x_{i-1} s_{i+1} = 3(\Delta x_i y'_{i-1} + \Delta x_{i-1} y'_i)$$

For a clamped boundary condition, we additionally have $s_1 = s_1^*$ and $s_n = s_n^*$ where s_1^* and s_n^* are given. For free boundary conditions, we additionally have

$$s_1 + \frac{s_2}{2} = \frac{3}{2} y'_1 \text{ and } \frac{s_{n-1}}{2} + s_n = \frac{3}{2} y'_{n-1}$$

Recall that

$$\Delta x_i = x_{i+1} - x_i \text{ and } y'_i = \frac{y_{i+1} - y_i}{\Delta x_i}$$

We will take a view at efficient cubic splines from a matrix perspective with the equation

$$T\mathbf{s} = \mathbf{r}.$$

For interior rows ($i > 1$ and $i < n$), we have

$$\begin{aligned} T_{i,i-1} &= \Delta x_i \\ T_{i,i} &= 2(\Delta x_{i-1} + \Delta x_i) \\ T_{i,i+1} &= \Delta x_{i-1} \end{aligned}$$

while for any other $k \neq i-1, i, i+1$, we have $T_{i,k} = 0$. Let the right-hand side vector have entries defined

$$r_i = 3(\Delta x_i y'_{i-1} + \Delta x_{i-1} y'_i)$$

For clamped boundary conditions, we additionally have $T_{1,1} = 1$ and $T_{1,k} = 0$ for $k \neq 1$. The right-hand is defined $r_1 = s_1^*$ where s_1^* is given. Similarly, $T_{n,n} = 1$ and $T_{n,k} = 0$ for $k \neq n$. We also have $r_n = s_n^*$ where s_n^* is given.

As for free boundary conditions, we have $T_{1,1} = 1$, $T_{1,2} = \frac{1}{2}$, and $T_{1,k} = 0$ for $k \neq 1, 2$. Also, $r_1 = \frac{3}{2}y_1'$. Similarly, $T_{n,n} = 1$, $T_{n,n-1} = \frac{1}{2}$, and $T_{n,k} = 0$ for $k \neq n, n-1$. $r_n = \frac{3}{2}y_{n-1}'$. Visually, a free boundary $n = 6$ system for an efficient cubic spline will look like

$$\begin{bmatrix} 1 & 0.5 & 0 & 0 & 0 & 0 \\ T_{2,1} & T_{2,2} & T_{2,3} & 0 & 0 & 0 \\ 0 & T_{3,2} & T_{3,3} & T_{3,4} & 0 & 0 \\ 0 & 0 & T_{4,3} & T_{4,4} & T_{4,5} & 0 \\ 0 & 0 & 0 & T_{5,4} & T_{5,5} & T_{5,6} \\ 0 & 0 & 0 & 0 & 0.5 & 1 \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \\ s_6 \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ r_4 \\ r_5 \\ r_6 \end{bmatrix}$$

Although the size of the system with efficient cubic splines only decreases by a constant factor compared to the naive cubic spline, the fact that the system is tridiagonal makes solving much faster. In fact, solving this system is $O(n)$ as we only have two types of operations we must repeat:

- Forward elimination (eliminate entries below the diagonal)
- Backward substitution (eliminate entries above the diagonal)

We repeat both operations n times each to solve the system. After we have solved for each s_i, we may refer back to the closed form Hermite equations to solve for the coefficients of each cubic polynomial in constant time.

Section 6

Week 3 - May 18, 2022

Some curves cannot be represented using a function. The example we will look at are curves that fold back over themselves (i.e. two y values corresponding to a single x value). These types of curves can be represented parametrically, where we let x and y be separate functions of a new parameter t.

$$\mathbf{P}(t) = (x(t), y(t))$$

Example A parametric form of the unit circle $x^2 + y^2 = 1$ can be written as $x(t) = \cos(\pi t)$ and $y(t) = \sin(\pi t)$ for $0 \leq t \leq 1$.

A vector giving the direction of the tangent line at $t = t_0$ can be written as

$$\frac{d\mathbf{P}(t_0)}{dt} = \mathbf{P}'(t_0) = (x'(t_0), y'(t_0))$$

and the tangent line itself can be represented with a parametric curve, using another parameter s,

$$x_{tangent}(s) = x(t_0) + x'(t_0)(s - t_0)$$

with a similar equation for $y_{tangent}$. Parametric curves allow multiple points to have the same x or y coordinate, so curves can wrap back over themselves. Therefore, self-intersections need no special handling.

A given parametrization of a curve is not unique. Parametrizations can travel in different directions and traverse curves at different rates.

We can apply existing interpolant types on parametric curves as usual by considering $x(t)$ and $y(t)$ separately. Of course, we need data for t to form points (t_i, x_i) and (t_i, y_i). Two options are:

- Node indices
$$t_i = i$$
- Approximate arc-length
$$t_{i+1} = t_i + \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}$$

Definition 16 (Bernstein Polynomial) The Bernstein polynomials of degree N are

$$B_{i,N}(t) = \binom{N}{i} t^i (1-t)^{N-i}$$

for $i = 0, ..., N$.

Theorem 6 (Properties of Bernstein Polynomials) Bernstein polynomials are:

- Recursive: $B_{i,N}(t) = (1-t)B_{i,N-1}(t) + tB_{i-1,N-1}(t)$
- Non-negative on $t \in [0, 1]$
- Partitions of unity:
$$\sum_{i=0}^{N} B_{i,N}(t) = 1$$

Any polynomial of degree N can be expressed as a weighted sum of Bernstein polynomials of degree N.

Definition 17 (Bezier Curve) Given points $(x_0, y_0), ..., (x_N, y_N)$, take a weighted sum of each coordinate according to the Bernstein polynomials of degree N:

$$P(t) = \sum_{i=0}^{N} B_{i,N}(t)(x_i, y_i)$$

where $t \in [0, 1]$.

Bezier curves are only guaranteed to go through (x_0, y_0) and (x_N, y_N). One of the polynomials becomes 1 at these endpoints when $t = 0$ or $t = 1$. The partition of unity tells us that these endpoints should match.

Differential Equations

Section 7

Week 4 - May 23, 2022

No lecture today (Victoria Day).

Section 8

Week 4 - May 25, 2022

Closed form solutions of differential equations are often unavailable in practice. Often, we know the relationship between variable y and its derivative y', given by a known function f.

$$y'(t) = f(t, y(t))$$

Some examples of simple ODEs with closed form solutions:

Example $y'(t) = ay(t)$ with initial population $y_0 = y(t_0)$ has the closed form

$$y(t) = y_0 e^{a(t-t_0)}$$

Verify:

$$\frac{d}{dt}y(t) = \frac{d}{dt}y_0 e^{a(t-t_0)} = y_0 e^{a(t-t_0)}a = ay(t)$$

Example $y'(t) = y(t)(a - by(t))$ with resource limit b has the closed form

$$y(t) = \frac{ay_0 e^{a(t-t_0)}}{by_0 e^{a(t-t_0)} + (a - y_0 b)}$$

This is known as logistic growth.

However, many ODEs ranging from simple to complex do not have closed form solutions. In these instances, numerical methods can be used to approximate solutions.

Definition 18 (Initial Value Problem (IVP)) An IVP is a problem such that $y' = f(t, y(t))$ is the *dynamics function* and y_0 at $t = t_0$ is the *initial value*.

Example Consider a model with multiple variables of interest:

$$x'(t) = f_x(t, x(t), y(t)) \text{ with } x(t_0) = x_0$$
$$y'(t) = f_y(t, x(t), y(t)) \text{ with } y(t_0) = y_0$$

We can write this as

$$\begin{bmatrix} x(t) \\ y(t) \end{bmatrix}' = \begin{bmatrix} f_x(t, x(t), y(t)) \\ f_y(t, x(t), y(t)) \end{bmatrix} \text{ with } \begin{bmatrix} x(t_0) \\ y(t_0) \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}$$

Definition 19 (Time Stepping) Given initial conditions, we repeatedly step sequentially forward to the next time instant using y' and timestep h.

$n \leftarrow 0$
$t \leftarrow t_0$
$y \leftarrow t_0$
while *true* **do**
  compute y_{n+1}
  $t_{n+1} \leftarrow t_n + h$
  $n \leftarrow n + 1$
end while

Definition 20 The **Forward Euler** is an explicit, single-step scheme. We have $y'_n = f(t_n, y_n)$ and step forward with that slope:

$$y_{n+1} = y_n + hy'_n = y_n + hf(t_n, y_n)$$

Given differential equation $y' = f(t, y(t))$, we take the finite difference approximation of y to give

$$y' \approx \frac{y_{n+1} - y_n}{t_{n+1} - t_n} = f(t_n, y_n)$$

which yields the result as desired. A standard Taylor series expansion may also yield the result:

$$y(t_n + h) \approx y(t_n) + hy'(t_n) + \frac{h^2}{2}y''(t_n) + O(h^3)$$

where we drop terms above first order in h. Note that the derivative approximation with Taylor series yields $O(h)$ error.

Section 9

Week 5 - May 30, 2022

Time-stepping is also known as time integration. We are integrating over time to approximate y from y'. The Forward Euler $y_{n+1} = y_n + hf(t_n, y_n)$ approximates the area using rectangles of size $h_i y'(t_i)$ where height is evaluated as $y'(t) = f(t, y(t))$ from the left.

We use the following notation for a true function y: $y(t_n)$ is the exact value at time t_n whereas y_n is its approximation.

Definition 21 (Linear ODE) A linear ODE can be written in the form of a linear combination of $y(t)$ and its derivatives. If $f(t, y(t))$ is linear to $y(t)$, then $y'(t) = f(t, y(t))$ is linear.

Absolute error at step n with a given time-stepping scheme is $|y_n - y(t_n)|$. Since Forward Euler makes a linear approximation at each step, smaller steps h imply more frequent slope estimates, thus resulting in less error. The $O(h^2)$ term in

$$y_{n+1} = y_n + hf(t_n, y_n) + O(h^2)$$

is called the local truncation error in the Forward Euler method. If we were to keep more terms beyond the first two in the Taylor expansion, we can push error to higher orders. However, we do not know $y''(t)$ and it may be costly to compute, so we take an

approximation as well.

$$\begin{aligned} y(t_{n+1}) &= y(t_n) + hy'(t_n) + \frac{h^2}{2}y''(t_n) + O(h^3) \\ &= y(t_n) + hy'(t_n) + \frac{h^2}{2}\left(\frac{y'(t_{n+1}) - y'(t_n)}{h}\right) + O(h^3) \\ &= y(t_n) + hy'(t_n) + \frac{h}{2}\left(f(t_{n+1}, y(t_{n+1})) + f(t_n, y(t_n))\right) + O(h^3) \end{aligned}$$

This is known as the *Trapezoidal rule*. Unlike the Forward Euler which is explicit (involves only known quantities), Trapezoid Rule is implicit and involves quantities from the currently unknown time t_{n+1}.

Definition 22 (Improved Euler) Improved Euler is a fully explicit scheme such that

$$y_{n+1} = y_n + \frac{h}{2}(f(t_n, y_n) + f(t_{n+1}, y^*_{n+1}))$$

where $y^*_{n+1} = y_n + hf(t_n, y_n)$.

Like Trapezoid rule, Improved Euler has a local truncation error $O(h^3)$. The error calculation of Improved Euler is as follows:

By the Forward Euler and definition of y^*_{n+1}, we have $y(t_{n+1}) - y^*_{n+1} = O(h^2)$. Then,

$$\begin{aligned} f(t_{n+1}, y(t_{n+1}) &= f(t_{n+1}, y^*_{n+1}) + \frac{\partial}{\partial y}f(t_{n+1}, y^*_{n+1})(y(t_{n+1}) - y^*_{n+1} + O((y(t_{n+1}) - y^*_{n+1})^2) \\ &= f(t_{n+1}, y^*_{n+1}) + O(h^2) \end{aligned}$$

so that

$$y(t_{n+1}) = y(t_n) + \frac{h}{2}(f(t_n, y(t_n)) + f(t_{n+1}, y^*_{n+1})) + O(h^3)$$

Definition 23 (Runge Kutta Improved Euler) Improved Euler can be equivalently written as

$$\begin{aligned} k_1 &= hf(t_n, y_n) \\ k_2 &= hf(t_n + h, y_n + k_1) \\ y_{n+1} &= y_n + \frac{k_1}{2} + \frac{k_2}{2} \end{aligned}$$

with $LTE = O(h^3)$.

Definition 24 (Explicit Midpoint Method) An explicit Runge Kutta scheme with LTE $O(h^3)$ such that

$$k_1 = hf(t_n, y_n) \qquad k_2 = hf\left(t_n + \frac{h}{2}, y_n + \frac{k_1}{2}\right) \qquad y_{n+1} = y_n + k_2$$

The geometric intuition of this method is each step has a height constant matching approximately at its midpoint.

Definition 25 (Classical Runge Kutta (RK4)) Classical Runge Kutta is a scheme with $LTE = O(h^5)$:

$$\begin{aligned} k_1 &= hf(t_n, y_n) \\ k_2 &= hf\left(t_n + \frac{h}{2}, y_n + \frac{k_1}{2}\right) \\ k_3 &= hf\left(t_n + \frac{h}{2}, y_n + \frac{k_2}{2}\right) \\ k_4 &= f(t_n + h, y_n + k_3) \\ y_{n+1} &= y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \end{aligned}$$

Similary schemes exist for higher orders with $LTE = O(h^\alpha)$ for $\alpha = 6, 7, \ldots$. They can be derived by evaluating $y' = f(t, y)$ at various intermediate positions and taking a specific linear combination to find y_{n+1}.

Definition 26 (Global Error) Global error is the net error after many steps. For a given scheme, we have

$$\text{Global Error} \leq LTE \times O(h^{-1})$$

Local truncation error measures error for a single step. Given n steps, start time t_0, and end time t_n, we have

$$n = \frac{t_n - t_0}{h} = O(h^{-1})$$

Thus, some examples of global error are:

- Foward Euler: $O(h)$
- Midpoint/trapzoidal: $O(h^2)$
- RK4: $O(h^4)$

Section 10

Week 5 - June 1, 2022

Definition 27 (Backward Euler Method) An implicit method similar to Forward Euler while using the slope at the end of each step.

$$y_{n+1} = y_n + hf(t_{n+1}, y_{n+1})$$

The derivation is as follows: Let $p(t)$ interpolate y_n, y_{n+1} at times t_n, t_{n+1}. Then, by Lagrange polynomials,

$$p(t) = y_n\left(\frac{t - t_{n+1}}{t_n - t_{n+1}}\right) + y_{n+1}\left(\frac{t - t_n}{t_{n+1} - t_n}\right)$$

so

$$p'(t) = \frac{y_n}{-h} + \frac{y_{n+1}}{h} = \frac{y_{n+1} - y_n}{h}$$

where $p'(t) = f(t_{n+1}, y_{n+1})$.

Definition 28 (Fixed Point Iteration) Fixed point iteration approximates with the following:

1. Set an initial guess y_{n+1}^0 (eg. $y_{n+1}^0 = y_n$)
2. Evaluate the next guess $y_{n+1}^{k+1} = y_n + hf(t_{n+1}, y_{n+}^k)$
3. Repeat step 2 until $y_{n+1}^{k+1} \approx y_{n+1}^k$

Although Backward Euler has the same LTE $O(h^2)$ as Forward Euler and despite the fact that it is an implicit, slower method, it provides more stability. Truncation error is not the full picture: Backward Euler tends not to have error grow as we proceed more.

Example The motion of a spring satisfying Hooke's Law follows $x'(t) = v(t)$ and $v'(t) = -ax(t)$ where $a > 0$. It should preserve energy

$$e = \frac{1}{2}v^2(t) + \frac{1}{2}ax^2(t)$$

Apply Forward Euler, we get $x_{n+1} = x_n + hv_n$ and $v_{n+1} = v_n - ahx_n$ to arrive at

$$\begin{bmatrix} x_{n+1} \\ v_{n+1} \end{bmatrix} = \begin{bmatrix} 1 & h \\ -ah & 1 \end{bmatrix} \begin{bmatrix} x_n \\ v_n \end{bmatrix}$$

No matter how small h is, the determinant $1 + ah^2 > 1$. Thus, the absolute values of x_n and v_n will blow up to infinity. However, applying Sympletic Euler where we use v_{n+1} rather than v_n to update x_n gives

$$\begin{bmatrix} x_{n+1} \\ v_{n+1} \end{bmatrix} = \begin{bmatrix} 1 - ah^2 & h \\ -ah & 1 \end{bmatrix} \begin{bmatrix} x_n \\ v_n \end{bmatrix}$$

where the determinant is exactly 1. Here, Sympletic Euler preserves a modified energy

$$e = \frac{1}{2}v^2(t) + \frac{1}{2}ax^2(t) - \frac{1}{2}ahv_nx_n$$

Definition 29 (BDF2) BDF2 is a Backward Differentiation Formula using current and previous step data:

$$y_{n+1} = \frac{4}{3}y_n - \frac{1}{3}y_{n-1} + \frac{2}{3}hf(t_{n+1}, y_{n+1})$$

with LTE $O(h^3)$.

We can derive BDF2 with interpolation by fitting Lagrange polynomials $p(t)$ and $p'(t)$ where $p'(t_{n+1}) = f(t_{n+1}, y_{n+1})$ with three timestamps: t_{n-1}, t_n, t_{n+1}. The same approach extends to higher orders, yielding schemes with the form

$$\sum_{k=0}^{s} a_k y_{n+k} = hf(t_{n+s}, y_{n+s})$$

with a_k being known coefficients and y_{n+s} being the unknown time.

Definition 30 (Adams-Bashforth) Adams-Bashforth is an explicit multi-step scheme

$$y_{n+1} = y_n + \frac{3}{2}hf(t_n, y_n) - \frac{1}{2}hf(t_{n-1}, y_{n-1})$$

with LTE $O(h^3)$.

A final comparison:

Name	Single/Multi-Step	Explicit/Implicit	Global Truncation Error
Forward Euler	Single	Explicit	$O(h)$
Improved Euler and Midpoint (2nd order Runge Kutta schemes)	Single	Explicit	$O(h^2)$
4th Order Runge Kutta	Single	Explicit	$O(h^4)$
Trapezoidal	Single	Implicit	$O(h^2)$
Backwards/Implicit Euler (BDF1)	Single	Implicit	$O(h)$
BDF2	Multi	Implicit	$O(h^2)$
2-step Adams-Bashforth	Multi	Explicit	$O(h^2)$
3rd order Adams-Moulton	Multi	Implicit	$O(h^3)$

Section 11

Week 6 - June 6, 2022

A time-stepping scheme is said to be unstable if initial error ϵ_0 grows for many steps $n \to \infty$. An approach to examinining stability of time-stepping schemes:

1. Apply a given time stepping scheme to our test equation
2. Find the closed form of its numerical solution and error behaviour
3. Find the conditions on the timestep h that ensure stability (error approaching 0)

Example Consider a simple linear ODE for $\lambda > 0$ where $y'(t) = -\lambda y(t)$ and $y(0) = y_0$.

Forward Euler gives $y_{n+1} = y_n - \lambda h y_n$. Expanding this until $n = 0$, we get

$$y_{n+1} = (1 - \lambda h)y_n = ... = (1 - \lambda h)^{n+1} y_0$$

Comparing to the exact solution $y(t) = y_0 e^{-\lambda t}$ which tends to 0 as $t \to \infty$, $y_{n+1} = (1 - \lambda h)^{n+1} y_0$ tends to 0 only when $|1 - \lambda h| \leq 1$, or $h < \frac{2}{\lambda}$.

Consider the case where we have round-off error at $t = 0$ where $y_0^p = y_0 + \epsilon_0$. Our solution becomes

$$y_{n+1}^p = (1 - \lambda h)^{n+1}(y_0 + \epsilon_0)$$

so

$$\epsilon_{n+1} = y_{n+1}^p - y_{n+1} = (1 - \lambda h)\epsilon_n$$

Initial error will be dampened (matter less) when $h < \frac{2}{\lambda}$.

Consider the stability of Backward Euler in this case.

$$y_{n+1} = y_n + hf(t_{n+1}, y_{n+1}) = y_n + h(-\lambda y_{n+1}) = \frac{y_n}{1 + \lambda h} = \frac{y_0}{(1 + \lambda h)^{n+1}}$$

so the error at $n + 1$ is

$$\epsilon_{n+1} = y_{n+1}^p - y_{n+1} = \frac{y_0 + \epsilon_0}{(1 + \lambda h)^{n+1}} - \frac{y_0}{(1 + \lambda h)^{n+1}} = \frac{\epsilon_0}{(1 + \lambda h)^{n+1}}$$

Initial error will be dampened for any h. Thus, Backward Euler is unconditionally stable for this problem.

Example (Determining Local Truncation Error) Recall that $LTE = y(t_{n+1}) - y_{n+1}$ where $y(t_{n+1})$ is exact and y_{n+1} is approximate. With Forward Euler, we first replace $y_n = y(t_n)$ and f with y':

$$y_{n+1} = y_n + hf(t_n, y(t_n)) = y(t_n) + hy'(t_n)$$

The exact Taylor expansion is

$$y(t_{n+1}) = y(t_n) + hy'(t_n) + \frac{h^2}{2}y''(t_n) + O(h^3)$$

so

$$LTE = y(t_{n+1}) - y_{n+1} = \frac{h^2}{2}y''(t_n) = O(h^2)$$

Although smaller h values yield less error, they are not always more practical. Taking smaller steps means a greater overall computation cost. We want to minimize cost by choosing the largest h such that error is still less than a desired tolerance. We want to adapt the h to keep error small. In areas of smaller variation, we may choose a larger h, and in areas of larger variation, we may choose a smaller h.

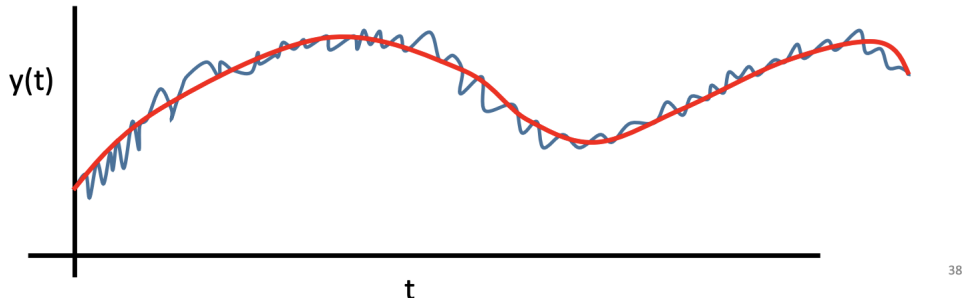
In general, we can determine LTE with the following steps where $y_{n+1} = RHS$:

1. Replace approximations on RHS with exact versions ($y_n \to y(t_n)$, $f(t_{n+1}, y_{n+1}) \to y'(t_{n+1})$, etc.
2. Taylor expand all RHS quantities about time t_n if necessary
3. Taylor expand the exact solution $y(t_{n+1})$ to compare against
4. Compute difference $y(t_{n+1}) - y_{n+1}$

Up until now, we have had constant h when time-stepping. Clearly, a smaller h value yields less error. However, the trade-off is computational cost. We would like to adapt the time-steps such that in areas of low variation, we can use larger h values, and in areas of higher variation, we can use smaller h values.

Definition 31 (Adaptive Time-Stepping) Given a tolerance,

1. Compute approximate solutions with 2 schemes of different orders
2. Estimate error by taking difference
3. While $error > tolerance$, set $h := h/2$ and redo steps 1 and 2
4. Estimate error coefficient and predict a good next step h_{new}
5. Repeat until end time is reached



Above is an example of a *stiff* problem, where the true solution is blue. If we only care about the overall oscillation (red), we could take relatively large time-steps. However, with adaptive time-stepping, we will realize that there is high variation throughout, so we will still be forced to take small time-steps with the stability condition. Implicit schemes are often better for this type of problem as they have weaker stability restrictions.

Fourier Transforms

Section 12

Week 6 - June 8, 2022

The basic idea of Fourier analysis:

1. Transform some function/data into a form that reveals freqiency of information in the data
2. Process it in this "frequency domain" form (makes some tasks easier than with original data)
3. Transform the processed data back

Given some continuous periodic function $f(t)$ with period T, we have the property $f(t \pm T) = f(t)$. Our goal is to represent $f(t)$ as an infinite sum of trigonometric functions:

$$f(t) = a_0 + \sum_{k=1}^{\infty} a_k \cos\left(\frac{2\pi kt}{T}\right) + \sum_{k=0}^{\infty} b_k \sin\left(\frac{2\pi kt}{T}\right)$$

where a_k, b_k indicate amplitude for each sinusoid for a specific period $\frac{T}{k}$ (or frequency $\frac{k}{T}$). More sinusoids give better approximations. For the case that $t \in [0, 2\pi]$ and $T = 2\pi$, we have

$$f(t) = a_0 + \sum_{k=1}^{\infty} a_k \cos(kt) + \sum_{k=1}^{\infty} b_k \sin(kt)$$

Definition 32 (Orthogonality) For all $k, j \in \mathbb{Z}$,

$$\int_0^{2\pi} \cos(kt)\sin(jt)dt = 0$$

Definition 33 (Orthogonality Relations) For all $k, j \in \mathbb{Z}$ where $k \neq j$,

$$\int_0^{2\pi} \cos(kt)\cos(jt) = 0$$
$$\int_0^{2\pi} \sin(kt)\sin(jt) = 0$$
$$\int_0^{2\pi} \sin(kt) = 0$$
$$\int_0^{2\pi} \cos(kt) = 0$$

Consider the integral $\int_0^{2\pi} f(t)\cos(jt)dt$ for $j \geq 1$. For our above $f(t)$,

$$\begin{aligned}
\int_0^{2\pi} f(t)\cos(jt)dt &= \int_0^{2\pi} \left(a_0 + \sum_{k=1}^{\infty} a_k \cos(kt) + \sum_{k=1}^{\infty} b_k \sin(kt)\right)\cos(jt)dt \\
&= \int_0^{2\pi} \left(\sum_{k=1}^{\infty} a_k \cos(kt) + \sum_{k=1}^{\infty} b_k \sin(kt)\right)\cos(jt)dt && a_0 \text{ constant} \\
&= \int_0^{2\pi} \left(\sum_{k=1}^{\infty} a_k \cos(kt)\right)\cos(jt)dt && \text{Orthogonality} \\
&= \int_0^{2\pi} (a_j \cos(jt))\cos(jt)dt && \text{Orthogonality} \\
&= \int_0^{2\pi} a_j \cos^2(jt)dt
\end{aligned}$$

so

$$a_j = \frac{\int_0^{2\pi} f(t)\cos(jt)dt}{\int_0^{2\pi} \cos^2(jt)dt}$$

Theorem 7 For $f(t)$ defined above,

$$f(t) = a_0 + \sum_{k=1}^{\infty} a_k \cos(kt) + \sum_{k=1}^{\infty} b_k \sin(kt) = \sum_{k=-\infty}^{\infty} c_k e^{ikt}$$

For $k > 0$, we have $a_0 = c_0$, $c_k = \frac{a_k}{2} - \frac{ib_k}{2}$ and $c_{-k} = \frac{a_k}{2} + \frac{ib_k}{2}$ and

$$|a_0| = |c_0| \text{ and } |c_k| = |c_{-k}| = \frac{1}{2}\sqrt{a_k^2 + b_k^2}$$

The modulus gives a given wave's frequency. The argument of c_k gives that frequency's phase, $\theta = \arg c_k$. It is also possible to derive c_k directly by

$$\int_0^{2\pi} e^{ikt}e^{-ilt}dt = \begin{cases} 0 & k \\ 2\pi & k = l \end{cases} \implies c_k = \frac{1}{2\pi}\int_0^{2\pi} e^{-ikt} f(t)dt$$

Section 13

Week 7 - June 13, 2022

No lecture today (Midterm Exam).

Section 14

Week 7 - June 15, 2022

The main idea of discrete Fourier transforms is: what if our input data is discrete $(f_0, f_1, ..., f_{N-1})$ rather than a smooth, continuous function f?

Assume that the data comes from an unknown function $f(t)$ evaluated at each $t_n =$

$n\Delta t = \frac{nT}{N}$ for $n = 0, ..., N-1$, so $f_n = f(t_n)$. For N points, we use N degrees of freedom to exactly interpolate the data. Assuming N is even:

$$f(t) \approx \sum_{k=-N/2+1}^{N/2} c_k e^{\frac{(2\pi i)kt}{T}}$$

For our N data points (t_n, f_n):

$$\begin{aligned} f_n &= \sum_{k=-N/2+1}^{N/2} c_k e^{\frac{(2\pi i)nk}{N}} \\ &= \sum_{k=0}^{N/2} c_k e^{\frac{(2\pi i)nk}{N}} + \sum_{k=-N/2+1}^{-1} c_k e^{\frac{(2\pi i)nk}{N}} \\ &= \sum_{k=0}^{N/2} c_k e^{\frac{(2\pi i)nk}{N}} + \sum_{j=N/2+1}^{N-1} c_{j-N} e^{i\frac{2\pi n(j-N)}{N}} \\ &= \sum_{k=0}^{N/2} c_k e^{\frac{(2\pi i)nk}{N}} + \sum_{j=N/2+1}^{N-1} c_{j-N} e^{i\frac{2\pi nj}{N}} \end{aligned}$$

Assuming that f is periodic on N, we have $j = k - N = k$:

$$\begin{aligned} \sum_{k=0}^{N/2} c_k e^{\frac{(2\pi i)nk}{N}} + \sum_{j=N/2+1}^{N-1} c_{j-N} e^{i\frac{2\pi nj}{N}} &= \sum_{k=0}^{N/2} c_k e^{\frac{(2\pi i)nk}{N}} + \sum_{j=N/2+1}^{N-1} c_k e^{i\frac{2\pi nk}{N}} \\ &= \sum_{k=0}^{N-1} c_k e^{i\frac{2\pi nk}{N}} \end{aligned}$$

For notation, we let $F_k := c_k$ and $W := e^{\frac{2\pi i}{N}}$. W is an Nth root of unity as $W^N = e^{2\pi i} = 1$. Thus,

$$f_n = \sum_{k=0}^{N-1} F_k W^{nk}$$

Definition 34 (Orthogonality Identity) For $k, l \in [0, N-1]$:

$$\sum_{j=0}^{N-1} W^{jk} W^{-jl} = \sum_{j=0}^{N-1} W^{j(k-l)} = N\delta_{k,l}$$

where $\delta_{k,l}$ is the Kronecker delta:

$$\delta_{k,l} = \begin{cases} 0 & k \neq l \\ 1 & k = l \end{cases}$$

With this identity, we can solve for F_k.

$$\begin{aligned} f_n = \sum_{j=0}^{N-1} F_j W^{nj} \implies \sum_{n=0}^{N-1} f_n W^{-nk} &= \sum_{n=0}^{N-1} \sum_{j=0}^{N-1} F_j W^{nj} W^{-nk} \\ &= \sum_{j=0}^{N-1} F_j \sum_{n=0}^{N-1} W^{n(j-k)} \\ &= \sum_{j=0}^{N-1} F_j \delta_{j,k} N \\ &= F_k N \end{aligned}$$

Remark: the discrete Fourier transform is invertible.

Section 15

Week 8 - June 20, 2022

Some properties of discrete Fourier transforms as a result of properties of nth roots of unity:

Definition 35 (DFT Properties) A DT has the properties:

- The sequence $\{F_k\}$ is doubly infinite and periodic. If we allow $k < 0$ or $k > N-1$, the coefficients repeat.
- Conjugate symmetry: If f_n is real, $F_k = \overline{F_{N-k}}$

Therefore, $|F_k|$ is symmetric about $k = \frac{N}{2}$.

Where $F_k = \frac{1}{N} \sum_{n=0}^{N-1} f_n W^{-nk}$, we write the DFT as a matrix transformation $F = Mf$, where M's kth column is

$$\frac{1}{N} \begin{bmatrix} W^0 = 1 \\ W^{-k} \\ W^{-2k} \\ \vdots \\ W^{-(N-1)k} \end{bmatrix}$$

For example, when $N = 4$:

$$\frac{1}{4} \begin{bmatrix} W^0 & W^0 & W^0 & W^0 \\ W^0 & W^{-1} & W^{-2} & W^{-3} \\ W^0 & W^{-2} & W^{-4} & W^{-6} \\ W^0 & W^{-3} & W^{-6} & W^{-9} \end{bmatrix} \begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \end{bmatrix} = \begin{bmatrix} F_0 \\ F_1 \\ F_2 \\ F_3 \end{bmatrix}$$

By the orthogonality identity, $\overline{M^T} M = \frac{1}{N} I$. Therefore, $M^{-1} = N\overline{M^T}$ and $f = M^{-1}F = N\overline{M^T}F$. So,

$$\begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \end{bmatrix} = 4 \times \frac{1}{4} \begin{bmatrix} W^0 & W^0 & W^0 & W^0 \\ W^0 & W^1 & W^2 & W^3 \\ W^0 & W^2 & W^4 & W^6 \\ W^0 & W^3 & W^6 & W^9 \end{bmatrix} \begin{bmatrix} F_0 \\ F_1 \\ F_2 \\ F_3 \end{bmatrix}$$

A direct implementation of F_k is slow: it takes $O(N^2)$ floating point operations. We want to design a faster algorithm (e.g. divide and conquer). The derivation will be

omitted from my notes as it is tedious, but here is the result:

Theorem 8 Given

$$F_k = \frac{1}{N}\sum_{n=0}^{N-1} f_n W^{-nk},$$

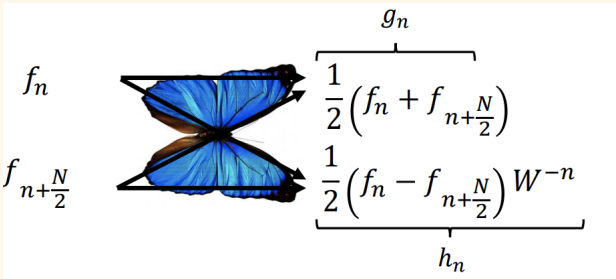
we can split the DFT of f_n into two DFT's of length $N/2$

$$g_n = \frac{1}{2}(f_n + f_{n+N/2})$$
$$h_n = \frac{1}{2}(f_n - f_{n+N/2})W^{-n}$$

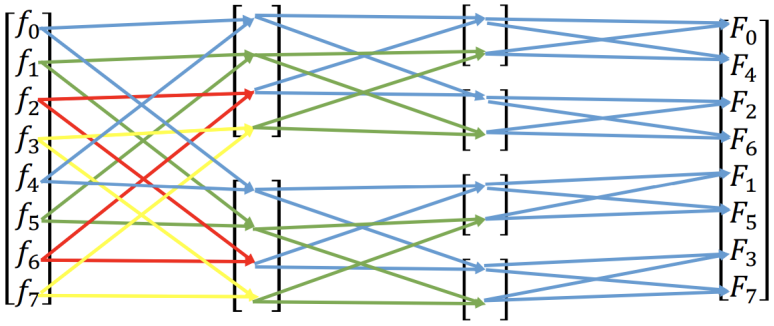
where $n \in [0, N/2-1]$.

We can recover the DFT of f by interleaving G and H where $F_{even} = G$ and $F_{odd} = H$.

Theorem 9 (Butterfly Fast Fourier Transform) Given sequence f_n, we can recursively compute the DFT of f as below:



An example of this with $N = 8$ is



The naive Fourier algorithm takes a total of $O(N^2)$ complex floating point operations, whereas FFT spends $O(N)$ operations to split each array into 2 at each stage. Solving for m where $N = 2^m$, we realize we need $m = \log_2 N$ stages so that the total cost is $O(N \log_2 N)$ which is faster.

Section 16

Week 8 - June 22, 2022

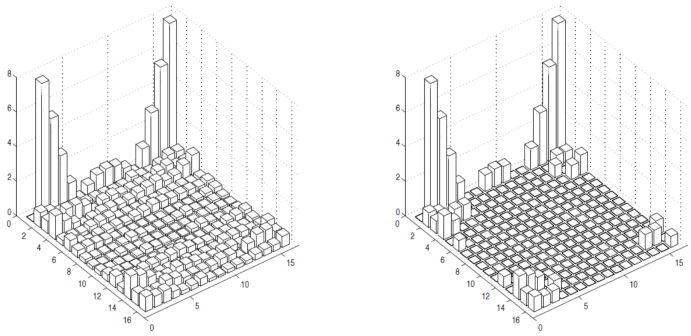
Some applications of discrete Fourier transforms:

Example (Image/data compression) For grayscale images, we can perform a DFT on pixel intensity. Many coefficients may have fairly small magnitude, so their frequencies contribute less to the overall image. A compression strategy would be to identify a tolerance and remove coefficients that have a magnitude less than the tolerance. To reconstruct the image, we would then run the inverse DFT to get the modified data pixel, and discard their imaginary parts. As images are usually in 2 dimensions, we will need a 2D Fourier transform

$$\begin{aligned} F_{k,l} &= \frac{1}{NM} \sum_{n=0}^{N-1} \sum_{j=0}^{M-1} f_{n,j} W_N^{-nk} W_M^{-jl} \\ &= \frac{1}{N} \sum_{n=0}^{N-1} W_N^{-nk} \left(\frac{1}{M} \sum_{j=0}^{M-1} f_{n,j} W_M^{-jl} \right) \\ &= \frac{1}{N} \sum_{n=0}^{N-1} H_{n,l} W_N^{-nk} \end{aligned}$$

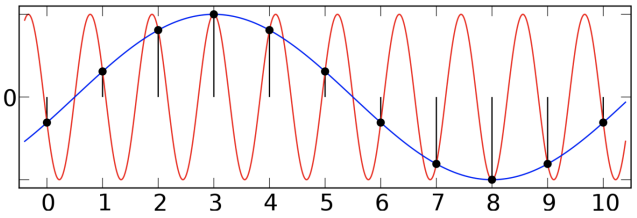
where $H_{n,l} = \left(\frac{1}{M} \sum_{j=0}^{M-1} f_{n,j} W_M^{-jl} \right)$ for an image X of size $M \times N$ and $0 \leq X(i,j) \leq 1$. The result is a 1D DFT per column on H to get f. Overall, with the FFT algorithm on X, we will have a total complexity of

$$O(MN(\log_2 M + \log_2 N)).$$



(a) Original (b) Compressed by 85%

Example (Aliasing) When a real input signal (red) has high frequencies, but the spacing of our discretely sampled data points is inadequate, "aliasing" can occur.



Our DFT coefficients F_k can be interpreted as the sum of true continuous Fourier series coefficients c_k of increasing frequency. Therefore, aliasing occurs when high frequencies from $c_k \notin [N/2-1, N/2]$ in the real signal alias as low frequency F_k for $k \in [N/2-1, N/2]$. If the original continuous signal has frequencies $c_k \neq 0$ for $|k| > N/2$, those frequencies alias to a lower frequency. We have to sample at twice the rate of the highest occurring frequency in the original signal to avoid aliasing. Otherwise, we could also filter out higher frequencies in the first place.

Example (Correlation) Finding correlation is the practice of finding similarities and relationships between different signals and data. There are applications in pattern matching, synchronizing communications, signal detection, statistics, etc.

Definition 36 (Correlation Function) Consider two real, periodic data sets y_i and z_i for $i \in [0, N-1]$. The correlation function Φ_n is

$$\Phi_n = \frac{1}{N} \sum_{i=0}^{N-1} y_{i+n} z_i$$

where n indicates offset.

We want to find n that maximizes Φ_n. If the maximum occurs at $n = p$, then shifting by p slots gives the best match between the data. In practice, signals are not always periodic, so we do not want the "wrap around" effect. In this case, we can pad our length N data sets with N 0's at the end. The naive algorithm to finding the max is having nested loops of length N ($O(N^2)$). However, with FFT, we can do this in $O(N \log_2 N)$. In frequency-domain,

$$\Phi_k = Y_k \overline{Z_k}$$

where $Y = FFT(y)$, $Z = FFT(z)$, and we compute products $\Phi_k = Y_k \overline{Z_k}$ for each $k \in [0, ..., N-1]$. Finally, we apply inverse FFT to get $\Phi = FFT(\Phi)$. Autocorrelation (correlation within a single function) can be computed similarly.

Section 17

Week 9 - June 27, 2022

Midterm solutions review.

Section 18

Week 9 - June 29, 2022

Assignment solutions review.

Numerical Linear Algebra

Section 19

Week 10 - July 4, 2022

Numerical linear algebra is the study of algorithms for performing linear algebra operations numerically. These methods vary from familiar exact methods due to efficiency, floating point error, stability, etc. First, we will study an alternative view of Gaussian elimination of the form $Ax = b$.

Definition 37 (LU Factorization) LU factorization is an algorithm to factorize a given matrix into a product of a lower triangular matrix L and an upper triangular matrix U.

for $k = 1, ..., n$ **do**
  for $i = k + 1, ..., n$ **do**
    $mult \leftarrow a_{ik}/a_{kk}$
    $a_{ik} \leftarrow mult$
    for $j = k + 1, ..., n$ **do**
      $a_{ij} \leftarrow a_{ij} - mult \times a_{kj}$
    end for
  end for
end for

The first step to solving $Ax = b$ is to factor A into LU. We then rewrite $L(Ux) = b$. To proceed, we let $z = Ux$ and solve $Lz = b$ for z. Then, we solve $Ux = z$ for x. Solving at these steps is straightforward as they only involve backward and forward substitutions.

There is a problem with the LU Factorization algorithm. The variable *mult* is initialized to be the quotient of two matrix entries. However, if $a_{kk} = 0$, then there will be divide-by-zero problem. The solution to this is row-pivoting: find the row with the largest magnitude entry in the current column beneath the current row, and swap those rows if larger than the current entry. Realistically, our system would be

$$PA = LU$$

where P is a permutation matrix. P is simply a row-swapped version of the identity matrix.

Example To swap rows 2 and 3 of a 3×3 matrix, we swap rows 2 and 3 of I. Therefore,

$$P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Overall, we will be solving $PAx = Pb$, which becomes $LUx = Pb = b'$. From there, we can proceed normally and solve for x. To determine P algorithmically, we can start with an identity matrix of the appropriate size, and swap rows correspondingly every time we swap rows in the LU factorization.

We want to perform asymptotic analysis on the cost to solve a system of size $n \times n$. We measure this cost in the number of floating point operations. Analyzing the LU factorization algorithm, we get the summation

$$\sum_{k=1}^{n} \sum_{i=k+1}^{n} \sum_{j=k+1}^{n} 2 = \frac{2n^3}{3} + O(n^2)$$

We can also analyze our forward and backward substitution steps to see $n^2 + O(n)$ for each. Thus, Gaussian elimination is $O(n^3)$ overall. Given a factorization, solving for a new RHS is cheap at $O(n^2)$. We can similarly show that the naive approach of inverting A in $Ax = b$ to solve for x is more expensive in FLOPs.

Section 20

Week 10 - July 6, 2022

Previously, we saw row swapping as a matrix P. We can do the same for row subtraction, where $A' = MA$ such that A' is the original matrix A after a subtraction operation defined by M. M is the identity matrix but with a zero entry replaced by the negative of the necessary multiplicative factor. So, the whole factorization process can be viewed as

$$M^{(n)}...M^{(2)}M^{(1)}A = U$$

for an $n \times n$ matrix A. For the 3×3 case,

$$A = (M^{(3)}M^{(2)}M^{(1)})^{-1}U$$

so that $L = (M^{(3)}M^{(2)}M^{(1)})^{-1}$. The inverse of a given $M^{(k)}$ is simply $M^{(k)}$ except with the negation of the multiplicative factor.

Definition 38 (Vector Norm) For a vector $\mathbf{x} = [x_1, ..., x_n]^T$, we have:

- 1-norm: $||\mathbf{x}||_1 = \sum_{i=1}^{n} |x_i|$
- 2-norm: $||\mathbf{x}||_2 = \sqrt{\sum_{i=1}^{n} x_i^2}$
- ∞-norm: $||\mathbf{x}||_\infty = \max_i |x_i|$
- p-norm: $|\mathbf{x}||_p = (\sum_{i=1}^{n} |x_i|^p)^{1/p}$ where $p = 1, 2, ..., \infty$

We may also refer to p-norms as l^p-norms.

Definition 39 (Norm Properties) For vectors $\mathbf{x}$ and $\mathbf{y}$, and scalar α,

- $||\mathbf{x}|| = 0 \implies x_i = 0 \forall i$
- $||\alpha\mathbf{x}|| = |\alpha|||\mathbf{x}||$
- $||\mathbf{x} + \mathbf{y}|| \leq ||\mathbf{x}|| + ||\mathbf{y}||$

Using the p-norms of matrices, we can induce the following:

$$||A|| = \max_{||x|| \neq 0} \frac{||A\mathbf{x}||}{||\mathbf{x}||}$$

with equivalent definitions $||A||_1 = \max_j \sum_{i=1}^n |A_{ij}|$ and $||A||_\infty = \max_i \sum_{j=1}^n |A_{ij}|$. The spectral norm, or 2-norm, can also be induced as

$$||A||_2 = \max_{||x||\neq 0} \frac{||A\mathbf{x}||_2}{||\mathbf{x}||_2}$$

relating to the eigenvalues. If λ_i are the eigenvalues of $A^T A$, then

$$||A||_2 = \max_i \sqrt{|\lambda_i|}$$

Definition 40 (Matrix Norm Properties) For matrices A, B, scalar α, and vector $\mathbf{x}$:

- $||A|| = 0 \iff A_{ij} = 0 \forall i, j$
- $||\alpha A|| = |\alpha| \cdot ||A||$
- $||A + B|| \leq ||A|| + ||B||$
- $||A\mathbf{x}|| \leq ||A|| \cdot ||\mathbf{x}||$
- $||AB|| \leq ||A|| \cdot ||B||$
- $||I|| = 1$

Definition 41 Two norms $||\cdot||_a$ and $||\cdot||_b$ are called equivalent if there exist constants C_1 and C_2 such that

$$C_1||\mathbf{x}||_a \leq ||\mathbf{x}||_b \leq c_2||\mathbf{x}||_a$$

where $\mathbf{x}$ is an element of vector space X.

Example Prove that 1-norm is equivalent to 2-norm.

Proof Note that

$$||\mathbf{x}||_1 = \sum_{i=1}^n |x_i| \leq \sqrt{\sum_{i=1}^n x_i^2}\sqrt{\sum_{i=1}^n 1^2} = \sqrt{n}||\mathbf{x}||_2$$

and

$$||\mathbf{x}||_2^2 = \sum_{i=1}^n x_i^2 \leq \sum_{i=1}^n |x_i| \sum_{i=1}^n |x_i| = ||\mathbf{x}||_1^2$$

which implies $||\mathbf{x}|| \leq ||\mathbf{x}||_1$. □

Conditioning describes how the output of a function, operation, or problem changes with respect to change in the input. This is independent of the algorithm used.

Definition 42 (Condition Number) The condition number of matrix A is denoted

$$\kappa(A) = ||A||||A^{-1}||$$

where $\kappa \approx 1$ implies A is well-conditioned, and $\kappa >> 1$ implies A is ill-conditioned.

For system $Ax = b$, $\kappa(A)$ provides upper bounds on relative change in $\mathbf{x}$ due to relative change in b.

$$\frac{||\Delta\mathbf{x}||}{||\mathbf{x}||} \leq \kappa(A)\frac{||\Delta b||}{||b||} \text{ and } \frac{||\Delta\mathbf{x}||}{||\mathbf{x} + \Delta\mathbf{x}||} \leq \kappa(A)\frac{||\Delta A||}{||A||}$$

If unspecificed, we assume the 2-norm for condition number.

Section 21

Week 11 - July 11, 2022

We represent the web's structure as a directed graph, where nodes represent pages and arcs represent links from one page to another. We use degree to refer to a node's outdegree: the number of arcs leaving that node.

Definition 43 (Adjacency Matrix) To store a directed graph, we use adjacency matrix G such that

$$G_{i,j} = \begin{cases} 1 & \text{if link } j \to i \text{ exists} \\ 0 & \text{otherwise} \end{cases}$$

so that the outdegree for node q is the sum of entries in column q.

Symmetry with respect to the diagonal implies bidirectionality.

We can interpret links as votes of importance. Outgoing links of a page j have equal influence, so an importance that j gives to a linked page i is $\frac{1}{\deg j}$.

Definition 44 (Random Surfer Algorithm) The Random Surfer Algorithm estimates overall importance of a given page based on a user who starts at a page and follows random links.

$Rank(m) \leftarrow 0$, $m \leftarrow 1, ..., R$
for $m \leftarrow 1, ..., R$ **do**
  $j \leftarrow m$
  for $k \leftarrow 1, ..., K$ **do**
    $Rank(j) \leftarrow Rank(j) + 1$
    Randomly select outlink l of page j
    $j \leftarrow l$
  end for
end for
$Rank(m) \leftarrow Rank(m)/(K \times r), m \leftarrow 1, ..., R$

Clearly, this algorithm has some problems: what if we reach a dead link, or a cycle in the graph? Also, the number of real web pages is monstrously huge. Instead, we think in terms of probabilities with the following Markov chain matrix.

$$P_{i,j} = \begin{cases} \frac{1}{\deg(j)} & \text{if link } j \to i \text{ exists} \\ 0 & \text{otherwise} \end{cases}$$

To deal with dead ends, we will "teleport" to a new, random page. We define the following column vector d:

$$d_i = \begin{cases} 1 & \deg(i) = 0 \\ 0 & \text{otherwise} \end{cases}$$

and $e = [1, 1, ..., 1]^T$. If R is the number of pages, we augment P to get P':

$$P' = P + \frac{1}{R} e d^T.$$

The matrix $\frac{1}{R} e d^T$ is the matrix of probabilities such that from any dead end page

($d_i = 1$), we transition to every other page with equal probability. To escape closed cycles of pages, we use a similar trick:

$$M = \alpha P' + (1-\alpha)\frac{1}{R}ee^T$$

where $\alpha P'$ is the instance when we surf randomly, and $(1-\alpha)\frac{1}{R}ee^T$ is the instance where we teleport randomly from any page to any page. The matrix $\frac{1}{R}ee^T$ looks like

$$\begin{bmatrix} \frac{1}{R} & \cdots & \frac{1}{R} \\ \vdots & \ddots & \vdots \\ \frac{1}{R} & \cdots & \frac{1}{R} \end{bmatrix}$$

We call M the "Google Matrix". Google purportedly uses $\alpha \approx 0.85$.

Definition 45 (Properties of M) The Google Matrix M satisfies:

- $0 \leq M_{i,j} \leq 1$ for all entries
- Each column of M sums to 1:

$$\sum_{i=1}^{R} M_{i,j} = 1$$

The interpretation is that if we are currently on a webpage, the probability that we are on *some* webpage after a transition is 1. These two properties are properties of a Markov matrix.

If we start our surfer on a random page, we can represent this as a probability vector with $q_i = 1/R$. We have the probability vector to describe the initial state p^0 (where superscript represents the number of transitions taken) and Markov matrix M representing the transition probabilities among pages. Then, for any step n,

$$p^{n+1} = Mp^n.$$

Theorem 10 If p^n is a probability vector, then $p^{n+1} = Mp^n$ is one also.

Proof We must satisfy the two properties of Markov matrices. First, $0 \leq p_i^{n+1} \leq 1$ since it is the sums and products of probabilities between 0 and 1. To show that $\sum_i p_i^{n+1} = 1$,

$$\sum_i p_i^{n+1} = \sum_i \sum_j M_{i,j} p_j^n = \sum_j p_j^n \sum_i M_{i,j} = \sum_j p_j^n = 1$$

□

With what probability does our surfer end up at each page after many steps, starting from $p_0 = \frac{1}{R}e$? That is, what is $p^\infty = \lim_{k\to\infty}(M)^k p_0$? We can conclude that higher probability in p^∞ implies greater importance/rank.

Section 22

Week 11 - July 13, 2022

In numerical linear algebra, we often deal with two kinds of matrices. Dense matrices are such that most entries are non-zero. Sparse matrices are such that most entries are zero. In this case, we store only the non-zero entries to save time and space.

A Google matrix is fully dense, but the transition matrix P is usually sparse. We attempt to utilize the sparsity in matrix P with linear algebra manipulations. Assuming p^n is a probability vector, we have

$$\frac{ee^t}{R}p^n = \frac{1}{R}e(e^t p^n) = \frac{e}{R}$$

so that

$$Mp^n = \alpha P' p^n + (1-\alpha)\frac{e}{R}$$

and

$$P'p^n = \left(p + \frac{ed^t}{R}\right)p^n = Pp^n + e\left(\frac{d^t p^n}{R}\right)$$

Overall,

$$Mp^n = \alpha\left(Pp^n + e\left(\frac{d^t p^n}{R}\right)\right) + (1-\alpha)\frac{e}{R}$$

where $Mp^n = p^{n+1}$ as before. The cost of this operation is $O(R)$. Overall, we can write the Page Rank Algorithm with respect to a tolerance tol.

$p^0 = e/R$
for $k = 1, ...,$ until converged **do**
  $p^k = Mp^{k-1}$
  if $\max_i |[p^k]_i - [p^{k-1}]_i| < tol$ **then**
    quit
  end if
end for

PageRank can be tweaked to incorporate factors such as user preference. For example, we can replace teleportation ($\frac{1-\alpha}{R}ee^T$) with $(1-\alpha)ve^T$ where a special probability vector v places extra weight on sites that the user visits more often.

Definition 46 (Eigenvalue) An eigenvalue λ and corresponding eigenvector $\mathbf{x}$ of a matrix Q are a non-zero scalar and vector, respectively, which satisfy

$$Q\mathbf{x} = \lambda\mathbf{x}$$

Rearranging gives $(\lambda I - Q)\mathbf{x} = 0$ implying that the matrix $(\lambda I - Q)$ must be singular. Recall that a matrix A is non-singular/invertible if the equation $A\mathbf{x} = 0$ only has the trivial solution $\mathbf{x} = 0$. Since a singular matrix A satisfies $\det A = 0$, we can find the eigenvalues λ of Q by solving the characteristic polynomial given by

$$\det(\lambda I - Q) = 0.$$

In terms of the Page Rank Algorithm, the Page Rank process is converging towards a specific eigenvector of the Markov matrix M. We are working towards proving that the Page Rank Algorithm converges.

Theorem 11 (Markov Matrix Properties) Given a Markov matrix Q,

1. Q has 1 as an eigenvalue
2. Every eigenvalue of Q satisfies $|\lambda| \leq 1$
3. If $Q_{ij} > 0$ for all i, j, then Q is a positive Markov matrix
4. If Q is a positive Markov matrix, then there is only one linearly independent eigenvector of Q with $|\lambda| = 1$

Proof (1). Since $Q^T e = 1e$, 1 is a eigenvalue of Q^T with eigenvector e by definition. Since $\det(Q) = \det\left(Q^T\right)$, 1 is also an eigenvalue of Q.

(2). For the sake of contradiction, assume there exists an eigenvalue $|\lambda| > 1$ with corresponding eigenvector v. Then, $Qv = \lambda v$, meaning that multiplying Q n times is equivalent to multipyling V n times. v grows exponentially since $|\lambda| > 1$. However, this implies Q^n has some entries larger than 1, thus a contradiction.

(3). Definition.

(4). Omitted. □

Theorem 12 If Q is a positive Markov matrix, then $Q\mathbf{x} = \mathbf{x}$ for some $\mathbf{x}$. If also $Q\mathbf{y} = \mathbf{y}$, then $\mathbf{y} = c\mathbf{x}$ for some scalar c. Eigenvector with $\lambda = 1$ is unique.

Theorem 13 PageRank converges.

Proof Using eigenvalues λ_i and eigenvectors v_i, we say

$$M^k p = \sum_{i=1}^{R} c_i (\lambda_i)^k v_i$$

If we say $\lambda_1 = 1$, because $|\lambda_i| < 1$ for any other i,

$$M^\infty p \sum_{i=1}^{R} c_i (\lambda_i)^\infty v_i = c_1 v_1.$$

The number of iterations required for PageRank to converge to the final vector p^∞ depends on the size of the second largest eigenvalue, λ_2. The 2nd largest eigenvalue dictates the slowest rate at which the "unwanted" components of p_0 are shrinking. □

Appendix

Section 23

Complex Numbers

Definition 47 (Complex Number Properties) Some properties of complex numbers:

- Euler's Formula:
$$e^{i\theta} = \cos(\theta) + i\sin(\theta)$$
$$e^{-i\theta} = \cos(\theta) - i\sin(\theta)$$
- Cosine Identity:
$$\cos(\theta) = \frac{e^{i\theta} + e^{-i\theta}}{2}$$
- Sine Identity:
$$\sin(\theta) = \frac{e^{i\theta} - e^{-i\theta}}{2i}$$
- Roots of Unity:

  Example (8th roots of unity)

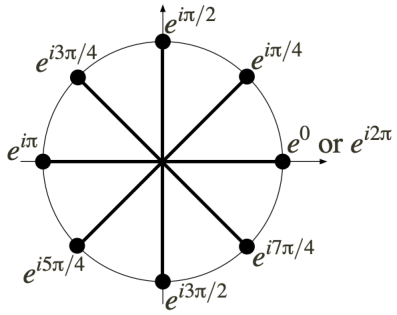


Figure 19: The 8th roots of unity plotted on the unit circle in the complex plane.

$$\begin{array}{rclrcl} e^{i\pi/4} &=& \frac{1}{\sqrt{2}} + \frac{1}{\sqrt{2}}i, & e^{i\pi/2} &=& i, \\ e^{3i\pi/4} &=& -\frac{1}{\sqrt{2}} + \frac{1}{\sqrt{2}}i, & e^{i\pi} &=& -1, \\ e^{5i\pi/4} &=& -\frac{1}{\sqrt{2}} - \frac{1}{\sqrt{2}}i, & e^{3i\pi/2} &=& -i, \\ e^{7i\pi/4} &=& \frac{1}{\sqrt{2}} - \frac{1}{\sqrt{2}}i, & e^{2i\pi} &=& 1. \end{array} \tag{13.7}$$

Nth roots of unity have the form $e^{2\pi ik/N}$ where $0 \leq k \leq N$. We evenly split "slices" across the unit circle where we can use Euler's Formula to compute the root.