CS 370 with Leili Rafiee Sevyeri

Eason Li

2025 W

Contents

1 Introduction

Lecture 0: Administrative - Monday, January 06

Definition 1.1: Symbolic and Numerical Computation

Considering what $\sqrt{2}$ is, it has two meanings. Symbolically, it is the positive root of $x^2 - 2$, and numerically, it is

$$1.414213562\cdots$$

In symbolic computation, we care about

1. Correct solution;
2. Efficiency;

while in numerical computation, we care about

1. Correct solution;
2. Efficiency;
3. Stability;
4. Condition.

Definition 1.2: Numerical Computation

In a nutshell, numerical computation is using computer algorithms to (approximately) solve a range of mathematical problems.

Example 1.1

1. Weather and climate modelling;
2. Financial modelling;
3. Computer graphics and animation;
4. Physics...

2 Floating Point Number System

Definition 2.1: Real Numbers

- Infinite in *extent*: There exists x such that $|x|$ is arbitrarily large;
- Infinite in *density*: Any interval $a \leq x \leq b$ contains infinitely many numbers.

Discovery 2.1

We note that computers cannot represent infinite/ arbitrarily large quantities. To resolve this issue, the standard (partial) solution is `to use floating point numbers to approximate the reals.`

Definition 2.2: Floating Number System

An approximate representation of real numbers using a finite number of bits.

2.1 Pitfalls in Floating Point Computation

Example 2.1: Numerical Errors

Consider the problem,

$$12 + \sum_{i=1}^{100} 0.01$$

We know that the real solution to this is 13. However, if we can only keep 2 digits of accuracy at a time, this would result in a value of 12:

$$\underbrace{(\underbrace{(\underbrace{12 + 0.01}_{12.01 \approx 12}) + 0.01}_{12.01 \approx 12}) + \cdots}_{} \approx 12$$

Example 2.2: Numerical Errors: Taylor Series

Recall the Taylor Series, to approximate $e^{-5.5}$, we may apply the Taylor Series to $f(x) = e^x$ around $a = 0$ and plug in $x = -5.5$. Using 5 digit of accuracy, we find

$$e^{-5.5} \approx 0.0026363$$

after 25 terms of calculation. Moreover, we have another approximation:

$$e^{-x} = \frac{1}{e^x} = \frac{1}{1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots}$$

Again using 5 digit of accuracy, we now get $e^{-5.5} \approx 0.0040865$.

```
The reason why first method fails is that we have subtractions,
which causes cancellation of information.
```

Lecture 2 - Wednesday, January 08

Discovery 2.2

Floating point numbers often don't quite behave like true real numbers. This can lead to subtle (yet huge) errors!

To be useful, our numerical algorithms must work eectively under floating point arithmetic.

2.2 Normalized Form

Definition 2.3: Normalized Form

After expressing the real number in the desired base β, we multiply by a power of β to shift it into a normalized form:

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

where:

- d_i are digits in base β, i.e. $0 \leq d_i < \beta$.
- normalized implies we shift to ensure $d_1 \neq 0$.
- exponent p is an integer.

Comment 2.1

You may see other normalization conventions outside this class, e.g. $d_1.d_2d_3d_4 \ldots \times \beta^p$

Result 2.1

Density (or precision) is bounded by limiting the number of digits, t.
Extent (or range) is bounded by limiting the range of values for exponent p.

2.2.1 Specific Floating Point System

Theorem 2.1: Specific Floating Point System

The four integer parameters $\{\beta, t, L, U\}$ characterize a **specific floating point system**, F, where $\beta =$ base, $t =$ mantissa, and L and U are the lower and upper bounds for the exponent.

Definition 2.4: Overflow and Underflow

When the exponent is too big ($> U$)/ or too small ($< L$), we get **overflow**/ or **underflow**.

Example 2.3: Exception Handling

	Example	Result
Invalid Output	$0/0$	NaN
Division by 0	$1/0$	$\pm\infty$
Overflow	$N_{max}+1$	$\pm\infty$
Underflow	$N_{min}/2$	0

Definition 2.5:

The two most common standardized floating point systems are:

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

Theorem 2.2

$S_m = 1$ bit (signed bit of matissa)	mantissa = 23 bit	$S_p = 1$ bit (signed bit of exponent)	exponent = 7 bits

Definition 2.6: Fixed Point Numbers

The number of digits after the decimal (or radix) point is fixed.

Discovery 2.3

Unlike fixed point, floating point numbers are not evenly spaced!

Example 2.4

For $F = \{\beta = 2, t = 3, L = -1, U = 2\}$ the representable (non-negative) values are spaced like:

2.3 Rounding vs. Truncation

When converting a real number into a representable FP number, we will only consider:

- `Round-to-nearest` — rounds to closest available number in F.
  - Usually the default.
  - Well break ties by simply rounding $1/2$ **up**. (Various other options exist ...)
- `Truncation/ ‘‘Chopping’’` — rounds to next number in F towards zero. i.e. simply discard any digits after the t-th.

Example 2.5

Express 253.9 in a floating point system with base $\beta = 10$, $t = 6$ digits, $L = -5$, $U = 5$.

Solution: The answer is 0.253900×10^3. ★

Example 2.6

Express 8.25 in a floating point system with base $\beta = 2$, $t = 7$ digits, $L = -5$, $U = 5$.

Solution: Expressing the number in base 2, we know that

$$8.25 = 1000.01$$

Thus we obtain that

$$8.25 = 0.100001 \times 2^4$$

after shifting to get the leading 0, where the exponent $p = 4$. The result is hence 0.1000010×2^4. ★

Example 2.7

Express 1030.9671 in a floating point system with base $\beta = 10$, $t = 6$ digits, $L = -5$, $U = 5$.

Solution: After rounding: 0.103097×10^4; After truncating: 0.103096×10^4. ★

2.4 Measuring Error

Definition 2.7: Absolute Error

We define **absolute error** to be

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

Definition 2.8: Relative Error

We define **relative error** to be

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

Comment 2.2

Relative error is more useful because

- is independent of the magnitudes of the numbers involved.
- relates to the number of significant digits in the result.

2.4.1 Correct to *roughly* s digits

Theorem 2.3

As a rule of thumb, a result is correct to *roughly* s digits if $E_{rel} \approx 10^{-s}$, or

$$0.5 \times 10^{-s} \leq E_{rel} \leq 5 \times 10^{-s}$$

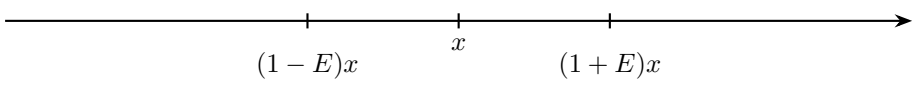
Given that for FP system F, rel. err. between $x \in \mathbb{R}$, and its FP approximation, $fl(x)$, has a bound, E. We deduce

$$\begin{aligned} E_{rel} = \frac{|x - fl(x)|}{|x|} &\leq E \\ \Rightarrow \quad |x - fl(x)| &\leq E|x| \\ \Rightarrow \quad |x| - |fl(x)| &\leq E|x| \\ \Rightarrow \quad (1-E)|x| &\leq |fl(x)| \end{aligned}$$

Similarly, one can also show that $|fl(x)| \leq (1+E)|x|$.

Comment 2.3

i.e., $fl(x)$ is in a neighborhood of x, somewhere!



Definition 2.9: Machine Epsilon

This maximum relative error, E, for a FP system is called **machine epsilon** or **unit round-oerror**. It is defined as the smallest value such that $fl(1+E) > 1$ under the given floating point system.

Example 2.8

Suppose we have $F = \{\beta, t, L, U\}$ with chopping. Therefore we have

$$1 = 0.\underbrace{10\ldots0}_{t} \times \beta^1$$

and

$$1 + E = 0.\underbrace{10\ldots01}_{t} \times \beta^1$$

Subtracting them we obtain

$$E = 0.\underbrace{0\ldots01}_{t} \times \beta^1 = \beta^{-t} \cdot \beta^1 = \beta^{1-t}$$

Result 2.2

We have the rule $fl(x) = x(1 + \delta)$ for some $|\delta| \leq E$.

Result 2.3

For an FP system (β, t, L, U) with ...

- `rounding to nearest`: $E = 1/2\beta^{1-t}$ (because this is rounded up).
- `truncation`: $E = \beta^{1-t}$.

2.5 Floating Point Error Analysis and Stability

Definition 2.10: Floating Point Addition

IEEE standard requires that for $w, z \in F$,

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

where $\oplus$ denotes **floating point addition**. Again, with $|\delta| \leq E$.

Theorem 2.4

This rule only applies to individual FP operations! Not sequences of operations. i.e., It is not generally true that

$$(a \oplus b) \oplus c = a \oplus (b \oplus c) = fl(a + b + c)$$

In other words, associativity is broken.

Lecture 3 - Monday, January 13

2.5.1 Error Bound of $(a \oplus b) \oplus c$

Recall that by the definition of E_{rel}, we have

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

Exercise: Show that

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

Comment 2.4

This analysis describes only the worst case error magnification, as a function of the input data. Actual error could be (much) less.

Discovery 2.4

In the error bound for $(a \oplus b) \oplus c$ as shown above, we have

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

Notice that the term $|a|+|b|+|c|/|a+b+c|$ essentially describes how much E could be scaled by to give the total error, when summing 3 numbers a, b, c.

Worst case magnification when the denominator is small. i.e., when $|a+b+c| \ll |a|+|b|+|c|$. This occurs when quantities have diering signs and similar magnitudes, leading to "cancellation".

Example 2.9

Perform $(a \oplus b) \oplus c$ with $a = 2000$, $b = -3.234$, $c = -2000$ for $F = \{10, 4, -10, 10\}$, with rounding. We compute to obtain:

- *True result:* -3.234
- *FP result:* $(2000 \oplus -3.234) \oplus -2000 = 1997 \oplus -2000 = -3$

Hence the error bound is

$$\begin{aligned} E_{rel} &\leq \frac{|a| + |b| + |c|}{|a + b + c|}(2E + E^2) \\ &\approx \frac{4003.234}{3.234} \cdot 2\left(\frac{1}{2}10^{-3}\right) \\ &\approx 1.238 \quad \text{(No correct digit)} \end{aligned}$$

Note that this is a rather weak bound. The actual relative error: $0.234/3.234 \approx 0.072$ (roughly one digit correct).

Example 2.10

Perform $(a \oplus c) \oplus b$ with $a = 2000$, $b = -3.234$, $c = -2000$ for $F = \{10, 4, -10, 10\}$, with rounding. We compute to obtain:

- *True result:* -3.234
- *FP result:* $(2000 \oplus -2000) \oplus -3.234 = 0 \oplus -3.234 = -3.234$

`Observation:` Re-ordering operations often changes the results.

Example 2.11

Perform $(a \oplus b) \oplus c$ with $a = -2000$, $b = -3.234$, $c = -2000$ for $F = \{10, 4, -10, 10\}$, with rounding. We compute to obtain:

- *True result:* -4003.234
- *FP result:* $(-2000 \oplus -3.234) \oplus -2000 = -2003 \oplus -2000 = -4003$

Hence the error bound:

$$E_{rel} \leq (2E + E^2) \approx 2E = 10^{-3}$$

while the actual relative error is $\sim 5.8 * 10^{-5}$.
`Observation:` Expressions without cancellation often have better (bounds on) error.

2.5.2 Catastrophic Cancellation

Theorem 2.5: Catastrophic Cancellation

In general, catastrophic cancellation occurs when subtracting numbers of about the same magnitude, when the input numbers contain error.

Result 2.4: Sources of errors so far

Adding large and small numbers (very dierent magnitudes).

- Smaller digits get lost or "swamped"!
- Rule of thumb: Prefer to sum numbers of approximately same size and sign.

Subtracting nearby numbers that contain error.

- Loss of accuracy due to catastrophic cancellation.
- Rule of thumb: Try to reformulate computations to avoid cancellation.

2.6 Conditioning of Problems

Definition 2.11: Well-conditioned

For problem P, with input I and output O, if a "small" change to the input, ΔI, gives a "small" change in the output O, P is **well-conditioned**. Otherwise, P is **ill-conditioned**.

2.7 Stability of Algorithms

Definition 2.12: Unstable

If any initial error in the data is magnified by an algorithm, the algorithm is considered numerically **unstable**.

Discovery 2.5

Note that

- An algorithm can be unstable even for a well-conditioned problem!
- An ill-conditioned problem limits how well we can expect any algorithm to perform.

2.7.1 Stability Analysis of an Algorithm

Consider the integration problem,

$$I_n = \int_0^1 \frac{x^n}{x+\alpha}\, dx$$

for a given n where α is some fixed constant. ~~We state without proof that this is a well-conditioned problem (which could be solved by algorithms for numerical integration, sometimes called quadrature algorithms)~~ Easy to observe that we have a recursive algorithm to solve it, for $n \geq 0$:

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

This is an algorithm, and we would like to perform a stability analysis for the recurrence equation.

Example 2.12

Suppose that the floating point representation of I_0 introduces some error ε_0. For simplicity, assume that no other errors are introduced at any stage of the computation after I_0 is computed. Let $(I_n)_A$ be the approximate value of I_n, i.e., the computed value of In including the effects of ε_0. Let $(I_n)_E$ be the exact value of In. Let

$$\varepsilon_n = (I_n)_A - (I_n)_E$$

i.e., the error at step n due to the initial error ε_0. The exact I_n satisfies

$$(I_n)_E = \frac{1}{n} - \alpha (I_{n-1})_E$$

while the approximate I_n satisfies $(I_n)_A = \frac{1}{n} - \alpha (I_{n-1})_A$. Subtracting these two equations gives

$$(I_n)_E - (I_n)_A = \alpha (I_{n-1})_A - \alpha (I_{n-1})_E$$

or

$$\varepsilon_n = (-\alpha)\varepsilon_{n-1} = (-\alpha)^n \varepsilon_0$$

If $|\alpha| > 1$ then any initial error ε_0 is magnified by an unbounded amount as $n \to \infty$. On the other hand, if $|\alpha| < 1$ then any initial error is "damped out".

2.8 Exercises for Floating Point Numbers

Exercise 2.1

The numbers in a floating point system are defined by a base β, a mantissa length t, and an exponent range $[L, U]$. A nonzero floating point number x has the form

$$x = \pm .b_1 b_2 \ldots b_t \times \beta^e$$

Here $.b_1 b_2 \ldots b_t$ is the *mantissa* and e is the *exponent*. The exponent satisfies $L \leq e \leq U$. The b_i are base-β digits and satisfy $0 \leq b_i \leq \beta - 1$. Nonzero floating point numbers are normalized: $b_1 \neq 0$. Zero is represented by both zero mantissa and zero exponent.

(a) What is the largest value of n so that $n!$ can be exactly represented in a floating point system where $(\beta, t, L, U) = (2, 5, -10, 10)$. To obtain full marks, you must show your work.

(b) On a base-2 machine, the distance between 7 and the next largest floating point number is 2^{-12}. What is the distance between 70 and the next largest floating point number?

(c) Assume that x and y are normalized positive floating numbers in a base-2 computer with t-bit mantissa. How small can $y - x$ be if $x < 8 \leq y$?

Solution: (a) The largest possible value representable in this floating point number system is

$$0.11111 \times 2^{10}$$

which is equivalent to 992 in decimal. Thus the biggest value n can possibly take is 6.

1. I think the answer is 2^{-8}, this is because the mantissa $t = 15$.

★

Exercise 2.2

Consider a fictitious floating number system composed of the following numbers:

$$S = \left\{ \begin{array}{l} \pm b_1.b_2b_3 \times 2^{\pm y} : b_2, b_3, y = 0 \text{ or } 1, \\ \text{ and } b_1 = 1 \text{ unless } b_1 = b_2 = b_3 = y = 0 \end{array} \right\}$$

i.e., Each number is normalized unless it is a zero.

(a) Plot the elements of S on the real axis.

(b) Show how many elements are contained in S. What are the values of OFL, UFL, and the machine epsilon?

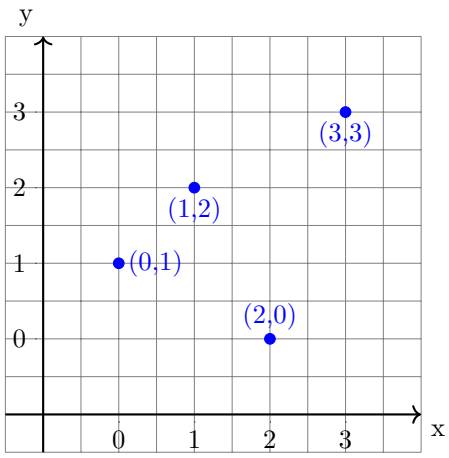
Exercise 2.3

Using the floating-point number format $(\beta, t, U, L) = (2, 20, -200, 200)$, store the distance between Earth and Sun (1.5×10^8 kilometers) and the distance between Toronto and Waterloo (75 kilometers). What length does the last bit of the mantissa represent in each case?

3 Interpolation

Lecture 4 - Wednesday, January 15

Given a set of data points, points (x_i, y_i) such that equation $y_i = p(x_i)$ is satisfied, from an (unknown) function $y = p(x)$, can we approximate p's value at other points? E.g.



Find a function p(x) that goes exactly through or interpolates all the points.

Comment 3.1

One solution for the above example coule be

$$y = \frac{4}{3}x^3 - \frac{11}{2}x^2 + \frac{31}{6}x + 1$$

but this solution is not necissarily unique.

Algorithm 3.1

We'll begin with methods for interpolating few points (often ≤ 6):

- Polynomial Interpolation
  - Vandermonde matrices
  - Lagrange form

Then, interpolation strategies for many points.

- Piecewise interpolants:
  - Piecewise linear
  - Cubic splines

3.1 Polynomial Interpolation

3.1.1 Unisolvence Theorem

Theorem 3.1: Unisolvence Theorem

Given n data pairs (x_i, y_i), $i = 1, \ldots, n$ with distinct x_i, there is a unique polynomial $p(x)$ of degree $\leq n-1$ that interpolates the data.

Proof. For n points, find all the coefficients c_i of the generic polynomial

$$p(x) = c_1 + c_2 x + c_3 x^2 + c_4 x^3 + \cdots + c_n x^{n-1}$$

As before, each (x_i, y_i) point gives one linear equation

$$y_i = c_1 + c_2 x_i + c_3 {x_i}^2 + c_4 {x_i}^3 + \cdots + c_n {x_i}^{n-1}$$

Then solve the $n \times n$ linear system. □

Example 3.1

Very first example had 4 (x_i, y_i) pairs: $(0,1)$, $(1,2)$, $(2,0)$, $(3,3)$. What is the linear system needed to recover the coefficients of the cubic polynomial?

Solution: We have

$$\begin{aligned} c_1 &= 1 \\ c_1 + c_2 + c_3 + c_4 &= 2 \\ c_1 + c_2 \cdot 2 + c_3 \cdot 4 + c_4 \cdot 8 &= 0 \\ c_1 + c_2 \cdot 3 + c_3 \cdot 9 + c_4 \cdot 27 &= 3 \end{aligned}$$

which yields us the solution $c_1 = 1$, $c_2 = 31/6$, $c_3 = -11/2$, and $c_4 = 4/3$. ★

3.1.2 Vandermonde Matrices

In general, we get a linear system:

$$\begin{bmatrix} 1 & x_1 & \cdots & {x_1}^{n-1} \\ 1 & x_2 & \cdots & {x_2}^{n-1} \\ & & \cdots & \\ 1 & x_n & \cdots & {x_n}^{n-1} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

or

$$V\vec{c} = \vec{y}$$

Definition 3.1: Vandermonde Matrix

V is called a **Vandermonde matrix**.

Discovery 3.1

The Vandermonde Matrix is invertible if all x_i's are distinct.

3.1.3 Monomial Basis

Definition 3.2: Monimial Form

The familiar form $p(x) = c_1 + c_2x + c_3x^2 + c_4x^3 + \cdots + c_nx^{n-1}$ is called the **monomial form**, and can also be written

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

Definition 3.3: Monomial Basis

The sequence $1, x, x_2, \ldots$ is called the **monomial basis**.

3.1.4 Lagrange Basis

Definition 3.4: Langrange Basis Polynomial

We will define the Lagrange basis functions, $L_i(x)$, to construct a polynomial as

$$p(x) = y_1L_1(x) + y_2L_2(x) + \cdots + y_nL_n(x) = \sum_i y_iL_i(x)$$

where y_i are the coefficients.

Comment 3.2

They are also our data values, $y_i = p(x_i)$.

Definition 3.5: $L_i(x)$

Given n data points (x_i, y_i), we define

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

Discovery 3.2

We notice that

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

Example 3.2

Consider the two points we fit a line to earlier:

$$(x_1, y_1) = (1, 2), \qquad (x_2, y_2) = (-1, 4)$$

What are the corresponding L_i functions, and the polynomial $p(x)$ in terms of the L_i's?

Solution: We have

$$p(x) = y_1 L_1(x) + y_2 L_2(x)$$

where

$$L_1(x) = \frac{x - x_2}{x_1 - x_2}, \qquad L_2(x) = \frac{x - x_1}{x_2 - x_1}$$

Hence

$$\begin{aligned} p(x) &= 2 \cdot \frac{x - x_2}{x_1 - x_2} + 4 \cdot \frac{x - x_1}{x_2 - x_1} \\ &= -x + 3 \end{aligned}$$

as desired. ★

Result 3.1

If they are the same polynomial in the end, why might we prefer the Lagrange basis to the monomial basis?

Answer: We can directly write down the polynomial from the Lagrange basis functions, L_i, and the data points, $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$.

No need to solve a linear system!

Discovery 3.3: Runge's phenomenon

Runge's phenomenon suggests that we need to use other methods for high-order polynomials as going to higher degrees doesn't necessarily yield a good "quality" fit ...

3.2 Piecewise Polynomials

Lecture 5 - Monday, January 20

We usually want continuity in our polynomials, so the simplest way is to fit a line in each pair of adjacent points. However, we want to also avoid the kinks at the data points.

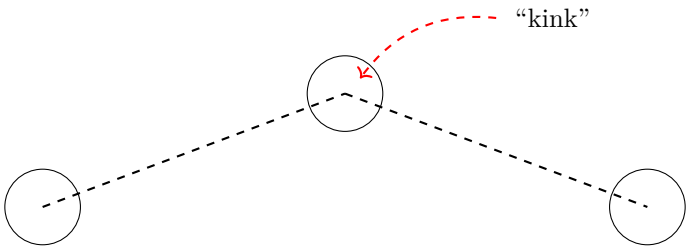


Figure 1: Example of kinks at data points.

Comment 3.3

We want greater smoothness, because

- For aesthetic purposes;
- For mathematical/numerical applications, we need (approximate) derivative information.

3.3 Hermite Interpolation

Definition 3.6: Hermite Interpolation

Hermite interpolation is the problem of fitting a polynomial given function values and *derivatives*.

Example 3.3

Given $p(0) = 0$, $p'(0) = 1$, $p(1) = 3$ and $p'(1) = 0$, determine the coefficients of the cubic polynomial $p(x) = a + bx + cx^2 + dx^3$.

Solution: We know that $p(0) = 0$, which tells us $a = 0$, so given that $p(1) = 3$, we know $b + c + d = 3$. Given $p'(0) = 1$, we have $b = 1$, and given $p'(1) = 0$, we have $b + 2c + 3d = 0$. Hence solving for c and d we get $c = 7$ and $d = -5$. ★

3.4 Piecewise Hermite Interpolation

Result 3.2

Notice that now

> We can fit many points given values and 1^{st} derivative with piecewise Hermite interpolation. Use one cubic per pair of (adjacent) points. Sharing the slope/derivative at points ensures first derivative continuity.

Closed-form Solution: Given points (x_i, y_i) and (x_{i+1}, y_{i+1}), let s_i, s_{i+1} to be the slope at the two points respectively. We define $p_i(x)$, the polynomial on the i^{th} interval, by

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

Hence we have

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & \Delta x_i & {\Delta x_i}^2 & {\Delta x_i}^3 \\ 0 & 1 & 2\Delta x_i & 3{\Delta x_i}^2 \end{bmatrix} \begin{bmatrix} a_i \\ b_i \\ c_i \\ d_i \end{bmatrix} = \begin{bmatrix} y_i \\ s_i \\ y_{i+1} \\ s_{i+1} \end{bmatrix}$$

solving for the polynomial coefficients we obtain:

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

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

Definition 3.7: Knots

Knots are points where the interpolant transitions from one polynomial / interval to another.

Definition 3.8: Nodes

Nodes are points where some control points/data is specified.

Discovery 3.4

For Hermite interpolation, knots and nodes are the same.

Question 3.1. For some applications we might still intend to create/control kinks

Solution: We specify different derivatives on either side of a node. ★

3.5 Cubic Splines Interpolation

More common problem: no derivative information s_i is given, only values y_i. Can we still fit a piecewise cubic to the set of points?

Algorithm 3.2

Fit a cubic, $S_i(x)$, on each interval, but now require matching first and second derivatives between intervals.

In particular, we require "interpolating conditions" on each interval $[x_i, x_{i+1}]$,

$$S_i(x_i) = y_i, \qquad S_i(x_{i+1}) = y_{i+1}$$

and "derivative conditions" at each interior point x_{i+1},

$$\begin{aligned} {S_i}'(x_{i+1}) &= {S_{i+1}}'(x_{i+1}) \\ {S_i}''(x_{i+1}) &= {S_{i+1}}''(x_{i+1}) \end{aligned}$$

3.5.1 Counting Unknowns

Assuming we have n data points. A cubic polynomial has 4 unknowns, and since we have $n-1$ intervals, there are in total $4n-4$ unknowns.

3.5.2 Counting Equations

Assuming we have n data points. For each interval we have two endpoints, which gives us in total $2(n-1)$ equations. We also get 2 derivative conditions per interior point, which yields us $2(n-2) = 2n-4$ equations.

Discovery 3.5

Notice that we do not have enough information for a unique solution. We need 2 more equations.

Theorem 3.2

The two more equations we needed are usually at domain endpoints, called boundary conditions or end conditions.

3.5.3 Boundary Conditions

Clamped/complete:

Definition 3.9: Clamped boundary conditions

Slope set to specific value.

$$S'(x_1) = specified, \qquad S'(x_n) = specified$$

If both boundaries clamped, it is a **complete or clamped spline**.

Free/natural/variational

Definition 3.10: Free boundary condition

Free boundary condition: Second derivatives set to zero.

$$S''(x_1) = 0, \qquad S''(x_n) = 0$$

If both boundaries are free, called a **natural cubic spline**.

Periodic

Definition 3.11: Periodic boundary conditions

Periodic Boundary Condition:

$$S'(x_1) = S'(x_n), \qquad S''(x_1) = S''(x_n)$$

Start and end derivatives match each other

Result 3.3

Hermite interpolation — each interval can be found independently.

- Solve $n-1$ separate systems of 4 equations.

Cubic spline — must solve for all polynomials together at once!

- Solve one system of with $4(n-1)$ equations.

3.6 Cubic splines via Hermite interpolation

Algorithm 3.3

1. Express unknown polys with closed form Hermite equations.
2. Treat the s_i (slopes at nodes) as unknowns.
3. Solve for s_i that give continuous 2^{nd} derivatives — i.e., force it to satisfy cubic spline definition.
4. Given s_i, plug into closed form Hermite equations to recover poly coefficients: a_i, b_i, c_i, d_i.

For cubic splines, we had three types of conditions (ignoring ends).

1. Values match at all interval endpoints.
2. First derivatives match at interior points.
3. Second derivatives match at interior points.

Comment 3.4

Notice that (1) and (2) are satisfied aleady by Hermite, to achieve item (3) we need to find s_i satisfying ${S_i}''(x) = {S_{i+1}}''(x)$.

3.6.1 Derivation of Cubic Splines Equations

We start with Hermite formulas:

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

The second derivative is

$${S_i}''(x) = 2c_i + 6d_i(x - x_i) = \frac{2(3{y_i}' - 2s_i - s_{i+1})}{\Delta x_i} + \frac{6(s_{i+1} + s_i - 2{y_i}')}{{\Delta x_i}^2}(x - x_i)$$

To force matching second derivative,

$${S_i}''(x_{i+1}) = {S_{i+1}}''(x_{i+1}) \qquad \text{for } i = 1, \ldots, n-2$$

Comment 3.5

Recall

$$\begin{aligned} {S_i}''(x_{i+1}) &= \frac{2({y_i}' - 2s_i - s_{i+1})}{\Delta x_i} + \frac{6(s_{i+1} + s_i - 2{y_i}')}{{\Delta x_i}^2}(x_{i+1} - x_i) \\ &= \frac{2({y_i}' - 2s_i - s_{i+1})}{\Delta x_i} + \frac{6(s_{i+1} + s_i - 2{y_i}')}{\Delta x_i} \\ {S_{i+1}}''(x_{i+1}) &= \frac{2({y_{i+1}}' - 2s_{i+1} - s_{i+2})}{\Delta x_{i+1}} + \frac{6(s_{i+2} + s_{i+1} - 2{y_{i+1}}')}{{\Delta x_{i+1}}^2}(x_{i+1} - x_{i+1}) \\ &= \frac{2({y_{i+1}}' - 2s_{i+1} - s_{i+2})}{\Delta x_{i+1}} \end{aligned}$$

Hence we have

$$\begin{aligned} \frac{2({y_i}' - 2s_i - s_{i+1})}{\Delta x_i} + \frac{6(s_{i+1} + s_i - 2{y_i}')}{\Delta x_i} &= \frac{2({y_{i+1}}' - 2s_{i+1} - s_{i+2})}{\Delta x_{i+1}} \\ \Delta x_i({y_{i+1}}' - 2s_{i+1} - s_{i+2}) &= \Delta x_{i+1}({y_i}' - 2s_i - s_{i+1} + 3(s_{i+1} + s_i - 2{y_i}')) \\ &= \Delta x_{i+1}(2s_{i+1} + s_i - 3{y_i}') \\ 3\Delta x_i {y_{i+1}}' + 3\Delta x_{i+1}{y_i}' &= \Delta x_{i+1}(2s_{i+1} + s_i) + \Delta x_i(2s_{i+1} + s_{i+1}) \\ &= \Delta x_{i+1}s_i + \Delta x_i s_{i+2} + 2s_{i+1}(\Delta x_i + \Delta x_{i+1}) \end{aligned}$$

or, setting $i \to i - 1$ yields us,

$$\begin{aligned} 3\Delta x_{i-1}{y_i}' + 3\Delta x_i {y_{i-1}}' =& \Delta x_i s_{i-1} + \Delta x_{i-1}s_{i+1} + 2s_i(\Delta x_{i-1} + \Delta x_i) \\ &\text{for } i = 2, \ldots, n-1 \end{aligned}$$

which gives us $n - 2$ equations for n unknowns as anticipated. Two extra equations are determined from the boundary equations:

Result 3.4

1. Clamped BC:
$$s_1 = {s_1}^*, \qquad s_n = {s_n}^*$$
2. Free BC:
$$s_1 + \frac{s_2}{2} = \frac{3}{2}y_1', \qquad \frac{s_{n-1}}{2} + s_n = \frac{3}{2}y_{n-1}'$$

Lecture 6 - Wednesday, January 22

3.6.2 Efficient Cubic Splines — Matrix Form

Theorem 3.3

We can write this linear system for the slopes in matrix/vector form, $T \cdot \vec{s} = \vec{r}$. Having solved for s, we can recover the a, b, c, d coefficients.

Example 3.4

What is the linear system for s_i to fit a spline to the 4 points $(0,1)$, $(2,1)$, $(3,3)$, $(4,-1)$ with clamped BC of $s_1 = 1$ and $s_4 = -1$?

Solution: Let's first compute all Δx_i's and all the y_i''s:

$$\begin{array}{l|l} \Delta x_1 = 2-0 = 2 & y_1' = \frac{1-1}{2} = 0 \\ \Delta x_2 = 3-2 = 1 & y_2' = \frac{3-1}{1} = 2 \\ \Delta x_3 = 4-3 = 1 & y_2' = \frac{-1-3}{1} = -4 \end{array}$$

Now we find the rows for $i = 1$. We know that $i = 1$ is a boundary, namely $s_1 = 1$, so

$$T_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}, \qquad r_1 = 1$$

$i = 2$ is an interior, so we have

$$\Delta x_2 s_1 + 2(\Delta x_2 + \Delta x_1)s_2 + \Delta x_1 s_3 = 3(\Delta x_2 y_1' + \Delta x_1 y_2')$$

which yields us $s_1 + 6x_2 + 2s_3 = 12$, sop

$$T_2 = \begin{bmatrix} 1 & 6 & 2 & 0 \end{bmatrix}, \qquad r_2 = 12$$

Likewise, we could obtain the system $T \cdot \vec{s} = \vec{r}$ as following:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 6 & 2 & 0 \\ 0 & 1 & 4 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \\ s_3 \\ s_4 \end{bmatrix} = \begin{bmatrix} 1 \\ 12 \\ -6 \\ -1 \end{bmatrix}$$

Solving for s_i's and plug into the Hermite closed form, we get coefficients a_i, b_i, c_i and d_i for each cubic polynomial $S_i(x)$. ★

Matrix Size: New system (for s_i): one equation per node, so matrix size is $n \times n$. Old system (for a_i, b_i, c_i, d_i): 4 coefficients each for $n-1$ intervals, so the matrix size is $(4n-4) \times (4n-4)$.

Matrix Structure: The matrix is tridiagonal. Only the entries on the diagonal and its two neighbours are ever non-zero!

Example 3.5

Give the conditions that should be satisfied for the piecewise function $S(x)$ to be a valid cubic spline, assuming we already know:

$$S(x) = \begin{cases} \frac{5}{3} + \frac{16}{3}x + ax^2 + x^3 & \text{on } [-1,1] \\ -\frac{7}{3} + bx + \frac{22}{3}x^2 + \frac{2}{3}x^2 & \text{on } [1,2] \end{cases}$$

Is there a choice of a and b to make $S(x)$ a valid cubic spline?

Solution: In this case, we only need to check the interpolating and derivative conditions hold at x_1, since no boundary conditions are stated. Interpolating:

$$S_1(1) = S_2(1) \quad \Longrightarrow \quad \frac{5}{3} + \frac{16}{3} + a + 1 = -\frac{7}{3} + b + \frac{22}{3} + \frac{2}{3}$$

First derivative:

$$S_1'(1) = S_2'(1) \quad \Longrightarrow \quad \frac{16}{3} + 2a + 3 = b + \frac{44}{3} + 2$$

Second derivative:

$$S_1''(1) = S_2''(1) \quad \Longrightarrow \quad 2a + 6 = \frac{44}{3} + 4$$

Now what's left to do is to check whether there exists values of a and b satisfying all three equations. ★

3.7 Exercises for Interpolation

Exercise 3.1

(a) Show that there is a unique cubic polynomial $p_3(x)$ for which

$$p_3(x_0) = f(x_0), \quad p_3(x_2) = f(x_2), \quad p_3'(x_1) = f'(x_1), \quad p_3''(x_1) = f''(x_1)$$

where $f(x)$ is a given function and $x_0 \neq x_2$.

(b) Derive base functions analogous to the Lagrange basis for $p_3(x)$.

Exercise 3.2

For Lagrange polynomial interpolation, n data points (x_i, y_i), $i = 1, 2, \ldots, n$ are given.

(a) What is the degree of each polynomial function $L_j(x)$ in the Lagrange basis?

(b) What function results if we sum the n functions in the global Lagrange basis, that is, what is

$$g(x) = \sum_{j=1}^{n} L_j(x)$$

Explain.

Solution: For part (a), it should be $n - 1$/ For part (b), the interpolating polynomial of f on the distinct points $x_0, \ldots, x_n$ is given by

$$g(n) = \sum_j f(x_j) L_j(x)$$

Therefore, if $f(x) = 1$, then we have

$$1 = \sum_j L_j(x)$$

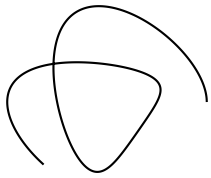
as desired. ★

Exercise 3.3

Let $S(x)$ be the natural spline interpolant of x^3 at $x = -3$, $x = -1$, $x = 1$ and $x = 3$. What is $S(0)$?

4 Parametric Curve

So far we've only handled functions $y = p(x)$, i.e. one input corresponds to one output. However, there are cases where the graph looks something like the following



In this case, it is impossible to find a *function*. Therefore we need to introduce "**Parametric curves**", which will let us handle more general curves.

Question 4.1. General Problem

For a polynomial function $y = p(x)$, x uniquely dictates y So two distinct points cannot have the same x coordinate!

Solution:

Let x and y each be separate functions of a new parameter, t. Then a point's position is given by the vector $\vec{P}(t) = (x(t), y(t))$.

★

4.1 Examples

4.1.1 Line Example

Discovery 4.1

The simple line $y = 3x + 2$ can equivalently be described by the two coordinate functions:

$$x(t) = t,$$
$$y(t) = 3t + 2$$

4.1.2 Semi-circle Example Version — 1

Consider a curve along a semi-circle with radius of 1 in the upper half plane, oriented from $(1, 0)$ to $(-1, 0)$. The usual implicit equation for a unit circle is $x^2 + y^2 = 1$.

Discovery 4.2

One parametric form is:

$$x(t) = \cos(\pi t), \qquad y(t) = \sin(\pi t) \qquad \text{for } 0 \leq t \leq 1.$$

Comment 4.1

A given curve can be "parameterized" in different ways, while yielding the exact same shape.

4.1.3 Semi-circle Example Version — 2

Consider a curve along a semi-circle with radius of 1 in the upper half plane, oriented from $(-1, 0)$ to $(1, 0)$.

Discovery 4.3

One parametric form is:

$$x(t) = \cos(\pi(1-t)), \qquad y(t) = \sin(\pi t) \qquad \text{for } 0 \le t \le 1.$$

Lecture 7 - Monday, January 27

4.1.4 Semi-circle Example Version — 3

Consider a curve along a semi-circle with radius of 1 in the upper half plane, oriented from $(1, 0)$ to $(-1, 0)$.

Discovery 4.4

An alternative parametric form is:

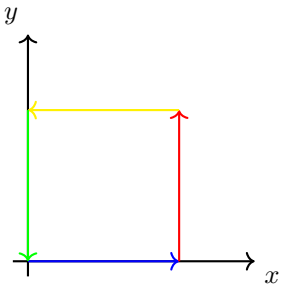
$$x(t) = \cos(\pi t^2), \qquad y(t) = \sin(\pi t^2) \qquad \text{for } 0 \le t \le 1.$$

The 2 parameterizations traverse the curve in the same direction, but at *different speeds/ rates.*

4.1.5 Square Example

A square can also be described as a parametric curve, using a piecewise definition.

$$\begin{array}{lrll}
(1) & x(t) = t, & y(t) = 0 & \text{for } 0 \le t \le 1 \\
(2) & x(t) = 1, & y(t) = t - 1 & \text{for } 1 \le t \le 2 \\
(3) & x(t) = 1 - (t-2), & y(t) = 1 & \text{for } 2 \le t \le 3 \\
(4) & x(t) = 0, & y(t) = 1 - (t-3) & \text{for } 3 \le t \le 4
\end{array}$$



Comment 4.2

Visually, this curve is not smooth. Mathematically, this is reflected in the fact that $x(t)$ and $y(t)$ do not have derivatives at $t = 1, 2, 3$.

4.2 Interpolating Curve Data by a Parametric Curve

Code 4.1: Arc-Length Parameterization

A common parameterization is to choose t as the distance along the curve.

Definition 4.1: Parametric Curve

Parametric curves are just the general concept of using a parameter to control multiple curves coordinates separately. Not a specific curve type!

We can combine all our existing interpolant types with parametric curves, by considering x(t) and y(t), separately.

- Use two Lagrange polynomials, $x(t)$, $y(t)$, to fit a small set of (t_i, x_i, y_i) point data.
- Use Hermite interpolation for $x(t)$, $y(t)$ given $(t_i, x_i, y_i, sx_i, sy_i)$ point/derivative data for many points.
- Fit separate cubic splines to $x(t)$, $y(t)$, given many points.

Discovery 4.5

Given ordered (x_i, y_i) point data, we don't yet have a parameterization. We need data for t, to form (t_i, x_i) and (t_i, y_i) pairs to fit curves to.

Algorithm 4.1

- **Option 1:** Use the node index as the parameterization, i.e. $t_i = i$ at each node.
- **Option 2:** (approximate arc-length parameterization): Set $t_1 = 0$ at first node. Recursively compute $t_{i+1} = t_i + \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}$.

Result 4.1

Expressing curves in a parametric form enables more general curves:

- No restriction on x or y values, provided t parameter is monotonic.
- Curves can have different parameterizations, which vary in their speed and direction.
- We can use them with any of our polynomial curve fitting methods.

5 Ordinary Differential Equations

Applied mathematicians or other scientists often need to construct or define a *mathematical model* of a problem. n many cases, these can take the form of an ODE.

"All models are wrong, but some are useful."

Definition 5.1: ODE

ODE (**ordinary differential equations**) are in the form of

$$F(x, y(x), y'(x), \ldots, y^{(n)}(x)) = 0$$

where $y^{(i)}(x)$ is the i^{th} derivative of $y(x)$.

Definition 5.2: Order

The highest power in terms of the derivative in an ode is called the **order** of the ODE.

Example 5.1

For eaxmple, first order differential equations are in the form of

$$F(x, y(x), y'(x)) = 0$$

5.1 Example: A Simple Population Model

Question 5.1.

Consider a mouse population, $y(t)$, over time. With enough food, we'll say the population changes as

$$y'(t) = a \cdot y(t)$$

where a is some constant (reproduction rate).

Comment 5.1

$y(t)$ is not given explicitly (in closed form)!

Solution: For this simple ODE with initial pop., $y_0 = y(t_0)$, there is a closed-form solution:

$$y(t) = y_0 \cdot \exp(a \cdot (t - t_0))$$

Aside: How can we verify it? To verify, we compute $y'(t)$ by differentiating the closed form $y(t)$ and compare:

$$y'(t) = \frac{d}{dx}\left[y_0 \cdot \exp(a \cdot (t - t_0))\right] = y_0 \cdot \exp(a \cdot (t - t_0)) \cdot a = a \cdot y(t)$$

which aligns with that of the ODE. ★

Question 5.2.

In reality, food supplies (and space and partners and ...) are limited. Consider a new model,

$$y'(t) = y(t) \cdot (a - b \cdot y(t))$$

where the b term expresses the effect of resource limits.

Discovery 5.1

- For small $y(t)$, we again have $y'(t) \approx a \cdot y(t)$. (Exponential growth.)
- For $y(t) \approx a/b$, we have $y'(t) \approx 0$. Population growth levels off!

Solution: This new population growth model also has a closed-form

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

This is logistic growth. ★

A very slightly more complex model is:

$$y'(t) = y(t) \cdot (a(t) - b(t) \cdot y(t)^\alpha)$$

This already has no general closed form solution. Even (fairly) simple mathematical models often lack closed form solutions, except in special cases.

Theorem 5.1: (Partial) Solution

We will develop "numerical methods" to instead find approximate solutions.

5.2 Approximating Methods

Definition 5.3: IVP

Initial Value Problem (IVP) is a differential equation

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

where f is specified, and the initial values are

$$y(t_0) = y_0$$

5.2.1 System of Differential Equations

Consider a model with multiple variables that interact. E.g. x and y coordinates of a moving object. This gives a system of differential equations, such as

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

which can be written in the vector form

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

or in vector notation

$$(x, y)' = \vec{f}(t, (x, y)(t)), \qquad \text{with}(x, y)(t_0) = (x_0, y_0)$$

Comment 5.2

The matrix notation is specially helpful when f_x and f_y are linear. If that is the case, then the $\vec{f}$ can be decomposed as a constant matrix and a vector containing the functions.

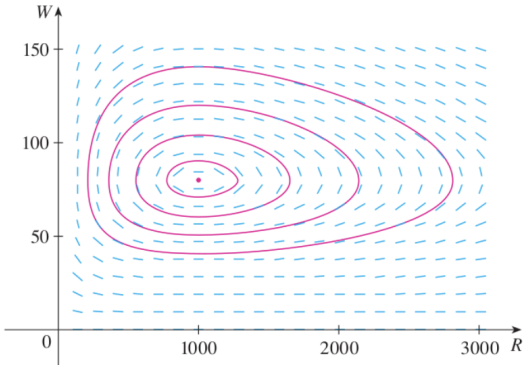
Example 5.2: Predator-Prey Systems

We have

$$R'(t) = k \cdot R(t) - a \cdot R(t) \cdot W(t)$$
$$W'(t) = -r \cdot W(t) + b \cdot R(t) \cdot W(t)$$

where

- $R(t)$: # of rabbits at time t
- $W(t)$: # of wolves at time t
- a, b, k and r are positive constants



5.2.2 Higher Order Differential Equations

Higher derivatives occur often in real problems. The order is the highest derivative in the equation:

$$y^{(n)}(t) = f(t, y(t), y'(t), y''(t), \ldots, y^{(n-1)}(t))$$

5.3 Approximating Methods

Definition 5.4: Time Stepping Method

The most common class of methods for determining a numerical solution for a first order initial value problem are called `time stepping methods`.

A time step is the interval $h_n = t_{n+1} - t_n$, which is determined by the method. Time stepping methods carry a candidate size hcand for the next time step which may be revised during each time step.

They have the following general form:

Algorithm 1: Example of a numerical integration algorithm

Data: Initialize $y_0, t_0, h_{\text{cand}}, n = 0$

1 **repeat**

2 Compute y_{n+1} and h_n using data $t_n, y_n, h_{\text{cand}}$ and $f(t, z)$

3 $t_{n+1} \leftarrow t_n + h_n$

4 Recompute h_{cand}

5 $n \leftarrow n + 1$

6 **until** *done*

Line 2 combines both advancing the solution (computing $y^{(n+1)}$) and time step size selection (computing hn), although line 4 is also part of time step size selection.

5.4 The Forward Euler Method

Definition 5.5: Forward Euler Method

Forward Euler is an explicit, single-step scheme.

Algorithm 5.1

Compute the current slope:

$$y'_n = f(t_n, y_n)$$

Step in a straight line with that slope:

$$y_{n+1} = y_n + h \cdot y'_n$$

Repeat.

Discovery 5.2

Time-stepping applies a recurrence relation to approximate the function values at later and later times

Comment 5.3

Later we'll analyze the stability of time-stepping methods.

Forward Euler — Summary

Theorem 5.2: Forward Euler Scheme

The Forward Euler Scheme is

$$y_{n+1} = y_n + h \cdot f(t_n, y_n)$$

Lecture 8 - Thursday, January 30

5.4.1 Example

Question 5.3.

Consider the simple IVP $y'(t) = 2y(t)$, with initial conditions at $t_0 = 1$ we have $y(t_0) = 3$.

Solution: We will do the following:

Algorithm 5.2

1. Write down the recurrence for Forward Euler, with step size $h = 1$.
2. Use forward Euler to approximate y at time $t = 5$.
3. Compare against the true solution, $y(t) = 3\exp[2(t - t_0)]$.
4. Repeat for $h = 1/2$.

The forward Euler formula is given by

$$y_{n+1} = y_n + 1 \cdot 2y_n = 3y_n$$

Therefore, given the initial condition, we have the following table of values for $h = 1$:

n	t_n	y_n	$y(t_n)$
0	1	3	3
1	2	9	22.2
2	3	27	163.8
3	4	81	1210
4	5	243	8943

where the rightmost column contains exact solutions. Repeat for $h = 1/2$, we obtain

n	t_n	y_n	$y(t_n)$
0	1	3	3
1	1.5	6	8.15
2	2	12	22.2
3	2.5	24	60.3
4	3	48	163.8
5	3.5	96	445
6	4	192	1210

★

Comment 5.4

$y(t_n)$ is the exact value of the true function y at time t_n. y_n is the approximate/discrete data at step n, i.e. at time t_n, $y_n \approx y(t_n)$.

5.4.2 Systems of Equations

Example 5.3

For systems of 1^{st} order ODEs, we apply Forward Euler to each row in exactly the same way. For example, given the system of differential equations:

$$\begin{bmatrix} y'(t) \\ z'(t) \end{bmatrix} = \begin{bmatrix} z(t) \\ tz(t) - ay(t) + \sin(t) \end{bmatrix}$$

with initial conditions:

$$\begin{bmatrix} y(0) \\ z(0) \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \end{bmatrix}$$

Applying Forward Euler gives a (vector) recurrence, we get:

$$\begin{bmatrix} y_{n+1} \\ z_{n+1} \end{bmatrix} = \begin{bmatrix} y_n \\ z_n \end{bmatrix} + h \begin{bmatrix} z_n \\ t_n z_n - a y_n + \sin(t_n) \end{bmatrix}$$

with the same initial conditions.

Question 5.4.

Consider a particle with coordinates $(x(t), y(t))$ satisfying the ODE system:

$$\begin{cases} x'(t) = -y(t) \\ y'(t) = x(t) \end{cases}$$

with initial conditions:

$$x(t_0) = 2, \quad y(t_0) = 0, \quad \text{and} \quad t_0 = 2$$

Solution: We wish to solve three steps with $h = 2$: In vector form, we have

$$\begin{bmatrix} x'(t) \\ y'(t) \end{bmatrix} = \begin{bmatrix} -y(t) \\ x(t) \end{bmatrix}$$

Apply forward Euler to this problem, we have

$$\begin{bmatrix} x_{n+1} \\ y_{n+1} \end{bmatrix} = \begin{bmatrix} x_n \\ y_n \end{bmatrix} + 2 \begin{bmatrix} -y_n \\ x_n \end{bmatrix} = \begin{bmatrix} x_n - 2y_n \\ y_n + 2x_n \end{bmatrix}$$

Hence we have

$$\text{Step 1:} \quad \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} 2 - 2(0) \\ 0 + 2(2) \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \end{bmatrix}$$

$$\text{Step 2:} \quad \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} 2 - 2(4) \\ 4 + 2(2) \end{bmatrix} = \begin{bmatrix} -6 \\ 8 \end{bmatrix}$$

$$\text{Step 3:} \quad \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} -22 \\ -4 \end{bmatrix}$$

★

5.4.3 Deriving Forward Euler

Theorem 5.3

Forward Euler Method works.

Proof. There are two ways to arrive at Forward Euler:

- `''Finite difference'' view.` We approximate the derivative y' with a finite difference. Recall the ODE form:

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

where the fintie difference approximation of y' gives

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

Rearranging yields

$$y_{n+1} = y_n + h \cdot f(t_n, y_n)$$

- `Taylor series view.` A Taylor series approximates a function in some neighbourhood using an infinite weighted sum of its derivatives. *Including more terms in the series gives better estimates of the function value.*

$$f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f'''(a)}{3!}(x-a)^3 + \cdots$$

We wish to approximate $y(t_{n+1})$ from t_n data. Hence

$$f \mapsto y, \quad a \mapsto t_n, \quad x \mapsto t_{n+1}$$

In other words, we have

$$\begin{aligned} y(t_{n+1}) &= y(t_n) + y'(t_n)(t_{n+1} - t_n) + \frac{y''(t_n)}{2!}(t_{n+1} - t_n)^2 + \frac{y'''(t_n)}{3!}(t_{n+1} - t_n)^3 + \cdots \\ &= y(t_n) + hy'(t_n) + \frac{h^2}{2!}y''(t_n) + \frac{h^3}{3!}y'''(t_n) + \cdots \end{aligned}$$

since $(t_{n+1} - t_n) =: h$. Assume that terms of order two or higher will be small as $h \to 0$, we can drop them and replace y' with an evaluation of the dynamics function f:

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

thus we have completed our proof. □

5.4.4 Error of Forward Euler Method

Forward Euler makes a linear approximation at each step, incurring a "local" error.

Discovery 5.3

Notice that

Smaller step size $h \to$ more frequent slope estimates $\to$ less error

Recall the Taylor series,

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

We notice that the difference between it and forward Euler is

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

Definition 5.6: LTE

The above error is known as **local truncation error**.

Comment 5.5

Notice that as h decreases, the error decreases quodratically.

5.5 Trapezoidal Rule ("Crank-Nicolson") and Modified Euler Methods

To approximate more accuretely, we could keep more terms in the taylor series.

Question 5.5. H

owever, notice that we do not know y''.

Solution: Use a finite difference to approximate y'':

$$y'' \approx \frac{y'(t_{n+1}) - y'(t_n)}{h} + O(h)$$

★

Result 5.1

Now we have

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

Since $y'(t) = f(t, y(t))$, we have

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

Comment 5.6

Therefore the local truncation error for trapezoidal rule is $O(h^3)$. Reducing step size h now reduces per-step error cubically in h!

Discovery 5.4

Observe that trapezoidal yields us an implicit formula, which is hard to solve. Hence we have the following challenge:

Question 5.6.

How can we make trapezoidal explicit?

5.5.1 Improved Forward Euler

Algorithm 5.3: Improved Forward Euler

- Take forward Euler step to estimate the end point.
- Evaluate slope there.
- Use this approximate end-of-step slope in the trapezoidal formula.

Lecture 9 - Monday, February 03

In particular, we have

$$y^*_{n+1} = y_n + h \cdot f(t_n, y_n)$$

and hence the improved Euler is

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

Discovery 5.5

Like trapezoidal, improved Euler has local truncation error (LTE) $O(h^3)$.

Theorem 5.4

Improved Euler works. The proof below is the derivation.

Proof. If we have a function of multiple variables, we can Taylor expand in one variable ...

$$f(x, y + h_y) = f(x, y) + h_y \cdot \frac{\partial f}{\partial y} + O(h_y^2)$$

or multiple variables,

$$f(x + h_x, y + h_y) = f(x, y) + h_x \frac{\partial f}{\partial x} + h_y \frac{\partial f}{\partial y} + O(h_x^2) + O(h_y^2)$$

Trapezoidal gives

$$y(t_{n+1}) = y(t_n) + \frac{h}{2}\left[f(t, y_n) + f(t_{n+1}), y_{n+1}\right] + O(h^3)$$

and Forward Euler gives

$$y(t_{n+1}) = y(t_n) + h \cdot f(t_n, y(t_n)) + O(h^2)$$

Now, let

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

Then the error is

$$y(t_{n+1}) - y^*_{n+1} = O(h^2)$$

We know that

$$f(x, y + h_y) = f(x, y) + h_y \cdot \frac{\partial f}{\partial y} + O(h_y^2)$$

Taylor expanding f at y^*_{n+1} gives us that

$$f(t_{n+1}, y(t_{n+1})) = f(t_{n+1}, y^*_{n+1}) + \frac{\partial f}{\partial y}(t_{n+1}, y^*_{n+1}) \cdot \underbrace{(y(t_{n+1}) - y^*_{n+1})}_{O(h^2)} + \underbrace{O((y(t_{n+1}) - y^*_{n+1})^2)}_{O(h^4)}$$

Therefore,

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

Use this approximate end-of-step slope in the Trapezoidal expression gives,

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

Improved Euler has local truncation error (LTE) of $O(h^3)$, like the trapezoidal method (but with different error coefficients). □

5.5.2 Improved Euler / Trapezoidal Example

Example 5.4: Improved Euler / Trapezoidal Example

We previously applied Forward Euler (F.E.) to the ODE system:

$$x'(t) = -y(t)$$
$$y'(t) = x(t)$$

with initial conditions $x(t_0) = 2$, $y(t_0) = 0$, $t_0 = 0$. We wish to apply improved Euler with time step size $h = 2$ to find x, y at $t = 4$.

Solution: In this problem, we have from Forward Euler:

$$\begin{bmatrix} x^*_{n+1} \\ y^*_{n+1} \end{bmatrix} = \begin{bmatrix} x_n \\ y + n \end{bmatrix} + 2 \begin{bmatrix} -y_n \\ x_n \end{bmatrix}$$

and from Trapezoidal, we have

$$\begin{bmatrix} x^*_{n+1} \\ y^*_{n+1} \end{bmatrix} = \begin{bmatrix} x_n \\ y_n \end{bmatrix} + \frac{2}{2} \begin{bmatrix} -y_n - y^*_{n+1} \\ x_n + x^*_{n+1} \end{bmatrix}$$

Apply to our IVP, we have

	Forward Euler			Improved Euler	
Step 1:	$\begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$	$= \begin{bmatrix} 2-2(0) \\ 0+2(2) \end{bmatrix}$	$= \begin{bmatrix} 2 \\ 4 \end{bmatrix}$	$\begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$	$= \begin{bmatrix} -2 \\ 4 \end{bmatrix}$
Step 2:	$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix}$	$= \begin{bmatrix} 2-2(4) \\ 4+2(2) \end{bmatrix}$	$= \begin{bmatrix} -6 \\ 8 \end{bmatrix}$	$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix}$	$= \begin{bmatrix} -4 \\ -8 \end{bmatrix}$

★

5.6 Global vs Local Error

Definition 5.7: Local (Truncation) Error

For a few methods, we saw order of the error for a single step – this is the **local (truncation) error**.

Definition 5.8: Global Error

The **Global error** is the total error accumulated at the final time, t_{final}.

Discovery 5.6

For a constant step h, computing from t_0 to end time t_{final},

$$\#\text{steps} = \frac{t_{\text{final}} - t_0}{h} = O(h^{-1})$$

Then, for a given method we have (roughly):

$$\text{Global Error} \leq (\text{Local Error}) \cdot O(h^{-1})$$

i.e., one degree lower.

5.6.1 Single-Step vs. Multistep schemes

Definition 5.9: Single-Step

The single-step methods we've discussed use info from the current time t_n (and forward) to solve for y_{n+1}.

Definition 5.10: Multistep

Multi-step methods also use information from earlier timesteps, that is, t_{n-1}, t_{n-2}, etc.

5.7 ODEs More Schemes

5.7.1 Backwards (Implicit) Euler method

Algorithm 5.4

Backwards Euler uses the slope from only the end of the step:

$$y_{n+1} = y_n + h \cdot f(t_{n+1}, y_{n+1})$$

Its local truncation error is $O(h^2)$, like forward Euler.

5.7.2 Explicit "Runge Kutta" schemes

Recall improved Euler:

1. *"FE-Step":*
$$y^*_{n+1} = y_n + hf(t_n, y_n)$$
2. *"Trapezoidal-Step":*
$$y_{n+1} = y_n + \frac{h}{2}\left[f(t_n, y_n) + f(t_{n+1}, y^*_{n+1})\right]$$

Algorithm 5.5

It can also be written as:

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

Definition 5.11: Runge Kutta methods

A family of explicit schemes, often written this way, are called **Runge Kutta methods**.

Result 5.2

Similar schemes exist for higher orders, $O(h^\alpha)$ for $\alpha = 3, 4, 5, 6, \ldots$. **"Classical" Runge-Kutta, or "RK4",** with LTE of $O(h^5)$:

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

5.7.3 Implicit multistep schemes: BDF methods

Definition 5.12: BDF

BDF stands for Backwards Differentiation Formulas.

Comment 5.7

BDF1, BDF2, BDF3, etc. – number indicates order of global error.

Example 5.5: BDF1

"BDF1" is backward Euler, so it's just a single-step scheme.

Example 5.6: BDF2

BDF2 uses current and previous step data:

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

BDF2 has local truncation error $O(h^3)$, and is a multi-step scheme.

Lecture 10 - Wednesday, February 05

5.7.4 Derive BDF

Theorem 5.5

Deriving BDF1 methods via Interpolation.

Proof. **idea of the proof:**

1. Fit an interpolant $p(t)$ with Lagrange polynomials to the unknown point, (t_{n+1}, y_{n+1}), and one or more earlier points.
2. Determine its derivative, $p'(t)$, by differentiating.
3. Require end-of-step slope to match: $p'(t_{n+1}) = f(t_{n+1}, y_{n+1})$ (i.e., dynamics function slope matches polynomial's derivative there.) Rearranging for unknown y_{n+1} gives a BDF scheme.

Now we may start the derivation. Fit a Langrange polynomial $p(t)$ to (t_n, y_n) and (t_{n+1}, y_{n+1}):

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

Taking derivative with respect to t we get

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

which gives the slope of the line. We require it to match the end point's slope, which is given by

$$f(t_{n+1}, y_{n+1})$$

Hence

$$p'(t) = \frac{y_{n+1} - y_n}{h} = f(t_{n+1}, y_{n+1})$$
$$\implies \quad y_{n+1} = y_n + h \cdot f(t_{n+1}, y_{n+1})$$

which is the Backwards (implicit) Euler's method, a.k.a. BDF1. □

Question 5.7. Exercise: Derive BDF2 via Interpolation

Now, fit Lagrange polynomial to 3 timesteps of data: t_{n-1}, t_n, t_{n+1}.

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

Determine $p'(t)$. Require $p'(t_{n+1}) = f(t_{n+1}, y_{n+1})$, like for BDF1. Work through the details to get the BDF2 recurrence.

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

5.7.5 Explicit multistep schemes: Adams-Bashforth

Example 5.7

We have 2^{nd} order Adams-Bashforth:

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

Discovery 5.7

LTE is $O(h^3)$.

5.7.6 Summary

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	$O(h^3)$

5.8 High Order ODEs — Convert higher order into first order system

Definition 5.13:

The general form for such a higher ODE looks like:

$$y^{(n)}(t) = f(t, y(t), y'(t), y''(t), y'''(t), \ldots, y^{(n-1)}(t))$$

i.e., all derivatives may be inter-related.

Comment 5.8

Fortunately, we can convert them to systems of first order ODE's. Then we can use all the time-stepping schemes we've seen.

Algorithm 5.6

For each variable y with more than a first derivative, introduce new variables

$$y_i = y^{(i-1)}$$

for $i = 1$ to n, so each derivative has a corresponding new variable. Substituting the new variables into the original ODE leads to:

1. One first order equation for each original equation.
2. One or more additional equations relating the new variables.

Example 5.8

Consider the IVP

$$y''(t) = t \cdot y(t)$$

with initial conditions, $y(1) = 1$, $y'(1) = 2$.

Solution: We introduce

$$y_1(t) = y^{(0)}(t) = y(t), \qquad y_2(t) = y^{(1)}(t) = y'(t)$$

and hence the system of 1^{st} order ODE's is:

$$y_1'(t) y_2(t)$$
$$y_2'(t) = t y_1(t)$$

with initial conditions $y_1(1) = 1$ and $y_2(1) = 2$. ★

Example 5.9

Convert

$$x''(t) + y'(t)x(t) + 2t = 0$$
$$y''(t) + (y(t))^2 x(t) = 0$$

to first order.

Solution: Introduce variables:

$$x_1(t) = x(t), \quad x_2(t) = x'(t), \quad y_1(t) = y(t), \quad y_2(t) = y'(t)$$

Hence we have

$$x_2'(t) + y_2(t)x_1(t) + 2t = 0$$
$$y_2'(t) + (y_1(t))^2 x_1(t) = 0$$
$$x_2(t) = x_1'(t)$$
$$y_2(t) = y_1'(t)$$

and hence we have

$$\begin{bmatrix} x_1'(t) \\ x_2'(t) \\ y_1'(t) \\ y_2'(t) \end{bmatrix} = \begin{bmatrix} x_2(t) \\ -y_2(t)x_1(t) - 2t \\ y_2(t) \\ -(y_1(t))^2 x_1(t) \end{bmatrix}$$

which is in the form of

$$\vec{Z}' = f(t, \vec{Z})$$

for $\vec{Z}$ defined as $\begin{bmatrix} x_1(t) & x_2(t) & y_1(t) & y_2(t) \end{bmatrix}^T$. ★

5.9 Stability

Comment 5.9

Since errors are generally $O(h^p)$ for some p, our schemes are less accurate for large h.

Discovery 5.8

But, error/perturbations in initial conditions may lead to vastly different or incorrect answers (as we saw with floating point).

$$y'(t) = f(t, y(t)), \quad y(0) = y_0 + \varepsilon_0$$

Definition 5.14: Unstable

If some initial error ε_0 grows exponentially for many steps ($n \to \infty$), our time-stepping scheme is unstable.

5.9.1 Test Equation

We'll consider a simple linear ODE, our "test equation",

$$y'(t) = -\lambda \cdot y(t), \qquad y(0) = y_0$$

for constant $\lambda > 0$. The exact solution is $y(t) = y_0 \exp(-\lambda t)$, which tends to 0 as $t \to \infty$.

Definition 5.15: Stability

Stability tells us what our numerical algorithm itself does to small errors/ perturbations.

Stability does not imply accuracy, large time steps still usually induce large (truncation) error.

Algorithm 5.7

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

5.9.2 Stability of Forward Euler

Recall that Forward Euler is

$$y_{n+1} = y_n + h \cdot f(t_n, y_n)$$

Hence

$$y_{n+1} = y_n + h \cdot (-\lambda \cdot y_n) = (1 - h\lambda) y_n$$

This has closed form

$$y_n = y_0 (1 - h\lambda)^n$$

We know that the true solution vanishes as t approaches infinity. Our Forward Euler solution vanishes to zero only when

$$|1 - h\lambda| < 1$$

which holds when

$$h < \frac{2}{\lambda} \qquad \text{or} \qquad -h\lambda < 0$$

Result 5.3

Hence Forward Euler method is stable when

$$0 < h < \frac{2}{\lambda}$$

Question 5.8.

If we perturb the initial condition with error ε_0, how does this method behave?

Solution: We have

$$y_n^{(p)} = (y_0 + \varepsilon_0)(1 - h\lambda)^n$$

and hence the error is

$$Error = \varepsilon_n = y_n^{(p)} - y_n = \varepsilon_0(1 - h\lambda)^n$$

which suggests that the error has the same behaviour. ★

We say Forward Euler is **conditionally stable** as it is stable when h satisfies the stability condition we desire.

5.9.3 Stability of Backward/Implicit Euler

Recall that Backward Euler is

$$y_{n+1} = y_n + h \cdot f(t_{n+1}, y_{n+1})$$

Hence

$$y_{n+1} = y_n + h \cdot (-\lambda \cdot y_{n+1}) \quad \Rightarrow \quad y_{n+1} = \frac{1}{1 + h\lambda} y_n$$

This has closed form

$$y_n = y_0 \left(\frac{1}{1 + h\lambda} \right)^n$$

We know that the true solution vanishes as t approaches infinity. Our Forward Euler solution vanishes to zero when

$$1 + h\lambda > 0$$

which holds when

$$h > 0$$

By the same logic, we find the error/ perturbation also follows the same form:

$$\varepsilon_n = \frac{\varepsilon_0}{(1 + h\lambda)^n}$$

Backwards Euler is **unconditionally stable**.

Lecture 11 - Monday, February 10

5.9.4 Stability of Improved Euler

Recall that Improved Euler is

$$y_{n+1} = y_n + \frac{h}{2} \left[f(t_n, y_n) + f(t_{n+1}, y_{n+1}^*) \right]$$

where $y_{n+1}^* = y_n + hf(t_n, y_n)$. Plugging into the equation, we have

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

$$y_{n+1} = y_n + \frac{h}{2} \left[-\lambda y_n + -\lambda y_{n+1}^* \right] \tag{2}$$

Plugging (1) into (2) we have

$$\begin{aligned} y_{n+1} &= y_n + \frac{h}{2}\left[-\lambda y_n + -\lambda(y_n + h(-\lambda y_n))\right] \\ &= y_n\left(1 - h\lambda + \frac{h^2\lambda^2}{2}\right) \end{aligned}$$

Hence the closed form is

$$y_{n+1} = y_0\left(1 - h\lambda + \frac{h^2\lambda^2}{2}\right)^n$$

and error will likewise follow

$$\varepsilon_n = \varepsilon_0\left(1 - h\lambda + \frac{h^2\lambda^2}{2}\right)^n$$

Therefore, it is stable when

$$-1 < 1 - h\lambda + \frac{h^2\lambda^2}{2} < 1$$

For the left hand side, the quadratic has no real roots.

Result 5.4

On the other side, the right hand side yields us

$$h < \frac{2}{\lambda}$$

Discovery 5.9

This is the same stability condition as Forward Euler.

5.10 Stability in General (beyond the test equation)

For our linear test equation $y' = -\lambda y$, forward Euler gave a bound related to λ:

$$\lambda = \left|\frac{\partial}{\partial y}(-\lambda y)\right| = \left|\frac{\partial f}{\partial y}\right|$$

For nonlinear problems, (linear) stability depends on $\frac{\partial f}{\partial y}$ (for dynamics function f) evaluated at a given point in time/space.
For systems of ODEs, stability relates to the eigenvalues of the "Jacobian" matrix,

$$\frac{\partial(f_1, f_2, f_3, \ldots, f_n)}{\partial(y_1, y_2, y_3, \ldots, y_n)}$$

5.11 Truncation Error and Adaptive Time Stepping

Definition 5.16: Truncation Error

Truncation error tells us how the accuracy of our numerical solution scales with time step h.

Question 5.9.

We want to know how to determine LTE in general, for methods *we haven't seen before.* For instance, given the expression for a time-stepping scheme:

$$y_{n+1} = \langle \texttt{ stuff } \rangle$$

derive its dominant truncation error, $O(h^p)$ i.e., find p.

Algorithm 5.8

Given a time-stepping scheme, $y_{n+1} = RHS$

1. Replace approximations on RHS with exact versions.
   e.g, $y_n \to y(t_n)$ and $f(t_{n+1}, y_{n+1}) \to y'(t_{n+1})$, 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 $y(t_{n+1}) - y_{n+1}$. Lowest degree non-canceling power of h gives the local truncation error.

5.11.1 Truncation Error Example — Forward Euler

Example 5.10

We already saw forward Euler:

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

1. Replace y_n with $y(t_n)$, and f with y':

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

2. Nothing to Taylor expand on RHS; everything is already at time t_n.
3. Exact solution Taylor expands to:

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

4. Difference is: LTE $= y(t_{n+1}) - y_{n+1} = \frac{h^2}{2}y''(t_n) + O(h^3) = O(h^2)$

5.11.2 Truncation Error Example — Trapezoidal

Example 5.11

We have Trapezoidal as:

$$y_{n+1} = y_n + \frac{h}{2}\left[f(t_n, y_n) + f(t_{n+1}, y_{n+1})\right]$$

1. Replace y_n with $y(t_n)$, and RHS quantities with exact counterparts:

$$y_{n+1} = y(t_n) + \frac{h}{2}\left[y'(t_n) + y'(t_{n+1})\right]$$

2. Taylor expand RHS quantities about t_n:

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

   Plugging in gives us

$$\begin{aligned} y_{n+1} &= y(t_n) + \frac{h}{2}\left[y'(t_n) + y'(t_n) + hy''(t_n) + \frac{h^2}{2}y'''(t_n) + O(h^3)\right] \\ &= y(t_n) + hy'(t_n) + \frac{h^2}{2}y''(t_n) + \frac{h^3}{4}y'''(t_n) + O(h^4) \end{aligned}$$

3. Exact solution Taylor expands to:

$$y(t_{n+1}) = y(t_n) + hy'(t_n) + \frac{h^2}{2}y''(t_n) + \frac{h^3}{6}y'''(t_n) + O(h^4)$$

4. Difference is: LTE $= y(t_{n+1}) - y_{n+1} = O(h^3)$.

5.11.3 Truncation Error Example — BDF2

Example 5.12

We have BDF2 as:

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

1. Replace y_n with $y(t_n)$, and RHS quantities with exact counterparts:

$$y_{n+1} = \frac{4}{3}y(t_n) - \frac{1}{3}y(t_{n-1}) + \frac{2h}{3}y'(t_{n+1})$$

2. Taylor expand RHS quantities about t_n:

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

   and

$$y(t_{n-1}) = y(t_n) - hy'(t_n) + \frac{h^2}{2}y''(t_n) - \frac{h^3}{6}y'''(t_n) + O(h^4)$$

   Plugging in gives us

$$y_{n+1} = \frac{4}{3}y(t_n) - \frac{1}{3}\left[y(t_n) - hy'(t_n) + \frac{h^2}{2}y''(t_n) - \frac{h^3}{6}y'''(t_n) + O(h^4)\right] + \frac{2h}{3}\left[y'(t_n) + hy''(t_n) + \frac{h^2}{2}y'''(t_n) + O(h^3)\right]$$

Simplifying it we obtain

$$y_{n+1} = y(t_n) + hy'(t_n) + \frac{h^2}{2}y''(t_n) + \frac{7h^3}{18}y'''(t_n) + O(h^4)$$

3. Exact solution Taylor expands to:

$$y(t_{n+1}) = y(t_n) + hy'(t_n) + \frac{h^2}{2}y''(t_n) + \frac{h^3}{6}y'''(t_n) + O(h^4)$$

4. Difference is: LTE $= y(t_{n+1}) - y_{n+1} = O(h^3)$.

5.11.4 Adaptive Time-Stepping

Comment 5.10

Adapt the time step during the computation to keep error small, while minimizing wasted effort.

Discovery 5.10

If we knew the error for a given h, we could choose a good h to always satisfy error $<$ tolerance ... but if we already knew the exact error, we'd also know the solution!

Lecture 12 - Wednesday, February 12

Therefore, we use two different time-stepping schemes together. Compare their results to estimate the error, and adjust h accordingly.

Example 5.13

For instance, if

1. $y_{n+1} = \langle$Method $A\rangle$ with $O(h^4)$.
2. $y_{n+1} = \langle$Method $B\rangle$ with $O(h^5)$.

Then the approximate the error as

$$\text{err} = |y^A_{n+1} - y^B_{n+1}|$$

Adaptive Time-Stepping using Order p and Order $p+1$ methods Suppose for method A

$$y^A_{n+1} = y(t_{n+1}) + O(h^p)$$

We can rewrite it as

$$y^A_{n+1} = y(t_{n+1}) + (h^p + O(h^{p+1}))$$

If B is one order more accurate, say $O(h^{p+1})$, then

$$y^B_{n+1} = y(t_{n+1}) + O(h^{p+1})$$

Methods A's true error is

$$|y^A_{n+1} - y(t_{n+1})| = \boxed{(h^p + O(h^{p+1}))}$$

Our estimated error is

$$|y^A_{n+1} - y^B_{n+1}| = \boxed{(h^p + O(h^{p+1}))}$$

Discovery 5.11

Therefore, the dominant (lowest power) component of the error matches, so our estimate is a decent approximation to use to adjust h.

How to estimate the next time step size We can estimate the error coefficient C as:

$$C \approx \frac{|y^A_{n+1} - y^B_{n+1}|}{(h_{old})^p}$$

where h_{old} is the most recent time step size. If C changes slowly in time, we can estimate the next step error as:

$$err_{next} = |y^A_{n+2} - y(t_{n+1})| \approx C(h_{new})^p$$

where h_{new} is the next step size to be determined. Plug in C to obtain

$$\begin{aligned} err_{next} &\approx \frac{|y^A_{n+1} - y^B_{n+1}|}{(h_{old})^p} \cdot (h_{new})^p \\ &= \left(\frac{h_{new}}{h_{old}}\right)^p |y^A_{n+1} - y^B_{n+1}| \end{aligned}$$

Given a desired error tolerance "tol". Set $err_{next} = tol$ and solve for h_{new}:

$$h_{new} = h_{old}\left(\frac{tol}{|y^A_{n+1} - y^B_{n+1}|}\right)^{1/p}$$

To (roughly) compensate for our approximation, we may be conservative by scaling our tolerance down by some factor α, say 1/2 or 3/4. This will allow step size to grow larger. Again, when it is likely safe to do so.

5.12 Exercises for ODEs

Exercise 5.1

True or false: With an unconditionally stable method, one can take arbitrarily large time steps in numerically solving a stable ODE within a given accuracy.

Exercise 5.2

State the following problem in first order form. Differentiation is respect to t:

$$u''(t) + 3v'(t) + 4u(t) + v(t) = t$$
$$v''(t) - v'(t) + u(t) + v(t) = \cos(t)$$

Exercise 5.3

Apply both the Forward Euler and Modified Euler methods to the IVP

$$\begin{cases} y'(t) & -5y(t) \\ y(0) & = 5 \end{cases}$$

Show the computation schemes for both methods and estimate the step size for each case that ensures stable computations.

Exercise 5.4

Suppose you are using an ODE solver to compute an approximate solution to the equation $y' = f(t, y)$. At some point t_i, you have an approximate solution y_i. Using the solver, you compute estimates y_{i+1}^h and $y_{i+1}^{h/2}$ using steps h and $h/2$, where $h = 0.01$. Note that the second estimate involves applying the ODE solver twice, first to get to $t_{i+\frac{1}{2}}$, and then again to get from $t_{i+\frac{1}{2}}$ to t_{i+1}. The method you are using is a second order method with third order local truncation error. Suppose

$$y_{i+1}^h = 3.269472\ldots, \qquad y_{i+1}^{h/2} = 3.269374\ldots$$

That is,

$$\|y_{i+1}^h - y_{i+1}^{h/2}\| \approx 10^{-4}.$$

Derive an *estimate* for the local truncation error at the point t_{i+1}. Show carefully how you arrived at your estimate.

6 Midterm — Cutoff (and solution)

Comment 6.1

This is the Midterm Cutoff.

Question 1

(a) The smallest (in magnitude) representable positive floating point number is

$$a = 0.100000 \times 10^{-6}$$

(b) With similar reason,

$$b = 0.100001 \times 10^{-6}$$

and hence the spacing between a and b is 10^{-11}.

(c) we have

$$c = 0.999999 \times 10^{6} \qquad \text{and} \qquad d = 0.999998 \times 10^{6}$$

and hence the spacing is 10.

(d) E is simply $\beta^{1-t} = 10^{-4}$.

(e) Real value is 5028.7626, int he FPS, we should have 0.50287×10^4.

(f) The absolute error is given by

$$E_{abs} = |0.50287 \times 10^4 - 5028.7626| = 0.0606$$

and so the relative error is

$$E_{rel} = \frac{E_{abs}}{|5028.7626|} = 1.124438 \times 10^{-5}$$

Question 2

(a) We have

$$\begin{aligned} fl(fl(x) + fl(y)) &= ((1+\delta_1)x + (1+\delta_2)y)(1+\delta_3) \\ &= (1 + \delta_1 + \delta_3 + \delta_1\delta_3)x + (1 + \delta_2 + \delta_3 + \delta_2\delta_3)y \end{aligned}$$

and hence we can calculate the relative error.

Question 3

(a) We have

$$L_2 = \frac{(x-(-1))(x-3)(x-5)}{(1-(-1))(1-3)(1-5)}$$

(b) We have

$$p(x) = \sum_{i=1}^{4} f(x_i)L_i(x) = f(-1)L_1(x) + f(1)L_2(x) + f(3)L_3(x) + f(5)L_4(x)$$

(c) We notice that

$$\deg f = 2 \qquad \text{and} \qquad \deg(p) \leq 3$$

and f and p interpolate at the same four data points, and hence by the Unisolvence Theorem, we have $f = p$, which implies that

$$p(x) = 2x^2 + 3x - 9$$

Question 4

Denote the first polynomial as $S_0(x)$ and the second as $S_1(x)$, solving $S_0(2) = S_1(2)$ we have

$$a = 4$$

Solving for $S_0'(x) = S_1'(x)$ we have

$$A = -12$$

and hence $S_0'(0) = 4$ and $S_1'(3) = 1$.

Question 5

(a) We have $y_1 = 181$

(b) We have

$$\begin{aligned} y_1 &= y_0 + hf(t_1, y_1) \\ &= 2 + 0.2 \cdot \left(\frac{900(1.2)}{(y_1 - 1)^3} - 5 \right) \\ &= 1 + \frac{216}{(y_1 - 1)^3} \end{aligned}$$

Solving for y_1 we have $y_1 \approx 4.83$

Question 6

We would eventually have

$$y_{n+1} \left(1 + \frac{h\lambda}{2} \right) = y_n \left(1 - \frac{h\lambda}{2} \right)$$

Solving for

$$-1 < \left(1 - \frac{h\lambda}{2} \right) / \left(1 + \frac{h\lambda}{2} \right) < 1$$

we have $h > 2/\lambda$.

Question 7

We have

$$\begin{aligned}
y_{n+1} &= y_n + hf\left(t_n + \frac{h}{2}, y(t_n) + \frac{h}{2} f\left(t_n, y(t_n)\right)\right) \\
&= y(t_n) + h\left[f(t_n, y(t_n)) + \frac{h}{2}\frac{\partial f(t_n, y(t_n))}{\partial t} + \frac{h}{2}\frac{f(t_n, y(t_n))}{\partial y} + O(h^2)\right] \\
&= y(t_n) + hf(t_n, y(t_n)) + \frac{h^2}{2}\left(\frac{f(t_n, y(t_n))}{\partial t} + f(t_n, y(t_n))\frac{\partial f(t_n, y(t_n))}{\partial y}\right) + O(h^3) \\
&= y(t_n) + hf(t_n, y(t_n)) + \frac{h^2}{2}\frac{\partial f(t_n, y(t_n))}{dt} + O(h^3) \\
&= y(t_n) + hy'(t_n) + \frac{h^2}{2}y''(t_n) + O(h^3)
\end{aligned}$$

as desired.

7 Fourier Analysis

Common examples of the applications of Fourier Analysis include:

- "signal processing" in general (filtering, denoising, etc.)
- sound/audio manipulation.
- image/video processing (e.g. JPEG-style compression).
- analyzing sensor data.
- electromagnetic signals.

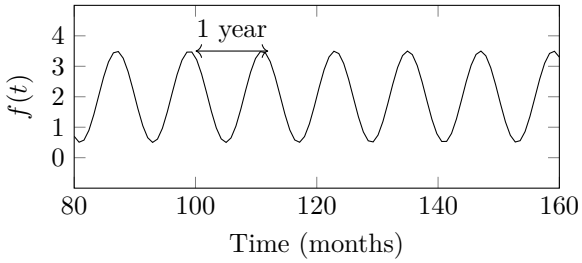
Definition 7.1: Fourier Analysis

Fourier Analysis takes in data/ functions, and outputs combinations of functions with different frequencies. This allows us to process and analyze the data which could be processed in Inverse Fourier Transform, yields us the data/ function in the domain.

Example 7.1: Orange Juice Pricing

Consider the price of orange juice over many months.

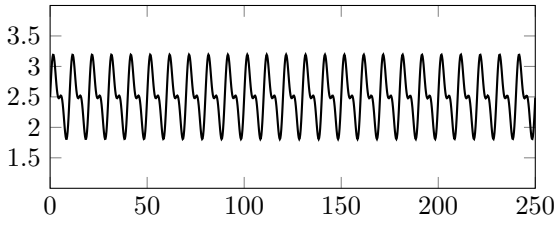
Typically cyclical or periodic (i.e., pattern repeats over time) due to e.g. seasonal variation in supply and demand. (We might approximate it with a sine curve.)



Example 7.2

Other recurring phenomena can have effects on different time scales: weather fluctuations, variation in import costs, El Nino, etc.

Data might actually look more like this:



Question 7.1.

How can we also represent these additional fluctuations in data?

Solution: Add more sinusoidal terms with different periods/ frequencies.x ★

Result 7.1

General form could be:

$$f(t) = a_0 + a_1 \cos(qt) + b_1 \sin(qt) + a_2 \cos(2qt) + b_2 \sin(2qt) + \cdots$$

7.1 Continuous Fourier Series

Definition 7.2: Periodic

A function f is said to be **periodic** if there exists T such that

$$f(t \pm T) = f(t)$$

i.e. f repeats after one length/ period T.

Comment 7.1

The goal is to represent any $f(t)$ as an infinite sum of trigonometric functions:

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

where a_k and b_k indicate the "information" or amplitude for each sinusoid of a specific period $\frac{T}{k}$, or frequency $\frac{k}{T}$.

Discovery 7.1

Higher integer k indicates shorter period & higher wave frequency.

Example 7.3

https://upload.wikimedia.org/wikipedia/commons/5/50/Fourier_transform_time_and_frequency_domains.gif.

7.1.1 Handy Identities

Code 7.1

We have

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

for any integers k and j. i.e. the integral of the product of $\cos(kt)$ and $\sin(jt)$ over $[0, 2\pi]$ is 0.

Definition 7.3: Orthogonal

We say that these two functions are **orthogonal** to each other.

Code 7.2

Here are more:

$$\int_0^{2\pi} \cos(kt)\cos(jt)\,dt = 0 \qquad \text{for } k \neq j,$$
$$\int_0^{2\pi} \sin(kt)\sin(jt)\,dt = 0 \qquad \text{for } k \neq j,$$
$$\int_0^{2\pi} \sin(kt)\,dt = 0,$$
$$\int_0^{2\pi} \cos(kt)\,dt = 0.$$

Comment 7.2

We can use these to extract a single Fourier coefficient at a time!

We want

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

Integrate over $[0, 2\pi]$ to use some of our orthogonality identites:

$$\int_0^{2\pi} f(t)\,dt = a_0 \int_0^{2\pi} dt + \sum_{k=1}^{\infty} a_k \cancel{\int_0^{2\pi} \cos(kt)\,dt} + \sum_{k=1}^{\infty} b_k \cancel{\int_0^{2\pi} \sin(kt)\,dt}$$

Therefore, we have

$$a_0 = \frac{\int_0^{2\pi} f(t)\,dt}{\int_0^{2\pi} dt} = \frac{1}{2\pi}\int_0^{2\pi} f(t)\,dt$$

which implies that a_0 is the average value of f over $[0, 2\pi]$. Now we wish to determine coefficients a_ℓ for $\ell > 0$. Again we starts with

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

Multiply by $\cos(\ell t)$ and integrate over $[0, 2\pi]$:

$$\int_0^{2\pi} f(t)\cos(\ell t)\,dt = \cancel{\int_0^{2\pi} a_0\cos(\ell t)\,dt} + \sum_{k=1}^{\infty} a_k \int_0^{2\pi} \cos(kt)\cos(\ell t)\,dt + \sum_{k=1}^{\infty} b_k \cancel{\int_0^{2\pi} \sin(kt)\cos(\ell t)\,dt}$$

Therefore, the only term left is when $k = \ell$ in the middle summation term. Hence

$$\begin{aligned} a_\ell &= \frac{\int_0^{2\pi} f(t)\cos(\ell t)\,dt}{\int_0^{2\pi} \cos^2(\ell t)\,dt} \\ &= \frac{1}{\pi}\int_0^{2\pi} f(t)\cos(\ell t)\,dt \end{aligned}$$

Comment 7.3

Similarly, we can also find that

$$b_\ell = \frac{1}{\pi}\int_0^{2\pi} f(t)\sin(\ell t)\,dt$$

Lecture 13 - Monday, February 24

7.2 Fourier Transforms – The Discrete Setting

7.2.1 Euler's Formula

Theorem 7.1

Euler's Formula tells us that

$$e^{i\theta} = \cos(\theta) + i\sin(\theta)$$

Discovery 7.2

As a result, we have

$$e^{-i\theta} = \cos(-\theta) + i\sin(-\theta) = \cos(\theta) - i\sin(\theta)$$

and thus

$$\cos(\theta) = \frac{e^{i\theta} + e^{-i\theta}}{2} \qquad \sin(\theta) = \frac{e^{i\theta} - e^{-i\theta}}{2}$$

Result 7.2

Now, given our earlier sinusoidal expression of a function $f(t)$

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

we can express it more concisely as

$$f(t) = \sum_{k=-\infty}^{\infty} c_k e^{ikt} \tag{3}$$

where the c_k coefficients are complex numbers.

7.2.2 Converting between c_k, and a_k and b_k

The coefficients are given by:

$$c_0 = a_0, \quad c_k = \frac{a_k}{2} - i\frac{b_k}{2}, \quad \text{and} \quad c_{-k} = \frac{a_k}{2} + i\frac{b_k}{2} \quad \text{for } k > 0$$

And we have the relationships:

$$|a_0| = |c_0|, \quad \text{and} \quad |c_k| = |c_{-k}| = \frac{1}{2}\sqrt{a^2 + b^2}$$

7.2.3 Finding c_ℓ coefficients

We can also obtain a formula for c_ℓ directly by noting that

$$\int_0^{2\pi} e^{ikt} e^{-i\ell t}\, dt = \begin{cases} 0 & \text{if } k \neq \ell, \\ 2\pi & \text{if } k = \ell. \end{cases}$$

If we multiply both sides of equation (3) by $e^{-i\ell t}$ and integrate term by term we obtain the formula

$$c_\ell = \frac{1}{2\pi} \int_0^{2\pi} e^{-i\ell t} f(t)\, dt$$

For typical functions $f(t)$, there is a small amount of information in the high-frequency harmonics. Therefore, we can approximate any input signal $f(t)$ by

$$f(t) \simeq \sum_{k=-M}^{M} c_k e^{ikt}$$

7.3 Discrete Fourier Transform

Usually, we have an input signal sampled with N samples over a period T, with samples being $\Delta t = T/N$ apart. Let us denoted the input signal by $f(n\Delta t) = f_n$, and so the input sampled data is given by the set $f_0, f_1, \ldots, f_{N-1}$. Note that $f(0) = f(T)$, so that given $f(0)$, $f(T)$ is redundant. Let the discrete sample time be

$$t_n = n\Delta t = \frac{nT}{N}, \qquad \text{for } n = 0, 1, \ldots, N-1$$

We can think of our problem as a problem in interpolation. We are given a finite set of points $(t_0, f_0), \ldots, (t_{N-1}, f_{N-1})$ and wish to find constants a_0, a_k, b_k such that

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

(for some M) interpolates these values.

Comment 7.4

As was the case in the previous section, it is convenient to use exponential rather than trigonometric forms for our expression. Since we have N sample points, we can fit this data exactly if we use N degrees of freedom. Therefore we approximate the input signal $f(t)$ by

$$f(t) \simeq \sum_{k=-N/2+1}^{N/2} c_k e^{\frac{i2\pi kt}{T}}$$

where we will assume from now on that N is even.

For the above formula to be useful we need to have a formula for the c_k in terms of the sampled values $f(t_n) = f(n\Delta t) = f_n$. Then

$$f(n\Delta t) = f_n = \sum_{k=-N/2+1}^{N/2} c_k e^{\frac{i2\pi nk}{N}}$$

For computational purposes (see section on FFT) we wish to write this formula in a more convenient fashion. Recall that we have assumed that $f(t+N) = f(t)$. Thus,

$$f_n = \sum_{k=-N/2+1}^{N/2} c_k e^{\frac{i2\pi nk}{N}} \tag{4}$$

$$= \sum_{k=0}^{N/2} c_k e^{\frac{i2\pi nk}{N}} + \sum_{k=-N/2+1}^{-1} c_k e^{\frac{i2\pi nk}{N}} \tag{5}$$

Letting $j = N + k$ in the second term in the above equation, we get

$$\sum_{k=-N/2+1}^{-1} c_k e^{\frac{i2\pi nk}{N}} = \sum_{j=N/2+1}^{N-1} c_{j-N} e^{\frac{i2\pi n(j-N)}{N}} \tag{6}$$

$$= \sum_{j=N/2+1}^{N-1} c_{j-N} e^{\frac{i2\pi nj}{N}} \tag{7}$$

For convenience, let us define for $j > N/2$

$$c_j = c_{j-N} \qquad \text{for } j = N/2+1, \ldots, N-1$$

that is, we have extended the definition of c_j periodically. By this we mean that $c_{j\pm N} = c_j$.

Code 7.3

Now equation (7) becomes

$$\sum_{k=-N/2+1}^{-1} c_k e^{\frac{i2\pi nk}{N}} = \sum_{k=N/2+1}^{N-1} c_k e^{\frac{i2\pi nk}{N}}$$

Result 7.3

Now equation (5) yields

$$f_n = \sum_{k=0}^{N-1} c_k e^{\frac{i2\pi nk}{N}}$$

Or, letting $F_k = c_k$, we can write this as

$$f_n = \sum_{k=0}^{N-1} F_k e^{\frac{i2\pi nk}{N}}$$

Definition 7.4: Roots of Unity

For notational convenience we defined:

$$W = e^{\left(\frac{2\pi i}{N}\right)}$$

W is an N^{th} **Root of Unity**, since it satisfies

$$W^N = e^{2\pi i} = 1$$

7.3.1 $f_n = \sum_{k=0}^{N-1} F_k W^{nk}$

Hence we have

$$f_n = \sum_{k=0}^{N-1} F_k e^{\frac{i2\pi nk}{N}} = \sum_{k=0}^{N-1} F_k W^{nk} \tag{fn}$$

Comment 7.5

Discrete data f_n is expressed as a sum of coefficients, F_k, times complex exponentials, W^{nk}.

Algorithm 7.1

To be useful, operations we need are:

1. Convert input time-domain data f_n to frequency-domain F_k.
2. Convert frequency-domain data F_k back to time-domain f_n.

Question 7.2.

We have derived #2, but how can we solve for F_k given f_n?

Solution: We use orthogonality ideas, as in the continuous case for a_k, b_k or c_k ... ★

7.3.2 Roots of Unity — Orthogonal Identity

Discovery 7.3

Properties of the N^{th} roots of unity:

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

Assuming (for now) that $k, \ell \in [0, N-1]$, where k, ℓ are both integers.

Proof. We have two cases to consider:

1. $k = \ell$: we simple have
$$\sum_{j=0}^{N-1} 1 = N$$
2. $k \neq \ell$: applying geometric series, we have
$$\sum_{j=0}^{N-1} \left(W^{k-\ell}\right)^j = \frac{\left(W^{k-\ell}\right)^N - 1}{W^{k-\ell} - 1} = 0$$

thus we have completed our proof. □

To find F_k, we'll need another useful property of our N^{th} roots of unity. Recall that we have

Lecture 13 - Wednesday, February 26

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

To obtain the k^{th} coefficient, we multiply the above equation by W^{-nk} and sum from 0 to $N-1$:

$$\sum_{n=0}^{N-1} f_n W^{-nk} = \sum_{n=0}^{N-1} \sum_{k=0}^{N-1} F_k W^{nk} W^{-nk} = \sum_{k=0}^{N-1} F_k \sum_{n=0}^{N-1} W^{n(j-k)} = \sum_{j=0}^{N-1} F_j N \delta_{j,k} = F_k \cdot N$$

7.3.3 $F_k = \frac{1}{N} \sum_{n=0}^{N-1} f_n W^{-nk}$

Therefore, we obtain

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

Definition 7.5: Discrete Fourier transform

The above two resultint formulae are our **Discrete Fourier Transform** pair.

Result 7.4

The discrete Fourier transform is *invertible*.

Example 7.4

Given four discrete Fourier coefficients F_k, find the corresponding four data points f_n. Let $F = \begin{bmatrix} -2 \\ 2+i \\ -2 \\ 2-i \end{bmatrix}$, what is the vector f?

Solution: We have $N = 4$, so

$$W = e^{\frac{2\pi i}{N}} = e^{\frac{2\pi i}{4}} = \cos\left(\frac{\pi}{2}\right) + i\sin\left(\frac{\pi}{2}\right) = i$$

and from the formula

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

we have

$$f = \begin{bmatrix} 0 \\ -2 \\ -8 \\ 2 \end{bmatrix}$$

as desired. ★

Example 7.5

Consider a set of N data points defined such that $f_n = \cos\left(\frac{2\pi n}{N}\right)$. Show that $F_1 = F_{N-1} = 1/2$; for all other coefficients, $F_k = 0$. Therefore we can express f_n in our Fourier representation as

$$f_n = \cos\left(\frac{2\pi n}{N}\right) = \sum_{k=0}^{N-1} F_k W^{nk} = \frac{1}{2}W^{n(1)} + \frac{1}{2}W^{n(N-1)}$$

Solution: Using Euler Formula, we have

$$\cos\theta = \frac{e^{i\theta} + e^{-i\theta}}{2}$$

Hence we have

$$\begin{aligned} F_k &= \frac{1}{N}\sum_{n=0}^{N-1} \frac{1}{2}\left(e^{i2\pi n/N} + e^{-i2\pi n/N}\right) W^{-nk} \\ &= \frac{1}{N}\sum_{n=0}^{N-1} \frac{1}{2}\left(W^{n} + W^{-n}\right) W^{-nk} \\ &= \frac{1}{2N}\sum_{n=0}^{N-1} \left(W^{n(1-k)} + W^{-n(1+k)}\right) \end{aligned}$$

Multiply the second term by $W^{Nn} = 1$ to adjust the exponent, we have

$$F_k = \frac{1}{2N} \sum_{n=0}^{N-1} \left(W^{n(1-k)} + W^{n(N-1-k)} \right)$$

Using orthogonal identity,

$$F_k = \frac{\delta_{1,k}}{2} + \frac{\delta_{N-1,k}}{2} = \begin{cases} \frac{1}{2} & \text{for } k = 1, N-1 \\ 0 & \text{otherwise} \end{cases}$$

as desired. ★

7.3.4 Two Properties of the DFT

Result 7.5

As a consequence of N^{th} roots of unity,

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

Comment 7.6

Hence the $|F_k|$ are symmetric about $k = N/2$.

Example 7.6

For $N = 8$:

$$F_0, \boxed{F_1, F_2, F_3}, \boxed{F_4}, \boxed{F_5, F_6, F_7}$$

with conjugate symmetry,

$$F_1 = \overline{F_7}, \quad F_2 = \overline{F_6}, \quad F_3 = \overline{F_5}$$

Proof. Proof for property one:
Given F_k for integer $k \in [0, N-1]$, then for arbitrary integer $\ell \in (-\infty, +\infty)$, F_ℓ is one of the existing values. We can express arbitrary ℓ as

$$\ell = mN + p$$

for $p \in [0, N-1]$. Then

$$W^{-k} = e^{-i\frac{2\pi}{N}(mN+p)} = \underbrace{e^{-i2\pi m}}_{=1} \cdot e^{\frac{-i2\pi p}{N}} = W^{-p}$$

and hence

$$\begin{aligned} F_k &= \frac{1}{N} \sum_{n=0}^{N-1} f_n W^{-nk} \\ &= \frac{1}{N} \sum_{n=0}^{N-1} f_n W^{-np} = F_p \end{aligned}$$

for $p \in [0, N-1]$. □

Proof. Proof for property two:
We will be using three facts:

- $W^{N-k} = W^{-k}$;
- $\overline{W^j} = W^{-j}$;
- For real number x, we have $\overline{x} = x$.

Hence we have

$$\begin{aligned}
\overline{F_{N-k}} &= \overline{\frac{1}{N}\sum_{n=0}^{N-1} f_n W^{-n(N-k)}} \\
&= \frac{1}{N}\sum_{n=0}^{N-1} f_n \overline{W^{-n(N-k)}} \\
&= \frac{1}{N}\sum_{n=0}^{N-1} f_n \overline{W^{-n(-k)}} \\
&= \frac{1}{N}\sum_{n=0}^{N-1} f_n W^{-nk} = F_k
\end{aligned}$$

as desired. □

Comment 7.7

Typically, we want to learn/achieve something by processing the data (Image data, audio samples, prices, intensities, etc.).

In theory, the time-domain data tells us everything!

In practice, Fourier coecients provide easier access to useful insights/information for certain problems.

Definition 7.6: Direct Current

We call F_0 **direct current** (DC).

Discovery 7.4

Coecient F_0 is always the average of data values.

7.4 Inverse Discrete Fourier Transform

The result (fn) can be established by looking at a matrix algebra view of equation (Fk) and using the orthogonality property. Let f be the column vector of the data samples, and let F be the corresponding column vector of Fourier coefficients. Then (Fk) can be written as

$$F = Mf$$

where M is an $N \times N$ matrix with j^{th} column $= 1/N \cdot \overline{W(j)}$. Now, the orthogonality property states that

$$\overline{M}^T M = \frac{1}{N} I$$

where I is the $N \times N$ identity matrix. In other words,

$$M^{-1} = N\overline{M}^T$$

and hence

$$f = N\overline{M}^T F$$

If we write this out componentwise, we find that it is (fn).

7.4.1 Lack of Standardization

Discovery 7.5

Various definitions of DFT/IDFT pairs can be found in the literature/code.

We use the following, with 0-based indexing:

$$f_n = \sum_{k=0}^{N-1} F_k W^{nk} \quad \text{and} \quad F_k = \frac{1}{N} \sum_{n=0}^{N-1} f_n W^{-nk}$$

Code 7.4

SciPy (and some other sources) use:

$$f_n = \frac{1}{N} \sum_{k=0}^{N-1} F_k W^{nk} \quad \text{and} \quad F_k = \sum_{n=0}^{N-1} f_n W^{-nk}$$

Comment 7.8

So be careful (a) when coding in Jupyter, and (b) reading other sources!

Lecture 15 - Wednesday, March 05

Comment 7.9

Lecture 14 is skipped due to midterm. This lecture is shorter because we present midterm solution in class.

7.5 Fast Fourier Transform

We want to explore a more efficient method.

Question 7.3.

1. What is the complexity of the naive method?
2. What property of the DFT allows it to be sped up?
3. How can we construct an algorithm from this property?
4. What is the complexity of the new method?

7.5.1 Slow Fourier Transform

A direct implementation of $F_k = \frac{1}{N}\sum_{n=0}^{N-1} f_n W^{-nk}$ takes $O(N^2)$ complex floating-point operations.

Code 7.5

Essentially two nested `for` loops:

```
For k = 0 : N - 1              // iterate over all k unknown coeffs
    F_k = 0                    // initialize coefficient to zero
    For n = 0 : N - 1          // iterate over all n data values
        F_k += f_n W^(-nk)     // increment by scaled data value
    End
    F_k = F_k / N              // normalize
End
```

7.5.2 Faster Fourier Transform

Theorem 7.2

We use *divide and conquer*.

Algorithm 7.2

1. Split the full DFT into two DFT's of half the length.
2. Repeat recursively.
3. Finish at the base case of individual numbers.

Comment 7.10

(If $N \neq 2^m$ for some m, we can pad our initial data with zeros.)

Question 7.4.

Key question: How can we split up the DFT?

Theorem 7.3

The usual DFT of the sequence f_n is:

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

We'll show we can express it with DFTs of two new arrays of **half the length** $(N/2)$:

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

where $n \in \left[0, \frac{N}{2} - 1\right]$. Then

$$F_{\text{even}} = G = DFT(g) \quad \text{and} \quad F_{\text{odd}} = H = DFT(h)$$

Solution: We have

$$F_k = \frac{1}{N}\sum_{n=0}^{N-1} f_n W^{-nk} = \frac{1}{N}\sum_{n=0}^{N/2-1} f_n W^{-nk} + \frac{1}{N}\sum_{n=N/2}^{N-1} f_n W^{-nk}$$

Assume N is even, we reindex second um using $m = n - N/2$,

$$F_k = \frac{1}{N}\sum_{n=0}^{N/2-1} f_n W^{-nk} + \frac{1}{N}\sum_{m=0}^{N/2-1} f_{m+N/2} W^{-(m+N/2)k}$$

Replace m back to n we have

$$F_k = \frac{1}{N}\sum_{n=0}^{N/2-1} f_n W^{-nk} + \frac{1}{N}\sum_{n=0}^{N/2-1} f_{n+N/2} W^{-nk} W^{-kN/2}$$

Combine sums, we have

$$F_k = \frac{1}{N}\sum_{n=0}^{N/2-1} \left(f_n + f_{n+N/2}\cdot W^{-kN/2}\right) W^{-nk}$$

Discovery 7.6

Notice that

$$W^{-kN/2} = \left(e^{2\pi i/N}\right)^{-kN/2} = e^{-\pi ik} = (-1)^k$$

Hence we have two cases to consider:

1. *k is even*: in this case we have

$$F_k = F_{2j} = \frac{1}{N}\sum_{n=0}^{N/2-1} \left(f_n + f_{n+N/2}\right) W^{-2nj}$$

for $j = 0$ to $N/2 - 1$.

2. *k is odd*: in this case we have

$$F_k = F_{2j+1} = \frac{1}{N}\sum_{n=0}^{N/2-1}\left(f_n - f_{n+N/2}\right)W^{-n(2j+1)}$$
$$= \frac{1}{N}\sum_{n=0}^{N/2-1}\left[(f_n - f_{n+N/2})W^{-n}\right]W^{-2nj}$$

for $j = 0$ to $N/2-1$.

Definition 7.7: Half-Length Vector

Now, it is fair for us to define the two new **half-length vectors** by

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

for $n = 0$ to $N/2-1$, and let $M = N/2$, then we observe:

$$F_{2k} = \frac{2}{N}\sum_{n=0}^{N/2-1} g_n W^{-2nk}$$
$$F_{2k+1} = \frac{2}{N}\sum_{n=0}^{N/2-1} h_n W^{-2nk},$$

for $k = 0, \ldots, N/2-1$. Also,

Discovery 7.7

Observe that

$$W^{-2kn} = e^{-\frac{2\pi i}{N}2kn} = \left(e^{-\frac{2\pi i}{N/2}}\right)^{kn}$$

so that W^2 is the W *factor* for an input signal of length $N/2$ (that is, it is the $(N/2)$-th complex root of unity). Thus, athe above equations can be regarded as

$$F_{\text{even}} = G = DFT(g) \quad \text{and} \quad F_{\text{odd}} = H = DFT(h), \quad \text{where}$$
$$g = \frac{1}{2}\left(f_n + f_{n+\frac{N}{2}}\right) \quad \text{and} \quad h = \frac{1}{2}\left(f_n - f_{n+\frac{N}{2}}\right)W^{-n}$$

★

Result 7.6

Therefore, we have converted the problem of computing a single DFT of N points into the computation of two DFT's of $N/2$ points. We can then reduce each of the $N/2$ length DFT's into two $N/4$ length

DFT's, and so on. There will be $\log_2 N$ of these stages. Each stage requires $O(N)$ complex floating point operations, so the complexity of the FFT is $O(N \log_2 N)$.

Lecture 16 - Monday, March 10

Example 7.7

As an example, we have

$$f = \begin{bmatrix} 2 \\ 2 \\ 1 \\ 3 \\ 2 \\ 2 \\ 1 \\ 3 \end{bmatrix} \quad \Rightarrow \quad g = \begin{bmatrix} 2 \\ 2 \\ 1 \\ 3 \end{bmatrix} \text{ and } h = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

DFT of g is $G = \begin{bmatrix} 2 & \frac{1+i}{4} & -\frac{1}{2} & \frac{1-i}{4} \end{bmatrix}$ and DFT of h is $H = \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix}$. Hence, recovering DFT of f by interleaving G and H we obtain:

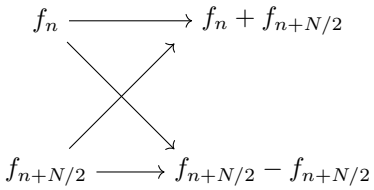
$$F = \begin{bmatrix} 2 & 0 & \frac{1+i}{4} & 0 & -\frac{1}{2} & 0 & \frac{1-i}{4} & 0 \end{bmatrix}$$

7.5.3 FFT Algorithm — Butterfly

Code 7.6

```
1  N = 2^m, W = e^{i2π/N}
2  for k = 1, ..., m do
3      for j = 1, ..., 2^{k-1} do
4          l := (j − 1) * N
5          for n = 0, ..., N/2 − 1 do
6              wfactor := (W^{−n})^{2^{k-1}}
7              temp := wfactor · (f_{n+l} − f_{n+l+N/2})
8              f_{n+l} := 1/2 · (f_{n+l} + f_{n+l+N/2})
9              f_{n+l+N/2} := 1/2 · temp
10         end
11     end
12     N := N/2
13 end
```

Each step of the FFT computes values using f_n and $f_{n+N/2}$ which can be visualized as following:



Comment 7.11

This diagram resembles the wings of a butterfly, hence the name of the algorithm.

Question 7.5.

Given the input $f_0, \ldots, f_{N-1}$, we want to compute the DFT $F_0, \ldots, F_{N-1}$. The above butterfly algorithm overwrites the original array $f_0, \ldots, f_{N-1}$ with $f'_0, \ldots, f'_{N-1}$. Where do we find $F_0, \ldots, F_{N-1}$?

Solution: In general, we can determine what value of $F_0, \ldots, F_{N-1}$ is in $X_0, \ldots, X_{N-1}$, by simply expressing the output array X index as an $\log_2 N$ digit binary number, reversing the bit sequence, and then converting back to decimal. ★

7.5.4 A Complete Butterfly example

Example 7.8

Consider the input data,

$$f = (1, 2, 3, 4, 1, 2, 3, 4) \qquad (N = 8)$$

We can obtain the half length sequences:

$$g = (4, 3, 2, 1) \qquad \text{and} \qquad h = (0, 0, 0, 0)$$

Now, we recurse on these two sequences until we reach the base case (a single element), each time overwriting the original array, f.

$$\mathbf{f} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} \rightarrow \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} \rightarrow \begin{bmatrix} 2 \\ 3 \end{bmatrix} \rightarrow [\,2.5\,] \\ [-0.5\,] \\ \begin{bmatrix} -1 \\ i \end{bmatrix} \rightarrow [-0.5 + 0.5i] \\ [-0.5 - 0.5i] \\ \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \rightarrow \begin{bmatrix} 0 \\ 0 \end{bmatrix} \rightarrow [\,0\,] \\ [\,0\,] \\ \begin{bmatrix} 0 \\ 0 \end{bmatrix} \rightarrow [\,0\,] \\ [\,0\,] \Longrightarrow \begin{bmatrix} 2.5 \\ 0 \\ -0.5 + 0.5i \\ 0 \\ -0.5 \\ 0 \\ -0.5 - 0.5i \\ 0 \end{bmatrix} = \mathbf{F}$$

Example 7.9: A smaller, complex FFT example

Find the sequence of vectors and the final result of applying the butterfly FFT approach to the data: $f = (2 + i, 3, -2, -2i)$.

Solution: The solution is $F = \left(\frac{3-i}{4}, \frac{3-i}{2}, \frac{-3+3i}{4}, \frac{1}{2}+i\right)$. ★

7.6 Image/ Data Compression

Lecture 17 - Wednesday, March 12

7.6.1 Compression of 1D Images

Algorithm 7.3: Compression Strategy:

1. Create an (approximate) compressed version of the image, f_n, by throwing away "small" Fourier coecients: $|F_k| < tol$;
2. To reconstruct the image, run the *inverse DFT* to get modified data (pixels), $\hat{f}_n$;
3. Discard the imaginary parts of $\hat{f}_n$, to ensure new data is strictly real.

7.6.2 Image Processing in 2D

Question 7.6.

How does the DFT work for 2D (grayscale) image data? i.e., we have a 2D array X of per-pixel intensities, of size $M \times N$. Assume we have scaled image data, so $0 \leq X(i,j) \leq 1$.

Solution: We apply a 2D extension of the DFT definition to the data, like so:

$$F_{k,\ell} = \frac{1}{NM} \sum_{n=0}^{N-1} \sum_{j=0}^{M-1} f_{n,j} W_N^{-nk} W_M^{-j\ell}$$

Essentially two standard (1D) DFT's combined. Note that two (possibly distinct) roots of unity are used, one per dimension:

$$W_N = e^{\frac{2\pi i}{N}} \quad \text{and} \quad W_M = e^{\frac{2\pi i}{M}}$$

Result is a 2D array of Fourier coefficients, $F_{k,\ell}$. ★

Discovery 7.8

The 2D Fast FT can be computed eciently using nested 1D FFTs:

1. First, transform each row (separately) using 1D FFTs.
2. Then, transform each column of the result, again using 1D FFTs.

Theorem 7.4

The following is the derivation of 2D FFT via 1D FFT.

Proof. We have

$$\begin{aligned} F_{k,\ell} &= \frac{1}{NM}\sum_{n=0}^{N-1}\sum_{j=0}^{M-1} f_{n,j} W_N^{-nk} W_M^{-j\ell} \\ &= \frac{1}{N}\sum_{n=0}^{N-1} W_N^{-nk} \underbrace{\left(\frac{1}{M}\sum_{j=0}^{M-1} f_{n,j} W_M^{-j\ell}\right)}_{\texttt{1D FFT per row}} \end{aligned}$$

Hence it is convenient to define

$$H_{n,\ell} = \frac{1}{M}\sum_{j=0}^{M-1} f_{n,j} W_M^{-j\ell}$$

and consequently

$$F_{k,\ell} = \frac{1}{N}\sum_{n=0}^{N-1} H_{n,\ell} W_N^{-nk}$$

which is another Fourier transform. □

Result 7.7: Complexity

Performing M 1D FFT's of length N: $M \cdot O(N \log_2 N) = O(MN \log_2 N)$.
Performing N 1D FFT's of length M: $N \cdot O(M \log_2 M) = O(MN \log_2 M)$.
So total complexity is

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

7.7 Aliasing

Definition 7.8: Aliasing

If a real input signal contains high frequencies, but the spacing of our discretely sampled data points is inadequate, "**aliasing**" can occur.

Example 7.10

Let's explore this mathematically to better understand what's happening in terms of Fourier analysis.

Solution: Recall our Fourier series of a continuous signal was

$$f(t) = \sum_{k=-\infty}^{+\infty} c_k \exp\left(\frac{i2\pi kt}{T}\right)$$

for a period T,. If we sample this true signal at points $t_n = n\Delta t = nT/N$, then

$$f_n = f(t_n) = \sum_{k=-\infty}^{+\infty} c_k \exp\left(\frac{i2\pi kn}{N}\right) = \sum_{k=-\infty}^{+\infty} c_k W^{nk}$$

This is exact for arbitrarily high frequencies. A plot of $|c_k|^2$ would give the *exact* power spectrum. Now, the coefficients of the DFT, F_k are defined by

$$f_n = \sum_{k=-N/2+1}^{N/2} F_k \exp\left(\frac{i2\pi kn}{N}\right) \tag{8}$$

Note that the index k runs from $[-N/2+1, \ldots, +N/2]$ so that it corresponds with equation above. A plot of $|F_k|^2$ would show the computed *approximate* power spectrum.

Question 7.7.

How closely does F_k correspond to c_k, for $k \in [-N/2+1, +N/2]$?

We can determine the precise relationship between F_k and c_k as follows. Recall the *orthogonality relation*:

$$\sum_{p=0}^{N-1} \exp\left(\frac{i2\pi p(\ell-k)}{N}\right) = N\delta_{k,\ell}$$

which can be written (by shifting the index) as

$$\sum_{p=-N/2+1}^{N/2} \exp\left(\frac{i2\pi p(\ell-k)}{N}\right) = N\delta_{k,\ell} \tag{9}$$

It therefore follows from equations (8) and (9) that

$$F_l = \frac{1}{N} \sum_{n=-N/2+1}^{N/2} f_n e^{-i\frac{2\pi nl}{N}}$$

Substituting gives

$$\begin{aligned} F_l &= \frac{1}{N} \sum_{n=-N/2+1}^{N/2} e^{-i\frac{2\pi nl}{N}} \sum_{k=-\infty}^{\infty} c_k e^{i\frac{2\pi kn}{N}} \\ &= \sum_{k=-\infty}^{\infty} c_k \frac{1}{N} \sum_{n=-N/2+1}^{N/2} e^{i\frac{2\pi n(k-l)}{N}} \end{aligned}$$

Since $k = -\infty, .., \infty$ in the above equation, we rewrite equatino (8):

$$\sum_{p=-N/2+1}^{N/2} e^{i\frac{2\pi p(l-k)}{N}} = N(\delta_{k,l} + \delta_{k,l+N} + \delta_{k,l-N} + \delta_{k,l+2N} + \ldots)$$

Consequently, we have

$$F_l \quad = \quad c_l + c_{l+N} + c_{l-N} + c_{l+2N} + c_{l-2N} + \ldots$$

★

Result 7.8

The above equation has the following interpretation: if the original signal contains frequencies with complex frequency $\frac{|p|}{T} > \frac{N}{2T}$ ($\frac{N}{2T}$ is the *Nyquist frequency*), then the power in this frequency is *aliased* down to a lower frequency. In other words a plot of $|F_k|^2$ will be misleading, since some of the power at a given k may actually be coming from a higher frequency. This effect can result in poor images captured by digital cameras. The cure for this problem is to sample at a higher rate, or to filter the signal before digitization.

7.8 Exercises

Exercise 7.1

Let $\{f_n\}$, $n = 0, 1, \ldots, N-1$, be N samples of a real signal, and N be even.

(a) Show that

$$W^{-nk} + W^{-(N-n)k} = 2\cos\left(\frac{2\pi nk}{N}\right).$$

(b) Suppose the signal $\{f_n\}$ is an even function; i.e. $f_n = f_{N-n}$, for $n = 1, 2, \ldots, N/2-1$. Show that F_k is real.

Exercise 7.2

Suppose the signal $\{f_n\}$ is a square signal, defined as:

$$f_n = \begin{cases} 0, & \text{if } n = 0, 1, \ldots, \frac{N}{4} - 1 \text{ or } n = \frac{3N}{4}, \frac{3N}{4} + 1, \ldots, N-1, \\ 1, & \text{if } n = \frac{N}{4}, \frac{N}{4} + 1, \ldots, \frac{3N}{4} - 1. \end{cases}$$

Show that $F_{2k} = 0$, $k = 1, 2, \ldots, \frac{N}{2} - 1$.

Solution: We have

$$\begin{aligned} F_{2k} &= \frac{1}{N}\sum_{n=0}^{N-1} f_n W^{-n2k} \\ &= \frac{1}{N}\sum_{n=N/4}^{3N/4-1} W^{-n2k} \end{aligned}$$

Change vairable, let $i = n - N/4$, and so $n = i + N/4$, we have

$$\begin{aligned} F_{2k} &= \frac{1}{N}\sum_{i=0}^{N/2-1} W^{-(i+N/4)2k} \\ &= \frac{1}{N}\cdot W^{-2Nk/2}\sum_{i=0}^{N/2-1} W^{-i2k} \\ &= \frac{(-1)^k}{N}\cdot\frac{1 - W^{-2(N/2)k}}{1 - W^{-2k}} = 0 \end{aligned}$$

as desired. ★

Exercise 7.3

For a given input sequence f_i, why is the first component of its DFT, F_0, always equal to the average of the components of f_i?

Exercise 7.4

For a constant c, determine the DFT of a signal $f_0 + c, \ldots, f_{N-1} + c$ in terms of the DFT of the signal $f_0, \ldots, f_{N-1}$.

Solution: We know that

$$\begin{aligned} F_k &= \frac{1}{N} \sum_{n=0}^{N-1} (f_n + c) W^{-nk} \\ &= \frac{1}{N} \sum_{n=0}^{N-1} f_n W^{-nk} + \frac{1}{N} \sum_{n=0}^{N-1} c W^{-nk} \end{aligned}$$

Recall orthogonal identity of N^{th} roots of unity, we obtain that

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

as desired. ★

Exercise 7.5

Calculate the Discrete Fourier Transform of the following periodic time sequences by hand, both using the direct DFT formula.

(i) $f[n] = (1, 2, 2, 1) \quad (n = 0, \ldots, 3; \quad N = 4)$

(ii) $f[n] = (1, 2, 3, 2) \quad (n = 0, \ldots, 3; \quad N = 4)$

Why is the resulting transform real in this case?

Exercise 7.6

(Parseval's Theorem) Let f_n, $n = 0, \ldots, N-1$ be given data values (real or complex) and let F_k, $k = 0, \ldots, N-1$ be the DFT of f_n. Show that

$$\sum_{k=0}^{N-1} F_k \overline{F_k} = \frac{1}{N} \sum_{n=0}^{N-1} f_n \overline{f_n}.$$

Solution: Recall that we have

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

Hence we have that

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

Therefore we know that

$$\begin{aligned}\sum_{n=0}^{N-1} f_n \overline{f_n} &= \sum_{n=0}^{N-1} \left(\sum_{k=0}^{N-1} F_k W^{nk} \right) \left(\sum_{\ell=0}^{N-1} \overline{F_\ell} W^{-n\ell} \right) \\ &= \sum_{k=0}^{N-1} \sum_{\ell=0}^{N-1} F_k \overline{F_\ell} \sum_{n=0}^{N-1} W^{n(k-\ell)}\end{aligned}$$

where we know that there are exactly n pairs of (k, ℓ) whose values satisfy $k = \ell$, hence we have

$$\sum_{n=0}^{N-1} f_n \overline{f_n} = N \sum_{k=0}^{N-1} F_k \overline{F_k}$$

which is what we wanted. ★

Exercise 7.7

Derive an algorithm for determining the FFT of two real signals of the same length by performing a single FFT of a complex signal.

8 Google Page Link

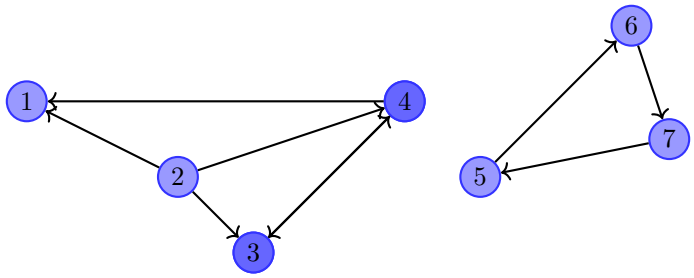
Question 8.1.

When one uses Google to search web pages for key words a set of pages is returned, each ranked in order of its importance. The question we try to answer in this section is how this ranking is obtained.

8.1 Introduction

We represent the web's structure as a directed graph

Example 8.1



Definition 8.1:

Nodes (circles) represent pages. **Arcs** (arrows) represent links from one page to another.

Definition 8.2: Degree

We will use **degree** to refer to a node's outdegree, the number of arcs leaving that node.

Comment 8.1

For instance,

$$\deg(4) = 2 \qquad \text{and} \qquad \deg(3) = 1$$

Definition 8.3: Adjacency Matrix

To store our directed graph, we can use a kind of *adjacency matrix*, G.

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

Then the (out)degree for node q is the sum of entries in column q.

Comment 8.2

Matrix G is not necessarily symmetric about the diagonal!

Example 8.2

For the above digraph, we have

$$G = \begin{array}{c} \begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{matrix} \\ \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{matrix} \end{array}$$

The basic page rank idea is as follows. A link from web page j to web page i can be viewed as a vote on the importance of page i by page j. We will assume that all outlinks are equally important (this could be changed easily), so that the importance conferred on page i by page j is simply $1/\deg(j)$ (assuming that there is a link from $j \to i$). But this simply gives some idea of the local importance of i, given that we are visiting page j. To get some idea of the global importance of page i, we have to have some idea of the importance of page j. This requires that we examine the pages which point to page j, and then we need to determine the importance of these pages, and so on.

8.1.1 The Random Surfer Model

Let us consider the hypothetical concept of a random surfer. This surfer selects each page in the Web in turn. From this initial page, the surfer then selects at random an outlink from this initial page, and visits this page. Another outlink is selected at random from this page, and so on. The surfer keeps track of the number of times each page is visited. After K visits, the surfer then begins the process again, by selecting another initial page.

Comment 8.3

This algorithm is presented below, where we assume that there are R pages in the web.

Code 8.1

```
Rank(m) = 0 , m = 1, ..., R
For m = 1, ..., R
    j = m
    For k = 1, ..., K
        Rank(j) = Rank(j) + 1
        Randomly select outlink l of page j
        j = l
    EndFor
EndFor
Rank(m) = Rank(m) / (K * R) , m = 1, ..., R
```

Notice that at the end, we estimate the overall importance as:

$$Rank(page\ i)\ =\ (Visits\ to\ page\ i)/(Total\ visits\ to\ all\ pages)$$

Discovery 8.1

Potential issues with this algorithm?

- The number of real web pages is monstrously huge, **many** iterations (large K, R) needed.
- Number of steps taken per random surf sequence must be large, to get a representative sample.
- What about dead end links? (Stuck on **one** page!)
- What about cycles in the graph? (Stuck on a **closed subset** of pages!)

Clearly, better strategies are needed.

8.2 Page Rank Modified

Definition 8.4: Markov Chain Matrix

Let P be a (large!) matrix of probabilities, where P_{ij} is the probability of randomly transitioning from page j to page i:

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

Comment 8.4

To create the **Markov Chain matrix** P from the adjacency matrix G, divide all entries of each column of G by the column sum (out-degree of the node).

8.2.1 Dead Ends

Comment 8.5

To deal with dead-end links, we will simply "teleport" to a new page at random!

Theorem 8.1

Mathematically, we define a column vector d such that:

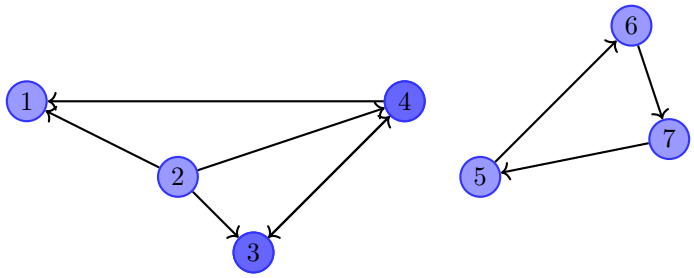
$$d_i = \begin{cases} 1, & \text{if } \deg(i) = 0 \\ 0, & \text{otherwise} \end{cases}$$

and vector $e = [1, 1, 1, \ldots, 1, 1]^T$ be a column vector of ones. Then if R is the number of pages, we augment P to get P' defined by:

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

Proof. Consider the example

Example 8.3



Notice that Page numbered 1 is an dead end, and hence we have

$$d = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T \qquad \text{and} \qquad R = 7$$

which gives that

$$\frac{1}{R} e d^T = \begin{bmatrix} 1/7 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1/7 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1/7 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1/7 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1/7 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1/7 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1/7 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

as desired. (This is an proof by example!) □

8.2.2 Cycles

Question 8.2.

How can we apply a similar trick to escape closed cycles of pages?

Solution: Most of the time (a fraction α), we follow links randomly, via P'. Occasionally, with some (usually small) probability, $(1 - \alpha)$, we teleport from any page to any other page:

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

★

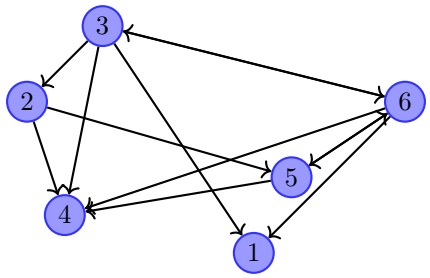
Discovery 8.2

Google purpotedly used $\alpha \approx 0.85$.

Definition 8.5: Google Matrix

The above result matrix M is called the **Google matrix**.

Example 8.4



The resulting Google matrix M is:

$$M = \begin{bmatrix} 1/6 & 1/12 & 5/24 & 1/6 & 1/12 & 5/24 \\ 1/6 & 1/12 & 5/24 & 1/6 & 1/12 & 1/12 \\ 1/6 & 1/12 & 1/12 & 1/6 & 1/12 & 5/24 \\ 1/6 & 1/3 & 5/24 & 1/6 & 1/3 & 5/24 \\ 1/6 & 1/3 & 1/12 & 1/6 & 1/12 & 5/24 \\ 1/6 & 1/12 & 5/24 & 1/6 & 1/3 & 1/12 \end{bmatrix}$$

8.3 Evolving The Probability Vector

Definition 8.6: Probability Vector

A probability vector is a vector q such that

$$0 \leq q_1 \leq 1 \qquad \forall\, i$$

and

$$\sum_i q_i = 1$$

Now we have:

- the probability vector describing the **initial state**, p^0.
- a Markov matrix M describing the **transition probabilities** among pages.

Their product Mp^0 tells us the probabilities of our surfer being at each page after **one transition**.

$$p^1 = Mp^0$$

Likewise, for any step n, next step probabilities are $p^{n+1} = Mp^n$.

Theorem 8.2

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

Proof. We know that $p_i^{n+1} \geq 0$ because it is the sum and products of probabilities. We can also show $\sum_i p_i^{n+1} = 1$, as follows:

$$\sum_i p_i^{n+1} = \sum_i \sum_j M_{ij} p_j^n = \sum_j \left(p_j^n \sum_i M_{ij} \right) = \sum_j p_j^n = 1$$

To be a probability vector, we also need $p_i^{n+1} \leq 1$, why is this true? □

Finally, Page Rank asks:

> With what **probability** does our surfer end up at each page after **many** steps, starting from $p^0 = \frac{1}{R}e$?

i.e., What is

$$p^\infty = \lim_{k \to \infty} (M)^k p^0$$

Higher probability in p^∞ vector implies greater importance. Then we can rank the pages by this importance measure.

8.4 Page Rank Summary

Result 8.1

1. Given a graph of a network, compute a corresponding Google transition (Markov) matrix...

$$M = \alpha \left(P + \frac{1}{R} e d^T \right) + (1 - \alpha) \frac{1}{R} e e^T$$

2. Repeatedly evolve a probability vector p^i via $p^{n+1} = Mp^n$ towards a steady state, approximating a "random surfer".

3. The site with the highest probability of being visited is considered most important/influential.

Code 8.2

$\mathbf{p}^0 = \mathbf{e}/R$
For $k = 1, ...,$ until converged
    $\mathbf{p}^k = M\mathbf{p}^{k-1}$
    If $\max_i |[\mathbf{p}^k]_i - [\mathbf{p}^{k-1}]_i| < tol$ then quit
EndFor

Lecture - Wednesday, March 19

8.5 Make Page Rank efficient

Definition 8.7: Dense

A matrix is **dense** if most or all entries are non-zero. Store in an $N \times N$ array, manipulate "normally".

Definition 8.8: Sparse

A matrix is **sparse** if most entries are zero.

Discovery 8.3

We wish to use a "sparse" data structure to save space (and time). Prefer algorithms that avoid "destroying" sparsity (i.e., filling in zero entries).

We have

$$M = \underbrace{\alpha\left(P + \frac{1}{R}ed^T\right)}_{sparse} + \underbrace{\frac{(1-\alpha)}{R}ee^T}_{dense}$$

Consider computing

$$Mp^n = \underbrace{\alpha P p^n}_{(1)} + \underbrace{\frac{\alpha}{R}ed^T p^n}_{(2)} + \underbrace{\frac{1-\alpha}{R}ee^T p^n}_{(3)}$$

Output p^{n+1} is a vector, and a sum of three vectors: first one is a sparse matrix-vector product, which can be done efficiently. The third one involves

$$ee^T p^n = e\,\underbrace{(e^T p^n)}_{\text{scalar}}$$

where the scalar $e^T p^n$ is simply one because p^n is a probability vector. Therefore the third expression can be simplified as

$$\frac{(1-\alpha)}{R}e^T$$

Similarly, for second expression, we compute

$$\frac{\alpha}{R}(d^T p^n)e$$

Therefore

$$p^{n+1} = Mp^n = (1) + (2) + (3)$$

has no dense matrix multiplication.

Comment 8.6

We never form M explicitly.

8.6 Convergence Analysis

Example 8.5

Returning to a small web where M is

$$M = \begin{bmatrix} \frac{1}{40} & \frac{1}{6} & \frac{37}{120} & \frac{1}{40} & \frac{1}{40} & \frac{1}{40} \\ \frac{9}{20} & \frac{1}{6} & \frac{37}{120} & \frac{1}{40} & \frac{1}{40} & \frac{1}{40} \\ \frac{9}{20} & \frac{1}{6} & \frac{1}{40} & \frac{1}{40} & \frac{1}{40} & \frac{1}{40} \\ \frac{1}{40} & \frac{1}{6} & \frac{1}{40} & \frac{1}{40} & \frac{9}{20} & \frac{7}{8} \\ \frac{1}{40} & \frac{1}{6} & \frac{37}{120} & \frac{9}{20} & \frac{1}{40} & \frac{1}{40} \\ \frac{1}{40} & \frac{1}{6} & \frac{1}{40} & \frac{9}{20} & \frac{9}{20} & \frac{1}{40} \end{bmatrix}$$

with a ranking vector computed via the power method yielding

$$[0.05205, \quad 0.07428, \quad 0.05782, \quad 0.34797, \quad 0.19975, \quad 0.26810]^T$$

after 10 *iterations. One can find that the lone eigenvector for eigenvalue 1 for* M *is then seen to be*

$$v = \left[\frac{3080}{59569}, \quad \frac{4389}{59569}, \quad \frac{3420}{59569}, \quad \frac{1184000}{3395433}, \quad \frac{9560}{47823}, \quad \frac{16000}{59569}\right]^T$$

which with floating point numbers becomes

$$v \approx [0.05170, \quad 0.07367, \quad 0.05741, \quad 0.34870, \quad 0.19990, \quad 0.26859]^T.$$

8.6.1 Some Technical Results

Theorem 8.3

Every Markov matrix Q has 1 as an eigenvalue.

Proof. The eigenvalues of Q and Q^T are the same (since Q and Q^T have the same characteristic polynomials). Since $Q^T e = e$, we have that $\lambda = 1$ is an eigenvalue of Q^T. Thus $\lambda = 1$ is an eigenvalue of Q. □

Theorem 8.4

Every (possibly complex) eigenvalue λ of a Markov matrix Q satisfies

$$|\lambda| \leq 1$$

Thus 1 is the largest eigenvalue of Q.

Proof. This result actually follows from the Gershgorin Circle Theorem (a result that can be found in linear algebra texts) applied to the eigenvalues of Q^T. Since the eigenvalues of Q and Q^T are the same, the theorem follows. □

Definition 8.9: Positive Markov Matrix

A Markov matrix Q is a positive Markov matrix if

$$Q_{ij} > 0, \qquad \forall\, i, j$$

Theorem 8.5

If Q is a positive Markov matrix, then there is only one linearly independent eigenvector of Q with $|\lambda| = 1$.

Proof. See, for example, (Grimmett and Stirzaker, Probability and Random Processes, Oxford University Press, 1989.) □

8.6.2 Convergence Proof

Theorem 8.6

If M is a positive Markov matrix, the iteration

$$p^{\infty} = \lim_{k\to\infty} (M^k) p^0$$

converges to a unique vector p^{∞}, for any initial probability vector p^0.

Proof. Let $\mathbf{x}_l$ be an eigenvector of M, corresponding to the eigenvalue λ_l. Suppose that M has a complete set of eigenvectors, in other words, we can represent $\mathbf{p}^0$ as

$$\mathbf{p}^0 = \sum_l c_l \mathbf{x}_l$$

for some scalars c_l. (We do not have to make this assumption, but it simplifies the proof). Suppose also that we order these eigenvectors so that $|\lambda_1| > |\lambda_2| \geq \ldots$ so that $\mathbf{x}_1$ corresponds to the unique eigenvector with $\lambda_1 = 1$. Then

$$(M)^k \mathbf{p}^0 = c_1 \mathbf{x}_1 + \sum_{l=2}^{R} c_l (\lambda_l)^k \mathbf{x}_l \,.$$

From Theorem (7.5.5), we have that $|\lambda_l| < 1$ for all $l > 1$, so that

$$\lim_{k\to\infty} (M)^k \mathbf{p}^0 = c_1 \mathbf{x}_1$$

for any $\mathbf{p}^0$. $c_1\mathbf{x}_1$ cannot be identically zero, since $\mathbf{p}^0$ is a probability vector, and hence $\mathbf{p}^{\infty}$ is a probability vector. Hence $c_1 \neq 0$ and $\mathbf{x}_1$ cannot be the zero vector. Uniqueness follows since if we start the iteration with another probability vector

$$\mathbf{q}^0 = \sum_l b_l \mathbf{x}_l$$

for some coefficients b_l, then

$$\mathbf{q}^{\infty} = \lim_{k\to\infty} (M)^k \mathbf{q}^0 = b_1 \mathbf{x}_1 \,.$$

But, for given $\mathbf{x}_1$, since $\mathbf{q}^{\infty}, \mathbf{p}^{\infty}$ are probability vectors, we have $b_1 = c_1$. □

9 Numerical Linear Algebra

Question 9.1.

Consider an $n \times n$ matrix A, a solution vector x and a right hand side vector b. We wish to solve

$$Ax = b$$

Comment 9.1

The classical way to solve a linear system of equations is via Gaussian elimination.

Result 9.1

However, the standard computer techniques referred to as Gaussian elimination for solving $Ax = b$ are based on factoring A into triangular factors. This view of Gaussian elimination has the following major steps:

1. Compute triangular factors L and U from A such that $A = LU$
2. Solve $Lz = b$
3. Solve $Ux = z$

Discovery 9.1

Of course this may not be possible without re-ordering the equations. In matrix terms, re-ordering the equations is accomplished by multiplying A and b by a 'permutation' matrix, P, that is solving an equivalent system $PAx = Pb$.

9.1 Solving via Matrix Factorization

Comment 9.2

In fact it is not difficult to see that a triangular factorization of A is related to the row reduction process. Here the factorization algorithm for factoring $A = LU$.

Example 9.1

We will see in the next section and the example below that the LU factorization process is almost identical to Gaussian elimination, the main difference being that we wish to retain the various multiplicative factors: Let

$$A = \begin{bmatrix} 10 & -7 & 0 \\ -3 & 2 & 6 \\ 5 & -1 & 5 \end{bmatrix}.$$

First stage of LU factorization of A:

$$\begin{bmatrix} 10 & -7 & 0 \\ -3 & 2 & 6 \\ 5 & -1 & 5 \end{bmatrix} \xrightarrow{R2-(\frac{-3}{10})R1} \begin{bmatrix} 10 & -7 & 0 \\ -0.3 & -0.1 & 6 \\ 5 & -1 & 5 \end{bmatrix} \xrightarrow{R3-(\frac{5}{10})R1} \begin{bmatrix} 10 & -7 & 0 \\ -0.3 & -0.1 & 6 \\ 0.5 & 2.5 & 5 \end{bmatrix}$$

Proceeding with the second (and final) stage of factorization, we obtain

$$\begin{bmatrix} 10 & -7 & 0 \\ -0.3 & -0.1 & 6 \\ 0.5 & -25 & 155 \end{bmatrix}$$

from which we extract the factors

$$L = \begin{bmatrix} 1 & 0 & 0 \\ -0.3 & 1 & 0 \\ 0.5 & -25 & 1 \end{bmatrix} \quad \text{and} \quad U = \begin{bmatrix} 10 & -7 & 0 \\ 0 & -0.1 & 6 \\ 0 & 0 & 155 \end{bmatrix}.$$

One can easily verify that $LU = A$ in this example.

Comment 9.3

The diagonal entries of L is always filled with 1's, and the entries below the diagonal are preserved from the original resulting matrix. These entries represent the multiplicative factors.

Definition 9.1: LU Factorization Notation

In order to save space, we store the LU factorization back in A: Consider the above example again, we have

$$A = LU = \begin{bmatrix} 10 & -7 & 0 \\ \boxed{-0.3} & -0.1 & 6 \\ \boxed{0.5} & \boxed{-25} & 155 \end{bmatrix}$$

whose boxed entries belong to L and the rest belong to U.

Code 9.1: LU factorization

```
    For k = 1, ..., n                // iterate over all rows
        For i = k + 1, ..., n        // iterate each row i below row k
            mult := aik/akk          // determine row i's multiplicative factor
            aik := mult              // store this factor
            For j = k + 1, ..., n    // iterate over all columns in the row
                aij := aij - mult * akj     // subtract the scaled data

// Note that the resulting factors are stored back into A for the sake of
    space
```

Question 9.2.

Why is solving $LUx = b$ better than solving $Ax = b$?

Solution: L and U are both triangular: all entries above, or below, the diagonal are zero, respectively. This makes them easier (i.e., more efficient) to solve. ★

9.1.1 Forward Solve & Backward Solve

Definition 9.2: Forward Solve

Solving $Lz = b$ for z is called **forward solve**.

Definition 9.3: Backward Solve

Solving $Ux = z$ for x is called **backward solve**.

Code 9.2: Forward Solve

```
For i = 1, ..., n
    zi := bi
    For j = 1, ..., i - 1
        zi := zi - lij * zj
```

Code 9.3: Backward Solve

```
For i = n, ..., 1
    xi := zi
    For j = i + 1, ..., n
        xi := xi - uij * x
```

Discovery 9.2

Recall that in the psuedocode for LU factorization, we have

$$mult := a_{ik}/a_{kk}$$

where $a_{kk} = 0$ should be avoided, and when its value is nearly zero, numerical instability could happen because a large factor can cause large floating point error during subtraction and magnify existing floating point error.

Theorem 9.1

To fix the problem we discovered, we do:

```
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.
```

9.1.2 Permutation Matrix

Comment 9.4

We wish to find a modified factorization of A such that $PA = LU$ where P is a permutation matrix.

Definition 9.4: Permutation Matrix

A **permutation matrix** P is a matrix whose eect is to swap rows of the matrix it is applied to (i.e., multiplied with).

Example 9.2

Solve for

$$\begin{bmatrix} 1 & 4 & 5 \\ -2 & 3 & 3 \\ 3 & 0 & 6 \end{bmatrix} x = \begin{bmatrix} 4 \\ 1 \\ -3 \end{bmatrix}$$

Solution: We first want to find the $PA = LU$ factorization. We first want to perform row swappings on A. Notice that we wish to swap row 1 and 3 first, and then 2 and 3, hence we have

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

With more computations, we obtain

$$\begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 4 & 5 \\ -2 & 3 & 3 \\ 3 & 0 & 6 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1/3 & 1 & 0 \\ -2/3 & 3/4 & 0 \end{bmatrix} \begin{bmatrix} 3 & 0 & 6 \\ 0 & 4 & 3 \\ 0 & 0 & 19/4 \end{bmatrix}$$

★

We want to know the (asymptotic) cost to solve a system of size $n \times n$. We will measure cost in total FLOPs: FLoating point OPerations.

Definition 9.5: FLOPS

FLOPS, or **FLoating point OPerations**, is approximated as the number of

$$adds + subtracts + multiplies + divides$$

9.1.3 Costs of Gaussian Elimination

Recall

```
For k = 1, ..., n
    For i = k + 1, ..., n
        mult := aik/akk
        aik := mult
        For j = k + 1, ..., n
            aij := aij - mult * akj
```

Result 9.2

Solution: In the inner for loop, there is one division, for the inner inner for loop, there is one subtraction and one multiplication. Hence summing over all the loops we get:

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

★

9.1.4 Cost of Triangular solve

Recall backward solve

```
For i = n, ..., 1
    xi := zi
    For j = i + 1, ..., n
        xi := xi - uij * x
```

Result 9.3

The total number of FLOPs is given by

$$\sum_{i=1}^{n}\left(1+\sum_{j>i+1}^{n}2\right)=n^2+O(n)$$

9.2 Gaussian Elimination ≡ Matrix Factorization

Comment 9.5

We viewed row-swapping as a matrix; we can do the same for row-subtraction!

Discovery 9.3

Zeroing a (sub-diagonal) entry of a column by row subtraction can be written as applying a specific matrix M such that

$$MA^{old}=A^{new}$$

where

- *Aold* is the original matrix.
- *Anew* is the matrix after subtracting the specific row.

Example 9.3

The operation

$$(2^{nd} \text{ row}) := (2^{nd} \text{ row}) - \frac{a_{2,1}}{a_{1,1}} (1^{st} \text{ row})$$

can be written as a matrix:

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -\frac{a_{2,1}}{a_{1,1}} & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

M is the identity matrix, but with a zero entry replaced by the (negative of the) necessary multiplicative factor.

The matrix left at the end is U.

Example 9.4

For example, in 3×3 case, we have shown:

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

Therefore

$$A = \left(M^{(3)} M^{(2)} M^{(1)}\right)^{-1} U = \underbrace{\left(M^{(1)}\right)^{-1} \left(M^{(2)}\right)^{-1} \left(M^{(3)}\right)^{-1}}_{L} U$$

Define

$$L = \left(M^{(1)}\right)^{-1} \left(M^{(2)}\right)^{-1} \left(M^{(3)}\right)^{-1}$$

and we have our factorization!

Question 9.3.

But what is $\left(M^{(k)}\right)^{-1}$?

Discovery 9.4

The inverse of this simple matrix form is the same matrix, but with the off-diagonal entry negated.

9.2.1 Remark: Solving $Ax = b$ by Matrix Inversion

One can show that the finding inverse is actually more expensive (in FLOPs) than using our "factor and triangular solve" strategy. It also generally incurs more F.P. error.

Result 9.4

Most numerical algorithms avoid ever computing A^{-1}.

9.3 Condition numbers and Norms

Definition 9.6: Norms

Norms are measurements of "size"/ magnitude for vectors or matrices.

Definition 9.7: Conditioning

Conditioning describes how the output of a function/operation/matrix changes due to changes in input.

The norm of a vector is a measure of its size. Let x be a vector with components

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

and we define

Definition 9.8: 1-norm

$$\|x\|_1 = \sum_{i=1}^{n} |x_i|$$

Definition 9.9: 2-norm

$$\|x\|_2 = \left(\sum_{i=1}^{n} x_i^2 \right)^{1/2}$$

Definition 9.10: ∞-norm

$$\|x\|_\infty = \max_i |x_i|$$

Definition 9.11: p-norm

Usually, these *norms* are referred to as the *p-norm*, i.e.,

$$\|x\|_p \; ; \; p = 1, 2, \infty$$

Code 9.4

Python computes these vector norms using the `np.linalg.norm` command.

9.3.1 Properties of Norms

We have

$$\begin{aligned} \|x\| = 0 \qquad &\to x_i = 0,\ \forall\, i \\ \|\alpha x\| = \alpha\|x\|, \qquad &\alpha = \text{scalar} \\ \|x+y\| \le \|x\| + \|y\| \end{aligned}$$

9.3.2 Matrix Norm

Definition 9.12: Matrix Norm

Matrix norms are often defined/"induced" as follows, *using p-norms* of vectors:

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

Discovery 9.5

There are simpler equivalent definitions in some cases:

$$\|A\|_1 = \underbrace{\max_j \sum_{i=1}^{n} |A_{ij}|}_{\text{(max absolute column sum)}} \qquad \|A\|_\infty = \underbrace{\max_i \sum_{j=1}^{n} |A_{ij}|}_{\text{(max absolute row sum)}}$$

Discovery 9.6

The matrix's 2-norm relates to the eigenvalues. Specifically, if λ_i are the eigenvalues of $A^T A$, then

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

9.3.3 Matrix Norm Properties

1. $\|A\| = 0 \iff A_{ij} = 0 \quad \forall i, j.$
2. $\|\alpha A\| = |\alpha| \cdot \|A\| \quad$ for scalar α.
3. $\|A + B\| \le \|A\| + \|B\|$
4. $\|Ax\| \le \|A\| \cdot \|\mathbf{x}\|.$
5. $\|AB\| \le \|A\| \cdot \|B\|$
6. $\|I\| = 1.$

Definition 9.13: Frobenius Norm

Another fun matrix norm that is not "induced" by a standard vector norm is the **Frobenius** norm.

$$\|A\|_F = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n} A_{ij}^2}$$

Lecture 22 - Monday, March 31

9.4 Conditioning

9.4.1 Perturbing b

Example 9.5: Conditioning Example – Perturbing b

Consider the system:

$$\begin{bmatrix} 1 & 2 \\ 2 & 3.999 \end{bmatrix} x = \begin{bmatrix} 4 \\ 7.999 \end{bmatrix}$$

And the similar system (perturbed b):

$$\begin{bmatrix} 1 & 2 \\ 2 & 3.999 \end{bmatrix} x = \begin{bmatrix} 4.001 \\ 7.998 \end{bmatrix}$$

How do the solutions differ?

Theorem 9.2

Analysis of conditioning when perturbing b.

Solution: Given an equation $Ax = b$, perturbing b by Δb. This gives

$$A(x + \Delta x) = b + \Delta b$$

Subtracting off $Ax = b$ yields us

$$A\Delta x = \Delta b \qquad \text{or} \qquad \Delta x = A^{-1}b$$

We wish to find the relative change in x, $\dfrac{\|\Delta x\|}{\|x\|}$, given a relative change in b, $\dfrac{\|\Delta b\|}{\|b\|}$. Apply norm rules to $Ax = b$:

$$\|b\| = \|Ax\| \leq \|A\|\|x\| \qquad \text{or} \qquad \frac{\|A\|}{\|b\|} \geq \frac{1}{\|x\|}$$

Recall that we also have

$$\|\Delta x\| \leq \|A^{-1}\|\|\Delta b\|$$

Combining the above two inequalities (by multiplication) we obtain

$$\frac{\|\Delta x\|}{\|x\|} \leq (\|A^{-1}\|\|\Delta b\|) \cdot \left(\frac{\|A\|}{\|b\|}\right) = \underbrace{(\|A^{-1}\|\|A\|)}_{\kappa} \cdot \left(\frac{\|\Delta b\|}{\|b\|}\right)$$

where the *condition number* κ bounds the relative change in x due to relative change in b. ★

9.4.2 Perturbing A

Theorem 9.3

Analysis of conditioning when perturbing A.

Solution: Consider perturbing A:

$$(A + \Delta A)(x + \Delta x) = b$$

Subtracting off $Ax = b$ and rearrance, we obtain

$$A(\Delta x) = -(\Delta A)(x + \Delta x) \qquad \text{or} \qquad \Delta x = -A^{-1}(\Delta A)(x + \Delta x)$$

Take norms on both sides:

$$\|\Delta x\| \leq \|A^{-1}\|\|\Delta A\|\|x + \Delta x\|$$

Multiply by $\dfrac{\|A\|}{\|A\|} = 1$ and take x terms to LHS:

$$\frac{\|\Delta x\|}{\|x + \Delta x\|} \leq \underbrace{(\|A\|\|A^{-1}\|)}_{\kappa} \cdot \frac{\|\Delta A\|}{\|A\|}$$

Condition number κ dictates a bound on relative change in x. ★

Result 9.5

Condition number of a matrix A is denoted $\kappa(A) = \|A\| \cdot \|A^{-1}\|$.

1. $\kappa \approx 1 \to A$ is well-conditioned.
2. $\kappa \gg 1 \to A$ is ill-conditioned.

Another useful property of vector/matrix norms is equivalence.

Discovery 9.7

The norms we've looked at differ from one another by no more than a constant factor.

$$C_1\|x\|_a \leq \|x\|_b \leq C_2\|x\|_a$$

for constants C_1, C_2, and norms $\|\cdot\|_a$ and $\|\cdot\|_b$.

9.4.3 Residual vs. Error

Definition 9.14: Residual

As a proxy (stand-in) for error, we often use the residual r:

$$r = b - A(x_{approx})$$

i.e., by how much does our computed solution fail to satisfy the original problem.

Question 9.4.

How does the residual r relate to error?

Solution: Assuming $x_{\text{approx}} = x + \Delta x$, we have

$$r = b - A(x + \Delta x)$$

or

$$A(x + \Delta x) = b - r$$

(r looks just like a perturbation of b!) So, applying our earlier bound using $\Delta b = r$, we have

$$\frac{\|\Delta x\|}{\|x\|} \leq \kappa(A) \frac{\|r\|}{\|b\|}$$

Result 9.6

The solution's relative error, $\dfrac{\|\Delta x\|}{\|x\|}$, is bounded by the condition number times the relative size of residual r w.r.t. to rhs b.

If $\kappa \approx 1$, a small residual indicates a small relative error. But, if κ is large, residual could still be small while error is quite large if problem is poorly conditioned.

★

For implementation in floating point arithmetic, Gaussian elimination with pivoting is as stable and accurate as any other method. This is one of the reasons that it is so commonly used. It is known that Gaussian elimination with pivoting produces a computed solution $\hat{x}$ which satisfies

$$(A + E)\hat{x} = b,$$

where $\|E\| = \epsilon_{\text{machine}}\|A\|$, i.e. Gaussian elimination with pivoting solves a *nearby* problem exactly. From the result from perturning A, we have

$$\frac{\|x - \hat{x}\|}{\|\hat{x}\|} \leq \kappa(A)\epsilon_{\text{machine}}$$

Note that conditioning is a property of the problem, not a property of the equation solving algorithm.

Example 9.6

Find the condition numbers $\kappa_1(A)$ and $\kappa_\infty(A)$ for the matrix

$$A = \begin{bmatrix} 3 & 0 & 0 \\ 0 & 2 & 1 \\ 0 & 1 & 2 \end{bmatrix}$$

using the fact that

$$A^{-1} = \begin{bmatrix} 1/3 & 0 & 0 \\ 0 & 2/3 & -1/3 \\ 0 & -1/3 & 2/3 \end{bmatrix}$$

Solution: We have

$$\|A\|_1 = \max_j \sum_i |A_{ij}| = \max\{(3+0+0), (0+2+1), (0+1+2)\} = 3$$

and

$$\|A^{-1}\|_1 = \max_j \sum_i |A_{ij}| = \max\{(1/3+0+0), (0+2/3+1/3), (0+1/3+2/3)\} = 1$$

and hence $\kappa_1(A) = 3 \times 1 = 3$. Similarly, we can find that $\kappa_\infty(A) = 3$ as well. ★

Example 9.7

Using the above matrix A, what is $\kappa_2(A)$, if we know the eigenvalues λ_i of $A^T A$ are $1, 9, 9$?

Solution: We know that

$$\|A\|_2 = \max_i \sqrt{|\lambda_i|} = \sqrt{|9|} = 3$$

For $\|A^{-1}\|_2$, we need the eigenvalues of $A^{-T}A^{-1} = (AA^T)^{-1}$. Since A happens to be symmetric, we know that $AA^T = A^T A$, so they have the same eigenvalues. Now we know that the eigenvalues of $A^{-T}A^{-1}$ are $1, 1/9, 1/9$, so

$$\|A^{-1}\|_2 = \max_i \sqrt{|\lambda_i'|} = \sqrt{|1|} = 1$$

and hence $\kappa_2(A) = 3 \times 1 = 3$. ★

Question 9.5.

Why is it true that $\kappa(\alpha A) = \kappa(A)$?

Solution: We have

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

★

Question 9.6.

Why is it true that $\kappa(A) \geq 1$?

Solution: We have

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

★

9.5 Exercises for Numerical Linear Algebra

Exercise 9.1

True or false: If x is any n-vector, then $\|x\|_1 \geq \|x\|_\infty$.

Exercise 9.2

True or false: If $\|A\| = 0$, then $A = 0$.

Exercise 9.3

Derive the algorithm to compute the UL decomposition of a matrix A, namely,

$$A = UL$$

where U and L are the upper and lower triangular matrices, respectively. We assume that no zero pivots will occur in this case, so no pivoting strategy is needed.

Exercise 9.4

Assume that you are given an LU factorization of an $n \times n$ matrix A. Show how to use this to solve

$$A^2\vec{x} = \vec{b}$$

with computational complexity $O(n^2)$.

Exercise 9.5

Assume that you are given the decomposition $A = LU$ where A is an $n \times n$ matrix. Describe how you can use this decomposition to solve the system

$$A^T\vec{x} = \vec{b}$$

with computational complexity $O(n^2)$.

Exercise 9.6

Classify each of the following matrices as well-conditioned or ill-conditioned, and provide a brief explanation for your answer in each case.

$$\begin{bmatrix} 10^{10} & \\ & 10^{-10} \end{bmatrix} \quad \begin{bmatrix} 10^{10} & \\ & 10^{10} \end{bmatrix} \quad \begin{bmatrix} 10^{-10} & \\ & 10^{-10} \end{bmatrix}$$

Index