Orthogonal Subspaces

4 Matrix Spaces:

1) $N(\underline{A})$ is the orthogonal complement of $C(\underline{A}^T)$ in $\mathbb{R}^m$,

2) $N(\underline{A}^T)$ is the orthogonal complement of the $C(\underline{A})$

Orthogonal complement means every vector in one space is orthogonal to all vectors in the other space.

$\underline{A}\,\underline{x} = \underline{b}$ let $\underline{y} \in N(\underline{A})$

$\Rightarrow \underline{A}\,\underline{x} + \underline{A}\,\underline{y} = \underline{b}$

Approximation & Round-off Errors

- In many cases, no analytic solution exists
- Numerical solutions provide approximate results that should be close to the true result.
- But: we can not typically compute the errors of the numerical method,
  - Rarely is the input exact
  - Algorithms introduce errors
  - → The results thus depend on both of these errors.

The question is this:

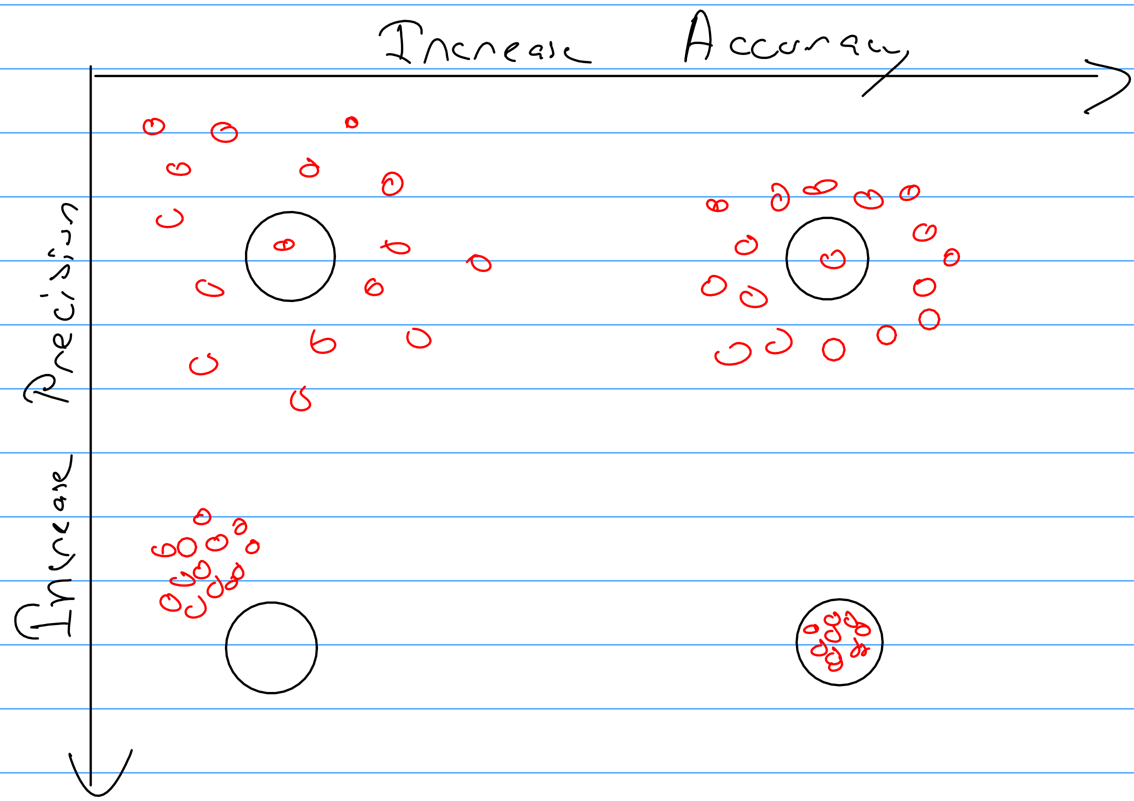
"How much error is present in our calculation & is that error tolerable?"

Holy Grail: Identification, quantification & minimization of error,

Accuracy vs Precision

- Accuracy: How the value to the true value, Typically thought of "how accurate is the method"?

- Precision: How closely do the computed values agree.



Numerical errors:
Errors arise from numerical approximations, model choice, system, etc.

Truth = approximate Value + Error

$$x^* = x + e$$

$\Rightarrow \quad e = x^* - x \qquad$ The error,

Relative Error : $\dfrac{\text{Truth} - \text{approx}}{\text{truth}}$

$$e_{rel} = \frac{x^* - x}{x^*}, \quad e_{rel\%} = e_{rel} * 100\%$$

Typically we can't compute the error. Why?

You need the true Solution!

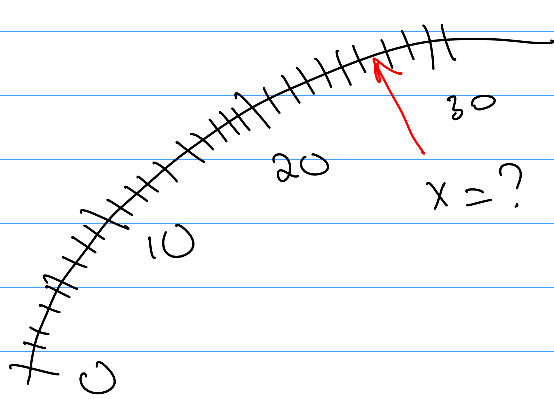
If you have that, why approximate?

Approximate error via:

$$e_{approx} = \frac{x^{new} - x^{old}}{x^{old}}$$

Significant Figures

Look at this gauge!



$x = 26.5$?

$= 26.55$?

Significant figures tell how much useful information do we really have.

Typically the digits that we are certain of + one more.

ex.) 53,800

5.38×10^4 3 sig digits
5.380×10^4 4
5.3800×10^4 5

ex.) leading zeros do not count towards this.

$$\left.\begin{array}{l} 0.001753 \\ 0.01753 \\ 0.1753 \end{array}\right\} \text{All have 4 sig. figures}$$

Sig figs tell us which numbers you can use with confidence.

- 32-bit machine : 7 significant figures.
- 64-bit machine : 15 significant figures.

- Double Precision : more precise than single precision, but slower to use.

$\pi = 3.14159265358979|3238462643...$

False Significant Figures

ex.) $\frac{3.25\cancel{X}}{1.96\cancel{X}} = 1.6581 6326 ....$ (from matlab)

In practice, only report 1.65 (Chopping) or 1.66 (Rounding)

why?

Because in reality we do not know the third decimal point value.

ex.) $3.259 / 1.960 = 1.662755...$
$3.250 / 1.969 = 1.65058...$ } Chopping

$3.254 / 1.955 = 1.66445...$
$3.245 / 1.964 = 1.6522...$ } Rounding

Numbering System

Base 10 (Decimal) 0, ..., 9

8 6 4 0 9

$9 \times 10^0 = 9$

$0 \times 10^1 = 0$

$4 \times 10^2 = 400$

$6 \times 10^3 = 6000$

$8 \times 10^4 = 80000$

86409

Base-2 (Binary) 0, 1

1 0 1 0 1 1 0 1

$1 \times 2^0 = 1$

$0 \times 2^1 = 0$

$1 \times 2^2 = 4$

$1 \times 2^3 = 8$

$0 \times 2^4 = 0$

$1 \times 2^5 = 32$

$0 \times 2^6 = 0$

$1 \times 2^7 = 128$

173
(in base-10)

Other Systems

Base - 8 (Octal) 0, 1, 2, 3, 4, 5, 6, 7

Base-16 (HexaDecimal) : 0, 1, , ..., 15

base-10 : $0.3125 = 3\times10^{-1} + 1\times10^{-2} + 2\times10^{-3} + 5\times10^{-4}$

base 2 : $101101 : 1\times2^5 + 0\times2^4 + 1\times2^3 + 1\times2^2 + 0\times2^1 + 1\times2^0$
$= 45$

$$0.1011 = 1\times2^{-1} + 0\times2^{-2} + 1\times2^{-3} + 1\times2^{-4} = \frac{11}{16}$$

Finite Precision Arithmetic

- One source of error is the limitation of the computer to represent a finite number of digits.

- Operations between these finite representations of numbers introduce round-off error,

Look at how numbers are stored.

Integers.

Look at an 8-bit word

There are 8 pieces of base-2 digits
8 values of 0 or 1

±

sign

0 => positive — Other 7 bits give the #
1 => negative — in base-2

ex.) 1 | 1 0 0 1 0 1 1

$\Rightarrow -(2^6 + 2^3 + 2^1 + 2^0) = -75$

Thus, the smallest # is:

$0000000_{\text{base } 2} = 0_{\text{base } 10}$

$1111111_{\text{base } 2} = 127_{\text{base } 10}$

with the first bit: -127 to 127

Since $+0 = -0$

$\Rightarrow$ Use -0 to represent -128

=> 8-bit can represent $2^8 = 256$ pieces of data $(-128 \to 127)$

- Integer Arithmetic is exact so long as no remainder!

$\frac{8}{2} = 4$ $\qquad$ $\frac{7}{2} = 3$

- You can get under or over flow,

Underflow: $-74 * 2 = -148$ ← outside of $-128 \to 127$

Overflow: $123 + 45 = 168$ ← outside of $-128 \to 127$

In modern computers: 32-bit or 64-bit

32-bit: $2^{31} = 2{,}147{,}483{,}648$

64-bit: $2^{63} = 9{,}2233720368 5 \times 10^{18}$

Side note! This is the reason that old computers could not use more than 4 Gb of memory, 2 Gb for OS + 2 Gb other.

Floating Point Representation

Real numbers (called floating point) are stored via the IEEE 754 specification.

Sign	signed exponent	mantissa

Sign is 1 or 0 for negative or positive

Signed exponent is the maximum value of the base

Mantissa contains the significant digits.

$$\boxed{\pm}\;\underbrace{\boxed{\pm}\;\boxed{e_1 \cdots e_m}}_{\text{Signed exponent}}\;\underbrace{\boxed{d_1 d_2 \cdots d_p}}_{\text{mantissa}}$$

$$N = \pm . d_1 d_2 d_3 \cdots d_p \, B^e = m B^e$$

m = mantissa
B = base of the # system
e = "signed" exponent

ex.) 8-bit base-10 representation

$$1 \mid 0\,9\,5 \mid 1\,4\,6\,7 \qquad B=10$$

Mantissa: $m = -(1\times10^{-1} + 4\times10^{-2} + 6\times10^{-3} + 7\times10^{-4})$
$= -0{,}1467$

Signed exp.: $e = +(9\times10^{1} + 5\times10^{0}) = 95$

$$10951467_{\text{base}10} = mB^{e} = -0{,}1467\times10^{95}$$

Normalization

$$1 \text{ in}^2 = \frac{1}{144} \text{ ft}^2 = 0.\overbrace{006944}^{\text{need to store}} \text{ ft}^2$$

This is less accurate than

$$1 \text{ in}^2 = 0.\underbrace{694444}_{\text{need to store}} \times 10^{-2} \text{ ft}^2$$

- Remove leading zeros by lowering the exponent.

In general you want the mantissa to be bounded by:

$$\frac{1}{B} \leq m < 1 \rightarrow \quad \text{base-10} \quad \frac{1}{10} \leq m < 1$$

$$\text{base-2} \quad \frac{1}{2} \leq m < 1$$

If $m < 1/2$, multiply by 2, increase exponent

Single Precision

A real number is stored in 4 bytes (for 32 bit)

bit (binary digit): 0 or 1
1 byte = 8 bits
Word = 4 bytes (32-bit), 8 bytes (64-bit)

32 bit $\begin{cases} 23 \text{ for digits} \\ 8 \text{ for signed exponent} \\ 1 \text{ for sign} \end{cases}$

Max exponent: $\pm 2^7 = \pm 128$

64 bit (Double Precision)

64 bits $\begin{cases} 52 \text{ for the digits} \\ 11 \text{ for signed exp.} \\ 1 \text{ for sign} \end{cases}$

Signed exponent: $\pm 2^{10} = \pm 1024$

Single vs. Double

	Smallest	Largest
Single	$\pm .10\ldots \times 2^{-126} \approx \pm 1.17\times 10^{-38}$	$\pm .111 \times 2^{127} \approx \pm 3.40\times 10^{38}$
Double	$\pm .10\ldots0 \times 2^{-1022} \approx 2.22\times 10^{-308}$	$\pm .111 \times 2^{1023} \approx 1.79\times 10^{308}$

Note: You can't access 10^{-128} or 10^{-1024} because those are reserved for infinity / NaN

Matlab uses Double precision.

Round off errors

Round off error is when you do not have enough bits to store the answer.

In Double precision, the precision is 2^{-52} (as determined by the mantissa)

This is roughly 2.22×10^{-16} ← called machine precision & given by eps in matlab

Simplified example of Round-off error.

Addition Problem: $0.99 + 0.0044 + 0.0042$
$= 0.9986$ (exact)

3-Digit arithmetic:
$(0.99 + 0.0044) + 0.0042 = 0.994 + 0.0042 = 0.998$

$0.99 + (0.0044 + 0.0042) = 0.99 + 0.0086 = 0.999$

There are ways to combat this.

Cancellation error:

Look at $x^2 - bx + 1 = 0$, b is large, r close to b

$r = \sqrt{b^2-4}$

$x_1 = \frac{b+r}{2}$ $\qquad$ $x_2 = \frac{b-r}{2}$ $\leftarrow$ If b & r are close → potential for errors.

Reformulate:

$$x_2 = \frac{b-r}{2}\,\frac{b+r}{b+r} = \frac{b^2-r^2}{2(b+r)} = \frac{4}{2(b+r)} = \frac{2}{b+r}$$

ex.) Let $b = 97$, $r = 96.9794$

$$x^2 - 97x + 1 = 0$$

exact: 0.01031
Standard: 0.01050 (3-digit math)
reformulate: 0.01031