Conditioning and Stability

- Analogous concepts:
  - Conditioning of a *problem* = sensitivity to data errors
  - Stability of an *algorithm* = sensitivity to errors in computation

- Conditioning of a problem
  - problem solution is a map from input $x$ to solution $f(x)$
  - PICTURE: error/uncertainty in data ($x^\wedge$), and error in solution ($f(x^\wedge)$)

- "backward error" $x - x^\wedge$
- "forward error" $f(x) - f(x^\wedge)$

- "well-conditioned" = insensitive
  "ill-conditioned" = sensitive

- How to make this notion *quantitative*?
- ratio of relative forward error to relative backward error

$$K = \frac{\text{rel. forward err.}}{\text{rel. backward err.}} = \frac{|f(x^\wedge) - f(x)| / |f(x)|}{|x^\wedge - x| / |x|}$$

- rearranging, see that $K$ acts like "amplification factor"

$$\text{rel. forward err.} = K \times \text{rel. backward err.}$$

- ill-conditioned --- large $K$
- well-conditioned --- small $K$ or $K$ close to 1
- Usually what we can derive is an upper bound for $K$, so that we get bound on rel. forward err.

$$\text{rel. forward err.} \leq K \times \text{rel. backward err.}$$

If $f$ is differentiable, $\Delta x = x + \Delta x$

$$f(x + \Delta x) - f(x) \sim \Delta x \cdot f'(x)$$

- then $K$ is

$$K_f = \frac{\left| dx \cdot f'(x) \right|}{\left| dx \right| \cdot x} = \frac{\left| f'(x) \cdot x \right|}{\left| f(x) \right|}$$

- so $K_f$ depends on properties of $f$ and value of $x$

- There's a relationship between cond# of problem and cond# of inverse problem
- Inverse problem of $y = f(x)$ is find $x$ s.t. $f(x) = y$, written $x = f^{-1}(y)$
- so

$$\frac{\text{rel. forward err.}}{\text{rel. backward err.}} = \frac{\left| g(y^*) - g(y) \right|}{\left| f(x^*) - f(x) \right|} = \frac{\left| y^* - y \right|}{\left| x^* - x \right|} = \frac{1}{K_f}$$

Example:

- Differentiable $f(x)$, and $g(y)$
  - $g(f(x)) = x$ by def'n
  - using chain rule, $g'(f(x)) \cdot f'(x) = 1$, so $g' = 1/f'$
  - so cond#

$$K_g = \frac{\left| g'(y) \cdot y \right|}{\left| g(y) \right|} = \frac{\left| 1/f'(x) \cdot f(x) \right|}{\left| x \right|} = \frac{1}{K_f}$$
- Lesson:
  - If $K_f$ near 1, both $f$ and $g$ well-conditioned
  - If $K_f$ big or small, either $K_f$ or $K_g$ ill-conditioned

- Side note: Above is "relative cond#". If seeing $x^*$ s.t. $f(x^*) = 0$, use "absolute cond#", defined analogously:

  $K = \frac{|f(x^*) - f(x)|}{|x^* - x|}$

- for differentiable $f$

  $K_{f\_abs} = \frac{|dx f'(x)|}{|dx|} = |f'(x)|$

- Example: $f(x) = \sqrt{x} = x^{1/2}$
  
  $f'(x) = 1/2 * x^{-1/2} = 1/(2f(x))$

  $K_f = \frac{|f'(x) x|}{|f(x)|} = \frac{|x|}{2 f(x) * f(x)} = \frac{1}{2}$

  - inverse problem: find $x$ s.t. $y = \sqrt{x}$, or $x = g(y) = y^2$

    $K_g = 2$

  - Conclusion: both $f$ and $g$ are well-conditioned

- Example: $f(x) = \tan(x)$

  $f'(x) = \sec^2(x) = 1 + \tan^2(x)$

  $K_f = \frac{|x(1+\tan^2(x))|}{|\tan(x)|} = \text{very large near } x = \pi/2$

  - at $x = 1.57079$, $K_f = 2.48275 \times 10^5$ (sensitive!!), so that

    $\tan(1.57079) \approx 1.58058 \times 10^5$, $\tan(1.57078) \approx 6.12490 \times$


\[ 10^4 \]

\[ \text{check:} \]

\[ \frac{(1.58058 \times 10^5 - 6.12490 \times 10^4)}{(6.12490 \times 10^4)} / \left( \frac{(1.57079 - 1.57078)}{1.57078} \right) = K_f \]

- \[ g(y) = \arctan(y), \text{ at } y = 1.58058 \times 10^5 \]

\[ K_g \approx 4.0278 \times 10^{-6} \text{ (insensitive!!)} \]

**Stability and Accuracy**

- An algorithm is *stable* if its results are insensitive to perturbations during computation
  - e.g., truncation, discretization, and rounding errors

- Or, using backward error, algorithm is stable if
  - effect of perturbations during computation is no worse than effect of small amount of data error
  - *however* if problem is ill-conditioned, effect of small data error is really bad!
    - won’t get a good (accurate) solution even with a stable algorithm

- So
  - well-conditioned problem + unstable algorithm => inaccurate solution
  - ill-conditioned problem + stable algorithm => inaccurate solution
  - well-conditioned problem + stable algorithm => accurate solution
Floating Point

- Generally use floating point, which is a *finite precision* system
  - introduced *rounding* errors

- standard is **IEEE 754 (1985)**
  - adherence made numerical code more portable and reliable

- as opposed to fixed point: point is always after the $10^0$ place
  
  1234.567
  1.3
  0.001

- floating point: point can "float"
  
  1.234567 * $10^3$
  1.3 * $10^0$
  1.0 * $10^{-3}$

**General floating point system**

\[
\begin{array}{lllll}
\text{b} & \text{base} & \text{p} & \text{number of digits of precision} & [U,L] & \text{exponent range} \\
\hline
\text{IEEE SP} & 2 & 23(+1)=24 & -126 & 127 & (1+8+23 = 32) \\
\text{IEEE DP} & 2 & 52(+1)=53 & -1022 & 1023 & (1+11+52 = 64) \\
\end{array}
\]

**Floating point number x**

\[
x = \pm ( \underbrace{ d_0 + d_1 + d_2 + \ldots + d_{(p-1)} }_{\text{mantissa}} ) \times b^E
\]

\[
0 \leq d_i \leq b-1, \quad i = 0, \ldots, p-1 \quad \text{(p digits)}
\]

\[
L \leq E \leq U
\]

mantissa: \[d0d1...d(p-1)\]
Example 1 (1):

\[ b = 2 \]
\[ p = 3 \]
\[ L = -1 \]
\[ U = 1 \]

start enumerating possibilities:

\[
\begin{array}{ccc}
+ & m & E \\
+ & 0.00 & -1 & \rightarrow 0 \\
+ & 0.00 & 0 & \rightarrow 0 \\
+ & 0.00 & +1 & \rightarrow 0 \\
+ & 0.01 & -1 & \rightarrow 0.001 \\
+ & 0.01 & 0 & \rightarrow 0.01 \\
+ & 0.01 & +1 & \rightarrow 0.1 \\
+ & 0.10 & -1 & \rightarrow 0.01 \\
+ & 0.10 & 0 & \rightarrow 0.1 \\
+ & 0.10 & +1 & \rightarrow 1.0 \\
\end{array}
\]

duplicates!

In general, number of possibilities
\[ 2 \cdot b^p \cdot (U - L + 1) \]

but

- lots of duplicates
- non-unique representation

**Normalization**
- require the leading digit to be non-zero
- so mantissa, \( m \)
\[ 1 \leq m < b \]
- nice because:
  - representation is now *unique*
  - don't waste digits on any leading 0's
  - for binary base, leading digit must be 1
- so don't need to store it, just assume number is 1.d1d2..dp
- gain an extra bit of precision!