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

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

- rearranging, see that \( K \) acts like "amplification factor"

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

- ill-conditioned \( \rightarrow \) large \( K \)
- well-conditioned \( \rightarrow \) 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.

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

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

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

- then $K$ is

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

- 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 $x = f^{-1}(y) = g(y)$
- so

$$\frac{\text{rel. forward err.}}{\text{rel. backward err.}} = \frac{|g(y^\perp) - g(y)|}{|y^\perp - y|}$$

$$= \frac{|y^\perp - y|}{|x^\perp - x|} \frac{|x^\perp - x|}{|f(x^\perp) - f(x)|} \frac{|f(x^\perp) - f(x)|}{|f(x)|} = 1$$

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#

$$K_g = \frac{|g'(y) y|}{|g(y)|} = \frac{|1/f'(x) f(x)|}{|x|} = \frac{1}{K_f}$$
- 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:

$$K = \frac{\text{abs. forward err.}}{\text{abs. backward err.}} = \frac{|f(x^*) - f(x)|}{|x^* - x|}$$

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

$$\tan(1.57079) \approx 1.58058 * 10^5, \quad \tan(1.57078) \approx 6.12490 *$$
10^4
check:
\[
((1.58058 \cdot 10^5 - 6.12490 \cdot 10^4) / (6.12490 \cdot 10^4)) / ((1.57079
- 1.57078)/1.57078) = K_f \checkmark
\]
- \( g(y) = \arctan(y) \), at \( y = 1.58058 \cdot 10^5 \)

\( K_g \approx 4.0278 \cdot 10^{-6} \) (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 \( \Rightarrow \) inaccurate solution
  - ill-conditioned problem + stable algorithm \( \Rightarrow \) inaccurate solution
  - well-conditioned problem + stable algorithm \( \Rightarrow \) 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
  \[ b \text{ base} \]
  \[ p \text{ number of digits of precision} \]
  \[ [U, L] \text{ exponent range} \]

<table>
<thead>
<tr>
<th></th>
<th>b</th>
<th>p</th>
<th>L</th>
<th>U</th>
<th>field width</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEEE SP</td>
<td>2</td>
<td>23(+1)=24</td>
<td>-126</td>
<td>127</td>
<td>(1+8+23 = 32)</td>
</tr>
<tr>
<td>IEEE DP</td>
<td>2</td>
<td>52(+1)=53</td>
<td>-1022</td>
<td>1023</td>
<td>(1+11+52 = 64)</td>
</tr>
</tbody>
</table>

- Floating point number $x$

  \[
  x = +-( \begin{array}{cccc}
  d_0 & d_1 & d_2 & \ldots & d(p-1) \\
  \end{array} ) \times b^E \\
  \begin{array}{cccc}
  b & b^2 & \ldots & b^{(p-1)} \\
  \end{array}
  \]

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

  \[
  L \leq E \leq U
  \]

  mantissa: \[ d_0d_1...d(p-1) \]
exponent: E

Example 1 (1):

\[
\begin{align*}
10^{-2} = 2^p \times (10^{L-U})
\end{align*}
\]

\[
\begin{align*}
b &= 2 \\
p &= 3 \\
L &= -1 \\
U &= 1
\end{align*}
\]

start enumerating possibilities:

\[
\begin{align*}
&+-0.00 \quad 0 \rightarrow 0 \\
&+-0.00 \quad 0 \rightarrow 0 \\
&+-0.00 \quad +1 \rightarrow 0 \\
&+-0.01 \quad -1 \rightarrow 0.001 \\
&+-0.01 \quad 0 \rightarrow 0.01 \\
&+-0.01 \quad +1 \rightarrow 0.1 \\
&+-0.10 \quad -1 \rightarrow 0.01 \\
&+-0.10 \quad 0 \rightarrow 0.1 \\
&+-0.10 \quad +1 \rightarrow 1.0
\end{align*}
\]

duplicates!

In general, number of possibilities

\[
2 \times b^p \times (U - L + 1)
\]

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!