Significant digits & Relative error revisited:

- ② If $E_{rel} \lesssim 0.5 \times 10^{-n}$ then x_* is good to n sig. figures
- If x_* is good to n sig. figures then $E_{rel} \lesssim 5 \times 10^{-n}$. ①

Example 2.10 (Root-finding). Suppose that we are given a smooth function $f : \mathbb{R} \to \mathbb{R}$ and want to find roots x with $f(x) = 0$. By Taylor's theorem, $f(x + \varepsilon) \approx f(x) + \varepsilon f'(x)$ when $|\varepsilon|$ is small. Thus, an approximation of the condition number for finding the root x is given by

$$C = \frac{\text{forward error}}{\text{backward error}} = \frac{|(x+\varepsilon) - x|}{|f(x+\varepsilon) - f(x)|} \approx \frac{|\varepsilon|}{|\varepsilon f'(x)|} = \frac{1}{|f'(x)|}.$$

This approximation generalizes the one in Example 2.9. If we do not know x, we cannot evaluate $f'(x)$, but if we can examine the form of f and *bound* $|f'|$ near x, we have an idea of the worst-case situation.

Trying to solve $f(x) = 0$ for x. Maybe use Newton's method... Whatever method, I get an approximation x_*. Plug it in to check how good it is. $f(x_*) = r$ where r is the residual and hopefully close to zero... Given C and the backwards error $\longrightarrow$ solve for the forward error $|x_* - x|$

Try to estimate c...

derivative that might be possible to estimate

$$r = f(x_*) = f(x_*) - 0 = f(x_*) - f(x) = f'(\xi)\underbrace{(x_* - x)}_{\text{forward error...}}$$

(MVT) where ξ is between x and x_*

$$c = \frac{\text{forward error}}{\text{backwards error}} = \frac{x_* - x}{f(x_*) - f(x)} = \frac{x_* - x}{f'(\xi)(x_* - x)} = \boxed{\frac{1}{f'(\xi)}}$$

forward error $\approx c$ (backward error)

to find an upper bound on forward error need an upper bound on c.

Since $c = \frac{1}{f'(\xi)}$ then I need a lower bound on $f'(\xi)$.

Example: finding $\sqrt{2}$. $f(x) = x^2 - 2$ $f'(x) = 2x$

$x_* = 1.4$

$r = (1.4)^2 - 2 = -0.04$ I know that $\xi \geq 1.4$

$|x_* - x| \leq c|r| < \frac{1}{2.8}(0.04)$ Thus $f'(\xi) \geq 2(1.4) = 2.8$

```
julia> fe=sqrt(2)-1.4
0.014213562373095234

julia> 1/2.8*(0.04)
0.014285714285714287
```

Problems with small condition numbers are well-conditioned, and thus backward error can be used safely to judge success of approximate solution techniques. Contrastingly, much smaller backward error is needed to justify the quality of a candidate solution to a problem with a large condition number.

Big condition numbers are bad while small condition numbers are good and allow the forward error to be easily bounded in terms of the backwards error... in our case c was less than one... that's good and we obtained a precise estimate on the forward error...

The condition number actually tells something about how accurately a problem can be solved no matter what method is used...be it Newton's method or some of other method...

Since residual r can be calculated, at best, to 15 digits of precision, one can't tell the difference numerically between two approximations of x for which the residuals agree to 15 digits.

Thus r is at best something like 1e-15.

But if c is big, for example, 1e8, then the bound on the forward error becomes...

$$\text{forward error} \leq c\,(\text{backward error}) \leq 10^{-7}$$

In this case the forward error can't be computed to better than 7 digits accuracy, since the condition number is so large...

Because of Rounding errors

$$a+(b+c) \neq (a+b)+c$$

in general.., Better to add the small #'s first...

Consider the sum

$$S_n = \sum_{k=1}^{n} \frac{1}{k}$$

Question: what is $\lim_{n\to\infty} S_n = \boxed{\infty}$

```
julia> n=1000000
1000000

julia> sf=0
       for k=1:n
           sf=sf+1/k
       end

julia> sf
14.392726722864989
```

wrong

```
julia> sr=0
       for k=n:-1:1
           sr=sr+1/k
       end

julia> sr
14.392726722865772
```

wrong

```
julia> sb=big(0.0)
       for k=n:-1:1
           sb=sb+1/k
       end

julia> sb
14.392726722865723577218399385161534675958705520315561443567276000976 5625
```

so many digits that the first 50 must be correct

for comparison

adding the smallest first yields a more accurate answer

Difficulty: In general don't want to spend time sorting the numbers before adding them up... and that may not, in general, be possible.

function KAHAN-SUM($\vec{x}$)

- $s, c \leftarrow 0$ ▷ Current total and compensation
- **for** $i \leftarrow 1, 2, \ldots, n$
  - $v \leftarrow x_i + c$ ▷ Try to add x_i and compensation c to the sum
  - $s_{\text{next}} \leftarrow s + v$ ▷ Compute the summation result of this iteration
  - $c \leftarrow v - (s_{\text{next}} - s)$ ▷ Compute compensation using the Kahan error estimate
  - $s \leftarrow s_{\text{next}}$ ▷ Update sum
- **return** s

Clever way of adding numbers so that one gets an accurate answer no matter what order is used...

```
julia> s=0; c=0
       for k=1:n
           v=1/k+c
           snext=s+v
           c=v-(snext-s)
           s=snext
       end

julia> s
14.392726722865724
```

This is the important step which determines how much rounding error is made with each addition in the sum...

last digit is correctly rounded

```
julia> sb=big(0.0)
       for k=n:-1:1
           sb=sb+1/k
       end

julia> sb
14.39272672286572357721839938516153467595870552031556144356727600009765625
```

for comparison

What does the vbl. c do?

print it out ...

```
julia> s=0; c=0
       for k=1:n
           println(c)
           v=1/k+c
           snext=s+v
           c=v-(snext-s)
           s=snext
       end
```

Printing a million lines takes too long...and all that scrolling...

```
-6.453442381176844e-16
-6.501824903124009e-16
-2.8199420879990167e-17
-8.431840295419768e-16
1.6891192220966356e-16
-8.345104121620928e-16
-5.918591896962588e-16
6.045206381918161e-16
6.846940007153302e-16
-6.461370609563144e-16
-1.3129349495620557e-16
1.5561011680598202e-16
-8.391386001858903e-17
6.267806640456591e-16
2.1039281970260115e-16
1.411089127489884e-16
1.1553529400548657e-16
-1.7095496567844093e-16
7.521449283710846e-16
8.014015883198167e-16
^C-3.3149142610565396e-16ERROR: InterruptException:
Stacktrace:
 [1] unsafe_write
```

Press ctrl-c to stop the calculation

Note that sometimes ctrl-c doesn't work to stop the calculation, especially when no printing is involved, and one has to resort to more drastic measures...

That is apparently one of the tradeoffs for being both free and just-in-time compiled...