Error Analysis

- Introduction
- Representation of Numbers in a Computer
- Numerical Computations: Stability and Error Analysis
- Application: Numerical Differentiation

Contents

Objectives

In this lecture we will learn about

- the fact that many computer programs store numbers with finite precision only
- the IEEE standard for storing floating point numbers
- applications, including numerical differentiation

Contents

Scientific Computing

Computers or calculators typically store numbers with finite precision:

- Example 1: $8 + 8 == 16$?
- Example 2: $(\sqrt{5})^2 == 5$?
- Example 3: $1.1 + 0.1 == 1.2$?

Let's try this with JULIA:

julia>$(1.1 + 0.1) == 1.2$	julia> $1.1 + 0.1$
`false`	1.2000000000000002

Problem: numerical error: $\approx 2 * 10^{-16}$.

Scientific Computing

Computers or calculators typically store numbers with finite precision:

- Example 1: $8 + 8 == 16$?
- Example 2: $(\sqrt{5})^2 == 5$?
- Example 3: $1.1 + 0.1 == 1.2$?

Let's try this with JULIA:

julia>$(1.1 + 0.1) == 1.2$	julia> $1.1 + 0.1$
`false`	1.2000000000000002

Problem: numerical error: $\approx 2 * 10^{-16}$.

Scientific Computing

Computers or calculators typically store numbers with finite precision:

- Example 1: $8 + 8 == 16$?
- Example 2: $(\sqrt{5})^2 == 5$?
- Example 3: $1.1 + 0.1 == 1.2$?

Let's try this with JULIA:

julia>$(1.1 + 0.1) == 1.2$	julia> $1.1 + 0.1$
`false`	1.2000000000000002

Problem: numerical error: $\approx 2 * 10^{-16}$.

Floating Point Numbers

IEEE standard for double-precision floating point numbers:

$$x = \pm(1+m)\cdot 2^e \quad \text{with} \quad m = \sum_{i=1}^{52} m_i 2^{-i} \quad \text{and} \quad e = \sum_{i=0}^{10} c_i 2^i - \bar{c}\,,$$

Names: m = mantissa, e = exponent.

Storage requirement:

- 1 bit to store the sign.
- 11 bits to store $c_{10}, \ldots, c_0 \in \{0,1\}$; offset $\bar{c} = 1023$.
- 52 bits to store $m_1, \ldots m_{52} \in \{0,1\}$.

In total: $(1+11+52)$ bits $= 64$ bits $= 8$ bytes.

Floating Point Numbers

IEEE standard for double-precision floating point numbers:

$$x = \pm(1 + m) \cdot 2^e \quad \text{with} \quad m = \sum_{i=1}^{52} m_i 2^{-i} \quad \text{and} \quad e = \sum_{i=0}^{10} c_i 2^i - \bar{c}\,,$$

Names: $m =$ mantissa, $e =$ exponent.

Storage requirement:

- 1 bit to store the sign.
- 11 bits to store $c_{10}, \ldots, c_0 \in \{0, 1\}$; offset $\bar{c} = 1023$.
- 52 bits to store $m_1, \ldots m_{52} \in \{0, 1\}$.

In total: $(1 + 11 + 52)$ bits $= 64$ bits $= 8$ bytes.

Floating Point Numbers

IEEE standard for double-precision floating point numbers:

$$x = \pm(1+m)\cdot 2^e \quad \text{with} \quad m = \sum_{i=1}^{52} m_i 2^{-i} \quad \text{and} \quad e = \sum_{i=0}^{10} c_i 2^i - \bar{c}\,,$$

Names: $m =$ mantissa, $e =$ exponent.

Storage requirement:

- 1 bit to store the sign.
- 11 bits to store $c_{10}, \ldots, c_0 \in \{0,1\}$; offset $\bar{c} = 1023$.
- 52 bits to store $m_1, \ldots m_{52} \in \{0,1\}$.

In total: $(1+11+52)$ bits $= 64$ bits $= 8$ bytes.

A closer look at the example

julia>bits(1.1)

"0011111111110001100110011001100110011001100110011001100110011010"

julia>bits(0.1)

"0011111110111001100110011001100110011001100110011001100110011010"

julia>bits(1.2)

"0011111111110011001100110011001100110011001100110011001100110011"

julia>bits(1.1+0.1)

"0011111111110011001100110011001100110011001100110011001100110100"

A closer look at the example

julia>`bits(1.1)`

"0011111111110001100110011001100110011001100110011001100110011010"

julia>`bits(0.1)`

"0011111110111001100110011001100110011001100110011001100110011010"

julia>`bits(1.2)`

"0011111111110011001100110011001100110011001100110011001100110011"

julia>`bits(1.1+0.1)`

"0011111111110011001100110011001100110011001100110011001100110100"

A closer look at the example

julia>bits(1.1)

"0011111111110001100110011001100110011001100110011001100110011010"

julia>bits(0.1)

"0011111110111001100110011001100110011001100110011001100110011010"

julia>bits(1.2)

"0011111111110011001100110011001100110011001100110011001100110011"

julia>bits(1.1+0.1)

"0011111111110011001100110011001100110011001100110011001100110100"

A closer look at the example

julia>`bits(1.1)`

"0011111111110001100110011001100110011001100110011001100110011010"

julia>`bits(0.1)`

"0011111110111001100110011001100110011001100110011001100110011010"

julia>`bits(1.2)`

"0011111111110011001100110011001100110011001100110011001100110011"

julia>`bits(1.1+0.1)`

"0011111111110011001100110011001100110011001100110011001100110100"

Floating Point Numbers

- Numbers between 1 and $1 + 2^{-52}$ cannot be represented.
- The (relative) rounding $\mathrm{eps} = 2^{-52}$ is called *machine precision*.
- The absolute rounding error $\mathrm{eps} * 2^e$ depends on exponent e.

  (if we work with larger numbers, we get larger rounding errors)

Important to remember: $\mathrm{eps} = 2^{-52} \approx 2 * 10^{-16}$.

Floating Point Numbers

- Numbers between 1 and $1 + 2^{-52}$ cannot be represented.
- The (relative) rounding $\mathrm{eps} = 2^{-52}$ is called *machine precision*.
- The absolute rounding error $\mathrm{eps} * 2^e$ depends on exponent e.

  (if we work with larger numbers, we get larger rounding errors)

Important to remember: $\mathrm{eps} = 2^{-52} \approx 2 * 10^{-16}$.

Contents

Numerical function evaluation

Let us evaluate the function

$$\Phi(x) = \sin(10^8 x)$$

at $x = \pi$. The exact solution is $\Phi(\pi) = 0$.

```
julia>sin(10^8 pi)
-3.9082928156687315e-8
```

Caution: The function Φ is ill-conditioned, i.e., the evaluation error is much larger than $\mathrm{eps} \approx 2 * 10^{-16}$!

Numerical function evaluation

Let us evaluate the function

$$\Phi(x) = \sin(10^8 x)$$

at $x = \pi$. The exact solution is $\Phi(\pi) = 0$.

julia>$\sin(10\hat{}8\,\mathrm{pi})$

$-3.9082928156687315e-8$

Caution: The function Φ is ill-conditioned, i.e., the evaluation error is much larger than $\mathrm{eps} \approx 2 * 10^{-16}$!

Numerical function evaluation

Let us evaluate the function

$$\Phi(x) = \sin(10^8 x)$$

at $x = \pi$. The exact solution is $\Phi(\pi) = 0$.

```
julia>sin(10^8 pi)
 -3.9082928156687315e-8
```

Caution: The function Φ is ill-conditioned, i.e., the evaluation error is much larger than $\mathrm{eps} \approx 2 * 10^{-16}$!

Storing Numbers in a Computer

There exists a variety of ways to represent numbers:

- Floating point numbers be it 64bit ("double precision") or 32bit ("single precision").
- Integers are often stored differently. Remark: julia>`bits(3)` is not the same as julia>`bits(3.)`!!!
- Arbitrary precision arithmetics are an alternative (not our focus).
- Verified arithmetics store intervals rather than single numbers.

Storing Numbers in a Computer

There exists a variety of ways to represent numbers:

- Floating point numbers be it 64bit ("double precision") or 32bit ("single precision").
- Integers are often stored differently. Remark: julia>`bits(3)` is not the same as julia>`bits(3.)`!!!
- Arbitrary precision arithmetics are an alternative (not our focus).
- Verified arithmetics store intervals rather than single numbers.

Storing Numbers in a Computer

There exists a variety of ways to represent numbers:

- Floating point numbers be it 64bit ("double precision") or 32bit ("single precision").
- Integers are often stored differently. Remark: julia>`bits(3)` is not the same as julia>`bits(3.)`!!!
- Arbitrary precision arithmetics are an alternative (not our focus).
- Verified arithmetics store intervals rather than single numbers.

Storing Numbers in a Computer

There exists a variety of ways to represent numbers:

- Floating point numbers be it 64bit ("double precision") or 32bit ("single precision").
- Integers are often stored differently. Remark: julia>`bits(3)` is not the same as julia>`bits(3.)`!!!
- Arbitrary precision arithmetics are an alternative (not our focus).
- Verified arithmetics store intervals rather than single numbers.

Contents

Finite Differences

The derivative of a twice continuously differentiable function $f : \mathbb{R} \to \mathbb{R}$ can be approximated by finite differences:

$$\frac{\partial f(x)}{\partial x} \approx \frac{f(x+h) - f(x)}{h}$$

- The mathematical approximation error, given by

$$\left| \frac{f(x+h) - f(x)}{h} - \frac{\partial f}{\partial x}(x) \right| \approx \frac{h}{2} \left| \frac{\partial^2 f}{\partial x^2}(x) \right| = \mathbf{O}(h)\,,$$

  tends to zero for $h \to 0$.

- The numerical error is approximately

$$\mathbf{O}\left(\frac{\text{eps}}{h}\right)$$

- If f is well conditioned, we choose

$$h \approx \text{argmin}_h \left(h + \frac{\text{eps}}{h} \right) = \sqrt{\text{eps}}\,.$$

Finite Differences

The derivative of a twice continuously differentiable function $f : \mathbb{R} \to \mathbb{R}$ can be approximated by finite differences:

$$\frac{\partial f(x)}{\partial x} \approx \frac{f(x+h) - f(x)}{h}$$

- The mathematical approximation error, given by

$$\left| \frac{f(x+h) - f(x)}{h} - \frac{\partial f}{\partial x}(x) \right| \approx \frac{h}{2} \left| \frac{\partial^2 f}{\partial x^2}(x) \right| = \mathbf{O}(h)\,,$$

  tends to zero for $h \to 0$.

- The numerical error is approximately

$$\mathbf{O}\left(\frac{\mathrm{eps}}{h}\right)$$

- If f is well conditioned, we choose

$$h \approx \text{argmin}_h \left(h + \frac{\mathrm{eps}}{h}\right) = \sqrt{\mathrm{eps}}\,.$$

Finite Differences

The derivative of a twice continuously differentiable function $f : \mathbb{R} \to \mathbb{R}$ can be approximated by finite differences:

$$\frac{\partial f(x)}{\partial x} \approx \frac{f(x+h) - f(x)}{h}$$

- The mathematical approximation error, given by

  $$\left| \frac{f(x+h) - f(x)}{h} - \frac{\partial f}{\partial x}(x) \right| \approx \frac{h}{2} \left| \frac{\partial^2 f}{\partial x^2}(x) \right| = \mathbf{O}(h)\,,$$

  tends to zero for $h \to 0$.
- The numerical error is approximately

  $$\mathbf{O}\left(\frac{\mathrm{eps}}{h}\right)$$

- If f is well conditioned, we choose

  $$h \approx \text{argmin}_h \left(h + \frac{\mathrm{eps}}{h} \right) = \sqrt{\mathrm{eps}}\,.$$

Finite Differences

The derivative of a twice continuously differentiable function $f : \mathbb{R} \to \mathbb{R}$ can be approximated by finite differences:

$$\frac{\partial f(x)}{\partial x} \approx \frac{f(x+h) - f(x)}{h}$$

- The mathematical approximation error, given by

  $$\left| \frac{f(x+h) - f(x)}{h} - \frac{\partial f}{\partial x}(x) \right| \approx \frac{h}{2} \left| \frac{\partial^2 f}{\partial x^2}(x) \right| = \mathbf{O}(h) \, ,$$

  tends to zero for $h \to 0$.

- The numerical error is approximately

  $$\mathbf{O}\left(\frac{\mathrm{eps}}{h} \right)$$

- If f is well conditioned, we choose

  $$h \; \approx \; \mathsf{argmin}_h \; \left(h + \frac{\mathrm{eps}}{h} \right) \; = \; \sqrt{\mathrm{eps}} \; .$$

Central Differences

In order to reduce the mathematical approximation error, we can use central differences

$$\frac{\partial f(x)}{\partial x} \approx \frac{f(x+h) - f(x-h)}{2h}$$

to approximate the derivative of f.

- The mathematical approximation error is now

$$\left| \frac{f(x+h) - f(x-h)}{2h} - \frac{\partial f}{\partial x}(x) \right| \leq \mathbf{O}(h^2).$$

- The numerical error is still in the order of

$$\frac{\text{eps}}{h} = \mathbf{O}\left(\frac{\text{eps}}{h}\right)$$

- In practice, if f is well conditioned, we choose $h \approx \sqrt[3]{\text{eps}}$.

Central Differences

In order to reduce the mathematical approximation error, we can use central differences

$$\frac{\partial f(x)}{\partial x} \approx \frac{f(x+h) - f(x-h)}{2h}$$

to approximate the derivative of f.

- The mathematical approximation error is now

$$\left| \frac{f(x+h) - f(x-h)}{2h} - \frac{\partial f}{\partial x}(x) \right| \leq \mathbf{O}(h^2)\,.$$

- The numerical error is still in the order of

$$\frac{\mathrm{eps}}{h} = \mathbf{O}\left(\frac{\mathrm{eps}}{h}\right)$$

- In practice, if f is well conditioned, we choose $h \approx \sqrt[3]{\mathrm{eps}}$.

Central Differences

In order to reduce the mathematical approximation error, we can use central differences

$$\frac{\partial f(x)}{\partial x} \approx \frac{f(x+h) - f(x-h)}{2h}$$

to approximate the derivative of f.

- The mathematical approximation error is now

$$\left| \frac{f(x+h) - f(x-h)}{2h} - \frac{\partial f}{\partial x}(x) \right| \leq \mathbf{O}(h^2)\,.$$

- The numerical error is still in the order of

$$\frac{\mathrm{eps}}{h} = \mathbf{O}\left(\frac{\mathrm{eps}}{h}\right)$$

- In practice, if f is well conditioned, we choose $h \approx \sqrt[3]{\mathrm{eps}}$.

Central Differences

In order to reduce the mathematical approximation error, we can use central differences

$$\frac{\partial f(x)}{\partial x} \approx \frac{f(x+h) - f(x-h)}{2h}$$

to approximate the derivative of f.

- The mathematical approximation error is now

$$\left| \frac{f(x+h) - f(x-h)}{2h} - \frac{\partial f}{\partial x}(x) \right| \leq \mathbf{O}(h^2)\,.$$

- The numerical error is still in the order of

$$\frac{\mathrm{eps}}{h} = \mathbf{O}\left(\frac{\mathrm{eps}}{h}\right)$$

- In practice, if f is well conditioned, we choose $h \approx \sqrt[3]{\mathrm{eps}}$.

Summary

- Programs often store numbers with finite precision only.
- IEEE double precision floating point numbers: $\mathrm{eps} \approx 2 * 10^{-16}$.
- Error and stability analysis is important!
- Application: numerical differentiation.