Computational Physics (PHYS6350)

Lecture 2: Data Visualization, Machine Precision

January 19, 2023

- Data visualization (plotting with matplotlib as an example)
- Accuracy of integer and floating-point number representation

Instructor: Volodymyr Vovchenko (vvovchenko@uh.edu)

Course materials

Apart from Teams, course materials will be maintained and updated on **GitHub**

https://github.com/vlvovch/PHYS6350-ComputationalPhysics

Data visualization

- Line plots
- Scatter plots
- Contour/density plots (2D data)

References: Chapter 3 of *Computational Physics* by Mark Newman

Matplotlib documentation

Plotting the data

Computer programs produce numerical data

Numbers alone do not always make it easy to understand the structure of behavior of the system

Consider a function $y = \sin(x)$

Let us calculate it for 10 equidistant points in the interval $\mathrm{x} = 0 ... 10$

Plotting the data

Computer programs produce numerical data

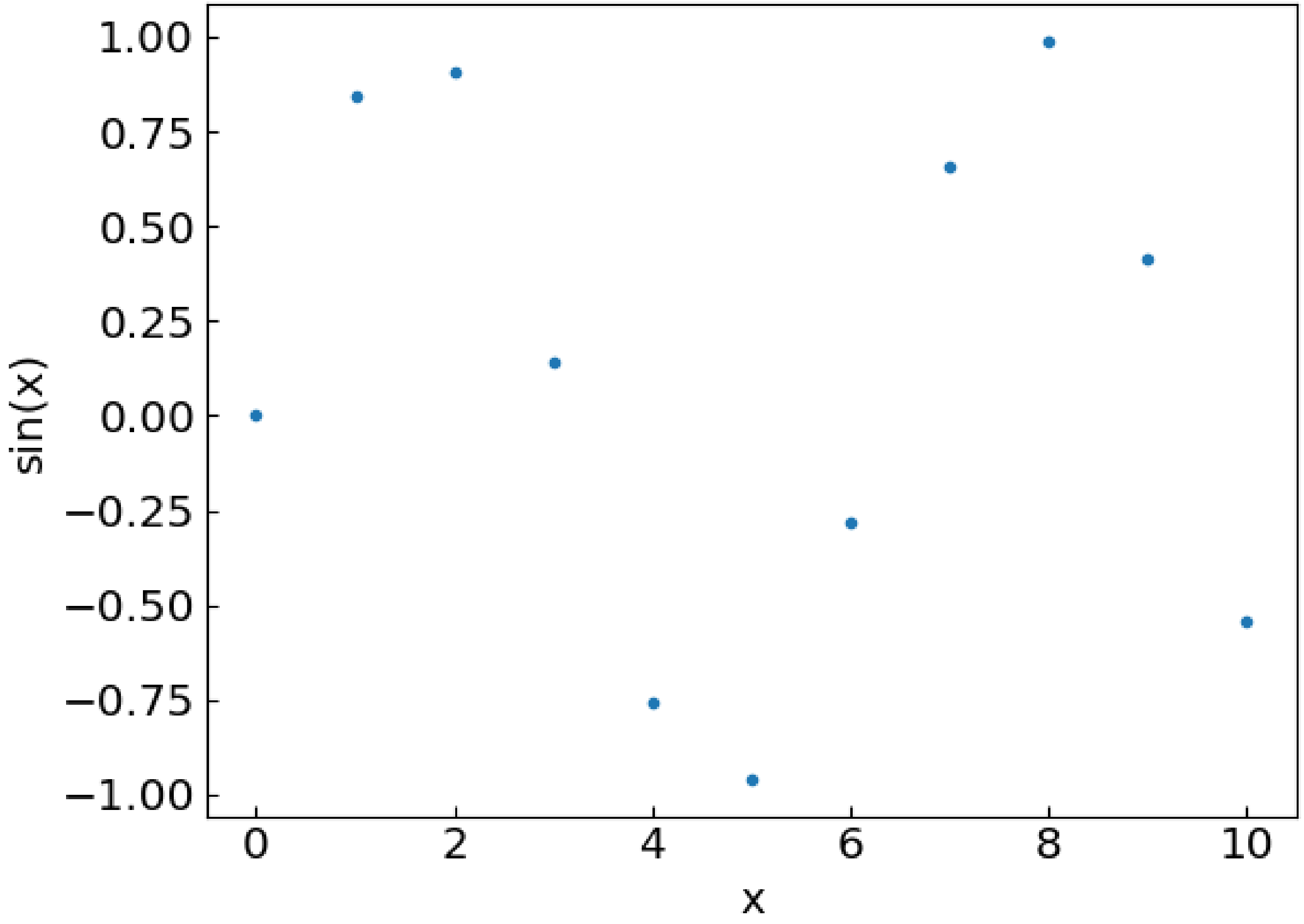
Numbers alone do not always make it easy to understand the structure of behavior of the system

Consider a function $y = \sin(x)$

Let us calculate it for 10 equidistant points in the interval $x = 0 \ldots 10$

x	sin(x)
0	0.
1	0.841471
2	0.9092974
3	0.14112
4	-0.7568025
5	-0.9589243
6	-0.2794155
7	0.6569866
8	0.9893582
9	0.4121185
10	-0.5440211

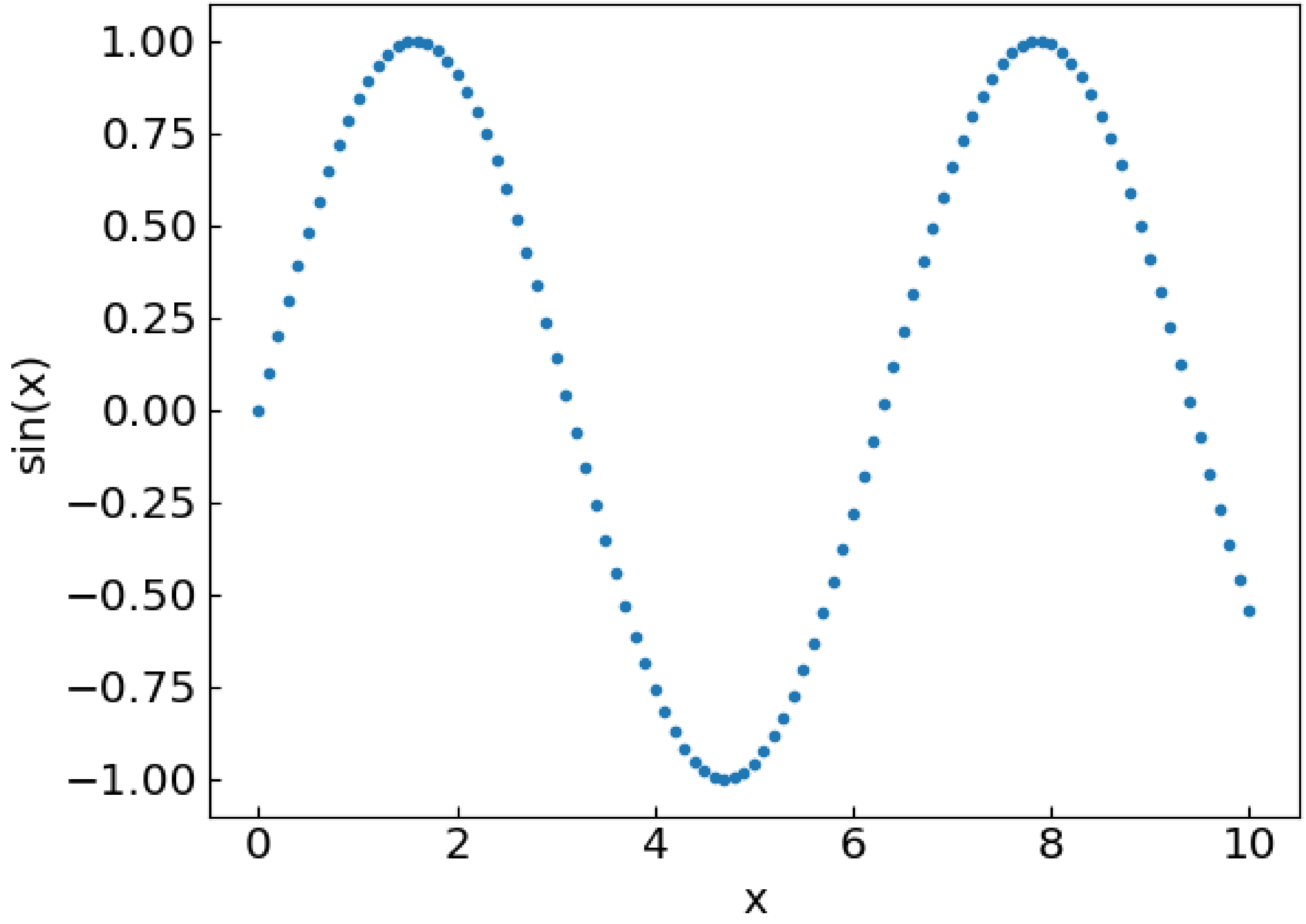
Putting it on a graph



x	sin(x)
0	0.
1	0.841471
2	0.9092974
3	0.14112
4	-0.7568025
5	-0.9589243
6	-0.2794155
7	0.6569866
8	0.9893582
9	0.4121185
10	-0.5440211

Let us add more points...

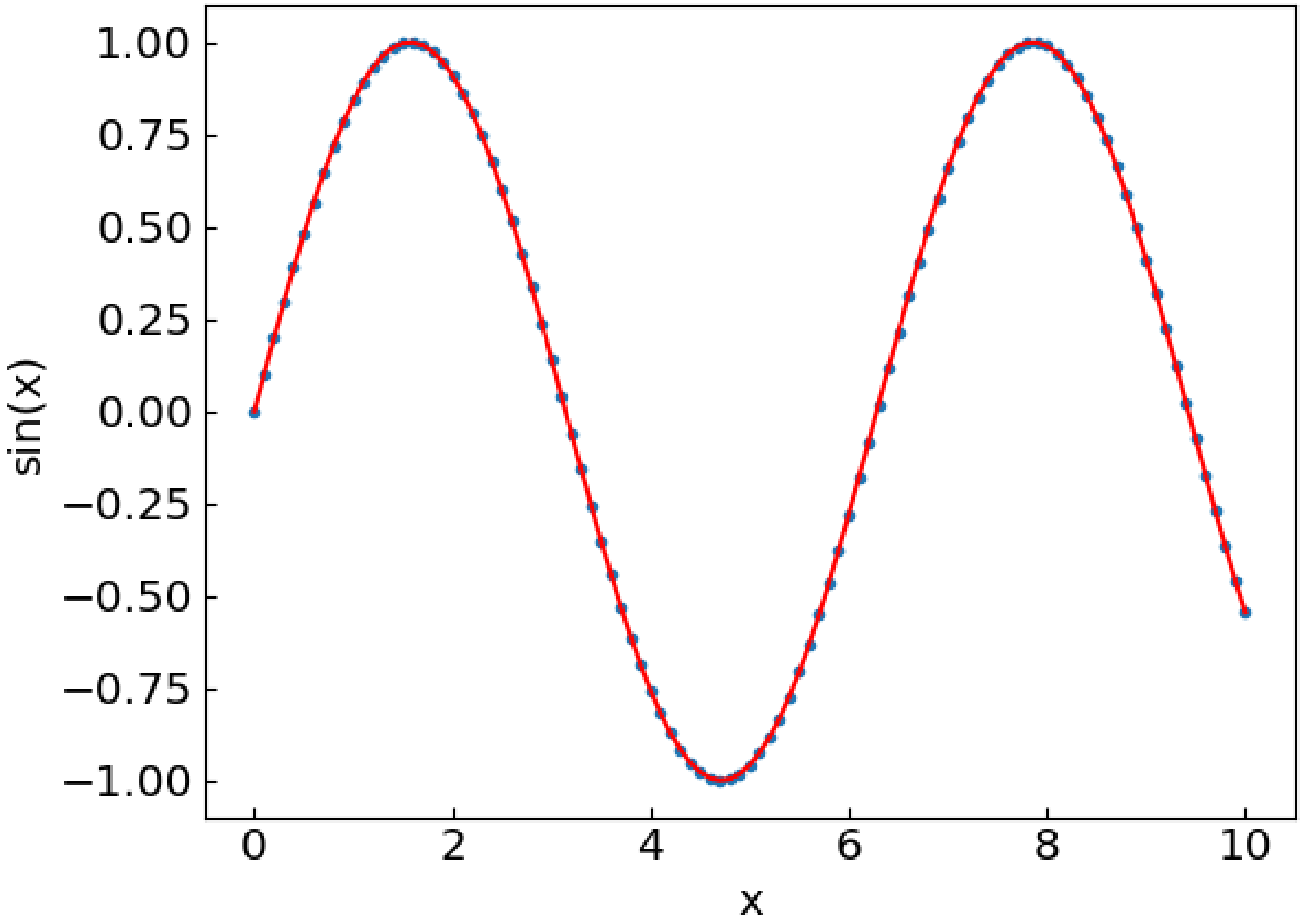
Putting it on a graph



x	sin(x)
0.	0.
0.1	0.09983342
0.2	0.1986693
0.3	0.2955202
0.4	0.3894183
0.5	0.4794255
0.6	0.5646425
0.7	0.6442177
0.8	0.7173561
0.9	0.7833269
1.	0.841471
	⋮
9.9	-0.4575359
10.	-0.5440211

Now we have enough points to join them by a smooth line

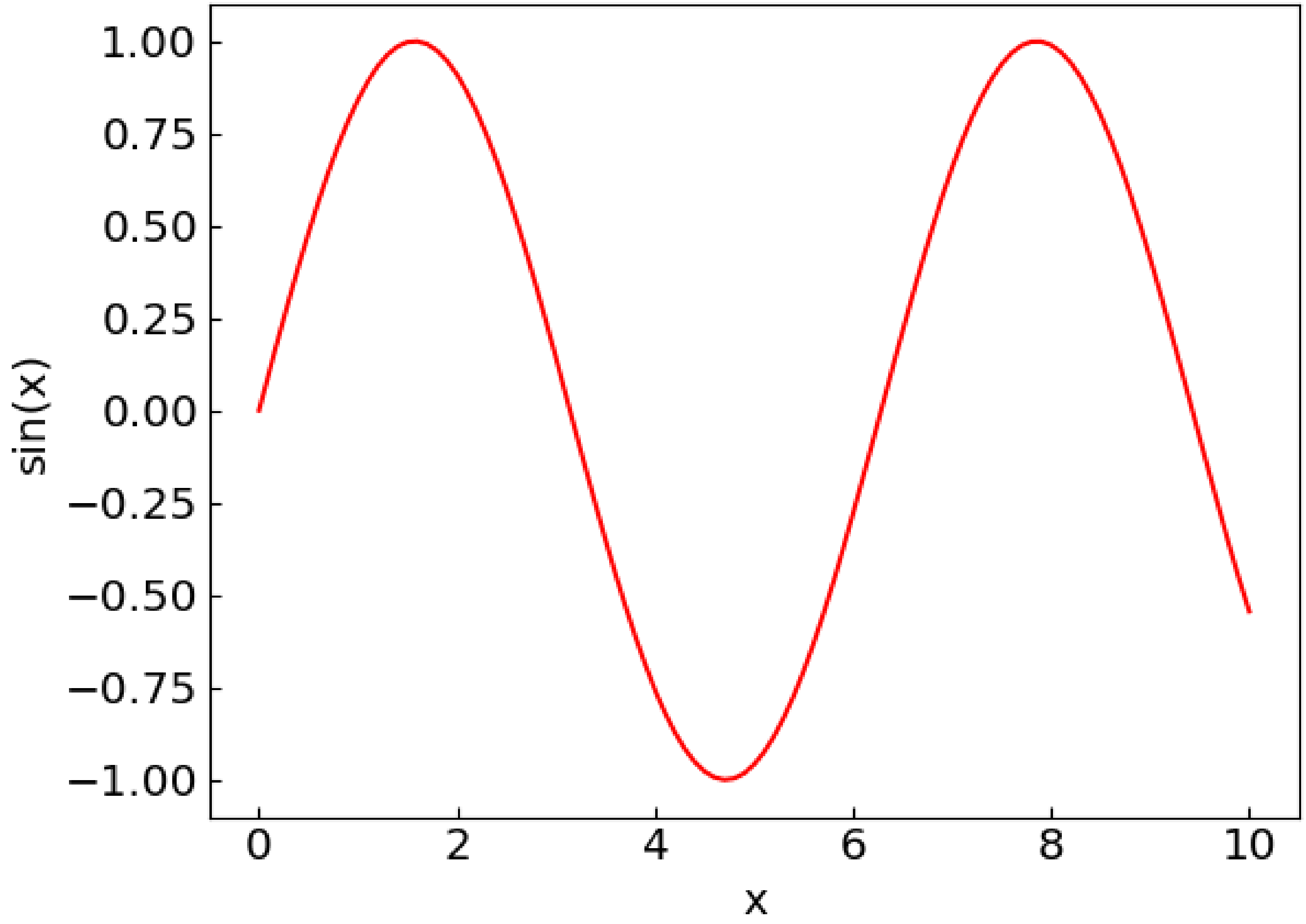
Putting it on a graph



x	sin(x)
0.	0.
0.1	0.09983342
0.2	0.1986693
0.3	0.2955202
0.4	0.3894183
0.5	0.4794255
0.6	0.5646425
0.7	0.6442177
0.8	0.7173561
0.9	0.7833269
1.	0.841471
	⋮
9.9	-0.4575359
10.	-0.5440211

Now we have enough points to join them by a smooth line

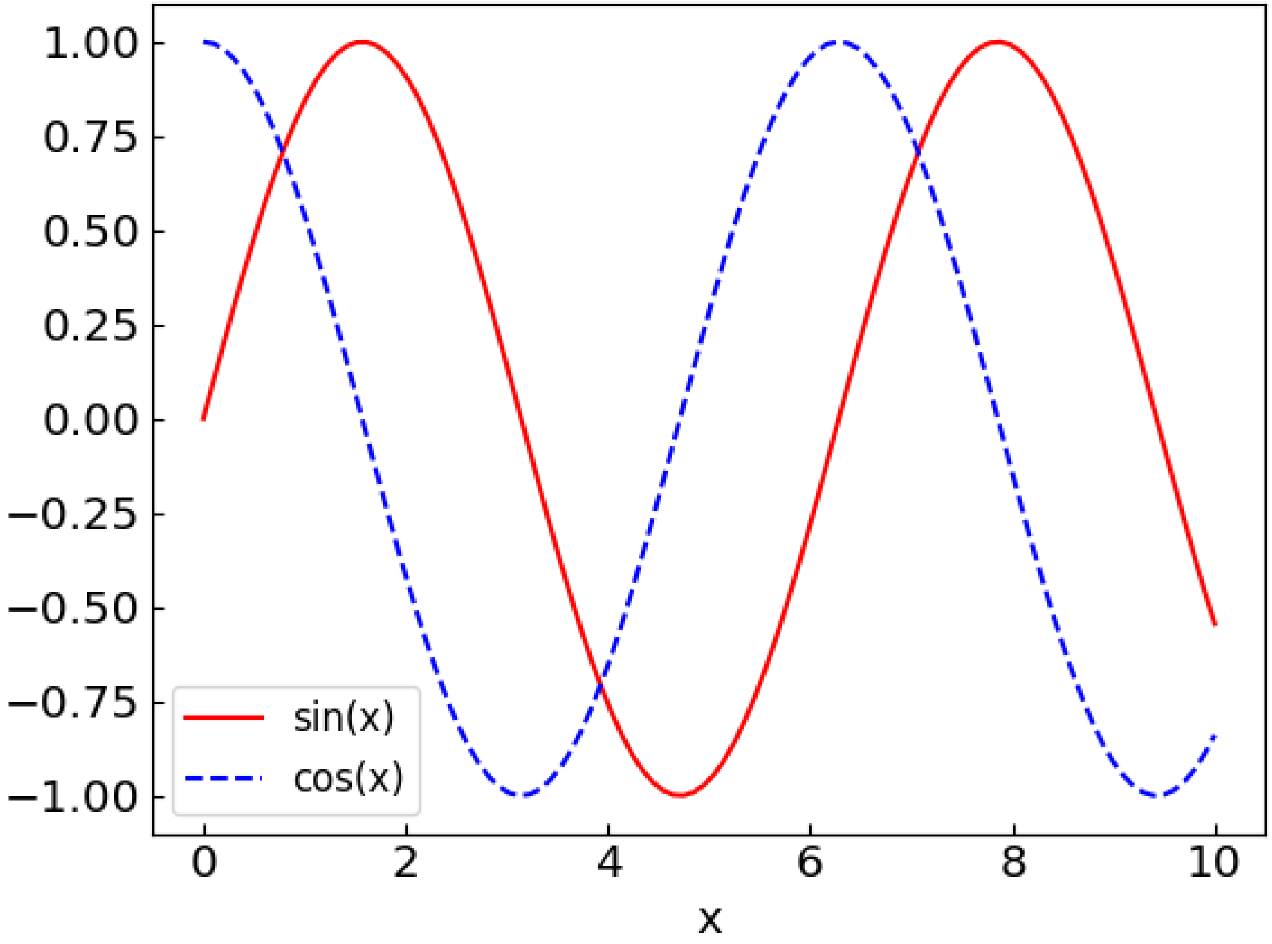
Putting it on a graph



x	sin(x)
0.	0.
0.1	0.09983342
0.2	0.1986693
0.3	0.2955202
0.4	0.3894183
0.5	0.4794255
0.6	0.5646425
0.7	0.6442177
0.8	0.7173561
0.9	0.7833269
1.	0.841471
	⋮
9.9	−0.4575359
10.	−0.5440211

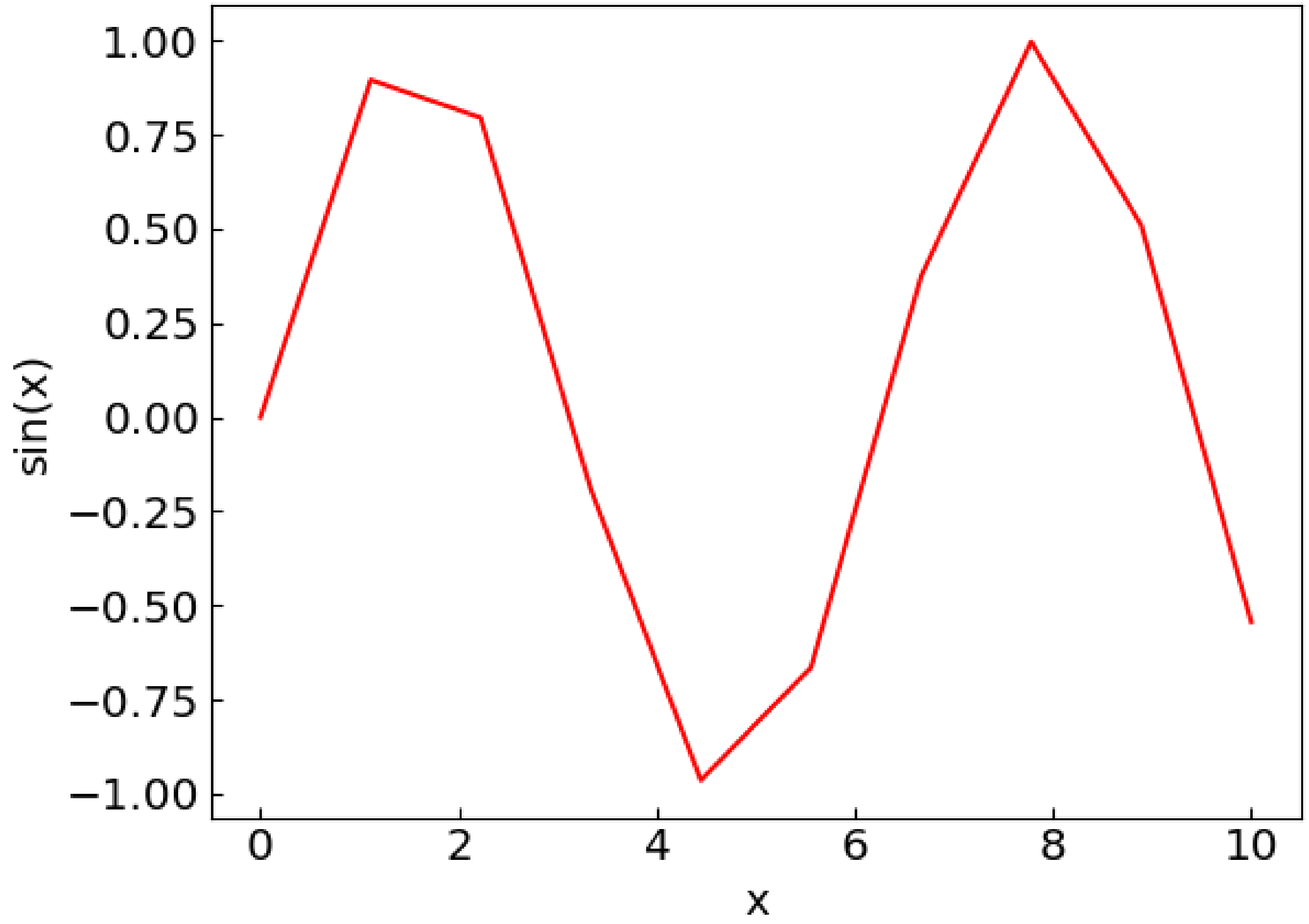
Now we have enough points to join them by a smooth line

Plot multiple lines to compare functions, profiles, etc.



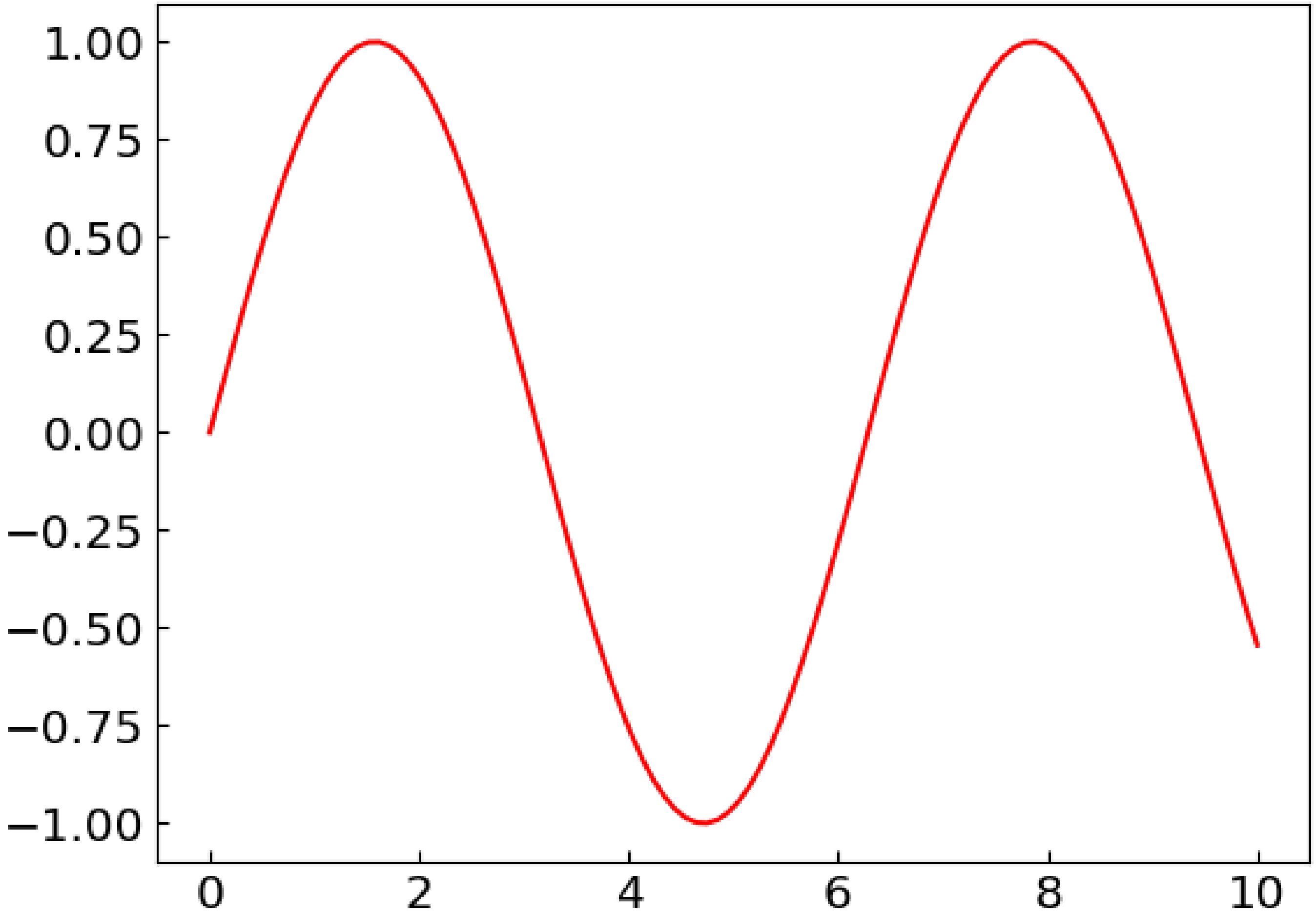
Things to avoid

Insufficient number of data points



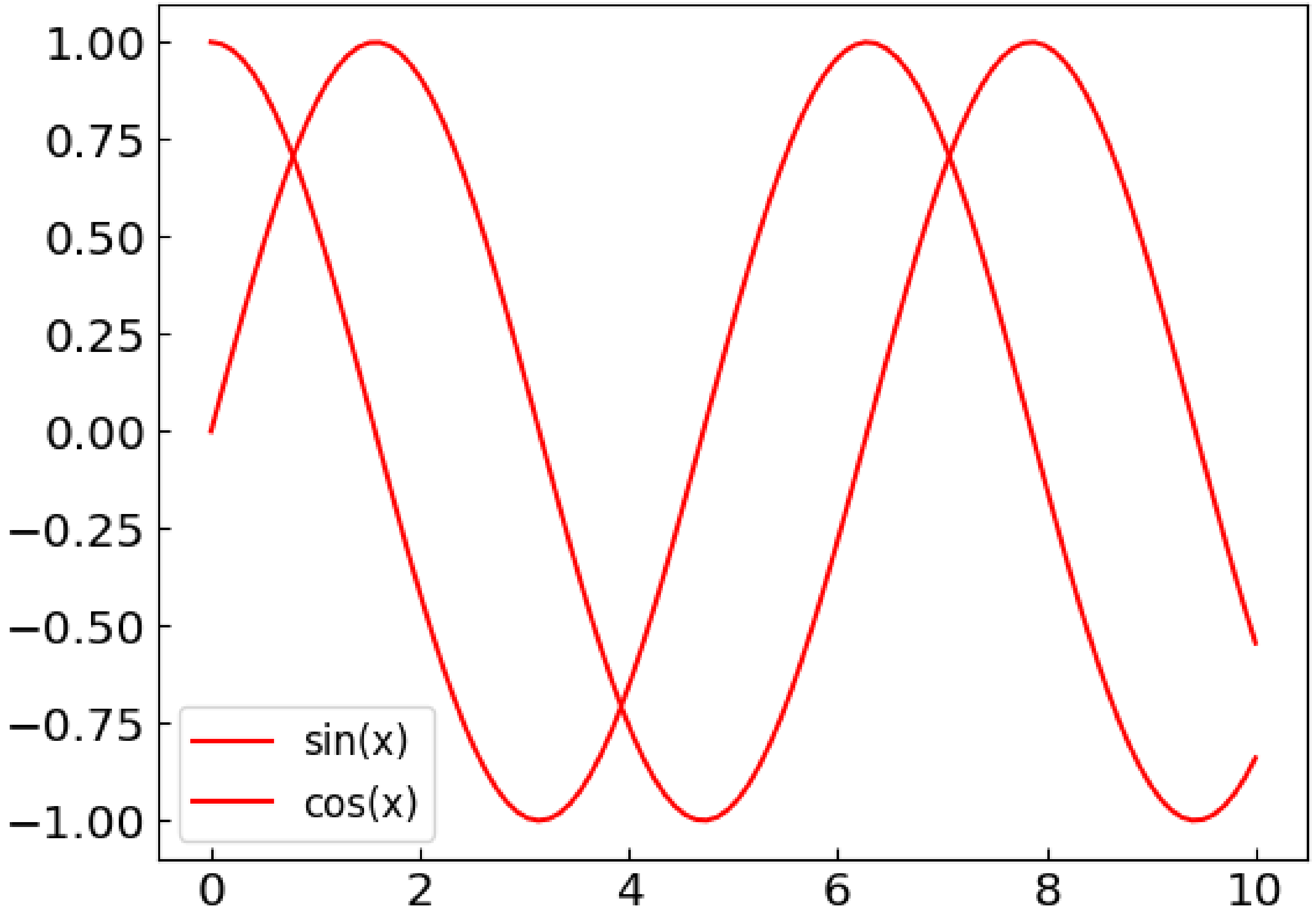
Things to avoid

Unlabeled axes



Things to avoid

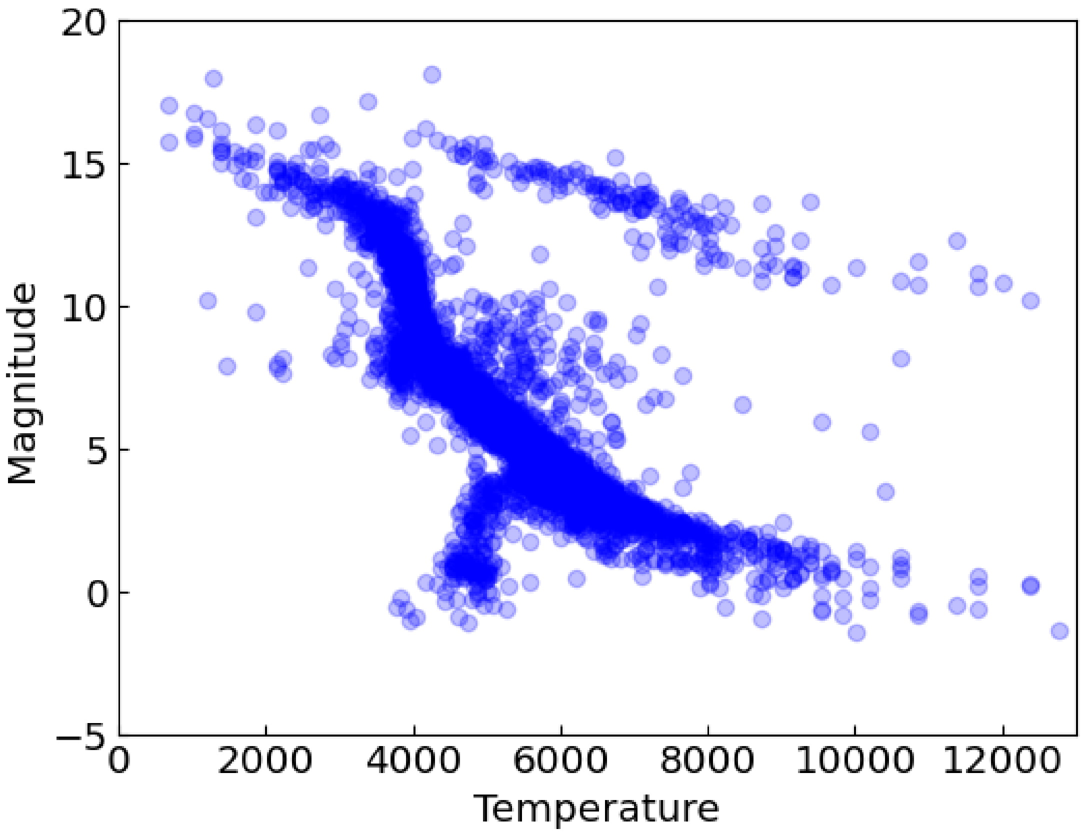
Indistinguishable line styles



Scatter plots

Not all data points are suitable to be joined by lines

Consider the observations of star surface temperature (= x) and brightness (= y)




```
683.14508541 15.73
683.14508541 17.01
1012.83217289 15.86
1012.83217289 15.98
1012.83217289 16.73
1195.25068152 10.19
1195.25068152 16.56
1289.42232154 17.99
1384.98930374 15.0
1384.98930374 15.38
1384.98930374 15.39
1384.98930374 15.56
1384.98930374 15.64
1384.98930374 16.15
1481.51656803 7.86
```

⋮

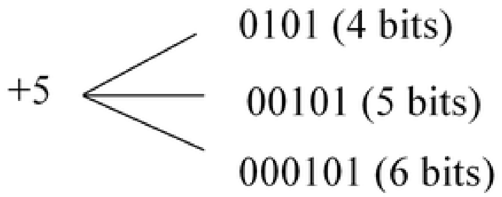
Errors and accuracy

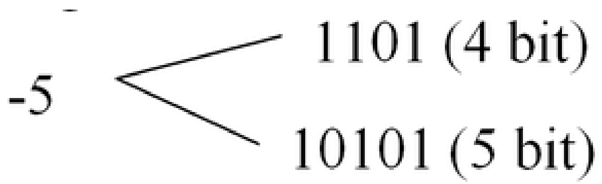
References: Chapter 4 of *Computational Physics* by Mark Newman

Chapter 1.1 of *Numerical Recipes Third Edition* by W.H. Press et al.

Integer representation

Numbers on a computer are represented by bits





Most typical native formats:

- 32-bit integer, range $-2{,}147{,}483{,}647$ (-2^{31}) to $+2{,}147{,}483{,}647$ (2^{31})
- 64-bit integer, range $\sim -10^{18}$ (-2^{63}) to $+10^{18}$ (2^{63})

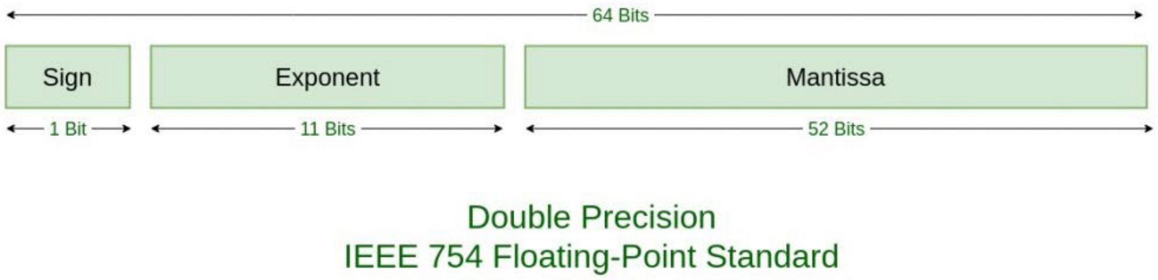
Python supports natively larger numbers but calculations can become slow

In C++ it is important to avoid under/overflow

Floating-point number representation

Floating point numbers represented by bit sequences as well separated into:

- Sign S
- Exponent E
- Mantissa M (significant digits)



$$x = S \times M \times 2^{E-e}$$

Main consequence: Floating-point numbers are not exact!

For example, with 52 bits one can store about 16 decimal digits

Range: from $\sim$ -10^{308} to 10^{308} for a 64-bit float

Floating-point number representation

When you write

$$x = 1.$$

What it means

$$x = 1. + \varepsilon_M, \qquad \varepsilon_M \sim 10^{-16} \qquad \text{for 64-bit float}$$

Example: Equality test

```python
x = 1.1 + 2.2

print("x = ",x)

if (x == 3.3):
    print("x == 3.3 is True")
else:
    print("x == 3.3 is False")
```

Example: Equality test

```python
x = 1.1 + 2.2

print("x = ",x)

if (x == 3.3):
    print("x == 3.3 is True")
else:
    print("x == 3.3 is False")
```

```
x =  3.3000000000000003
x == 3.3 is False
```

Example: Equality test

```python
x = 1.1 + 2.2

print("x = ",x)

if (x == 3.3):
    print("x == 3.3 is True")
else:
    print("x == 3.3 is False")
```

```
x =  3.3000000000000003
x == 3.3 is False
```

Instead you can do

```python
print("x = ",x)

# The desired precision
eps = 1.e-12

# The comparison
if (abs(x-3.3) < eps):
    print("x == 3.3 to a precision of",eps,"is True")
else:
    print("x == 3.3 to a precision of",eps,"is False")
```

```
x =  3.3000000000000003
x == 3.3 to a precision of 1e-12 is True
```

Error accumulation

$x = 1. + \varepsilon_M, \qquad \varepsilon_M \sim 10^{-16}$ unavoidable round-off error

Errors also accumulate through arithmetic operations, e.g.

$$y = \sum_{i=1}^{N} x_i$$

- $\sigma_y \sim \sqrt{N} \epsilon_M$ is errors are independent
- $\sigma_y \sim N \epsilon_M$ is errors are correlated
- σ_y can be large in some other cases

Example: Two large numbers with small difference

Let us have $x = 1$ and $y = 1 + \delta\sqrt{2}$

$$\delta^{-1}(y - x) = \sqrt{2} = 1.41421356237\ldots$$

Let us test this relation on a computer for a very small value of $\delta = 10^{-14}$

Example: Two large numbers with small difference

Let us have $x = 1$ and $y = 1 + \delta\sqrt{2}$

$$\delta^{-1}(y - x) = \sqrt{2} = 1.41421356237\ldots$$

Let us test this relation on a computer for a very small value of $\delta = 10^{-14}$

```python
from math import sqrt

delta = 1.e-14
x = 1.
y = 1. + delta * sqrt(2)
res = (1./delta)*(y-x)
print(delta,"* (y-x) = ",res)
print("The accurate value is sqrt(2) = ", sqrt(2))
print("The difference is ", res - sqrt(2))
```

```
1e-14 * (y-x) =  1.4210854715202004
The accurate value is sqrt(2) =  1.4142135623730951
The difference is  0.006871909147105226
```

Other examples (see the sample code)

- Roots of a quadratic equation with $|ac| << b^2$ (cancellation of two large numbers)

$$ax^2 + bx + c = 0$$

$$x = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

- Simple numerical derivative

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

Sometimes small h is too small