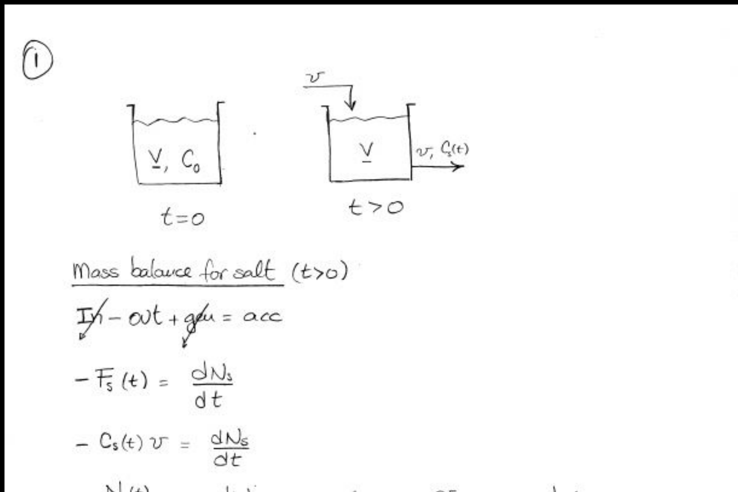
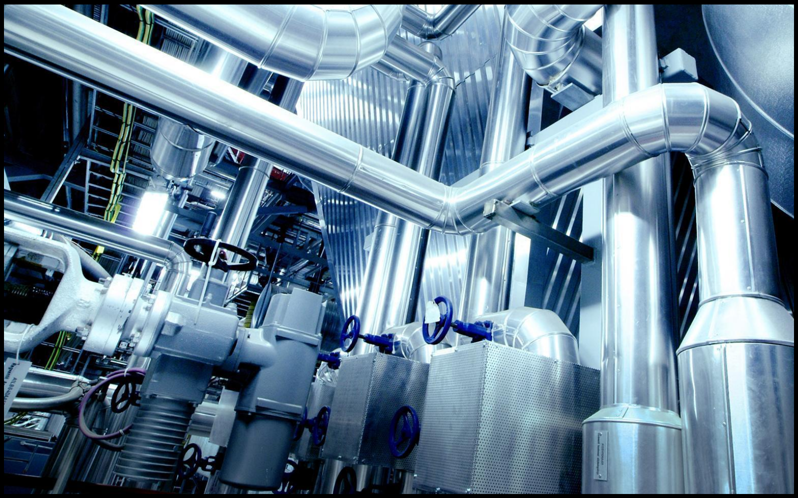




ChE352

Numerical Techniques for Chemical Engineers

Professor Stevenson

Lecture 3

Numpy arrays

np.array of ints

```
import numpy as np

x = np.array([1, 4, 3])

y = np.array([[1, 4, 3], [9, 2, 7]])

print('x.shape:', x.shape, ' x.size:', x.size)

print('y.shape:', y.shape, ' y.size:', y.size)
```

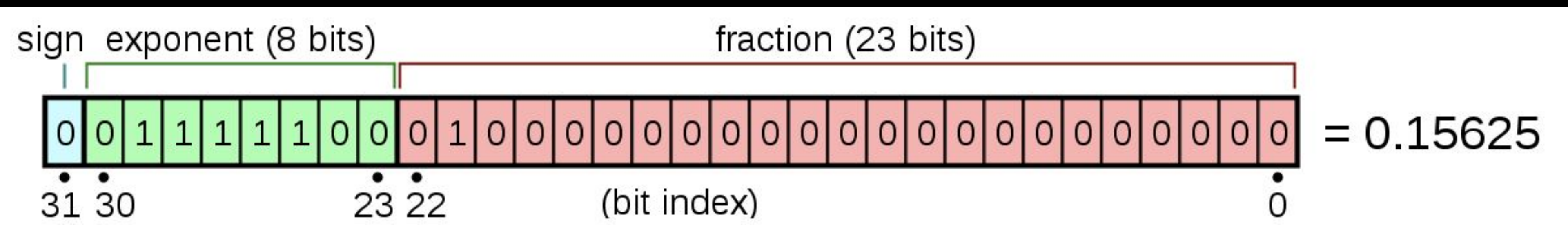
```
x.shape: (3,)  x.size: 3
y.shape: (2, 3)  y.size: 6
```

tuple of ints

int

Floating point numbers

- Computer math is almost always floating point
- Like scientific notation with powers of 2 only



- np.float32 holds ~7 decimal digits
- np.float64 holds ~16 decimal digits
- Not every real number can be represented
- Too big = overflow, too small = underflow
- Only binary fractions (no exact 1/3, 1/5, etc)

```python
print('Using f-strings, print 20 digits for each number')
for float_type in [np.float64, np.float32, np.float16]:
    x1 = float_type(1)
    print(f'{float_type} 1.0 is really {x1:.20f}')
    x2 = float_type(0.1)
    print(f'{float_type} 0.1 is really {x2:.20f}')
```

How does this code work?

What do you expect it to print?

```python
print('Using f-strings, print 20 digits for each number')
for float_type in [np.float64, np.float32, np.float16]:
    x1 = float_type(1)
    print(f'{float_type} 1.0 is really {x1:.20f}')
    x2 = float_type(0.1)
    print(f'{float_type} 0.1 is really {x2:.20f}')
```

```
Using f-strings, print 20 digits for each number
  float64 1.0 is really 1.00000000000000000000
  float64 0.1 is really 0.10000000000000000555
  float32 1.0 is really 1.00000000000000000000
  float32 0.1 is really 0.10000000149011611938
  float16 1.0 is really 1.00000000000000000000
  float16 0.1 is really 0.09997558593750000000
```

Floating-point operations (flops)

- A computer can do billions/second (Gflops)
  - This is why we tolerate floating point issues
- Floating point arithmetic has <u>round-off error</u>
- <u>Swamping</u> (A+B ≈ A) or <u>Cancelation</u> (A–B ≈ 0) are the most common types of round-off error

What values of A, B might cause swamping?

What values might cause cancelation?

Try it in Google Colab!

Noticing round-off error

1. Avoid subtracting very similar numbers

   Why?

2. Avoid adding big + small or dividing big/small

3. Avoid multiplying big*big or small*small

4. Check your answer if possible

5. Test for "nearness" instead of exact equality

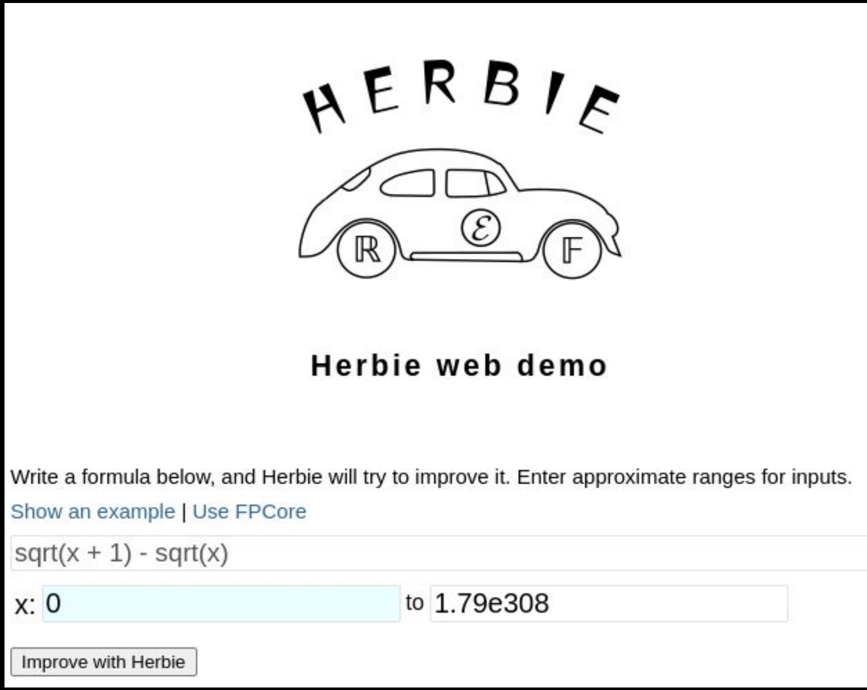
Noticing round-off error

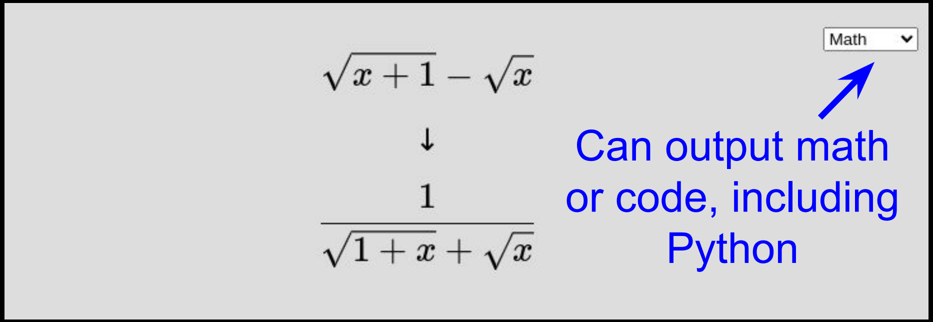
1. Avoid subtracting very similar numbers
   - sqrt(x + 1) - sqrt(x) = 1 / (sqrt(x + 1) + sqrt(x))
2. Avoid adding big + small or dividing big/small
   - big / small ≅ big / (small + epsilon)
3. Avoid multiplying big*big or small*small
   - log(A*A) = 2 log(A)
4. Test your answer if possible
   - if solving for f(x) = 0, check f(x)
5. Test for close instead of exact equality
   - abs(f(x)) < 0.001, not f(x) == 0.0

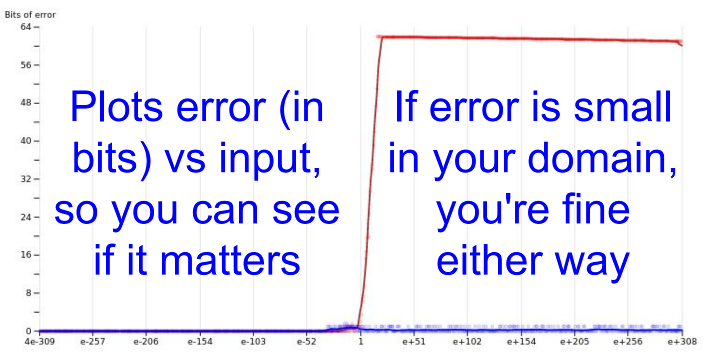
Dealing with round-off error

Try an expression analyzer such as
https://herbie.uwplse.org/demo/

Improves accuracy by testing alternate expressions over a random sample of inputs







Which float do you get by default?

```
x = 0.1  # default Python float
print(f'{type(x).__name__} 0.1 is really {x:.20f}')
xx = np.array([0.1])  # default Numpy array
print(f'{type(xx[0]).__name__} 0.1 is really {xx[0]:.20f}')
```

How does this code work?

What do you expect it to print?

Which float do you get by default?

```python
x = 0.1  # default Python float
print(f'{type(x).__name__} 0.1 is really {x:.20f}')
xx = np.array([0.1])  # default Numpy array
print(f'{type(xx[0]).__name__} 0.1 is really {xx[0]:.20f}')
```

```
float 0.1 is really 0.10000000000000000555
float64 0.1 is really 0.10000000000000000555
```

Python and Numpy both use float64 by default
(most precise float type implemented in hardware,
thus most precision available while keeping speed)

NumPy array examples

Try the following exercises from PNM 2.8, 17-21:

17. Generate an array with size 100 evenly spaced between -10 to 10 using *linspace* function in Numpy.
18. Let array_a be an array [-1, 0, 1, 2, 0, 3]. Write a command that will return an array consisting of all the elements of array_a that are larger than zero. Hint: Use logical expression as the index of the array.
19. Create an array $y = \begin{pmatrix} 3 & 5 & 3 \\ 2 & 2 & 5 \\ 3 & 8 & 9 \end{pmatrix}$ and calculate the transpose of the array.
20. Create a zero array with size (2, 4).
21. Change the 2nd column in the above array to 1.

Next reading: Taylor's Theorem (PNM 18.3), Real Analysis (F&B 1.2),