Welcome to
CHE 384T: Computational Methods in Materials Science

Introduction to Computational Materials Science

Programming Day 2

The University of Texas at Austin
McKetta Department
of Chemical Engineering
Cockrell School of Engineering

Programming Day Agenda

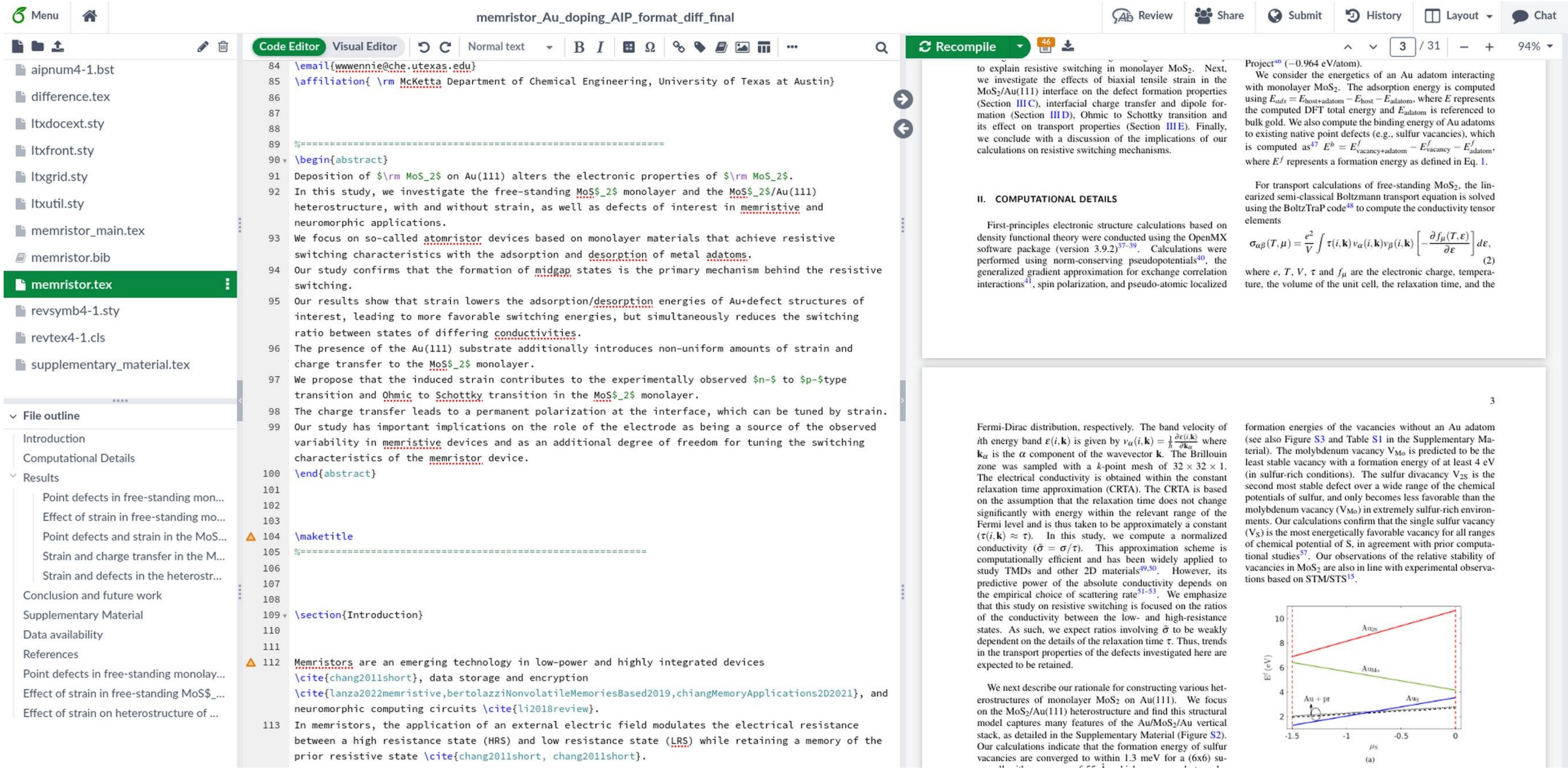
Wrap up of setting up your environment:
- LaTeX
- Helpful packages

Python primers:
- the `Dataframe` object
- global v local variables

Some logistics on Peer-review day and grading

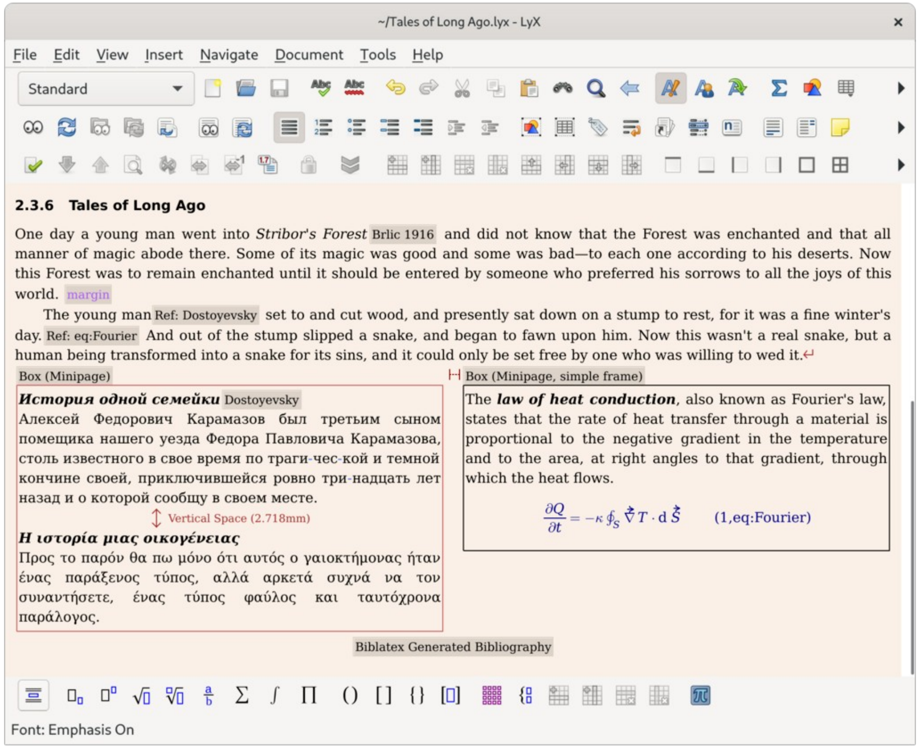
Coding considerations for the Random Walk Model
- Binning
- Random number generators (RNGs)
- Code outline for RW model

LaTeX: document typesetting



Document typesetting: WYSIWYM

LyX



Joplin

Python Primer(s): manage your environment

Some modules and packages
(`pip install` or `conda`):

- **numpy**: lean array computing
- **scipy**: extends numpy for array computing
- **matplotlib.pyplot**: basic visualization
- **seaborn**: prettified matplotlib
- **plotly**: interactive plots, great for data science
- **icecream**: debugging beyond `print()`

Related to materials science:

- **ase**: interface with materials science simulations,
  lots of open-source codes
- **pymatgen**: interface with materials science simulations,
  mostly VASP
- **pyscf:** quantum chemistry code

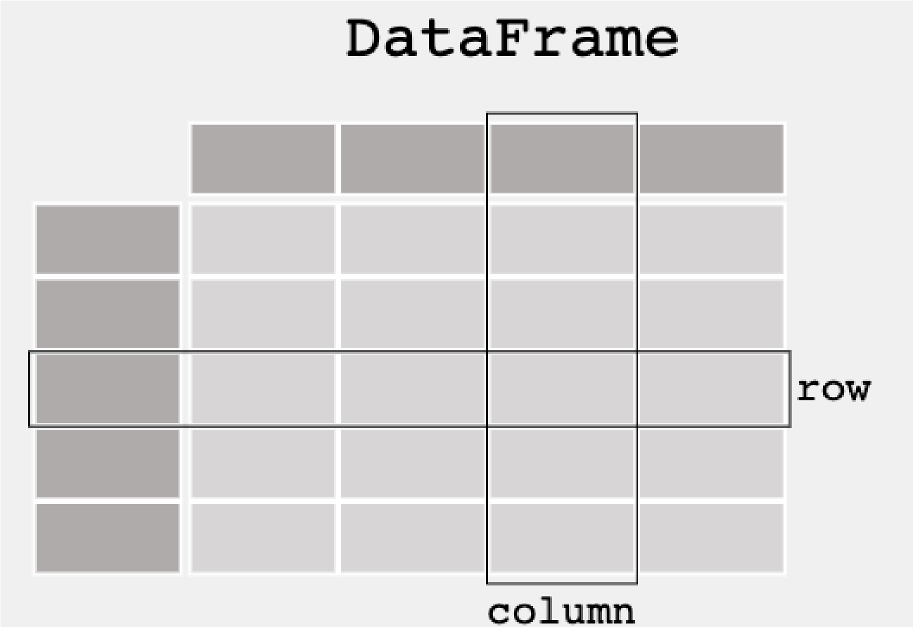
Related to data science:

- **Pandas**: dataframes
- **Sklearn**: some basic ML
- **PyTorch**: for deep learning,
  good for prototyping
- **Tensorflow**: for deep learning

`pip`: python package management

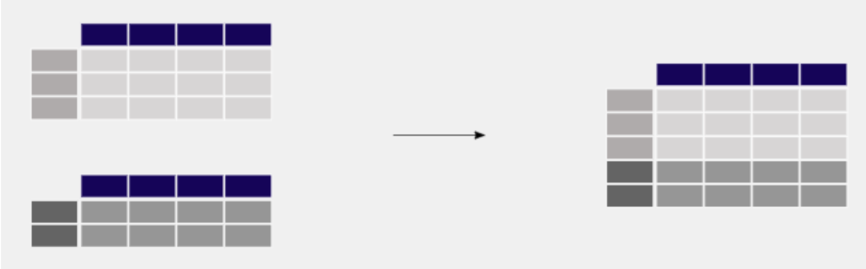
`conda`: general system package management

Python Primer(s): Pandas

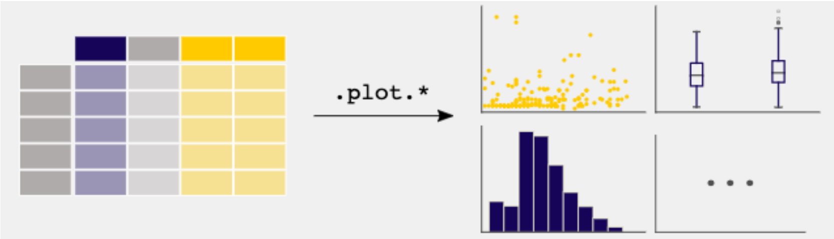


Import/export to multiple tabular formats

"Clean up" data, merge data, subselect data



Visualize and characterize data



https://www.nvidia.com/en-us/glossary/pandas-python/

Python Primer(s): Pandas

```
# Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt

# Generate sample data with NumPy
x_np = np.linspace(0, 10, 100)  # 100 points
between 0 and 10
y_np = np.sin(x_np)  # Sine wave using NumPy

# Create a NumPy plot
plt.figure(figsize=(8, 4))
plt.plot(x_np, y_np, label='NumPy Sine Wave')
plt.title('Plotting with NumPy')
plt.xlabel('X')
plt.ylabel('sin(X)')
plt.legend()
plt.show()
```

```
# Import necessary libraries
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

# Generate the same data using Pandas
x_pd = pd.Series(np.linspace(0, 10, 100))  # 100
points between 0 and 10 using pandas.Series
y_pd = x_pd.apply(np.sin)  # Apply sine function to
the Series

# Create a Pandas DataFrame
df_pd = pd.DataFrame({'X': x_pd, 'Sine Wave': y_pd})

# Create the plot using Pandas
df_pd.plot(x='X', y='Sine Wave', figsize=(8, 4),
title='Plotting with Pandas Generated Data',
legend=True)
plt.xlabel('X')
plt.ylabel('sin(X)')
plt.show()
```

Python Primer(s): global v local variables

Return local variable

```
x = 42
def f():
    x = 'alice'
    return x
```

```
x = 42

def f():
    x = 'alice'
    return x

print(f())
# alice

# Has the global variable x changed?
print(x)
# Output: 42 - no it has not changed.
```

Return global variable

```
x = 42
def f():
    global x
    x = 'alice'
    return x
```

```
x = 42

def f():
    global x
    x = 'alice'
    return x

print(f())
# alice

# Has the global variable x changed?
print(x)
# Output: alice - yes it has changed.
```

Peer review day

Submit your report (*.pdf) and code (*.zip) to Canvas

Rubric

Example solutions report

We will do 1-2 rounds of peer review

Peer review of your code, example questions:

- Is the code readable?
- Is the logic of the code implemented understandable and easy to follow?
- Compare and contrast choices of code organization
- Compare and contrast choices of code implementation

Peer review of your report:

- Is the report organized, self-contained and written clearly?
- Are the figures complete, legible, and clear?
- Compare and contrast choices of computational parameters, e.g., statistical sampling.
- Compare and contrast choices in formatting and analysis.

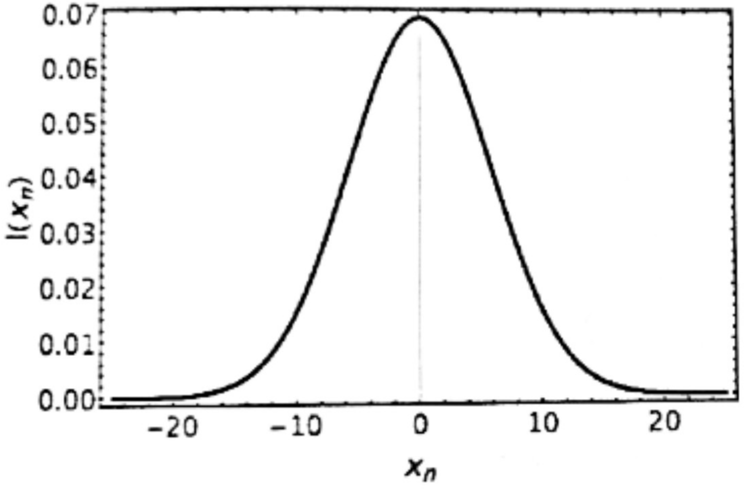
Reflection on peer review, Opportunity for resubmission: Sept. 16, 11:59pm

Random walk diffusion: a small simulation

https://rwd2d-mercury.runmercury.com/

Statistics of the Random Walk Model: End-to-end distribution

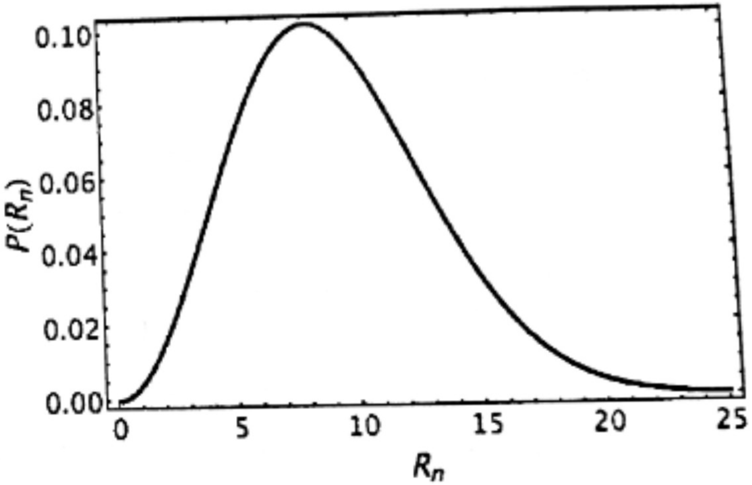
(In 1D)



$$\mathcal{P}(n_{tot}, q) = \frac{1}{2}^{n_{tot}} \frac{n_{tot}!}{(\frac{n_{tot}+q}{2})!(\frac{n_{tot}-q}{2})!}$$

Statistics of the Random Walk Model: End-to-end distribution

(In 3D)



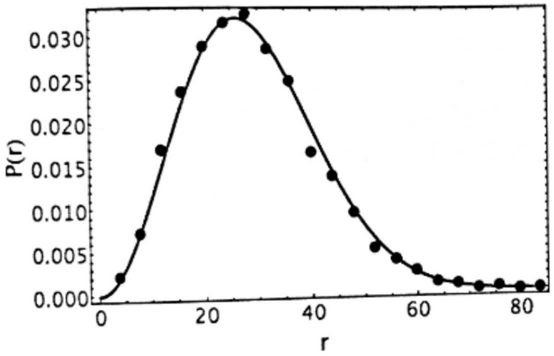
Coding considerations:

Binning distributions

Discretizing continuous functions
Appendix I.3

Choose n_{bin}

$$\Delta = \frac{R_n^{max} - R_n^{min}}{n_{bin}}$$




```
#Pseudocode for binning procedure

def binning(…):
    #given n_bin, figure out bin intervals

    # count frequency of each item into each bin

    # compute the probability of each frequency

    # normalize the distribution
    # so that the total integration of the
    # probability distribution is equal to one

    return … normed_prob_distribution
```

Donald R. Askeland. The Science and Engineering of Materials 3rd ed.

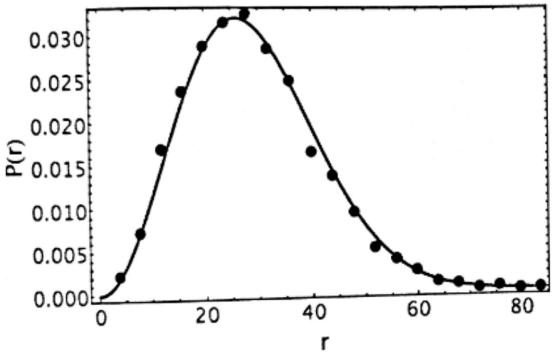
Coding considerations:

Binning distributions

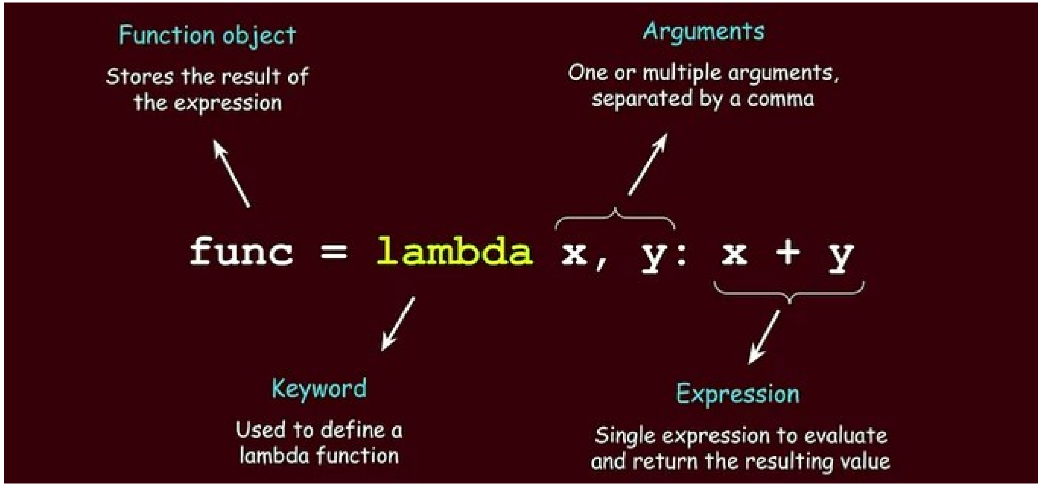
Discretizing continuous functions
Appendix I.3

Choose n_{bin}

$$\Delta = \frac{R_n^{max} - R_n^{min}}{n_{bin}}$$




```
# Lambda functions:
# small anonymous functions
```




```
print(func(1,2))
>>> 3
```

Donald R. Askeland. The Science and Engineering of Materials 3rd ed.

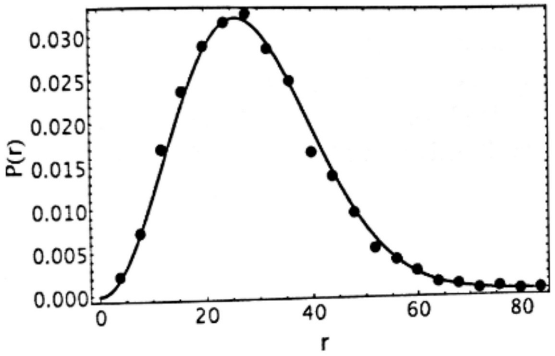
Coding considerations:

Binning distributions

Discretizing continuous functions
Appendix I.3

Choose n_{bin}

$$\Delta = \frac{R_n^{max} - R_n^{min}}{n_{bin}}$$



Random Number Generator

Appendix I.2

$$I_{i+1} = aI_i (\mathrm{mod}\ m)$$

$$a = 7^5 = 16807, m = 2^{31} - 1 = 2147483647$$

$$F = I_i/m, 0 < F_i < 1$$

To reproduce a “random” result,
pick a consistent *seed*

Python:

numpy.random.rand(...): generates (pseudo-)random number over [0,1)

numpy.ceil(…): round to next highest integer

Donald R. Askeland. The Science and Engineering of Materials 3rd ed.

An implementation of the 2D Random Walk Diffusion

Objective	Concept	Python Representation
User chooses *nt* = number of time steps	Variable, integer	`nt`

An implementation of the 2D Random Walk Diffusion

Objective

User chooses *nt* = number of time steps

Keep track of position of random walker at each time step.
Let's assume it starts at the origin.

Concept

Variable, integer

Array (list of items)
e.g., [3, 4.5, 8, -1]

2D → *x* and *y* coordinate for each position

Python Representation

```
nt
```

Use the library numpy, shorthand is np:

```
x = np.zeros(nt+1)
y = np.zeros(nt+1)
```

An implementation of the 2D Random Walk Diffusion

Objective	Concept	Python Representation
User chooses *nt* = number of time steps	Variable, integer	`nt`

Keep track of position of random walker at each time step.
Let's assume it starts at the origin.
	Array (list of items)
	e.g., [3, 4.5, 8, -1]
	
	2D → *x* and *y* coordinate for each position
		Use the library numpy, shorthand is np:
		
		`x = np.zeros(nt+1)`
		`y = np.zeros(nt+1)`

Specify how the position changes at each time step.		`delx = np.array([?,?,?,?])` `dely = np.array([?,?,?,?])`

An implementation of the 2D Random Walk Diffusion

Objective	Concept	Python Representation
User chooses *nt* = number of time steps	Variable, integer	`nt`

Keep track of position of random walker at each time step.
Let's assume it starts at the origin.
	Array (list of items)
	e.g., [3, 4.5, 8, -1]
	
	2D → *x* and *y* coordinate for each position
		Use the library numpy, shorthand is np:
		
		`x = np.zeros(nt+1)`
		`y = np.zeros(nt+1)`

Specify how the position changes at each time step.		`delx = np.array([?,?,?,?])` `dely = np.array([?,?,?,?])`

Save each new position of the diffusion path	Index the array
	i.e., access a specific element
	"zero index"
		`x = [1,2,3]`
		`x[0] = 1`
		`x[1] = 2`

An implementation of the 2D Random Walk Diffusion

Objective	Concept	Python Representation

Repeat for *nt* times	for loop
	range function
		Input:
		`for i in range(3):`
		`    print(i)`
		Output:
		`0`
		`1`
		`2`

Encode the random number to a change in position of the random walker	Generate a (pseudo)-random number	`np.floor(4* np.random.rand(nt))` Generate random number b/t 0 and 1 Random number b/t 0 and 4 Random integer: 0, 1, 2, 3