Automation with python
(for atomistic models)

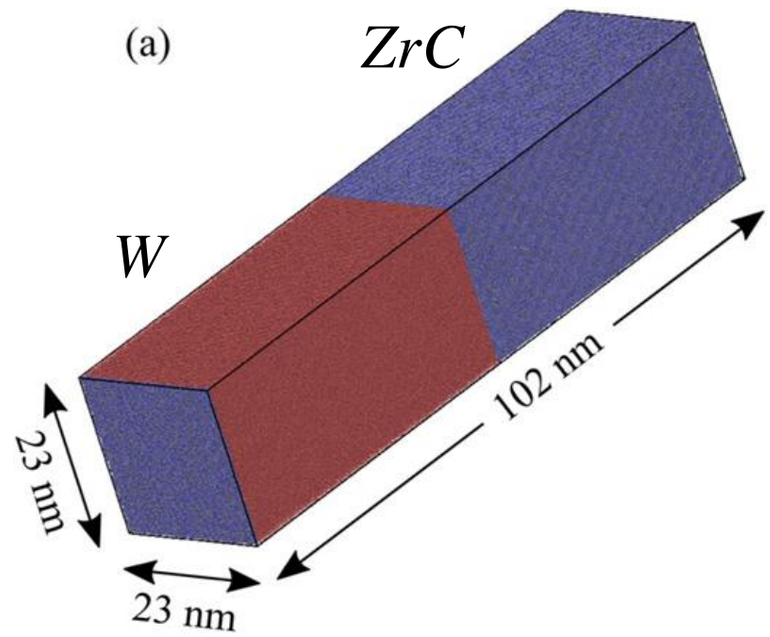
James Goff

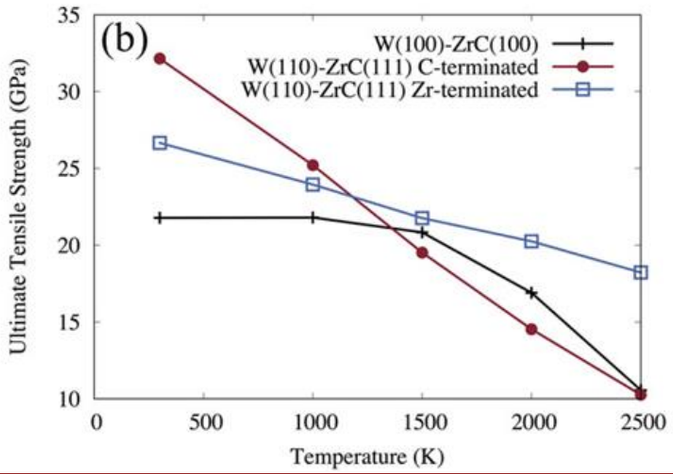
Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

General motivation

Large-scale atomistic simulations for materials science

Sikorski et. al.
JCP 2021

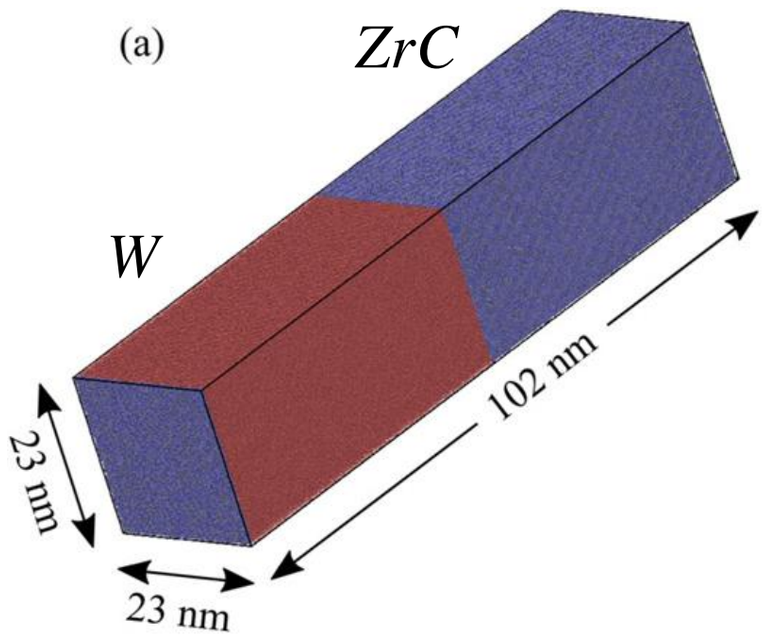


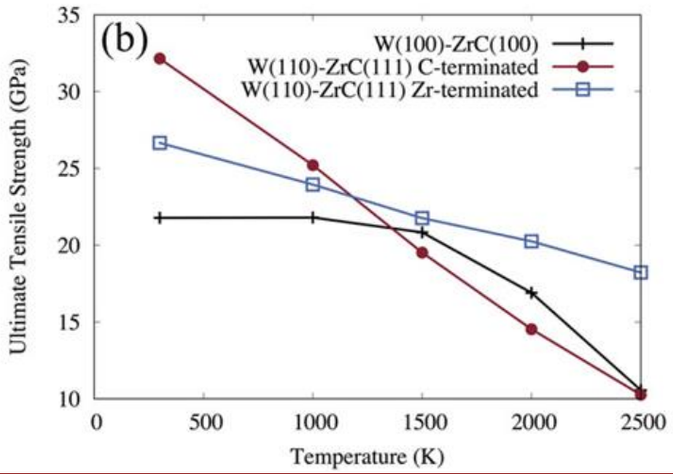


General motivation

Machine learned atomistic models require a lot of data

Sikorski et. al.
JCP 2021





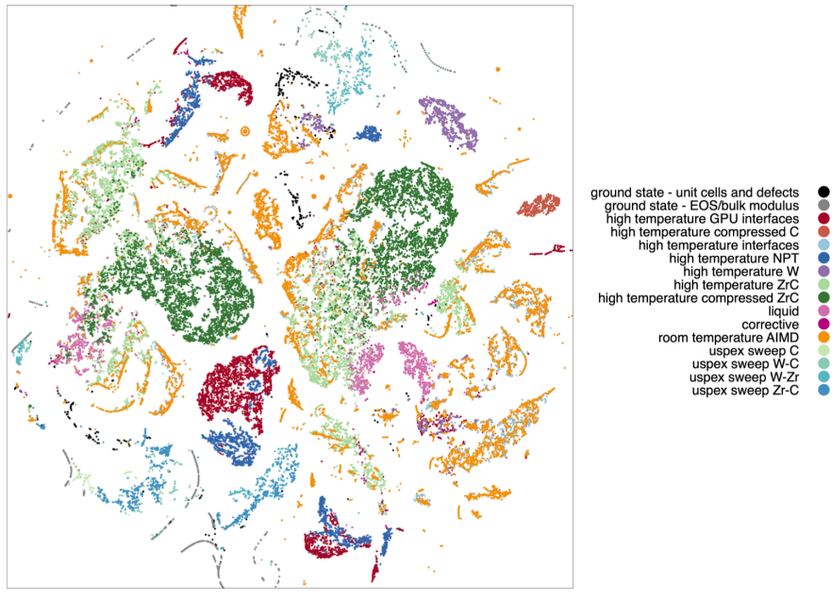
Model and training data?

$$U = \sum_k \beta_k B_k$$

B_k ← Bispectrum component (4-body SNAP descriptors)

β_k ↑ Linear parameter (m in y=mx +b)

Training data:



Outline

1. Lecture/topical
   - Brief overview of coding topics for automation
   - Automation frameworks
   - Application-specific automation (what we will be doing mostly)

2. Applied examples and tutorial
   - How to use python modules (e.g. pymatgen) to help automate VASP calculations
   - How to automate Quantum Espresso calculations
   - Review tools for automatically parsing DFT output
   - Automate LAMMPS simulations
   - Automate FitSNAP (automated ML model training)

Learning outcomes

Technical outcomes

1. Learn what automation is, and for relevant cases
2. Learn vocab. about common coding concepts in automation and basic usage
3. Know where to look in the future for automation needs (what is state-of-the-art)

Practical outcomes

1. Learn how to use tools to automate DFT calculation setup and submission
2. Learn how to automate LAMMPS molecular dynamics simulations
3. Learn how to automate FitSNAP model training

What is automation?

Let's ask something that required a lot of automation

Output from chatGPT

Definition:

- Automation in scientific computing involves using computer programs and tools to simplify and streamline repetitive or time-consuming tasks in scientific research and data analysis.
- It includes developing algorithms, scripts, and workflows that automate various aspects of the scientific process.
- Automation can be applied to tasks such as data collection, experiment control, data processing and cleaning, numerical simulations, statistical analysis, optimization, and result visualization.

Benefits of automation:

- Automation in scientific computing improves efficiency, reduces human error, and accelerates the research process.
- It enables scientists to focus more on result interpretation, hypothesis formulation, and scientific discovery.
- Automation facilitates reproducibility and enables the analysis of large and complex datasets.

Coding concepts

Go to your automation/python_concepts folder

Coding concepts for automation: functions

See: `automation/python_concepts/functions`

From `example_function.py` :

```
def add_numbers(a,b):
    result = a + b
    return result

number1 = 1
number2 = 7

result = add_numbers(number1,number2)
print (result)
```

function: Self-contained block of reusable code that does some task

Define the function using `def` **and specify inputs**

Indent and write what the function should do with inputs

Return the result

Call the function as many times as you want with inputs: `number1` **and** `number2`

Coding concepts for automation: classes

See: `automation/python_concepts/classes`

```python
class Number_Pair:
    def __init__(self,a,b):
        self.a = a # classes have attributes -we are making a and b attributes
        self.b = b
        self.added = None
        self.subtracted = None
        self.multiplied = None
        self.add() # we can call some class functions from within the init function
        return None

    def add(self):
        self.added = self.a + self.b

    def subtract(self):
        self.subtracted = self.a - self.b
        return self.subtracted

    def multiply(self,a,b):
        self.multiplied = a * b
        return self.multiplied
```

Objects possess attributes (variables) and methods (functions)

Great if you need to use a lot of functions on the same variables!

Coding concepts for automation: classes

See: `automation/python_concepts/classes`

```python
from example_class import Number_Pair

a = 1
b = 7

numspair = Number_Pair(a,b)

# print the attribute that corresponds to the addition of a and b
print(numspair.added)

# print all of the attributes
print (vars(numspair))

# use another method

numspair.subtract()

#print the result

print(numspair.subtracted)
```

We may define a Number_Pair object and use its methods in other scripts!

Addition is done by default

And we make it do subtraction

Coding concepts for automation: control flow/logic

See: `automation/python_concepts/control_flow`

```
from example_class import Number_Pair



a_lst = [1,2,3,5,7]
b_lst = [2,2,1,2,5]

for a,b in zip(a_lst,b_lst): # zip is a way to iterate over the two lists above
    nmpr = Number_Pair(a,b)
    nmpr.subtract()
    nmpr.multiply(a,b)
    print (a,b,'addition:', nmpr.added, 'subtracted:', nmpr.subtracted)
```

We need to be able to do for loops!

Coding concepts for automation: control flow/logic

See: `automation/python_concepts/control_flow`

```
from example_class import Number_Pair


a_lst = [1,2,3,5,7]
b_lst = [2,2,1,2,5]

for a,b in zip(a_lst,b_lst): # zip is a way to iterate over the two lists above
    nmpr = Number_Pair(a,b)
    if a > b:
        subtraction_order = 'a-b'
    else:
        subtraction_order = 'b-a'
    nmpr.subtract(subtraction_order)
    nmpr.multiply(a,b)
    print (a,b,'addition:', nmpr.added, 'subtracted:', nmpr.subtracted)
```

We often need to control how and what we operate on

Coding concepts for automation: file I/O

See: `automation/python_concepts/file_io`

From `read_file.py` :

```
# path to file
f = './data/example.txt'
# use the open function in read 'r' mode
with open(f, 'r') as readin:
    # read in line per line
    lines = readin.readlines()
    for line in lines:
        print (line)
```

We may open files and read through them in python

Coding concepts for automation: file I/O

See: `automation/python_concepts/file_io`

From `read_collect_write.py` :

```python
import numpy as np

# path to file
f = './data/example.txt'
# use the open function in read 'r' mode
with open(f, 'r') as readin:
    # read in line per line
    lines = readin.readlines()

for line in lines:
    l = line.split()

    if 'energy' in line:
        energy = float(l[-1])
        with open('energy_only.txt', 'w') as writeout:
            writeout.write('%1.2f' % energy)

    if 'force' in line:
        frc_vec = [float(k) for k in l[1:]]
        frc_vec = np.array(frc_vec)
        np.save('force.npy',frc_vec)
```

We may open files and read through them in python

As we do so we can manipulate and store certain file contents we want

Coding concepts for automation: file I/O

See: `automation/python_concepts/file_io`

From `read_collect_write_alt.py` :

```
from subprocess import call

# path to file
f = './data/example.txt'

# use subprocess call to execute a shell command instead

command = """grep 'energy' %s | awk '{print ($NF)}' > energy_only2.txt""" % f

call(command, shell = True)
```

Sometimes, bash/shell commands are more efficient than python functions are for parsing info.
You can still use those commands in python

You can run pretty much any shell command from python using this call funtion

Other coding concepts for automation

Other important concepts we won't cover in detail

Data types

- File and data types for storing information (json, pandas, arrays, dictionaries)

More modules

- Some modules such as json, pandas, and pysql are great for data storage and management

Error and exception handling

- try, except statements
- More sophisticated error handlers (details in AiiDA)

High-throughput computational frameworks

Streamline and homogenize methods we just covered

- Homogenize computational environments and efforts
- Manage procedures
  - For high-volume approaches, automation
- Store all data in a convenient, systematic way

Some computational workflow managers

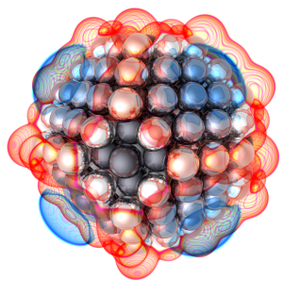
FireWorks

AiiDA

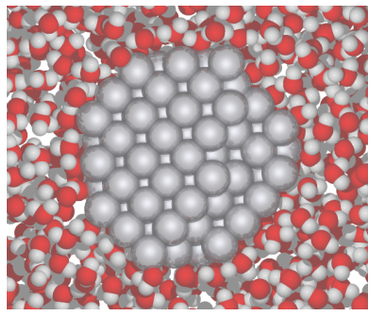
pyiron

AiiDA schematic

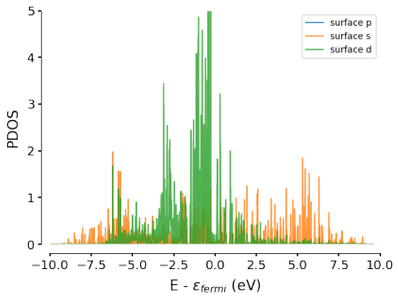
Q.E.



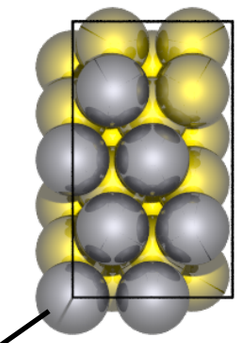
LAMMPS…



DOS…



Structures



Multiple Users

Python Code/ shells

Calc. ORM | Data ORM | Scheduling Manager

AiiDA API
(Main Python Code)

AiiDA Daemon
(submission, retrieval, and parsing of calculations)

SQL Database

PBS

SLURM

SSH

Local

Application specific automation

For VASP, QE, FitSNAP, LAMMPS, etc.

Application-specific automation

You write your own code (python, bash, etc.) to automate tasks in your workflow

Pros

- You can do this for any* task, simulation, or other part of a workflow
- Initial barrier/learning curve is lower than for automation frameworks
- Usually faster to set up for your current project(s)

Cons

- Can be difficult to look at/reuse years from now (also for someone else to use)
- We are “reinventing the wheel” a bit when we do this - some of our effort has already been done
- Often need to do more work to change systems and applications compared to frameworks

Why use AiiDA? Existing plugins/interfaces

Many pre-existing workflows and for DFT and MD

- Quantum Espresso, (relaxations, band structures, self-consistent Hubbard parameters, CP-MD)
- VASP (relaxations, other simple calculations/ simulations)
- NWChem, Gaussian, Psi4 (local basis set codes)
- Site-dependent self-consistent Hubbard U + V algorithms
- Fireworks (as an alternative transport)
- LAMMPS, GROMACS, GULP (Classical MD)
- ICET Monte Carlo and fixed-lattice (cluster expansion modeling and simulation)
- Phonopy phonon/vibrational analysis
- aenet machine-learning IAP training **(very dated)**

AiiDA database visualization for refractory metals tests

Database for W, Nb, Ta, Ti, Mo Bravais lattice relaxations