Python and its libraries for scientific applications

Andrey A. Golov

What is Python?

Python is an interpreted, interactive, object-oriented programming language. It incorporates modules, exceptions, dynamic typing, very high level dynamic data types, and classes. Python combines remarkable power with very clear syntax. It has interfaces to many system calls and libraries, as well as to various window systems, and is extensible in C or C++. It is also usable as an extension language for applications that need a programmable interface. Finally, Python is portable: it runs on many Unix variants, on the Mac, and on Windows 2000 and later.

https://www.python.org/

https://docs.python.org/3/faq/

Why Python?

- **Batteries included** Rich collection of already existing bricks of classic numerical methods, plotting or data processing tools. We dont want to re-program the plotting of a curve, a Fourier transform or a fitting algorithm. Dont reinvent the wheel!
- **Easy to learn** Most scientists are not payed as programmers, neither have they been trained so. They need to be able to draw a curve, smooth a signal, do a Fourier transform in a few minutes.
- **Easy communication** To keep code alive within a lab or a company it should be as readable as a book by collaborators, students, or maybe customers. Python syntax is simple, avoiding strange symbols or lengthy routine specifications that would divert the reader from mathematical or scientific understanding of the code.
- **Efficient code** Python numerical modules are computationally efficient. But needless to say that a very fast code becomes useless if too much time is spent writing it. Python aims for quick development times and quick execution times.
- **Universal** Python is a language used for many different problems. Learning Python avoids learning a new software for each new problem.

https://scipy-lectures.org/

Python Syntax

3. An Informal Introduction to Python

In the following examples, input and output are distinguished by the presence or absence of prompts (>>> and ...): to repeat the example, you must type everything after the prompt, when the prompt appears; lines that do not begin with a prompt are output from the interpreter. Note that a secondary prompt on a line by itself in an example means you must type a blank line; this is used to end a multi-line command.

Many of the examples in this manual, even those entered at the interactive prompt, include comments. Comments in Python start with the hash character, `#`, and extend to the end of the physical line. A comment may appear at the start of a line or following whitespace or code, but not within a string literal. A hash character within a string literal is just a hash character. Since comments are to clarify code and are not interpreted by Python, they may be omitted when typing in examples.

Some examples:

```
# this is the first comment
spam = 1  # and this is the second comment
          # ... and now a third!
text = "# This is not a comment because it's inside quotes."
```

3.1. Using Python as a Calculator

Let's try some simple Python commands. Start the interpreter and wait for the primary prompt, `>>>`. (It shouldn't take long.)

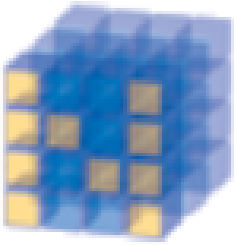
https://docs.python.org/

https://docs.python.org/3.8/tutorial/

Python Syntax

```
>>> print("Hello, World!")
Hello, World!
```

SciPy.org



NumPy

Base N-dimensional array package

IP[y]: IPython

IPython

Enhanced interactive console

SciPy library

Fundamental library for scientific computing

SymPy

Symbolic mathematics

Matplotlib

Comprehensive 2-D plotting

pandas

Data structures & analysis

NumPy

NumPy is the fundamental package for scientific computing with Python. It contains among other things:

- a powerful N-dimensional array object
- sophisticated (broadcasting) functions
- tools for integrating C/C++ and Fortran code
- useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

https://numpy.org/devdocs/user/quickstart.html

NumPy

Array

```
>>> import numpy as np
>>> a = np.arange(15).reshape(3, 5)
>>> a
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
>>> a.shape
(3, 5)
>>> a.ndim
2
>>> a.dtype.name
'int64'
>>> a.itemsize
8
>>> a.size
15
```

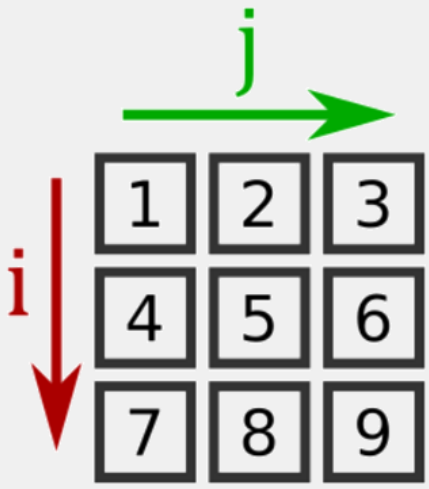
NumPy

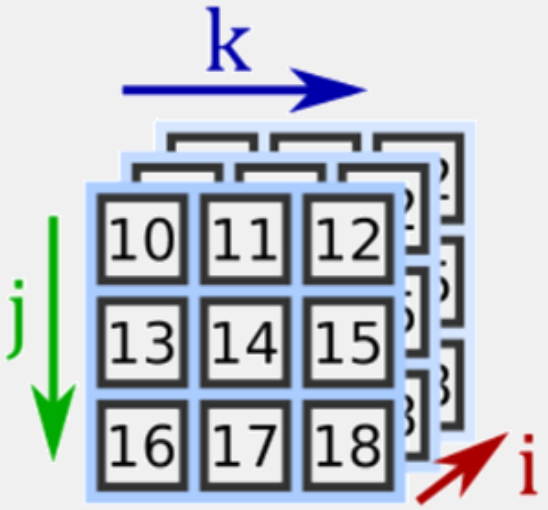
Array

```
>>> np.zeros( (3,4) )
array([[ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.]])
>>> np.ones( (2,3,4), dtype=np.int16 )
be specified
array([[[ 1, 1, 1, 1],
        [ 1, 1, 1, 1],
        [ 1, 1, 1, 1]],
       [[ 1, 1, 1, 1],
        [ 1, 1, 1, 1],
        [ 1, 1, 1, 1]]], dtype=int16)
```

NumPy *Indexing and slicing*

```
v = [[1, 2, 3],
     [4, 5, 6],
     [7, 8, 9]]

print(v[2][1]) # 8
print(a2[2]) # [7, 8, 9]
print(a2[:, 1]) # [2, 5, 8]
```





https://www.pythoninformer.com/python-libraries/numpy/index-and-slice/

NumPy

Basic Operations

```
>>> a = np.array( [20,30,40,50] )
>>> b = np.arange( 4 )
>>> b
array([0, 1, 2, 3])
>>> c = a-b
>>> c
array([20, 29, 38, 47])
>>> b**2
array([0, 1, 4, 9])
>>> 10*np.sin(a)
array([ 9.12945251, -9.88031624,  7.4511316 , -2.62374854])
>>> a<35
array([ True, True, False, False])
```

NumPy

Basic Operations

```
>>> A = np.array( [[1,1],
...                [0,1]] )
>>> B = np.array( [[2,0],
...                [3,4]] )
>>> A * B                          # elementwise product
array([[2, 0],
       [0, 4]])
>>> A @ B                          # matrix product
array([[5, 4],
       [3, 4]])
>>> A.dot(B)                       # another matrix product
array([[5, 4],
       [3, 4]])
```

>>>

NumPy

Linear Algebra

```
>>> a = np.array([[1.0, 2.0], [3.0, 4.0]])
>>> print(a)
[[ 1.  2.]
 [ 3.  4.]]

>>> a.transpose()
array([[ 1.,  3.],
       [ 2.,  4.]])

>>> np.linalg.inv(a)
array([[-2. ,  1. ],
       [ 1.5, -0.5]])

>>> u = np.eye(2) # unit 2x2 matrix; "eye" represents "I"
>>> u
array([[ 1.,  0.],
       [ 0.,  1.]])
```

NumPy

Linear Algebra

```
>>> j = np.array([[0.0, -1.0], [1.0, 0.0]])

>>> j @ j        # matrix product
array([[-1.,  0.],
       [ 0., -1.]])

>>> np.trace(u)  # trace
2.0

>>> y = np.array([[5.], [7.]])
>>> np.linalg.solve(a, y)
array([[-3.],
       [ 4.]])

>>> np.linalg.eig(j)
(array([ 0.+1.j,  0.-1.j]), array([[ 0.70710678+0.j        ,  0.70710678-0.j        ],
       [ 0.00000000-0.70710678j,  0.00000000+0.70710678j]]))
```

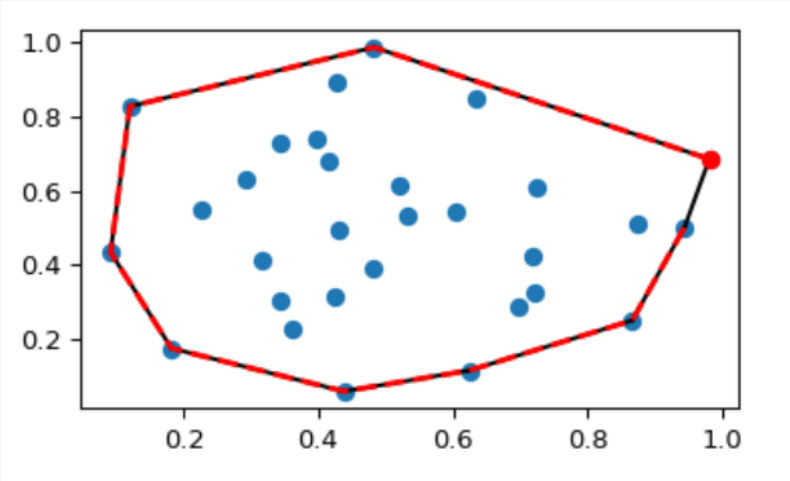
EXAMPLE

- Special functions (scipy.special)
- Integration (scipy.integrate)
- Optimization (scipy.optimize)
- Interpolation (scipy.interpolate)
- Fourier Transforms (scipy.fft)
- Signal Processing (scipy.signal)
- Linear Algebra (scipy.linalg)
- Sparse eigenvalue problems with ARPACK
- Compressed Sparse Graph Routines (scipy.sparse.csgraph)
- Spatial data structures and algorithms (scipy.spatial)
- Statistics (scipy.stats)
- Multidimensional image processing (scipy.ndimage)

SciPy.org

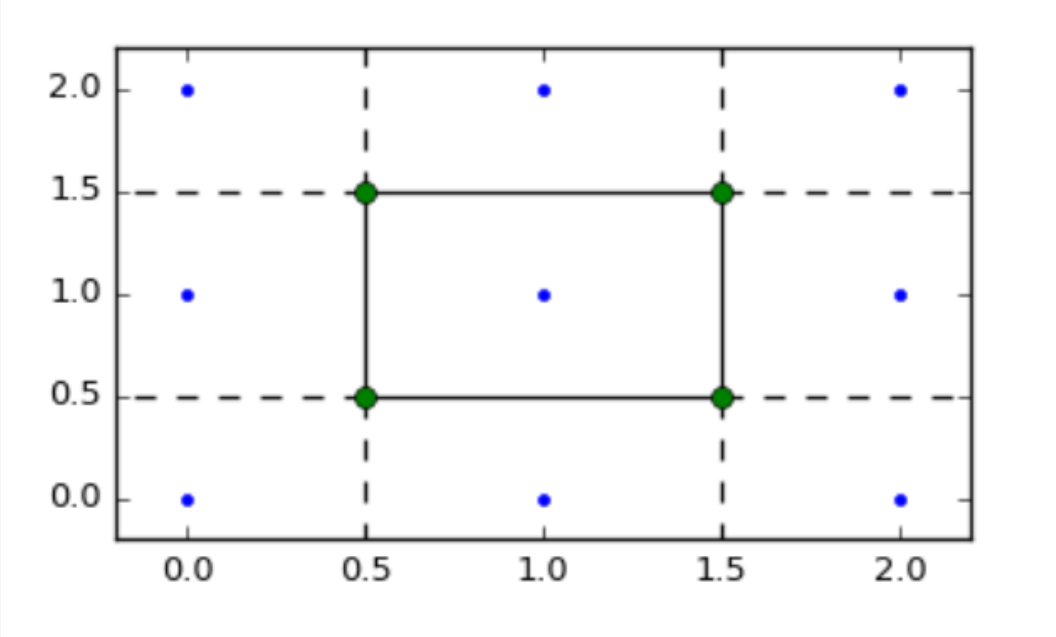
ConvexHull

```
>>> from scipy.spatial import ConvexHull, convex_hull_plot_2d
>>> points = np.random.rand(30, 2)   # 30 random points in 2-D
>>> hull = ConvexHull(points)

>>> import matplotlib.pyplot as plt
>>> plt.plot(points[:,0], points[:,1], 'o')
>>> for simplex in hull.simplices:
...     plt.plot(points[simplex, 0], points[simplex, 1], 'k-')

>>> plt.plot(points[hull.vertices,0], points[hull.vertices,1], 'r--', lw=2)
>>> plt.plot(points[hull.vertices[0],0], points[hull.vertices[0],1], 'ro')
>>> plt.show()
```



Voronoi

```
>>> points = np.array([[0, 0], [0, 1], [0, 2], [1, 0], [1, 1], [1, 2],
...                    [2, 0], [2, 1], [2, 2]])
>>> from scipy.spatial import Voronoi, voronoi_plot_2d
>>> vor = Voronoi(points)

>>> import matplotlib.pyplot as plt
>>> voronoi_plot_2d(vor)
>>> plt.show()
```



Integration

The function **quad** is provided to integrate a function of one variable between two points. The points can be ±∞ (± inf) to indicate infinite limits. For example, suppose you wish to integrate a bessel function jv(2.5, x) along the interval [0,4.5].

$$I = \int_0^{4.5} J_{2.5}(x)\, dx$$

$$x^2 \frac{d^2y}{dx^2} + x\frac{dy}{dx} + (x^2 - \alpha^2)y = 0$$

```
>>> import scipy.integrate as integrate
>>> import scipy.special as special
>>> result = integrate.quad(lambda x: special.jv(2.5,x), 0, 4.5)
>>> result
(1.1781793807832 49, 7.866317248189 9801e-09)
```

Integration

For n-fold integration, scipy provides the function **nquad**. The integration bounds are an iterable object: either a list of constant bounds, or a list of functions for the non-constant integration bounds. The order of integration (and therefore the bounds) is from the innermost integral to the outermost one.

$$I_n = \int_0^\infty \int_1^\infty \frac{e^{-xt}}{t^n}\, dt\, dx = \frac{1}{n}$$

```
>>> from scipy import integrate
>>> N = 5
>>> def f(t, x):
...     return np.exp(-x*t) / t**N
...
>>> integrate.nquad(f, [[1, np.inf],[0, np.inf]])
(0.2000000000002294, 1.2239614263187945e-08)
```

Optimization

scipy.optimize.minimize(fun, x0, args=(), method=None, jac=None, hess=None, hessp=None, bounds=None, constraints=(), tol=None, callback=None, options=None)

- Nelder-Mead
- Powell
- CG
- BFGS
- Newton-CG
- L-BFGS-B
- TNC
- COBYLA
- SLSQP
- trust-constr
- dogleg
- trust-ncg
- trust-exact
- trust-krylov

SciPy.org

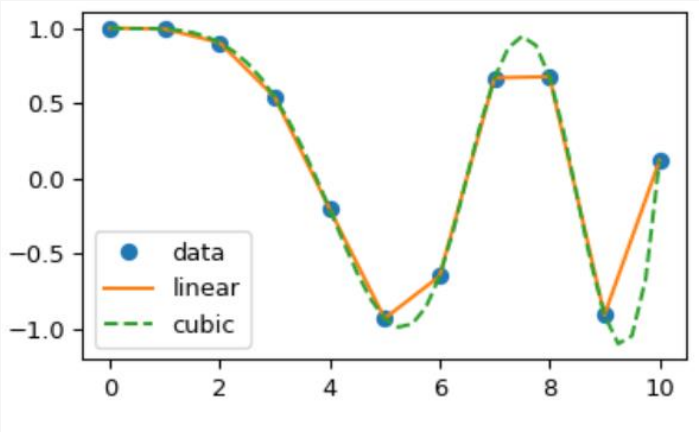
Optimization

```
>>> import numpy as np
>>> from scipy.optimize import minimize

>>> def rosen(x):
...     """The Rosenbrock function"""
...     return sum(100.0*(x[1:]-x[:-1]**2.0)**2.0 + (1-x[:-1])**2.0)

>>> x0 = np.array([1.3, 0.7, 0.8, 1.9, 1.2])
>>> res = minimize(rosen, x0, method='nelder-mead',
...                options={'xatol': 1e-8, 'disp': True})
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 339
         Function evaluations: 571

>>> print(res.x)
[1. 1. 1. 1. 1.]
```

interpolation

The **interp1d** class in **scipy.interpolate** is a convenient method to create a function based on fixed data points, which can be evaluated anywhere within the domain defined by the given data using linear interpolation. An instance of this class is created by passing the 1-D vectors comprising the data. The instance of this class defines a __call__ method and can therefore by treated like a function which interpolates between known data values to obtain unknown values (it also has a docstring for help). Behavior at the boundary can be specified at instantiation time. The following example demonstrates its use, for linear and cubic spline interpolation:

```
>>> from scipy.interpolate import interp1d
>>> x = np.linspace(0, 10, num=11, endpoint=True)
>>> y = np.cos(-x**2/9.0)
>>> f = interp1d(x, y)
>>> f2 = interp1d(x, y, kind='cubic')
>>> xnew = np.linspace(0, 10, num=41, endpoint=True)
>>> import matplotlib.pyplot as plt
>>> plt.plot(x, y, 'o', xnew, f(xnew), '-', xnew, f2(xnew), '--')
>>> plt.legend(['data', 'linear', 'cubic'], loc='best')
>>> plt.show()
```

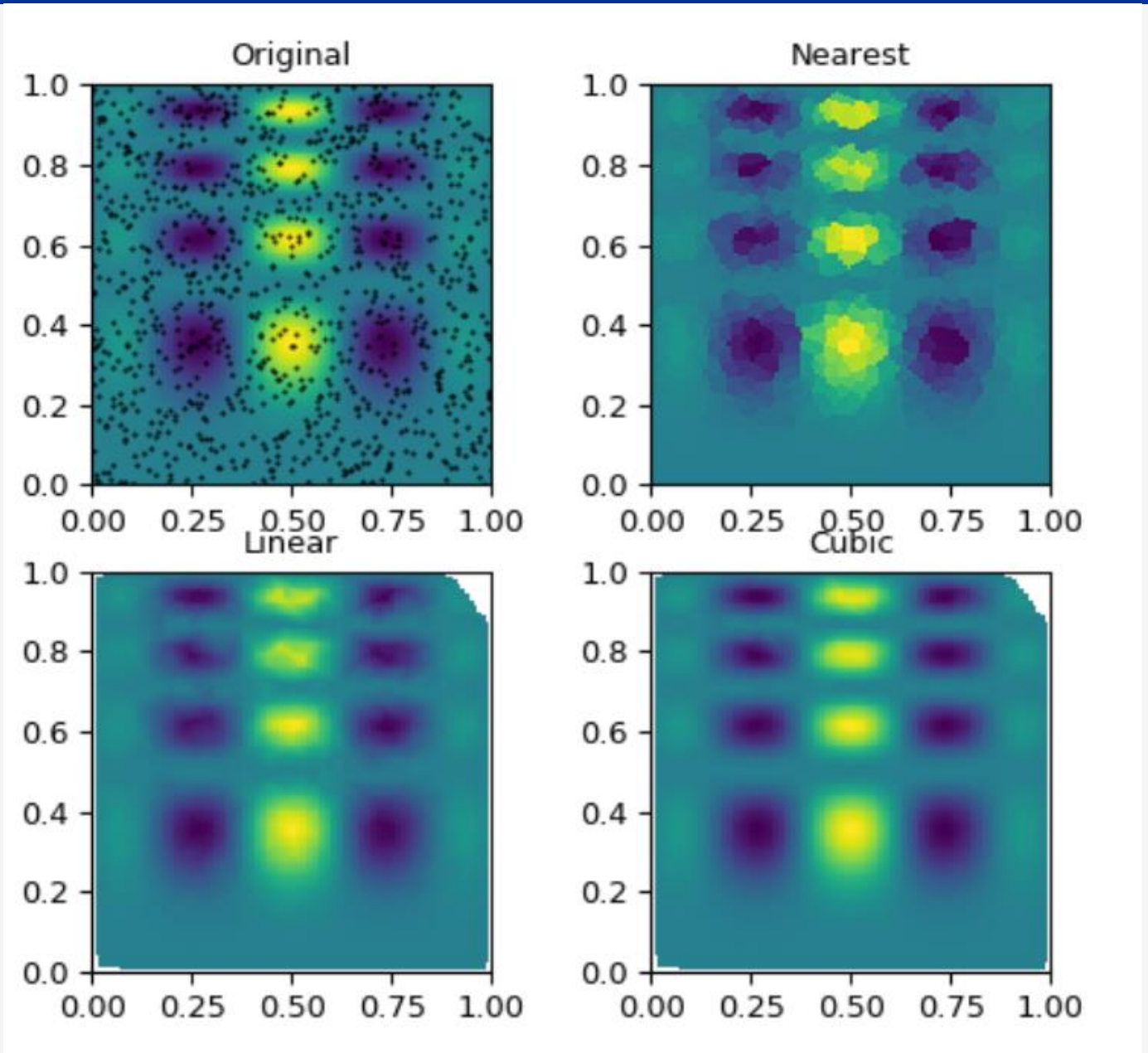


SciPy.org *Multivariate data interpolation*

Suppose you have multidimensional data, for instance, for an underlying function *f(x, y)* you only know the values at points *(x[i], y[i])* that do not form a regular grid.
Suppose we want to interpolate the 2-D function

```
>>> def func(x, y):
...     return x*(1-x)*np.cos(4*np.pi*x) * np.sin(4*np.pi*y**2)**2
>>> grid_x, grid_y = np.mgrid[0:1:100j, 0:1:200j]

>>> points = np.random.rand(1000, 2)
>>> values = func(points[:,0], points[:,1])

>>> from scipy.interpolate import griddata
>>> grid_z0 = griddata(points, values, (grid_x, grid_y), method='nearest')
>>> grid_z1 = griddata(points, values, (grid_x, grid_y), method='linear')
>>> grid_z2 = griddata(points, values, (grid_x, grid_y), method='cubic')
```

Multivariate data interpolation



Example

SymPy

SymPy is a Python library for symbolic mathematics. It aims to become a full-featured computer algebra system (CAS) while keeping the code as simple as possible in order to be comprehensible and easily extensible. SymPy is written entirely in Python.

```
>>> from sympy import *
>>> x = symbols('x')
>>> a = Integral(cos(x)*exp(x), x)
>>> Eq(a, a.doit())
Eq(Integral(exp(x)*cos(x), x), exp(x)*sin(x)/2
```

https://docs.sympy.org/

SymPy

Lambdify

This module provides convenient functions to transform sympy expressions to lambda functions which can be used to calculate numerical values very fast.

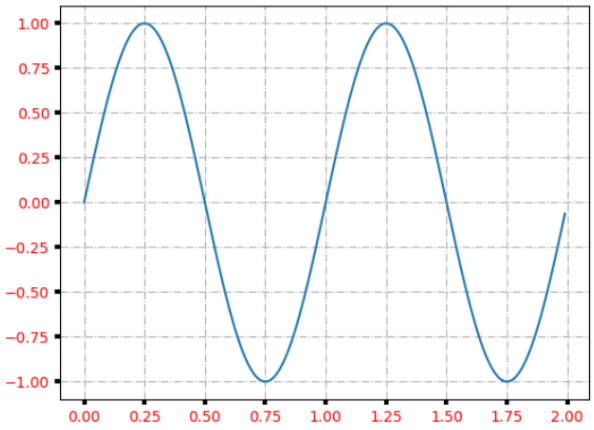
```
>>> from sympy.abc import x
>>> from sympy.utilities.lambdify import lambdify, implemented_function
>>> from sympy import Function
>>> f = implemented_function('f', lambda x: x+1)
>>> lam_f = lambdify(x, f(x))
>>> lam_f(4)
5
```

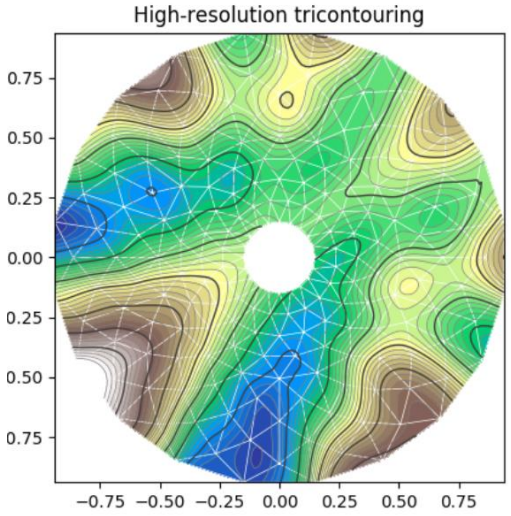
https://docs.sympy.org/

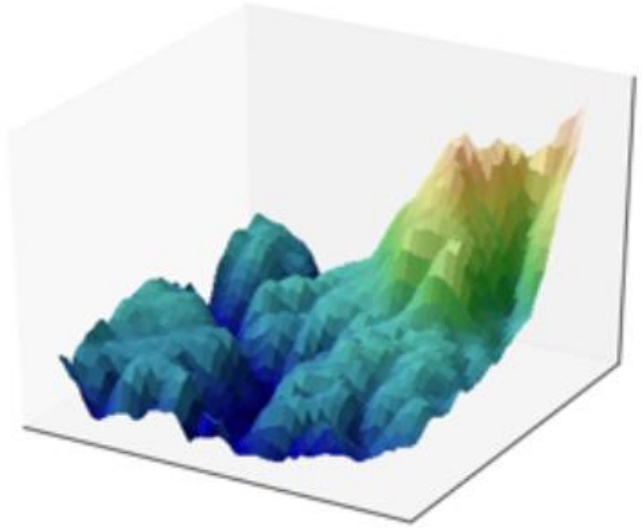
matplotlib

Version 3.1.2

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shells, the Jupyter notebook, web application servers, and four graphical user interface toolkits.







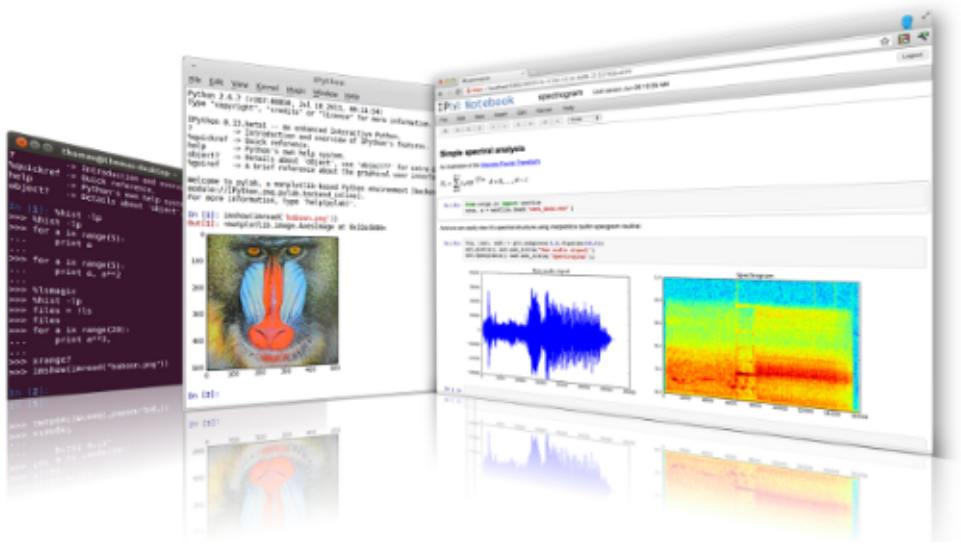
https://matplotlib.org/

IPython
Interactive Computing

Install · Documentation · Project · Jupyter · News · Cite · Donate · Books

IPython provides a rich architecture for interactive computing with:

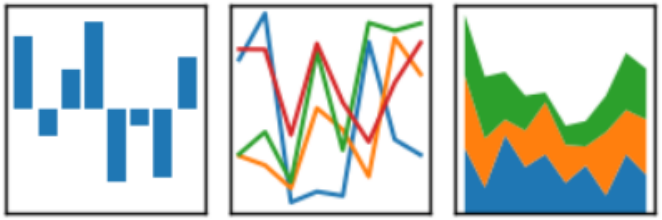
- A powerful interactive shell.
- A kernel for Jupyter.
- Support for interactive data visualization and use of GUI toolkits.
- Flexible, embeddable interpreters to load into your own projects.
- Easy to use, high performance tools for parallel computing.



https://ipython.org/

pandas

$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$



Python Data Analysis Library

pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.

pandas is a NumFOCUS sponsored project. This will help ensure the success of development of *pandas* as a world-class open-source project, and makes it possible to donate to the project.

A Fiscally Sponsored Project of

NUMFOCUS

OPEN CODE = BETTER SCIENCE

v0.25.3 Final (October 31, 2019)

VERSIONS

Release Candidate
1.0.0 - January 2020
download // docs // pdf

Release
0.25.3 - November 2019
download // docs // pdf

Development
1.0.0 - September 2019
github // docs

Previous Releases
0.25.2 - download // docs // pdf
0.25.1 - download // docs // pdf
0.25.0 - download // docs // pdf
0.24.2 - download // docs // pdf
0.24.1 - download // docs // pdf
0.24.0 - download // docs // pdf
0.23.4 - download // docs // pdf

https://pandas.pydata.org/

Overview

NetworkX is a Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks.

NetworkX provides:

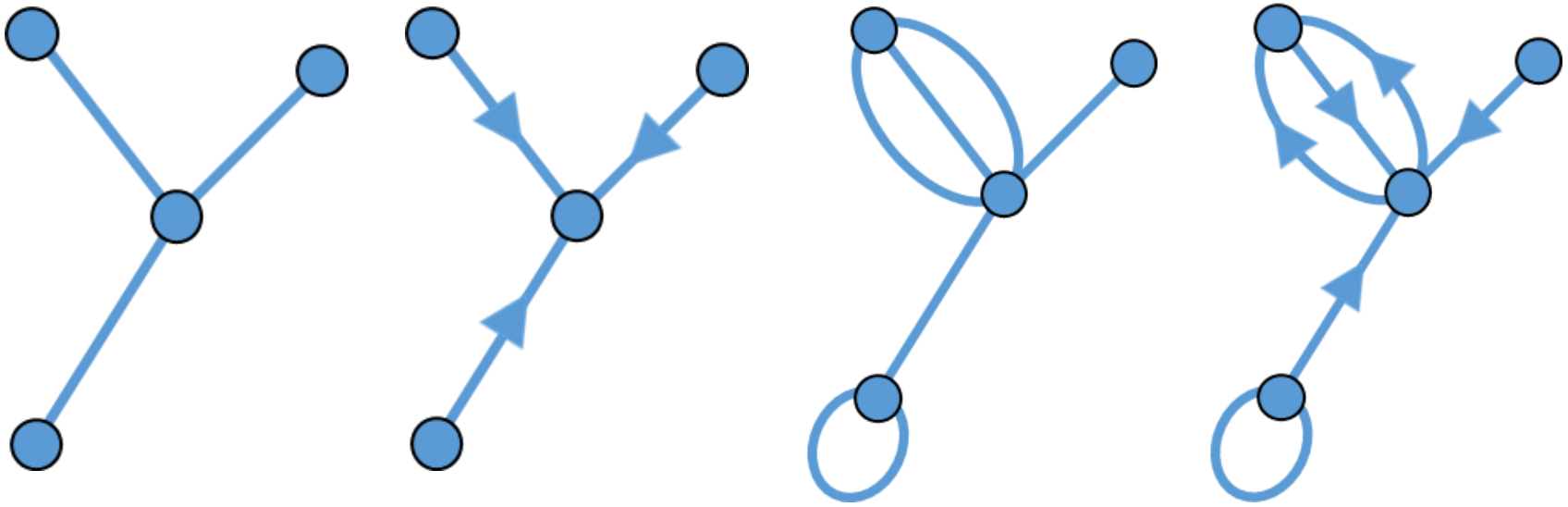
- tools for the study of the structure and dynamics of social, biological, and infrastructure networks;
- a standard programming interface and graph implementation that is suitable for many applications;
- a rapid development environment for collaborative, multidisciplinary projects;
- an interface to existing numerical algorithms and code written in C, C++, and FORTRAN; and the ability to painlessly work with large nonstandard data sets.

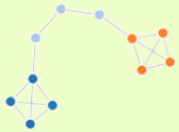
https://networkx.github.io/

NetworkX

Graph types

- **Graph** - Undirected graphs without self loops and parallel edges
- **DiGraph** - Directed graphs without self loops and parallel edges
- **MultiGraph** - Undirected graphs with self loops and parallel edges
- **MultiDiGraph** - Directed graphs with self loops and parallel edges





NetworkX

Algorithms

```
>>> import networkx.algorithms.isomorphism as iso

>>> G1 = nx.DiGraph()
>>> G2 = nx.DiGraph()
>>> nx.add_path(G1, [1,2,3,4], weight=1)
>>> nx.add_path(G2, [10,20,30,40], weight=2)
>>> em = iso.numerical_edge_match('weight', 1)
>>> nx.is_isomorphic(G1, G2)  # no weights considered
True
>>> nx.is_isomorphic(G1, G2, edge_match=em) # match weights
False
```

What is it?

Open source toolkit for cheminformatics

- Business-friendly BSD license
- Core data structures and algorithms in C++
- Python 3.x wrappers generated using Boost.Python
- Java and C# wrappers generated with SWIG
- 2D and 3D molecular operations
- Descriptor generation for machine learning
- Molecular database cartridge for PostgreSQL
- Cheminformatics nodes for KNIME (distributed from the KNIME community site: https://www.knime.com/rdkit)

https://www.rdkit.org/

Reading and Writing Molecules

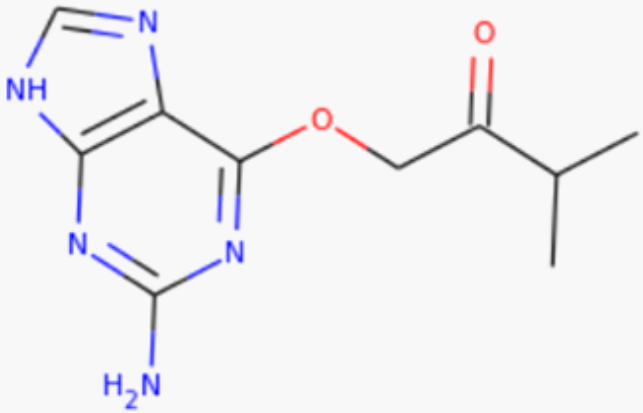
```
>>> from __future__ import print_function
>>> from rdkit import Chem
```

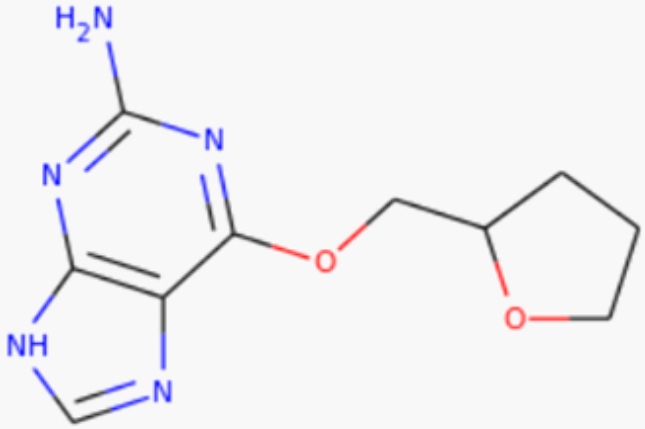
```
>>> m = Chem.MolFromSmiles('Cc1ccccc1')
>>> m = Chem.MolFromMolFile('data/input.mol')
>>> stringWithMolData=open('data/input.mol','r').read()
>>> m = Chem.MolFromMolBlock(stringWithMolData)
>>> m = Chem.MolFromMolFile('data/chiral.mol')
>>> Chem.MolToSmiles(m)
'C[C@H](O)c1ccccc1'
>>> Chem.MolToSmiles(m,isomericSmiles=False)
'CC(O)c1ccccc1'
```

```
>>> Chem.MolToSmiles(Chem.MolFromSmiles('C1=CC=CN=C1'))
'c1ccncc1'
>>> Chem.MolToSmiles(Chem.MolFromSmiles('c1cccnc1'))
'c1ccncc1'
>>> Chem.MolToSmiles(Chem.MolFromSmiles('n1ccccc1'))
'c1ccncc1'
```

Working with 2D and 3D molecules

```
>>> m = Chem.MolFromSmiles('c1nccc2n1ccc2')
>>> AllChem.Compute2DCoords(m)

>>> m = Chem.MolFromSmiles('C1CCC1OC')
>>> m2=Chem.AddHs(m)
>>> AllChem.EmbedMolecule(m2)

>>> Draw.MolToFile(ms[0],'images/cdk2_mol1.o.png')
>>> Draw.MolToFile(ms[1],'images/cdk2_mol2.o.png')
```





Substructure Searching

```
>>> m = Chem.MolFromSmiles('c1ccccc1O')
>>> patt = Chem.MolFromSmarts('ccO')
>>> m.HasSubstructMatch(patt)
True
>>> m.GetSubstructMatch(patt)
(0, 5, 6)


>>> m.GetSubstructMatches(patt)
((0, 5, 6), (4, 5, 6))
```

Maximum Common Substructure

```
>>> from rdkit.Chem import rdFMCS
>>> mol1 = Chem.MolFromSmiles("O=C(NCc1cc(OC)c(O)cc1)CCCC/C=C/C(C)C")
>>> mol2 = Chem.MolFromSmiles("CC(C)CCCCCC(=O)NCC1=CC(=C(C=C1)O)OC")
>>> mol3 = Chem.MolFromSmiles("c1(C=O)cc(OC)c(O)cc1")
>>> mols = [mol1,mol2,mol3]
>>> res=rdFMCS.FindMCS(mols)
>>> res
<rdkit.Chem.rdFMCS.MCSResult object at 0x...>
>>> res.numAtoms
10
>>> res.numBonds
10
>>> res.smartsString
'[#6]1(-[#6]):[#6]:[#6](-[#8]-[#6]):[#6](:[#6]:[#6]:1)-[#8]'
>>> res.canceled
False
```

Molecular fingerprints and Similarity

- MACCS Keys
- Atom Pairs
- Topological Torsions
- Morgan/Circular
- 2D Pharmacophore
- Pattern
- Extended Reduced Graphs

Molecular descriptors

- Gasteiger PC
- Marsili PC
- BalabanJ
- BertzCT
- Ipc
- HallKierAlpha
- Kappa1 - Kappa3
- Chi0, Chi1
- Chi0n - Chi4n
- Chi0v - Chi4v
- MolLogP
- MolMR
- MolWt
- ExactMolWt
- HeavyAtomCount
- HeavyAtomMolWt
- NHOHCount
- NOCount
- NumHAcceptors
- NumHDonors
- NumHeteroatoms
- NumRotatableBonds
- NumValenceElectrons
- NumAmideBonds
- RingCount
- FractionCSP3
- NumSpiroAtoms
- NumBridgeheadAtoms
- TPSA
- LabuteASA
- Spherocity Index
- Autocorr3D
- PEOE_VSA1 - PEOE_VSA14
- SMR_VSA1 - SMR_VSA10
- SlogP_VSA1 - SlogP_VSA12
- EState_VSA1 - EState_VSA11
- VSA_EState1 - VSA_EState10
- MQNs
- Topliss fragments
- Autocorr2D
- PMI1, PMI2, PMI3
- NPR1, NPR2
- Radius of gyration
- Inertial shape factor
- Eccentricity
- Asphericity

Example

PyCifRW

PyCIFRW provides support for reading and writing CIF (Crystallographic Information Format) files using Python. It was developed at the Australian National Beamline Facility (ANBF), run by the Australian Synchrotron Research Program (ASRP), as part of a larger project to provide CIF input/output capabilities for data collection. It is now (Mar 2017) maintained and developed within the Australian Nuclear Science and Technology Organisation (ANSTO).

https://pypi.org/project/PyCifRW/4.3/

Example

Atomic Simulation Environment

The Atomic Simulation Environment (ASE) is a set of tools and Python modules for setting up, manipulating, running, visualizing and analyzing atomistic simulations. The code is freely available under the GNU LGPL license.

ASE provides interfaces to different codes through Calculators which are used together with the central Atoms object and the many available algorithms in ASE.

Supported

https://wiki.fysik.dtu.dk/ase/

Calculations

Basic property calculations

- Atomization energy
- Equation of state (EOS)
- Finding lattice constants using EOS and the stress tensor
- Local optimization

Molecular dynamics

Global optimization

- Constrained minima hopping (global optimization)
- Optimization with a Genetic Algorithm
- Genetic algorithm Search for stable FCC alloys
- Determination of convex hull with a genetic algorithm

Calculation of reaction coordinates

- Nudged Elastic Band (NEB) method
- Dimer method

TSASE is an extension to ASE for transition state calculations. If you are looking for our Solid State NEB or Kinetic Database code, you are in the right place.

Saddle Searches

Several variaties of saddle searches methods are available in this library including solid-state dimer, lanczos, bgsd, and solid-state neb.

- Solid State Dimer
- Biased Gradient Squared Descent
- Solid State Nudge Elastic Band
- Lanczos

https://theory.cm.utexas.edu/tsase/

Example

SCIENTIFIC REPORTS

Network topological model of reconstructive solid-state transformations

Vladislav A. Blatov, Andrey A. Golov, Changhao Yang, Qingfeng Zeng, & Artem A. Kabanov

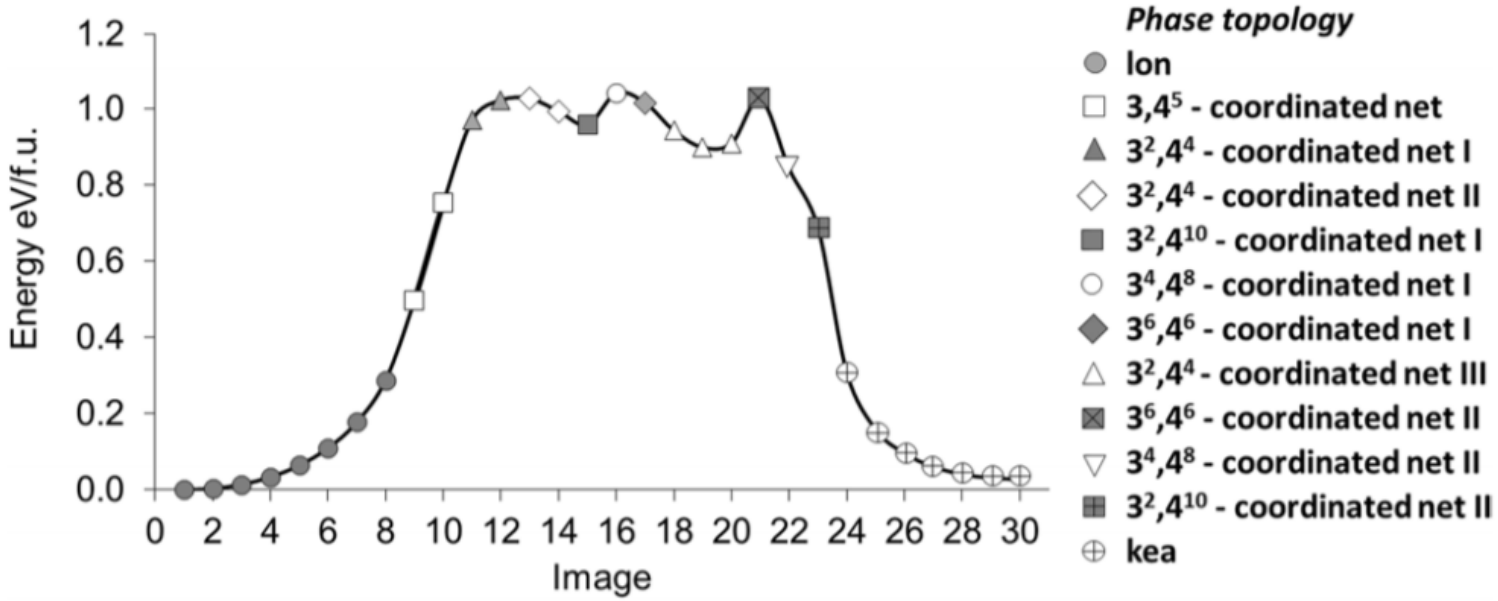
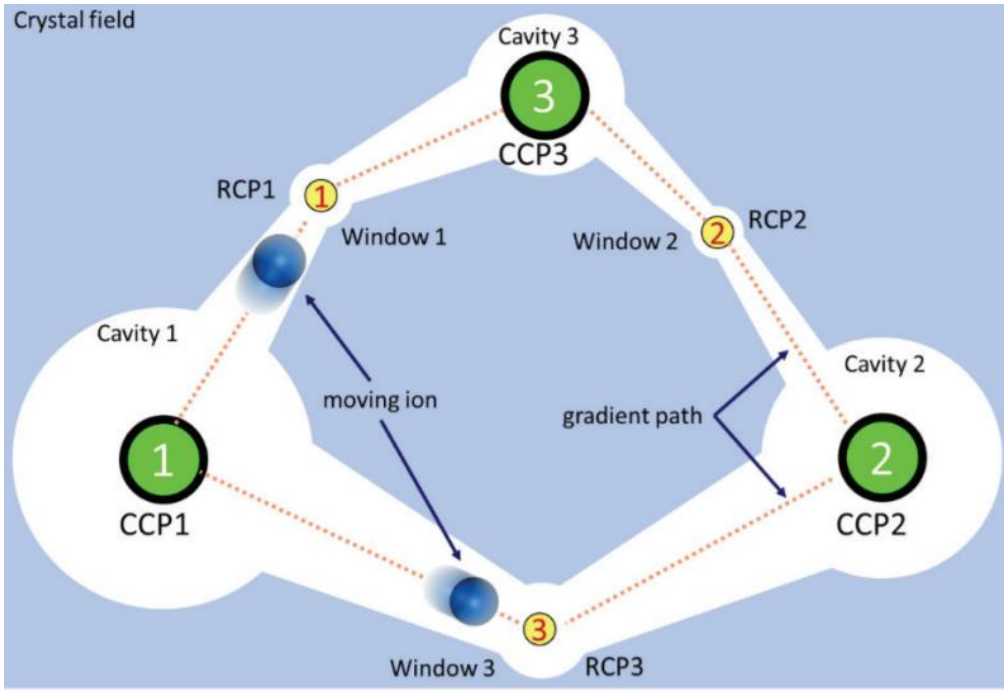
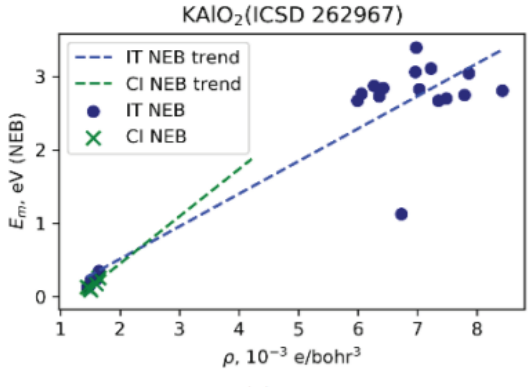


Figure 4. Energy profile for the tridymite ↔ keatite transition.

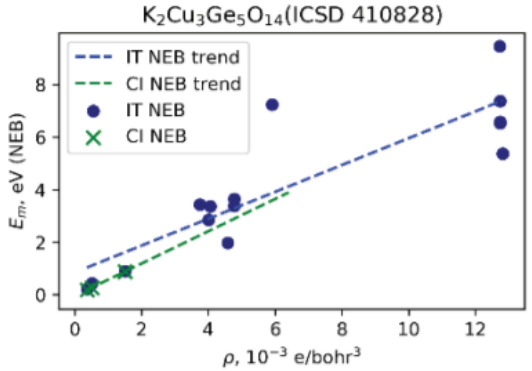
Topological analysis of procrystal electron densities as a tool for computational modeling of solid electrolytes: A case study of known and promising potassium conductors

Pavel N. Zolotarev, Andrey A. Golov, Nadezhda A. Nekrasova, and Roman A. Eremin





(a)



Thank you for your attention!