Numpy & Scipy

Created by John C.S. Lui, June 2, 2018.

NumPy & SciPy

- NumPy and SciPy are open-source add-on modules to Python that provide common mathematical and numerical routines in pre-compiled, fast functions.
- **NumPy** (Numeric Python) package provides basic routines for manipulating large arrays and matrices of numeric data
- **SciPy** (Scientific Python) package extends the functionality of NumPy with a substantial collection of useful algorithms, like minimization, Fourier transformation, regression, and other applied mathematical techniques

Scientific Python building blocks

- **Python**
- **Jupyter**: An advanced interative web tool
- **Numpy** : provides powerful numerical arrays objects, and routines to manipulate them. http://www.numpy.org/ (http://www.numpy.org/)
- **Scipy** : high-level data processing routines. Optimization, regression, interpolation, etc http://www.scipy.org/ (http://www.scipy.org/)
- **Matplotlib** : 2-D visualization, “publication-ready” plots http://matplotlib.org/ (http://matplotlib.org/)
- **Mayavi** : 3-D visualization http://code.enthought.com/projects/mayavi/ (http://code.enthought.com/projects/mayavi/)

NumPy basic

- **NumPy's** main object is the homogeneous multidimensional array. It is a table of elements (usually numbers), all of the same type, indexed by a tuple of positive integers.
- In Numpy dimensions are called **axes**. The number of axes is **rank**.
- E.g., the coordinates of a point in 3D space [1, 2, 1] is an array of rank 1, because it has one axis.
- That axis has a length of 3.
- Let say we have the following array: [[1.0, 1.0, 2.0], [0.0, 2.0, 1.0]]
  - It has rank of 2 (or 2 dimensions)
  - Its first dimension (or axis) has a length of 2
  - It second dimension (or axis) has a length of 3

Importing NumPy module

- Several ways:
  - **immport numpy**
  - **import numpy as np**
  - **from numpy import ***
- The basic building block of Numpy is **array**
- Arrays are similar to lists in Python, the function array takes two arguments:
  - the list to be converted into the array and,
  - the type of each member of the list

In [ ]:
```
import numpy as np
a  = np.array([1, 4, 5, 8], float)
print(a)
print (type(a))
```

In [ ]:
```
# We can index array just like we have done to list in Python
print(a[:2])
print(a[3])
a[3] = 100.0
print(a)
```

In [ ]:
```
# Let's try higher dimensional array
a = np.array([[1,2,3], [4,5,6]], float)  # define 2x3 array
print ('a =', a)
a[0,0] = 15.0   # we can re-assign elements in the array
a[1,0] = 12.0
print('a =',a)
```

In [ ]:
```
# we can even using slicing
a = np.array([[1,2,3], [4,5,6]], float)  # define 2x3 array
print('a[1,:]=', a[1,:])
print('a[:,2]=', a[:,2])
print('a[-1:,-2:] =', a[-1:, -2:])
```

Methods on Array

Let's consider some useful **methods** which we can apply to array

In [ ]:
```
# To find out the "dimension" of an array
a = np.array([[1,2,3], [4,5,6]], float)  # define 2x3 array

print(a.shape)
```

In [ ]:
```
# To find out the "type" of an array
print (a.dtype)
```

In [ ]:
```
# To find out the length of the first axis:
print('length of the first axis =', len(a))
print('length of the second axis =', len(a[0]))  # length of the 2nd axis
```

In [ ]:
```
# To test whether an element is in the array
a = np.array([[1,2,3], [4,5,6]], float)  # define 2x3 array

print ("Is 2 in a? ", 2 in a)  # test  for membership
print ("Is 9 in a? ", 9 in a)
```

In [ ]:
```
# To generate an array of numbers using the function range()
a = np.array(range(10), float)  # generate float array with a single item
print('float a =', a)

a = np.array([range(3), range(3)], int) # generate integer array with 2 items
print('integer a = ', a)
```

In [ ]:
```
# use reshape() method to re-arrange an array
a = np.array(range(10), float)
print('Before reshape(), a =', a)

# reshape array a
a = a.reshape(2,5)
print('After 1st reshape(), a =', a)

# reshape array a
a = a.reshape(5,2)
print('After 2nd reshape(), a =', a)
```

Array assignment is just a reference copy

Note that in NumPy, array assignment is just a reference copy.

In [ ]:
```
a = np.array(range(5), float)
b = a
print ('a=', a, 'b=', b)

a[0] = 10.0   # reassign an item in a
print ('a=', a, 'b=', b)
```

If we want to have another array, use copy method

In [ ]:
```
a = np.array(range(5), float)
b = a
c = a.copy()
print ('Before:', 'a=', a, 'b=', b, 'c=', c)

a[0] = 10.0   # reassign an item in a
print ('After: ', 'a=', a, 'b=', b, 'c=',c)
```

Define an array and fill it with some entries

In [ ]:
```
a = np.array(range(5), float)
print('a =', a)
a.fill(0)   # use fill method to initialize the array
print('a =', a)
a.fill(100.0)
print('a =', a)
```

In [ ]:
```
a = np.array([range(3), range(3)], int)  # generate array with 2 items
print('a =', a)
a.fill(0)   # use fill method to initialize the array
print('a =', a)
a.fill(100.0)
print('a =', a)
```

Transpose method

In [ ]:
```
a = np.array([range(5), range(5)], int)  # generate array with 2 items
print('a =', a)

a = a.transpose()
print('a =', a)
```

Concatenate method

In [ ]:
```
a = np.array([1,2], float)          # float type
b = np.array([3,4,5], float)        # float type
c = np.array([7,8,9,10], int)       # integer type
d = np.concatenate((a,b,c))
print('a =',a)
print('b =',b)
print('c =',c)
print('d =',d)
```

arange method

arange method is similar to *range()* function except it returns an array.

In [ ]:
```
a = np.array(range(5), float)
print('a = ', a)
print('type of a: ', type(a))

b = np.arange(5, dtype=float)
print('b = ', b)
print('type of b: ', type(b))
```

zeros and ones

We use **zeros** and **ones** method to initialize an array

In [ ]:
```
a = np.zeros(7,dtype=int)
print('a=',a)

b = np.zeros((2,3),dtype=int)
print('b=',b)

c = np.ones((3,2),dtype=float)
print('c=',c)
```

We use zeros_like and ones_like functions to mimic and initialize arrays

In [ ]:
```
a = np.array([[1,3,3], [4,5,6]],int)
b = np.zeros_like(a)  # create a new array b with all zeros
c = np.ones_like(a)   # create a new array c with all ones
print('a=',a)
print('b=',b)
print('c=',c)
```

Creating identity matrix and diagonal matrix

- use identity method
- use eye method: which returns matrices with ones along the k^{th} diagonal entries

In [ ]:
```
a = np.identity(5, dtype=float)  # create a 5x5 identity matrix
print('a=',a)

a = np.eye(5, k=1, dtype=float)  # create a 5x5 matrix with 1st diagonal being 1
print('a=',a)
```

Array mathematics

Of course we can do addition, subtraction, multiplication and division,...etc

In [ ]:
```
a = np.array([1,2,3,4,5],int)
b = np.array([5,4,3,2,1],float)
print('a =', a, '; b =', b)
print('a+b =', a+b)
print('a-b =', a-b)
print('a*b =', a*b)
print('a/b =', a/b)
print('a%b =', a%b)
print('a**b =', a**b)
```

In [ ]:
```
# for higher dimenion arrays, it remains to be an element-wise operation
c = np.array([[1,2,3,4,5],[1,1,1,1,1]],int)
d = np.array([[5,4,3,2,1],[2,2,2,2,2]],float)
print('\nc =', c)
print('d =',d)
print('c+d =', c+d)
print('c-d =', c-d)
print('c*d =', c*d)
print('c/d =', c/d)
print('c%d =', c%d)
print('c**d =',c**d)
```

The "broadcast" feature of NumPy

Arrays that do not match in the number of dimensions will be **broadcasted** (or adjusted to fit with appropriate dimension)

In [ ]:
```
a = np.array([[1,2], [3,4], [5,6]], int)
b = np.array([-1,3], float)
print('a =', a)
print('b =', b, end='\n\n')
print('a+b =', a+b, end='\n\n')
print('a*b =', a*b, end='\n\n')
```

Math libraries in NumPy

There are **many libraries** like *abs, sign, sqrt, log, log10, exp, sin, cos, tan, arcsin, arccos, arctan, sinh, cosh, tanh, arcsinh, arccosh, arctanh*,....
Make sure to read the documentation

In [ ]:
```
a = np.array([1,4,9,16,25],float)

print ('Square root of a =', np.sqrt(a)) # use sqaure root
print ('Sign of a =', np.sign(a)) # use sign
print ('exp of a =', np.exp(a)) # use exponential
print ('log of a =', np.log(a)) # use log
print ('log10 of a =', np.log10(a)) # use log10
```

In [ ]:
```
a = np.array([1.1,4.3,9.8,16.5,25.6],float)

print('a =', a)
print ('floor of a =', np.floor(a)) # use floor
print ('ceil of a =', np.ceil(a)) # use ceil
print ('rint of a =', np.rint(a)) # use rint
```

Various constants in NumPy

In [ ]:
```
print('Pi is = ', np.pi)
print('e is = ', np.e)
```

Array iteration in NumPy

In [ ]:
```
a = np.array([1,4,5], dtype=int)

for num in a:
    print('number = ', num)
```

In [ ]:
```
a = np.array([[1,2],[3,4],[5,6]], dtype=int)

for num in a:
    print('number = ', num)
```

In [ ]:
```
a = np.array([[1,2],[3,4],[5,6]], dtype=int)

for [num1, num2] in a:
    print('num1 = ', num1, '; num2 = ', num2, '; num1*num2 = ', num1*num2)
```

Basic Array operations

In [ ]:
```
a = np.array([2,4,3], dtype=float)
b = np.array([[1,2],[3,4]], dtype=float)

print('a =', a)
print('b =', b, end='\n\n')
print('element sum of a = ', a.sum())  # sum all elements
print('element sum of b = ', b.sum())  # sum all elements
print('element prduct of a = ', a.prod())  # multiply all elements
print('element product  of b = ', b.prod())  # multiply all elements
```

In [ ]:
```
## Another way is to use NumPy method with array as argument
print('element sum of a = ', np.sum(a))  # sum all elements
print('element sum of b = ', np.sum(b))  # sum all elements
print('element prduct of a = ', np.prod(a))  # multiply all elements
print('element product  of b = ', np.prod(b)) # multiply all elements
```

In [ ]:
```
# to sort all entries in an array
a = np.array([6, 2, 5, -1, 0],float)
print('a =', a)
print('sorted form of a =', sorted(a))      # sort a but not to alter the array a
print('clipped form of a =', a.clip(0,4.1)) # specify lower/upper bound
```

In [ ]:
```
# finding unique entries in an array
a = np.array([1, 1, 2, 2, 3, 4, 4, 5, 5, 5],float)
print('a =', a)
print('unqiue entries of a :', np.unique(a))
```

In [ ]:
```
# finding diagonal entries in an array
a = np.array([[1,2,3],[4,5,6],[7,8,9]], int)
print('a =', a)
print('diagonal entries of a are :', a.diagonal())
```

Array comparison & testing

In [ ]:
```
a = np.array([1, 3, 0], float)
b = np.array([0, 3, 2], float)
print('a=',a, '; b=', b)

print('Is a > b: ', a>b)   # a>b returns an array of boolean
print('Is a == b:', a==b)
print('Is a <= b:', a<=b)
```

In [ ]:
```
# You can use methods like any or all to test condition
c = a > b    # c is now an array of boolean
print('c =', c)
print('There is at least one "True" in c: ', any(c))
print('all entries in c are "True" : ', all(c))
```

In [ ]:
```
# use of logical_and, logical_or and logical_not in array
a = np.array([1,3,0], float)
print('a =', a)
b = np.logical_and(a>0, a<3)
print('Are entries in a > 0 AND a < 3: ', b)
c = np.logical_not(b)
print('Use of logical_not in a: ', c)
print('Use of logical_or: ', np.logical_or(b,c))
```

Use of method where() in NumPy

where forms a new array from two arrays of equivalent size using a Boolean filter to choose between elements of the two. Its basic syntax is

np.where (boolarray, truearray, falsearray)

In [ ]:
```
a = np.array([1, 3, 0], float)
b = np.where(a > 0, a+1, a)
c = np.where(a>0, 5.0, -1.0)
print('a =', a)
print('b =', b)
print('c =', c)
```

In [ ]:
```
## It is also possible to test whether or not values are NaN ("not a number") or finite
a = np.array([2, np.NaN, np.Inf], float)
print('a =', a)
print('Entry is not a number :', np.isnan(a))
print('Entry is finite :', np.isfinite(a))
```

Finding statistics in an array

In [ ]:
```
# We want to find the mean and variance of a series
a = np.array([2, 1, 3, 10.0, 5.3, 18.2, 16.3],dtype=float)
mean = a.sum()/len(a)
print('mean of a =', mean)
print('mean of a =', a.mean(), end='\n\n')
print('variance of a =', a.var())
print('my standard deviation of a =', np.sqrt(a.var()))
print('standard deviation of a =', a.std())
```

In [ ]:
```
# We want to find the min or max or argmin or argmax in a series
a = np.array([2, 1, 3, 10.0, 5.3, 18.2, 16.3],dtype=float)

print('a =', a)
print('minimum element in a =', a.min())
print('minimum occurs in index:', a.argmin())
print('maximum element in a =', a.max())
print('maximum occurs in index:', a.argmax())
```

In [ ]:
```
# We can even control which axis to take the statistics
a = np.array([[0,2],[3,-1],[3,5]], dtype=float)
print('a =', a)

print('mean in axis 0 = ', a.mean(axis=0))
print('mean in axis 1 = ', a.mean(axis=1))
print('min  in axis 0 = ', a.min(axis=0))
print('min  in axis 1 = ', a.min(axis=1))
print('max  in axis 0 = ', a.max(axis=0))
print('max  in axis 1 = ', a.max(axis=1))
```

Array item selection and manipulation

In [ ]:
```
a = np.array([[6,4], [5,9]], float)  # define a 2x2 array
print('a=', a)

b = a >= 6  # define a boolean array with the same dimension as 'a'
c = a[b]    # select entries in 'a' which are correspondinly true in 'b'
print('b = ', b, '. Type of b:', type(b))
print('c = ', c, '. Type of c:', type(c))
```

In [ ]:
```
a = np.array([[6,4], [5,9]], float)  # define a 2x2 array
print('a=', a)

b = (a >= 6)
print('b = ', b, '. Type of b:', type(b))

c = a[b]
print('c = ', c, '. Type of c:', type(c))
```

In [ ]:
```
a = np.array([[6,4], [5,9]], float)  # define a 2x2 array
print('a=', a)

b = a[np.logical_and(a > 5, a <9)]
print('b = ', b, '. Type of b:', type(b))
```

In [ ]:
```
a = np.array([2, 4, 6, 8], float)
print('a=', a, '; b=', b)
b = np.array([0, 0, 1, 3, 2, 1], int)  # it has to be an integer array
c = a[b]   # create array c
print('c = ', c, '. Type of c:', type(c))
```

Multidimensional array selection

For multidimensional arrays, we have to use multiple one-dimensional integer array to the selection bracket, **one for each axis**

In [ ]:
```
a = np.array([[1,4], [9,16]], float)
b = np.array([0, 0, 1, 1, 0], int)
c = np.array([0, 1, 1, 1, 1], int)
d = a[b,c]

print('a =', a)
print('b =', b)
print('c =', c)
print('d =', d, '; type of d is:', type(d))
```

take method

The function **take** is available to perform selection with integer arrays

In [ ]:
```
a = np.array([2, 4, 6, 8], float)
b = np.array([0, 0, 1, 3, 2, 1], int)

print('a =', a, '; b=', b)
print('a.take(b) = ', a.take(b))
```

put method

The function **put** takes values from a soure array and place them at specified indices

In [ ]:
```
a = np.array([0, 1, 2, 3, 4, 5], float)
print('a=', a)
b = np.array([9, 8, 7], float)

a.put([0, 3, 5], b)  # put entries of b in a according to a given indices
print('a=', a)
```

In [ ]:
```
# for put method, it can repeat if necessary
a = np.array([0, 1, 2, 3, 4, 5], float)
print('a=',a)

a.put([0,2,3], 5)
print('a=', a)
```

Vector and matrix mathematics

We can do *dot* products and matrix multiplication,...etc

In [ ]:
```
a = np.array([1,2,3], float)
b = np.array([0,1,1], float)

print('a=', a, '; b=', b)
print('a * b =', np.dot(a,b))
```

In [ ]:
```
a = np.array([[1,2], [3,4]], float)
b = np.array([2,3], float)
c = np.array([[1,1], [4,0]], float)
print('a=', a, end='\n\n')
print('b=', b, end='\n\n')
print('c=', c, end='\n\n')
print('b*a =', np.dot(b,a))
print('a*b =', np.dot(a,b))
print('a*c =', np.dot(a,c))
print('c*a =', np.dot(c,a))
```

inner, outer and cross prodcuts of matrices and vectors

In [ ]:
```
a = np.array([1,4,0], float)
b = np.array([2,2,1], float)
print('a=', a, '; b=', b)
print('np.outer(a,b)=', np.outer(a,b))
print('np.inner(a,b)=', np.inner(a,b))
print('np.cross(a,b)=', np.cross(a,b))
```

Determinants, inverse and eigenvalues

In [ ]:
```
a = np.array([[4, 2, 0], [9, 3, 7],[1, 2, 1]], float)
print ('a =', a)
print('determinant of a is: ', np.linalg.det(a)) # get determinant
vals, vecs = np.linalg.eig(a)   # get eigenvalues/engenvectors
print('eigenalues: ', vals)
print('eigenvectors: ', vecs)

b = np.linalg.inv(a)     # compute inverse of a
print('inverse of a = ', b)

print("let's check, a * b = ", np.dot(a,b))
```

Singular Value Decomposition (SVD)

In [ ]:
```
a = np.array([[1,3,4], [5,2,3]], float)
U, s, Vh = np.linalg.svd(a)   # get SVD of a
print ('U is:', U)
print('s is:', s)
print('Vh is:', Vh)
```

Polynomail mathematics

Given a set of roots, it is possible to show the polynomial coefficient using the *poly* method in NumPy.

Let say for the following polynomial: $x^4 - 11x^3 + 9x^2 + 11x - 10$. If we know the roots are (-1, 1, 1, 10), we use *poly* method to find the coefficient.

In [ ]:
```
print('Polynomial coefficients are: ', np.poly([-1, 1, 1, 10]))
```

Given a set of coefficients in a polynomail, we can find the roots also via *roots* method. Let say we have the following polynomail:
$x^3 + 4x^2 - 2x + 3$.

In [ ]:
```
np.roots([1, 4, -2, 3])   # get roots
```

We can do polynomail integration via the *polyint* method. For example, we have the following polynomial:
$x^3 + x^2 + x + 1$

If we integrate it, we should have $x^4/4 + x^3/3 + x^2/2 + x + C$, where C is a constant.

In [ ]:
```
np.polyint([1,1,1,1])  # perform integration on a polynomial (specify coefficients)
```

In [ ]:
```
np.polyder([1/4, 1/3, 1/2, 1, 0])  # perform derivative of polynomial (specify coefficients)
```

Note that there are other polynomail methods

- polyadd
- polysub
- polymul
- polydiv
- polyval

Read the documentation of Numpy

Evaluate polynomail at a given point

Let's say we want to evaluate $x^3 - 2x^2 + 2$ at $x = 4$.

In [ ]:
```
np.polyval([1,-2,0, 2], 4)
```

Curve fitting

We can use *polyfit()* method, which fits a polynomial of specified order to a set of data using the *least-square method*.

In [ ]:
```
x = [1, 2, 3, 4, 5, 6, 7, 8]
y = [0, 2, 1, 3, 7, 10, 11, 19]

# use polyfit to find the least square fit using polynomail of degree 2
np.polyfit(x,y,2)     # it returns all coefficients
```

Statistics

We can use *mean, median, var, stad* methods to find the statistics of vectors or matrices

In [ ]:
```
a = np.array([1,4,3,8,9,2,3], float)  # find median
print("sorted a is: ", sorted(a))
print('median of a: ', np.median(a))
```

In [ ]:
```
# find correlation coefficient matrix (corrcoef)
a = np.array([[1,2,1,3], [5,3,1,8]], float)
c = np.corrcoef(a)
print('Correlation coefficient matrix: \n', c)
```

In [ ]:
```
# find covariane of data (cov)
print('Covariance of data: \n', np.cov(a))
```

Use random number generators in NumPy

We need random number generators in simulation, statistical analysis and machine learning tasks.

In [ ]:
```
# uniformly generate 5 random number in [0.0, 1.0)
np.random.rand(5)
```

In [ ]:
```
# random numbers for array, and use of *reshape()* method
print('A 2x3 matrix with random # between [0, 1.0):\n',
       np.random.rand(2,3))
```

In [ ]:
```
# or we can use reshape()
print('A 2x3 matrix with random # between [0, 1.0):\n',
       np.random.rand(6).reshape(2,3))
```

In [ ]:
```
# generate a SINGLE random number between [0, 1)
np.random.rand()
```

In [ ]:
```
# generate a random INTEGER in the range [min, max]
np.random.randint(5,10)
```

In [ ]:
```
# Generate a random number from a Poisson distribution with a mena of 6.0
np.random.poisson(6.0)
```

In [ ]:
```
# Generate a random number from a Normal distribution with a mena and standard deviation
np.random.normal(1.5, 4.0)
```

In [ ]:
```
# Generate a random number from a standard normal distribution
np.random.normal()
```

In [ ]:
```
# Draw multiple values using the *size* argument
np.random.normal(size=5)
```

SciPy

- Greatly extends the functionality of NumPy
- We can use "import scipy" to import the module
- SciPy has **many packages**
- To explore, do **help(scipy)**
- scipy.constants: many mathematical & physical constants
- scipy.speical: many special functions like gamma, beta, bessel,..etc
- scipy.integrate: perform numerical integration using methods like trapezoidal, Simpson's Romberg,..etc
- scipy.optimize: minimization and maximization routines
- scipy.linalg: linear algebra routines (broader than NumPy)
- scipy.sparse: routines for working with large and sparce matrices
- scipy.interpolate: interpolation routines
- scipy.fftpack: Fast Fourier transform routines
- scipy.signal: signal processing routines, e.g., convolution
- scipy.stats: Huge library of various statistical distributions and functions
- scipy.ndimage: n-dimensioanl image processing routines
- scipy.cluster: Vector quantization and Kmeans
- scipy.io: Data input and output
- scipy.spatial: Spatial data strcutures and algorithms

Let's illustrate some of them

In [ ]:
```
import numpy as np
from scipy import stats
from scipy import io as spio

a = np.ones((3,3))  # generate a 3x3 matrix with all 1's

# save a to a MatLab file
spio.savemat('file.mat', {'a': a})  # savemat expects a dictionary

# load the file to another variable
data = spio.loadmat('file.mat', struct_as_record=True)
print('The matrix a in data is: \n', data['a'])
```

SciPy: Linear Algebra (scipy.linalg)

In [ ]:
```
# Matrix determinant
import numpy as np
from scipy import linalg
arr = np.array([[1,2], [3,4]], float)
print('arr =\n', arr)
print("arr's determinant is: ", linalg.det(arr))
```

In [ ]:
```
# Matrix inverse
arr_inv = linalg.inv(arr)
print("arr's inverse is:\n", arr_inv)

# Let's do a test
print("\narr * arr_inv is:\n", np.dot(arr, arr_inv))
```

Let's examine the optimizatoin and search routines in SciPy

In [ ]:
```
import numpy as np
from scipy import optimize       # import optimization library
import matplotlib.pyplot as plt   # import ploting routine
```

```
# Define function
def f(x):
    return x**2 + 10*np.sin(x)


# plot it

x = np.arange(-10, 10, 0.1)
plt.plot(x, f(x))
plt.show()


# This function has a global minimum round -1.3
# and a local minimum around 3.8

# Conduct a gradient descent from zero to find minimum
print ('find minima starting from 0:')
optimize.fmin_bfgs(f, 0)   # start from 0 and see it finds local min only

# conduct gradient descent from 3 to find minimum
print ('find minima starting from 3:')
optimize.fmin_bfgs(f,3)


# To find the global optimal, we use scipy.optimize.basinhopping()
# which combines a local optimizer with stochastic sampling of
# starting points for the local optimizer

print ('find global minimum')
minimizer_kwargs = {"method": "BFGS"}
ret = optimize.basinhopping(f, 0.0, \
          minimizer_kwargs= minimizer_kwargs, niter=200)
print ('global minimum: x = %.4f', ret.x, 'f(x0)= %.4f', ret.fun)
```

In [ ]:
```
## Finding the roots of a scalar function
import numpy as np
from scipy import optimize   # import optimization package

# define function
def f(x):
    return x**2 + 10*np.sin(x)

root = optimize.fsolve(f,1)   # our initial guess
print("Guess from 1: ", root)

root = optimize.fsolve(f, -2.5) # another guess
print("guess from -2.5:", root)
```

In [ ]:
```
# Let's see some optimization
import numpy as np
from scipy import optimize

# define function
def f(x):
    return x**2 + 10*np.sin(x)

xdata = np.linspace(-10,10, num=20) # generate 20 pts of x
ydata = f(xdata) + np.random.rand(xdata.size) # generate 20 points of y

# define another function
def f2(x, a, b):
    return a + x**2 + b*np.sin(x)

# use scipy.optimize.curve_fit()
guess = [2,2]  # initial guess
params, params_covariance = optimize.curve_fit(f2, xdata, ydata, guess)

print("params =", params)
```

In [ ]: