OSSP - NumPy

The Quintessential Python Package

by Tristan Hunt

Overview

General purpose software for scientific computing, various linear algebra applications enabled by n-dimensional array data structure (ndarray).

NumPy is a sponsored project of the non-profit NumFOCUS, which is dedicated to developing open-source software for various public/private problem solving applications.

The project also receives funding from other sponsors: Gordon & Betty Moore Foundation, Alfred P. Sloan Foundation, Chan Zuckerberg Initiative, and Tidelift.

Partnerships/Users

The following institutional partners support NumPy and employ people to contribute to its development:

UC Berkeley, QuanSight, Nvidia

Anyone with a need for numerical computation in Python development can use the library and its functions, it is commonplace in Machine Learning projects done in Python.

The Pandas library is built upon NumPy and dependent on it.

Performance

The software strives for performance approaching similar functions in C/C++ standards, with Pythons improved syntax and usability (as described in ReadMe file).

The Accuracy is important to all whose applications depend on NumPy, which is why various independent companies contract people to continue updating its performance and insure accuracy…

Automated testing exists to ensure changes are accurate.

NumPy Roots/Interpolation

numpy.roots() will return the roots of any polynomial represented in array form.

- For functions with non-real roots, imaginary roots are given…

numpy.interp() will return an interpolated function on a 1D arrays of values.

Looking for the implementations of these functions proved quite challenging, they appear to be achieved with conglomerates of extremely basic functionalities written in C.

Look for: Rootfinding, Interpolation, Regression

How do concepts like accuracy, conditioning, stability, and cost appear?

* If the software is accurate, what does accuracy mean? Could you make a plot, say accuracy vs cost? How would you label the axes to make it relevant to a stakeholder?

* Would different stakeholders want different axes (because they care about different things)?

* Are there modeling decisions made in the interest of good conditioning? Are there algorithmic choices made for stability?

stability and conditioning

Example/Uses

Here is an implementation of a function to find Correlation Coefficients I implemented previously using NumPy…

```python
import numpy as np

#10-vectors of weather forecasts in 3 different cities (highs in F)
#Forecast 1: Charleston
char = np.array([85, 84, 86, 89, 89, 89, 86, 81, 83, 82])
#Forecast 2: Boulder
boul = np.array([78, 86, 88, 86, 65, 68, 74, 74, 74, 74])
#Forecast 3: Rio de Janeiro
rdj = np.array([70, 71, 74, 79, 75, 78, 70, 71, 75, 78])

def correlationCoefficient(a, b):
    if(a.size != b.size):
        return "vectors must have same dimension"
    ones = np.ones(a.size, dtype=int)
    #find demeaned vectors
    a_d = a - np.dot(np.average(a), ones)
    b_d = b - np.dot(np.average(b), ones)
    #calculate correlation coefficient 'p'
    p = np.dot(a_d, b_d)/(np.linalg.norm(a_d)*np.linalg.norm(b_d))
    return p

print(correlationCoefficient(char, boul))    #output --> -0.1, small negative correlation
print(correlationCoefficient(char, rdj))     #output --> 0.4, somewhat significant positive correlation
print(correlationCoefficient(boul, rdj))     #output --> -0.1, small negative correlation
```

```
-0.10070845669202841
0.39610389610389607
-0.1137958610290461
```

Comparing Runtime of Algorithms… A Start

Here I implemented an inner product function and attempted to compare its runtime to the NumPy inner product function with varying vector sizes…

With this simple an implementation, no difference was observed (I doubt there is any way of making this IP more efficient). An interesting future project would be to implement increasingly complex functions locally and compare their runtime to NumPy… this cost could easily be graphed and compared to assess the true value of NumPy's use of C for faster functionality.

```python
import numpy as np
import time
import datetime

def local_inner_prod(vec1, vec2):   #takes 2 vectors and returns their inner product (a scalar value)
    if vec1.size == vec2.size:
        ip = 0
        for i in range(vec1.size):    #
            ip += vec1[i] * vec2[i]
        return ip
    else:
        return("vectors must be same size to compute inner product")

def np_inner_prod(vec1, vec2):
    return np.inner(vec1, vec2)


#_____________COMPARISONS_____________#
for i in range(10000):
    vec1 = np.random.rand(i)
    vec2 = np.random.rand(i)
    if(local_inner_prod(vec1, vec2) == np_inner_prod(vec1, vec2)):  #can move function into variable and only call once if runtime too long
        print("_____________________________", "i = ", i, "_____________________________________________________")
        starttime1 = datetime.datetime.now()
        print(local_inner_prod(vec1, vec2))
        endtime1 = datetime.datetime.now()
        print("Time to complete local ip function: ", endtime1 - starttime1)

        starttime2 = datetime.datetime.now()
        print(np_inner_prod(vec1, vec2))
        endtime2 = datetime.datetime.now()
        print("Time to complete local ip function: ", endtime1 - starttime1)
        print("_____________________________________________________________________________________")
    else:
        print("The functions did not return the same value")
```