

31/03/2019

Google Summer of Code Project **Proposal**



Applying Numba Project wide

Aniruddha Banerjea
Bits Pilani, India

About Me:

It is a pleasure to introduce myself to all of you. My name is Aniruddha Banerjea and I am a twenty year old college senior at Birla Institute of Technology and Science, Pilani, India pursuing a major in Chemical Engineering. During my sophomore year, I started using Python for machine learning and ever since the field of Data Science has been of great interest to me. I came across ArviZ a few months and I have been interested in it's development ever since.

Contact Info

Name: Aniruddha Banerjea

Phone No: +91 9983614078

Mail: aniruddhab8@gmail.com

Github: <https://github.com/Ban-zee>

Project Description

Abstract:

ArviZ is a python package for exploratory analysis of Bayesian models. ArviZ can be used to plot Numpy arrays, dictionaries of arrays and has built in support for PyMC3, Pystan, Pyro and Emcee. ArviZ uses a number of dependencies, one of them being Numba. Numba is a JIT(just in time) compiler that compiles a given section of the code (specified by the user) at a given time instead of compiling the entire code at once. The aim of the project is to identify the possible bottlenecks in the code in order to speed up the overall code execution by applying Numba project wide. Another part of this project is to vectorize the functions (specifically for stats and diagnostics) using xarray ufuncs to broadcast across xarrays.

Current state of the project:

As mentioned in the abstract above, ArviZ includes Numba as a conditional dependency and a conditional jit function has been included in arviz/utils.py for systems in which Numba is pre-installed.

The aim of the proposed project is to extend this functionality all across the ArviZ codebase with particular emphasis on stats and diagnostics.

The Project:

The most important part of this project is to identify the possible bottlenecks in the code base and then apply Numba to speed up the execution. The most convenient way of identifying the bottleneck would be to use a code profiler. In my case, I have decided to use default python cProfiler for the aforementioned purpose.

After identifying each bottleneck, a decision needs to be made on whether to use Numba decorators on that particular function or not. Applying Numba on a specific function can generally be accomplished by using it's collection of decorators like `jit`, `njit` and `generated_jit`, however in certain cases, certain changes need to be in order to apply Numba on the function.

The code snippet below describes the situation aptly:

Over here I have displayed the difference in execution time with and without Numba of `stats.bfmi` function.

```
import numpy as np
import arviz as az
from Numba import jit, generated_jit, njit
x = np.random.randn(10000, 2000)
```

```
def bfmi(energy):
    energy_mat = np.atleast_2d(energy)
    num = np.square(np.diff(energy_mat, axis=1)).mean(axis=1)
    den = np.var(energy_mat, axis=1)
    return num / den
```

```
%timeit -n 10 -r 1 bfmi(x) :: 467 ms ± 0 ns per loop (mean ± std.
dev. of 1 run, 10 loops each).
```

Here is bfmi with Numba jit decorator applied:

```
@jit(cache=True)
def bfmi_jit(energy):
    energy_mat = np.atleast_2d(energy)
    num = np.square(np.diff(energy_mat, axis=1)).mean(axis=1)
    den = np.var(energy_mat, axis=1)
    return num / den
```

```
%timeit -n 10 -r 1 bfmi_jit(x) :: 452 ms ± 0 ns per loop (mean ± std.
dev. of 1 run, 10 loops each).
```

Here is the cProfile result of the same program during another run:

Name	Call Count	Time (ms)		Own Time (ms)	
<method 'read' of 'io.FileIO' objects>	1070	8620	19.0%	8620	19.0%
bfmi	1	9158	20.1%	4038	8.9%
__call__	1033	2981	6.6%	2981	6.6%
<built-in method _imp.create_dynamic>	147	3321	7.3%	2762	6.1%
<built-in method posix.stat>	8923	2309	5.1%	2309	5.1%
<method 'read' of 'io.BufferedReader' objects>	336	2166	4.8%	2166	4.8%
<method 'reduce' of 'numpy.ufunc' objects>	28	2044	4.5%	2044	4.5%
<built-in method marshal.loads>	1070	2032	4.5%	2032	4.5%
get_data	1070	10525	23.2%	1905	4.2%
bfmi_jit	1	2240	4.9%	1895	4.2%

bfmi_jit takes almost a quarter of the time of the non-jitted **bfmi** function in the profiler.

My main aim is to repeat the above steps for different arviz functions and apply Numba project wide.

The second part of my project is to implement xarray ufuncs for arviz/diagnostic functions like geweke and others after consulting the project mentors. The implementation would be similar to the ufunc implementation of rhat which is already included in arviz.diagnostics.

Gsoc 2019 Timeline

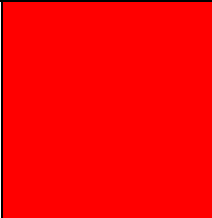

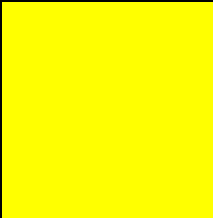
The duration of Gsoc is approximately 12 weeks barring the 3 weeks of community bonding period.

The majority of my time during the project would be spent on applying Numba across the project (**approx 80%**) with the remaining time being devoted to working on xarray ufuncs.

During the community bonding period, I'll be travelling for a couple of days between 14th and 16th of May.

The detailed timeline is given below.

Legend (Importance Level) :  >  > 

Time Frame	Start Date	End Date	Task	
Application Review Period	April 9	May 6	Detailed study of Numba and xarray documentation and other viable sources.	
			Opening and fixing ArviZ issues.	
Community Bonding Period	May 7	May 26	Complete a partial implementation of Numba on specific functions(plot_kde, convert_to_inferenc	

			e data etc) and discuss the issues faced with the mentors.	
Work Period I	May 27	June 24		
	May 27	June 4	Numba implementation on stats	
	June 5	June 12	Numba implementation on diagnostics	
	June 12	June 24	Partial implementation of Numba on ArviZ plots.	
Phase one Evaluation	June 24	June 28		
Work Period II	June 29	July 22		
	June 29	July 5	Complete the Numba implementation on plots.	
	July 5	July 22	Implement Numba on ArviZ data.	
Phase 2 evaluations	July 22	July 26		
Work Period III	July 27	August 19		

	July 27	August 8	Apply Numba on ArviZ/tests.	
	August 9	August 19	Work on xarray ufuncs after discussing with mentors.	
	August 19	August 26	Submit the final prototype and make the changes suggested.	
Final Evaluation	August 26	Sept 3		

Post Gsoc

I would continue to be a part of the ArviZ community and contribute actively in its development.

My Contributions:

Pull requests merged

- <https://github.com/arviz-devs/arviz/pull/565>
- <https://github.com/arviz-devs/arviz/pull/577>
- <https://github.com/arviz-devs/arviz/pull/600>
- <https://github.com/arviz-devs/arviz/pull/605>
- <https://github.com/arviz-devs/arviz/pull/627>
- <https://github.com/arviz-devs/arviz/pull/635>

[WIP] Pull Requests:

- <https://github.com/arviz-devs/arviz/pull/625>

- <https://github.com/arviz-devs/arviz/pull/622>

Will open a few more before the application review process starts.

Why Me?

I have been contributing regularly to ArviZ for the past few months and as a result, I am familiar of it's codebase to a large extent and am very comfortable with using it on a regular basis. As an added advantage, I have already gone through Numba documentation and applied it partially on a few of ArviZ's functions while working on this proposal and have gained some confidence on working with the package. Furthermore with my exams ending by the 5th of May and my next semester starting in August, I have a complete summer to work diligently on the proposed project and meet the desired requirements.

Really looking forward to working with the developers this summer.

Regards,
Aniruddha Banerjea.