Stat 502 Project Proposal

Arman Bilge, Cheng Wang, and Zexuan Zhou

November 2, 2016

Background

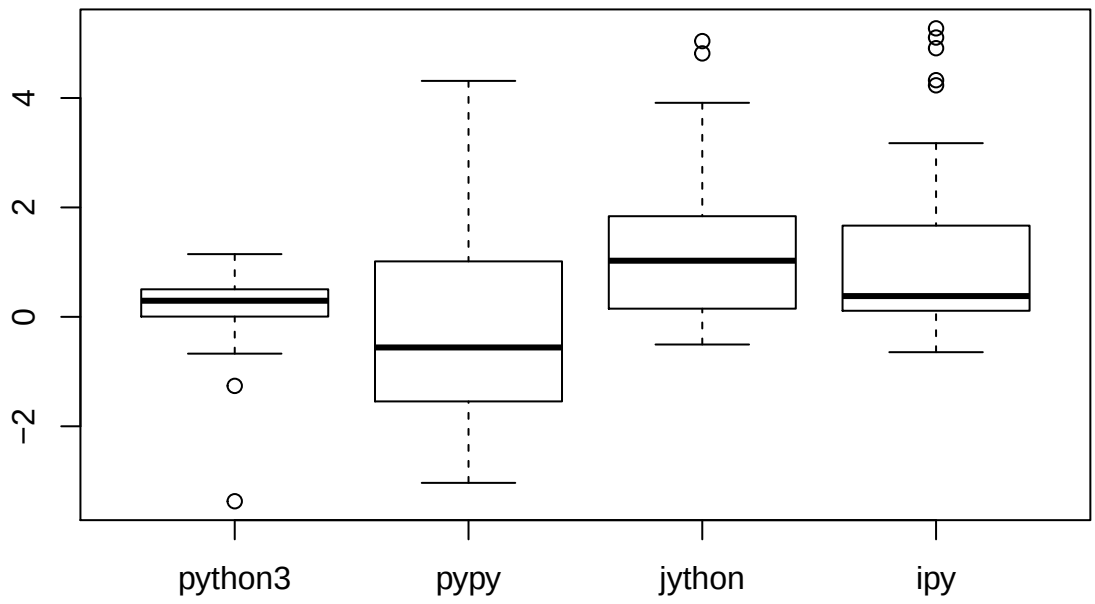
Python is one of the most popular programming languages, with many applications in scientific computing. Thus, maximizing the performance of Python code is of particular interest. Because Python is an interpreted language, its performance depends on the particular runtime used. Interpreted languages often rely on a just-in-time (JIT) compiler to dynamically optimize the code at runtime, which the official CPython[1] interpreter lacks. Fortunately, there are many options for Python runtimes besides CPython, including PyPy,[2] Jython,[3] and IronPython.[4] PyPy uses its own JIT compiler, while Jython and IronPython are built on top of the Java and .NET frameworks, respectively, which encompass their own JIT compilers. In this study we hope to determine whether the use of a runtime with a JIT compiler offers a performance benefit over the official CPython implementations.

Pilot study

Each experimental unit is a small program that is a solution to one of the Project Euler[5] problems. In this pilot study we considered 47 such programs. Each program was run once in each of the CPython (versions 2 and 3), PyPy, Jython, and IronPython interpreters on a personal laptop running macOS 10.9. The program run time was measured using the `time` module in the Python standard library from just before to just after code execution. Note that this purposefully excludes the time that the interpreters took to start up or shut down.

```
boxplot(df[,2:5])
```

[1] https://www.python.org/
[2] http://pypy.org/
[3] http://www.jython.org/
[4] http://ironpython.net/
[5] https://projecteuler.net/



We would like to do an one-way ANOVA test with the following hypothesis:

$$H_0 : \mu_1 = \mu_2 = \mu_3 = \mu_4 \text{ vs } \quad H_a : \text{At least one differs from the others.}$$

Where $\mu_1, \mu_2, \mu_3, \mu_4$ correpsonds the true population average compiling time ratio of Python3 to Python, PyPy to Python, Jython to Python, and IronPython to Python, respectively.

```
anova(lm(values ~ ind, data=stack(df)))
```

```
## Analysis of Variance Table
##
## Response: values
##            Df Sum Sq Mean Sq F value  Pr(>F)
## ind         4     88   22.10    14.4 1.8e-10 ***
## Residuals 230    354    1.54
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The ANOVA test gives a p-value of 1.8e-10 << 0.05, with degrees of freedom (4, 230). Therefore we reject the null and conclude that one of the compilers have different compiling time ratio.

Proposal for larger experiment

We will implement some additional strategies to ensure robustness of our results, starting with performing multiple replicates for each experimental unit. As it is challenging to completely control for the state of

the computer that we are running the tests on, we expect there to be some noise in our measurements. Performing multiple replicate runs for each program should help reduce the effects of this noise on our results. Furthermore, it is not straightforward to accurately measure the performance of a runtime using a JIT compiler. A method may need to be called hundreds or thousands of times before the JIT compiler determines that it is worth optimizing. Because we want to assess the asymptotic performance of the interpreters, we will do several "warm-up" runs of the program we are testing before taking any measurements to provide ample opportunity for the JIT compiler to make its optimizations.

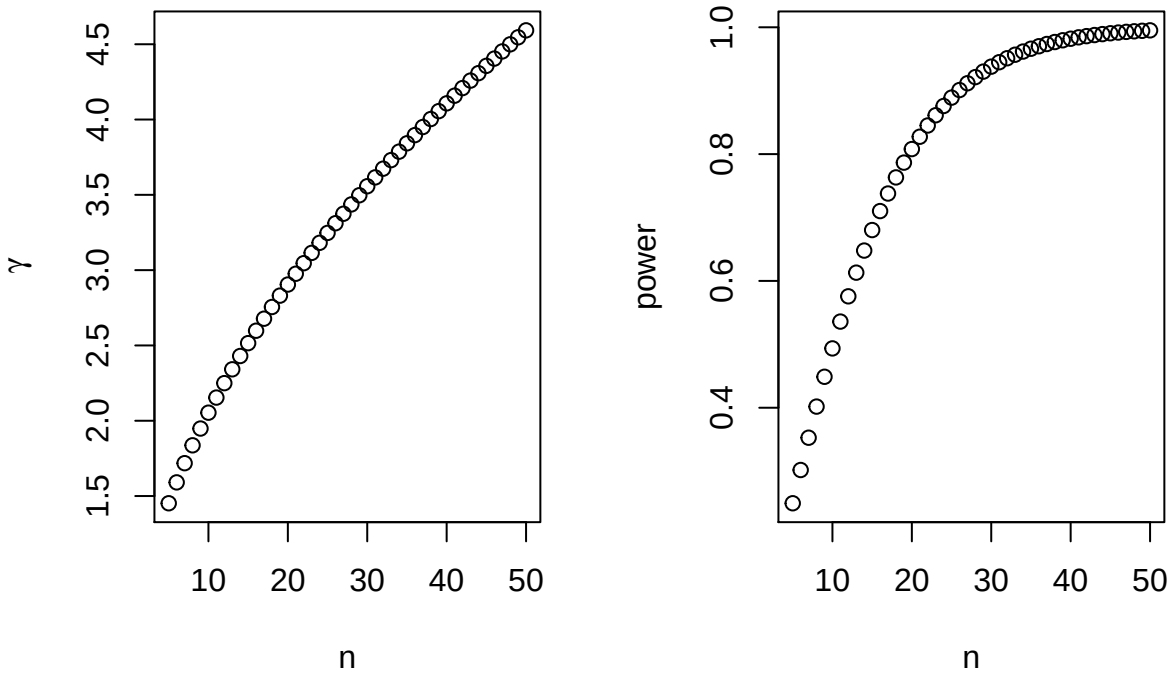
Power calculation

We want ot calculate an proper sample size n to ensure at least a power of 80% for the following hypothesis:

$$\mu_3 - \mu_1 = 1$$

We use the pooled variance to estimate the true population variance.

```
attach(mtcars)
par(mfrow=c(1,2))
plot(x=n,y=t.gamma,ylab=expression(gamma))
plot(x=n,y=t.power,ylab="power")
```



From the graph we see that we need at a sample size of at least 22, which we will use in the future study of our project.