jmh_results

October 6, 2021

[1]:
```
import re
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
plt.style.use('seaborn-darkgrid')

import seaborn as sns
sns.set_theme(style="darkgrid")
```

[2]:
```
def parse_line(line):
    line = re.sub(r'±|\n', '', line)
    return re.split(r'\s+', line)
```

[3]:
```
def read_results():
    with open('jmh-results.txt','r') as f:
        lines = (parse_line(line) for line in f)
        header = next(lines)
        df = pd.DataFrame(lines, columns=header, )
    df[['Benchmark', 'Method']] = df.Benchmark.str.split('.', n=1, expand=True)
    df.Score = df.Score.astype(float)
    df.Error = df.Error.astype(float)
    df['Error%'] = df.Error * 100 / df.Score
    return df

results = read_results()
results
```

[3]:
```
             Benchmark (exponent)    (n)   Mode Cnt          Score  \
0   FibonacciBenchmark        N/A     10  thrpt  15   2.031906e+06
1   FibonacciBenchmark        N/A    100  thrpt  15   8.291059e+05
2   FibonacciBenchmark        N/A   1000  thrpt  15   3.143388e+05
3   FibonacciBenchmark        N/A  10000  thrpt  15   2.797877e+04
4   FibonacciBenchmark        N/A     10  thrpt  15   7.767480e+06
5   FibonacciBenchmark        N/A    100  thrpt  15   6.309666e+05
6   FibonacciBenchmark        N/A   1000  thrpt  15   2.618531e+04
7   FibonacciBenchmark        N/A  10000  thrpt  15   4.857550e+02
```

```
8   FibonacciBenchmark        N/A     10  thrpt  15  2.291288e+06
9   FibonacciBenchmark        N/A    100  thrpt  15  1.826508e+05
10  FibonacciBenchmark        N/A   1000  thrpt  15  1.341035e+04
11  FibonacciBenchmark        N/A  10000  thrpt  15  3.201160e+02
12      PowerBenchmark          3    N/A  thrpt  15  2.225904e+07
13      PowerBenchmark         10    N/A  thrpt  15  1.484000e+07
14      PowerBenchmark        100    N/A  thrpt  15  5.716120e+06
15      PowerBenchmark       1000    N/A  thrpt  15  1.533683e+06
16      PowerBenchmark      10000    N/A  thrpt  15  8.461656e+04
17      PowerBenchmark          3    N/A  thrpt  15  2.009381e+07
18      PowerBenchmark         10    N/A  thrpt  15  1.378799e+07
19      PowerBenchmark        100    N/A  thrpt  15  5.210801e+06
20      PowerBenchmark       1000    N/A  thrpt  15  1.429798e+06
21      PowerBenchmark      10000    N/A  thrpt  15  8.423767e+04
22      PowerBenchmark          3    N/A  thrpt  15  2.346446e+07
23      PowerBenchmark         10    N/A  thrpt  15  1.380514e+07
24      PowerBenchmark        100    N/A  thrpt  15  4.913601e+06
25      PowerBenchmark       1000    N/A  thrpt  15  1.406250e+06
26      PowerBenchmark      10000    N/A  thrpt  15  8.410896e+04
27      PowerBenchmark          3    N/A  thrpt  15  2.095599e+07
28      PowerBenchmark         10    N/A  thrpt  15  6.319776e+06
29      PowerBenchmark        100    N/A  thrpt  15  5.165915e+05
30      PowerBenchmark       1000    N/A  thrpt  15  2.220312e+04
31      PowerBenchmark      10000    N/A  thrpt  15  3.197790e+02
32      PowerBenchmark          3    N/A  thrpt  15  2.049335e+07
33      PowerBenchmark         10    N/A  thrpt  15  6.812648e+06
34      PowerBenchmark        100    N/A  thrpt  15  4.836735e+05
35      PowerBenchmark       1000    N/A  thrpt  15  2.208013e+04
36      PowerBenchmark      10000    N/A  thrpt  15  3.170330e+02

         Error  Units      Method     Error%
0    61963.903  ops/s        fast   3.049546
1    42809.302  ops/s        fast   5.163309
2    11219.042  ops/s        fast   3.569092
3     1058.518  ops/s        fast   3.783290
4   327983.389  ops/s      simple   4.222520
5    16850.376  ops/s      simple   2.670565
6     4216.533  ops/s      simple  16.102669
7       14.559  ops/s      simple   2.997190
8    86593.758  ops/s     simple2   3.779261
9     6257.970  ops/s     simple2   3.426194
10     637.116  ops/s     simple2   4.750929
11       5.641  ops/s     simple2   1.762174
12  684701.481  ops/s        fast   3.076060
13  956855.240  ops/s        fast   6.447812
14  155912.919  ops/s        fast   2.727601
15   27815.144  ops/s        fast   1.813617
```

```
16     1497.069  ops/s            fast   1.769239
17   698242.534  ops/s           fast2   3.474914
18   375131.647  ops/s           fast2   2.720713
19   267590.529  ops/s           fast2   5.135305
20    36605.211  ops/s           fast2   2.560166
21     1355.753  ops/s           fast2   1.609438
22   626019.949  ops/s   fastRecursive   2.667949
23  1964664.864  ops/s   fastRecursive  14.231398
24   225389.753  ops/s   fastRecursive   4.587058
25    51340.233  ops/s   fastRecursive   3.650860
26     1735.093  ops/s   fastRecursive   2.062911
27   736467.874  ops/s          simple   3.514354
28   113522.293  ops/s          simple   1.796302
29    12890.154  ops/s          simple   2.495231
30      609.584  ops/s          simple   2.745488
31       14.020  ops/s          simple   4.384278
32   729584.003  ops/s         simple2   3.560101
33   415636.263  ops/s         simple2   6.100951
34     8630.732  ops/s         simple2   1.784413
35      457.857  ops/s         simple2   2.073615
36        7.673  ops/s         simple2   2.420253
```

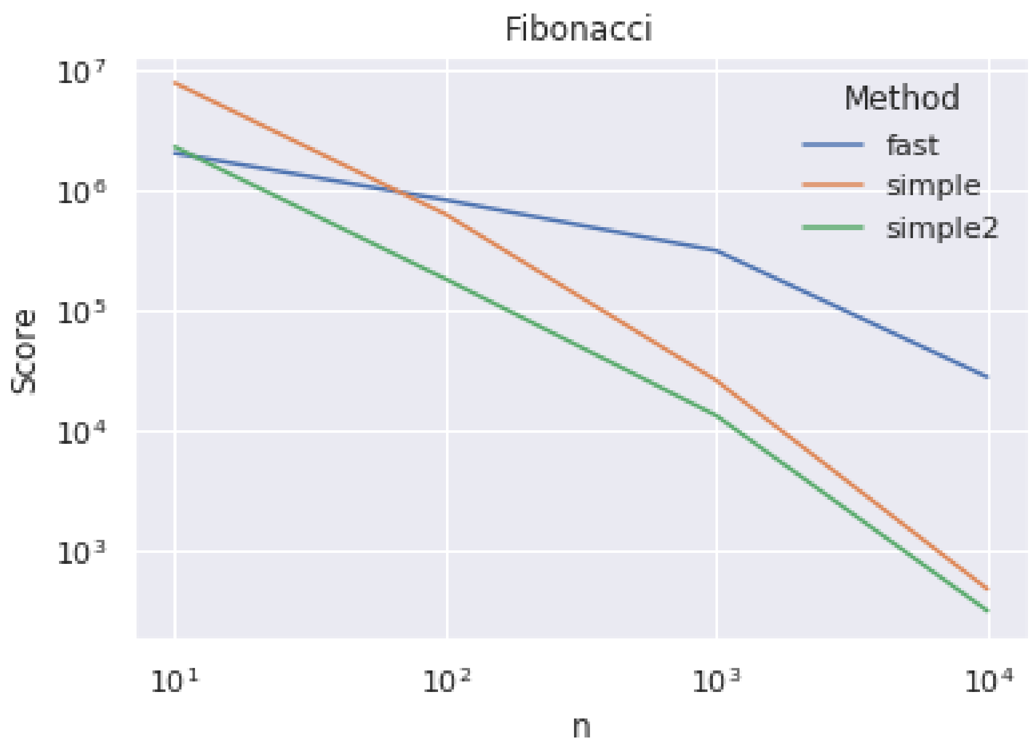
[4]:
```python
def get_fibonacci_results(data):
    df = data.copy() # fuck SettingWithCopyWarning
    df = df[df.Benchmark == 'FibonacciBenchmark']
    df['n'] = df['(n)'].astype(int)
    return df[['Method', 'n', 'Score', 'Error', 'Error%']]

fib = get_fibonacci_results(results)
fib
```

[4]:
```
     Method      n        Score       Error     Error%
0      fast     10  2031905.665   61963.903   3.049546
1      fast    100   829105.926   42809.302   5.163309
2      fast   1000   314338.832   11219.042   3.569092
3      fast  10000    27978.768    1058.518   3.783290
4    simple     10  7767479.899  327983.389   4.222520
5    simple    100   630966.624   16850.376   2.670565
6    simple   1000    26185.305    4216.533  16.102669
7    simple  10000      485.755      14.559   2.997190
8   simple2     10  2291288.296   86593.758   3.779261
9   simple2    100   182650.801    6257.970   3.426194
10  simple2   1000    13410.347     637.116   4.750929
11  simple2  10000      320.116       5.641   1.762174
```

[5]:
```python
# Seaborn is easy and nice, but I don't know how to draw errors with it.
# So there are a lot of manual mechanics below.
```

```
g = sns.lineplot(x="n", y="Score", hue="Method", data=fib)
g.axes.set_xscale('log')
g.axes.set_yscale('log')
g.set_title('Fibonacci')
```

```
[5]: Text(0.5, 1.0, 'Fibonacci')
```



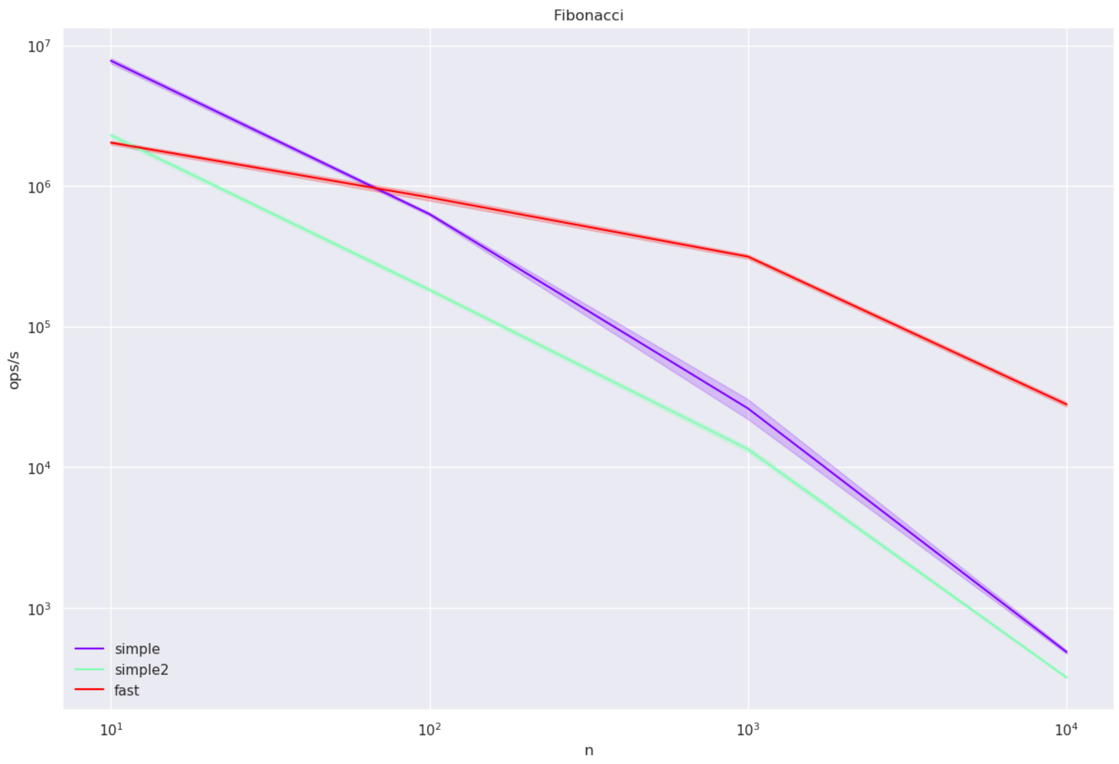
```
[6]: def plot_score(data, methods, xlabel):
         n = data.index
         scores = data.Score
         errs = data.Error
         colors = [plt.cm.rainbow(x) for x in np.linspace(0, 1, len(methods))]
         plt.figure(figsize=(15, 10), dpi=100)
         for i, method in enumerate(methods):
             score = scores[method]
             err = errs[method]
             color = colors[i]
             plt.plot(n, score, '-', color=color, label=method)
             plt.fill_between(n, score - err, score + err, color=color, alpha=0.2)
         plt.xscale('log')
         plt.yscale('log')
         plt.xlabel(xlabel)
         plt.ylabel('ops/s')
```

```
    plt.legend(loc='lower left')
```

[7]:
```
fib.Method.unique()
```

[7]:
```
array(['fast', 'simple', 'simple2'], dtype=object)
```

[8]:
```
def plot_fib_score(data):
    data = data.pivot(index='n', columns='Method', values=['Score', 'Error'])
    plot_score(data, ['simple', 'simple2', 'fast'], 'n')
    plt.title('Fibonacci')

plot_fib_score(fib)
```



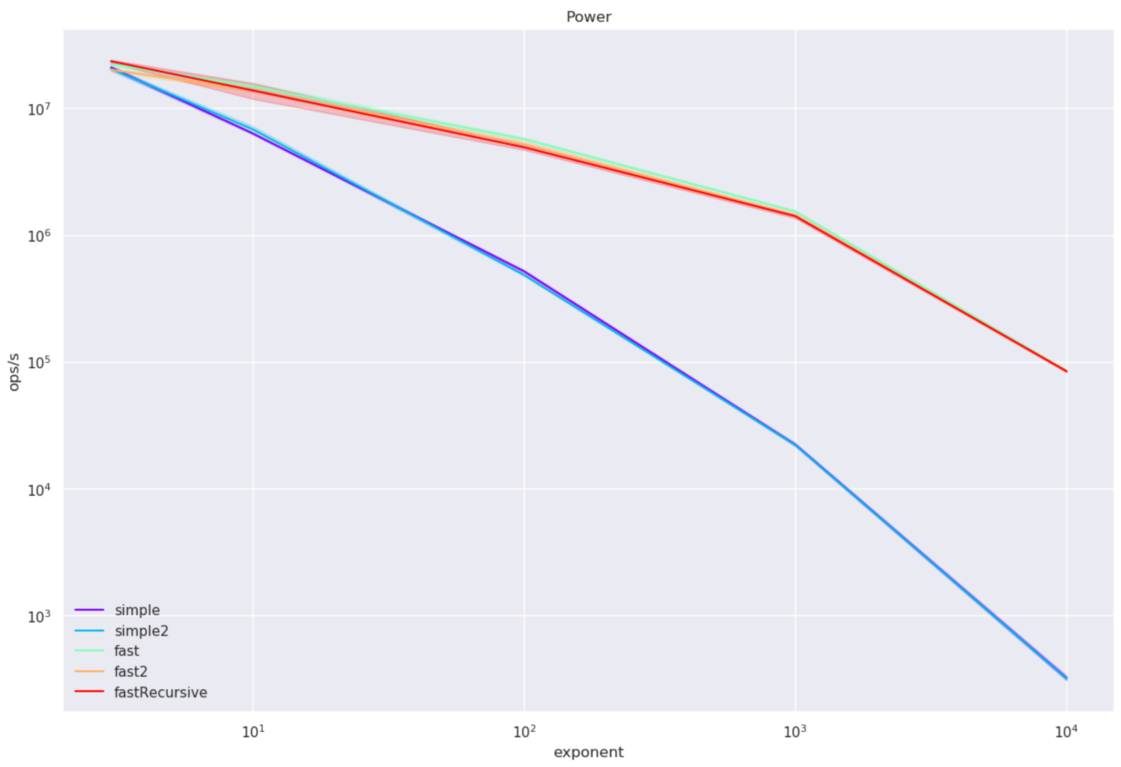
[9]:
```
def get_power_results(data):
    df = data.copy() # fuck SettingWithCopyWarning
    df = df[df.Benchmark == 'PowerBenchmark']
    df['exponent'] = df['(exponent)'].astype(int)
    df = df[['Method', 'exponent', 'Score', 'Error', 'Error%']]
    return df

power = get_power_results(results)
power
```

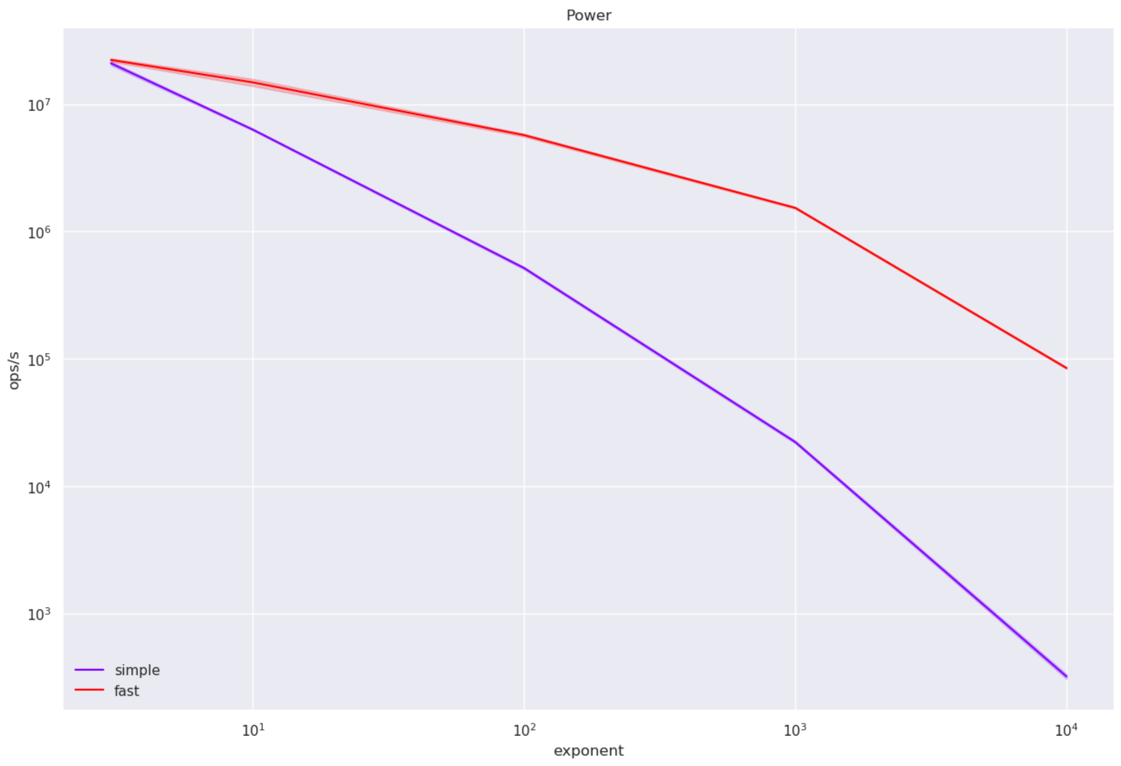
[9]:
```
             Method  exponent         Score        Error     Error%
12             fast         3  2.225904e+07   684701.481   3.076060
13             fast        10  1.484000e+07   956855.240   6.447812
14             fast       100  5.716120e+06   155912.919   2.727601
15             fast      1000  1.533683e+06    27815.144   1.813617
16             fast     10000  8.461656e+04     1497.069   1.769239
17            fast2         3  2.009381e+07   698242.534   3.474914
18            fast2        10  1.378799e+07   375131.647   2.720713
19            fast2       100  5.210801e+06   267590.529   5.135305
20            fast2      1000  1.429798e+06    36605.211   2.560166
21            fast2     10000  8.423767e+04     1355.753   1.609438
22    fastRecursive         3  2.346446e+07   626019.949   2.667949
23    fastRecursive        10  1.380514e+07  1964664.864  14.231398
24    fastRecursive       100  4.913601e+06   225389.753   4.587058
25    fastRecursive      1000  1.406250e+06    51340.233   3.650860
26    fastRecursive     10000  8.410896e+04     1735.093   2.062911
27           simple         3  2.095599e+07   736467.874   3.514354
28           simple        10  6.319776e+06   113522.293   1.796302
29           simple       100  5.165915e+05    12890.154   2.495231
30           simple      1000  2.220312e+04      609.584   2.745488
31           simple     10000  3.197790e+02       14.020   4.384278
32          simple2         3  2.049335e+07   729584.003   3.560101
33          simple2        10  6.812648e+06   415636.263   6.100951
34          simple2       100  4.836735e+05     8630.732   1.784413
35          simple2      1000  2.208013e+04      457.857   2.073615
36          simple2     10000  3.170330e+02        7.673   2.420253
```

[10]:
```python
power.Method.unique()
```

[10]:
```
array(['fast', 'fast2', 'fastRecursive', 'simple', 'simple2'],
      dtype=object)
```

[11]:
```python
def plot_power_score(data, methods):
    data = data.pivot(index='exponent', columns='Method', values=['Score',␣
↪'Error'])
    plot_score(data, methods, 'exponent')
    plt.title('Power')

plot_power_score(power, ['simple', 'simple2', 'fast', 'fast2', 'fastRecursive'])
plot_power_score(power, ['simple', 'fast'])
```


Power
ops/s
exponent
simple
simple2
fast
fast2
fastRecursive


Power
ops/s
exponent
simple
fast