Runtimes of different *linprog* methods

Mathematics 441 – portfolio object 3

Mauricio Soroco
60785250

October 2022

Contents

1 Introduction

In this portfolio I am exploring the difference in runtimes between the different linprog methods to solve linear programming problems. Linprog provides four methods for solving LP problems:

- HIGHS-ds
- HIGHS-ipm
- Interior point method (default)
- Simplex
- Revised Simplex

The two HIGHS methods listed are the dual simplex and the interior point methods respectively. When testing these, I specified "HIGHS" and let linprog choose which version of HIGHS was best for my problem (this is problem dependent). The HIGHS solver is the the fastest LP solvers in SciPy, especially for large sparse problems. At a basic level, HIGHS takes advantage of threading to be able to perform calculations in parallel in addition to its exploitation of sparsity for solving LP problems.

2 Methods

I constructed the linear programming problem:

$$\min_{Ax=-b,\quad x\geq 0} \vec{c}\cdot\vec{x} \tag{1}$$

Where A "is" $m \times n$ and b is $m \times 1$ and c is $1 \times n$. Each of A, b, and c were constructed by sampling a normal distribution. As n and m varied in $[1, \quad 50]$, linprog would solve 100 problems with the given n, m dimensions, and the average time in seconds was stored. Below we have plots showing the time linprog took to solve a random linear programming problem over different n and m.

See the Python code attached at the end of this document.

The code was run on the Compute Science undergraduate server, since its execution took hours to finish. More specifically to generate the graph for HIGHS from start to finish, it took approximately 7 seconds. To generate the interior point graph from start to finish, it took a couple minutes. And to generate the remaining two graphs, it took approximately 2 hours in total (for which I just let the code run in the background).

3 Results

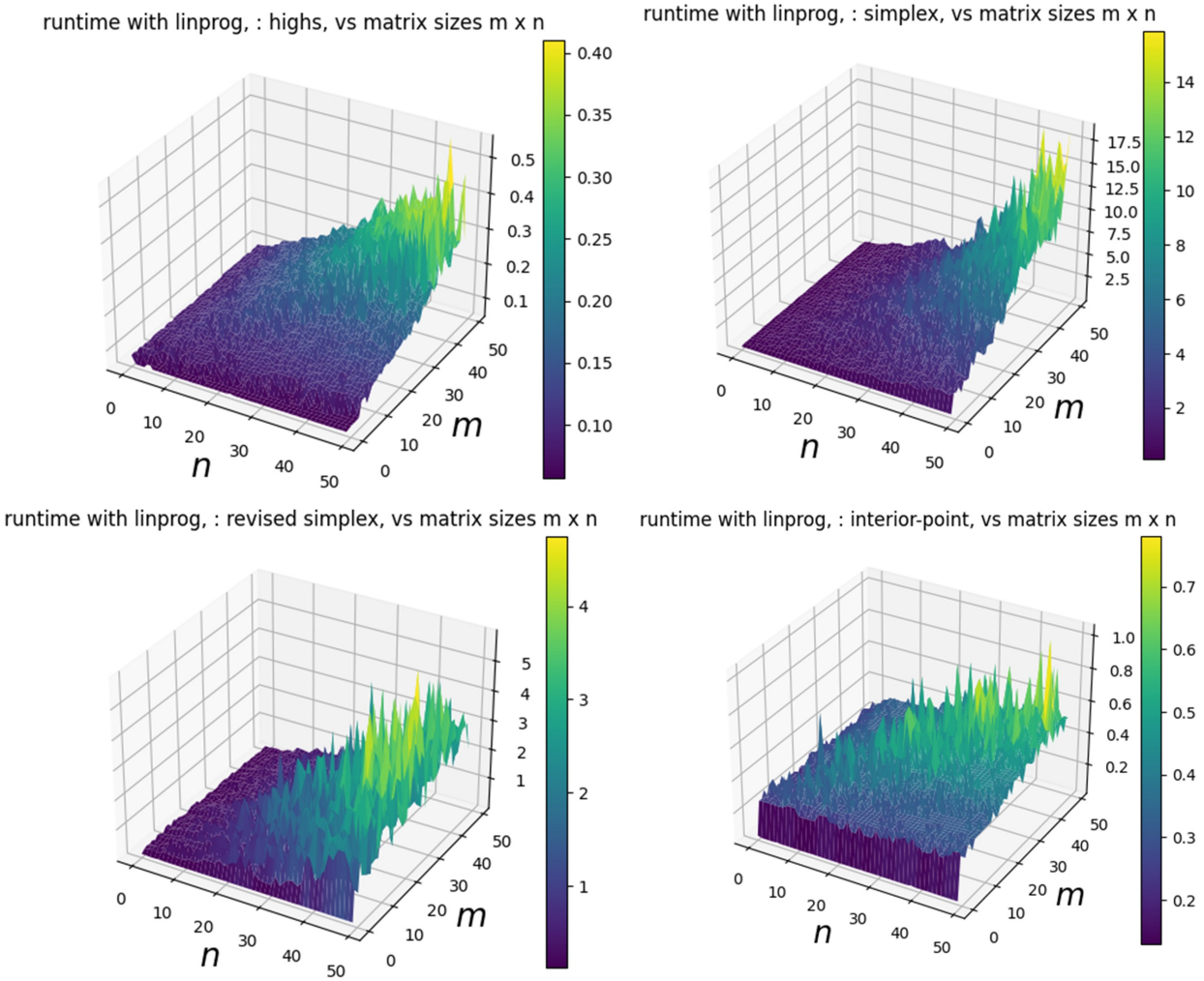

Figure 1: Runtimes of different linprog methods as dimensions of the $m \times n$ linear programming system vary. All times are in seconds.

4 Discussion

We see that the HIGHS method is the fastest of the four. The default linprog method, the Interior point method, (at the time of this writing, Interior Point is the default linprog method but will be changed to HIGHS in a future update) did perform slower but was more comparable to HIGHS than the other two methods. The linprog documentation even recommends switching to HIGHS for this reason, though we can see why the interior point method is the default before HIGHS was invented. While the interior point method starts fast, and seems to jump up to 0.4 seconds for the lower dimensions, we see that its growth onwards (from 0.4 to 0.8) is about the same growth amount we see in HIGHS (from 0.1 at its fastest to 0.4 at its slowest). We see that the revised simplex method took about $m/10$ seconds to run. It is five times slower than the interior point method and 10 times slower than HIGHS at the largest sizes of n and m. What is interesting to note is that the fastest revised simplex trials (at $n, m = 10$) is about the same speed as the interior point method at its slowest ($n, m = 50$). The worst performance was by far the simplex method. At $n = 10, m = 10$ it was already performing slower than the interior point method at $n = 50, m = 50$. We can clearly see polynomial or exponential growth for the simplex method.

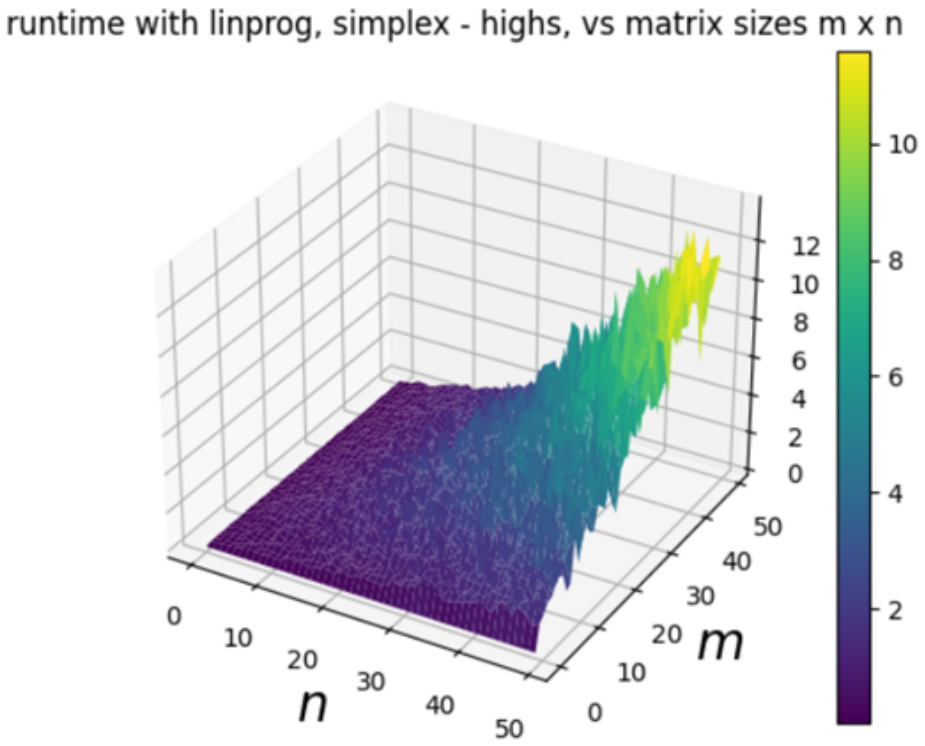

Figure 2: Runtime difference of simplex method and HIGHS as dimensions of the $m \times n$ linear programming system vary. All times are in seconds.

The figure above shows the difference between the simplex surface plot and the HIGHS surface plot. We see that for simpler problems (as above) that the runtimes for both are

similar (hence the 0 seconds of time). However the growth in the time as the LP problem becomes more complex shows that there is a polynomial or exponential difference between the two runtimes.

5 Conclusion

Overall, it appears that the interior point method and HIGHS have similar growth rates (though HIGHS is consistently faster). The simplex method seems to have somewhat of a polynomial or exponential growth rate. On a similar note, Klee and Minty showed that the simplex method could require as much as $O(2^n)$ iterations to solve (Vanderbei p 45). Just like in Vanderbei (p53) problems where $n << m$ or $m << n$ ran faster as they seem to be in some sense "easier". Furthermore, Vanderbei also illustrates that the simplex method follows an exponential growth rate agreeing with our plots.

```python

import sys
import numpy as np
import scipy.linalg as la
import pandas as pd
import matplotlib.pyplot as plt
from scipy.optimize import linprog
from mpl_toolkits import mplot3d
from timeit import timeit


import string
from numpy import float64

def time(rows, cols, method='interior-point'):
    # create multi-dim array by providing shape
    seconds = np.empty(shape=(rows, cols), dtype=float64)
    X = np.empty(shape=(rows, cols), dtype=float64)
    Y = np.empty(shape=(rows, cols), dtype=float64)

    for n in range(rows):
        for m in range(cols):
            A = np.random.randn(m+1, n+1)
            b = np.random.randn(m+1)
            c = np.random.randn(n+1)

            seconds[m, n]= timeit(lambda: linprog(-c,A,b, method=method), number=100)
            X[m, n] = m
            Y[m, n] = n

    return X, Y, seconds


def plot(X, Y, seconds, title, method: float):
    ax = plt.axes(projection='3d')

    p = ax.plot_surface(X, Y, seconds, rstride=1, cstride=1, cmap='viridis',
edgecolor='none')
    # p = ax.scatter3D(X, Y, seconds, cmap='viridis', edgecolor='none')
    # p = ax.contour3D(X, Y, seconds, rstride=1, cstride=1, cmap='viridis', edgecolor='none')
    ax.set_title('runtime with linprog, ' + method + ', vs matrix sizes m x n')
    ax.set_xlabel('$n$', fontsize=20)
    ax.set_ylabel('$m$', fontsize=20)
    ax.set_zlabel('$Time (seconds)$', fontsize=10, rotation=60)
    # ax.set_zlim3d(0, 1000) # to set scaling TODO: use this to make plots more comparable
    # ax.view_init(60, 35) # change viewing
    plt.colorbar(p, ax=ax)
    plt.savefig(title)
    plt.close()


def time_and_plot(rows, cols, title, method:float ='interior-point'):
    (X, Y, seconds) = time(rows, cols, method)
    plot(X, Y, seconds, title, method)
    return seconds


```

```
57  def time_difference(rows, cols, method1, method2):
58      # create multi-dim array by providing shape
59      seconds = np.empty(shape=(rows, cols), dtype=float64)
60      X = np.empty(shape=(rows, cols), dtype=float64)
61      Y = np.empty(shape=(rows, cols), dtype=float64)
62
63      for n in range(rows):
64          for m in range(cols):
65              A = np.random.randn(m+1, n+1)
66              b = np.random.randn(m+1)
67              c = np.random.randn(n+1)
68
69              seconds[m, n]= timeit(lambda: linprog(-c,A,b, method=method1), number=100) -
    timeit(lambda: linprog(-c,A,b, method=method2), number=100)
70              X[m, n] = m
71              Y[m, n] = n
72
73      return X, Y, seconds
74
75
76
77  def plot_difference(X, Y, seconds, title, method1, method2):
78      ax = plt.axes(projection='3d')
79
80      p = ax.plot_surface(X, Y, seconds, rstride=1, cstride=1, cmap='viridis',
    edgecolor='none')
81      # p = ax.scatter3D(X, Y, seconds, cmap='viridis', edgecolor='none')
82      # p = ax.contour3D(X, Y, seconds, rstride=1, cstride=1, cmap='viridis', edgecolor='none')
83      ax.set_title('runtime with linprog, ' + method1 + ' - ' + method2 + ', vs matrix sizes m
    x n')
84      ax.set_xlabel('$n$', fontsize=20)
85      ax.set_ylabel('$m$', fontsize=20)
86      ax.set_zlabel('$Time (seconds)$', fontsize=10, rotation=60)
87      # ax.set_zlim3d(0, 1000) # to set scaling TODO: use this to make plots more comparable
88      # ax.view_init(60, 35) # change viewing
89      plt.colorbar(p, ax=ax)
90      plt.savefig(title)
91      plt.close()
92
93  def time_and_plot_difference(rows, cols, title, method1, method2):
94      (X, Y, seconds) = time_difference(rows, cols, method1, method2)
95      plot_difference(X, Y, seconds, title, method1, method2)
96      return seconds
97
98
99
100 def main():
101
102     size = 10
103
104     args = sys.argv[1:]
105     file_name = args[0]
106     if len(args) > 1:
107         size = int(args[1])
108
109
110     s = time_and_plot_difference(size, size, file_name + '_simplex_highs', 'simplex',
    'highs')
```

```
111
112     r = time_and_plot(size, size, file_name + '_revised simplex', 'revised simplex')
113     r = time_and_plot(size, size, file_name + '_simplex', 'simplex')
114     r = time_and_plot(size, size, file_name + '_interior-point', 'interior-point')
115     r = time_and_plot(size, size, file_name + '_highs', 'highs')
116
117
118
119 if __name__ == "__main__":
120     main()
121
```