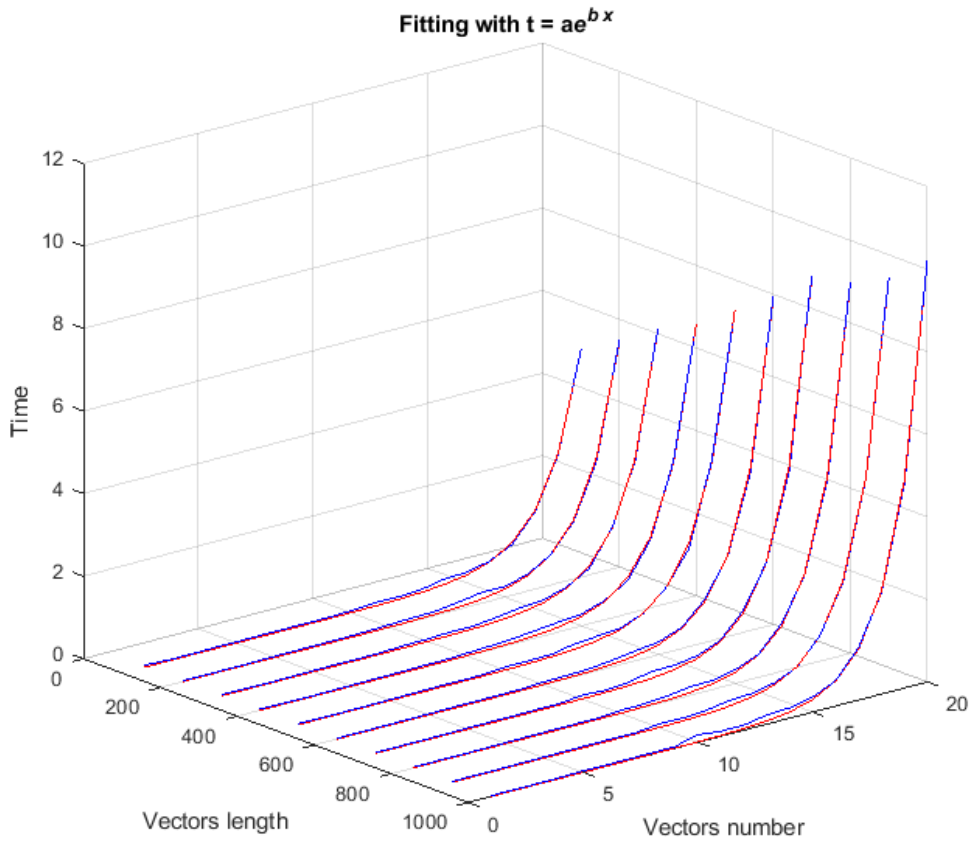
Это моя вторая попытка решить поставленную передо мной задачу, так что опущу условие и сразу перейду к делу. В своем первом решении я умножал две матрицы. Это однозначно приводило верному результату, но не экономило место и время от слова совсем.

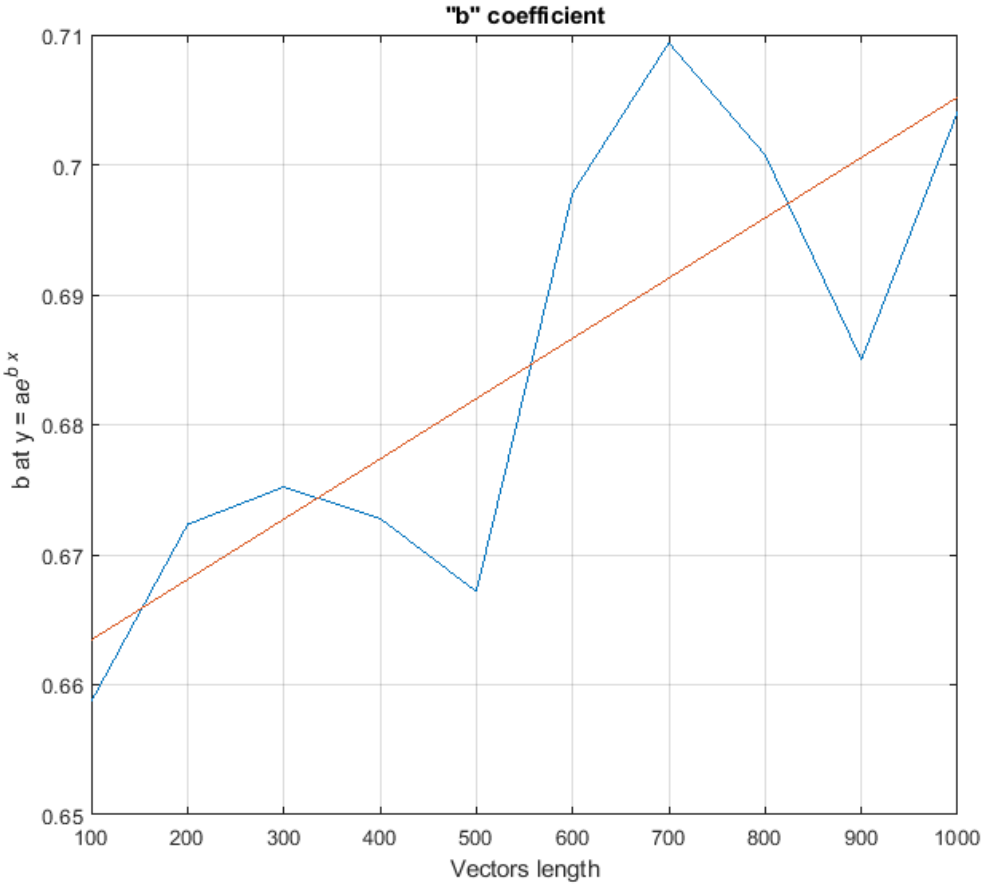
$$\begin{pmatrix} 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & & 0 & 1 \\ \vdots & & \ddots & \vdots & \\ 1 & 1 & \cdots & 1 & 0 \\ 1 & 1 & & 1 & 1 \end{pmatrix}_{2^k \times k} \cdot \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_{k-1} \\ a_k \end{pmatrix}_{k \times z} = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_{2^k-1} \\ v_{2^k} \end{pmatrix}_{2^k \times z}$$

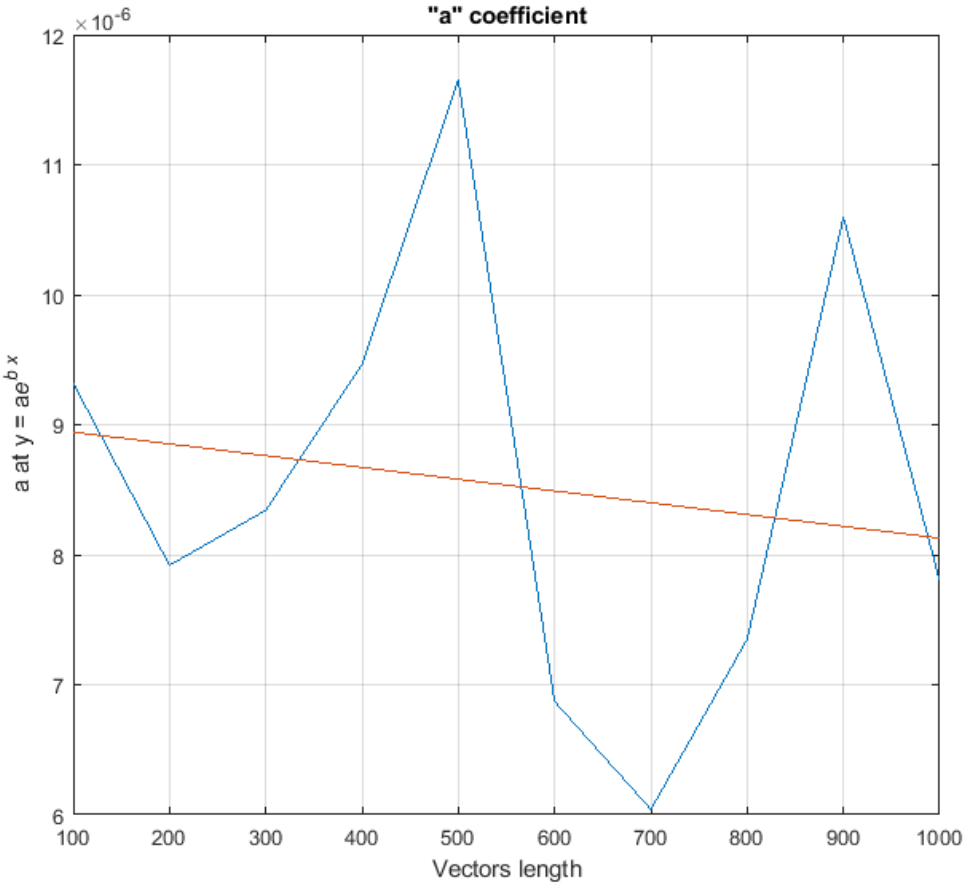
Что, если элемент какой-либо строки матрицы коэффициентов β равен нулю? Это значит, что нет необходимости умножать вектор исходного множества на ноль. Так же если все элементы вектора исходного множества равны нулю, то он не привнесет в линейную комбинацию векторов никакого вклада, это я учел в своем новом алгоритме.

Более того нет необходимости хранить результат перемножений двух (достаточно больших) матриц, ибо мне нужна сумма по модулю двух элементов строк этой матрицы. К среде MatLab не получилось реализовать параллельный подсчет сумм, как бы я ни пытался, поэтому приходится хранить значения сумм строк матрицы в виде вектора-столбца размерностью 2^k. Наверное, это главное упущение новой системы подсчета.

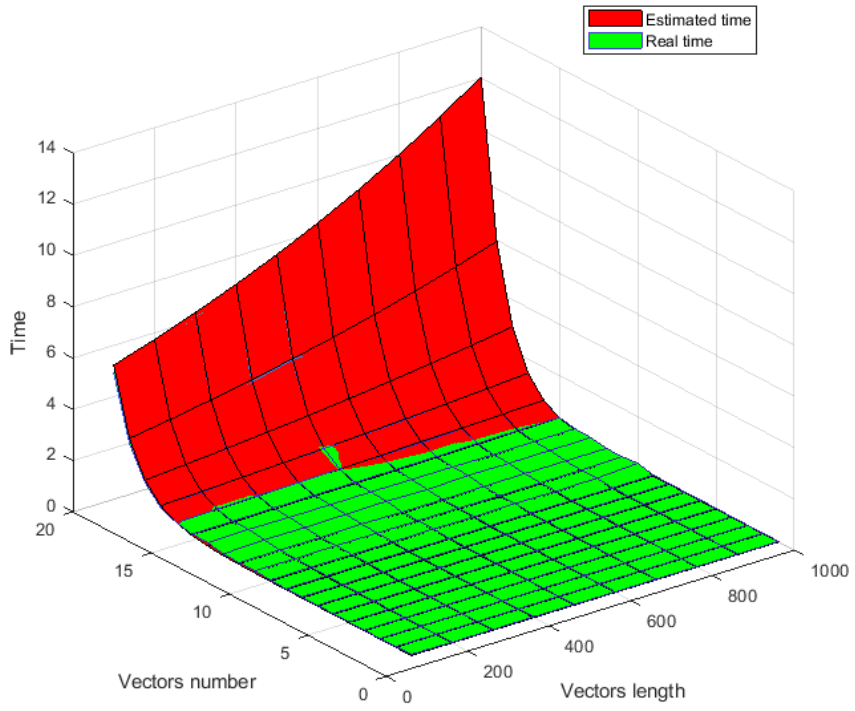
Что касается затрат на подсчет времени выполнения задачи. Отчетливо видна закономерность. Во-первых время зависит от количества исходных векторов. Допустим у нас было n векторов, тогда время можно оценить по следующей формул $t = ae^{bn}$, где a и b некие коэффициенты. В идеале использовать формулу $t = ae^{bn} + d$, но d настолько мал, что я решил избавить себя от него. Далее можно заметить, что коэффициент a можно принять за некую константу, никак не зависящую от количества или длины исходных векторов (в приближении).

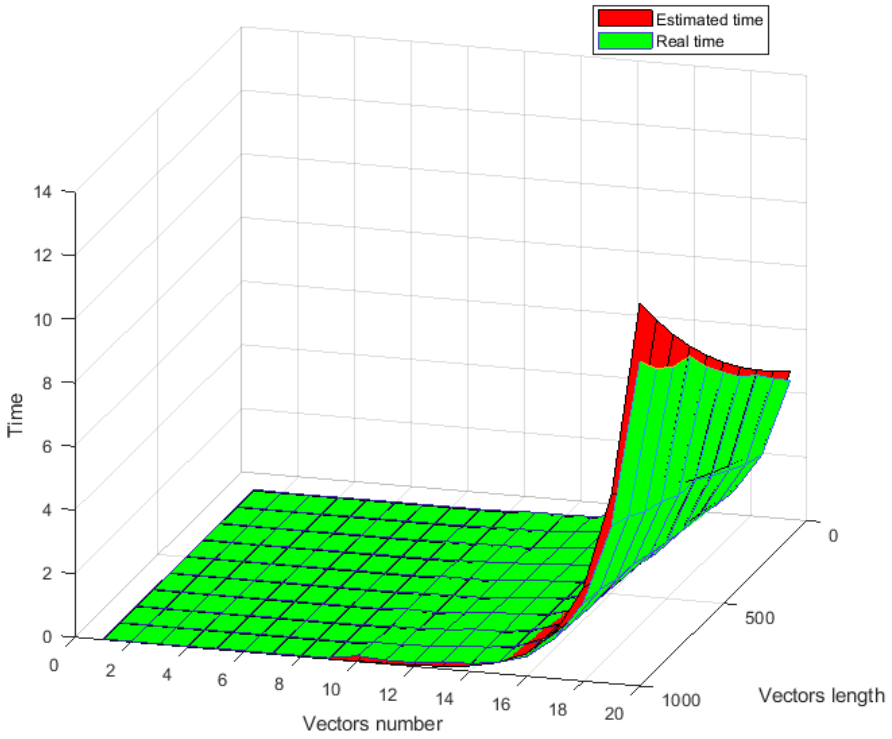



"b" coefficient
b at y = ae^{b x}
0.71
0.7
0.69
0.68
0.67
0.66
0.65
100
200
300
400
500
600
700
800
900
1000
Vectors length


"a" coefficient
×10^{-6}
a at y = ae^{b x}
12
11
10
9
8
7
6
100
200
300
400
500
600
700
800
900
1000
Vectors length

Исходя из этих данных видно, что коэффициент b явно зависит от длины векторов. Приняв a за некую константу можно написать формулу, которая будет оценивать время выполнения алгоритма, а именно $t = ae^{(b+cx)\cdot y}$, где x – количество векторов, а y – их количество. Итак, вот результат на рандомных значениях.

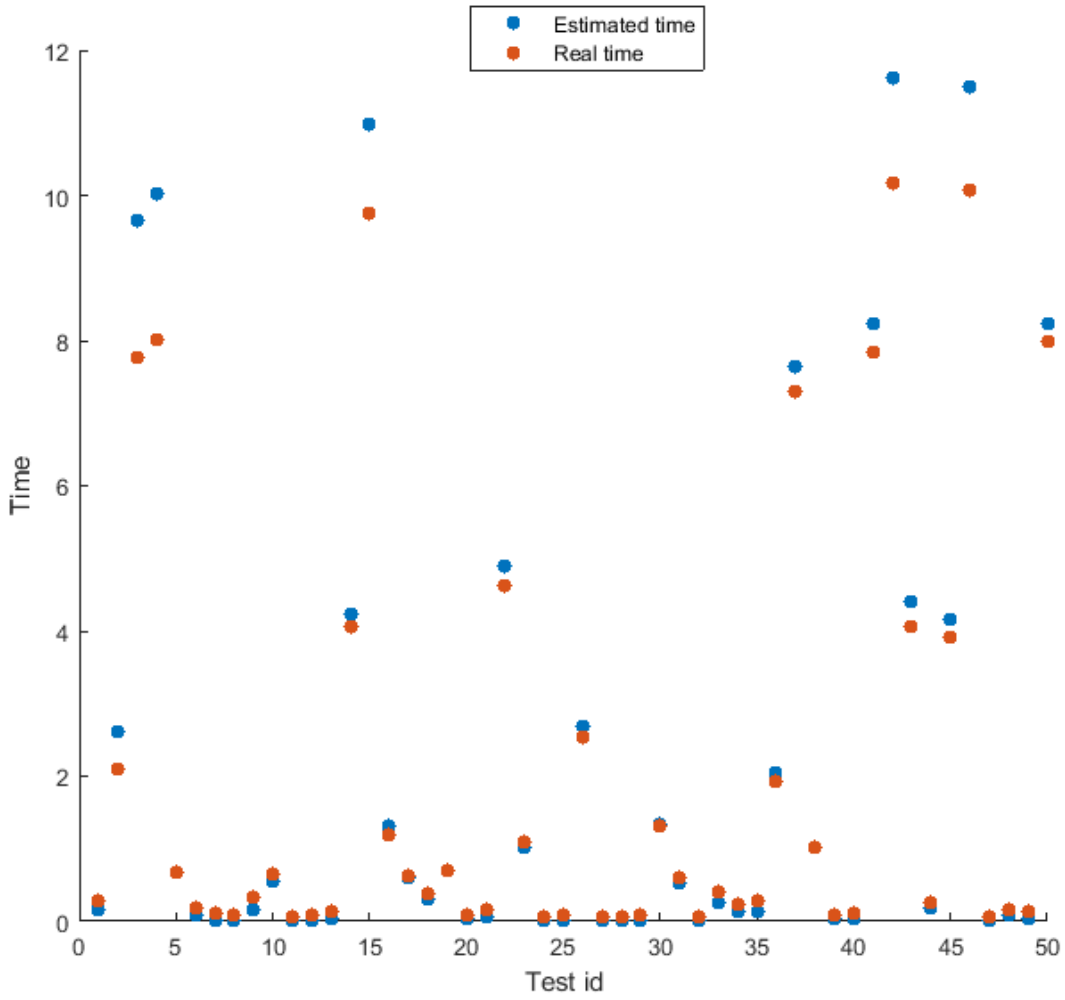




Как видно, моя функция довольно хорошо аппроксимирует время выполнения задачи. Естественно, если бы я тестировал на большем количестве данных, совпадение было бы лучше и предсказать время было бы проще. Стоит отметить, что данные коэффициенты работают лишь на

моей машине. Протестировать на другой и написать программу, которая бы позволяла оценить время на другом аппарате не было, увы (экзамены). Однако это можно организовать.

Я запустил алгоритм по поиску весов рандомного количества векторов с рандомной длиной и получил следующее.



Коэффициент корреляции примерно 0.98, что я лично для себя считаю маленькой победой.

К слову, функция времени в моем случае имеет следующий вид: $t = ae^{(b+cx)\cdot y}$, где y – количество строк, x – количество столбцов, $a \approx 9 \cdot 10^{-6}, b \approx 4.6 \cdot 10^{-5}, c \approx 0.66$.

Вопрос «зачем оценивать время?». Вопрос хороший. Наверно для понимая успеешь ли ты попить кофе во время работы алгоритма или нет.

В планах реализовать приближенное вычисление спектра весов данных векторов. В целом ясно, что они будут распределены нормально, а вот найти коэффициенты и зависимость этих коэффициентов от количества векторов и их длины – задача поинтересней.