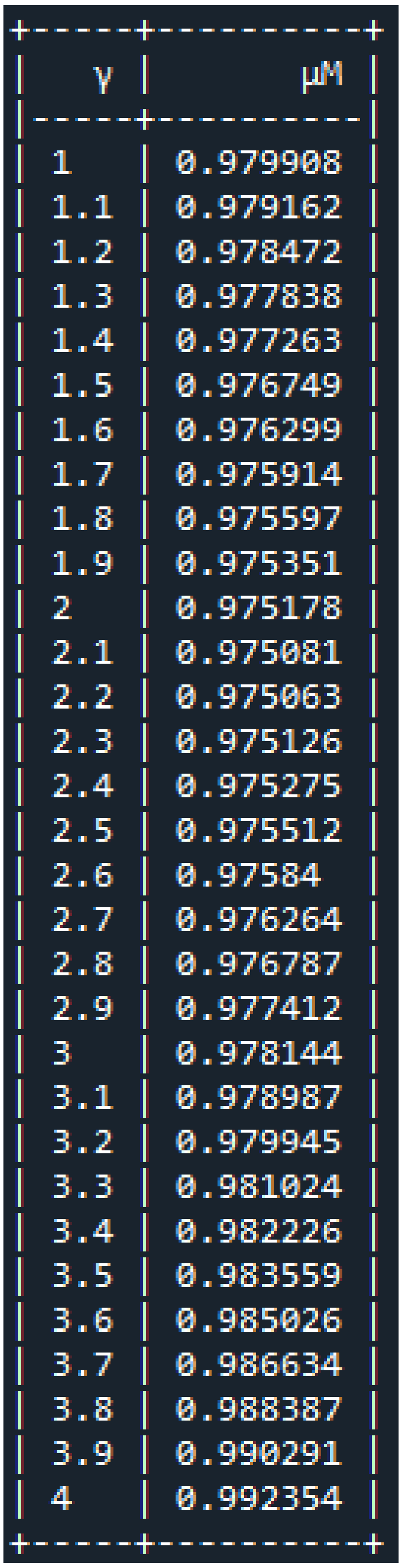
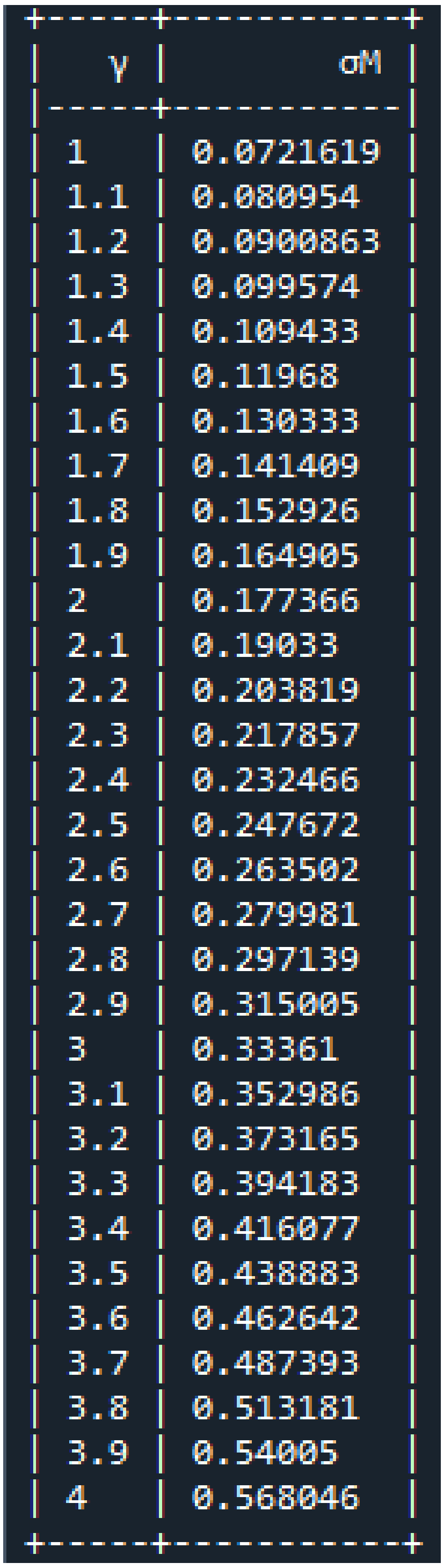
1) **Calculate μ_M and σ_M for each value of γ,**

μ_M =



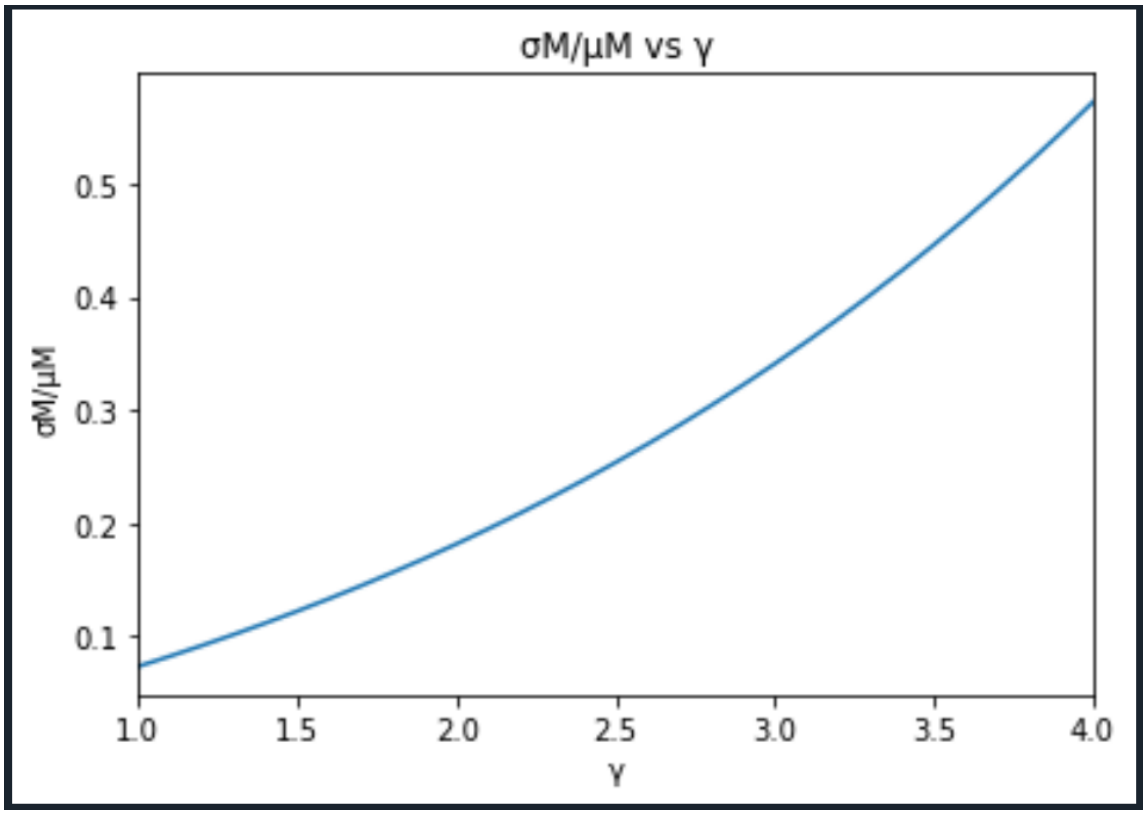
```
+-----+----------+
|   γ |       µM |
|-----+----------|
| 1   | 0.979908 |
| 1.1 | 0.979162 |
| 1.2 | 0.978472 |
| 1.3 | 0.977838 |
| 1.4 | 0.977263 |
| 1.5 | 0.976749 |
| 1.6 | 0.976299 |
| 1.7 | 0.975914 |
| 1.8 | 0.975597 |
| 1.9 | 0.975351 |
| 2   | 0.975178 |
| 2.1 | 0.975081 |
| 2.2 | 0.975063 |
| 2.3 | 0.975126 |
| 2.4 | 0.975275 |
| 2.5 | 0.975512 |
| 2.6 | 0.97584  |
| 2.7 | 0.976264 |
| 2.8 | 0.976787 |
| 2.9 | 0.977412 |
| 3   | 0.978144 |
| 3.1 | 0.978987 |
| 3.2 | 0.979945 |
| 3.3 | 0.981024 |
| 3.4 | 0.982226 |
| 3.5 | 0.983559 |
| 3.6 | 0.985026 |
| 3.7 | 0.986634 |
| 3.8 | 0.988387 |
| 3.9 | 0.990291 |
| 4   | 0.992354 |
+-----+----------+
```

σ_M =



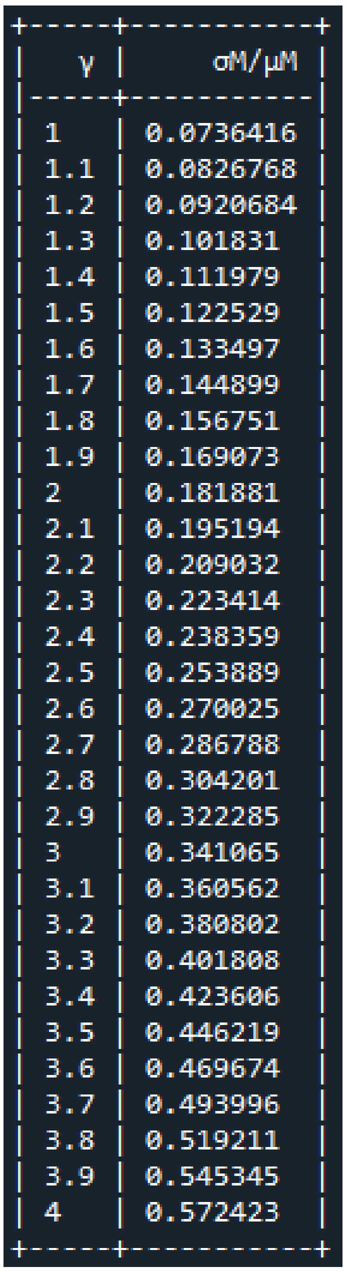
```
+------+------------+
|    γ |         σM |
|------+------------|
| 1    | 0.0721619  |
| 1.1  | 0.080954   |
| 1.2  | 0.0900863  |
| 1.3  | 0.099574   |
| 1.4  | 0.109433   |
| 1.5  | 0.11968    |
| 1.6  | 0.130333   |
| 1.7  | 0.141409   |
| 1.8  | 0.152926   |
| 1.9  | 0.164905   |
| 2    | 0.177366   |
| 2.1  | 0.19033    |
| 2.2  | 0.203819   |
| 2.3  | 0.217857   |
| 2.4  | 0.232466   |
| 2.5  | 0.247672   |
| 2.6  | 0.263502   |
| 2.7  | 0.279981   |
| 2.8  | 0.297139   |
| 2.9  | 0.315005   |
| 3    | 0.33361    |
| 3.1  | 0.352986   |
| 3.2  | 0.373165   |
| 3.3  | 0.394183   |
| 3.4  | 0.416077   |
| 3.5  | 0.438883   |
| 3.6  | 0.462642   |
| 3.7  | 0.487393   |
| 3.8  | 0.513181   |
| 3.9  | 0.54005    |
| 4    | 0.568046   |
+------+------------+
```

and plot σ_M/μ_M (on the vertical axis) vs γ (on the horizontal axis).



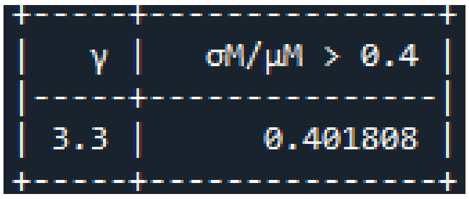
2) Find the smallest value of γ (in your data) for which $\sigma_M/\mu_M > 0.4$, so that the Hansen–Jagannathan bound is satisfied.

σ_M/μ_M =



```
+-----+-----------+
|   γ |     σM/μM |
|-----+-----------|
| 1   | 0.0736416 |
| 1.1 | 0.0826768 |
| 1.2 | 0.0920684 |
| 1.3 | 0.101831  |
| 1.4 | 0.111979  |
| 1.5 | 0.122529  |
| 1.6 | 0.133497  |
| 1.7 | 0.144899  |
| 1.8 | 0.156751  |
| 1.9 | 0.169073  |
| 2   | 0.181881  |
| 2.1 | 0.195194  |
| 2.2 | 0.209032  |
| 2.3 | 0.223414  |
| 2.4 | 0.238359  |
| 2.5 | 0.253889  |
| 2.6 | 0.270025  |
| 2.7 | 0.286788  |
| 2.8 | 0.304201  |
| 2.9 | 0.322285  |
| 3   | 0.341065  |
| 3.1 | 0.360562  |
| 3.2 | 0.380802  |
| 3.3 | 0.401808  |
| 3.4 | 0.423606  |
| 3.5 | 0.446219  |
| 3.6 | 0.469674  |
| 3.7 | 0.493996  |
| 3.8 | 0.519211  |
| 3.9 | 0.545345  |
| 4   | 0.572423  |
+-----+-----------+
```

smallest value of γ (in your data) =

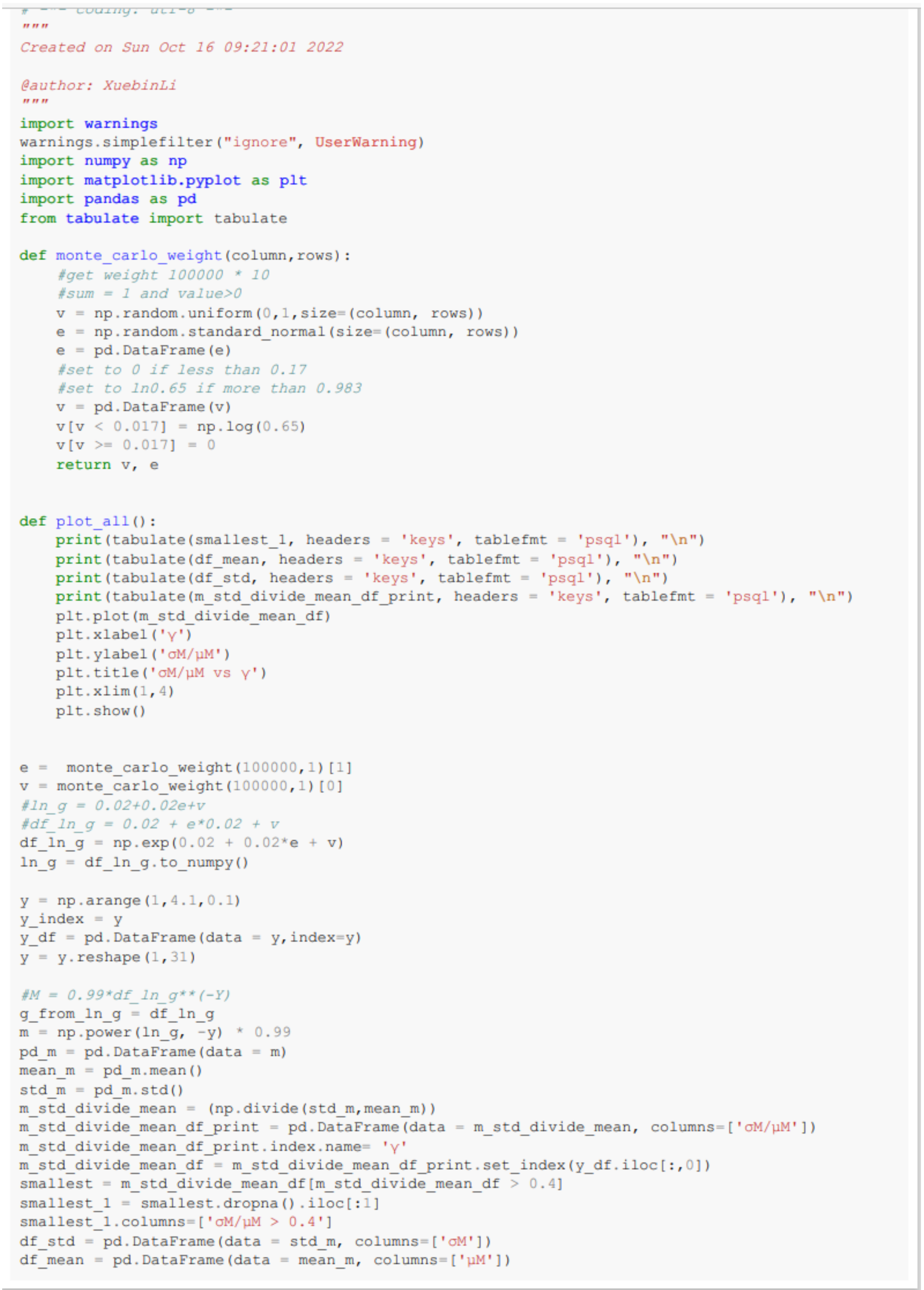


```
+-----+---------------+
|   γ |   σM/μM > 0.4 |
|-----+---------------|
| 3.3 |      0.401808 |
+-----+---------------+
```

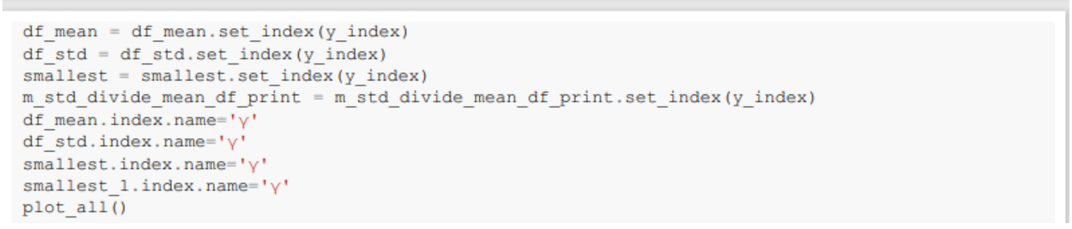
3) Explain the economic significance of this result.

Hansen–Jagannathan bound is a theorem that describes the ratio of standard deviation of a stochastic discount factor to its mean exceeds the Sharpe ratio of any portfolio. Hansen–Jagannathan bound is a mean-variance frontier. It allows us to say something about moments of the stochastic discount factor which is not observable. In terms of moments of returns that is observable in principle. If the observed Sharpe ratio say around 0.4, the bound tells us that the stochastic discount factor must be at least just as volatile. Left hand side of Hansen–Jagannathan is Sharpe ratio of any risky asset, while right hand side of Hansen–Jagannathan is "volatility ratio" for pricing kernel. But Hansen–Jagannathan also applies to any portfolio since pricing formula for Consumption CAPM applies to any portfolio.
Hence volatility ratio of pricing kernel cannot be less than highest Sharpe ratio out of all possible portfolios
Annual risk premium of around 7% and annual standard deviation of around 17% for U.S. stock market returns $\Longrightarrow$ Sharpe ratio of around 0.4, so pricing kernel is highly volatile
Pricing kernel has lower limit of zero but no upper limit $\Longrightarrow$ probability distribution should be heavily skewed on right side.

Appendix:



```
# -*- coding: utf-8 -*-
"""
Created on Sun Oct 16 09:21:01 2022

@author: XuebinLi
"""
import warnings
warnings.simplefilter("ignore", UserWarning)
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from tabulate import tabulate

def monte_carlo_weight(column,rows):
    #get weight 100000 * 10
    #sum = 1 and value>0
    v = np.random.uniform(0,1,size=(column, rows))
    e = np.random.standard_normal(size=(column, rows))
    e = pd.DataFrame(e)
    #set to 0 if less than 0.17
    #set to ln0.65 if more than 0.983
    v = pd.DataFrame(v)
    v[v < 0.017] = np.log(0.65)
    v[v >= 0.017] = 0
    return v, e


def plot_all():
    print(tabulate(smallest_1, headers = 'keys', tablefmt = 'psql'), "\n")
    print(tabulate(df_mean, headers = 'keys', tablefmt = 'psql'), "\n")
    print(tabulate(df_std, headers = 'keys', tablefmt = 'psql'), "\n")
    print(tabulate(m_std_divide_mean_df_print, headers = 'keys', tablefmt = 'psql'), "\n")
    plt.plot(m_std_divide_mean_df)
    plt.xlabel('γ')
    plt.ylabel('σM/μM')
    plt.title('σM/μM vs γ')
    plt.xlim(1,4)
    plt.show()


e =  monte_carlo_weight(100000,1)[1]
v = monte_carlo_weight(100000,1)[0]
#ln_g = 0.02+0.02e+v
#df_ln_g = 0.02 + e*0.02 + v
df_ln_g = np.exp(0.02 + 0.02*e + v)
ln_g = df_ln_g.to_numpy()

y = np.arange(1,4.1,0.1)
y_index = y
y_df = pd.DataFrame(data = y,index=y)
y = y.reshape(1,31)

#M = 0.99*df_ln_g**(-Y)
g_from_ln_g = df_ln_g
m = np.power(ln_g, -y) * 0.99
pd_m = pd.DataFrame(data = m)
mean_m = pd_m.mean()
std_m = pd_m.std()
m_std_divide_mean = (np.divide(std_m,mean_m))
m_std_divide_mean_df_print = pd.DataFrame(data = m_std_divide_mean, columns=['σM/μM'])
m_std_divide_mean_df_print.index.name= 'γ'
m_std_divide_mean_df = m_std_divide_mean_df_print.set_index(y_df.iloc[:,0])
smallest = m_std_divide_mean_df[m_std_divide_mean_df > 0.4]
smallest_1 = smallest.dropna().iloc[:1]
smallest_1.columns=['σM/μM > 0.4']
df_std = pd.DataFrame(data = std_m, columns=['σM'])
df_mean = pd.DataFrame(data = mean_m, columns=['μM'])
```



```
df_mean = df_mean.set_index(y_index)
df_std = df_std.set_index(y_index)
smallest = smallest.set_index(y_index)
m_std_divide_mean_df_print = m_std_divide_mean_df_print.set_index(y_index)
df_mean.index.name='γ'
df_std.index.name='γ'
smallest.index.name='γ'
smallest_1.index.name='γ'
plot_all()
```