In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [6]:
```
#prameters setting
#lng = 0.02 + 0.02*epsilon + nu
#M = 0.99*g^(-gamma)
#M = 0.99*exp(b_0*(1-g))
epsilon_num = int(1e4)
pre_nu_num = int(1e4)

dis_val = np.log(0.6)
nor_val = 0
dis_probability = 0.02

consumption_growth_intercept = 0.02
consumption_growth_slope = 0.02

#ε (epsilon)
df_consumption_growth = pd.DataFrame(np.random.randn(epsilon_num),columns=[

lst_pre_nu = list(np.random.rand(pre_nu_num))

lst_nu = list(map(lambda x: dis_val if x < dis_probability else nor_val, ls

df_consumption_growth1 = df_consumption_growth.copy()
df_consumption_growth1["nu"] = lst_nu
df_consumption_growth1

#lng = 0.02 + 0.02*epsilon + nu
df_consumption_growth2 = df_consumption_growth1.copy()
df_consumption_growth2["lng"] = (consumption_growth_intercept
                                  +
                                  consumption_growth_slope*df_consumption_gr
                                  +
                                  df_consumption_growth2["nu"]
                                 )

df_consumption_growth3 = df_consumption_growth2.copy()
df_consumption_growth3["g"] = np.exp(df_consumption_growth3["lng"])

#parameters settings
gamma_start = 0
gamma_end = 4.1
gamma_increment = 0.1

M_coefficient = 0.99

#M = 0.99*g^(-gamma)
lst_cols = []

df_consumption_growth4 = df_consumption_growth3.copy()

for gamma in np.arange(gamma_start,gamma_end, gamma_increment):
    col = "M_" + str(round(gamma,1))
    lst_cols.append(col)
    df_consumption_growth4[col] = M_coefficient*np.exp((1-df_consumption_gr

df_M = df_consumption_growth4[lst_cols].apply(lambda x: [np.mean(x), np.st
df_M.columns = ["mu_M", "xigma_M", "xigma_M/mu_M"]
df_M
```

Out[6]:

	mu_M	xigma_M	xigma_M/mu_M
M_0.0	0.990000	1.713074e-13	1.730378e-13
M_0.1	0.988852	6.198524e-03	6.268406e-03
M_0.2	0.987743	1.260288e-02	1.275927e-02
M_0.3	0.986677	1.922218e-02	1.948174e-02
M_0.4	0.985652	2.606585e-02	2.644528e-02
M_0.5	0.984672	3.314373e-02	3.365965e-02
M_0.6	0.983738	4.046599e-02	4.113495e-02
M_0.7	0.982850	4.804323e-02	4.888157e-02
M_0.8	0.982010	5.588645e-02	5.691025e-02
M_0.9	0.981221	6.400707e-02	6.523208e-02
M_1.0	0.980483	7.241694e-02	7.385845e-02
M_1.1	0.979798	8.112841e-02	8.280114e-02
M_1.2	0.979169	9.015427e-02	9.207225e-02
M_1.3	0.978596	9.950781e-02	1.016842e-01
M_1.4	0.978082	1.092029e-01	1.116500e-01
M_1.5	0.977630	1.192538e-01	1.219826e-01
M_1.6	0.977240	1.296754e-01	1.326956e-01
M_1.7	0.976915	1.404834e-01	1.438031e-01
M_1.8	0.976658	1.516937e-01	1.553192e-01
M_1.9	0.976470	1.633230e-01	1.672586e-01
M_2.0	0.976354	1.753888e-01	1.796365e-01
M_2.1	0.976313	1.879091e-01	1.924680e-01
M_2.2	0.976350	2.009027e-01	2.057691e-01
M_2.3	0.976466	2.143890e-01	2.195559e-01
M_2.4	0.976666	2.283883e-01	2.338448e-01
M_2.5	0.976951	2.429216e-01	2.486528e-01
M_2.6	0.977326	2.580109e-01	2.639968e-01
M_2.7	0.977792	2.736787e-01	2.798946e-01
M_2.8	0.978354	2.899489e-01	2.963639e-01
M_2.9	0.979015	3.068459e-01	3.134230e-01
M_3.0	0.979779	3.243951e-01	3.310901e-01
M_3.1	0.980649	3.426233e-01	3.493842e-01
M_3.2	0.981630	3.615578e-01	3.683241e-01
M_3.3	0.982724	3.812274e-01	3.879291e-01
M_3.4	0.983938	4.016618e-01	4.082187e-01
M_3.5	0.985274	4.228920e-01	4.292125e-01
M_3.6	0.986738	4.449502e-01	4.509302e-01
M_3.7	0.988335	4.678697e-01	4.733919e-01

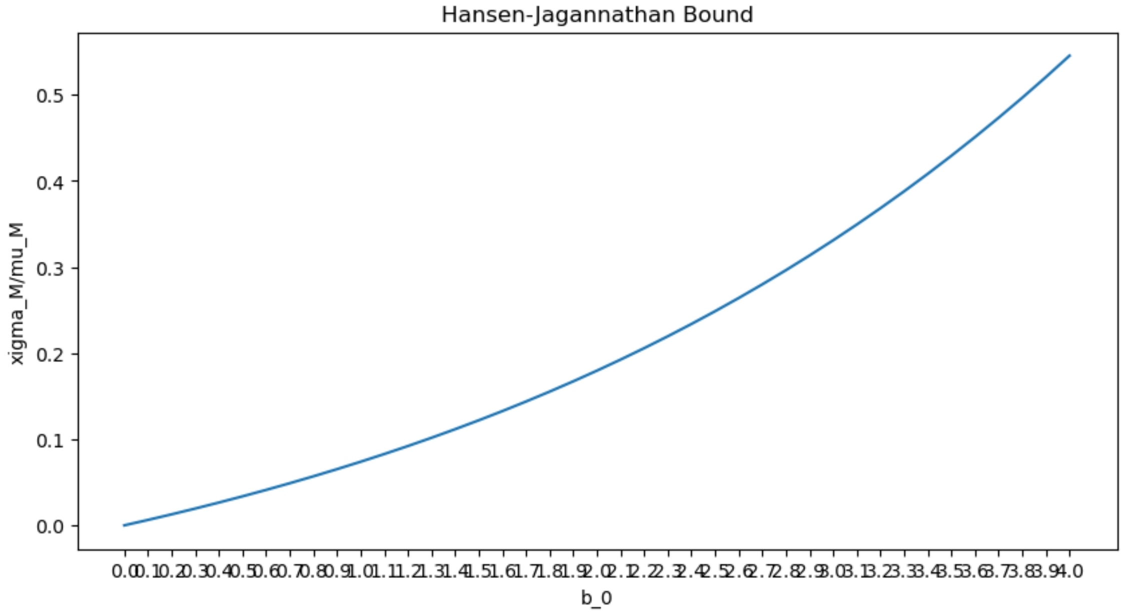
	mu_M	xigma_M	xigma_M/mu_M
M_3.8	0.990068	4.916854e-01	4.966176e-01
M_3.9	0.991945	5.164335e-01	5.206273e-01
M_4.0	0.993969	5.421514e-01	5.454412e-01

In [8]:
```
df_M1 = df_M.copy()
df_M1["gamma"] = [round(gamma, 1) for gamma in np.arange(gamma_start, gamma

plt.figure(figsize=[10,5])
plt.plot(df_M1["gamma"], df_M1["xigma_M/mu_M"])
plt.xticks(np.arange(gamma_start,gamma_end,gamma_increment))

plt.xlabel("b_0")
plt.ylabel("xigma_M/mu_M")
plt.title("Hansen-Jagannathan Bound")

plt.show()
```



Explain (in words) how to find the smallest value of b0 for which the Hansen–Jagannathan bound is satisfied, and report the result for your data set.

need to know sharpe ratio first, and let say sharp ratio = 0.4, then find samllest b0 that make xigma_M/mu_M larger than sharpe ratio

In [10]:
```
smallest_gamma = np.min(df_M1[df_M1["xigma_M/mu_M"]>0.4]["gamma"])
print(smallest_gamma)
```

```
3.4
```