

File Edit View History Bookmarks Tools Help

Bay X build a Zoom ENME4 probab measur Inferen Warnin What m Bound Trendin Stats Stats Getting img to +

https://colab.research.google.com/drive/1lwJ0VgpojNaCn9udRDH2v9BZtt60Fm_s#scrollTo=...

M Personal M University M Extra ELMS TESTUDO WileyPLUS Google Drive Your Orders LinkedIn YouTube Netflix - Watch TV Sh... MHEC Hofmann Engineering ... > Disk Editing

Bayesian System Identification of a Harmonic Oscillator using Probabilistic Programming

In this problem you will use probabilistic programming to infer the mass, spring constant, damping, and initial velocity of a Mass-Spring-Damper system, given only noisy measurements of that system.

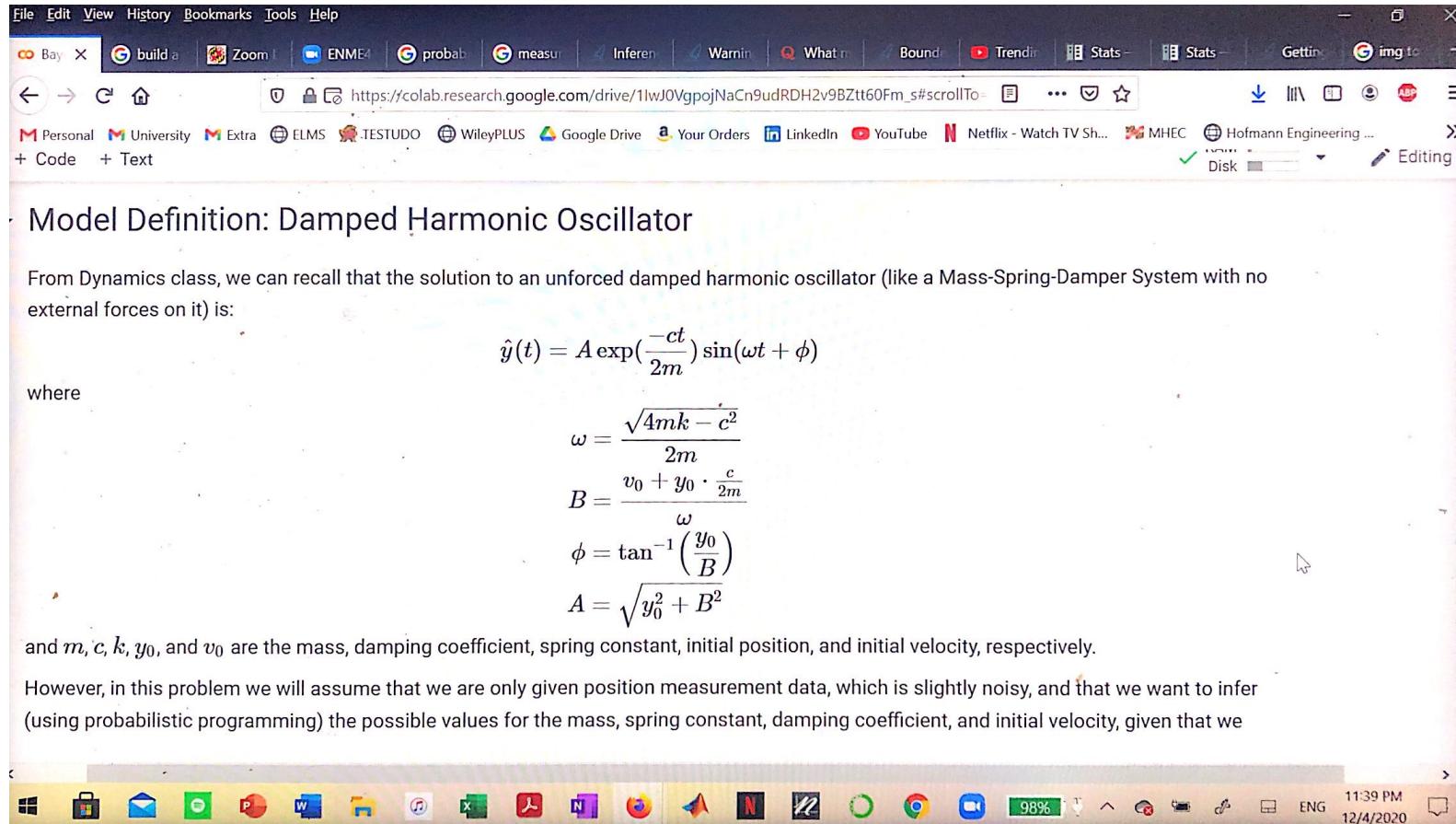
For this, you can use any probabilistic programming library that you like (e.g., PyMC3, Tensorflow Probability, Pyro, Edward, Turing, etc.). If you want to leverage some of the examples from class then we happened to use PyMC3, however this problem is still solvable using any other library.

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

Model Definition: Damped Harmonic Oscillator



11:39 PM 12/4/2020 ENG

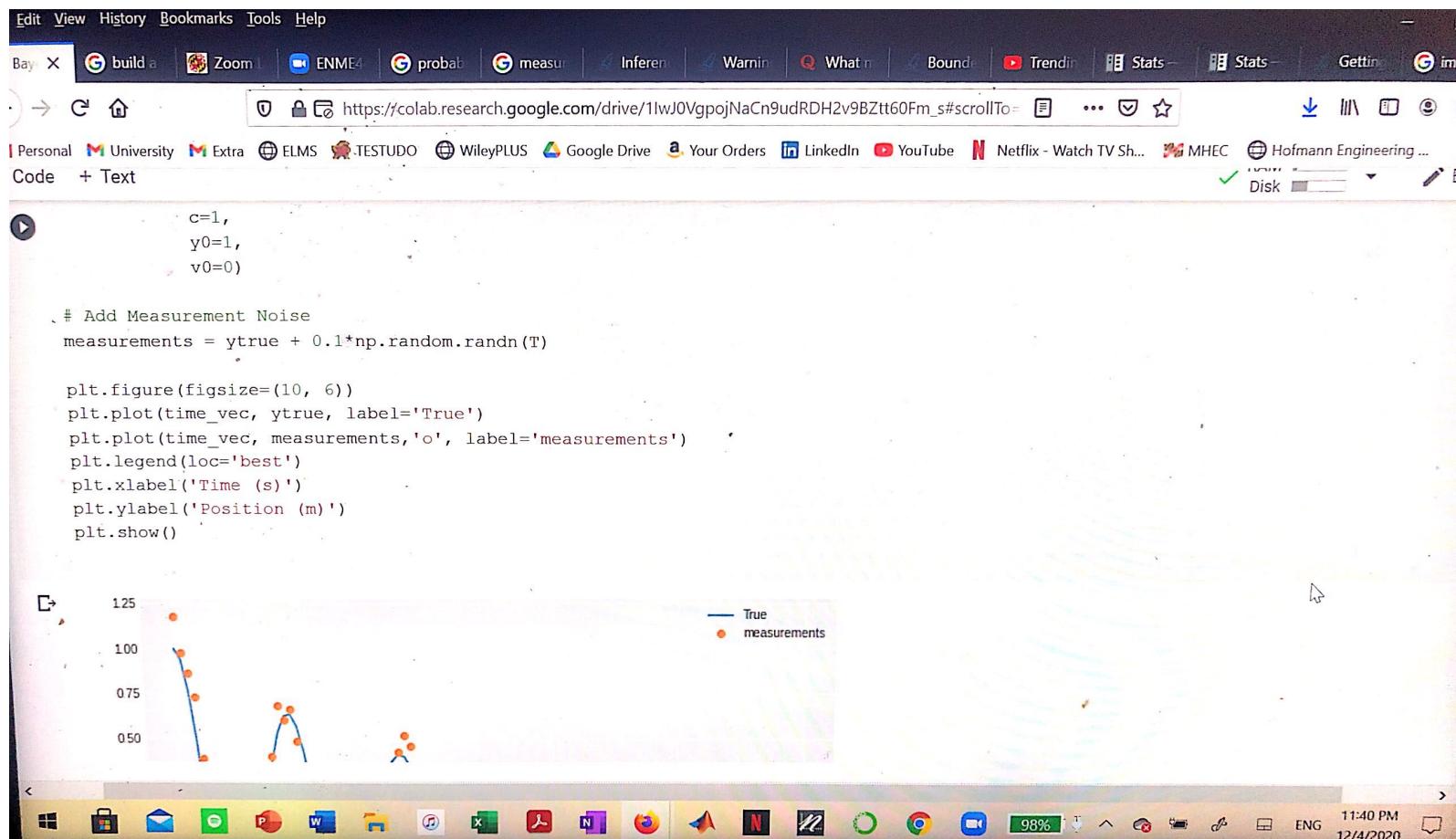


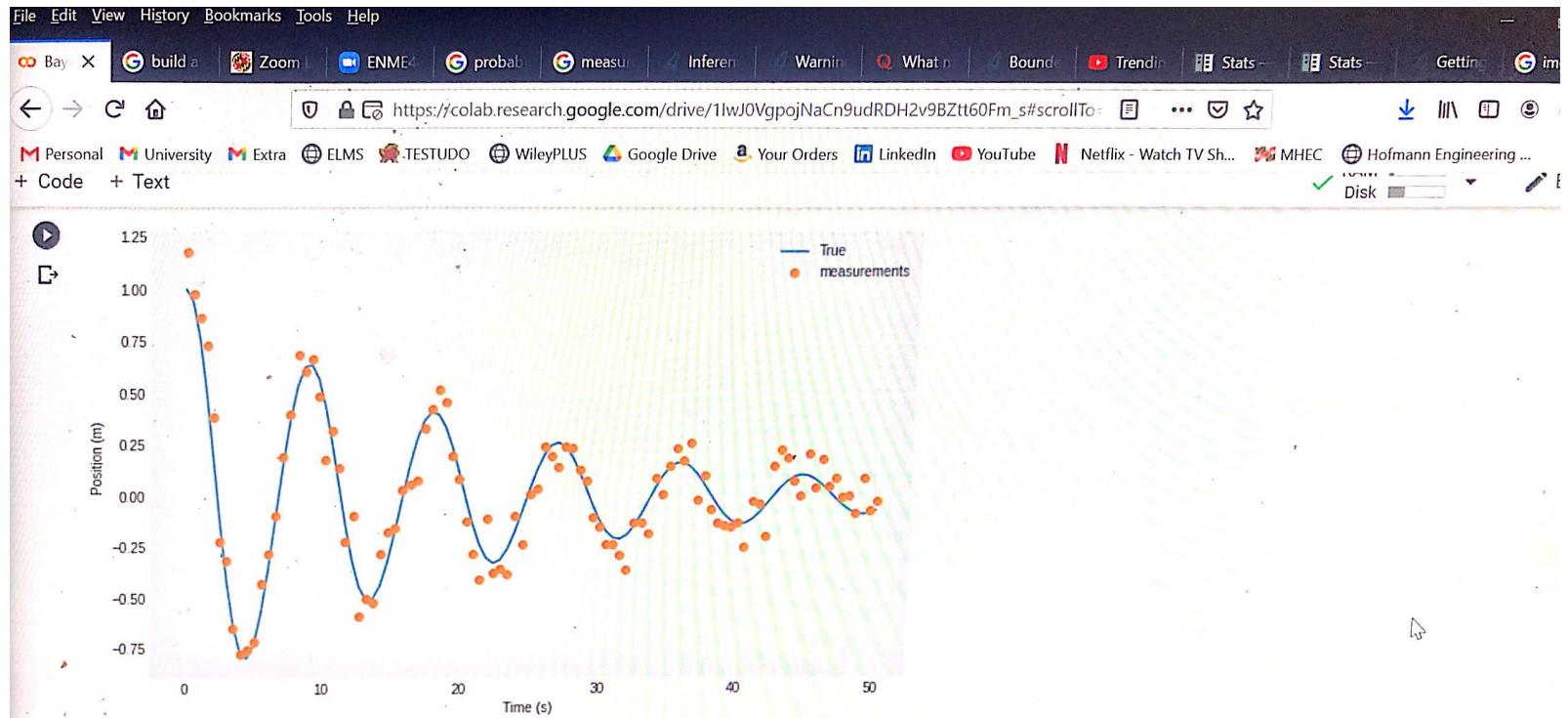
The screenshot shows a Google Colab notebook interface. The URL in the address bar is https://colab.research.google.com/drive/1lwJ0VqpojNaCn9udRDH2v9BZtt60Fm_s#scrollTo. The notebook contains the following Python code:

```
# Code to generate the noisy measurements
# Number of Timesteps
T = 100
np.random.seed(0)

# Textbook solution to Unforced Mass-Spring-Damper Harmonic Oscillator
def MSD(t,m,k,c,y0,v0):
    omega = np.sqrt(4*m*k-c**2)/(2*m)
    B = (v0+y0*c/(2*m))/omega
    phi = np.arctan(y0/B)
    A = np.sqrt(y0**2 + B**2)
    y = A*np.exp(-c*t/(2*m))*np.sin(omega*t+phi)
    return y

# Simulate the true system:
time_vec = np.linspace(0, 50, T)
ytrue = MSD(time_vec,
            m=10,
            k=5,
            c=1,
            v0=1)
```





Task1: Build the Probabilistic Model



The screenshot shows a Google Colab notebook interface. The title of the notebook is "Task1: Build the Probabilistic Model". The content discusses building a probabilistic model of position as a function of time using a chosen probabilistic programming language, with prior beliefs about system parameters. It provides the mathematical formula for the joint probability distribution and lists the prior distributions for each parameter. Below the text, there is a list of four steps for implementing the model using probabilistic programming. The bottom of the screen shows the Windows taskbar with various application icons.

Task1: Build the Probabilistic Model

Using the probabilistic programming language of your choice, build a probabilistic model of the position as a function of time, with the following prior beliefs about the other system parameters:

$$P(\mathbf{y}, m, k, c, v_0 | T) = \sum_{t \in T} \mathcal{N}(y_t | \hat{y}(t, m, k, c, v_0), \sigma^2) P(m) P(c) P(k) P(v_0) P(\text{sigma})$$

where

$$\begin{aligned} P(m) &\sim \mathcal{N}(\mu_m = 10, \sigma_m = 0.01) \\ P(k) &\sim \mathcal{N}(\mu_k = 10, \sigma_k = 5) \\ P(c) &\sim \mathcal{N}(\mu_c = 5, \sigma_c = 2) \\ P(v_0) &\sim \mathcal{N}(\mu_{v_0} = 0, \sigma_{v_0} = 5) \\ P(\text{sigma}) &\sim \text{Gamma}(\alpha = 0.2, \beta = 1) \end{aligned}$$

While this seems like perhaps a complicated model, using probabilistic programming this basically means:

1. Instantiate probabilistic variables `m`, `c`, `k`, `v0` and `sigma` with the appropriate Normal and Gamma prior distributions.
2. Use them when computing the dynamics solution above.
3. Set the output variable `y` to be the observed variable using `measurements` as the observations.
4. Sample the model with N=15K samples. (e.g., I would suggest the NUTS sampler with 15K samples and using around 2000 tuning

4. Sample the model with N=15K samples. (e.g., I would suggest the NUTS sampler with 15K samples and using around 2000 tuning samples). Use at least two chains for comparison sake later.

Hint: If you have issues getting your sampling chains to converge, you can try: (1) increasing the number of samples, (2) increasing the number of tuning samples you allow `tuning = ####`, or (3) increasing the target acceptance rate for the MCMC samples (`target_accept=0.95` or `.99`, etc.). You can also increase the number of independent chains via `chains=3` or `4` or other numbers, though I would try increasing the number of samples first.

Hint 2: You will notice that our prior belief about the mass is really tight ($\sigma_m = 0.01$). This is on purpose and not a typo. Task 3 will explore what happens if this is not the case. In more realistic problems, we may not know the mass with such accuracy, though for this task we will assume that we do.

Hint 3: Particularly eagle-eyed students may question the use of, e.g., the Normal distribution on parameters like Mass or damping coefficient, etc., since, technically, that could mean there is a probability of getting a negative mass or stiffness or damping etc. which is not physically sound and so forth. This is true, and it would be in fact far more reasonable to pick a distribution on the positive reals, such as the Log-Normal, Half-Cauchy, Truncated Normal, Exponential, or others. Feel free to do so if you like, but for simplicity sake I am asking everyone to do Normals for now.

```
[53]: import pymc3 as pm  
basic_model = pm.Model()
```

The screenshot shows a Google Colab notebook interface. The code in the cell is as follows:

```
import pymc3 as pm
basic_model = pm.Model()

with basic_model:
    # Priors for unknown model parameters
    y0=1
    m = pm.Normal('mass', mu=10, sd=0.01)
    k = pm.Normal('spring-constant', mu=10, sd=5)
    c = pm.Normal('damping-coefficient', mu=5, sd=2)
    v0 = pm.Normal('initial velocity', mu=0, sd=5)
    sigma =pm.Gamma('sigma', 0.2, 1)
    y = MSD(time_vec,
            m,
            k,
            c,
            y0,
            v0)

    # Add Measurement Noise
    measurements = y + 0.1*np.random.randn(T)
    y_obs = pm.Normal('Y_obs', sd=sigma, observed = measurements)
```

```
[53]
    m,
    k,
    c,
    y0,
    v0)

# Add Measurement Noise
measurements = y + 0.1*np.random.randn(T)
y_obs = pm.Normal('Y_obs', sd=sigma, observed = measurements)

%%time
# Sampling
with basic_model:
    # draw posterior samples
    # unsufficient time to run full basic model. Wall time too long. Produced reduced model not including measurements or y_obs.
    #not well calibrated
    mc_trace = pm.sample(15000, chains=4, target_accept=0.97, tune=2000)

Auto-assigning NUTS sampler...
Initializing NUTS using jitter+adapt_diag...
Sequential sampling (4 chains in 1 job)
NUTS: [sigma, initial velocity, damping-coefficient, spring-constant, mass]
Sampling chain 0, 0 divergences: 100%|██████████| 17000/17000 [00:47<00:00, 360.42it/s]
Sampling chain 1, 0 divergences: 100%|██████████| 17000/17000 [00:38<00:00, 447.24it/s]
```

[54] Auto-assigning NUTS sampler...

```
Initializing NUTS using jitter+adapt_diag...
Sequential sampling (4 chains in 1 job)
NUTS: [sigma, initial velocity, damping-coefficient, spring-constant, mass]
Sampling chain 0, 0 divergences: 100%|██████████| 17000/17000 [00:47<00:00, 360.42it/s]
Sampling chain 1, 0 divergences: 100%|██████████| 17000/17000 [00:38<00:00, 447.24it/s]
Sampling chain 2, 0 divergences: 100%|██████████| 17000/17000 [00:33<00:00, 511.39it/s]
Sampling chain 3, 0 divergences: 100%|██████████| 17000/17000 [00:48<00:00, 348.09it/s]
CPU times: user 2min 47s, sys: 2.1 s, total: 2min 50s
Wall time: 2min 48s
```

Task 2: Inspect the model

Now that you have your samples, go ahead and inspect the output of the probabilistic program. Specifically:

Plot the trace plots of the uncertain parameters.

```
[56] pm.traceplot(mc_trace)

/usr/local/lib/python3.6/dist-packages/arviz/plots/backends/matplotlib/distplot.py:38: UserWarning: Argument backend_kwargs has not
"Argument backend kwargs has not effect in matplotlib.plot dist."
```

File Edit View History Bookmarks Tools Help

Bay X G build a Zoom ENME4 G probab G measur 4 Inferen 4 Warnin Q What in 4 Bound Stats Stats Getting G img to +

https://colab.research.google.com/drive/1lwJ0VgpojNaCn9udRDH2v9BZtt60Fm_s#scrollTo

M Personal M University M Extra ELMS TESTUDO WileyPLUS Google Drive Your Orders LinkedIn YouTube Netflix - Watch TV Sh... MHEC Hofmann Engineering ... >

+ Code + Text Disk Editing

FIT THE trace plots of the uncertain parameters.

```
pm.traceplot(mc_trace)
```

/usr/local/lib/python3.6/dist-packages/arviz/plots/backends/matplotlib/distplot.py:38: UserWarning: Argument backend_kwarg has not
"Argument backend_kwarg has not effect in matplotlib.plot_dist"
array([[[<matplotlib.axes._subplots.AxesSubplot object at 0x7f123d818828>,
<matplotlib.axes._subplots.AxesSubplot object at 0x7f123d5d39b0>],
<matplotlib.axes._subplots.AxesSubplot object at 0x7f123d7014e0>,
<matplotlib.axes._subplots.AxesSubplot object at 0x7f123d631a58>],
<matplotlib.axes._subplots.AxesSubplot object at 0x7f123d68c9e8>,
<matplotlib.axes._subplots.AxesSubplot object at 0x7f123d6be588>],
<matplotlib.axes._subplots.AxesSubplot object at 0x7f123d73db00>,
<matplotlib.axes._subplots.AxesSubplot object at 0x7f123d7b2fd0>],
<matplotlib.axes._subplots.AxesSubplot object at 0x7f123d562630>,
<matplotlib.axes._subplots.AxesSubplot object at 0x7f123d594ba8>]),
dtype=object)

mass mass

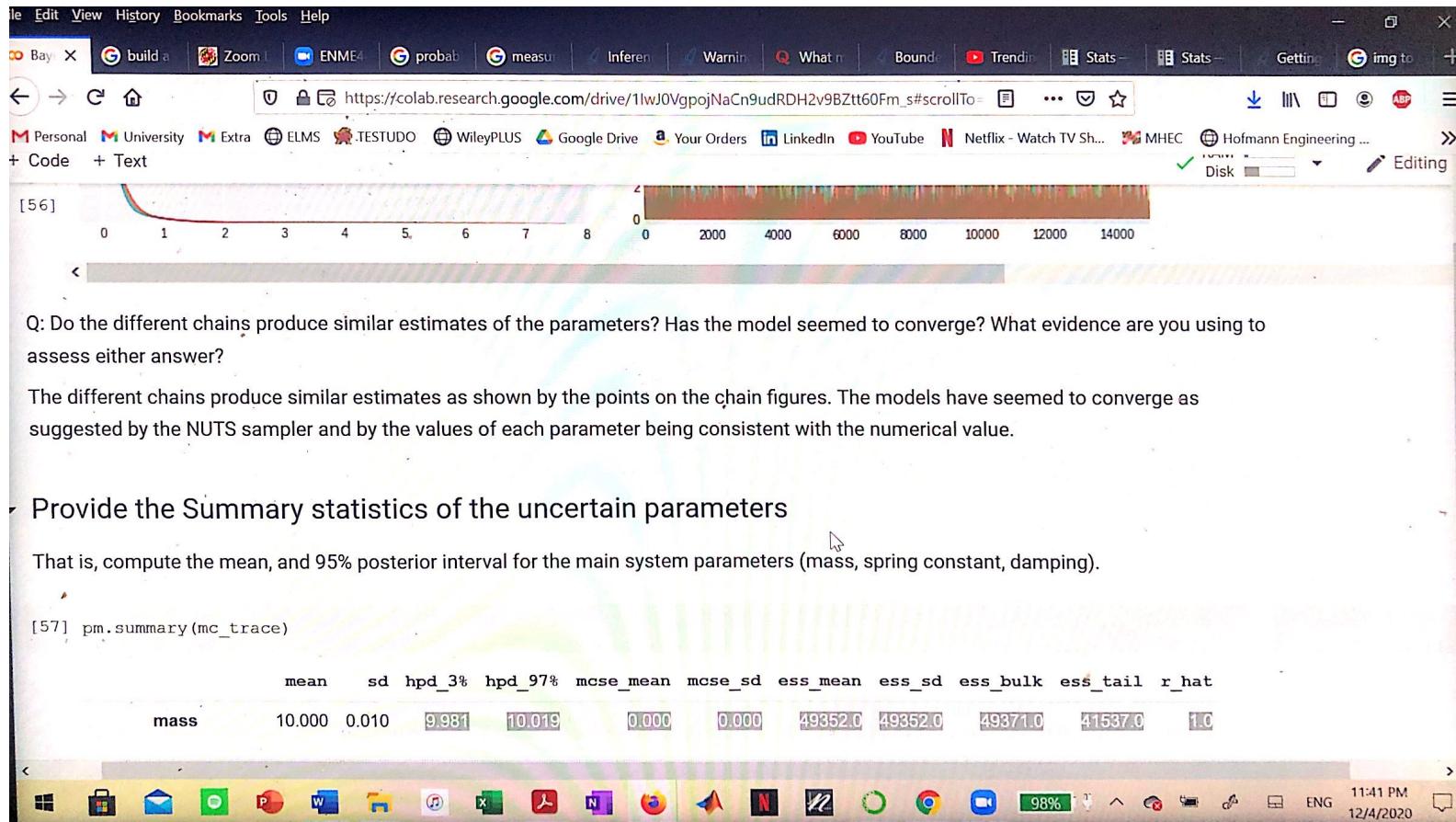
10.04
10.02
10.00
9.98

Windows Mail Word Excel Powerpoint OneDrive File Explorer MATLAB Notepad N 98% ENG 11:40 PM 12/4/2020





Scanned with CamScanner



File Edit View History Bookmarks Tools Help

Bay X build a Zoom ENME4 probab measur Inferen Warnin Whatin Bound Trendin Stats Stats Getting img to +

https://colab.research.google.com/drive/1lwJ0VgpojNaCn9udRDH2v9BZtt60Fm_s#scrollTo

M Personal M University M Extra ELMS TESTUDO WileyPLUS Google Drive Your Orders LinkedIn YouTube Netflix - Watch TV Sh... MHEC Hofmann Engineering ... >

+ Code + Text Disk Editing

Provide the summary statistics of the uncertain parameters

That is, compute the mean, and 95% posterior interval for the main system parameters (mass, spring constant, damping).

```
pm.summary(mc_trace)
```

	mean	sd	hpdi_3%	hpdi_97%	mcse_mean	mcse_sd	ess_mean	ess_sd	ess_bulk	ess_tail	r_hat
mass	10.000	0.010	9.981	10.019	0.000	0.000	49352.0	49352.0	49371.0	41537.0	1.0
spring-constant	9.979	4.994	0.551	19.347	0.024	0.017	43407.0	41412.0	43406.0	39395.0	1.0
damping-coefficient	4.992	1.992	1.190	8.669	0.009	0.007	47474.0	45102.0	47476.0	40328.0	1.0
initial velocity	0.017	5.048	-9.563	9.429	0.022	0.022	52104.0	26759.0	52120.0	38076.0	1.0
sigma	0.204	0.450	0.000	0.940	0.002	0.001	60060.0	55206.0	17419.0	11660.0	1.0

Plot the 95% CI bound on the predicted true function

Using the samples you gathered while running your probabilistic program, produce a plot that compares (with time on the x-axis):

1. The provided measurements (measurements)

The screenshot shows a Google Colab notebook interface. The top bar includes File, Edit, View, History, Bookmarks, Tools, and Help. The toolbar has various icons for search, zoom, and navigation. The URL in the address bar is https://colab.research.google.com/drive/1lwJ0VgpojNaCn9udRDH2v9BZtt60Fm_s#scrollTo. The sidebar on the left has sections for Personal, University, Extra, ELMS, TESTUDO, WileyPLUS, Google Drive, Your Orders, LinkedIn, YouTube, Netflix - Watch TV Sh..., MHEC, Hofmann Engineering..., Code, and Text. A sigma value of 0.204 is displayed. The main area contains the following text:

Plot the 95% CI bound on the predicted true function

Using the samples you gathered while running your probabilistic program, produce a plot that compares (with time on the x-axis):

1. The provided measurements (`measurements`)
2. The true function (`ytrue`)
3. The mean predicted function (from your model)
4. The 95% CI upper bound on the function over time (from your model)
5. The 95% CI lower bound on the function over time (from your model).

Ideally, your 95% CI should contain the true function, if your probabilistic program is well-calibrated (i.e., has an appropriate measure of its own uncertainty)

```
[60] #not well calibrated as 95% CI should contain the function
ytrue = MSD(time_vec,
             m=10,
             k=5,
             c=1.
```

The bottom status bar shows system icons, battery level at 98%, ENG, 11:41 PM, and 12/4/2020.

Ideally, your 95% CI should contain the true function, if your probabilistic program is well-calibrated (i.e., has an appropriate measure of its own uncertainty)

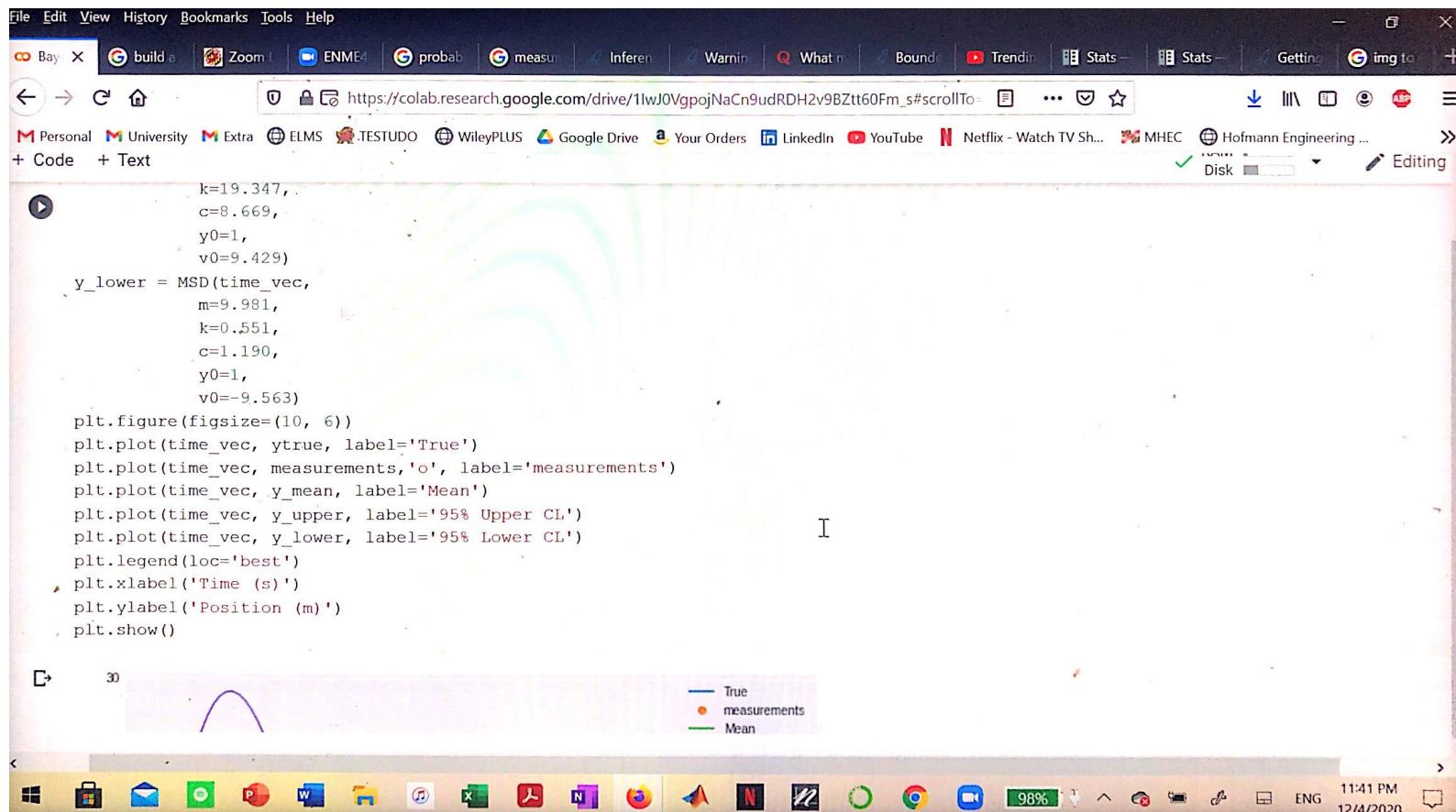
```
#not well calibrated as 95% CI should contain the function
ytrue = MSD(time_vec,
             m=10,
             k=5,
             c=1,
             y0=1,
             v0=0)
measurements = ytrue + 0.1*np.random.randn(T)
y_mean = MSD(time_vec,
              m=10.000,
              k=9.979,
              c=4.992,
              y0=1,
              v0=0.017)
y_upper = MSD(time_vec,
               m=10.019,
               k=19.347,
               c=8.669,
```

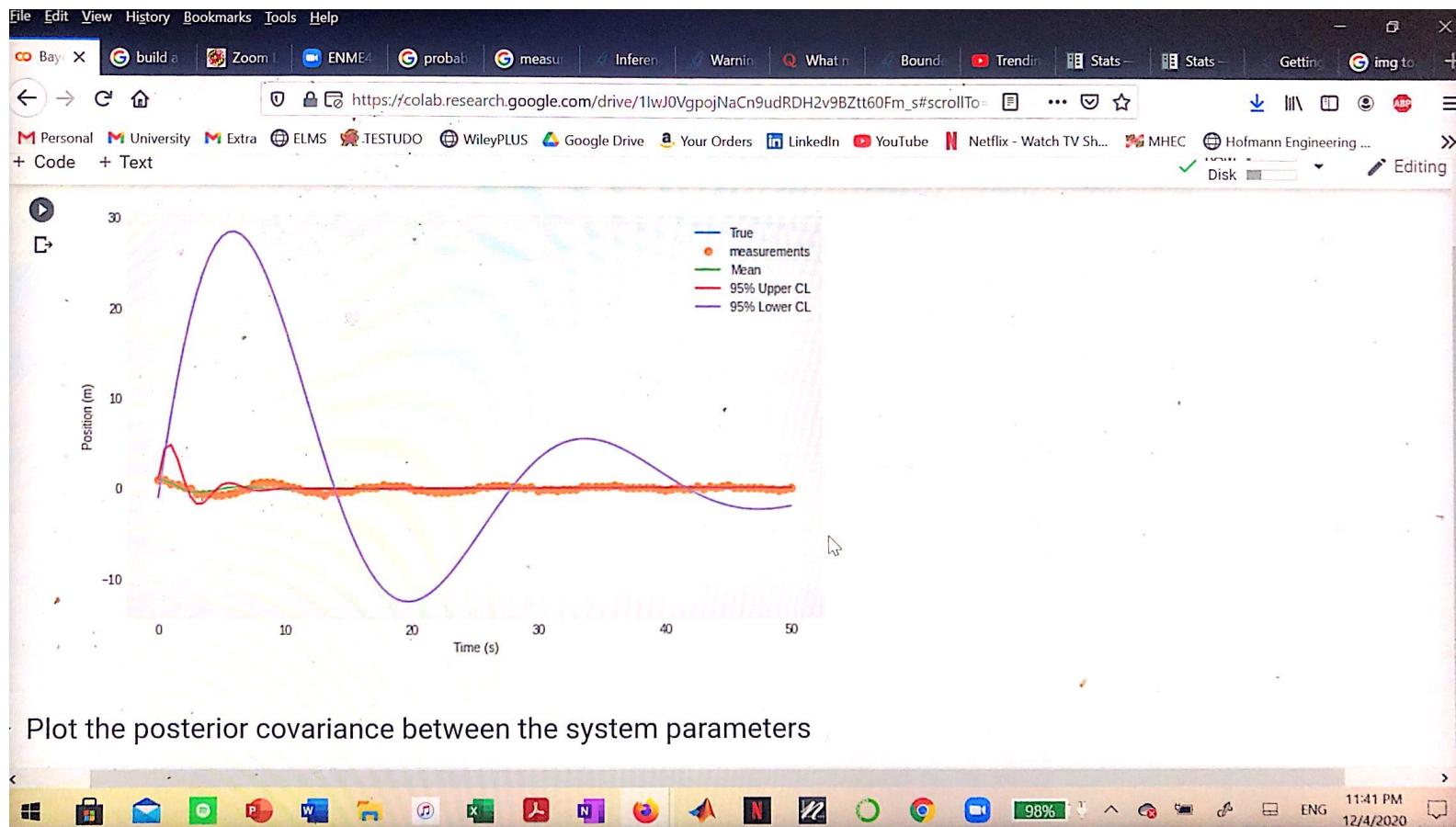
File Edit View History Bookmarks Tools Help

Bay X G build a Zoom ENME4 probab measur Infern Warnin What n Bound Trendir Stats Stats Getting img to +

Personal University Extra ELMS TESTUDO WileyPLUS Google Drive Your Orders LinkedIn YouTube Netflix - Watch TV Sh... MHEC Hofmann Engineering ... + Code + Text Disk

c=1,
y0=1,
v0=0)
measurements = ytrue + 0.1*np.random.randn(T)
y_mean = MSD(time_vec,
m=10.000,
k=9.979,
c=4.992,
y0=1,
v0=0.017)
y_upper = MSD(time_vec,
m=10.019,
k=19.347,
c=8.669,
y0=1,
v0=9.429)
y_lower = MSD(time_vec,
m=9.981,
k=0.551,
c=1.190,
y0=1,
v0=-9.563)
plt.figure(figsize=(10, 6))





The screenshot shows a Google Colab notebook interface. The title bar includes 'File Edit View History Bookmarks Tools Help'. The toolbar has various icons for file operations like 'Build a', 'Zoom', 'ENME4', 'probab', 'Inferen', 'Warnin', 'What.m', 'Bounds', 'Trendin', 'Stats', 'Getting', and 'img to'. The address bar shows the URL https://colab.research.google.com/drive/1lwJ0VgpojNaCn9udRDH2v9BZtt60Fm_s#scrollTo. The sidebar contains links to 'Personal', 'University', 'Extra', 'ELMS', 'TESTUDO', 'WileyPLUS', 'Google Drive', 'Your Orders', 'LinkedIn', 'YouTube', 'Netflix - Watch TV Sh...', 'MHEC', 'Hofmann Engineering ...', 'Code', and 'Text'. The status bar at the bottom right shows 'Editing', 'Disk', '11:41 PM', 'ENG', and '12/4/2020'.

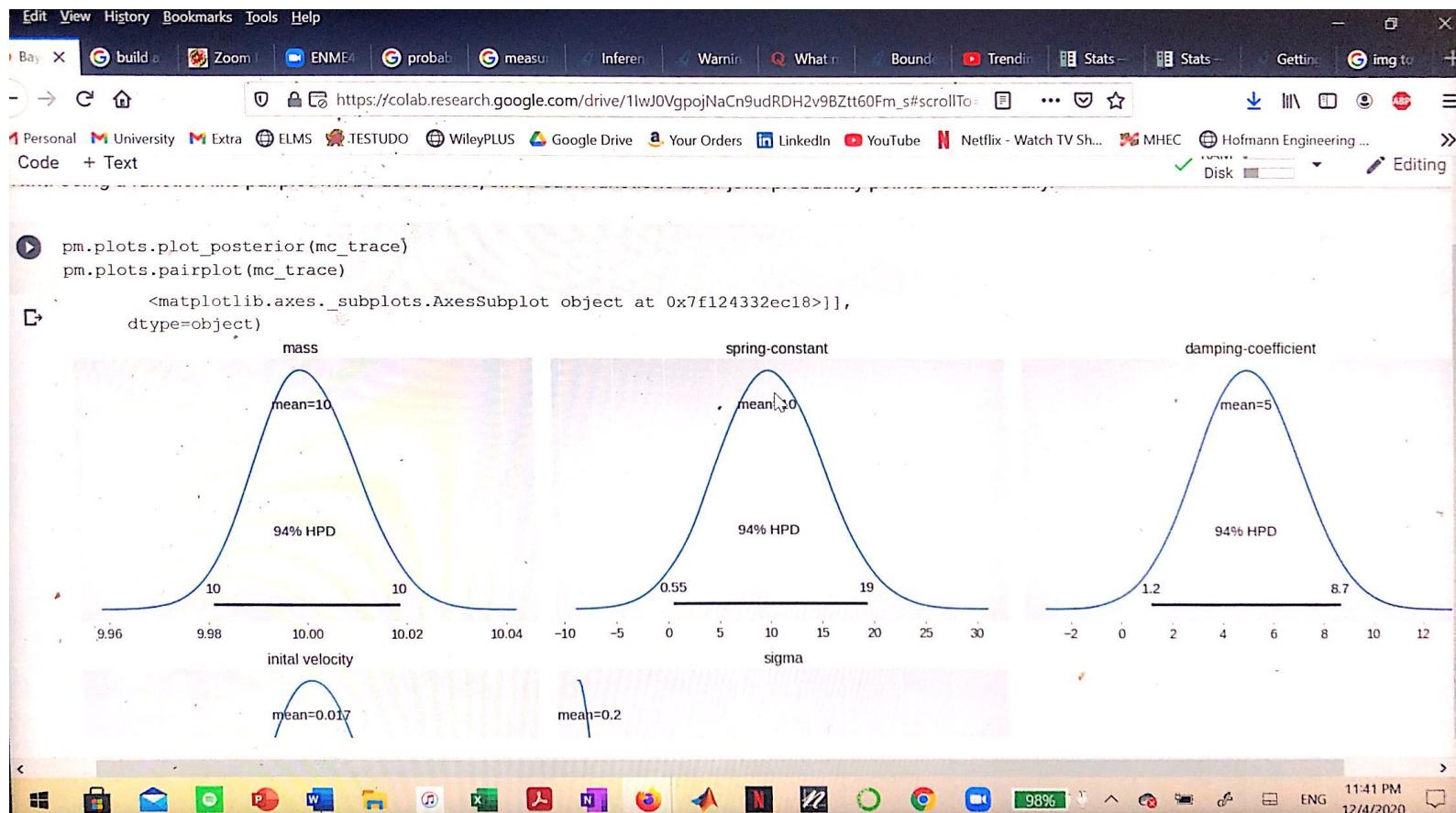
Plot the posterior covariance between the system parameters

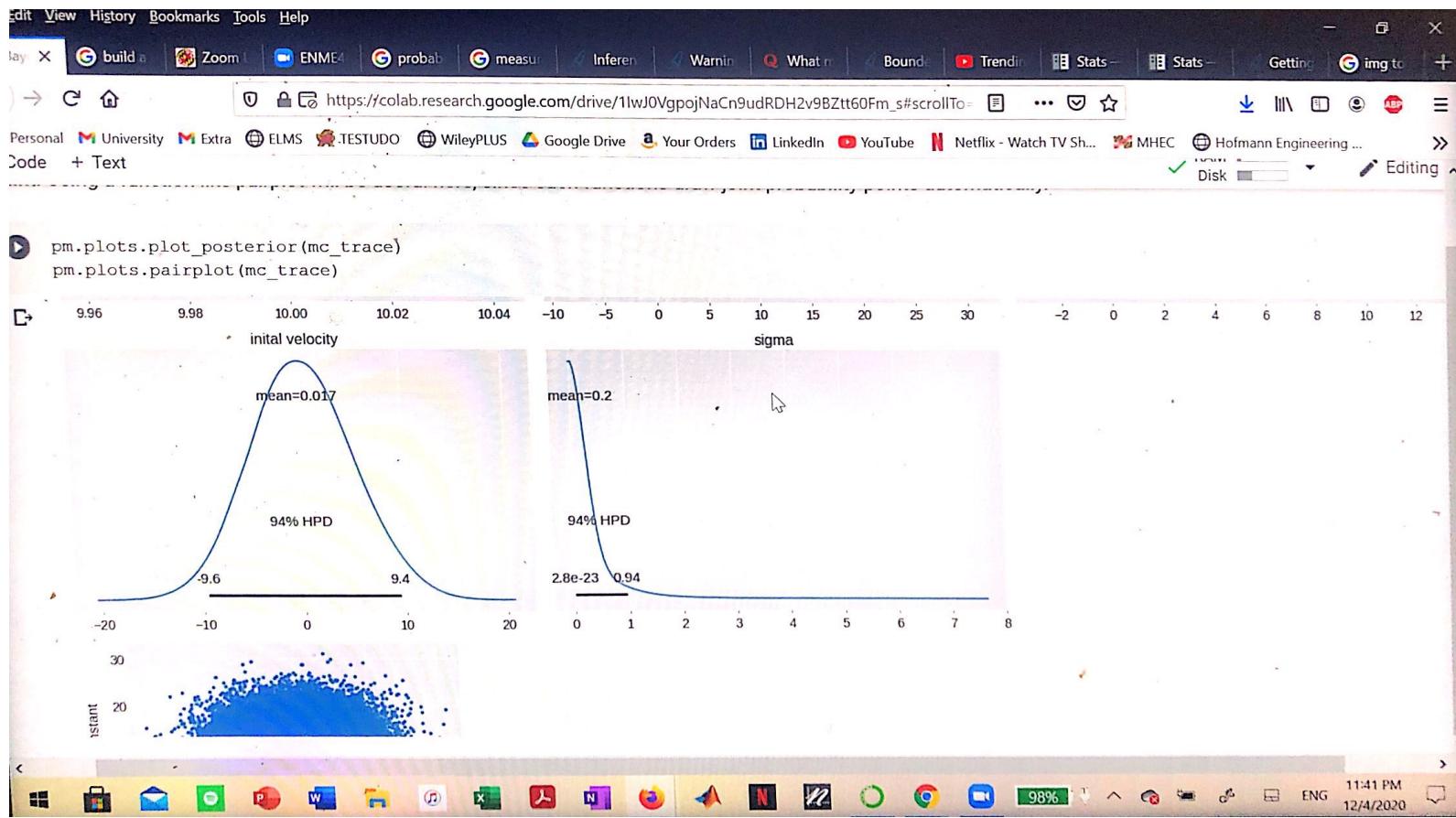
Specifically, plot the pairwise joint probability density between the posterior distributions of the variables for mass, spring constant, and damping. You can do this by plotting for each point on the trace, the pairwise relationship between m vs k, k vs c, and m vs c.

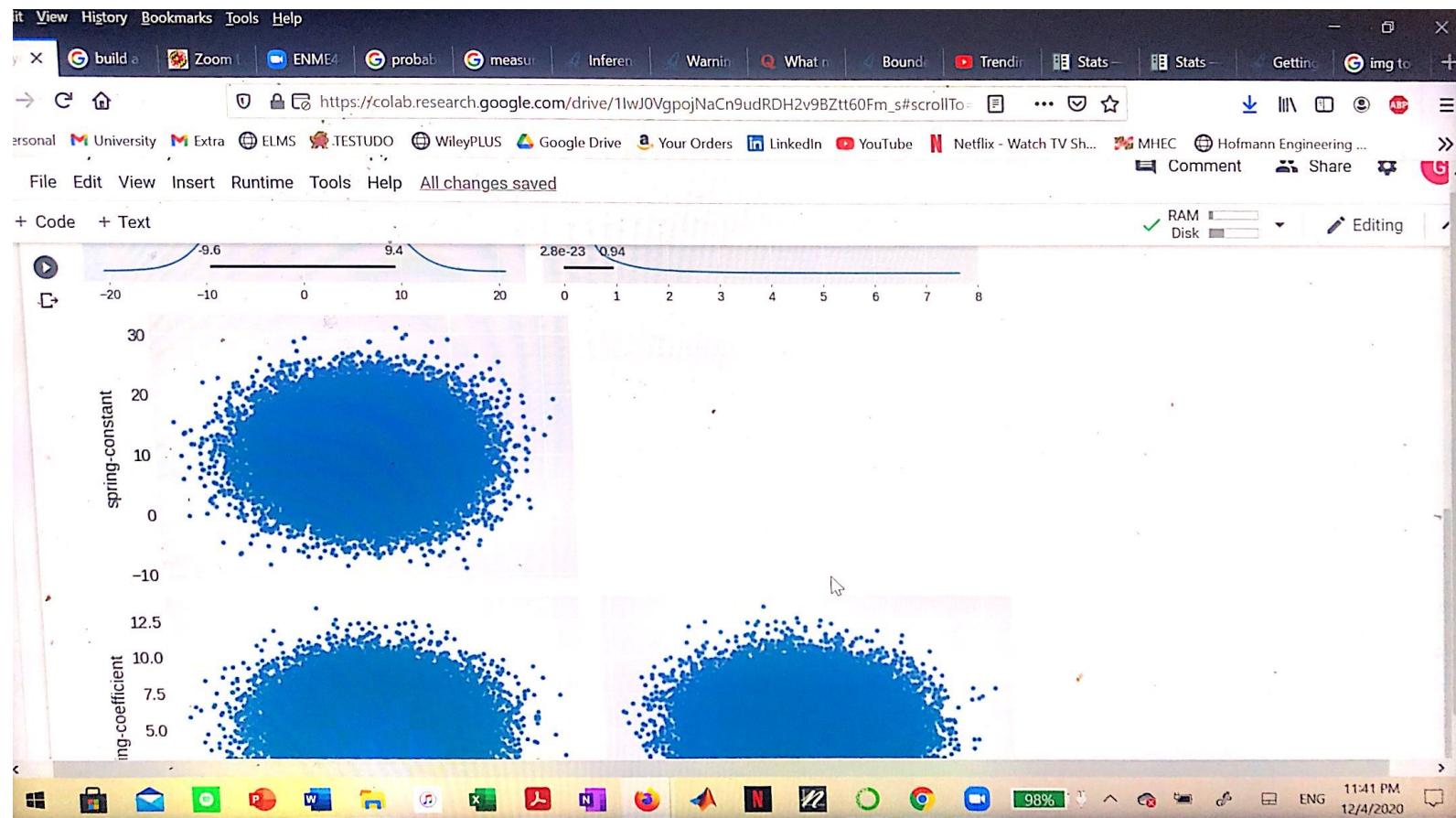
Hint: Using a function like pairplot will be useful here, since such functions draw joint probability points automatically.

```
▶ pm.plots.plot_posterior(mc_trace)
pm.plots.pairplot(mc_trace)

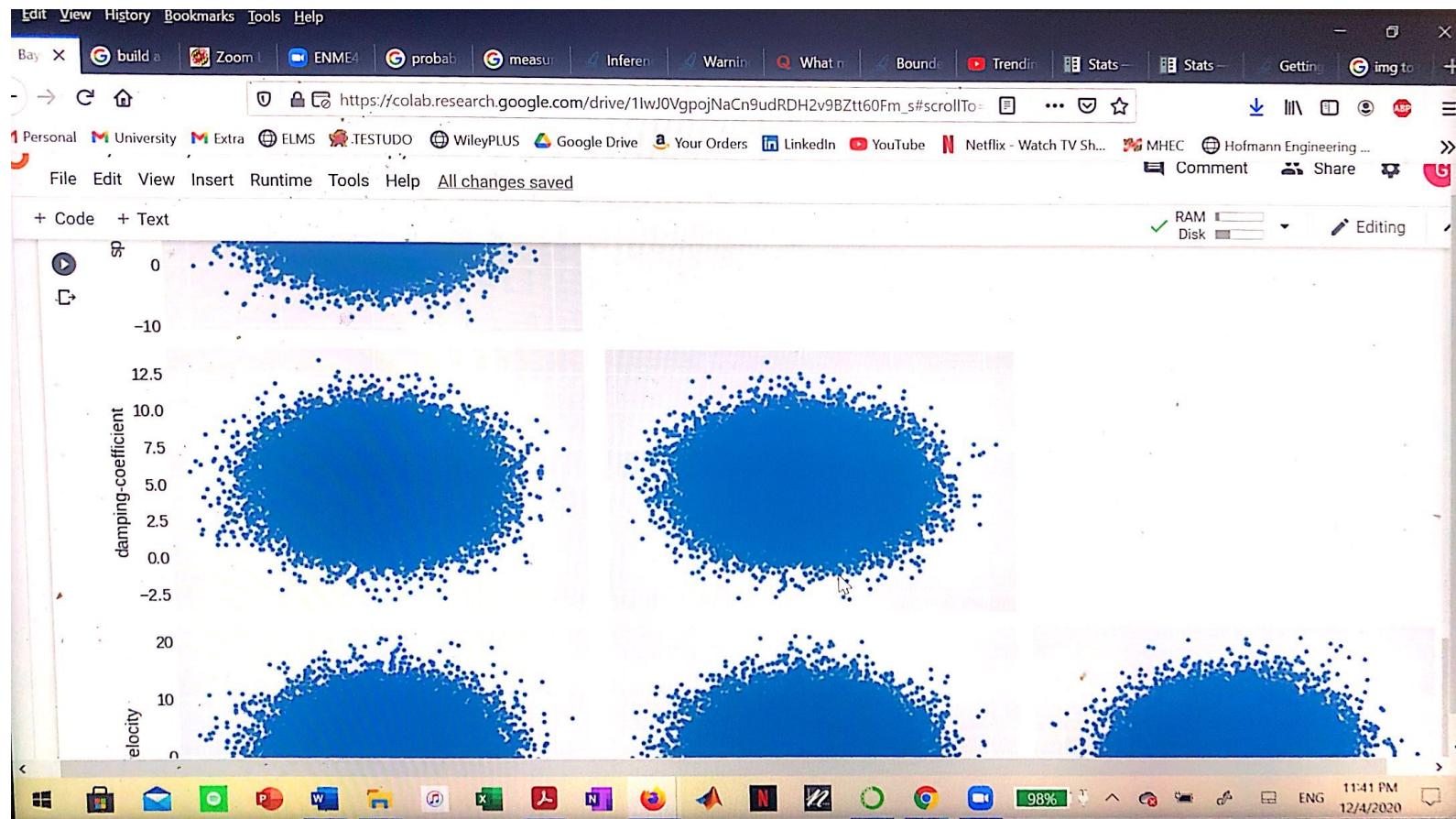
↳ array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f12431f30f0>,
          <matplotlib.axes._subplots.AxesSubplot object at 0x7f1250fd4eb8>,
          <matplotlib.axes._subplots.AxesSubplot object at 0x7f1245c8f1d0>,
          <matplotlib.axes._subplots.AxesSubplot object at 0x7f124319a048>],
         [<matplotlib.axes._subplots.AxesSubplot object at 0x7f1251c39978>,
          <matplotlib.axes._subplots.AxesSubplot object at 0x7f1245dca6a0>,
          <matplotlib.axes._subplots.AxesSubplot object at 0x7f1255874748>,
          <matplotlib.axes._subplots.AxesSubplot object at 0x7f1250fd1710>],
         [<matplotlib.axes._subplots.AxesSubplot object at 0x7f1246f53e80>,
          <matplotlib.axes._subplots.AxesSubplot object at 0x7f12473fc5c0>,
          <matplotlib.axes._subplots.AxesSubplot object at 0x7f12431f30f0>]]
```



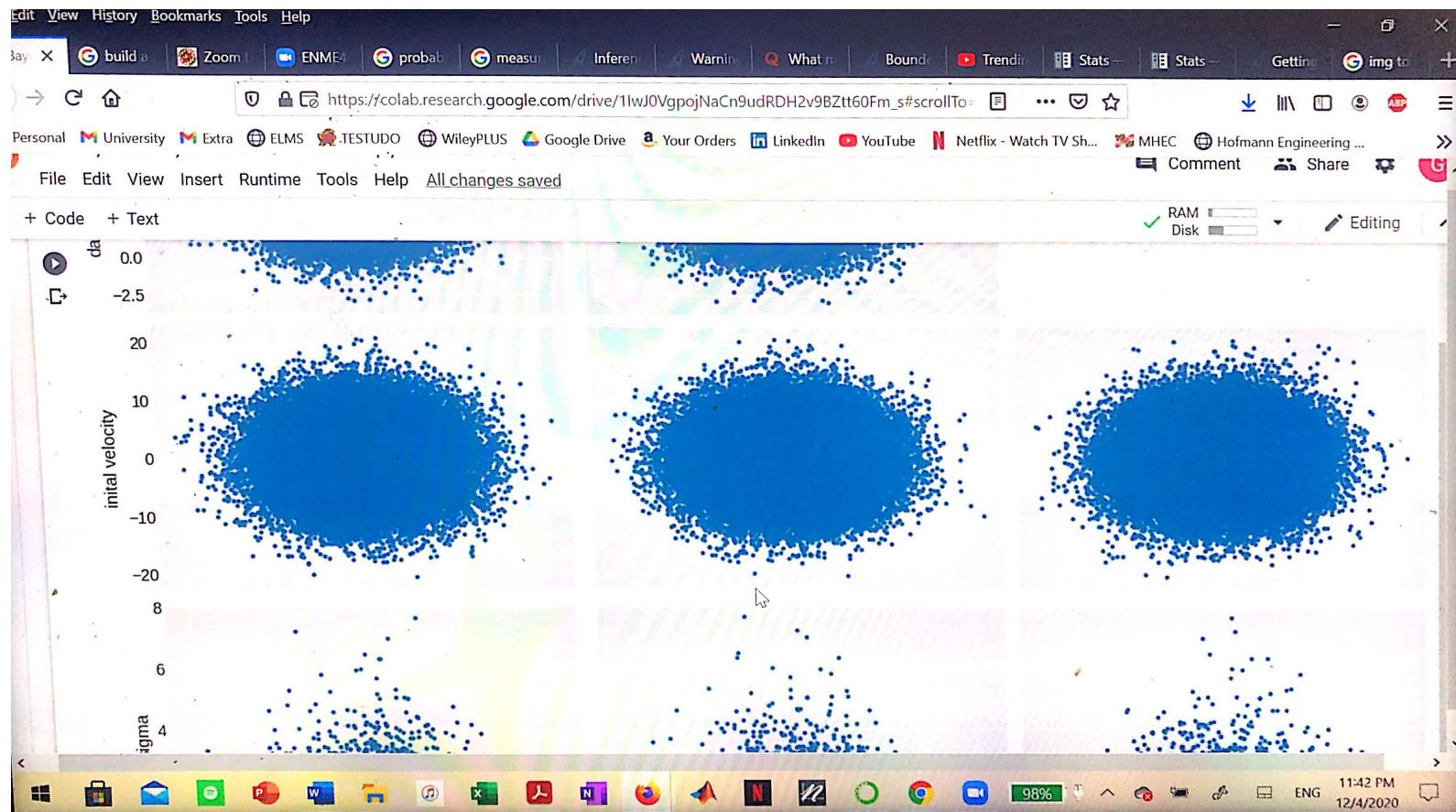




Scanned with CamScanner



Scanned with CamScanner



Scanned with CamScanner

