phase_transition_locator

April 1, 2025

[1]:

```
import matplotlib as mp
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import math
import scipy as sp
import time
import os
import corner
```

[2]:

```
save_plot = False
```

[3]:

```
#opens .dat files to be converted into CSVs
#change pathprefix on different computers
mass_radius_files = "/home/tplohr/proj/SF 24-25/diettim NMMA master␣
 ↪EOS-chiralEFT_MTOV/"
EOS_files = "/home/tplohr/proj/SF 24-25/data/Peps_024_new/"
def loadfile(files, number):
    file = open(files + str(number) + '.dat', 'r')
    return file
```

[4]:

```
#number of EOSs
numofEOS = 5000
#creates an empty array to contain all EOSs
EOSs = [None] * numofEOS
#EOSs contains the M-R curves and actual_EOSs contains pressure, energy␣
 ↪density, and number density
actual_EOSs = [None] * numofEOS
#fills EOSs and actual_EOSs with arrays of values
for i in range(numofEOS):
    #pandas dataframes to compile data from .dat files
    df = pd.read_csv(loadfile(mass_radius_files, i+1), delimiter = '\t', names␣
 ↪= ["radius", "mass", "deformability"])
    dataframe = pd.read_csv(loadfile(EOS_files, i+1), delimiter = '\t', names =␣
 ↪["num_density", "pressure", "erg_density"])
    #EOSs[i] is an array of two arrays for mass and radius
    EOSs[i] = np.array([df["radius"], df["mass"]])
    #actual_EOSs[i] is an array of many values
    #fourth element is is the sound speed throughout, fifth element is whether␣
 ↪there is a phase transition or not,
    #sixth element is start epsilon, seventh element is end epsilon, eight␣
 ↪element is delta epsilon, ninth element
    #is max sound speed, tenth is epsilon at max sound speed, eleventh is sound␣
 ↪speed after the phase transition
    actual_EOSs[i] = np.array([dataframe["num_density"]/0.16,␣
 ↪dataframe["erg_density"], dataframe["pressure"], np.
 ↪zeros(len(dataframe["erg_density"])-1), 0, 0, 0, 0, 0, 0, 0], dtype=object)

#EOSs[i] is EOS_i
#EOSs[i][0] is the list of radius values of EOS_i
#EOSs[i][1] is the list of mass values of EOS_i
#EOSs[i][0][j] is the radius value indexed j of EOS_i
#EOSs[i][1][j] is the mass value indexed j of EOS_i
```

[5]:

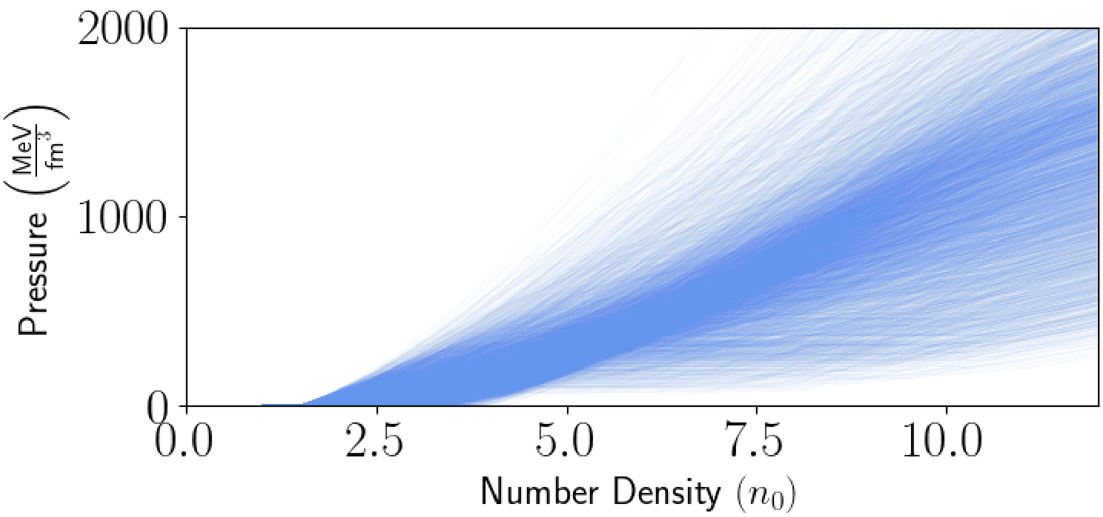
```
#saved probability values of the EOSs
#step B is the maximum mass constraint, step C is the NICER 1 constraint, step␣
 ↪D is the LIGO constraint
files = []
files.append("Step_B/01-21_09.txt")
files.append("Step_C/01-21_16.txt")
files.append("Step_D/01-21_06.txt")
files.append("NICER2/01-20_23.txt")
files.append("NICER2+3/01-20_19.txt")
#saved central pressure values from the TOV solver
file2 = "TOV/02-28_07.txt"
fileprefix = "/home/tplohr/proj/SF 24-25/saves/"

#sets the paths of each file
paths = []
for file in files:
    paths.append(os.path.join(fileprefix, file))
path2 = os.path.join(fileprefix, file2)

#sets P_EOS_given_obs as an array of probability arrays
#the first array has probabilities of 1/5000 since every EOS is equally likely
P_EOS_given_obs = []
P_EOS_given_obs.append(np.ones(5000)/5000)
for path in paths:
    P_EOS_given_obs.append(np.loadtxt(path))
#loads central pressures
central_pressures = np.loadtxt(path2)
```

[6]:

```
#plots each EOS in number density and pressure
plt.rcParams['figure.figsize'] = [8,4]
plt.xlim([0,12])
plt.ylim([0,2000])
#plt.xscale('log')
#plt.yscale('log')
plt.xlabel(r"Number Density $(n_0)$", fontsize=18)
plt.ylabel(r"Pressure $\left(\frac{\text{MeV}}{\text{fm}^{3}}\right)$",␣
 ↪fontsize=18)
for i in range(numofEOS):
    plt.plot(actual_EOSs[i][0], actual_EOSs[i][2], 'cornflowerblue', linewidth=.
 ↪02)
plt.tight_layout()
#plt.savefig("/home/tplohr/proj/SF 24-25/figs/EOSs/pressure_n_0_EOSs")
```



[7]:

```
#phase transitions after 1.5 sat density or 150 MeV/fm^3

##finds the slope of every point in a EOS
def find_dpdeps(EOS_number):
    #number of points
    num_points = len(actual_EOSs[EOS_number][1])-1
    #dpdeps is a np array with one less point that the number of eps points
    dpdeps = np.zeros(num_points)
    #for each point, evaluate the slope
    for i in range(num_points):
        #pressure and eps at two consecutive points
        p1 = actual_EOSs[EOS_number][2][i]
        p2 = actual_EOSs[EOS_number][2][i+1]
        eps1 = actual_EOSs[EOS_number][1][i]
        eps2 = actual_EOSs[EOS_number][1][i+1]
        #calculate slope
        dpdeps[i] = (p2-p1)/(eps2-eps1)
    return dpdeps

##check for a phase transition
def check_phase_trans(EOS_number):
    #initalize phase_trans which becomes true if there is a phase transition
    phase_trans = False
    #counter for how many consecutive points have a zero slope
    counter = 0
    #threshold value that slope has to be under (since the slope wont be exactly zero)
    threshold_slope = 10**-3
    #threshold length of the phase transition
    threshold_eps_diff = 50
    #the section of the EOS where the phase transition must be to be inside the core
    #the energy density is at least 150 MeV/fm^-3 and goes up to the central pressure
    threshold_min_eps = 150
    threshold_max_eps = central_pressures[EOS_number]
    #get dpdeps values
    dpdeps = find_dpdeps(EOS_number)
    #stores the values inside actual_EOSs for later analyzing
    actual_EOSs[EOS_number][3] = dpdeps.copy()
    #calculates the max sound speed of the EOS and the energy density at this point for analyzing
    max_sound_speed = np.max(dpdeps)
    eps_at_max_speed = actual_EOSs[EOS_number][0][np.argmax(dpdeps)]

    ##algorithm to check for a phase transition
    #loop over each point
    for i in range(len(dpdeps)):
        #curr_eps is the current epsilon value
        curr_eps = actual_EOSs[EOS_number][1][i]
        #if the slope is less than the threshold and the current epsilon is past 150, add to the counter
        if (dpdeps[i] < threshold_slope and curr_eps > threshold_min_eps):
            counter += 1
            #if the counter is 1, set the start eps
            if (counter == 1):
                start_eps = curr_eps
            #set the end_eps
            end_eps = curr_eps
            #find delta_eps
            delta_eps = end_eps - start_eps
            #if delta eps is big enough, there is a phase transition
            if (delta_eps > threshold_eps_diff):
                phase_trans = True
        #if the point failed, check if the point is past the already found␣
 ↪phase transition
        elif (phase_trans):
            #if so, the sound speed right after the phase transition is stored␣
 ↪for analyzing
            sound_speed_after = dpdeps[i]
            break
        #otherwise, reset the phase variables
        else:
            sound_speed_after = 0
            counter = 0
            start_eps = 0
            end_eps = 0
            delta_eps = 0
    #a final check if the start eps is greater than the central pressure
    if (start_eps > threshold_max_eps):
        phase_trans = False
    #return whether there is a phase transition, what the start, end, and delta␣
 ↪epsilon where, etc
    return phase_trans, start_eps, end_eps, delta_eps, max_sound_speed,␣
 ↪eps_at_max_speed, sound_speed_after
```

[8]:

```
#run the checker for every EOS and store the values
for i in range(5000):
    actual_EOSs[i][4], actual_EOSs[i][5], actual_EOSs[i][6], actual_EOSs[i][7],␣
 ↪actual_EOSs[i][8], actual_EOSs[i][9], actual_EOSs[i][10] =␣
 ↪check_phase_trans(i)
```

[9]:

```
#finds the number of EOSs with a phase transition and prints the percentage
num_of_phase_transitions = sum(EOS[4] for EOS in actual_EOSs)
print(num_of_phase_transitions/50, "% have phase transitions", "(",␣
 ↪num_of_phase_transitions, ")")
```

```
11.22 % have phase transitions ( 561 )
```

[10]:

```
##function for calculating the relative probability of no phase and phase
def calc_relative_prob(P_EOS_given_obs):
    #initialize the probabilities
    prob_phase = 0
    prob_no_phase = 0
    #for every EOS, check if it has a phase transition or not␣
 ↪(actual_EOSs[i][4] == 1, or 0)
    #if it does have a phase transition, add the probability of that EOS to␣
 ↪prob_phase, vice versa for no phase transition
```

```
    for i in range(5000):
        if (actual_EOSs[i][4] == 1):
            prob_phase += P_EOS_given_obs[i]
        else:
            prob_no_phase += P_EOS_given_obs[i]
    return prob_phase, prob_no_phase
```

[11]:

```
#initialize global prob_phase and prob_no_phase to store relative probabilities␣
 ↪at every step of contraint
prob_phase = np.zeros(6)
prob_no_phase = np.zeros(6)
#calculates the probabilities for each set of probabilities in P_EOS_given_obs
for i in range(6):
    prob_phase[i], prob_no_phase[i] = calc_relative_prob(P_EOS_given_obs[i])
```

[12]:

```
#factor to account for the fact that there is more representation of no phase␣
 ↪transitions
#89% of EOSs don't have a phase transition and 11% do
factor = (num_of_phase_transitions/5000)/(1-num_of_phase_transitions/5000)
#each line prints the probability of no phase, then the probability of phase,␣
 ↪and then the relative probability
#initially, they are equally likely (~1 to 1)
for i in range(6):
    print(prob_no_phase[i], prob_phase[i], "No phase transition is",␣
 ↪prob_no_phase[i]/prob_phase[i]*factor, "times more probable")
```

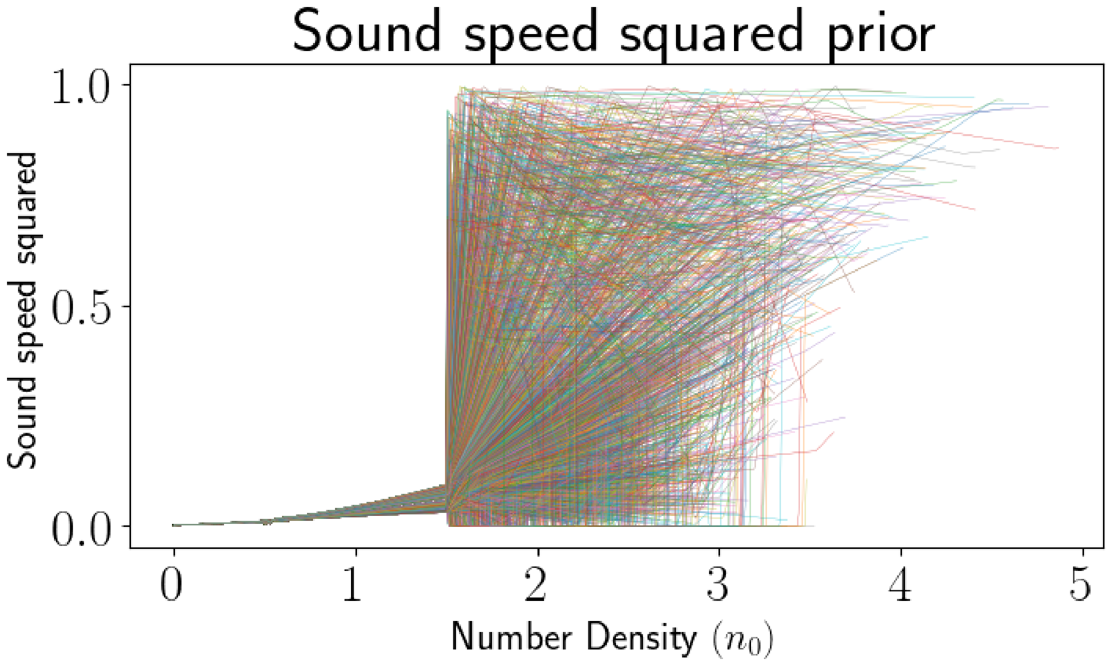
```
0.8877999999999349 0.11220000000000116 No phase transition is 0.9999999999999162
times more probable
0.9925649159034908 0.007435084096507787 No phase transition is 16.87138559441008
times more probable
0.9934169315215622 0.006583068478436553 No phase transition is 19.07132649564784
times more probable
0.9934471114840969 0.006552888515904785 No phase transition is
19.159743390846593 times more probable
0.9936204664907845 0.006379533509213799 No phase transition is
19.683817135832964 times more probable
0.9939560921435738 0.006043907856427324 No phase transition is
20.783901789915436 times more probable
```

[13]:

```
#we want to analyze the variables we stored but only the ones from EOSs with␣
 ↪phase transitions
only_phase_eps_start, only_phase_eps_diff, only_phase_max_sound_speed,␣
 ↪only_phase_sound_speed_after, only_phase_prob = [], [], [], [], []
for i in range(5000):
    if actual_EOSs[i][4]==1:
        only_phase_eps_start.append(actual_EOSs[i][5])
        only_phase_eps_diff.append(actual_EOSs[i][7])
        only_phase_max_sound_speed.append(actual_EOSs[i][8])
        only_phase_sound_speed_after.append(actual_EOSs[i][10])
        only_phase_prob.append(P_EOS_given_obs[0][i])
```

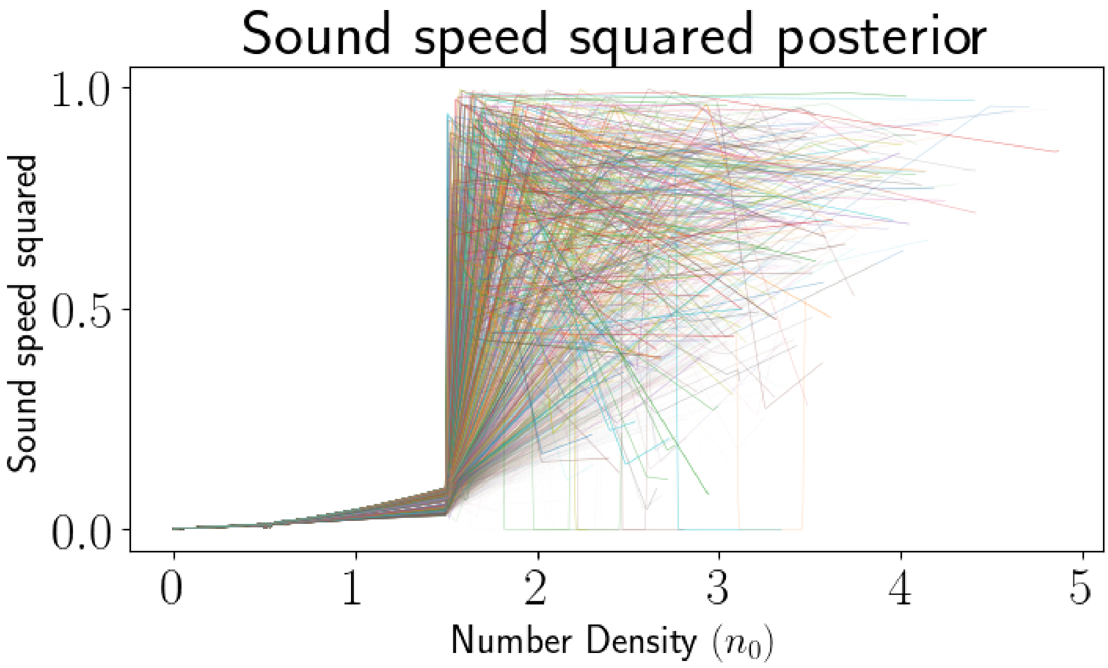
[14]:

```
#plots prior of the EOSs with a mask to not plot regions past the central␣
 ↪pressure
plt.xlabel(r"Number Density $(n_0)$", fontsize=18)
plt.ylabel("Sound speed squared", fontsize=18)
plt.title("Sound speed squared prior")
for i in range(5000):
    mask = actual_EOSs[i][1] < central_pressures[i]
    plt.plot(actual_EOSs[i][0][mask], actual_EOSs[i][3][mask[:-1]], linewidth =␣
 ↪0.2)
#plt.savefig("/home/tplohr/proj/SF 24-25/figs/sound_speed/
 ↪sound_speed_squared_prior")
plt.show()
```



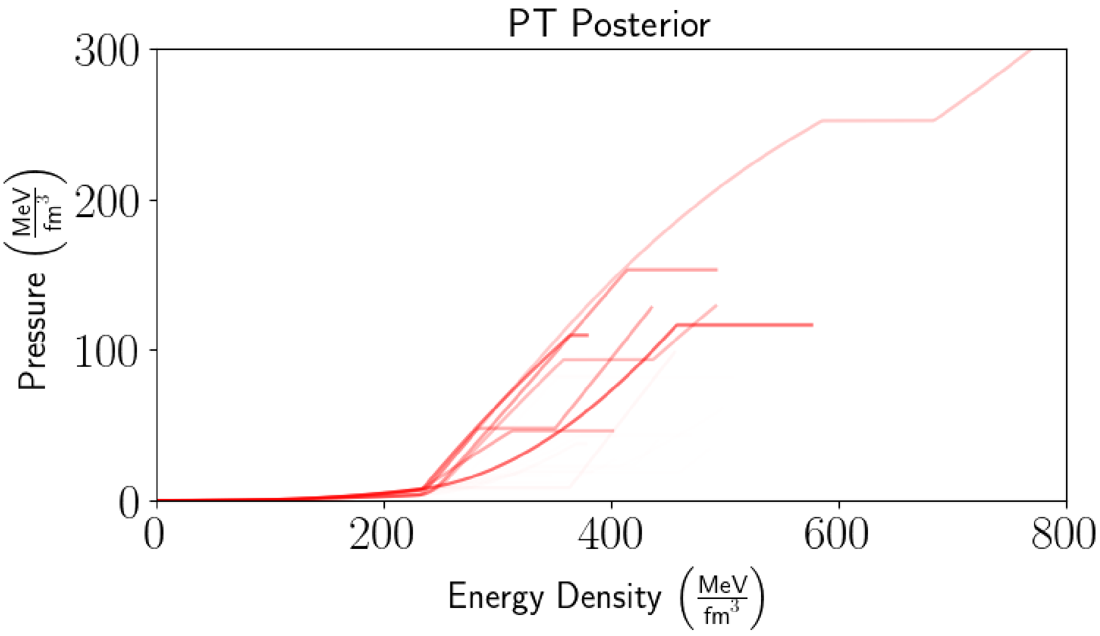
[15]:

```
#plots the posterior in the same manor as the prior
norm = plt.Normalize(np.min(P_EOS_given_obs), np.max(P_EOS_given_obs))
plt.xlabel(r"Number Density $(n_0)$", fontsize=18)
plt.ylabel("Sound speed squared", fontsize=18)
plt.title("Sound speed squared posterior")
for i in range(5000):
    alpha = norm(P_EOS_given_obs[-1][i])
    mask = actual_EOSs[i][1] < central_pressures[i]
    plt.plot(actual_EOSs[i][0][mask], actual_EOSs[i][3][mask[:-1]],␣
 ↪alpha=alpha, linewidth = 0.4)
#plt.savefig("/home/tplohr/proj/SF 24-25/figs/sound_speed/
 ↪sound_speed_posterior")
plt.show()
```

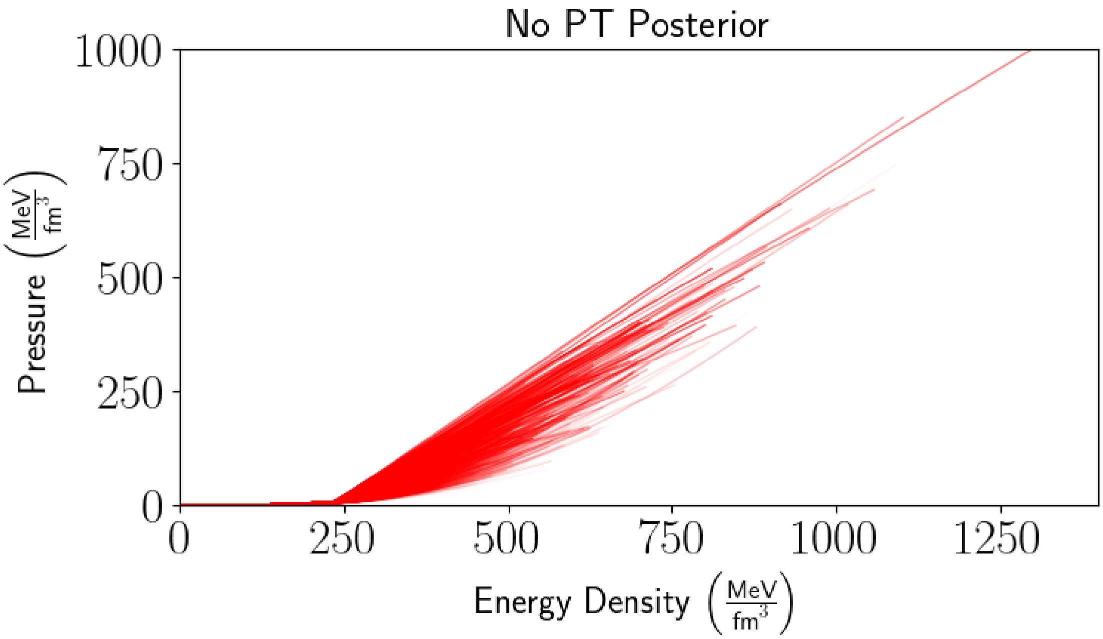


[16]:

```
#plots only the EOSs with a phase transition in erg density and pressure
plt.xlabel(r"Energy Density $\left(\frac{\text{MeV}}{\text{fm}^{3}}\right)$",␣
 ↪fontsize=18)
plt.xlim(0, 800)
plt.ylabel(r"Pressure $\left(\frac{\text{MeV}}{\text{fm}^{3}}\right)$",␣
 ↪fontsize=18)
plt.ylim(0, 300)
plt.title("PT Posterior", fontsize=20)
for i in range(5000):
    if actual_EOSs[i][4]==1:
        alpha = norm(P_EOS_given_obs[-1][i])
        mask = actual_EOSs[i][1] <= central_pressures[i]
        plt.plot(actual_EOSs[i][1][mask], actual_EOSs[i][2][mask], alpha=alpha,␣
 ↪linewidth = 1.6, color="red")
#plt.savefig("/home/tplohr/proj/SF 24-25/figs/phase_and_no_phase/
 ↪PT_posterior_p_erg", bbox_inches="tight")
plt.show()
```

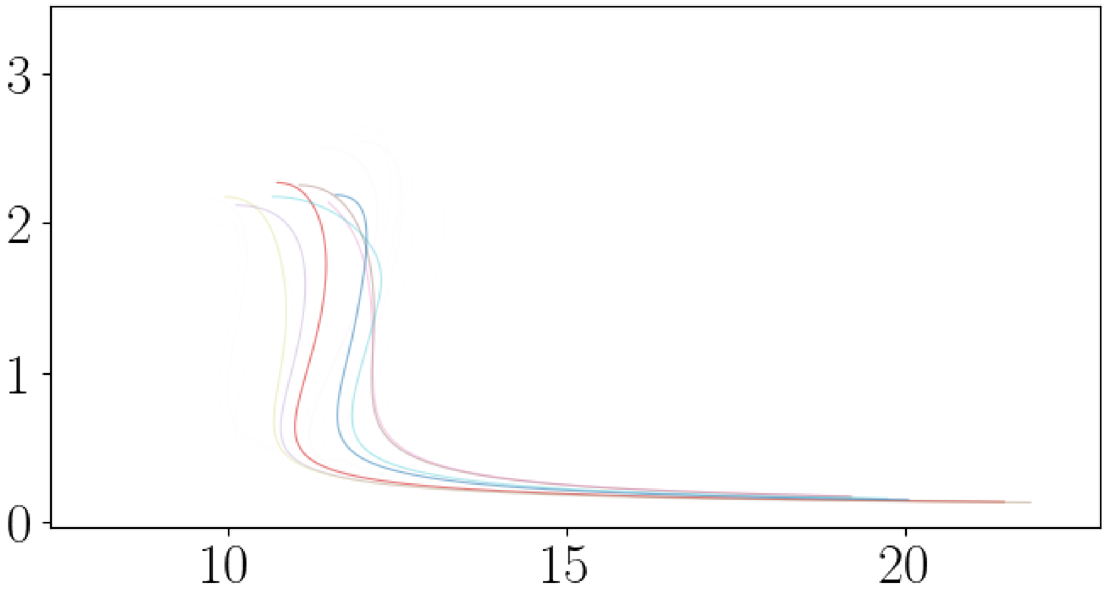


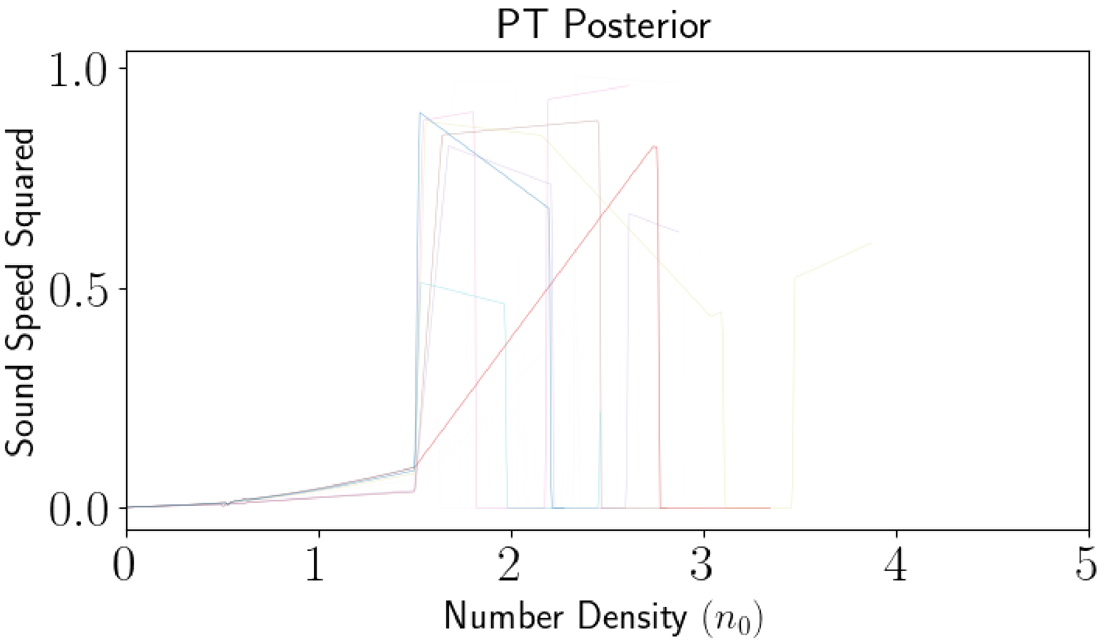
[17]:
```
#plots EOSs without a phase transition in erg density and pressure
plt.xlabel(r"Energy Density $\left(\frac{\text{MeV}}{\text{fm}^{3}}\right)$",␣
 ↪fontsize=18)
plt.xlim(0, 1400)
plt.ylabel(r"Pressure $\left(\frac{\text{MeV}}{\text{fm}^{3}}\right)$",␣
 ↪fontsize=18)
plt.ylim(0, 1000)
plt.title("No PT Posterior", fontsize=20)
for i in range(5000):
    if actual_EOSs[i][4]==0:
        alpha = norm(P_EOS_given_obs[-1][i])
        mask = actual_EOSs[i][1] <= central_pressures[i]
        plt.plot(actual_EOSs[i][1][mask], actual_EOSs[i][2][mask], alpha=alpha,␣
 ↪linewidth = 0.8, color="red")
plt.savefig("/home/tplohr/proj/SF 24-25/figs/phase_and_no_phase/
 ↪No_PT_posterior_p_erg", bbox_inches="tight")
plt.show()
```



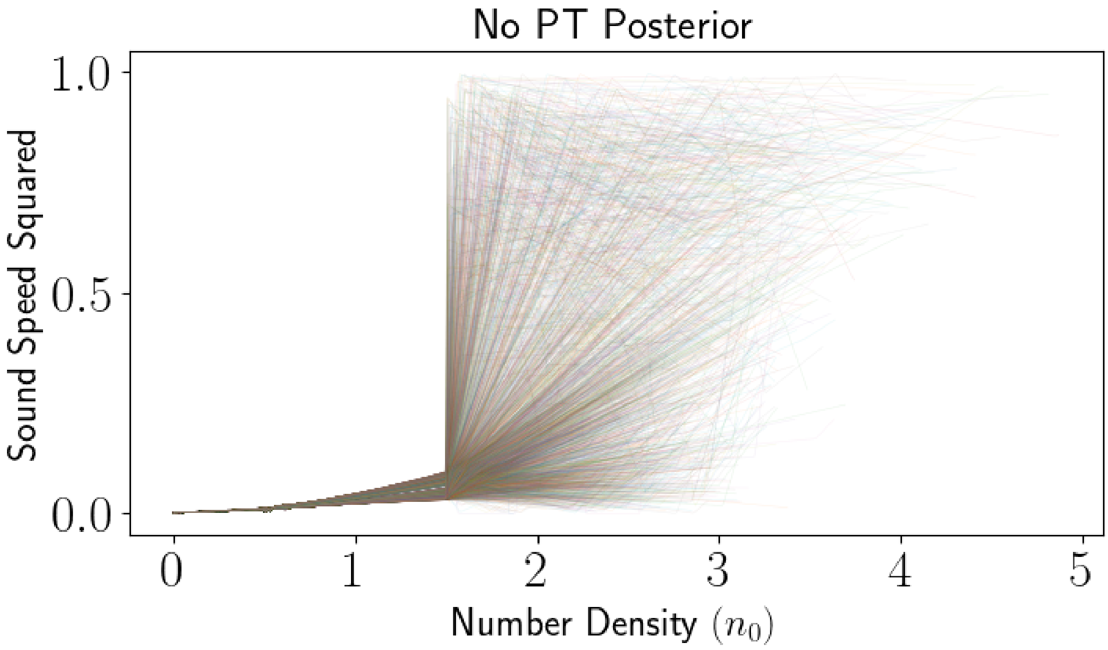
[18]:

```
##plots EOSs with a phase transition in mass vs radius and sound speed squared␣
 ↪vs number density
for i in range(5000):
    if actual_EOSs[i][4]==1:
        alpha = norm(P_EOS_given_obs[-1][i])
        mask = actual_EOSs[i][1] < central_pressures[i]
        plt.plot(EOSs[i][0], EOSs[i][1], alpha=alpha, linewidth = 0.8)
plt.show()
plt.xlabel(r"Number Density $(n_0)$", fontsize=18)
plt.ylabel("Sound Speed Squared", fontsize=18)
plt.title("PT Posterior", fontsize=20)
plt.xlim(0, 5)
for i in range(5000):
    if actual_EOSs[i][4]==1:
        alpha = norm(P_EOS_given_obs[-1][i])
        mask = actual_EOSs[i][1] < central_pressures[i]
        plt.plot(actual_EOSs[i][0][mask], actual_EOSs[i][3][mask[:-1]],␣
 ↪alpha=alpha, linewidth = 0.4)
plt.savefig("/home/tplohr/proj/SF 24-25/figs/sound_speed/
 ↪sound_speed_phase_posterior")
```





[19]:
```python
#plots no phase transition in sound speed squared and number density
plt.xlabel(r"Number Density $(n_0)$", fontsize=18)
plt.ylabel("Sound Speed Squared", fontsize=18)
plt.title("No PT Posterior", fontsize=20)
for i in range(5000):
    if actual_EOSs[i][4]==0:
        alpha = norm(P_EOS_given_obs[0][i])
        mask = actual_EOSs[i][1] < central_pressures[i]
        plt.plot(actual_EOSs[i][0][mask], actual_EOSs[i][3][mask[:-1]],␣
 ↪alpha=alpha, linewidth = 0.4)
plt.savefig("/home/tplohr/proj/SF 24-25/figs/sound_speed/
 ↪sound_speed_no_phase_posterior")
plt.show()
```

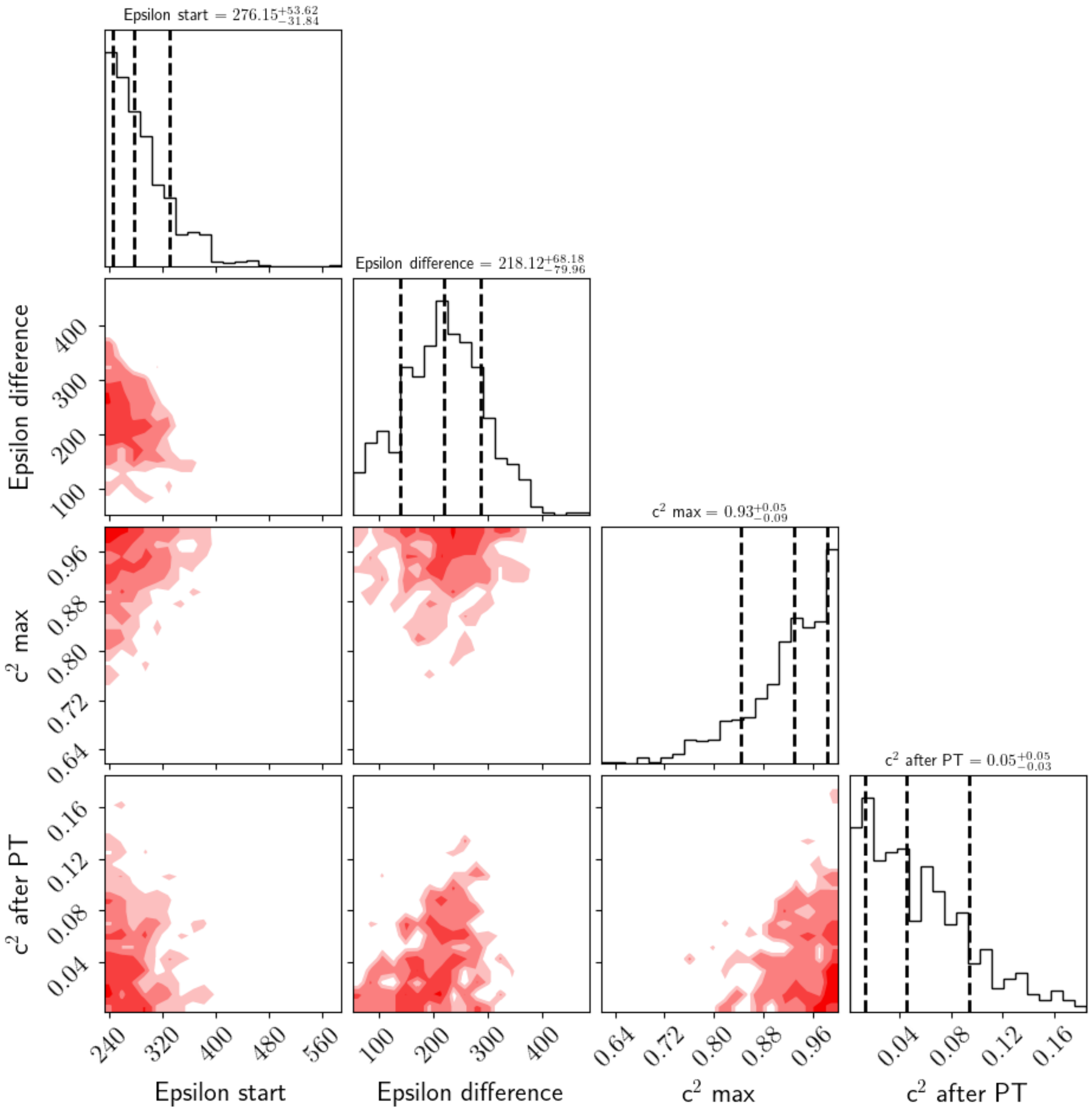


[20]:
```
#number of variables and samples for a correlation plot
ndim, nsamples = 4, num_of_phase_transitions
#stacks the variables into data
data = np.stack([only_phase_eps_start, only_phase_eps_diff,␣
 ↪only_phase_max_sound_speed, only_phase_sound_speed_after], axis=1)
```

[84]:
```
##plotting correlation plot of stored variables

#red colormap
red_cmap = mp.colors.LinearSegmentedColormap.from_list(
    "pure_red", [(0, "#FFFFFF"), (1, "#f50000")]
)
#color levels from the cmap
colors = [red_cmap(0), red_cmap(0.25), red_cmap(0.50), red_cmap(0.75),␣
 ↪red_cmap(0.99)]
```

```
#plots the figure
figure = corner.corner(
    data,
    labels=["Epsilon start", "Epsilon difference", r"$\text{c}^{2}$ max", r"$\text{c}^{2}$ after PT"],
    plot_contours=True,
    fill_contours=True,
    quantiles=[0.16, 0.50, 0.84],
    show_titles=True,
    weights=only_phase_prob,
    title_kwargs={"fontsize": 10},
    label_kwargs={'fontsize': 16},
    tick_label_fontsize=8,
    contour_kwargs={"colors": colors},
    contourf_kwargs={"colors": colors},
)
for ax in figure.axes:
    #change the font size of tick labels
    for tick in ax.get_xticklabels() + ax.get_yticklabels():
        tick.set_fontsize(16)
plt.savefig("/home/tplohr/proj/SF 24-25/figs/pair_plot")
```



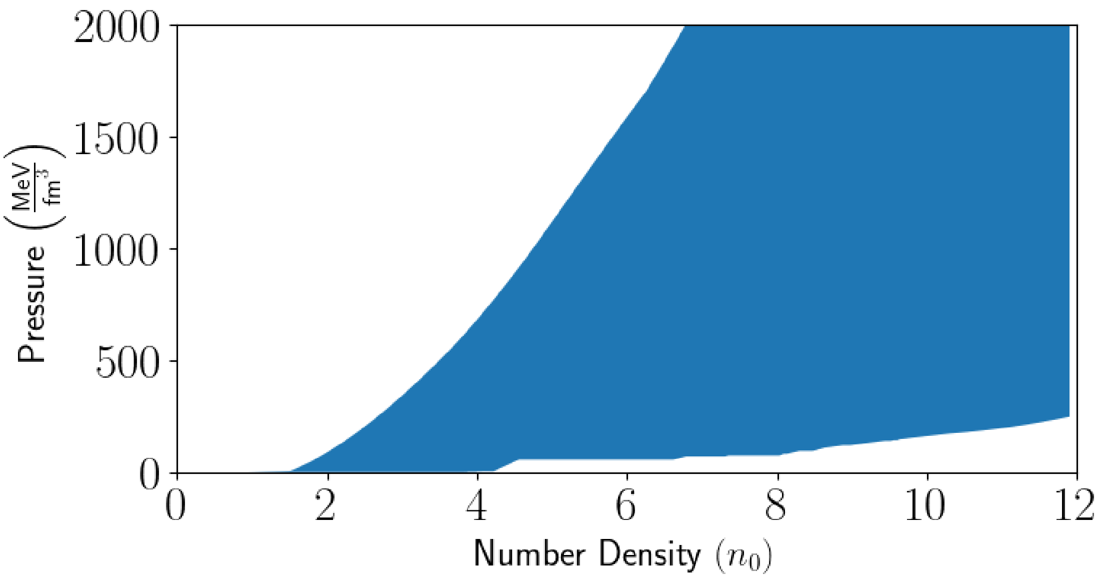
[22]:
```
##bands for the EOSs in pressure vs. number density
max_n = 11.9
min_n = 0.01
#generates points between min n and max n
n_points = np.linspace(min_n, max_n, 1000)
num_n_points = len(n_points)
#stores the lower and upper bounds prior and posterior (weighted)
lower_bounds = np.zeros(num_n_points)
upper_bounds = np.zeros(num_n_points)
lower_bounds_weighted = np.zeros(num_n_points)
upper_bounds_weighted = np.zeros(num_n_points)
#makes pressure(numberdensity) for each EOS
```

```
p_funcs = [None] * 5000
for i in range(5000):
    p_funcs[i] = sp.interpolate.interp1d(actual_EOSs[i][0], actual_EOSs[i][2])
```

[23]:
```
#input points, whether using number or energy density, the pressure functions,␣
 ↪percentiles, and weights
def pressure_percentiles(n, d, p_funcs, lower_percentile, upper_percentile,␣
 ↪weights=None):
    #initialize
    pressures = []
    usable_weights = []
    #for every EOS, if the point isn't past the EOS, append the pressure and␣
 ↪weight of that pressure value
    if weights is not None:
        for i in range(5000):
            if n<np.max(actual_EOSs[i][d]):
                pressures.append(float(p_funcs[i](n)))
                usable_weights.append(weights[i])
    else:
        for i in range(5000):
            if n<np.max(actual_EOSs[i][d]):
                pressures.append(float(p_funcs[i](n)))
                usable_weights.append(1/5000)
    #find the lower and upper bound
    lower_bound = np.percentile(pressures, lower_percentile, axis=0,␣
 ↪weights=usable_weights, method='inverted_cdf')
    upper_bound = np.percentile(pressures, upper_percentile, axis=0,␣
 ↪weights=usable_weights, method='inverted_cdf')
    return lower_bound, upper_bound
```

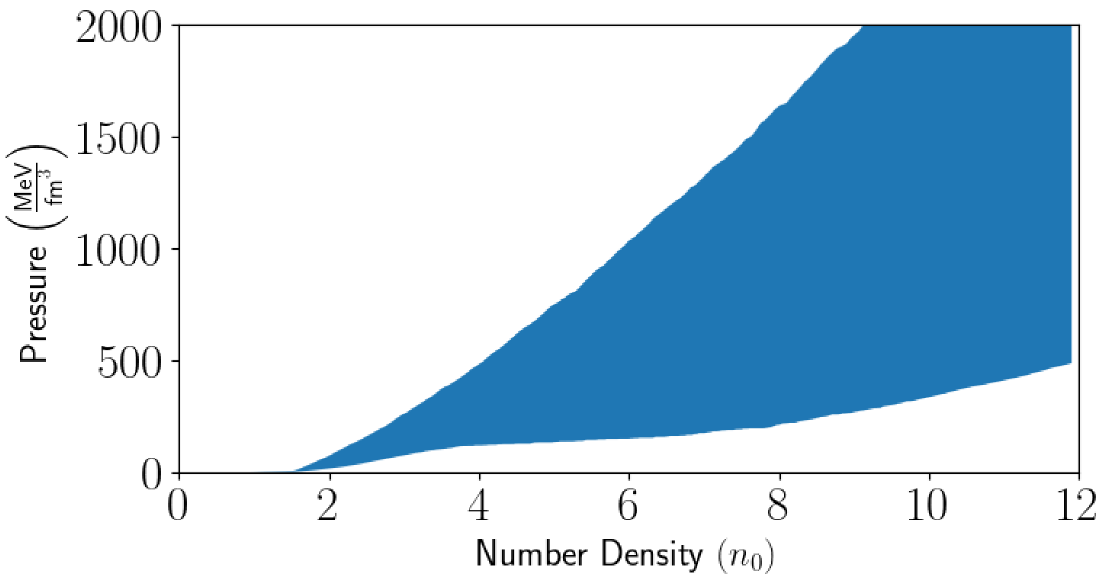
[24]:
```
#calculate the lower and upper bounds for the 0th and 100th percentile to show␣
 ↪the intial range of EOSs
for i in range(num_n_points):
    lower_bounds[i], upper_bounds[i] = pressure_percentiles(n_points[i], 0,␣
 ↪p_funcs, 0, 100)
```

[25]:
```
#plot the intial range
plt.rcParams['figure.figsize'] = [8,4]
plt.xlim([0,12])
plt.ylim([0,2000])
plt.xlabel(r"Number Density $(n_0)$", fontsize=18)
plt.ylabel(r"Pressure $\left(\frac{\text{MeV}}{\text{fm}^{3}}\right)$",␣
 ↪fontsize=18)
plt.fill_between(n_points, lower_bounds, upper_bounds)
plt.show()
```



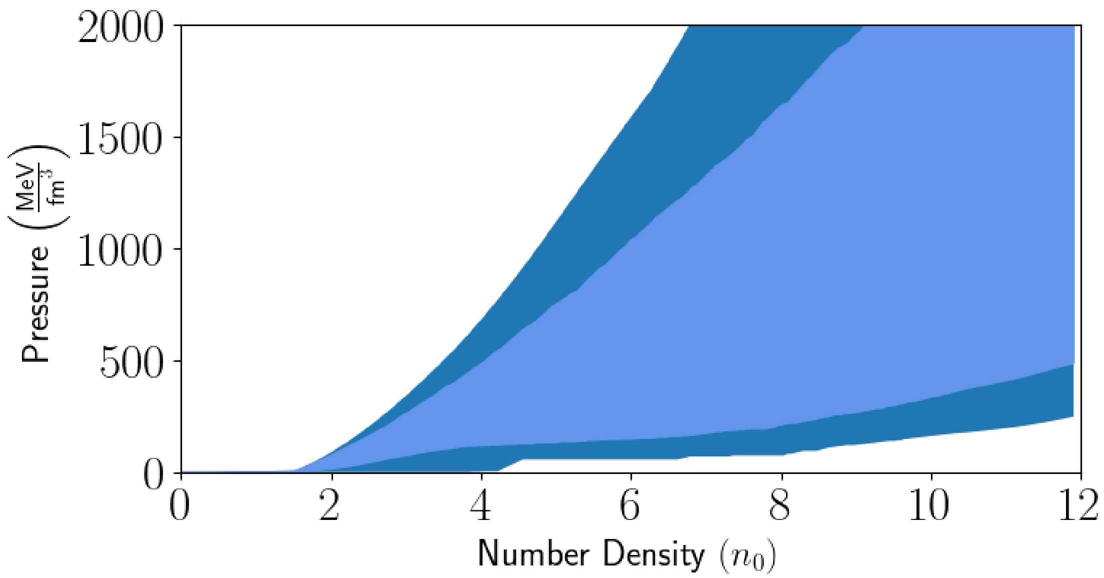
```
[26]: #find the lower and upper bounds of the pressure after the constrains, the␣
       ↪weights are the probabilities from the last set
      for i in range(num_n_points):
          lower_bounds_weighted[i], upper_bounds_weighted[i] =␣
       ↪pressure_percentiles(n_points[i], 0, p_funcs, 5, 95, P_EOS_given_obs[-1])
```

```
[27]: #plot the constrained bands
      plt.rcParams['figure.figsize'] = [8,4]
      plt.xlim([0,12])
      plt.ylim([0,2000])
      plt.xlabel(r"Number Density $(n_0)$", fontsize=18)
      plt.ylabel(r"Pressure $\left(\frac{\text{MeV}}{\text{fm}^{3}}\right)$",␣
       ↪fontsize=18)
      plt.fill_between(n_points, lower_bounds_weighted, upper_bounds_weighted)
      plt.show()
```



[28]:

```
#overlay the bands
plt.rcParams['figure.figsize'] = [8,4]
plt.xlim([0,12])
plt.ylim([0,2000])
plt.xlabel(r"Number Density $(n_0)$", fontsize=18)
plt.ylabel(r"Pressure $\left(\frac{\text{MeV}}{\text{fm}^{3}}\right)$",␣
 ↪fontsize=18)
plt.fill_between(n_points, lower_bounds, upper_bounds)
plt.fill_between(n_points, lower_bounds_weighted, upper_bounds_weighted,␣
 ↪color="cornflowerblue")
#plt.savefig("/home/tplohr/proj/SF 24-25/figs/bands/p_num_bands",␣
 ↪bbox_inches="tight")
plt.show()
```

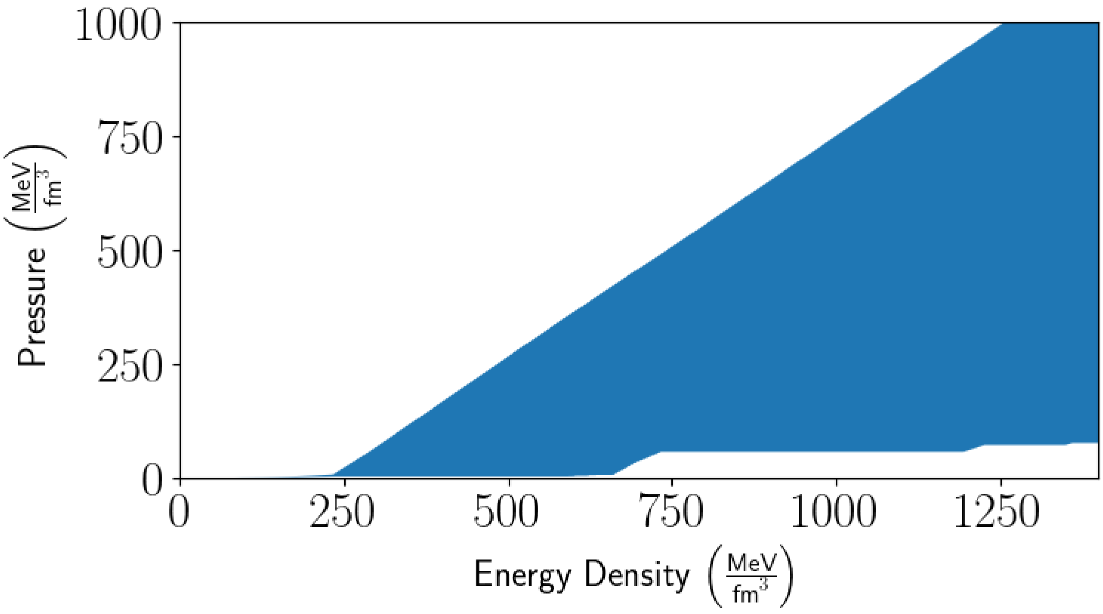


[29]:
```
#bands for the EOSs in pressure v. energy density
#instead of number density, we have points in energy density
max_n2 = 1400
min_n2 = 0.01
n_points2 = np.linspace(min_n2, max_n2, 1000)
num_n_points2 = len(n_points2)
lower_bounds2 = np.zeros(num_n_points)
upper_bounds2 = np.zeros(num_n_points)
lower_bounds_weighted2 = np.zeros(num_n_points)
upper_bounds_weighted2 = np.zeros(num_n_points)
#the pressure functions are pressure(erg_density)
p_funcs2 = [None] * 5000
for i in range(5000):
    p_funcs2[i] = sp.interpolate.interp1d(actual_EOSs[i][1], actual_EOSs[i][2])
```

[30]:
```
#inital bands with energy density as the x axis
for i in range(num_n_points2):
    lower_bounds2[i], upper_bounds2[i] = pressure_percentiles(n_points2[i], 1,␣
 ↪p_funcs2, 0, 100)
```

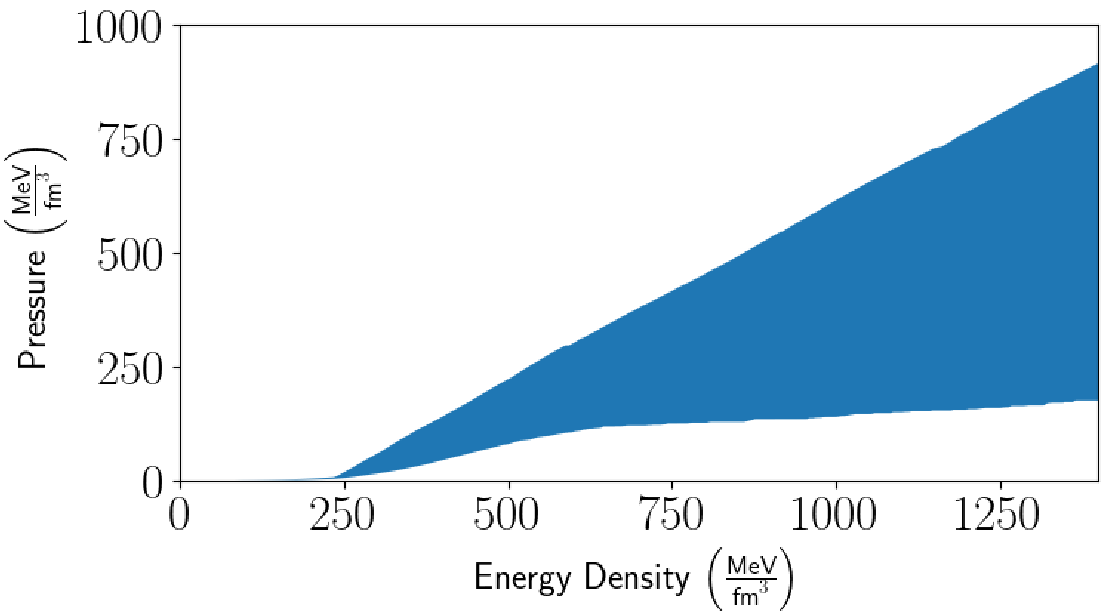
[31]:
```
#plot the initial bands
plt.rcParams['figure.figsize'] = [8,4]
plt.xlim(0,1400)
plt.ylim(0,1000)
plt.xlabel(r"Energy Density $\left(\frac{\text{MeV}}{\text{fm}^{3}}\right)$",␣
 ↪fontsize=18)
```

```
plt.ylabel(r"Pressure $\left(\frac{\text{MeV}}{\text{fm}^{3}}\right)$",␣
 ↪fontsize=18)
plt.fill_between(n_points2, lower_bounds2, upper_bounds2)
plt.show()
```



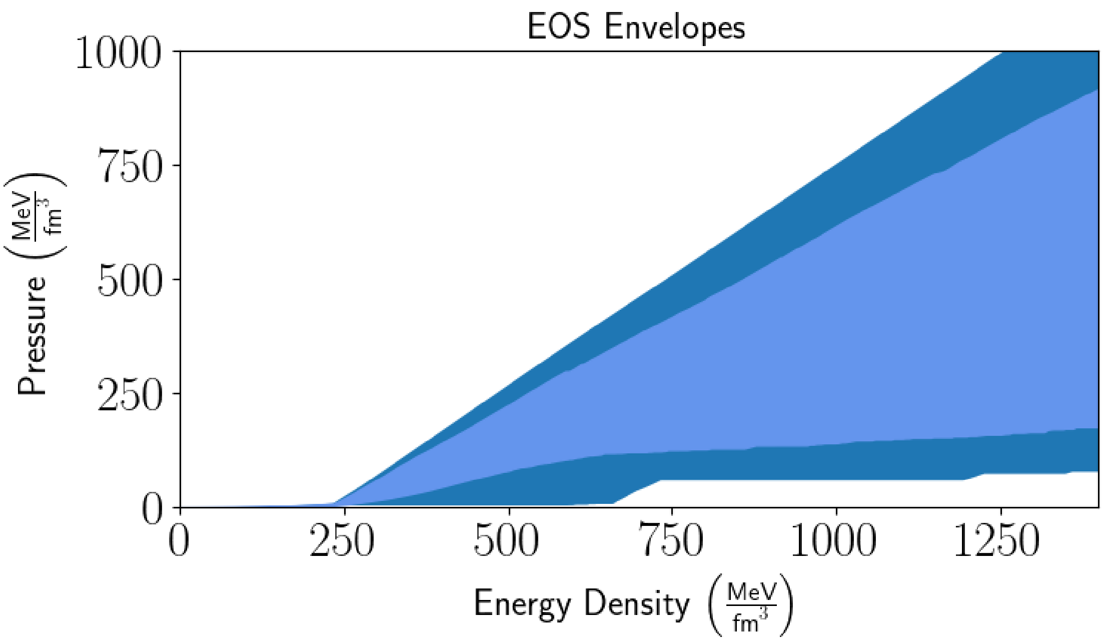
[32]:
```
#constrained bands in erg density
for i in range(num_n_points2):
    lower_bounds_weighted2[i], upper_bounds_weighted2[i] =␣
 ↪pressure_percentiles(n_points2[i], 1, p_funcs2, 5, 95, P_EOS_given_obs[-1])
```

[33]:
```
#plot constrained bands in erg density
plt.rcParams['figure.figsize'] = [8,4]
plt.xlim(0,1400)
plt.ylim(0,1000)
plt.xlabel(r"Energy Density $\left(\frac{\text{MeV}}{\text{fm}^{3}}\right)$",␣
 ↪fontsize=18)
plt.ylabel(r"Pressure $\left(\frac{\text{MeV}}{\text{fm}^{3}}\right)$",␣
 ↪fontsize=18)
plt.fill_between(n_points2, lower_bounds_weighted2, upper_bounds_weighted2)
plt.show()
```



[34]:

```
#overlay the plots
plt.title("EOS Envelopes", fontsize=18)
plt.rcParams['figure.figsize'] = [8,4]
plt.xlim([0,1400])
plt.ylim([0,1000])
plt.xlabel(r"Energy Density $\left(\frac{\text{MeV}}{\text{fm}^{3}}\right)$",␣
 ↪fontsize=18)
plt.ylabel(r"Pressure $\left(\frac{\text{MeV}}{\text{fm}^{3}}\right)$",␣
 ↪fontsize=18)
plt.fill_between(n_points2, lower_bounds2, upper_bounds2)
plt.fill_between(n_points2, lower_bounds_weighted2, upper_bounds_weighted2,␣
 ↪color="cornflowerblue")
plt.savefig("/home/tplohr/proj/SF 24-25/figs/bands/p_erg_bands",␣
 ↪bbox_inches="tight")
plt.show()
```

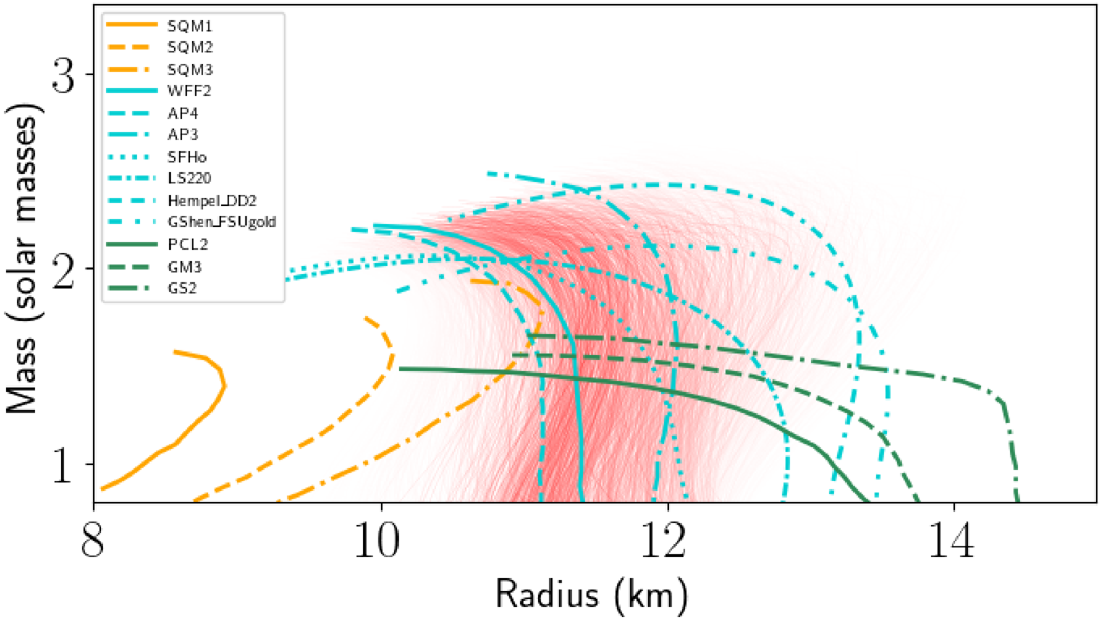


[120]:

```
files = []
files.append("SQM1.txt")
files.append("SQM2.txt")
files.append("SQM3.txt")
#files.append("WFF1.txt")
files.append("WFF2.txt")
files.append("AP4.txt")
files.append("AP3.txt")
files.append("SFHo_MR")
files.append("LS220_MR")
files.append("Hempel_DD2_MR")
files.append("GShen_FSUgold_MR")
files.append("PCL2.txt")
files.append("GM3.txt")
#files.append("MS1.txt")
files.append("GS2.txt")
#files.append("MS0.txt")
other_EOSs = [None] * len(files)
for i in range(6):
    path = open("/home/tplohr/proj/SF 24-25/data/other_EOSs/" + files[i], 'r')
    df = pd.read_csv(path, names = ["radius", "mass"], delimiter=", ",␣
 ↪engine="python")
    other_EOSs[i] = np.array([df["radius"], df["mass"]])
for i in range(6, 10):
    path = open("/home/tplohr/proj/SF 24-25/data/other_EOSs/" + files[i], 'r')
    df = pd.read_csv(path, names = ["radius", "mass"], sep='\s+')
    other_EOSs[i] = np.array([df["radius"], df["mass"]])
for i in range(10, len(files)):
    path = open("/home/tplohr/proj/SF 24-25/data/other_EOSs/" + files[i], 'r')
    df = pd.read_csv(path, names = ["radius", "mass"], delimiter=", ",␣
  ↪engine="python")
    other_EOSs[i] = np.array([df["radius"], df["mass"]])
```

[145]:

```
#plotting each EOS; low probability is white, high probability is red
cmap = mp.colors.LinearSegmentedColormap.from_list("white_to_red",␣
  ↪["white","red"])
norm = plt.Normalize(P_EOS_given_obs[-1].min(), P_EOS_given_obs[-1].max())
#labels
labels = ["SQM1", "SQM2", "SQM3", "WFF2", "AP4", "AP3", "SFHo", "LS220",␣
  ↪"Hempel_DD2", "GShen_FSUgold", "PCL2", "GM3", "GS2"]
#colors
colors = ["orange", "orange", "orange", "darkturquoise", "darkturquoise",␣
  ↪"darkturquoise", "darkturquoise", "darkturquoise", "darkturquoise",␣
  ↪"darkturquoise", "seagreen", "seagreen", "seagreen"]
#linetypes
linestyles = ['-', '--', '-.', '-', '--', '-.', ':', (0, (3, 1, 1, 1)), (0, (3,␣
  ↪2, 3, 2, 1, 2)), (0, (3, 3, 1, 3, 1, 3)), '-', '--', '-.']
#plots each EOS
plt.rcParams['figure.figsize'] = [8,4]
plt.xlim([8,15])
plt.ylim([.8,3.35])
plt.xlabel("Radius (km)", fontsize=18)
plt.ylabel("Mass (solar masses)", fontsize=18)
for i in range(numofEOS):
    color = cmap(norm(P_EOS_given_obs[-1][i]))
    plt.plot(EOSs[i][0], EOSs[i][1], color=color, linewidth=.1)
for i in range(len(files)):
    plt.plot(other_EOSs[i][0], other_EOSs[i][1], label=labels[i],␣
  ↪color=colors[i], linestyle=linestyles[i])
plt.legend(loc='upper left', fontsize=7)
plt.savefig("/home/tplohr/proj/SF 24-25/figs/other_EOSs", bbox_inches="tight")
plt.show()
```


SQM1
SQM2
SQM3
WFF2
AP4
AP3
SFHo
LS220
Hempel_DD2
GShen_FSUgold
PCL2
GM3
GS2
Mass (solar masses)
1
2
3
8
10
12
14
Radius (km)

[ ]: