Constraining_EOSs

April 2, 2025

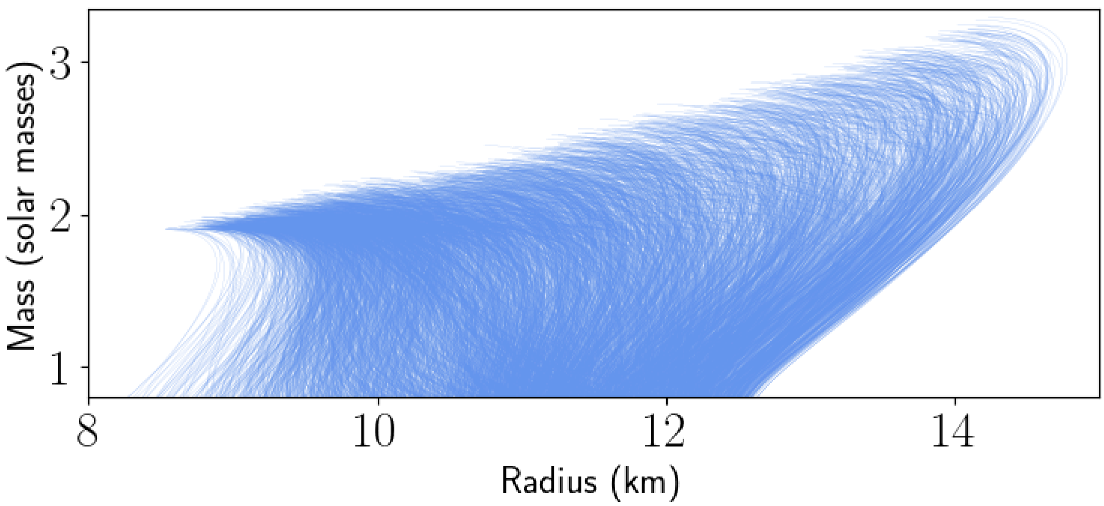
[1]:
```
import matplotlib as mp
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import math
import scipy as sp
import time
import os
```

[2]:
```
#opens .dat files to be converted into CSVs
#change pathprefix on different computers
EOS_files = "/home/tplohr/proj/SF 24-25/diettim NMMA master EOS-chiralEFT_MTOV/"
NICER_1_data = "/home/tplohr/proj/SF 24-25/data/J0030_Amsterdam_2019.dat"
NICER_2_data = "/home/tplohr/proj/SF 24-25/data/J0437.dat"
NICER_3_data = "/home/tplohr/proj/SF 24-25/data/J0740.dat"
LIGO_data = "/home/tplohr/proj/SF 24-25/data/GW170817.dat"

def loadfile(number):
    file = open(EOS_files + str(number) + '.dat', 'r')
    return file
```

[3]:
```
#creates an empty array to contain all EOSs
numofEOS = 5000
EOSs = [None] * numofEOS

#fills "EOSs" with arrays of each EOS's mass and radius values
for i in range(numofEOS):
    df = pd.read_csv(loadfile(i+1), delimiter = '\t', names = ["radius",␣
 ↪"mass", "deformability"])
    EOSs[i] = np.array([df["radius"], df["mass"], df["deformability"]])
#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][2] is the list of deformability 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
#EOSs[i][2][j] is the deformability value indexed j of EOS_i
```

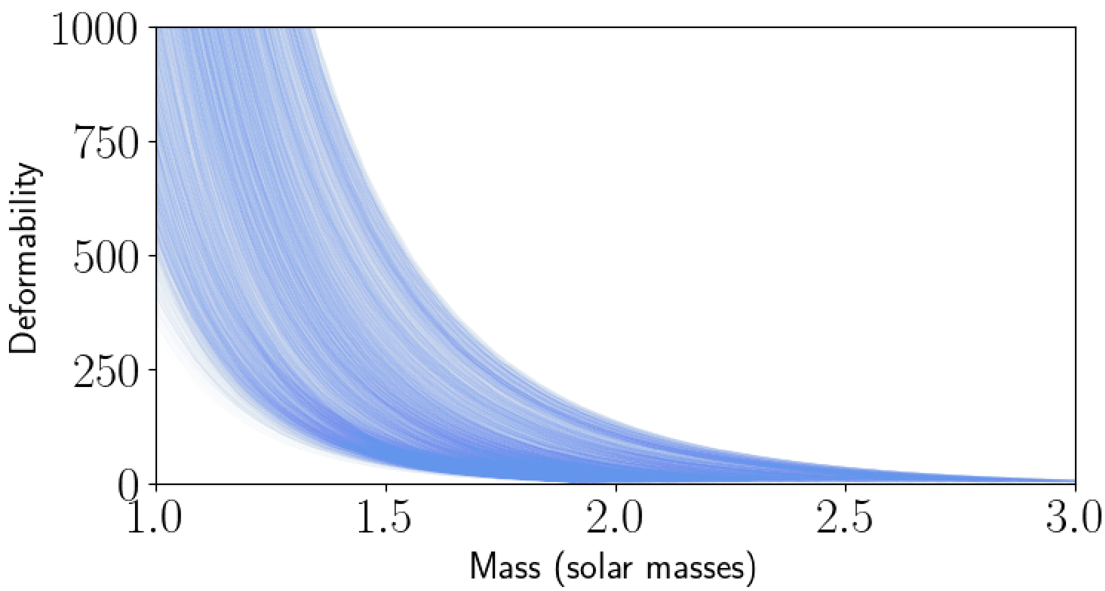
[4]:

```
#plots each EOS in mass and radius
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):
    plt.plot(EOSs[i][0], EOSs[i][1], 'cornflowerblue', linewidth=.1)
plt.tight_layout()
#plt.savefig("/home/tplohr/proj/SF 24-25/figs/m_r_EOS_plot.png")
```



[5]:

```
#plots each EOS
plt.rcParams['figure.figsize'] = [8,4]
plt.xlim([1,3])
plt.ylim([0,1000])
plt.xlabel("Mass (solar masses)", fontsize=18)
plt.ylabel("Deformability", fontsize=18)
for i in range(numofEOS):
    plt.plot(EOSs[i][1], EOSs[i][2], 'cornflowerblue', linewidth=.01)
```



[6]:
```
#initialization of probability arrays

#probabilities of the EOSs given the observations, each index corresponds to␣
↪one EOS; posterior
P_EOS_given_obs = np.zeros(numofEOS)
#probabilities of the observations given an EOS; likelihood
P_obs_given_EOS = np.zeros(numofEOS)
#probabilities of the EOSs without a condition; prior
P_EOS = np.ones(numofEOS)/numofEOS
#list of radius values each EOS predicts at M=1.4
r_at_given_mass = np.zeros(numofEOS)
```

[7]:
```
#for the max mass data, we have averages and deviations of each dataset␣
↪representing one neutron star
mu_1 = 2.14
mu_2 = 2.01
mu_3 = 1.908
mu_4 = 2.16
sigma_1 = 0.1
sigma_2 = 0.04
sigma_3 = 0.016
sigma_4 = 0.17
#the PDF is a set of cumulative distribution functions (CDFs) multiplied
#to find the probability of a specific maximum mass, we put it into this␣
↪function
def pdf_1_evaluate(max_m):
```

```
    return sp.stats.norm.cdf(max_m, mu_1, sigma_1) * sp.stats.norm.cdf(max_m,␣
↪mu_2, sigma_2) * sp.stats.norm.cdf(max_m, mu_3, sigma_3) * (1-sp.stats.norm.
↪cdf(max_m, mu_4, sigma_4))
```

[8]:
```
#reading in the data from NICER dataset 1
dataframe1 = pd.read_csv(NICER_1_data, delim_whitespace=True, header=None)
selected_columns1 = dataframe1.iloc[:,[1,2]]
data_NICER_1 = selected_columns1.to_numpy()[:10000]
print(np.shape(data_NICER_1))
#using a KDE to get the probability density function for the likelihood
pdf_2 = sp.stats.gaussian_kde((data_NICER_1[:,1], data_NICER_1[:,0]))
```

```
(10000, 2)
```

[9]:
```
#reading in the data from LIGO
dataframe2 = pd.read_csv(LIGO_data, delim_whitespace=True, header=0)
selected_columns2 = dataframe2.iloc[:,[2,3,4,5]]
#data[:,0] is m1, data[:,1] is m2, data[:,2] is lambda1, data[:,3] is lambda2
raw_data = selected_columns2.to_numpy()
#making data have the same shape
data_LIGO = np.copy(raw_data)
#divide mass by 1.0099 since the raw data is in our reference frame but we need␣
↪the neutron star's reference frame mass
data_LIGO[:, [0,1]] = raw_data[:, [0,1]]/1.0099
data_LIGO[:, [2,3]] = raw_data[:, [2,3]]
print(np.shape(data_LIGO))
#using a KDE to get the probability density function for the likelihood
pdf_3 = sp.stats.gaussian_kde((data_LIGO[:,0], data_LIGO[:,1], data_LIGO[:,2],␣
↪data_LIGO[:,3]))
```

```
(3952, 4)
```

[10]:
```
#reading in the data from NICER dataset 2
dataframe3 = pd.read_csv(NICER_2_data, delim_whitespace=True, header=None)
selected_columns3 = dataframe3.iloc[:,[0,1]]
data_NICER_2 = selected_columns3.to_numpy()[:10000]
print(np.shape(data_NICER_2))
#using a KDE to get the probability density function for the likelihood
pdf_4 = sp.stats.gaussian_kde((data_NICER_2[:,1], data_NICER_2[:,0]))
```

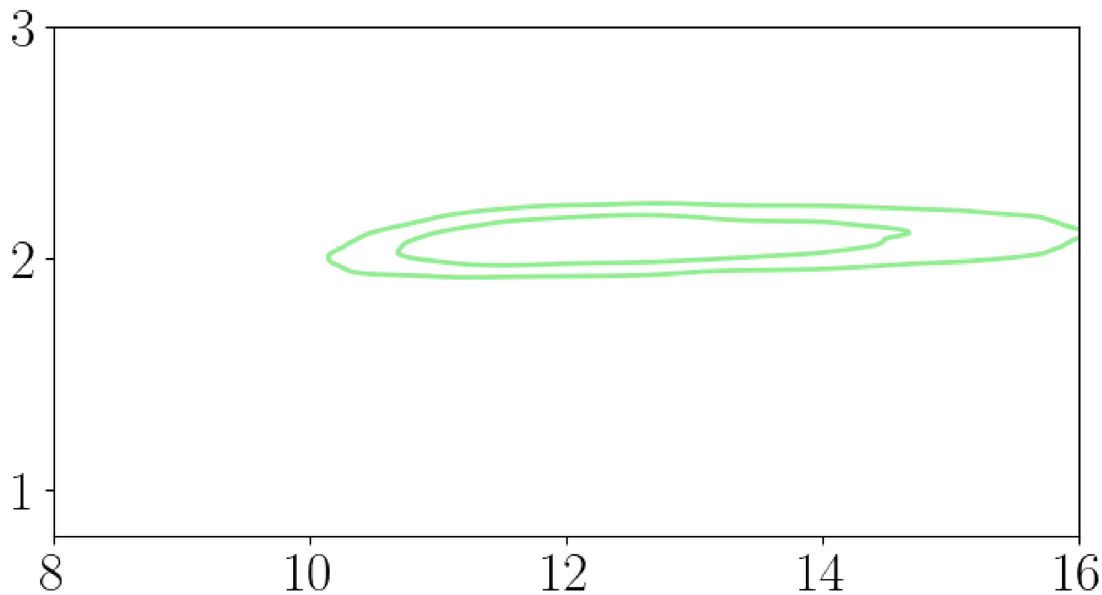
```
(10000, 2)
```

[11]:
```
#reading in the data from NICER dataset 3
dataframe4 = pd.read_csv(NICER_3_data, delim_whitespace=True, header=None)
selected_columns4 = dataframe4.iloc[:,[0,1]]
data_NICER_3 = selected_columns4.to_numpy()[:10000]
print(np.shape(data_NICER_3))
#using a KDE to get the probability density function for the likelihood
pdf_5 = sp.stats.gaussian_kde((data_NICER_3[:,1], data_NICER_3[:,0]))
```

(10000, 2)

[12]:
```python
##contours of the NICER 3 dataset

#create the X and Y grids
X = np.linspace(8, 16, 250)
Y = np.linspace(0.8, 3.0, 250)
#create meshgrid for X and Y
X_grid, Y_grid = np.meshgrid(X, Y)
#array with 250 * 250 points
points = np.zeros((250 * 250, 3))
#probability array to store probability values
prob = np.zeros((250, 250))
#loop to evaluate and populate points and prob
for i in range(250):
    for j in range(250):
        Z = pdf_5.evaluate((X[i], Y[j]))[0]  #evaluate the PDF
        points[i + j * 250, :] = [X[i], Y[j], Z]  #store the values in points␣
 ↪array
        prob[i, j] = Z  #store Z value in prob array

#mormalize prob and calculate the integral
norm_prob = prob / prob.sum()
n = 100
#list of probabilities to check
t = np.linspace(0, norm_prob.max(), n)
#norm_prob >= t[:, None, None] is a boolean array for every point in prob, for␣
 ↪every value of t
#if the norm_prob at a point is greater than t, the value is True (1) which is␣
 ↪then multiplied by the norm_prob
#if the norm_prob is not greater than t, the value is False (0) which is then␣
 ↪multiplied by the probability
#if it is True, the value is the probability, otherwise the value is zero
#sum these probabilities to get the total probability of every point with a␣
 ↪probability value lower than t
#integral stores an array of these sums for every t value
integral = ((norm_prob >= t[:, None, None]) * norm_prob).sum(axis=(1, 2))

#interpolate the integral to find corresponding t values
from scipy import interpolate
f = interpolate.interp1d(integral, t)
#find the probabilities where the integral of all the probabilities less than␣
 ↪equals 0.95 and 0.68
t_contours = f(np.array([0.95, 0.68]))

#contour plot using meshgrid for X and Y
plt.rcParams['figure.figsize'] = [8,4]
```

```
plt.contour(X_grid, Y_grid, norm_prob.T, t_contours, colors=["lightgreen",
 ↪"lightgreen"])
plt.show()
```



[13]:
```
##calculate the probability of the observations given an EOS; likelihood

#number of points of integration on the m-r curves for NICER and LIGO datasets
 ↪(LIGO is much more computationally expensive)
#LIGO data is 4D (m1, m2, lambda1, lambda2) since there are two neutron stars
 ↪in one dataset. Therefore, you have to
#integrate num_points_LIGO**2 points for every mass-radius curve.
num_points_NICER = 1000
num_points_LIGO = 100
def calc_P_obs_given_EOS(i):
    #various mass values
    min_m = np.min(EOSs[i][1])
    max_m = np.max(EOSs[i][1])
    len_m = max_m-min_m


    ###NICER section
    #r is the interpolated function r(m)
    r = sp.interpolate.interp1d(EOSs[i][1], EOSs[i][0])
    #radius at mass 1.4
    r_at_given_mass[i] = r(1.4)
    #important to note that num_points is set, not delta_m
```

```
    delta_m_NICER = len_m/(num_points_NICER-1)
    #m_NICER is an array of all the mass values to sum over
    m_NICER = np.linspace(min_m, max_m, num_points_NICER)

    #initialize probabilities of datasets
    P_NICER_1 = P_NICER_2 = P_NICER_3 = 0
    #loop over each radius value space by delta_m
    for j in range(num_points_NICER):
        #probability of EOS_i, using the NICER data, is the sum of the␣
↪probability of each point multiplied by the distance to the next point
        P_NICER_1 += pdf_2.evaluate((r(m_NICER[j]), m_NICER[j])) * delta_m_NICER
        P_NICER_2 += pdf_4.evaluate((r(m_NICER[j]), m_NICER[j])) * delta_m_NICER
        P_NICER_3 += pdf_5.evaluate((r(m_NICER[j]), m_NICER[j])) * delta_m_NICER

    ###LIGO section
    #lambda is the interpolated function lambda(m)
    lambda_func = sp.interpolate.interp1d(EOSs[i][1], EOSs[i][2])
    #change in mass between every point
    delta_m_LIGO = len_m/(num_points_LIGO-1)
    #m_LIGO is an array of all the mass values to sum over
    m_LIGO = np.linspace(min_m, max_m, num_points_LIGO)

    P_LIGO = 0
    #loop over each mass value to set m1 and lambda1
    for j in range(num_points_LIGO):
        m1 = m_LIGO[j]
        lambda1 = lambda_func(m1)
        #loop over each mass value to set m2 and lambda2
        for k in range(num_points_LIGO):
            m2 = m_LIGO[k]
            lambda2 = lambda_func(m2)

            #probability of EOS_i is the sum of the probability of each point␣
↪multiplied by the distance to the next point
            P_LIGO += pdf_3.evaluate((m1, m2, lambda1, lambda2)) *␣
↪delta_m_LIGO**2


    #then multiply the individual probabilities
    #P is the joint probability of the EOS according to every dataset
    P = P_NICER_1 * P_NICER_2 * P_NICER_3 * P_LIGO * pdf_1_evaluate(max_m)
    return P
```

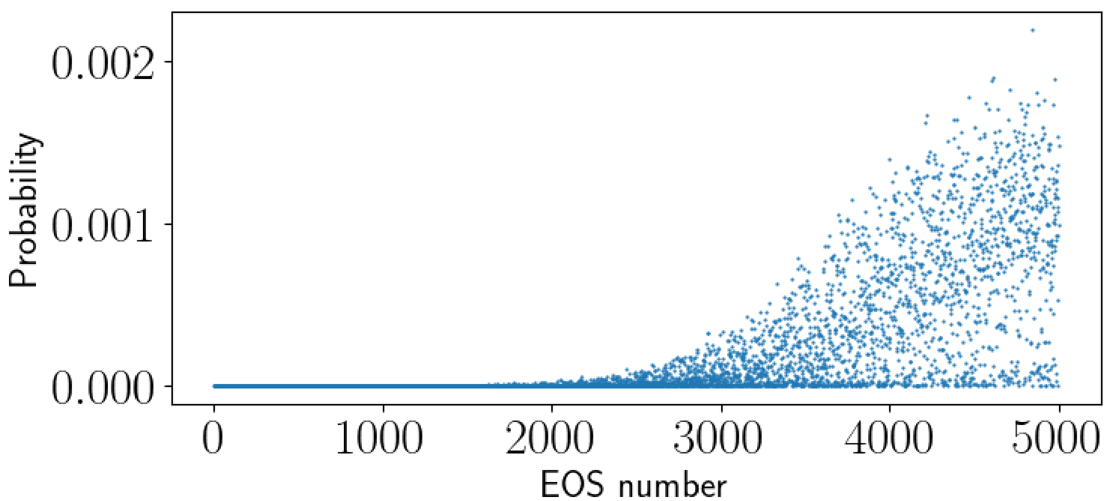
[14]:
```
start=time.time()
norm_factor = 0
for i in range(numofEOS):
    #calculate the likelihood
```

```
    P_obs_given_EOS[i] = calc_P_obs_given_EOS(i)
    #calculate the normalizing factor (sum of the numerator)
    norm_factor += P_obs_given_EOS[i]

#calculate the posterior which has a normalizing factor to make the total␣
 ↪probability one.
P_EOS_given_obs = P_obs_given_EOS / norm_factor
end=time.time()
print((end-start),'s', (end-start)/60, "m")
```

```
25946.604757785797 s 432.44341262976326 m
```

[15]:

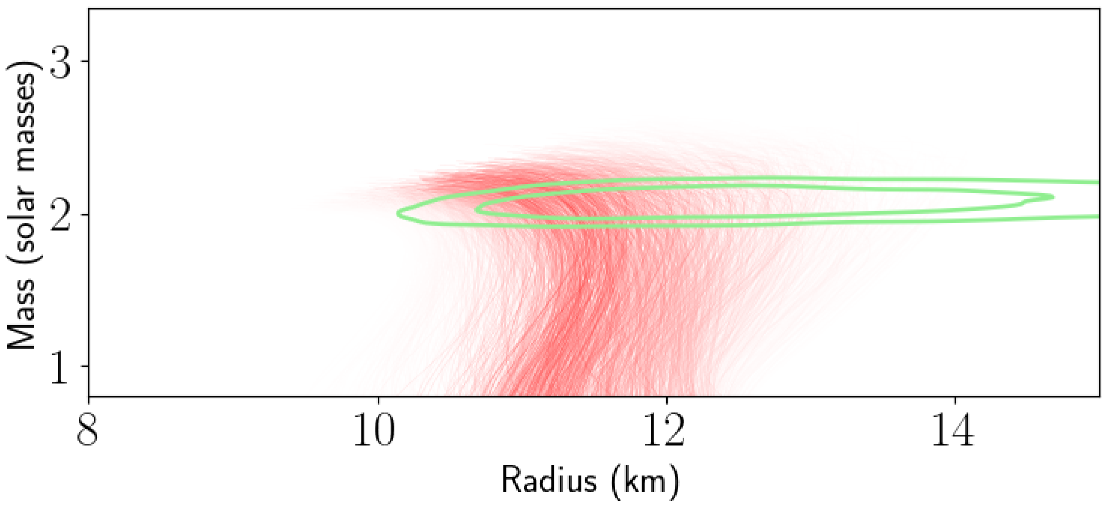
```
#plot every EOS with its probability
#there is a clear pattern since the EOSs are ordered by max mass
plt.xlabel("EOS number", fontsize=18)
plt.ylabel("Probability", fontsize=18)
plt.scatter(np.linspace(0,numofEOS,5000), P_EOS_given_obs, s=0.4)
plt.tight_layout()
plt.savefig("/home/tplohr/proj/SF 24-25/figs/NICER_2+3/NICER_2+3_scatter_plot.
 ↪png")
```



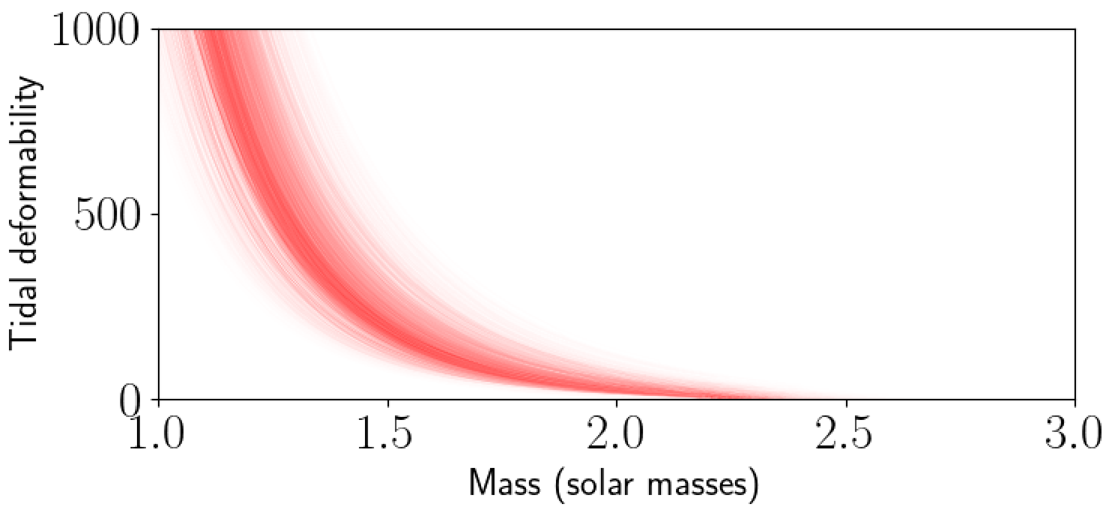
[16]:

```
#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.min(), P_EOS_given_obs.max())
#plots each EOS
plt.rcParams['figure.figsize'] = [8,4]
for i in range(numofEOS):
    color = cmap(norm(P_EOS_given_obs[i]))
    plt.xlim([8,15])
```

```
    plt.ylim([.8,3.35])
    plt.xlabel("Radius (km)", fontsize=18)
    plt.ylabel("Mass (solar masses)", fontsize=18)
    plt.plot(EOSs[i][0], EOSs[i][1], color=color, linewidth=.1)
plt.tight_layout()
plt.contour(X_grid, Y_grid, norm_prob.T, t_contours, colors=["lightgreen",␣
 ↪"lightgreen"])
plt.savefig("/home/tplohr/proj/SF 24-25/figs/NICER_2+3/NICER_2+3_EOS_prob_plot.
 ↪png")
```



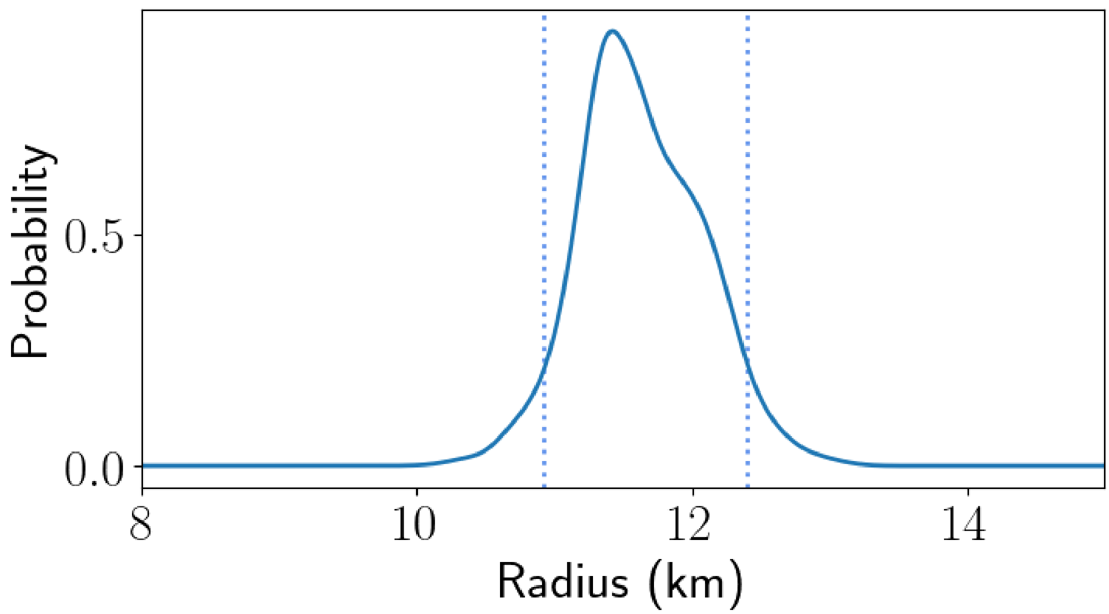
[17]:
```
#plots each EOS
plt.rcParams['figure.figsize'] = [8,4]
for i in range(numofEOS):
    color = cmap(norm(P_EOS_given_obs[i]))
    plt.xlim([1,3])
    plt.ylim([0,1000])
    plt.ylabel("Tidal deformability", fontsize=18)
    plt.xlabel("Mass (solar masses)", fontsize=18)
    plt.plot(EOSs[i][1], EOSs[i][2], color=color, linewidth=.1)
plt.tight_layout()
plt.savefig("/home/tplohr/proj/SF 24-25/figs/NICER_2+3/
 ↪NICER_2+3_deformability_EOS_plot.png")
```



[22]:

```
r_pdf = sp.stats.gaussian_kde((r_at_given_mass), weights=P_EOS_given_obs)
#plotting the radius pdf
Y = np.linspace(8, 15, 2000)
prob2=np.zeros(2000)
for i in range(2000):
    prob2[i] = r_pdf.evaluate(Y[i])

cdf = np.cumsum(prob2) * (Y[1] - Y[0])
indices = [np.searchsorted(cdf, 0.05), np.searchsorted(cdf, 0.5), np.
 ↪searchsorted(cdf, 0.95)]
percentiles = [Y[indices[0]], Y[indices[1]], Y[indices[2]]]
print(percentiles)
plt.xlabel("Radius (km)")
plt.ylabel("Probability")
plt.axvline(x=percentiles[0], color='cornflowerblue', linestyle='dotted',␣
 ↪label="5th Percentile")
plt.axvline(x=percentiles[2], color='cornflowerblue', linestyle='dotted',␣
 ↪label="95th Percentile")
plt.xlim(min(Y), max(Y))
plt.rcParams['figure.figsize'] = [8,4]
plt.plot(Y, prob2)
#plt.tight_layout()
plt.savefig("/home/tplohr/proj/SF 24-25/figs/NICER_2+3/NICER_2+3_r_PDF.png")
```

```
[10.920460230115058, 11.596298149074537, 12.401700850425213]
```



[23]:
```
current_time = time.localtime()
filename = time.strftime('%m-%d_%H', current_time)  # Month-Day_Hour (e.g.␣
↪01-20_15)
file_path = os.path.join("/home/tplohr/proj/SF 24-25/saves/NICER2+3/",␣
↪f"{filename}.txt")

# Save the probabilities to the text file
np.savetxt(file_path, P_EOS_given_obs)
```

[ ]:
```
```