StatisticsProblems2Solutions

November 9, 2022

1 Problem 2 - The Calorimeter Resolution (Solutions)

A calorimeter is a generic term for a device that measures the energy of some object. Traditionally this was done by measuring small temperature changes in some material due to the energy deposited by the object, but in particle physics experiments, since the energies are very small, the devices use sophisticated processes to measure the energies of incoming particles.

Designing a high performing calorimeter is crucial in the design of a good high energy particle physics detector. Having a good energy resolution is a key design challenge.

In this question, you will need to read in a dataset and estimate the energy resolution as a function of energy from the data.

In the file `problem_2_data.csv` is a dataset of recorded energy measurements of high energy electrons from a calorimeter. This is calibration data for the experiment so the true energy `e_true` is known for every electron. The recorded energy `e_measured` is provided by the calorimeter. We want to measure the resolution of the calorimeter using `e_measured - e_true` as a function of the true energy.

You can use a Jupyter notebook to help you structure your answers. You should clearly label any plots and present results clearly - **marks will be lost for poor quality figures or unclear results!** Some useful python modules are imported below.

First, read in the data from `electron_energies.csv`. You could read this in using Pandas or convert the data into some other format if you prefer. The file contains rows with different beam energy (`e_true`) values. These should be the following energies $e_{true} = 10, 15, 20, 25, 30, 35, 40, 45, 50$ GeV.

1.1 Part 1

a) [**1 mark**] For each `e_true` value draw draw a histogram of $e_{diff} = e_{measured} - e_{true}$.

b) [**2 marks**] We can assume that due to the resolution of the calorimeter, e_{diff} will be distributed as a Gaussian distribution $\phi(\mu_{e_{true}}, \sigma_{e_{true}})$, with parameters that are different for different values of e_{true}. For each value of e_{true} find the maximum likelihood estimators $\hat{\mu}_{e_{true}}$ and $\hat{\sigma}_{e_{true}}$. You can do this numerically using `scipy.optimize.minimize` or any other method. HINT: remember that to find the maximum likelihood estimator, you first need to write the likelihood function $L(\mu_{e_{true}}, \sigma_{e_{true}})$ for a particular dataset and then maximise it. Remember, it's nearly always easier to minimize $-\ln(L)$.

c) **[2 marks]** Plot the resulting Gaussian distribution $\phi(\hat{\mu}_{e_{true}}, \hat{\sigma}_{e_{true}})$ on top of the histograms from i). Remember, for this to look right, you're histograms should be drawn as *densities* (e.g. you can specify `density=True` in the plotting of the histogram). Also, plot the ratio of maximum likelihood estimator to the value of e_{true} i.e plot $\frac{\hat{\sigma}_{e_{true}}}{e_{true}}$ as a function of e_{true}.

1.2 Part 2

There are a number of different contributing factors to the resolution of a calorimeter but in calorimeters like the one at the CMS experiment, one large contribution is from the limited amount of scintillation light (number of photons) which is called *stochastic noise*. The resolution can be parameterised as

$$\sigma_{e_{true}} = S\sqrt{e_{true}}$$

where S is called the *stochastic term*. With the data, we can try to estimate S.

a) **[2 marks]** Write a new likelihood function $L(S, \vec{\mu}_{e_{true}})$ *combined* across the data from different e_{true} values. Remember that the total likelihood is the product over the individual likelihoods (or the sum if using $-\ln(L)$. The vector of values $\vec{\mu}_{e_{true}}$ are still free parameters but this time you should replace $\sigma_{e_{true}}$ in the Gaussian pdfs with $\sigma_{e_{true}} = S\sqrt{e_{true}}$. Calculate the maximum likelihood estimator $\hat{S}$. HINT: You will probably find it helpful to initialise the minimization with a value around $S \approx 0.03$.

b) **[2 marks]** What is the standard deviation on your estimator $\hat{S}$? To obtain this, you can use a bootstrapping method on the original dataset and calculate the sample standard deviation of the bootstrap samples - this may be quite slow so you should use no more than 100 bootstrap samples.

c) **[1 mark]** Finally make a plot of the function $\frac{\hat{S}\sqrt{e_{true}}}{e_{true}}$ overlaid on the individual fits you obtained in Part 1. How well do the points agree with your curve?

1.3 Answers below

[1]:
```python
%matplotlib inline

import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from scipy.stats import norm
#from scipy.optimize import minimize
from iminuit import minimize
from scipy.interpolate import interp1d

plt.rcParams.update({'font.size': 10})
```

1.4 Part 1

First, read in the data from `electron_energies.csv`. You could read this in using Pandas or convert the data into some other format if you prefer. The file contains rows with different beam energy (`e_true`) values. These should be the following energies $e_{true} = 10, 15, 20, 25, 30, 35, 40, 45, 50$ GeV.

a) **[1 mark]** For each `e_true` value draw draw a histogram of $e_{diff} = e_{measured} - e_{true}$.

```
[2]: E_points = range(10,55,5)
     n_points = len(E_points)
     print(list(E_points))
```

```
[10, 15, 20, 25, 30, 35, 40, 45, 50]
```

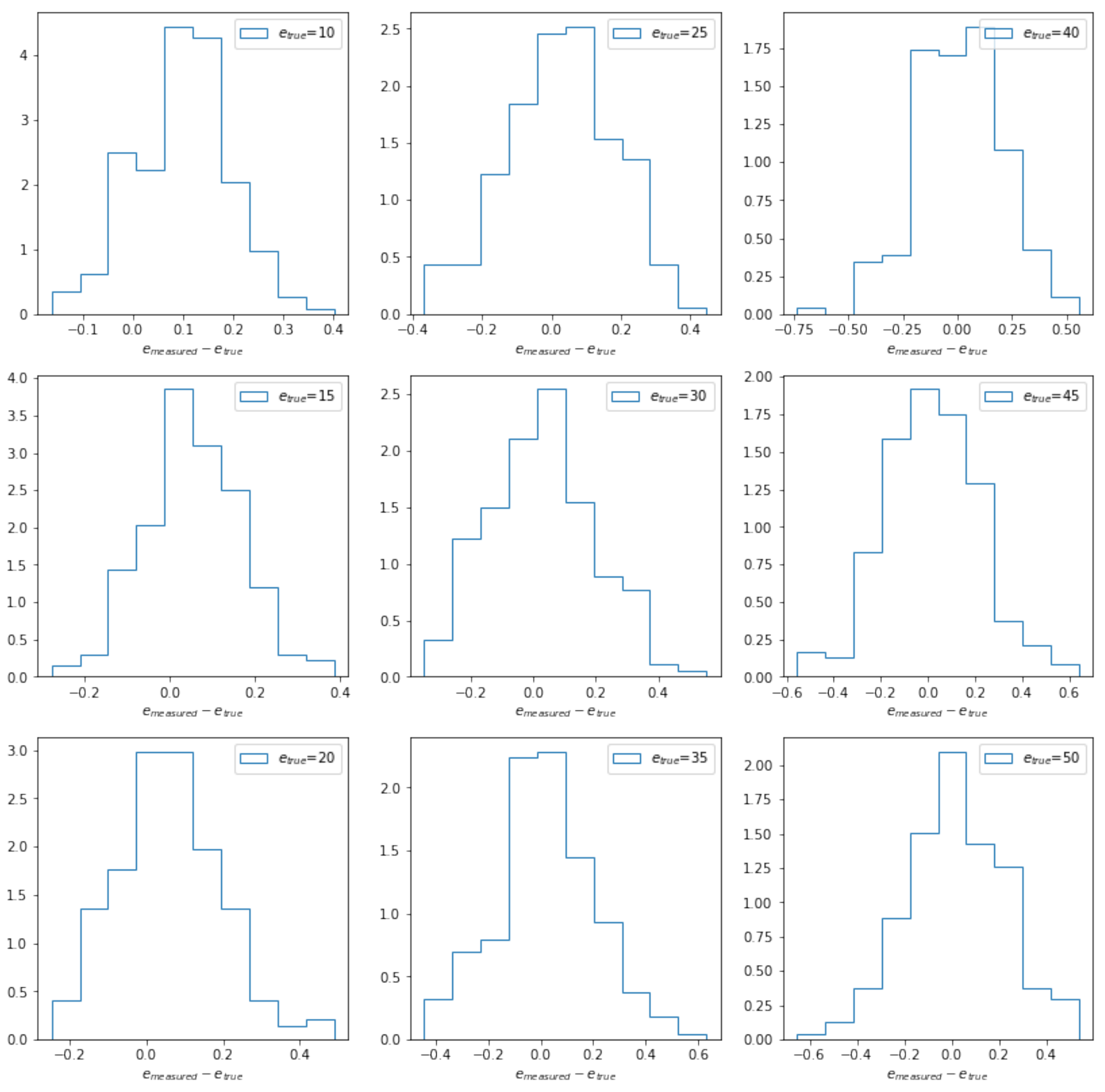
```
[3]: df = pd.read_csv('problem_2_data.csv')
     df.head()
```

```
[3]:    e_true  e_measured
     0      10   10.066899
     1      10   10.050873
     2      10    9.929206
     3      10   10.060034
     4      10   10.100859
```

```
[4]: df["ediff"]=df['e_measured']-df['e_true']
     df.head()
```

```
[4]:    e_true  e_measured     ediff
     0      10   10.066899  0.066899
     1      10   10.050873  0.050873
     2      10    9.929206 -0.070794
     3      10   10.060034  0.060034
     4      10   10.100859  0.100859
```

```
[5]: fig,axes = plt.subplots(3,3,figsize=(14,14))

     # we can draw the resulting fit on each histogram in a separate panel
     for i,E in enumerate(E_points):
         axes[i%3][int(i/3)].hist(df[df.
      ↪e_true==E]["ediff"],histtype='step',label="$e_{true}$=%d"%E,density=True)
         axes[i%3][int(i/3)].set_xlabel("$e_{measured} - e_{true}$")
         axes[i%3][int(i/3)].legend()
```



b) **[2 marks]** We can assume that due to the resolution of the calorimeter, e_{diff} will be distributed as a Gaussian distribution $\phi(\mu_{etrue}, \sigma_{etrue})$, with parameters that are different for different values of e_{true}. For each value of e_{true} find the maximum likelihood estimators $\hat{\mu}_{etrue}$ and $\hat{\sigma}_{etrue}$. You can do this numerically using `scipy.optimize.minimize` or any other method. HINT: remember that to find the maximum likelihood estimator, you first need to write the likelihood function $L(\mu_{etrue}, \sigma_{etrue})$ for a particular dataset and then maximise it. Remember, it's nearly always easier to minimize $-\ln(L)$.

[6]:
```
# fit ediff for each Energy value and make a plot

# define the function -2ln(L(mu,sigma)), call it q, for a single energy value.
# we'll minimize with respect to both parameters mu and sigma so this is called␣
↪q_free.
```

```
def q_free(x,data):
    mu=x[0]
    sigma=x[1]
    return -2*sum(np.log(norm.pdf(data,mu,sigma)))

mus   =[]
sigmas=[]

for i,E in enumerate(E_points):
    sigma_guess = np.std(df[df.e_true==E]["ediff"])
    mu_guess    = np.mean(df[df.e_true==E]["ediff"])
    res = minimize(q_free,[mu_guess, sigma_guess],args=[df[df.
 ↪e_true==E]["ediff"]])
    mus.append(res.x[0])
    sigmas.append(res.x[1])
```

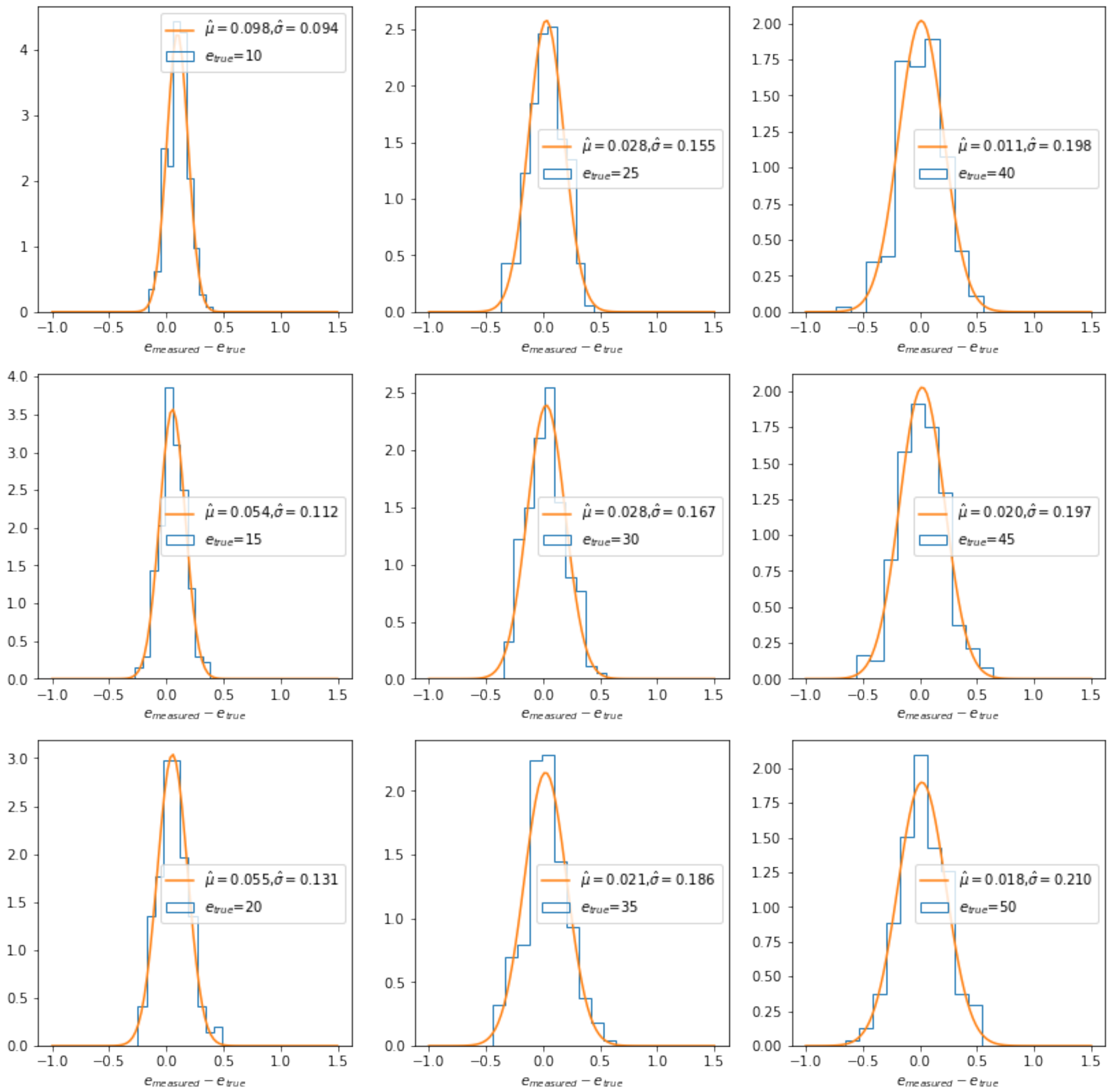
Print out the values of the maximum likelihood estimators, why not in a nice table?!

[7]:
```
print("e_true | mu_hat | sigma_hat")
print("----------------------------")
for i,E in enumerate(E_points):
    print("%.2f  |  %.3f | %.3f"%(E,mus[i],sigmas[i]))
```

```
e_true | mu_hat | sigma_hat
---------------------------
10.00  |  0.098 | 0.094
15.00  |  0.054 | 0.112
20.00  |  0.055 | 0.131
25.00  |  0.028 | 0.155
30.00  |  0.028 | 0.167
35.00  |  0.021 | 0.186
40.00  |  0.011 | 0.198
45.00  |  0.020 | 0.197
50.00  |  0.018 | 0.210
```

c) **[2 marks]** Plot the resulting Gaussian distribution $\phi(\hat{\mu}_{e_{true}}, \hat{\sigma}_{e_{true}})$ on top of the histograms from i). Remember, for this to look right, you're histograms should be drawn as *densities* (e.g. you can specify `density=True` in the plotting of the histogram). Also, plot the ratio of maximum likelihood estimator to the value of e_{true} i.e plot $\frac{\hat{\sigma}_{e_{true}}}{e_{true}}$ as a function of e_{true}.

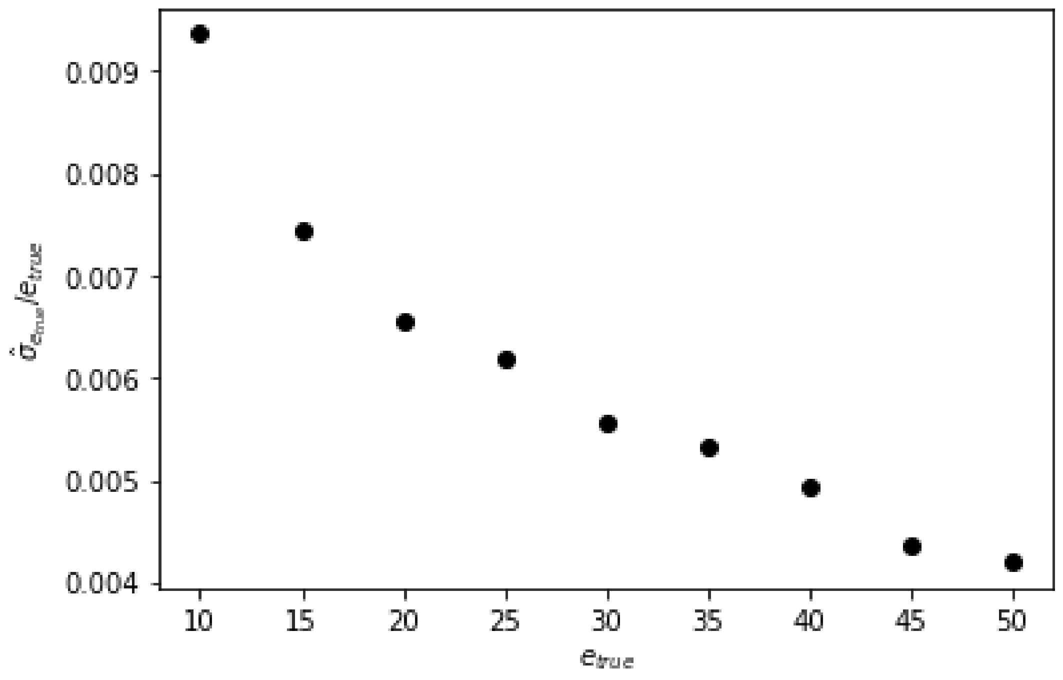
[8]:
```
# we can draw the resulting fit on each histogram in a separate panel

xrange = np.linspace(-1,1.5,100)

fig,axes = plt.subplots(3,3,figsize=(14,14))

for i,E in enumerate(E_points):
    axes[i%3][int(i/3)].hist(df[df.
↪e_true==E]["ediff"],histtype='step',label="$e_{true}$=%d"%E,density=True)
    mu_fitted = mus[i]
    sigma_fitted = sigmas[i]
    axes[i%3][int(i/3)].plot(xrange,norm.pdf(xrange,mu_fitted,sigma_fitted)
                                ,label="$\hat{\mu}=$%.3f,$\hat{\sigma}=$%.
↪3f"%(mu_fitted,sigma_fitted))
    axes[i%3][int(i/3)].set_xlabel("$e_{measured} - e_{true}$")
    axes[i%3][int(i/3)].legend()
```



[9]:
```
# And also plot the values of hat{sigma}/e_true vs e_true
```

```
plt.plot(E_points,[s/E for s,E in␣
 ↪zip(sigmas,E_points)],marker="o",color='black',linestyle="")
plt.xlabel("$e_{true}$")
plt.ylabel("$\hat{\sigma}_{e_{true}}/e_{true}$")
```

```
[9]: Text(0, 0.5, '$\\hat{\\sigma}_{e_{true}}/e_{true}$')
```



1.5 Part 2

There are a number of different contributing factors to the resolution of a calorimeter but in calorimeters like the one at the CMS experiment, one large contribution is from the limited amount of scintillation light (number of photons) which is called *stochastic noise*. The resolution can be parameterised as

$$\sigma_{e_{true}} = S\sqrt{e_{true}}$$

where S is called the *stochastic term*. With the data, we can try to estimate S.

a) **[2 marks]** Write a new likelihood function $L(S, \vec{\mu}_{e_{true}})$ *combined* across the data from different e_{true} values. Remember that the total likelihood is the product over the individual likelihoods (or the sum if using $-\ln(L)$. The vector of values $\vec{\mu}_{e_{true}}$ are still free parameters but this time you should replace $\sigma_{e_{true}}$ in the Gaussian pdfs with $\sigma_{e_{true}} = S\sqrt{e_{true}}$. Calculate the maximum likelihood estimator $\hat{S}$. HINT: You will probably find it helpful to initialise the minimization with a value around $S \approx 0.03$.

[10]:
```
# Set-up a new fit that estimates the term S.
# We'll also look at the likelihood curve as a function of S to estimate an␣
 ↪uncertainty
def res_S(S,E):
    return S*np.sqrt(E)

def q_partial(x,args):
    # mu will still be a free parameter
    mu=x[1]

    E    = args[0]
    data = args[1]

    # sigma is now a function of S
    sigma=res_S(x[0],E)

    return sum(np.log(norm.pdf(data,mu,sigma)))

def q_S_free(x,args):
    S = x[0]
    mus = x[1:]
    df = args
    return -2*sum([q_partial([S,mus[i]],[E_points[i],df[df.
 ↪e_true==E_points[i]]["ediff"]])
                for i in range(n_points)] )
```

[11]:
```
mu_guesses = [np.mean(df[df.e_true==E]["ediff"]) for E in E_points]
init = [0.03]
init.extend(mu_guesses)

boundaries = [[0.01,0.1]].extend([[-1,2] for m in mu_guesses])

res_free = minimize(q_S_free,init,args=[df],bounds=boundaries)
s_hat = res_free.x[0]
print("Maximum Likelihood Estimator for S =",s_hat)
```

```
Maximum Likelihood Estimator for S = 0.03011035779601544
```

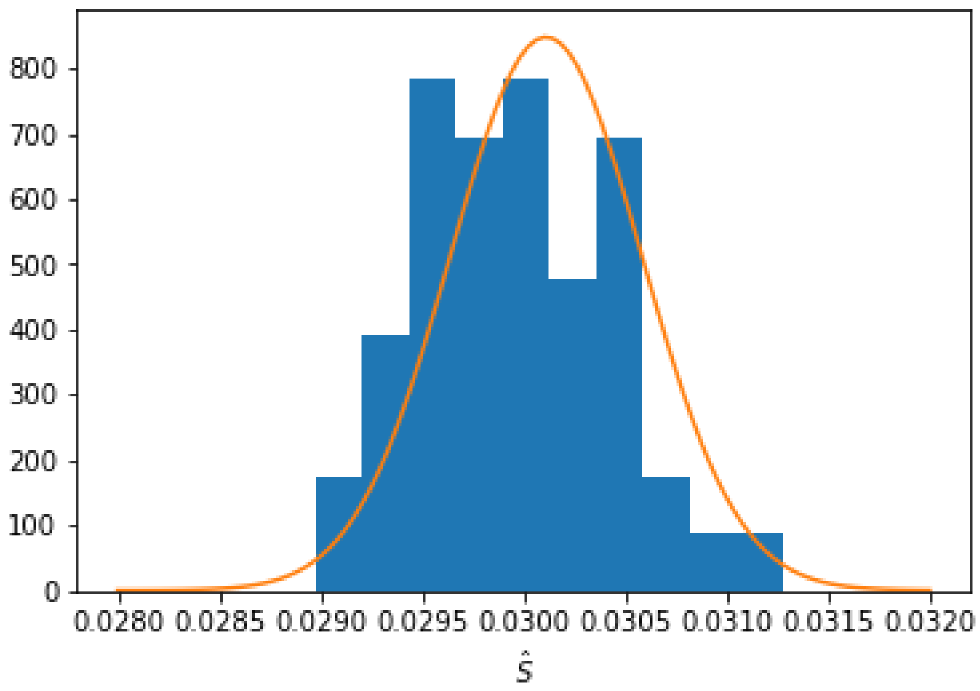
b) [**2 marks**] What is the standard deviation on your estimator $\hat{S}$? To obtain this, you can use a bootstrapping method on the original dataset and calculate the sample standard deviation of the bootstrap samples - this may be quite slow so you should use no more than 100 bootstrap samples.

[12]:
```
import random

def bootstrap_dataset(data):
    # create a new dataframe from sampling the original one
    # first choose the rows we are going to keep (with replacement)
    data_size = len(data.index)
    bs_indeces = random.choices(range(data_size),k=data_size)
    df_new = data.iloc[bs_indeces]
    return df_new

    # note that I could also have used the pandas dataframe function "sample"
    # as in below, but I would need to sample from the full dataframe and run␣
 ↪many
    # more bootstrap samples to get an accurate answer

    # df_new = data.sample(n=200*9,replace=True)
    # return df_new

bs_estimators = []

for i in range(100):
    new_df = bootstrap_dataset(df)
    bs_res = minimize(q_S_free,init,args=[new_df],bounds=boundaries)
    bs_s_hat = bs_res.x[0]
    bs_estimators.append(bs_s_hat)
```

We could also plot a Gaussian on top of the distribution with $\mu = \hat{S}$ and $\sigma = Sd(\hat{S})$ to check that standard deviation looks about right.

[13]:
```
plt.hist(bs_estimators,density=True)
plt.xlabel("$\hat{S}$")
std_dev_s = np.std(bs_estimators)



xlin = np.linspace(0.028,0.032,100)
plt.plot(xlin,norm.pdf(xlin,loc=s_hat,scale=std_dev_s))

print("s_hat=",s_hat)
print("standard deviation of s_hat from bootstrap = ",std_dev_s)
```

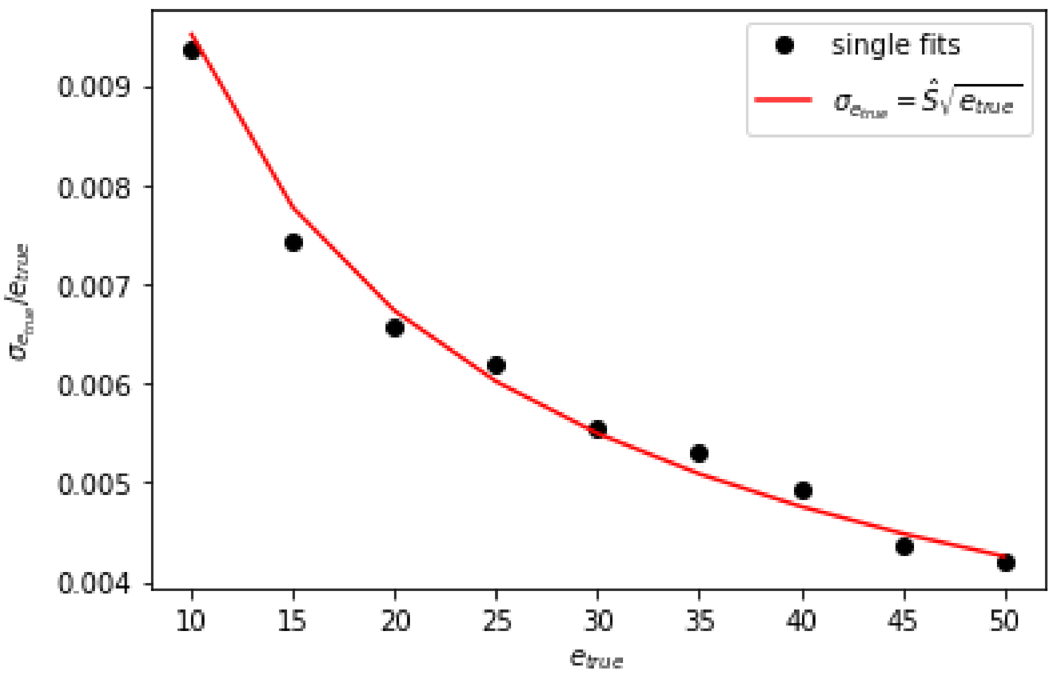
```
s_hat= 0.03011035779601544
standard deviation of s_hat from bootstrap =  0.00047021941798761676
```



c) [**1 mark**] Finally make a plot of the function $\frac{\hat{S}\sqrt{e_{true}}}{e_{true}}$ overlaid on the individual fits you obtained in Part 1. How well do the points agree with your curve?

```
[14]: plt.plot(E_points,[s/E for s,E in␣
       ↪zip(sigmas,E_points)],marker="o",label='single␣
       ↪fits',color='black',linestyle="")
      plt.plot(E_points,[res_S(s_hat,E)/E for E in␣
       ↪E_points],color='red',label='$\sigma_{e_{true}}=\hat{S}\sqrt{e_{true}}$')
      plt.xlabel("$e_{true}$")
      plt.ylabel("$\sigma_{e_{true}}/e_{true}$")
      plt.legend()
```

```
[14]: <matplotlib.legend.Legend at 0x7fc578d49a50>
```



The points agree well with the curve for the MLE of S. This is enough to get all of the marks for this Question.

We could have included a shaded band around the curve to indicate how much the curve could vary within the standard deviation we calculated for S. There are a few ways this could be done but below is one method which is fairly robust.

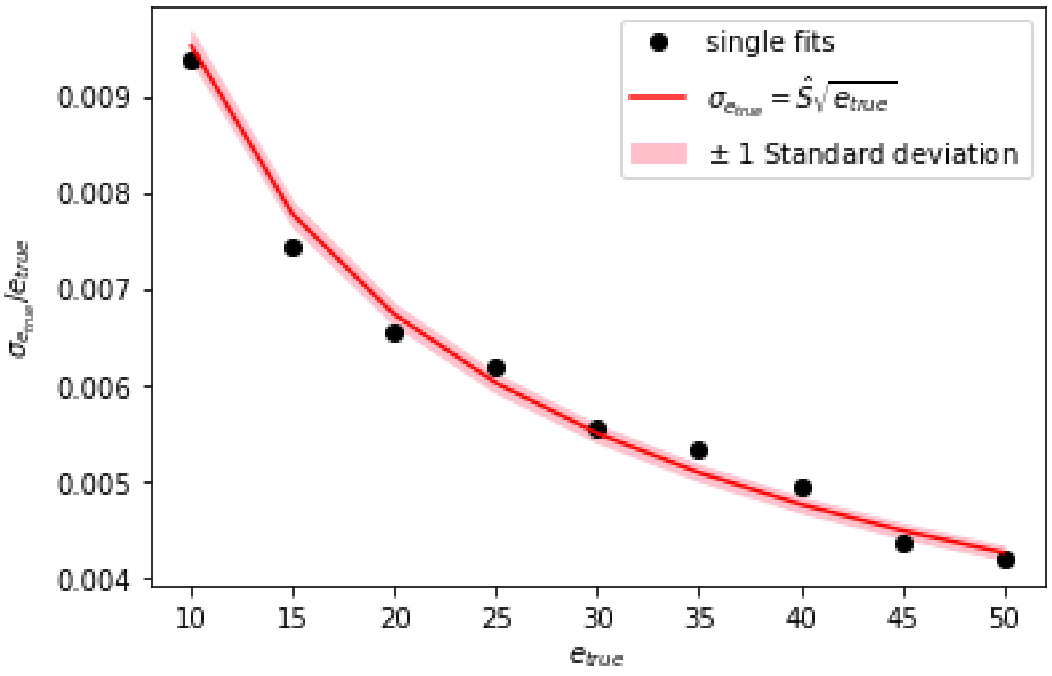
For each point at which the curve is evaluated, calculate the standard deviation of $\hat{S}/\sqrt{(}e_{true})$. Since e_{true} is a constant, we know that $V\left(\frac{\hat{S}}{\sqrt{(}e_{true}}\right) = \frac{1}{e_{true}}V(\hat{S})$ and since the variance (V) is just the square of the standard deviation (let's call it "Sd"), we find

$$Sd\left(\frac{\hat{S}}{\sqrt{(}e_{true}}\right) = \frac{1}{\sqrt{e_{true}}}Sd(\hat{S})$$

We could also include for each point the standard deviation of the estimator of $\hat{\sigma}$ from each e_{true} individual dataset as error bars - we could bootstrap each of these too (but i'll leave that as an extra exercise if you're interested).

```
[15]: plt.plot(E_points,[s/E for s,E in␣
       ↪zip(sigmas,E_points)],marker="o",label='single␣
       ↪fits',color='black',linestyle="")
      plt.plot(E_points,[res_S(s_hat,E)/E for E in␣
       ↪E_points],color='red',label='$\sigma_{e_{true}}=\hat{S}\sqrt{e_{true}}$')
      plt.fill_between(E_points
                       ,[res_S(s_hat,E)/E-std_dev_s/np.sqrt(E) for E in E_points]
                       ,[res_S(s_hat,E)/E+std_dev_s/np.sqrt(E) for E in E_points]
                       ,color='pink'
                       ,label='$\pm$ 1 Standard deviation')
      plt.xlabel("$e_{true}$")
      plt.ylabel("$\sigma_{e_{true}}/e_{true}$")
      plt.legend()
```

[15]:
```
<matplotlib.legend.Legend at 0x7fc5785357d0>
```


single fits
$\sigma_{e_{true}} = \hat{S}\sqrt{e_{true}}$
± 1 Standard deviation
$\sigma_{e_{true}}/e_{true}$
e_{true}