MoMeDa_4

June 16, 2024

1 Moderne Methoden der Datenanalyse SS2024

2 Practical Exercise 4

2.1 Exercise 4: Combination of correlated measurements

A common problem in science is the combination of several measurements to one single result, e.g., the average value. Not only the uncertainties of the individual measurements have to be taken into account, but also the correlations between them. A wrong treatment of correlations or common systematic effects can lead to biased results.

2.2 Exercise 4.1: Combination of W mass measurements (voluntary)

At the LEP accelerator at CERN the mass of the W boson m_W was measured in two different channels:

$$\begin{aligned} e^+e^- &\rightarrow W^+W^- \rightarrow q_1q_2q_3q_4 \\ e^+e^- &\rightarrow W^+W^- \rightarrow l\nu q_1q_2 \end{aligned}$$

The experimental signature in the detector for the first channel with four quarks are four reconstructed jets. The second channel is identified by a lepton (electron or muon) and two jets. The neutrino is not detected. The measured W masses and the uncertainties are:

$$\begin{aligned} \text{4 jets channel:} \quad m_W &= (80457 \pm 30 \pm 11 \pm 47 \pm 17 \pm 17)\ \text{MeV} \\ \text{lepton + 2 jets channel:} \quad m_W &= (80448 \pm 33 \pm 12 \pm\ 0 \pm 19 \pm 17)\ \text{MeV} \end{aligned}$$

To facilitate the interpretation of the results, different uncertainties are given, originating from different sources: The first two uncertainties are the statistical and systematic experimental uncertainties, which are uncorrelated. The third uncertainty originates from the theory applied for the analysis and is only present in the first channel. The fourth uncertainty comes from a common theoretical model applied for both channels, and thus is 100% correlated. Also the last uncertainty is 100% correlated between both measurements, since it represents the uncertainty on the LEP accelerator beam energy.

- Construct a covariance matrix of the two W mass measurements taking into account all uncertainties and their correlations. Use this covariance matrix to define a χ^2 expression

containing the average W mass $\bar{m}_W$ as a free parameter. Determine $\bar{m}_W$ and its uncertainty by minimizing the χ^2 expression using the IMinuit package.

For this exercise, you have to write your own χ^2-function to be minimized; see the previous exercise to learn how this can be done. There we have already used `iminuit`, however this time we'll be using the object-oriented interface. You can find some hints on how to use this in the documentation here.

Hint: Make sure, that you are using a current version of `iminuit`, e.g. 2.11.2! Check this with the command in the next cell:

[3]:
```
!pip list | grep iminuit
```

```
iminuit                       2.25.2
```

[4]:
```
import numpy as np
from iminuit import Minuit
from scipy.linalg import inv
import matplotlib.pyplot as plt
```

[5]:
```
u1 = np.array([30,11,47,17,17])
u2 = np.array([33,12,0,19,17])
mw = np.array([80457,80448])

cov = np.array([[np.sum(u1[[0,2,3,4]]**2), np.sum(u1[[3,4]]*u2[[3,4]])],
       [np.sum(u1[[3,4]]*u2[[3,4]]), np.sum(u2[[0,2,3,4]]**2)]])
cov
```

[5]:
```
array([[3687,  612],
       [ 612, 1739]])
```

[6]:
```
def chi_square_creator(measurement_vector, inv_cov):

    # Using this outer function to pass the measured values (measurement_vector)␣
 ↪and the inverse covariance matrix (inv_cov) to the function so we don't need␣
 ↪to define them globally

    def chi_square_function(par):
        #cnorm = np.sqrt(np.linalg.det(inv_cov)/4*np.pi**2)
        #(measurement_vector - par)

       #chi2_value = -(cnorm * np.exp(-1/2*np.sum((measurement_vector - par)*np.
 ↪dot(inv_cov,measurement_vector - par))))**2
        #print((measurement_vector - par), np.dot(inv_cov,measurement_vector -␣
 ↪par))

        chi2_value = np.sum((measurement_vector - par)*np.
 ↪dot(inv_cov,measurement_vector - par))
```

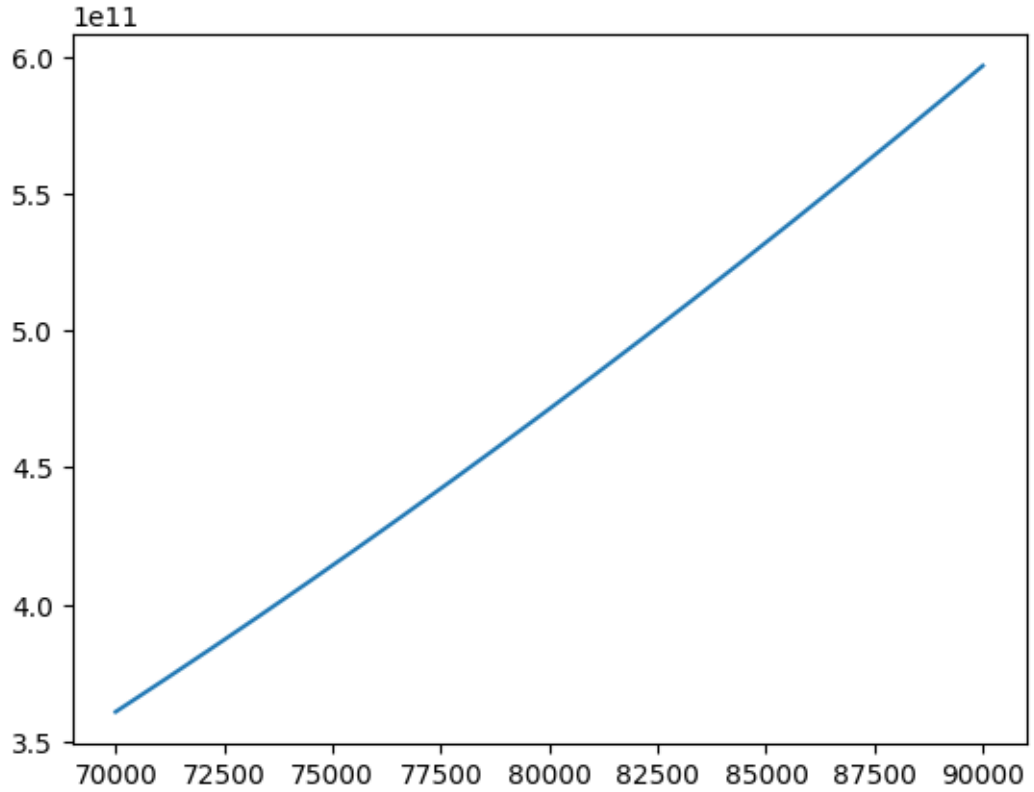
```
        return chi2_value # return the chi2 value

    return chi_square_function # return the function which calculates the value
```

As this is using a different interface to IMinuit than in Exercise 3.2, here is a hint: These are the base two lines you'll need:

```
minuit_instance = Minuit(function, parametername=initial_parametervalue)
res = minuit_instance.migrad()
```

The first line initializes the minuit object, gives the cost function and passes the initial parameter names and values. The second line performs the minimization using the `MIGRAD` algorithm.

[13]:
```
p = np.linspace(70000,90000,200)
plt.plot(p, [chi_square_creator(mw, inv_cov)(pp) for pp in p])
plt.show()
```



[14]:
```
# TODO: Add code here to calculate the (inverse) covariance matrix
# You can use scipy.linalg.inv to invert the matrix
inv_cov = inv(cov)
```

```
# Perform the minimization of the chi2 function
minuit_instance = Minuit(chi_square_creator(mw, inv_cov), par=80400)

# Get results of the minimization and plot or print them
res = minuit_instance.migrad()
res
```

[14]:

Migrad	
FCN = 4.587 EDM = 6.12e-10 (Goal: 0.0002)	Nfcn = 15
Valid Minimum	Below EDM threshold (goal x 10)
No parameters at limit	Below call limit
Hesse ok	Covariance accurate

	Name	Value	Hesse Err	Minos Err-	Minos Err+	Limit-	Limit+	Fixed
0	par	8.23	0.12					

	par
par	0.0136

- Because the minimization of the χ^2 expression in exercise 4.1 is a linear problem it can be solved analytically. Determine $\bar{m}_W$ and its error analytically and compare them to the result from above.

TODO: Add your calculations here using the Latex syntax

- Estimate the contributions from statistical, systematic, theoretical, and accelerator based uncertainties to the error of the combined W mass measurement. Use the quadratic difference between the total error and the error calculated with a covariance matrix where one component is removed.

[15]:
```
# TODO: Add your code here to calculate the contribution of each uncertainty␣
↪component
```

2.3 Exercise 4.2: Normalisation uncertainty (obligatory)

Two measurements $y_1 = 8.0$ and $y_2 = 8.5$ of the same physical quantity with an uncorrelated relative statistical error of 2 % and a common normalisation error of 10 % should be combined.

- Construct a covariance matrix and a χ^2 expression and determine its minimum with `iminuit` or analytically. If you need hints on how to use `iminuit`, consult the exercises 3.2 and 4.1.

[16]:
```
#import ROOT # Only need for plotting... Feel free to replace the plot method␣
↪with your own pure python implementation!
```

[17]:
```
from scipy.stats import t

cov = np.array([[(8*0.02)**2+(8*0.1)**2, (8*0.1)*(8.5*0.1)],
                [(8*0.1)*(8.5*0.1), (8.5*0.02)**2+(8.5*0.1)**2]])

display(cov)
mw = np.array([8,8.5])

def chi2_creator_4_2_1(measurement_vector, inv_cov):

    # We are using this outer function to pass the measured values␣
↪(measurement_vector) and the inverted covariance matrix (inv_cov)
    # to the function so we don't need to define them globally...

    def chi2_function(par):
        '''
        calculate chi2 using 1 parameter for the mean
        '''
        #cnorm = np.sqrt(np.linalg.det(inv_cov)/4*np.pi**2)
        #(measurement_vector - par)

       #chi2_value = -(cnorm * np.exp(-1/2*np.sum((measurement_vector - par)*np.
↪dot(inv_cov,measurement_vector - par))))**2
        chi2_value = np.sum((measurement_vector - par)*np.
↪dot(inv_cov,measurement_vector - par))

        return chi2_value # return the chi2 value

    return chi2_function

# TODO: Add code here to calculate the (inverse) covariance matrix
# You can use scipy.linalg.inv to invert the matrix
inv_cov = inv(cov)

# Perform the minimization of the chi2 function
minuit_instance = Minuit(chi_square_creator(mw, inv_cov), par=8)
```

```
# Get results of the minimization and plot or print them
res = minuit_instance.migrad()
res
```

```
array([[0.6656, 0.68  ],
       [0.68  , 0.7514]])
```

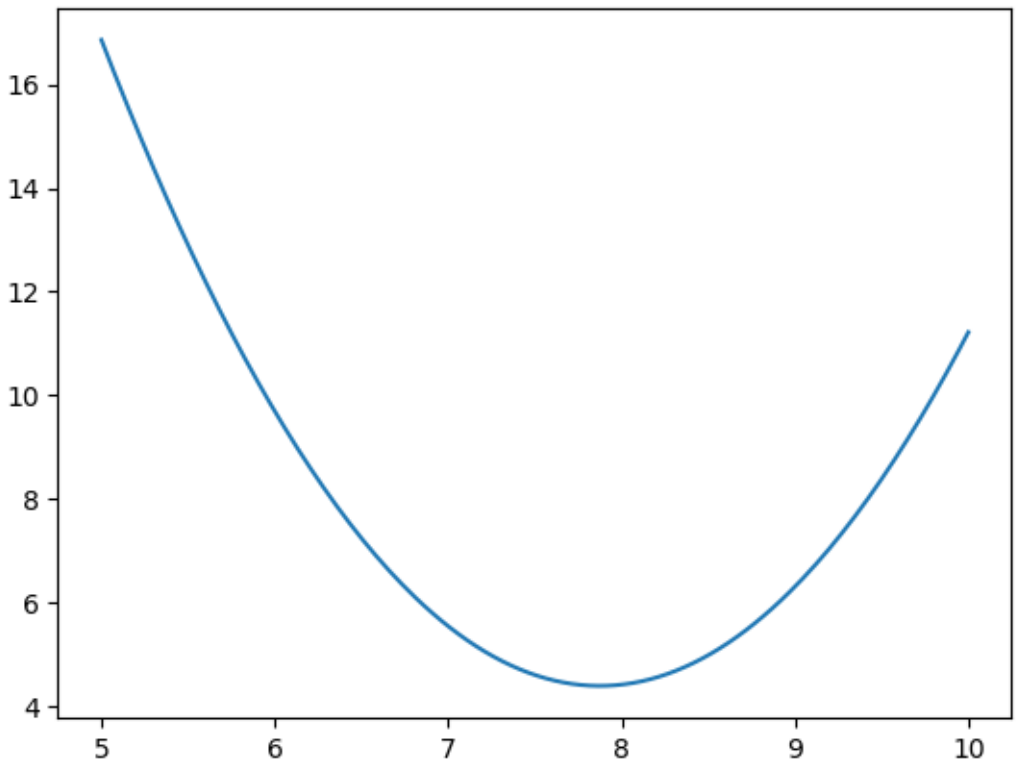
[17]:

Migrad	
FCN = 4.386 EDM = 8.64e-24 (Goal: 0.0002)	Nfcn = 11
Valid Minimum	Below EDM threshold (goal x 10)
No parameters at limit	Below call limit
Hesse ok	Covariance accurate

	Name	Value	Hesse Err	Minos Err-	Minos Err+	Limit-	Limit+	Fixed
0	par	7.9	0.8					

	par
par	0.662

[18]:
```
p = np.linspace(5,10,200)
plt.plot(p, [chi_square_creator(mw, inv_cov)(pp) for pp in p])
plt.show()

from scipy.optimize import minimize
minimize(chi_square_creator(mw, inv_cov), x0=(8,))
```



[18]:
```
  message: Optimization terminated successfully.
  success: True
   status: 0
      fun: 4.385964912280704
        x: [ 7.874e+00]
      nit: 1
      jac: [ 0.000e+00]
 hess_inv: [[1]]
     nfev: 6
     njev: 3
```

- Is the result reasonable? What could be the cause for the unexpected value? Make a plot of the covariance ellipse in the $y_1'y_2'$ plane defined by

$$\Delta y^T V^{-1} \Delta y = c^2, \quad \Delta y = \begin{pmatrix} y_1 - y_1' \\ y_2 - y_2' \end{pmatrix}$$

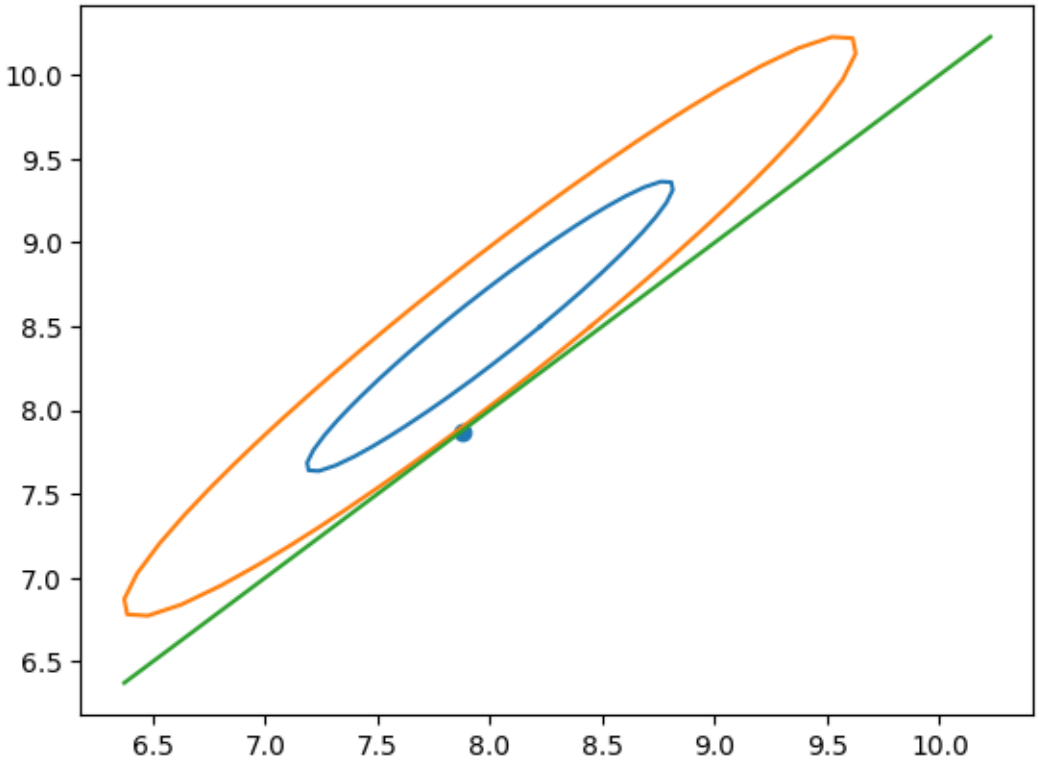
  for $c = 1$ and $c = 2$ together with the line $y_1' = y_2'$. V is the covariance matrix. To draw the ellipse a TGraph object can be used. The points on the ellipse can be calculated as a function of the angle ϕ if Δy is expressed by ϕ and the radius r. You can use the function below to draw the ellipse. Pay attention to where the bisector intersects with the ellipse.

3 Das Ergebnis ist nicht plausibel. Es sollte zwischen 8 und 8.5 liegen. Das Problem ist, dass die relative Unsicherheit relativ zum Messwert und nicht relativ zum tatsächlichen Wert betrachtet wird.

[19]:
```
import warnings

def drawCovEllipse(CI: list, vec: list, mean: float):
    '''
    CI : inverse covariance matrix
    vec: measured values i.e. [y1, y2]
    mean: calculated mean value from chi2 minimization

    Draws and returns the covariance ellipses as well as the bisect line.
    The outputs have to be stored in some variables or they will not be displayed.
    '''

    npoints = 200
    e1 = []
    e2 = []
    #ellipse1 = ROOT.TGraph(npoints + 1)
    #ellipse2 = ROOT.TGraph(npoints + 1)
    for i in range (npoints + 1):
        phi = 2 * i * np.pi / npoints
        V = [np.cos(phi), np.sin(phi)]
        v = np.matrix(V)
        sp = np.dot(v, np.dot(CI, v.getT()))
        R = np.sqrt(1.0 / sp)
        with warnings.catch_warnings():
            warnings.simplefilter("ignore")
            e1.append( (float(vec[0] + R * V[0]), float(vec[1] + R * V[1])) )
            e2.append( (float(vec[0] + 2 * R * V[0]), float(vec[1] + 2 * R *␣
 ↪V[1])) )
        #ellipse1.SetPoint(i, vec[0] + R * V[0], vec[1] + R * V[1])
        #ellipse2.SetPoint(i, vec[0] + 2 * R * V[0], vec[1] + 2 * R * V[1])

    #print(*zip(*e1))
    plt.plot(*zip(*e1))
    plt.plot(*zip(*e2))

    a,b = np.min([e1,e2]),np.max([e1,e2])
    a = a
    b = b

    plt.plot([a,b],[a,b])
    plt.scatter([mean],[mean])
```

```
    plt.show()
```

[20]:
```
drawCovEllipse(inv_cov, mw, res.params["par"].value)
```



- Use an additional normalisation parameter N for the treatment of the common normalisation uncertainty σ_{norm} instead of taking it into account in the covariance matrix of y_1 and y_2. Add a term to the χ^2 expression for the normalisation with an expected value of 1 and an error of 10 %. The normalisation factor N can be applied either to the measured values y_i or to the fit parameter $\bar{y}$. Try out both ways (using `iminuit` for the χ^2 minimisation) and compare the results. Which one is the more meaningful result and why?

Determine $\bar{y}$ from the correct χ^2 expression in an analytical way. How does the normalisation error affect the averaged value and its error?

$$0 = \frac{\partial \chi^2}{\partial p} = -\frac{2n(-np + x1)}{\sigma_x^2} - \frac{2n(-np + x2)}{\sigma_x^2}$$

$$0 = (-np + x1) + (-np + x2)$$

$$p = \frac{x1 + x2}{2n}$$

$$\Rightarrow \chi^2 = \frac{(n-1)^2}{\sigma_n^2} + \frac{(\mathrm{x1} - \mathrm{x2})^2}{2\sigma_x^2}$$

$$0 = \frac{d\chi^2}{dn} = \frac{2(n-1)}{\sigma_n^2}$$

$$\Rightarrow n = 1$$

$$\Rightarrow p = \frac{x1 + x2}{2}$$

Nach Mathematica gilt folgt:

$$\chi^2(\frac{x1 + x2}{2n}, n) = \chi^2(\frac{x1 + x2}{2}, 1) + 1$$

$$\Rightarrow n = 1 \pm \sigma_n$$

[21]:
```
cov = np.array([[(8*0.02)**2, 0],
                [0, (8.5*0.02)**2]])

display(cov)
mw = np.array([8,8.5])

def chi2_creator_4_2_2(measurement_vector, inv_cov, sigma_norm):

    # Now we additionally have to set the normalisation uncertainty sigma_norm␣
 ↪in the cost function.

    def chi2_function(par, N):
        '''
        calculate the chi2 including an additional parameter for the␣
 ↪normalization of the measured values (version 1)
        '''
        chi2_value = np.sum((measurement_vector*N - par)*np.
 ↪dot(inv_cov,measurement_vector*N - par)) + (N-1)**2/sigma_norm**2

        return chi2_value # return the chi2 value

    return chi2_function

def chi2_creator_4_2_3(measurement_vector, inv_cov, sigma_norm):

    def chi2_function(par, N):
        '''
```

```
        calculate the chi2 including an additional parameter for the␣
 ↪normalization of the mean value (version 2)
        '''
        chi2_value = np.sum((measurement_vector - par*N)*np.
 ↪dot(inv_cov,measurement_vector - par*N)) + (N-1)**2/sigma_norm**2

        return chi2_value # return the chi2 value

    return chi2_function

inv_cov = inv(cov)

minuit_instance1 = Minuit(chi2_creator_4_2_2(mw, inv_cov, 0.1), par=8, N=1)
res1 = minuit_instance1.migrad()
display(res1)


minuit_instance2 = Minuit(chi2_creator_4_2_3(mw, inv_cov, 0.1), par=8, N=1)
res2 = minuit_instance2.migrad()
display(res2)
```

```
array([[0.0256, 0.    ],
       [0.    , 0.0289]])
```

Migrad	
FCN = 4.386 EDM = 4.2e-20 (Goal: 0.0002)	Nfcn = 34
Valid Minimum	Below EDM threshold (goal x 10)
No parameters at limit	Below call limit
Hesse ok	Covariance accurate

	Name	Value	Hesse Err	Minos Err-	Minos Err+	Limit-	Limit+	Fixed
0	par	7.9	0.8					
1	N	0.96	0.10					

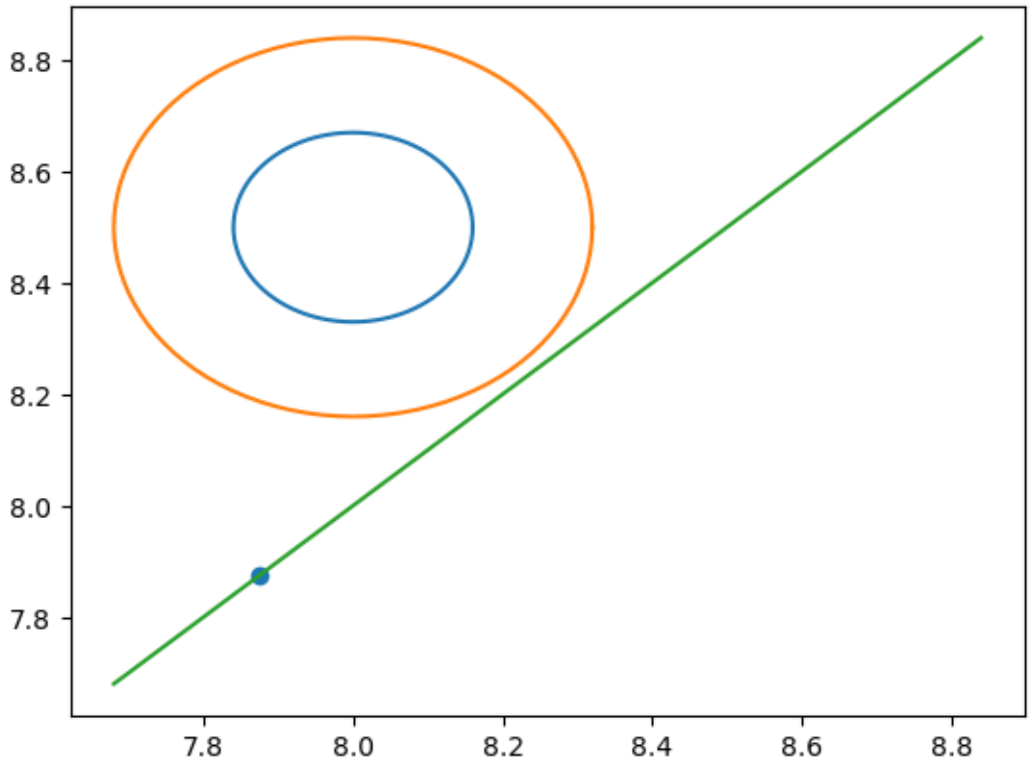
	par	N
par	0.662	0.079
N	0.079	0.00956

Migrad	
FCN = 4.587 EDM = 2.1e-06 (Goal: 0.0002)	Nfcn = 45
Valid Minimum	Below EDM threshold (goal x 10)
No parameters at limit	Below call limit
Hesse ok	Covariance accurate

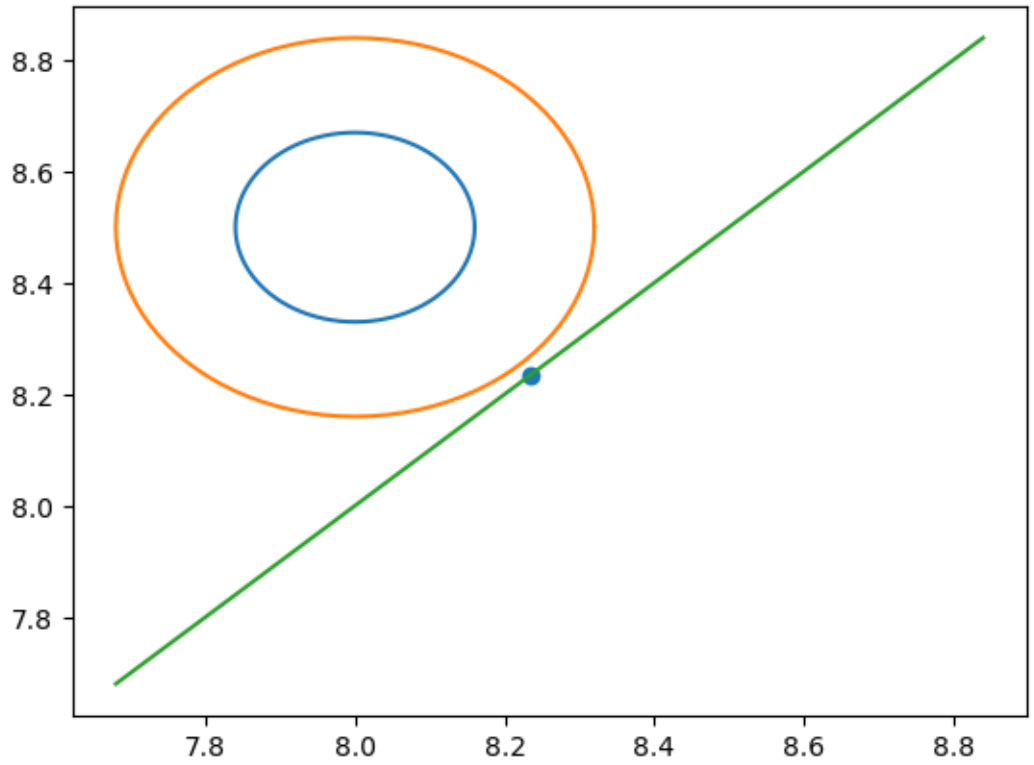
	Name	Value	Hesse Err	Minos Err-	Minos Err+	Limit-	Limit+	Fixed
0	par	8.2	0.8					
1	N	1.0	0.1					

	par	N
par	0.693	-0.083
N	-0.083	0.01

[22]:
```
drawCovEllipse(inv_cov, mw, res1.params["par"].value)
display(res1.params["par"])
drawCovEllipse(inv_cov, mw, res2.params["par"].value)
display(res2.params["par"])
```



	Name	Value	Hesse Err	Minos Err-	Minos Err+	Limit-	Limit+	Fixed
0	par	7.9	0.8					



	Name	Value	Hesse Err	Minos Err-	Minos Err+	Limit-	Limit+	Fixed
0	par	8.2	0.8					

- Construct a covariance matrix of y_1 and y_2 containing the normalisation uncertainty of 10,% relative to the average value $\bar{y}$. Solve the corresponding χ^2 minimisation with `iminuit` and plot the covariance ellipse.

[26]:
```
#cov = np.array([[(8.25*0.02)**2, 0],
#                 [0, (8.25*0.02)**2]])
#cov

def chi2_creator_4_2_4(measurement_vector, cov, sigma_norm):

    # Here we pass only the non-inverted covariance matrix as the inverted␣
 ↪matrix should depend on the normalisation. This means we have to do the␣
 ↪matrix inversion inside chi2_function(par).
    def chi2_function(par):
        #print(par, N)
```

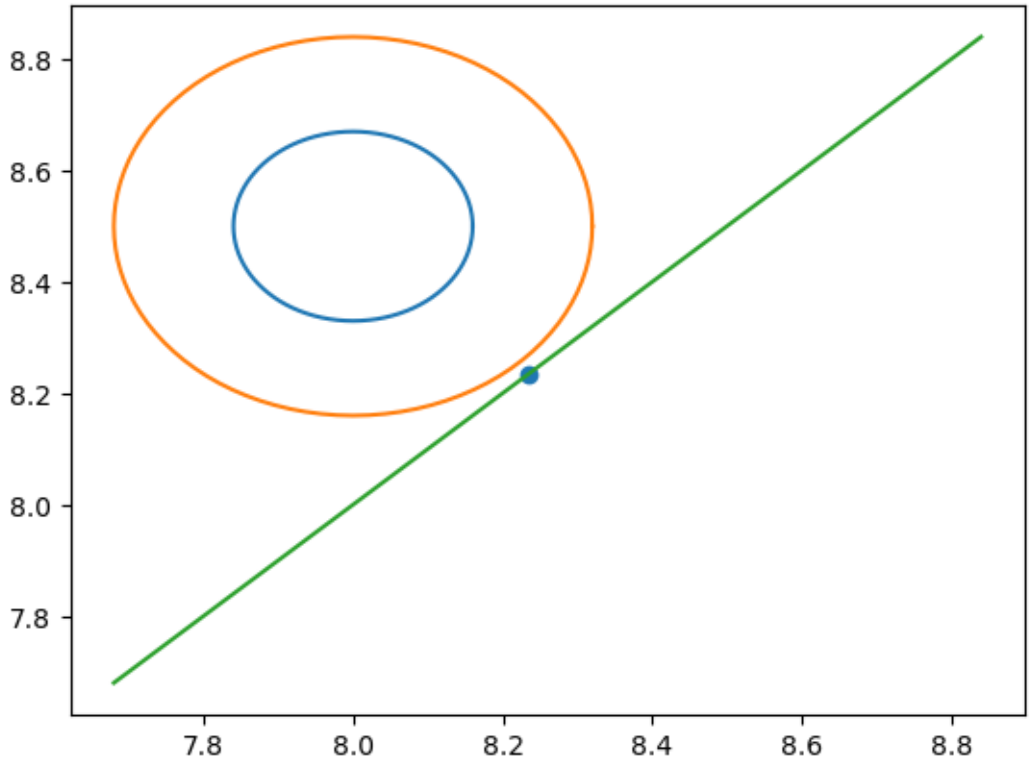
```
        # Hier ist wieder die Normalisierung mit N gemacht und die einzelnen␣
↪Unsicherheiten sind jetzt auch relativ zum Fit-Parameter.
        cov = np.array([[(8*0.02)**2+(par*0.1)**2, (par*0.1)**2],
                        [(par*0.1)**2, (8.5*0.02)**2+(par*0.1)**2]])
        inv_cov = np.linalg.inv(cov)


        '''
        calculate the chi2 including an additional parameter for the␣
↪normalization of the mean value (version 2)
        '''
        chi2_value = np.sum((measurement_vector - par)*np.
↪dot(inv_cov,measurement_vector - par))

        return chi2_value # return the chi2 value

    return chi2_function

minuit_instance = Minuit(chi2_creator_4_2_4(mw, inv_cov, 0.1), par=8)
res = minuit_instance.migrad()
display(res)
print(res2.params["par"].value, res2.params["N"].value)
drawCovEllipse(inv_cov, mw, res2.params["par"].value)
```

Migrad	
FCN = 4.587 EDM = 7.85e-05 (Goal: 0.0002)	Nfcn = 12
Valid Minimum	Below EDM threshold (goal x 10)
No parameters at limit	Below call limit
Hesse ok	Covariance accurate

	Name	Value	Hesse Err	Minos Err-	Minos Err+	Limit-	Limit+	Fixed
0	par	8.2	0.8					

	par
par	0.688

```
8.233675329754469 1.0001379297899027
```



[24]:
```
a = np.array([[1,2]])
b = np.array([[7,8]])

a*b.T
```

[24]:
```
array([[ 7, 14],
       [ 8, 16]])
```

[ ]:

[ ]:

[ ]: