blatt08_nitschke_grisard

December 13, 2018

1 Blatt 8

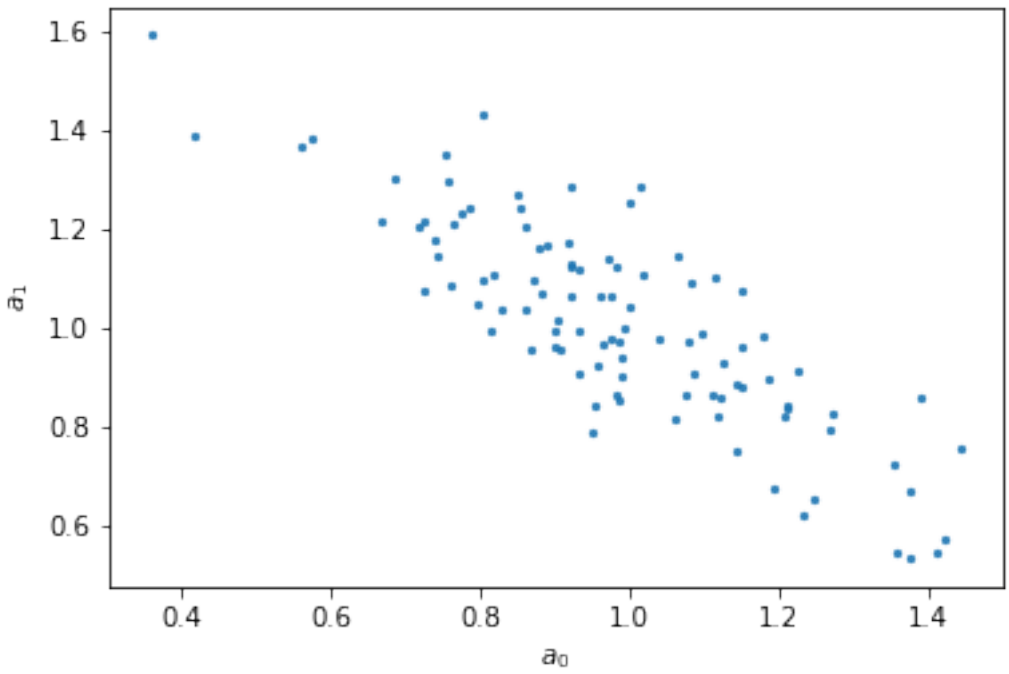
1.1 Aufgabe 22: Fehlerfortpflanzung

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import uncertainties.unumpy as unp
from uncertainties import ufloat
```

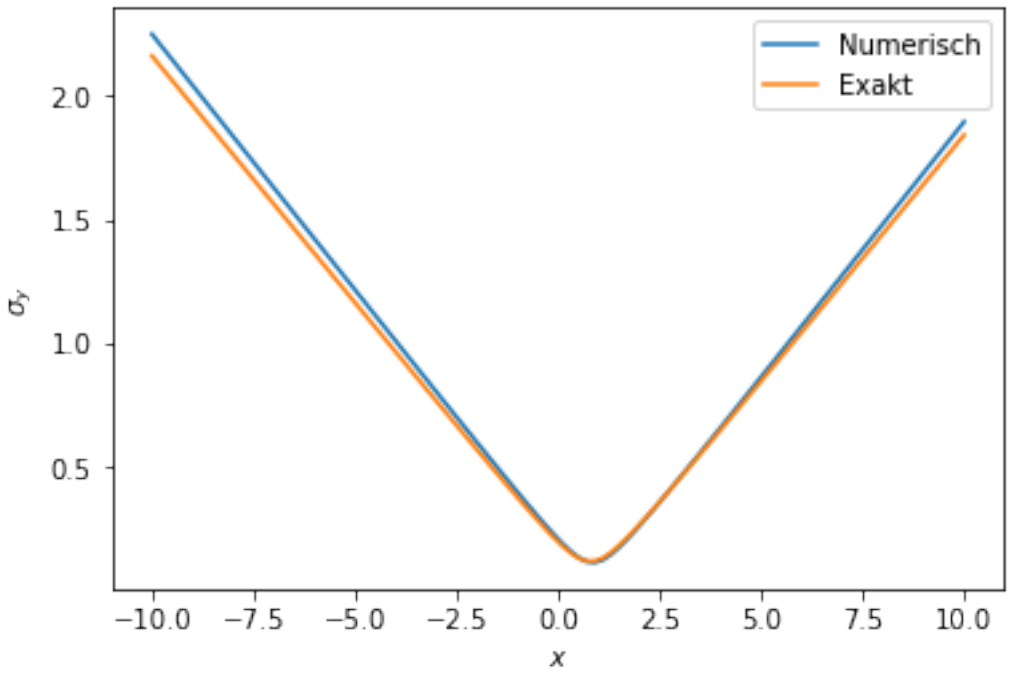
In [2]:
```
sig0 =  0.2
sig1 =  0.2
rho  = -0.8
cov = [[sig0**2, rho * sig0 * sig1], [rho * sig0 * sig1, sig1**2]]

a = np.random.multivariate_normal(mean = [1, 1], cov = cov, size = 100)
```

In [3]:
```
plt.scatter(a[:, 0], a[:, 1], alpha = 0.9, s = 5)
plt.xlabel('$a_0$')
plt.ylabel('$a_1$')
```

Out[3]:
```
Text(0,0.5,'$a_1$')
```




```
In [4]: x = np.linspace(-10, 10, 100)
        stdy = [np.std(a[:, 0] + a[:, 1] * x_i) for x_i in x]
        plt.plot(x, stdy, label = 'Numerisch' )

        def exact(x):
            return np.sqrt(sig0**2 + x**2 * sig1**2 + 2 * rho * sig0 * sig1 * x)
        plt.plot(x, exact(x), label = 'Exakt')
        plt.xlabel('$x$')
        plt.ylabel('$\sigma_y$')
        plt.legend()
        plt.show()
```




```
In [5]: x = [-3, 0, 3]
        for x_i in x:
            y = a[:, 0] + a[:, 1] * x_i
            print(f'x = {x_i}:\t y = {np.mean(y)} +/- {np.std(y)}')
```

```
x = -3:         y = -2.055500218300318 +/- 0.8068818144837163
x = 0:         y = 0.9836304809349856 +/- 0.21175884014833143
x = 3:         y = 4.022761180170289 +/- 0.4588214095750574
```

Kommentar: Die numerischen Ergebnisse stimmen recht gut mit den exakt bestimmten Werten überein (siehe Handschriftliches). Die Präzision nimmt für groSSe Beträge von x ab (siehe Plot oben). Durch eine VergröSSerung des Samples könnte die Präzision gesteigert werden.

1.2 Aufgabe 24: F(uckin')-Praktikum

```
In [6]: Asymmetrie = np.matrix([-0.032, 0.01, 0.057,
                               0.068, 0.076, 0.08,
                               0.031, 0.005, -0.041,
                               -0.09, -0.088, -0.074])
        Psi = np.arange(0, 331, 30)
```

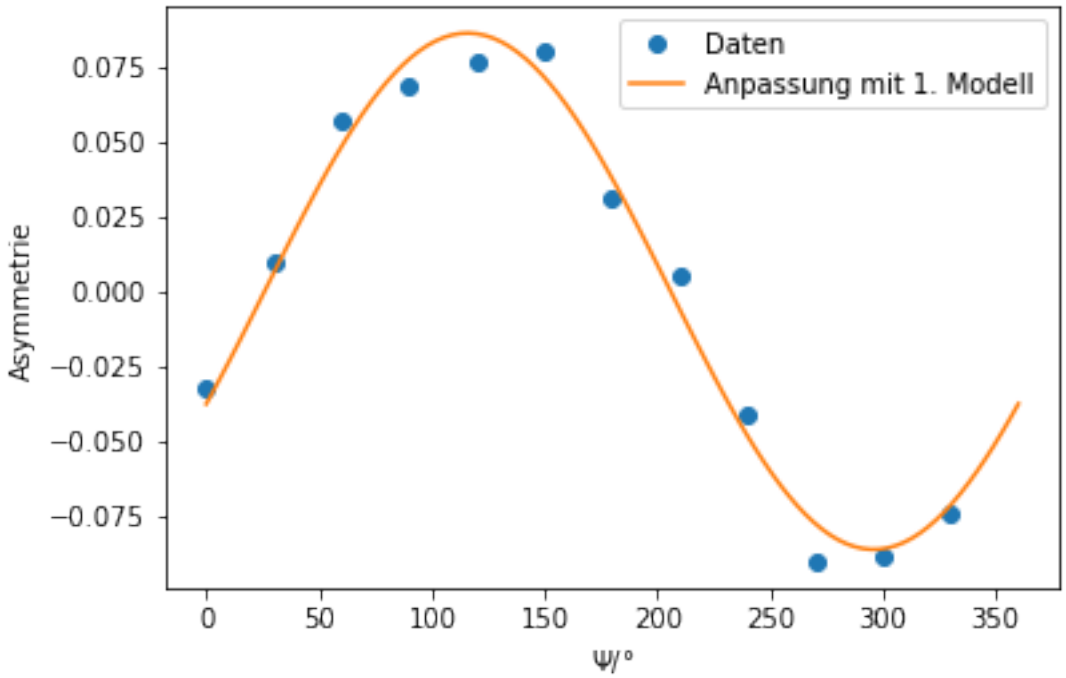
```
In [7]: A = np.matrix([np.cos(Psi / 180 * np.pi), np.sin(Psi / 180 * np.pi)]).T
```

```
In [8]: a = (A.T * A).I * A.T * Asymmetrie.T
        a
```

```
Out[8]: matrix([[-0.0375063 ],
                [ 0.07739978]])
```

```
In [9]: Psi_plot = np.linspace(0, 360, 100)
        plt.plot(Psi, Asymmetrie.T, 'o', label = 'Daten')
        plt.xlabel('$\Psi / ř$')
        plt.ylabel('Asymmetrie')
        plt.plot(Psi_plot,
                 a[0, 0] * np.cos(Psi_plot / 180 * np.pi) + a[1,0] * np.sin(Psi_plot / 180 * np
                 label = 'Anpassung mit 1. Modell')
        plt.legend()
```

```
Out[9]: <matplotlib.legend.Legend at 0x1f69b997c18>
```




```
In [10]: sigma = 0.011
         Cov = np.array(sigma**2 * (A.T * A).I)
```

```
In [11]: rho = Cov[0, 1] / np.sqrt(Cov[0, 0] * Cov[1, 1])
         print(f'a1 = {ufloat(a[0, 0], np.sqrt(Cov[0, 0]))}')
         print(f'a2 = {ufloat(a[1, 0], np.sqrt(Cov[1, 1]))}')
         print(f'Korrelationskoeffizient: {rho}')
```

```
a1 = -0.038+/-0.004
a2 = 0.077+/-0.004
Korrelationskoeffizient: -6.542726332681614e-17
```

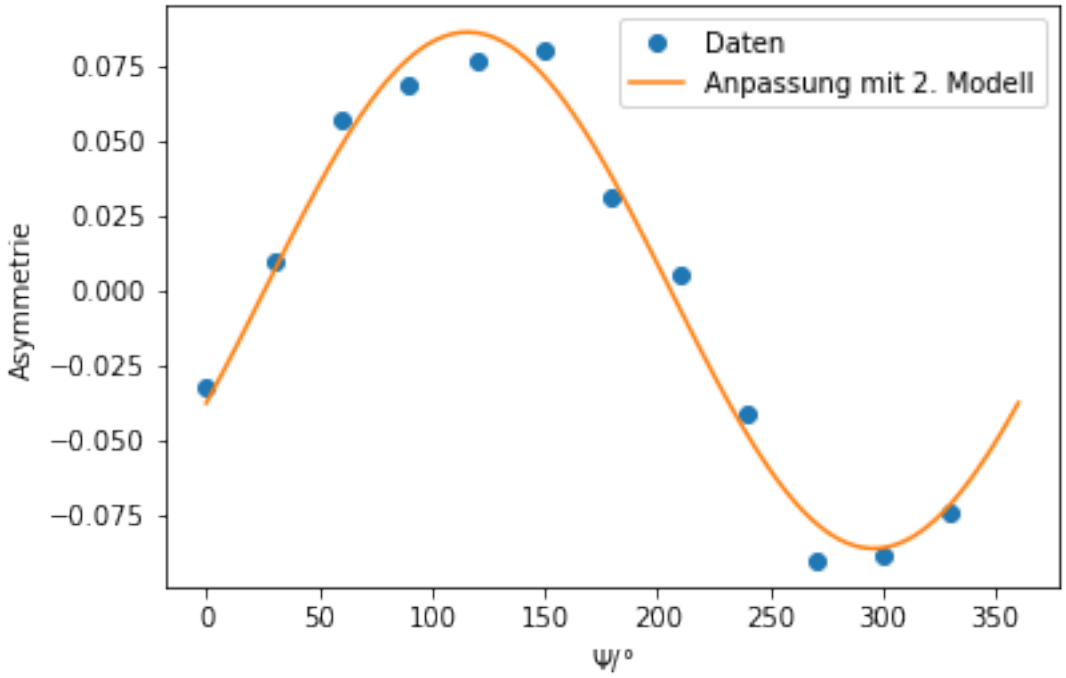
Die Korrelation ist fast 0. Das Modell ist also gut gewählt, was letztlich an der Orthogoalität der Funktionen Sinus und Kosinus liegt.

```
In [12]: delta = unp.arctan(- a[1, 0] / a[0, 0])
         A0 = a[0, 0] / unp.cos(unp.arctan(- a[1, 0] / a[0, 0]))

In [13]: a1 = a[0,0]
         a2 = a[1,0]

         J = np.matrix([[a1 / np.sqrt(a1**2 + a2**2), a2 / np.sqrt(a1**2 + a2**2)],
              [a2 / (a1**2 + a2**2), -1 / (a1 + a2**2 / a1)]])
         Cov2 = J * Cov * J.T
         print(f'A_0 = {ufloat(A0, np.sqrt(Cov2[0, 0]))}')
         print(f'delta = {ufloat(delta, np.sqrt(Cov2[1, 1]))}')
         print(f'Korrelation: {Cov2[1, 0] / np.sqrt(Cov2[0, 0] * Cov2[1, 1]) }')
```

```
A_0 = -0.086+/-0.004
delta = 1.12+/-0.05
Korrelation: -1.155998080162034e-16
```

Die Korrelation ist ebenfalls sehr klein aber etwa 10x grösSer als bei dem anderen Modell.

```
In [14]: Psi_plot = np.linspace(0, 360, 100)
         plt.plot(Psi, Asymmetrie.T, 'o', label = 'Daten')
         plt.xlabel('$\Psi / ř$')
         plt.ylabel('Asymmetrie')
         plt.plot(Psi_plot, A0 * unp.cos(Psi_plot / 180 * np.pi + delta), label = 'Anpassung m
         plt.legend()

Out[14]: <matplotlib.legend.Legend at 0x1f69be37278>
```


Daten
Anpassung mit 2. Modell
Asymmetrie
0.075
0.050
0.025
0.000
−0.025
−0.050
−0.075
0
50
100
150
200
250
300
350
Ψ/°