Praktikum 1: Repetition Wahrscheinlichkeit

Das Ziel dieser Übungen ist es, sich mit Wahrscheinlichkeitsverteilungen vertraut zu machen, Stichproben zu generieren und Resultate zu plotten. Wir werden folgende `python`-Packages verwenden:

- `numpy` für Array-Manipulationen und Berechnungen https://numpy.org/
- `scipy.stats` für Wahrscheinlichkeitsverteilungen https://docs.scipy.org/doc/scipy/tutorial/stats.html
- `matplotlib` für das Generien von Plots

Übung 1. `Scipy.stats` bietet diverse statistische Werkzeuge an, insbesondere verschiedene Wahrscheinlichkeitsverteilungen (siehe https://docs.scipy.org/doc/scipy/reference/stats.html#statsrefmanual).
Für diese Übung werden wir die Datei `Ex_1_Probability_Distributions_skeleton.py` benutzen. Für diverse Wahrscheinlichkeitsverteilungen existiert jeweils eine Funktion, welche Stichproben generiert und dazu verschiedene Plots erstellt. Die Code-Stellen, welche mit `#Code here` gekennzeichnet sind, müssen von Ihnen vervollständigt werden.

(a) Betrachten Sie die Funktion `random_sample_bernoulli` und ändern Sie die Paremeter `p` und `n_throws`.

(b) Vervollständigen Sie den Code der Funktion `random_sample_binom` welche die Parameter `n` und `p` der Binomialverteilung und `sample_size` als Inputparameter hat. Die Funktion soll die Stichproben ausgeben und eine Balkendiagramm mit der Anzahl der Vorkommnissen der verschiedenen Werte $\{0, 1, \ldots, n\}$ darstellen. Zusätzlich sollten Sie die Daten normalisieren und das normalisierte Balkendiagramm mit der Wahrscheinlichkeitsdichte (PDF) der Binomialverteilung vergleichen.

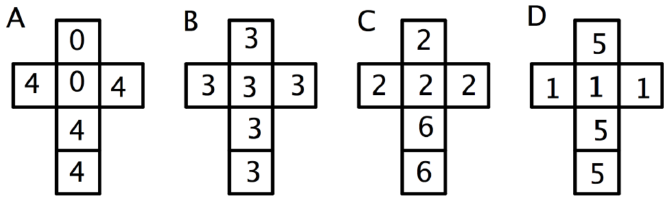
(c) Vervollständigen Sie die Funktion `random_sample_normal`, welche die Parameter `mean`, `std_dev` und `sample_size` als Inputparameter hat. Sie sollte ein Histogram der Stichproben einer Normalverteilung mit den entsprechenden Parametern darstellen (benutzen Sie `plt.hist`). Erstellen Sie einen Histogram Plot mit normalisierten Stichproben und vergelichen Sie ihn mit der Wahrscheinlichkeitsdichte (PDF). Zusätzlich können Sie der Funktion einen Paramter namens `n_bins` hinzufügen, welcher die Anzahl der Klassen des Histograms (auf Englisch "bins") kontrolliert.

(d) In diesem Teil werden wir die Beta-Verteilung $\text{Beta}_{a,b}(x)$ benutzen, welche von den Parameter $a > 0, b > 0$ abhängt und für $0 \leq x \leq 1$ definiert ist. Die Beta-Verteilung ist eine asymmetrische Verteilung, insbesondere unterscheiden sich dabei der Durchschnitt und der Median. Betrachten Sie die Funktion `plot_beta_pdf` und untersuchen Sie sie für die Paare $(a, b) \in \{(0.5, 0.5), (5, 1), (1.5, 3), (2, 2), (2, 5)\}$. Vervollständigen Sie die Funktion `plot_beta_mean_median` mit die Parameter `a`, `b`, welche der Beta-Verteilung entsprechen. Sie sollte den dazugehörigen Durchschnitt und Median berechnen und diese neben der PDF darstellen. Sie können vertikale Linen (`vlines`) benutzen, um den Durchschnitt und den Median darzustellen.

Übung 2. Wir sind daran gewöhnt, dass sich Eigenschaften transitiv verhalten:

- Wenn Alice stärker ist als Bob und Bob stärker ist als Carole, wird Alice auch stärker sein als Carole.
- Wenn $a > b$ und $b > c$, dann gilt auch $a > c$.

Bradley Efron (*1938) beschrieb einen Satz an Würfeln, welche sich nicht transitiv verhalten. Die Netze der Würfel sehen wie folgt aus:



Offensichtlich ist Würfel A besser als Würfel B, da er mit der einer Wahrscheinlichkeit von $\frac{4}{6}$ gewinnt. Gleichermassen kann man errechnen, dass Würfel B besser als Würfel C, Würfel C besser als Würfel D und Würfel D besser als Würfel A ist. Dementsprechend sind die Würfel nicht transitiv. Das führt dazu, dass - falls ihre gegnerische Person ihren Würfel zuerst wählt - Sie immer einen Würfel wählen können, welcher eine höhere Gewinnwarscheinlichkeit aufweist.
Benutzen Sie in dieser Übung die Datei `Ex_2_Efron_Dice_skeleton.py` und vervollständigen Sie den Code an der Stelle, welche mit `#Code here` gekennzeichnet ist

(a) Mit der Funktion `rv_discrete` aus `scipy.stats` können Sie ihre eigene diskrete Wahrscheinlichkeitsverteilung generieren, indem Sie die Ergebnisse und die dazugehörigen Wahrscheinlichkeiten definieren. Danach können Sie mit der Funktion `rvs()` Stichproben generieren. Simulieren Sie 20 würfe mit dem Würfel A.

(b) Simulieren Sie 100 Spiele mit den Würfeln C und D, wobei ein Spiel aus 100 Würfen mit jedem Würfel besteht. Vervollständigen Sie dazu die Funktion `dice_game`. Sie sollte zurückgeben, wie oft jeder Würfel gewonnen hat. Wie hoch ist die Wahrscheinlichkeit, mit Würfel C zu gewinnen, wenn ihre gegnerische Person Würfel D auswählt? Erhalten Sie mit der Simulation die gleiche Antwort wie aus der Theorie?

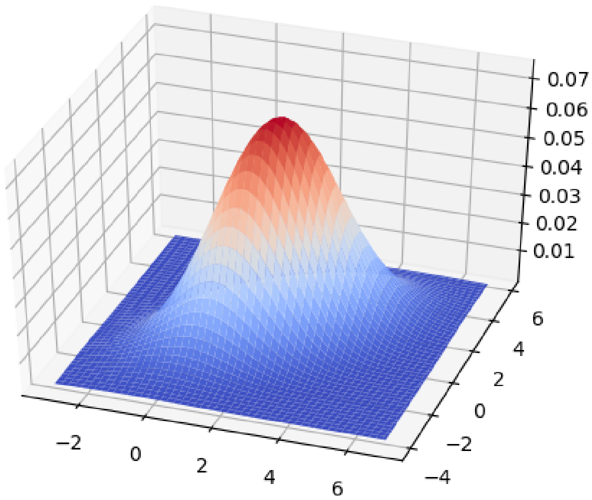
(c) Vervollständigen Sie die Funktion `compute_sample_mean_var`, welche den Durchschnitt $\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ und die Varianz $\overline{s}^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \overline{x})^2$ einer Reihe an Würfen berechnet. Testen Sie die Funktion mit dem Würfel D. Finden Sie ausserdem einen Weg, den theoretischen Durschnitt und die theoretische Varianz direkt zu berechnen (benutzen Sie dazu `scipy.stats`).

(d) Untersuchen Sie die Funktion `plot_mean_var` und benutzen Sie sie, um die Entwicklung des Durchschnitts und der Varianz des Würfels D darzustellen.

Übung 3. Die n-dimensionale, multivariate Normal-Verteilung ist definiert durch

$$f_{\boldsymbol{\mu},\Sigma}(\boldsymbol{x}) = \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu},\Sigma) = \frac{1}{(2\pi)^{\frac{n}{2}}} \cdot \frac{1}{\det(\Sigma)^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T \Sigma^{-1} (\boldsymbol{x}-\boldsymbol{\mu})\right)$$

wobei $\boldsymbol{x} \in \mathbb{R}^n$, $\boldsymbol{\mu} \in \mathbb{R}^n$ der Durchschnittsvektor und $\Sigma \in \mathbb{R}^{n \times n}$ die positiv definite, symmetrische Kovarianzmatrix darstellen. Hier sehen Sie die PDF der multivariaten Normal-Verteilung mit $\boldsymbol{\mu} = (2,1)^T$ und $\Sigma = \begin{pmatrix} 2.5 & 1.3 \\ 1.3 & 2.5 \end{pmatrix}$:



Arbeiten Sie mit der Datei `Ex_3_Multivariate_Normal_Distribution_skeleton.py` und vervollständigen Sie den Code an der Stelle, welche mit `#Code here` gekennzeichnet ist.

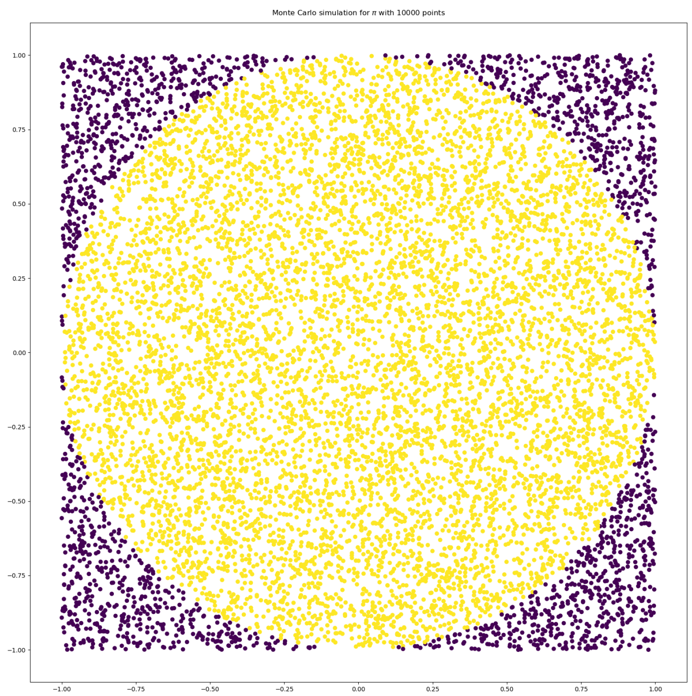
(a) Untersuchen Sie die Funktionen `multivariate_normal_computation` und `multivariate_normal_plot`. Benutzen Sie `multivariate_normal_plot`, um die PDF für Σ

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}, \begin{pmatrix} 3 & 1 \\ 1 & 3 \end{pmatrix}, \begin{pmatrix} 3 & 2.5 \\ 2.5 & 3 \end{pmatrix}, \begin{pmatrix} 0.8 & 0.5 \\ 0.5 & 0.8 \end{pmatrix}$$

darzustellen und beobachten Sie, wie sich der Oberflächenplot verändert. Sie müssen dazu die Definition der Kovarianzmatrizen vervollständigen.

(b) Vervollständigen Sie die Funktion `multivariate_contour_plot`. Nutzen Sie sie danach, um einen Konturenplot der Oberfläche von (a) darzustellen.

Übung 4. Monte Carlo Simulationen nutzen Zufall, um eine Quantität zu berechnen (z.B. ein Integral). Um die Idee zu verstehen, werden wir π mithilfe einer Monte Carlo Simulation berechnen. Die Idee lautet wie folgt: In dem Quadrat $[-1,1] \times [-1,1]$ werden zufällige Punkte generiert. Bei jedem Punkt überprüfen wir, ob sich dieser innerhalb des Einheitskreises befindet. Das Verhältnis der Anzahl Punkte innerhalb und ausserhalb des Kreises ist ungefähr $q \approx \frac{A_{\text{circle}}}{A_{\text{square}}} = \frac{\pi}{4}$, daher gilt $\pi \approx 4 \cdot q$.



Arbeiten Sie mit der Datei `Ex_4_Monte_Carlo_Simulation_skeleton.py` und vervollständigen Sie den Code an der Stelle, welche mit `#Code here` gekennzeichnet ist.

(a) Vervollständigen Sie die Funktion `monte_carlo_pi` um π zu approximieren. Nutzen Sie eine gleichmässige Verteilung, um die Punkte in dem Quadrat $[-1,1] \times [-1,1]$ zu generieren.

(b) Stellen Sie einen Plot mit den generierten Punkten her und heben Sie diejenigen, welche sich innerhalb des Einheitskreises befinden, farblich hervor.