9/9

Uebung02_Aufgabe3_lebron_221103_01

November 14, 2022

<div style="float:left;width:50%">Albert-Ludwigs-Universität Freiburg</div>
<div style="float:left;width:50%;text-align:right">Wintersemester 2022/23</div>

<h1 style="margin-top:20px;padding:0px">Datenanalyse für Naturwissenschaftler*Innen</h1>
<h2 style="margin:5px;padding:0px">Statistische Methoden in Theorie und Praxis</h2>
Vorlesung: Dr. Andrea Knue
Übungsleitung: Dr. Constantin Heidegger
<h1 style="margin:10px;padding:0px">Übung 2</h1>
Ausgabe: 28. Oktober 2022 10:00 Uhr, Abgabe: 4. November 2022 bis 10:00 Uhr via Ilias

Aufgabe 3: Histogramme, Graphen und ein Spiel (9P)

a) Histogramme und Schleifen (2P)

Als erstes üben und wiederholen wir Schleifen in Python und die Definition von Histogrammen. Binden Sie dazu als erstes `matplotlib` ein, die Standardbibliothek zum Plotten in Python.

[1]:
```python
import matplotlib.pyplot as plt
import numpy as np
```

Lassen Sie die ganzen Zahlen von 0 bis 15 am Bildschirm mittels einer `for`-Schleife ausgeben und berechnen Sie die Summe.

[2]:
```python
result = 0
for i in range(0,16):
    print(i)
    result += i
print(result)
```

```
0
1
2
3
4
5
6
7
8
9
10
11
12
```

```
13
14
15
120
```

Lassen Sie sich nur die ungeraden Zahlen von 0 bis 15 ausgeben. Wenn die Zahl 3 erreicht wird, soll zudem ein zusätzlicher Kommentar auf dem Bildschirm ausgegeben werden (z.B. “Ich bin bei der drei!”).

[3]:
```
result = 0
for i in range(0,16):
    if i % 2 != 0:
        if i == 3:
            print(i, "Ich bin bei der drei!")
        else:
            print(i)
```

```
1
3 Ich bin bei der drei!
5
7
9
11
13
15
```

Füllen Sie die ungeraden Zahlen in eine Liste.

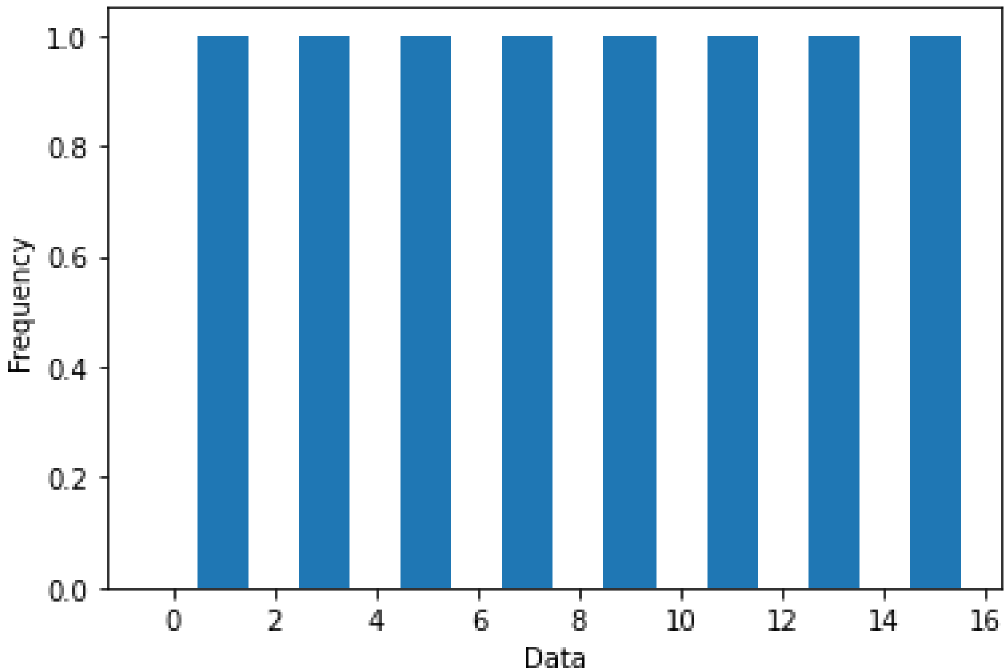
[4]:
```
l = [i for i in range(0,16) if i % 2 != 0]
l
```

[4]:
```
[1, 3, 5, 7, 9, 11, 13, 15]
```

Erstellen Sie ein Histogramm mit 16 Bins zwischen $-0{,}5$ und 15,5, füllen Sie alle ungerade Zahlen ein und zeichnen es auf dem Bildschirm. Warum wurden hier halbzahlige Grenzen gewählt?

[5]:
```
plt.hist(l, bins=16, range=[-0.5,15.5])
plt.ylabel('Frequency')
plt.xlabel('Data')

# Die Anzahl von Elemente einer Liste zwischen -0,5 und 15,5 beträgt 16.
```

[5]:
```
Text(0.5, 0, 'Data')
```



b) Funktionen und Diagramme (2P)

Definieren Sie eine Funktion `func` die für eine Variable x das Resultat $x^2 + 4$ zurückgibt.

[6]:
```
def func(x):
    return x**2+4
```

Kreieren Sie nun eine Liste von -5 bis $+5$ in Schritten von 0.5

[7]:
```
l = [i for i in np.arange(-5, 5.5, 0.5)]
l
```

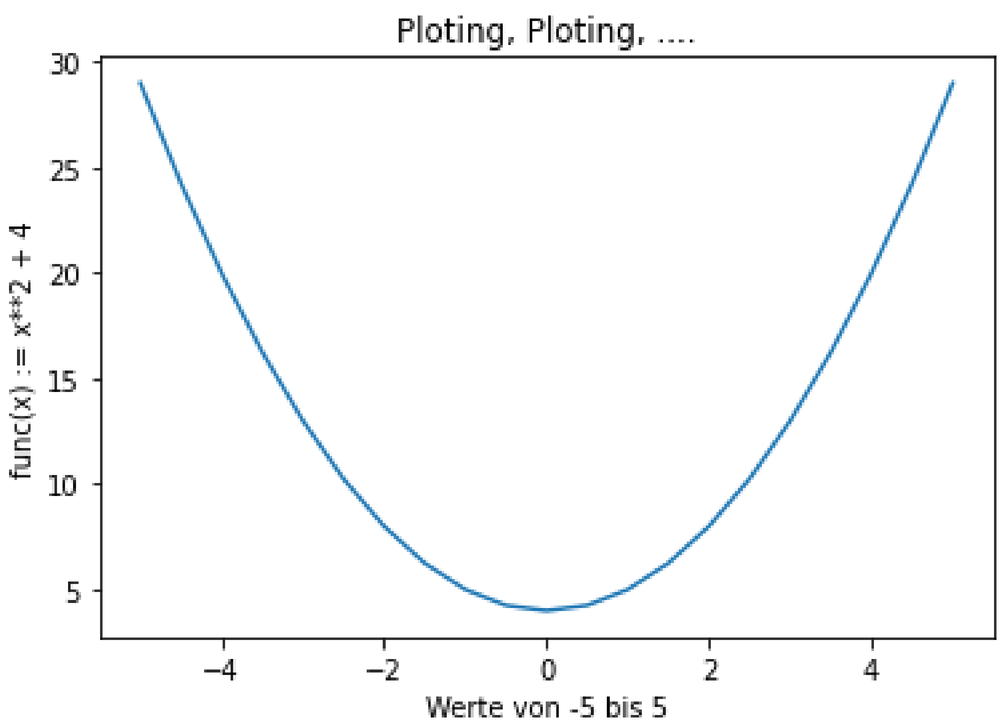
[7]:
```
[-5.0,
 -4.5,
 -4.0,
 -3.5,
 -3.0,
 -2.5,
 -2.0,
 -1.5,
 -1.0,
 -0.5,
 0.0,
 0.5,
 1.0,
 1.5,
```

```
 2.0,
 2.5,
 3.0,
 3.5,
 4.0,
 4.5,
 5.0]
```

Plotten Sie nun die Funktion in einem Diagramm. Verwenden Sie dazu die Liste, die Sie gerade erzeugt haben. Fügen Sie eine sinnvolle Achsenbezeichnung bei.

[8]:
```
ll = [func(i) for i in l]
plt.plot(l, ll)
plt.title("Ploting, Ploting, ....")
plt.xlabel("Werte von -5 bis 5")
plt.ylabel("func(x) := x**2 + 4")
```

[8]:
```
Text(0, 0.5, 'func(x) := x**2 + 4')
```



c) Wechseln oder nicht - simuliert! (5P)

Erinnern wir uns an das letzte Übungsblatt.

In einer Spielshow gibt es drei Türen. Hinter einer Tür wartet ein Gewinn, hinter den anderen Türen befinden sich Nieten. Nachdem eine der Kandidat*Innen eine Tür gewählt hat, wird eine der anderen Türen geöffnet, hinter der sich eine Niete befindet. Nun darf erneut eine der übrigen Türen gewählt werden.

Es gibt verschiedene Ansätze, wie man das Spiel simuliert. Da das Spiel mehrmals durchgeführt wird, soll das Problem erstmal in kleinere Abschnitte eingeteilt werden.

Die Türen sollen durch die Zahlen 0, 1 und 2 dargestellt werden. Um Zufallszahlen zu ziehen verwenden wir die `randrange` Funktion von `random` Package, was wir in nachfolgender Weise einbetten:

[9]:
```
from random import randrange
Preis = randrange(0,3)
```

Beachten Sie, dass `randrange` der Standard-Funktion `range` nachempfunden ist, z.B. `range(4)=0,1,2,3`.

i) Schreiben Sie zunächst eine Funktion `auswahl()`, die eine zufällig ausgewählte Tür zurückgeben soll (eine Zahl von 0-2), wobei eine (oder mehrere) Türen von der Ziehung ausgeschlossen werden können sollen. Die Anforderungen an die Funktion sind daher:

- Ein Liste von nicht-erlaubten Zahlen `nichterlaubt` und der Bereich `bereich`, der als Parameter für `randrange` übergeben werden soll.
- Eine Zufallszahl soll solange gezogen werden, bis die gezogene Zahl nicht in der Liste ist. (Oder anders formuliert: Zunächst wird eine Zufallszahl gewählt. Falls die Zufallszahl in der Liste der nicht-erlaubten Zahlen ist, soll eine neue Zufallszahl gewählt werden.)

[10]:
```
def auswahl(nichterlaubt = set(), bereich = 0):
    done = False
    while not done:
        aw = randrange(bereich)
        if aw not in nichterlaubt:
            done = True
            return aw
        else:
            continue
```

[11]:
```
# Alternative (verwendet in spiel() sowie spielen()!)
def auswahl02(nichterlaubt:set = set(), bereich: int = 0):
    """
    returns: random element from a range bounded by a value
    Parameters:
        # 1. nichterlaubt: set
        # 2. bereich: int
    """
    aw = randrange(bereich)
    while aw in nichterlaubt:
        aw = randrange(bereich)
    return aw
```

ii) Nun kann das Spiel simuliert werden. Dazu eignet sich wieder eine Funktion, in der man angeben kann, welche Strategie verfolgt werden soll (0=bleibt bei seiner Wahl, 1=wechselt immer). Praktischerweise soll die Funktion noch zwei weitere Parameter haben: einen Zufallsgenerator `zufallsgen` und einen Parameter `verbose`, der angeben soll, ob Meldungen ausgegeben werden sollen, oder nicht. Der default Wert soll `True` sein für `verbose`.

- Das Spiel beginnt damit, dass zunächst eine Gewinn-Tür `gewinnTuer` ausgewählt wird.
- Dann wird eine Tür durch die/den Spieler/in in der Variable `spielerinTuer` ausgewählt.
- Der/Die Moderator/in wählt eine Tür, die geöffnet wird. Diese wird in der Variable `moderatorinTuer` gespeichert. Natürlich muss man die Gewinn-Tür und die Spieler/in-Tür von der Auswahl ausschliessen.
- Je nach Strategie passiert nichts oder die/der Spieler/in wählt eine neue Tür, die jetzige Spieler/in-Tür und die Moderator/in-Tür wird ausgeschlossen. (Hinweis: `if`, `else`.)
- Am Ende soll überprüft werden, ob die aktuelle Spieler/in-Tür der Gewinner-Tür entspricht. Es soll dann `True` oder `False` zurückgegeben werden.

Damit nachvollziehbar ist, was passiert, soll nach wichtigen Schritten eine kurze Bildschirmausgabe folgen. Diese benutzen dann das Argument `verbose`:

```
if verbose:
    print("Jetzt passiert hier was")
```

[13]:
```
def spiel(strategie:int=0, bereich:int=3, verbose:bool=True):
    """
    returns: True or False

    Parameters:
        # 1. strategie: int
            Mit Wechsel (MW = 1) oder ohne (OW = 0) beim 2. Chance nach Öffnung␣
↪der Tür
            OW (default)
            MW (optional)
        # 2. bereich: int
            Anzahl der Tür-Optionen
        # 3. verbose: bool
            Erläuterung der Schritte urch ausgegebene Strings
    """
    # For now just default value
    nichterlaubt = set()
    gewinnTuer = auswahl02(nichterlaubt, bereich)
    if verbose == True:
        print(f"Tür {gewinnTuer} als Gewinntür ausgewählt")
        nichterlaubt.add(gewinnTuer)

    # Spielerin-Tür
    spielerinTuer = auswahl02({}, bereich)
    if verbose == True:
        print(f"Tür {spielerinTuer} als Spielerintür ausgewählt")
    nichterlaubt.add(spielerinTuer)

    # geöffnete Tür
```

```python
    moderatorinTuer = auswahl02(nichterlaubt, bereich)
    if verbose == True:
        print(f"Tür {moderatorinTuer} als Moderatorintür ausgewählt")
    nichterlaubt.add(moderatorinTuer)

    if strategie == 1:
        if gewinnTuer in nichterlaubt:
            nichterlaubt.remove(gewinnTuer)
        spielerinTuer = auswahl02(nichterlaubt, bereich)
        if verbose == True:
            print(f"Spielerin wechselt zu Tür {spielerinTuer}")

    if spielerinTuer == gewinnTuer:
        if verbose == True:
            print(
                    f"Gewinn-Tür {gewinnTuer} == Spielerin-Tür {spielerinTuer}␣
↪Spielerin gewann.")
            return True
        elif verbose == False:
            return True
    elif spielerinTuer != gewinnTuer:
        if verbose == True:
            print(f"Gewinn-Tür {gewinnTuer} != Spielerin-Tür {spielerinTuer} ->␣
↪Spielerin verlor.")
            return False
        elif verbose == False:
            return False
    #nichterlaubt.clear()


spiel(1, 3, False)
```

[13]: True

iii) Jetzt soll das Spiel mehrfach ausgeführt werden und der Anteil der gewonnenen Spiele gegen die Anzahl der Spiele in einem Diagramm aufgetragen werden. Praktischerweise kann man das wieder als Funktion programmieren. Eingangsparameter dieser Funktion sind: Die Anzahl der Spiele `anzahl`, eine Zahl für den Bereich `bereich`, eine Zahl für die Strategie `strategie` und wieder `verbose`. Es soll ein `TGraph` und der Anteil der gewonnenen Spiele zurückgegeben werden. Die Funktion soll folgendermassen funktionieren:

- Bauen Sie iene Schleife die `anzahl` Iterationen durchführt.
- Definieren Sie eine (float) Variable `gewinnAnteil` als laufenden Durchschnitt der gewonnenen Spiele.
- In der Schleife der Spiele führen Sie die Funktion `spiel()` aus und übergeben Sie die passenden Argumente. Denken Sie daran, dass Sie Ergebnis speichern müssen: `ergebnis=spiel()`
- Wenn Sie nun mit `int(ergebnis)` das Ergebnis in eine Zahl umwandeln, so ergibt sich `1` für

einen Gewinn, sonst erhalten Sie `0`.

- Der Durchschnitt lässt sich mit der Rekursionsformel bei einer Stichprobe der Grösse `N` berechnen:
$$\bar{x}_N = \bar{x}_{N-1} + \frac{x_n - \bar{x}_{N-1}}{N}$$
- Tragen Sie die aktuelle Anzahl der Spieliterationen und den aktuellen Gewinn-Anteil in zwei Listen ein, `pointsx` und `pointsy`.
- Aus diesen zwei Listen, erstellen Sie ein Diagramm und benennen Sie die x und y Achsen richtig.
- Geben Sie wieder einige Informationen nach jedem Spiel aus, wenn `verbose` gesetzt (=wahr) ist.
- Geben Sie den Plot und den Gewinn-Anteil zurück.

[14]:
```
def mean(a: list):
    """
    returns: running average (mean)
    Parameters:
        a: list
    """
    if len(a) == 1:
        return a[0]
    else:
        n = len(a)
        return (a[0] + (n - 1) * mean(a[1:])) / n
```

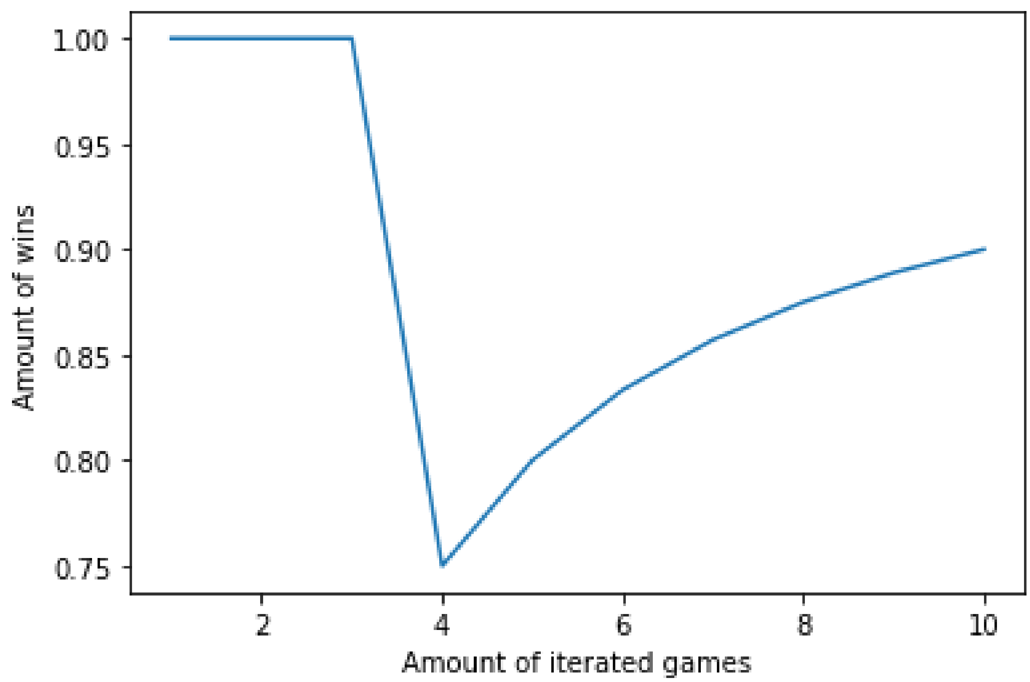
[15]:
```
def spielen(anzahl=10, bereich=3, strategie=1, verbose=True):
    """
    ## returns:
        * str with results
        * plot
    ## Parameters:
    ### 1. anzahl: int
            Anzahl von iterierten Spielen
    ### 2. bereich: int
            Obergrenze der Auswahlmöglichkeiten
    ### 3. strategie: int
        Mit Wechsel (MW = 1) oder ohne (OW = 0) beim 2. Chance nach Öffnung der␣
↪Tür
            OW (Optional)
            MW (defalt)
    ### 4. verbose: bool
        Erläuterung der Schritte durch ausgegebene Strings
    """
    GewinnAnteil = []
    Results = []
    Nr_Games = []
    Wins = 0
    for i in range(1,anzahl+1):
```

```
        Nr_Games.append(i)
        result = int(spiel(strategie,bereich,verbose))
        Wins += result
        Results.append(result)
        GewinnAnteil.append(mean(Results))

    anteil = f"Gewonnen: {Wins} aus {anzahl} Spiele bzw. {round((Wins/
↪anzahl)*100)} % gewonnen"
    p = plt.plot(Nr_Games,GewinnAnteil)
    plt.xlabel("Amount of iterated games"), plt.ylabel("Amount of wins")

    return anteil, p

spielen()
```

```
Tür 0 als Gewinntür ausgewählt
Tür 1 als Spielerintür ausgewählt
Tür 2 als Moderatorintür ausgewählt
Spielerin wechselt zu Tür 0
Gewinn-Tür 0 == Spielerin-Tür 0 Spielerin gewann.
Tür 2 als Gewinntür ausgewählt
Tür 0 als Spielerintür ausgewählt
Tür 1 als Moderatorintür ausgewählt
Spielerin wechselt zu Tür 2
Gewinn-Tür 2 == Spielerin-Tür 2 Spielerin gewann.
Tür 1 als Gewinntür ausgewählt
Tür 2 als Spielerintür ausgewählt
Tür 0 als Moderatorintür ausgewählt
Spielerin wechselt zu Tür 1
Gewinn-Tür 1 == Spielerin-Tür 1 Spielerin gewann.
Tür 0 als Gewinntür ausgewählt
Tür 0 als Spielerintür ausgewählt
Tür 2 als Moderatorintür ausgewählt
Spielerin wechselt zu Tür 1
Gewinn-Tür 0 != Spielerin-Tür 1 -> Spielerin verlor.
Tür 2 als Gewinntür ausgewählt
Tür 1 als Spielerintür ausgewählt
Tür 0 als Moderatorintür ausgewählt
Spielerin wechselt zu Tür 2
Gewinn-Tür 2 == Spielerin-Tür 2 Spielerin gewann.
Tür 1 als Gewinntür ausgewählt
Tür 2 als Spielerintür ausgewählt
Tür 0 als Moderatorintür ausgewählt
Spielerin wechselt zu Tür 1
Gewinn-Tür 1 == Spielerin-Tür 1 Spielerin gewann.
Tür 2 als Gewinntür ausgewählt
Tür 0 als Spielerintür ausgewählt
Tür 1 als Moderatorintür ausgewählt
```

```
Spielerin wechselt zu Tür 2
Gewinn-Tür 2 == Spielerin-Tür 2 Spielerin gewann.
Tür 2 als Gewinntür ausgewählt
Tür 1 als Spielerintür ausgewählt
Tür 0 als Moderatorintür ausgewählt
Spielerin wechselt zu Tür 2
Gewinn-Tür 2 == Spielerin-Tür 2 Spielerin gewann.
Tür 2 als Gewinntür ausgewählt
Tür 2 als Spielerintür ausgewählt
Tür 0 als Moderatorintür ausgewählt
Spielerin wechselt zu Tür 2
Gewinn-Tür 2 == Spielerin-Tür 2 Spielerin gewann.
Tür 2 als Gewinntür ausgewählt
Tür 2 als Spielerintür ausgewählt
Tür 0 als Moderatorintür ausgewählt
Spielerin wechselt zu Tür 2
Gewinn-Tür 2 == Spielerin-Tür 2 Spielerin gewann.
```

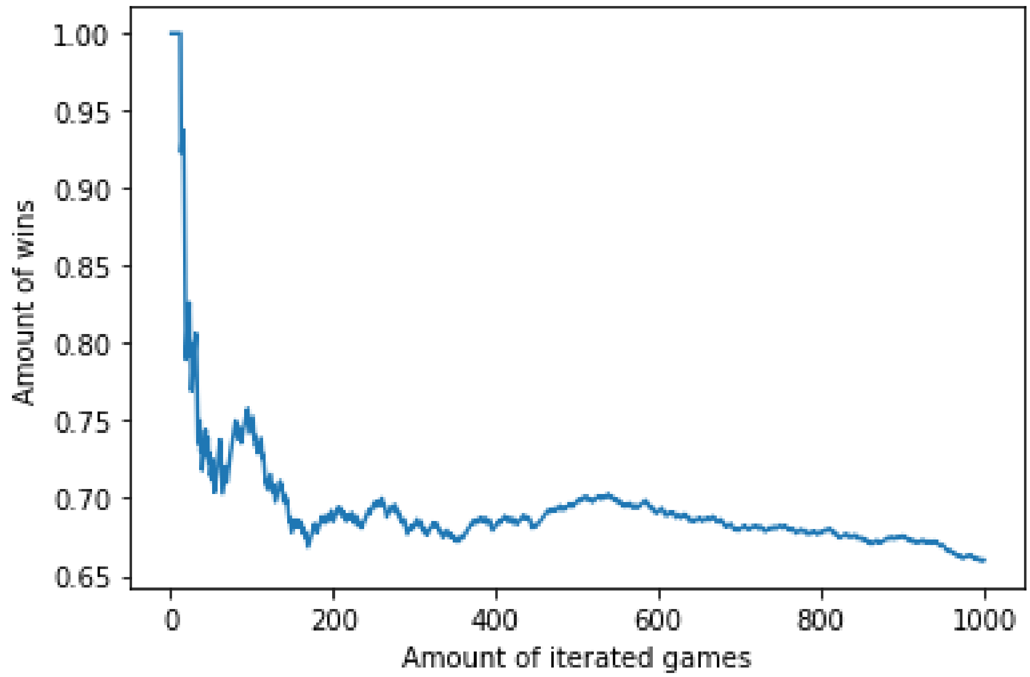
```
[15]: ('Gewonnen: 9 aus 10 Spiele bzw. 90 % gewonnen',
       [<matplotlib.lines.Line2D at 0x7f3fad015970>])
```



iv) Folgende Zeilen sollen als Beispiel dienen, wie eine Reihe von Spielen durch geführt werden soll und wie das Ergebnis dargestellt werden soll (Achtung, die Zeilen funktionieren nur, wenn Sie die obigen Funktionen korrekt definiert haben):

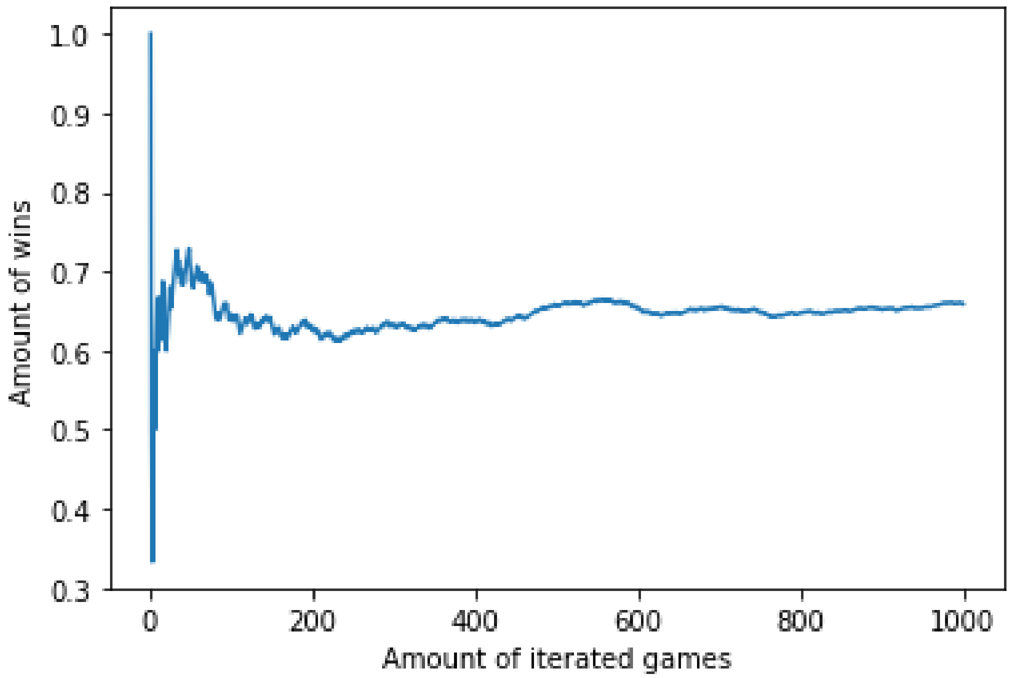
[16]:
```
spielen (1000, 3, 1, False)
```

[16]:
```
('Gewonnen: 660 aus 1000 Spiele bzw. 66 % gewonnen',
 [<matplotlib.lines.Line2D at 0x7f3facfe5610>])
```



[17]:
```
anteil, p = spielen (1000, 3, 1, False)
anteil
```

[17]:
```
'Gewonnen: 659 aus 1000 Spiele bzw. 66 % gewonnen'
```



v) Probieren Sie nun einige Spiele aus, bei der Sie die Anzahl der Spiele und die Strategie variieren. Vergessen Sie dabei nicht, `verbose=False` zu stellen.

- **Was beobachten Sie am Diagramm und am Durchschnitt, wenn Sie nur wenige Spiele durchführen? Können Sie zwischen den beiden Strategien unterscheiden?**
- **Was passiert, wenn Sie nun die Anzahl der Spiele groß wählen?**

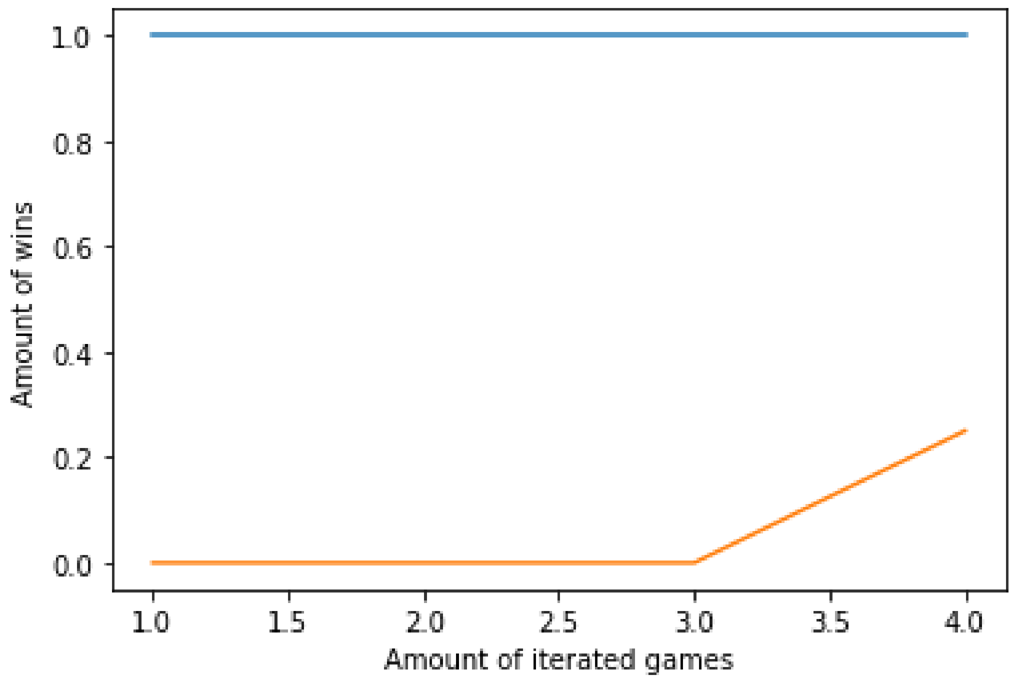
Schreiben Sie bitte die Antworten auf die obigen Fragen auch in das Notebook.

0.0.1 Antwort auf Aufgabe 3 b) v)

- Bei dem Diagramm mit einer kleinen Anzahl von Spiele:
  - die OW-Strategie immer hat immer wieder bis zum 50 % gewonnent.
  - der Unterschied zw. der OW- und der MW-Strategie nicht so deutlich ist.
- Bei dem Diagramm mit einer größen Anzahl von Spiele:
  - Je größer die Spielanzahl ist, desto auffälliger ist der Unterschied zw. der MW- und der OW-Strategie, weil der Gewinnanteil deutlich unterschiedlich wird.
  - Gewinnen durch die Anwendung der MW-Strategie ist mindestens doppelt so hoch als die Anwendung der OW-Strategie.

[18]:
```
## Wenige Spiele (n=4)
## MW-Strategie
print("MW-Strategie: ", spielen(4, 3, 1, False))
## OW-Strategie
print("OW-Strategie: ",spielen(4, 3, 0, False))
```

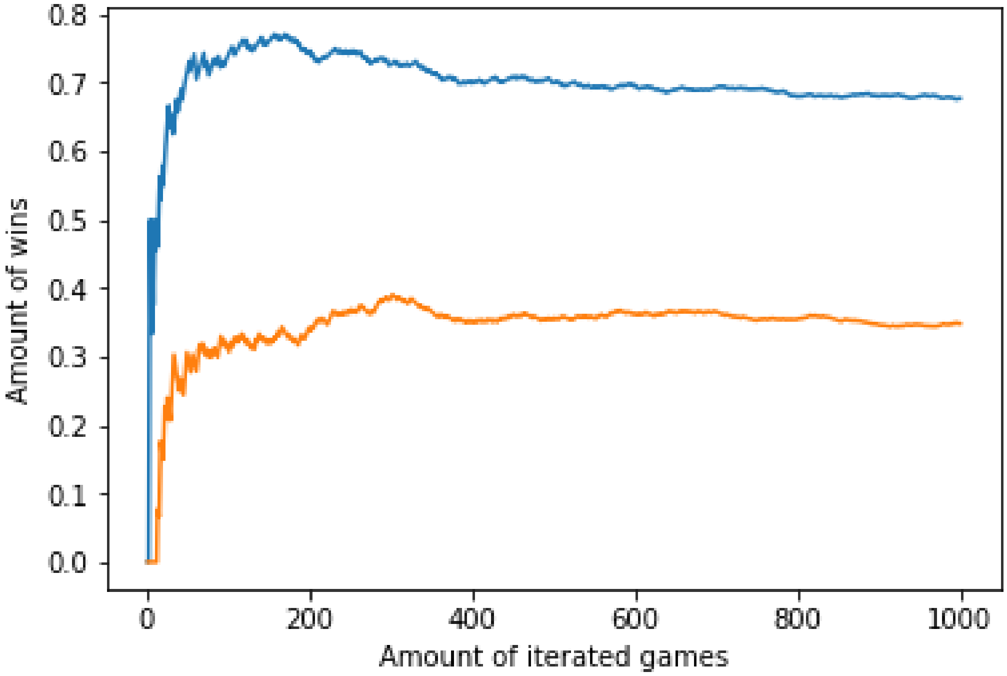
```
MW-Strategie:  ('Gewonnen: 4 aus 4 Spiele bzw. 100 % gewonnen',
[<matplotlib.lines.Line2D object at 0x7f3facedd0d0>])
OW-Strategie:  ('Gewonnen: 1 aus 4 Spiele bzw. 25 % gewonnen',
[<matplotlib.lines.Line2D object at 0x7f3facedd550>])
```



[19]:
```python
## Viele Spiele (n=100)
## MW-Strategie
print("MW-Strategie: ", spielen(1000, 3, 1, False))
## OW-Strategie
print("OW-Strategie: ",spielen(1000, 3, 0, False))
```

```
MW-Strategie:  ('Gewonnen: 677 aus 1000 Spiele bzw. 68 % gewonnen',
[<matplotlib.lines.Line2D object at 0x7f3face81430>])
OW-Strategie:  ('Gewonnen: 348 aus 1000 Spiele bzw. 35 % gewonnen',
[<matplotlib.lines.Line2D object at 0x7f3facee7430>])
```

nice!



0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
Amount of wins
0
200
400
600
800
1000
Amount of iterated games


```
[ ]:
```