12/20P.

SMD-Übungsblatt ?

Abgabe: ??.??.18

Yvonne Kasper yvonne.kasper@udo.edu ,
Robert Appel robert.appel@udo.edu ,
Julian Schröer julian.schroeer@udo.edu

1 Aufgabe1

a) Um x Zufallszahlen mit der Rückweisungsmethode zu ziehen, müssen zwei mal x Zahlen gezogen werden, welche einem x-Wert und einem y-Wert zugeordnet werden. Das Intervall für die x-Werte war mit $x = 0$ und $x = 20$ eingegrenzt. Die Untergrenze der y-Werte liegt bei 0 und für den maximal möglichen Wert wurde das Maximum der Planck-Verteilung numerisch bestimmt. Die Rechung ist in 2 zu finden.
Die Ableitung wurde gebildet und mit **scipy.optimize.brentq** die Nullstelle bestimmt.
Als Grenze ergibt sich $y_{\text{max}} = 0.21888647009110665$.

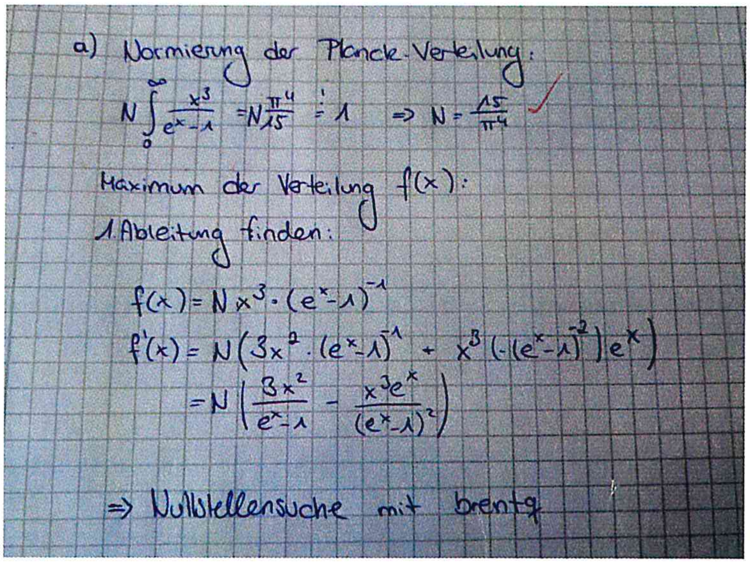


Abbildung 1: Rechnung der Normierung und ersten Ableitung.

Es werden nun immer zwei Zufallszahlen gezogen und als (x,y) Punkt interpretiert. Gilt nun

$$y_{\text{zufall}} \leq f(x_{\text{zufall}})$$

wird das Zufallspaar akzeptiert und gespeichert, sonst verworfen. Eine while-Schleife durchläuft dieses Verfahren bis 10^5 Paare gefunden worden sind. Die x-Werte entsprechen den gesuchten Zufallszahlen und sind in ?? aufgetragen. Dabei wurden 339253 Zufallszahlen verworfen und die Methode hat im Durchschnitt 3, 3 Sekunden gebraucht. 4/4P.

b) Um die Zahl der verworfenen Zahlen zu minimieren wird eine gestückelte Majorante $g(x)$ definiert. Diese ist in 3 dagestellt. Es kann eine Transformationsmethode verwendet werden, weil die Stammfunktion von $g(x)$ intervierbar ist und dies für die Methode benötigt wird. Um ein Samplen aus dieser Verteilung zu ermöglichen muss die Funktion normiert werden. Die Rechnung ist in 4 zu finden.

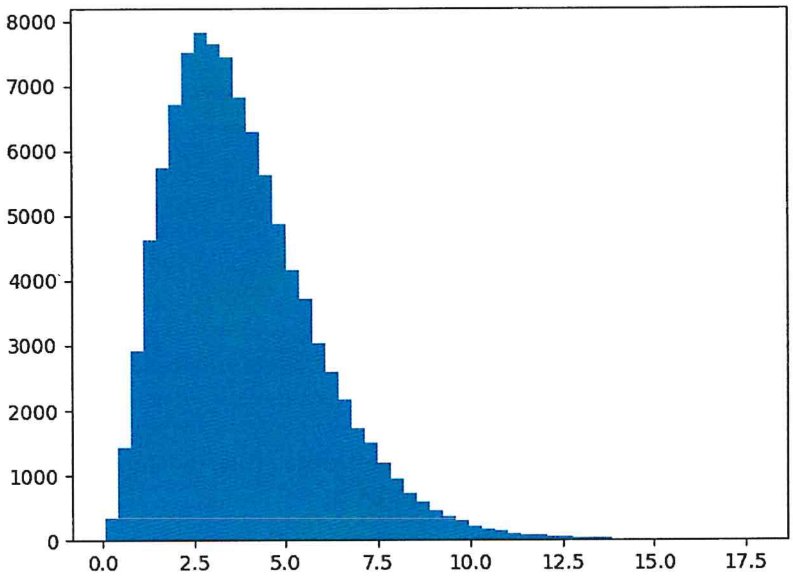


Abbildung 2: Zufallszahlen nach der Planck Verteilung. ?

a) ~~oder b)~~

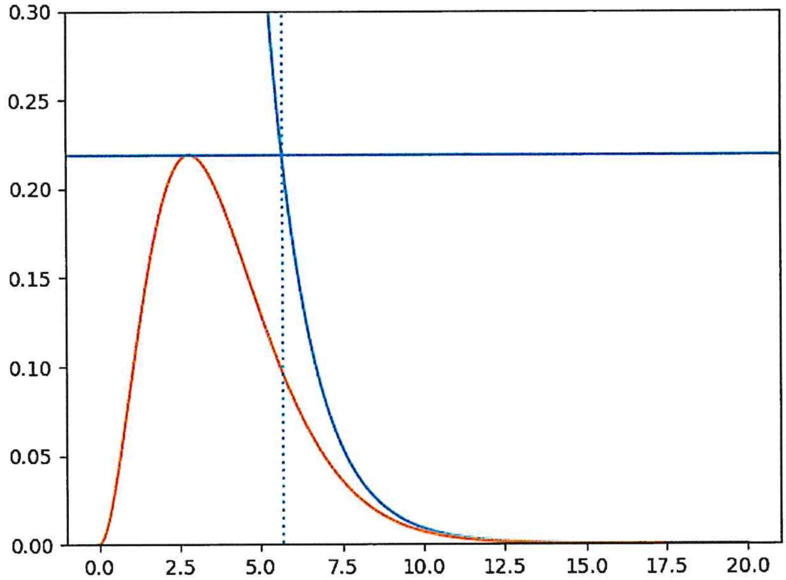


Abbildung 3: Planckverteilung und die Majoranten.

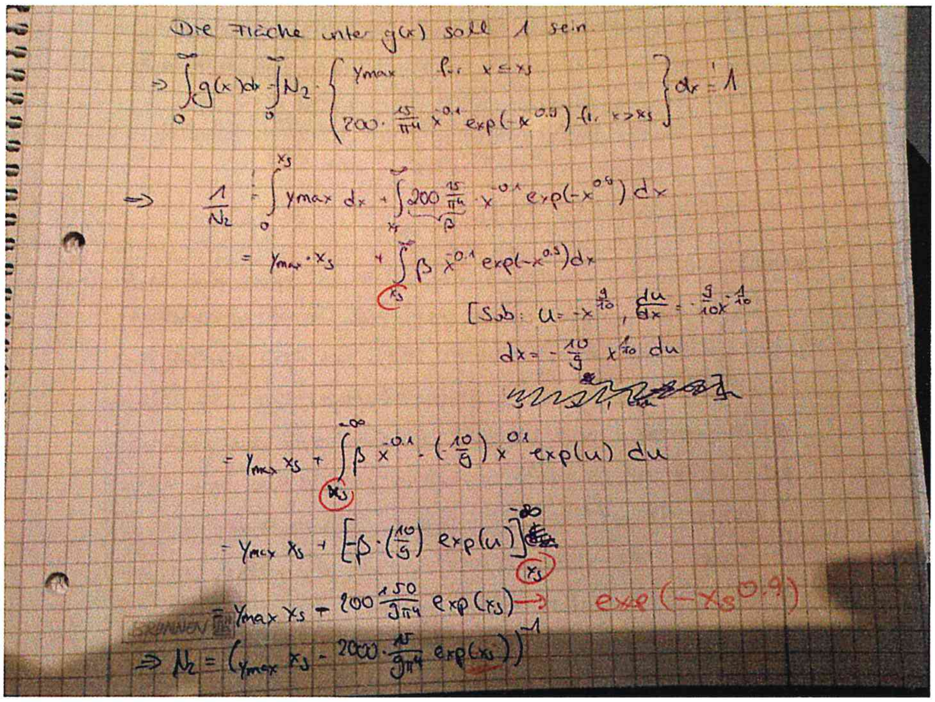


Abbildung 4: Berechnung der Normierung für g(x).

2/6P.

c) 1/2P.

~~7P.~~

2 Aufgabe 2

a) Aufgrund der Symmetrie der Gauß-Kurve folgt

$$g(x_j|x_i) = g(x_i|x_j) \implies \frac{g(x_j|x_i)}{g(x_i|x_j)} = 1\ . \tag{1}$$

Damit er gibt sich die Übergangswahrscheinlichkeit

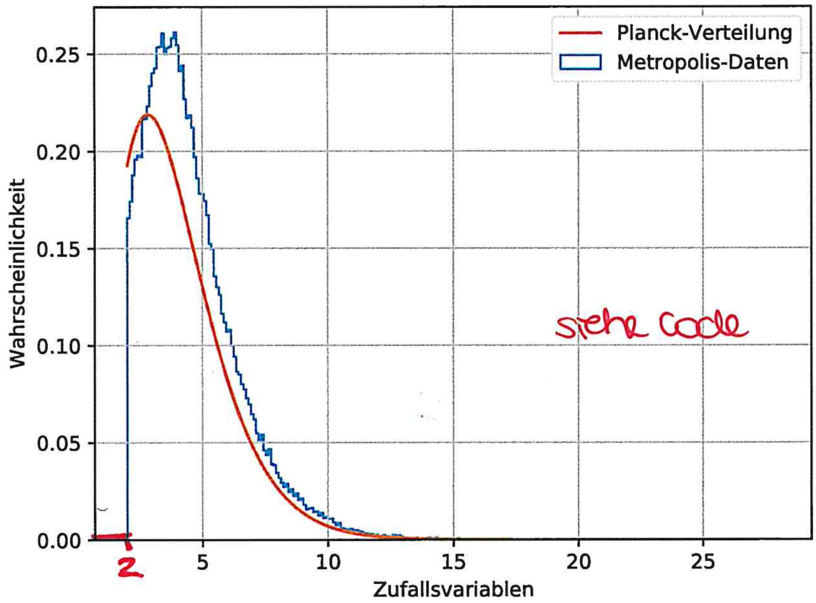
min min

$$M_{i\to j} = \left(1, \frac{f(x_j)}{f(x_i)}\frac{g(x_j|x_i)}{g(x_i|x_j)}\right) = \left(1, \frac{f(x_j)}{f(x_i)}\right) = \text{Metroplolis-Algorithmus}\ . \tag{2}$$

0.5/1P.

b) Der Algorithmus wurde als Funktion initialisiert, diese nimmt den Startwert, eine Anzahl an zu generierenden Werten, eine Schrittweite und eine Wahrscheinlickeitsverteilung entgegen. Zuerst werden die Variablen `position`, `countDoku` und die Arrays `randoms`,`Iteration` definiert. Die Arrays werden nachher aufgefüllt und stellen die Rückgabe da. Die `position` stellt den aktuellen Wert des Algorithmus da. Die Variable `countDoku` dient hier als Zähler und gibt nachher den Iterationsschritt an, bei dem die Zufallszahl gezogen wurde, dieser wird im Array Iteration gespeichert. Beides ist Bestandteil des Aufgabenteils d). Die Variable `position` wird auf den Startwert gesetzt. Der Zähler natürlich auf Null. Nach der Definition folgt eine while-Schleife die erst beendet wird, wenn genügend Werte vom Algoritmus gezogen wurden. In der Schleife wird zuerst eine neue Zufallsvariable aus einer Gleichverteilung gezogen, diese stellt die zu überprüfende nächste Position da. Danach wird der Zähler erhöht. Dann wird die Übergangswahrscheinlichkeit wie in (2) mit der übergebenen Wahrscheinlickeitsverteilung berechnet. Danach wird eine Zufallsvariable zwischen Null und Eins gezogen. Ist diese kleiner-gleich der Übergangswahrscheinlichkeit dann wird die aktuelle Position auf die nächste Position gesetzt und der Iterationsschritt (aktueller Wert des Zählers) gespeichert. Tritt der Fall ein, das die gezogene Zufallszahl größer ist als die Übergangswahrscheinlichkeit, wird ohne das etwas gespeichert wird wieder mit der Schleife begonnen. Dies ist im Code in Zeile 64-81 zu finden. 2/3P.

c) Die Planckverteilung wurde (wie im Code von Zeile 94-98 zufinden) initalisiert. Die Wahrscheinlick-



siehe Code

2

Abbildung 5: Gegenüberstellung der Daten des Metropolis Algorithmuses und der tatsächlichen Wahrscheinlickeitsverteilung.

eitsverteilung und die dazugehörigen Daten aus dem Metroplolis Algorithmus mit Starwert 30 und Schrittweite zwei sind in Abbildung 5 gezeigt.

2/3P.

d) Das Traceplot ist in der Abbildung 6 dagestellt.

0.5/1P.

2 Fehler im Code

1. Die left, right Geschichte in der Planck Funktion zieht die Verteilung auf minimale Werte bei x=2 Bedingung lautet nicht „wenn x - stepsize > 0" sondern „wenn x > 0"

2. Immer „position" appenden, wenn x > 0 oder rand > P eben die vorherige Zahl für Position

/5P.

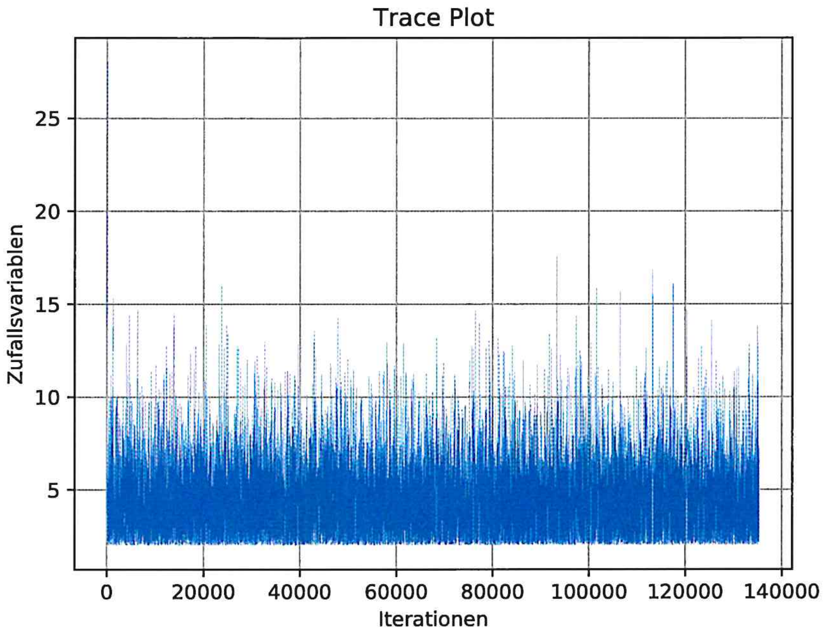


Abbildung 6: Traceplot für die Werte des Metroplolis Algorithmuses.

Code fuer Blatt03

Kasper, Appel, Schroeer

9. November 2018

../B/1/Blatt03_Kasper_Appel_Schroeer/Blatt3_Kasper_Appel_Schroeer.py

```python
def Aufgabe1():

    # Definitionen und so
    def Planck(x):
        return(15/(np.pi)**4)*(x**3)/(np.exp(x)-1)

    def PlanckAbl(x):
            return((15/(np.pi)**4)
                    * ((3*(x**2)/(np.exp(x)-1))
                    - (((x**3)*np.exp(x))/(np.exp(x)-1)**2)))

    def SchnittMajo(x, ymax):
        return (ymax-200*(15/(np.pi)**4) * (x**(-0.1)) * (np.exp(-x**(0.9))))

    def Majorante2(x):
        return(200*(15/(np.pi)**4) * (x**(-0.1)) * (np.exp(-x**(0.9))))

    def Majorante1(x):
        return(0.21888647009110665)

    # Vorbereitungsbums
    x = np.linspace(0.0001, 20, 10000)
    xmax = optimize.brentq(PlanckAbl, 2.5, 5)
    ymax = Planck(xmax)
    xs = optimize.brentq(SchnittMajo, 4.5, 6, args=(ymax))
    plt.plot(x, Majorante2(x))
    plt.plot(x, Planck(x))
    plt.ylim(0, 0.3)
    plt.axhline(y=ymax)
    plt.axvline(x=xs, linestyle=':')
    plt.savefig('Majoranten.png')
    plt.clf()
    # Aufgabenteil a)

    def Rueckweisung():
        zaehler = 0
        Verworfenezahlen = 0
        Zufallszahlen = np.empty((100000, 2))
        while zaehler < 100000:
            xwert = np.random.uniform(0, 20)
            ywert = np.random.uniform(0, ymax)
            if(ywert <= Planck(xwert)):
                Zufallszahlen[zaehler] = [xwert, ywert]
                zaehler += 1
            else:
                Verworfenezahlen += 1
        print("Es werden", Verworfenezahlen, "Zahlen verworfen")
        return(Zufallszahlen)

    # Zeit = timeit.timeit(TEST_RUECKWEISUNG, number=1)
    start = time.time()
    Zahlen = Rueckweisung()
    end = time.time()
    print("ymax =", ymax)
    print("Der Schnittpunkt liegt bei x=", xs)
    print("Die Rückweisungsmethode aus a) braucht", (end-start), "Sekunden")
    plt.hist(Zahlen[:, 0], bins=50)
```

```python
    plt.savefig('Histogramm.png')


def Aufgabe2():

    ## Dat Algorithm ##
    def Metropolis(xzero,num, step,PDF):
        position = xzero
        countDooku = 0 # d)
        randoms = np.array([])
        Iteration = np.array([]) # d)
        while (num != randoms.size):
            xnext = np.random.uniform(position-step,position + step,1)
            countDooku+=1 # d)
            P = min(1,PDF(xnext,xnext-step,xnext+step) /
    PDF(position,position-step,position+step))
            rand = np.random.uniform(0,1,1)
            if rand <= P:
                position = xnext
                randoms = np.append(randoms,position)
                Iteration = np.append(Iteration,countDooku) # d)

            else:
                continue
        return randoms,Iteration # a), d)
    ## Whole lotta Plancking ##
    #def Planckdestr(left,right,num):
    #   if left <0:
    #       return stat.planck.rvs(lambda_,size=num)
    #   else:
    #       return stat.planck.rvs(lambda_,loc=left,size=num)
    #
    #def PlanckPDF(x,left,right):
    #   if left <0:
    #       return stat.planck.pmf(x,lambda_)
    #   else:
    #       return stat.planck.pmf(x,lambda_,loc = left)
    def PlanckPDFBlatt(x,left,right):
        if left <0:
            return 0
        else:
            return (15/np.pi**4)*(x**3)/(np.exp(x) -1)


    ## Planck Plott ##
    plancks,planckitis = Metropolis(30,10**5,2,PlanckPDFBlatt)
    plt.hist(plancks,bins= 'auto',density = 'True',histtype='step', label="Metropolis-Daten")
    x = np.linspace(min(plancks),max(plancks),1000)
    plt.plot(x,PlanckPDFBlatt(x,min(plancks),max(plancks)),label= "Planck-Verteilung")
    plt.legend(loc='best')
    plt.xlabel(r'Zufallsvariablen')
    plt.ylabel(r'Wahrscheinlichkeit')
    plt.grid()
    plt.savefig('Planckvergleich.pdf')
    plt.clf()
    ## Trace Plot ##
    plt.title('Trace Plot')
    plt.plot(planckitis,plancks,linewidth = 0.001)
    plt.xlabel(r'Iterationen')
    plt.ylabel(r'Zufallsvariablen')
    plt.grid()
    #plt.legend(loc='best')
    plt.savefig('Traceplot.pdf')


if __name__ == '__main__':
    import matplotlib.pyplot as plt
    import numpy as np
    from scipy import optimize
    import time
```

statt hier die <0 Problematik anzugehen (klappt so nicht!) einfach if xnext<0 nehmen

Immer appenden!

?

Wenn x-stepsize <0, return 0 zieht Verteilung nach oben

x-stepsize <0

$\Rightarrow x < 2 \Rightarrow$ Planck(x)=0

deshalb fängt eure Verteilung bei x=2 an

Eure Bedingung muss heißen

if x<0:
  return 0

```
Aufgabe1()
Aufgabe2()
```