Aufgabe9

November 8, 2018

1 Aufgabe 9 Der Metropolis-Hastings-Algorithmus

1.1 Aufgabenteil a)

Für eine gegebene Wahrscheinlichkeitsverteilung $W(x)$ ist die Wahrscheinlichkeit, dass ein Punkt vom Metropolis-Hastings-Algorithmus angenommen wird

$$P = \min\left(1, \frac{W(x)P(x|y)}{W(Y)P(y|x)}\right)$$

für eine symmetrische Funktion $P(x)$ gilt dabei

$$P(x|y) = P(y|x)$$

Dadurch kürzt sich die Wahrscheinlichkeit zur Annahme eines Punktes zu

$$P = \min\left(1, \frac{W(x)}{W(Y)}\right)$$

Dies entspricht dem Metropolis-Algorithmus.

1.2 Aufgabenteil b)

Der Algorithmus orientiert am Ablaufschema aus der Vorlesung:

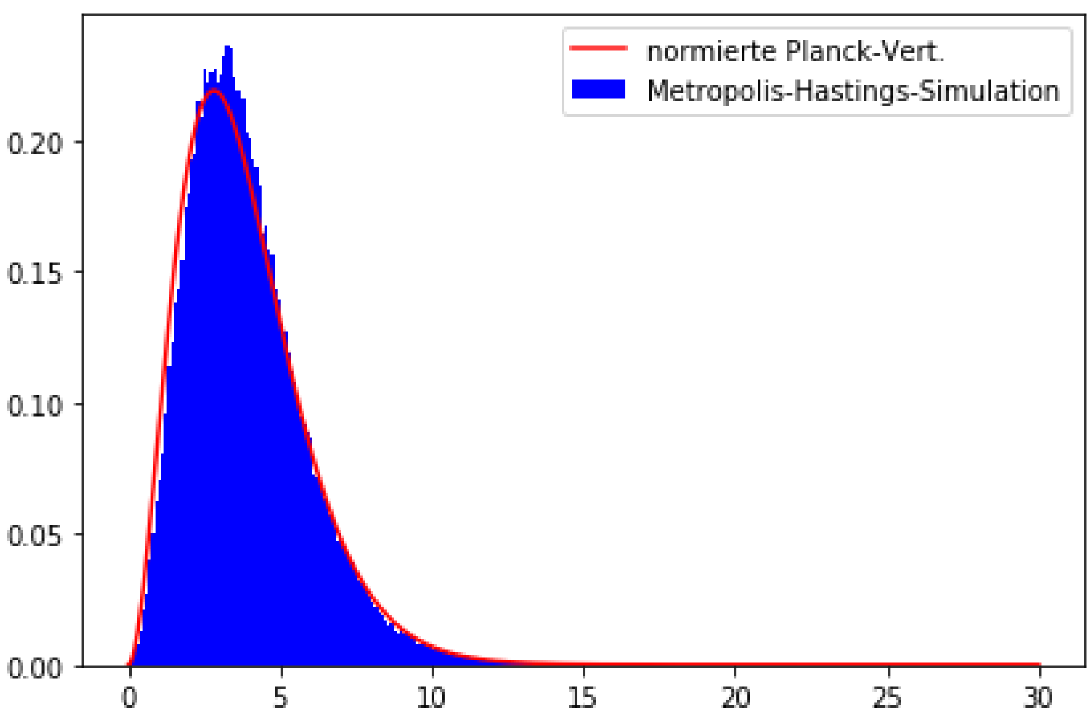
```
In [1]: import numpy as np
        import matplotlib.pyplot as plt

In [2]: def f(x):
            return (15*x**3)/((np.pi)**4*(np.exp(x)-1))

        def fMetro(s,x0,number):
            w = [x0]     #Hier werden die Ausgabewerte gespeichert
            time=[0]     #Hier werden die Iterationen für den Trace Plot mit gezählt
            for i in range(1,number):
                y=w[i-1]+np.random.uniform(-s,s)    #Gehe einen zufälligen Schritt
                                                    #innerhalb der Schrittweite s
                while min(1,f(y)/f(w[i-1])) < np.random.uniform(0,1) or y < 0:
                                        #inverses Annahmekriterium + Definitionsbereich
```

```
                y=w[i-1]+np.random.uniform(-s,s) #Neuer Schritt bei Ablehnung
            w.append(y) #Wird der Schritt angenommen,
                        #so wird der Wert der Ausgabe hinzu gefügt
            time.append(i) #Die Iteration wird mitgezählt
        return time,w
```

1.3 Aufgabenteil c)

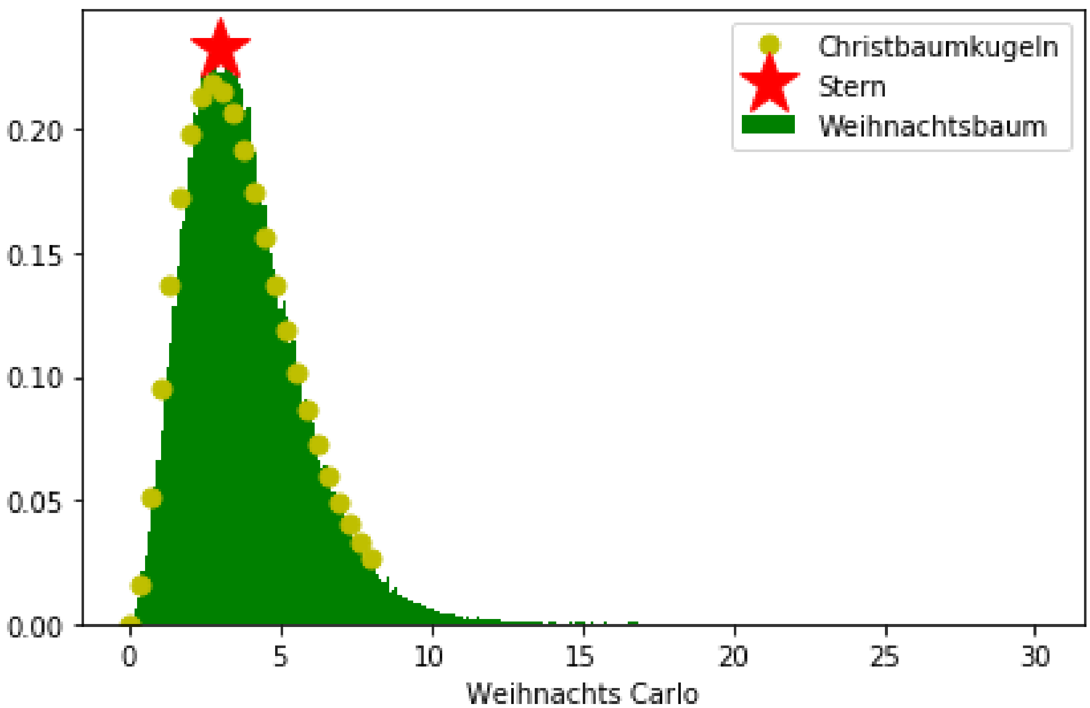
Im folgenden die Simulation mit den angegebenen Parametern verglichen mit der Verteilungsfunktion:

```
In [3]: t,w=fMetro(2,30,100000) #Die Verteilung wird nach Angaben erstellt..
        plt.hist(w,bins='auto',normed=True,color='blue',
                 label='Metropolis-Hastings-Simulation') #..und geplottet
        ln = np.linspace(0.000001,30, 1000)
        plt.plot(ln,f(ln),'r-',label='normierte Planck-Vert.')
        plt.legend(loc='best')
        plt.tight_layout()
        plt.show()
```



1.4 Weihnachts Carlo (Vllt. ein bisschen früh aber egal)

```
In [4]: t,w=fMetro(2,30,100000) #Die Verteilung wird nach Angaben erstellt..
        plt.hist(w,bins='auto',normed=True,color='green',
                 label='Weihnachtsbaum')#..und geplottet
        ln = np.linspace(0.000001,8, 24)
        plt.plot(ln,f(ln),'yo',markersize=7,label='Christbaumkugeln')
        plt.plot(3,0.232,'r*',markersize=24.0,label='Stern')
        plt.xlabel('Weihnachts Carlo')
        plt.legend(loc='best')
        plt.tight_layout()
        plt.show()
```



1.5 Aufgabenteil d)

Im folgenden Traceplot ist zu erkennen welcher Wert bei welcher Iteration erzeugt wurde. Zu Beginn ist eine Nahezu senkrechte gerade zu sehen welche fom Startwert zum Häufungswert (Maximum der Planckfuntion läuft). Der Algorithmus braucht hierzu ca. 40 Iterationen. Danach hält sich der Algorithmus meist in der Häufungsregion auf mit seltenen gelegentlichen ausschläge, welche im gewichteten Plot jedoch kaum noch zu sehen sind. Die positve Flanke fällt dabei wesentlich flacher, wie es die Verteilungsfunktion erwarten lässt.

```
In [5]: plt.plot(t,w,'c.',label='Trace Full alpha')
        plt.plot(t,w,'b.',alpha=0.01)
```

```
plt.plot(80000,30,'b.',label='Trace valued (alpha = 1%)')
plt.xlabel('Iterationen')
plt.ylabel('simulierter Wert')
plt.legend(loc='upper right')
plt.tight_layout()
plt.show()
```