Theoretische Physik 3 - Elektrodynamik

Prof. Marc Wagner
ITP, Universität Frankfurt - WS 2016/2017

Organisation der Übungen: Antje Peters Raum 2.107 peters@th.physik.uni-frankfurt.de

---

Blatt 1

vom 21.10.2016, Abgabe am 28.10.2016 in der Vorlesung

2) Schwingende Saite in 3 Raumdimensionen (mündlich) *(2+2+1=5 Punkte)*

Startpunkt dieser Aufgabe ist eine in x-Richtung gespannte Saite, die transversal in z-Richtung schwingt, genau wie in der Vorlesung diskutiert. Erweitere die in der Vorlesung angestellten Überlegungen von 2 auf 3 Raumdimensionen, d.h. die transversalen Schwingungen finden nun in der y-z-Ebene statt und werden durch $(y(x,t), z(x,t))$ beschrieben.

i. Stelle die Lagrange-Funktion und Lagrange-Dichte auf.

ii. Bestimme mit Hilfe von Variationsrechnung die Feldgleichungen und eventuell auftretende zusätzliche Bedingungen.

iii. Gib die allgemeine Lösung der Feldgleichungen an.

3) Klassische Stringtheorie (schriftlich) *(3+2+3+3=11 Punkte)*

Betrachte eine Saite, die sich aus N identischen Massenpunkten (Masse m, Positionen $\mathbf{X}_j(t)$) und N identischen Federn (Federkonstante k, Ruhelänge 0) zusammensetzt und sich beliebig in 3 Raumdimensionen bewegen kann. Die letzte Feder ist mit dem ersten Massenpunkt verbunden, d.h. die Saite ist geschlossen, es liegen also periodische Randbedingungen vor. (Man spricht auch von einem "String".)

i. Führe auf sinnvolle Weise den Kontinuumslimes durch und gib die resultierende Lagrange-Dichte an (wähle zur Beschreibung des Strings $\mathbf{X}(s,t) = (x(s,t), y(s,t), z(s,t))$, $0 \leq s < 1$ beziehungsweise $\mathbf{X}(s+1,t) = \mathbf{X}(s,t)$). *Hinweis: Beachte, dass* $\mathbf{X}_0(t) = \mathbf{X}_N(t)$.

ii. Bestimme mit Hilfe von Variationsrechnung die Feldgleichungen und eventuell auftretende zusätzliche Bedingungen.

iii. Bestimme die Bewegung des Strings für Anfangsbedingungen $\mathbf{X}(s,0) = \mathbf{f}(s)$, $\dot{\mathbf{X}}(s,0) = \mathbf{g}(s)$.

iv. Skizziere (z.B. mit Hilfe eines Computers) die zeitliche Entwicklung des Strings für $\mathbf{f}(s) = (2r\cos(2\pi s), r\sin(2\pi s), 0)$, $\mathbf{g}(s) = (0,0,v)$, d.h. fertige Momentaufnahmen des Strings zu mindestens 10 verschiedenen Zeitpunkten an.

Randbemerkung: Solche geschlossenen "Gummibänder" sind in quantisierter, relativistischer Form die wesentlichen Bausteine der Stringtheorie, einer modernen Theorie, die möglicher Weise die vier fundamentalen Kräfte sowie sämtliche Elementarteilchen einheitlich und quantisiert beschreibt (siehe z.B. `https://de.wikipedia.org/wiki/Stringtheorie`*).*

4) Elektrostatisches Potential und E-Feld einer Punktladung (mündlich) ***(2+2=4 Punkte)***

Die Beziehung zwischen elektrostatischem Potential und E-Feld lautet

$$\mathbf{E} \quad = \quad -\nabla\Phi(\mathbf{r}). \tag{1}$$

In der Vorlesung wurde bzw. wird das elektrostatische Potential und E-Feld einer Punktladung q_j bei $\mathbf{r}_j$ als

$$\Phi(\mathbf{r}) \quad = \quad \frac{q_j}{|\mathbf{r}-\mathbf{r}_j|} \quad , \quad \mathbf{E}(\mathbf{r}) \quad = \quad \frac{q_j}{|\mathbf{r}-\mathbf{r}_j|^2}\frac{\mathbf{r}-\mathbf{r}_j}{|\mathbf{r}-\mathbf{r}_j|} \tag{2}$$

angegeben. Zeige, dass (2) die Beziehung (1) erfült. Führe diesen Nachweis auf zwei unterschiedlichen Wegen aus:

i. durch Ableitung von Φ;

ii. durch geeignete Integration von $\mathbf{E}$.

Python Übung 2 – Plotten einer beidseitig eingespannten Saite mit Dirichlet- und Neumann-Randbedingungen

In Kapitel 5.1 des Skripts „Theoretische Physik 2“[1] wurde die beidseitig eingespannte Saite mit Dirichlet-Randbedingungen betrachtet. Der angegebene Python-Code erzeugt den im Skript gezeigten Plot. Starten Sie den Code auf Ihrem Computer und reproduzieren Sie den Plot. Modifizieren Sie den Python-Code so, dass statt der Dirichlet-Randbedingungen Neumann-Randbedingungen verwendet werden.

Hinweise zum Code:

- `def f(x):` definiert die Funktion f(x). Der darauffolgende Funktionsrumpf muss eingerückt werden.
- `return` gibt den Rückgabewert (d.h. das Ergebnis) einer Funktion an.
- für Experten: `np.vectorize(f)` vektorisiert die Funktion `f`. `f` akzeptiert nun ein Array als Argument und wirkt auf jedes Element dieses Arrays.
- `for i in range(10):` definiert eine for-Schleife in Python. `i` nimmt nacheinander die Werte 0 bis 9 an. Der Rumpf der for-Schleife muss eingerückt werden.

Hinweis für Linux-Nutzer:

`ipython` ist eine Kommandooberfläche, die sich zum Arbeiten mit Python unter Linux empfiehlt. Sie bietet unter anderem Befehlsergänzung und die Möglichkeit zum interaktiven Arbeiten mit Python. Mit dem Befehl `ipython` wird ipython gestartet. Mit dem Befehl `%run programm.py` wird das Progamm programm.py ausgeführt.

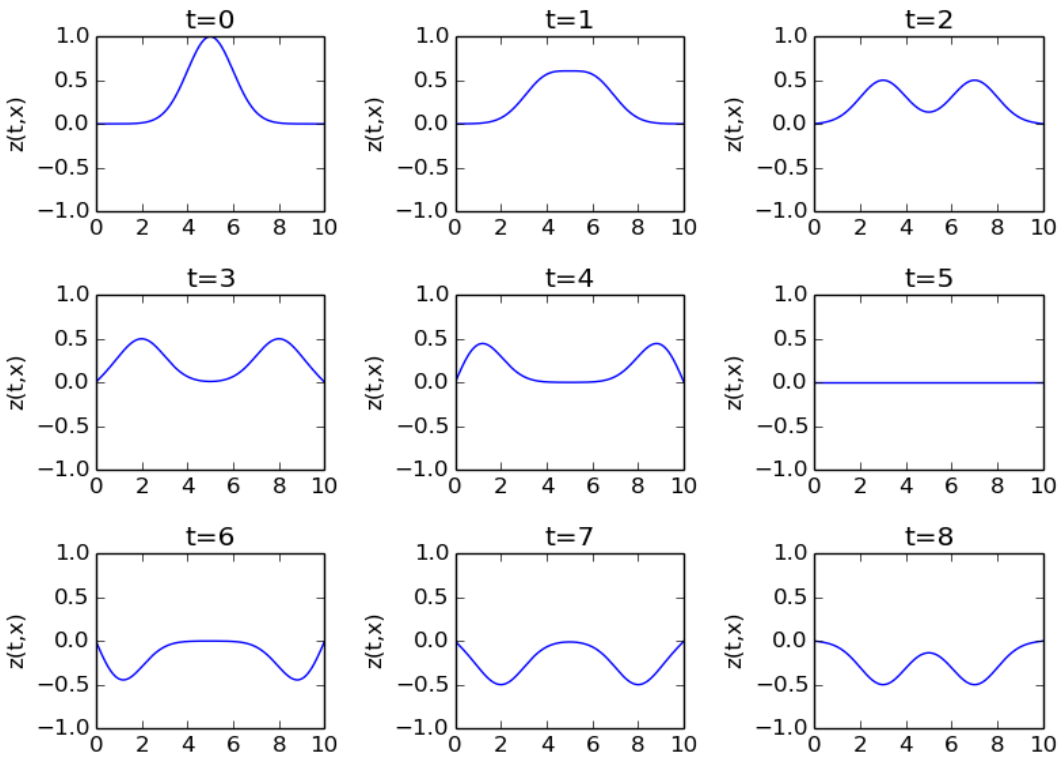

Illustration 1: Plot der beidseitig eingespannten Saite mit Dirichlet-Bedingungen

1 http://th.physik.uni-frankfurt.de/~mwagner/mcwagner.html#theo2

Code:

```
import numpy as np
import matplotlib.pyplot as plt
from math import exp

# Plotten der beidseitig
# eingespannten Saite
# mit Dirichlet-Randbedingungen
# vgl. Skript "Theoretische
# Physik 2", Kapitel
# 5.1.


##### Definition of variables

Delta = 1.0
sigma = 1.0
L = 10.0
v=1.0

# Define array of steps in x
# direction
x=np.arange(0,10,0.01)



##### Definition of functions

# define function f
def f(u):
   return Delta * (exp(-pow((u -
0.5*L),2)    /    (2.0*pow(sigma,
2.0))) - exp(-pow(0.5*L, 2.0) /
(2.0*pow(sigma, 2.0))))

# define function G
def G(u):
   return 0.0
```

```
# define function z_plus and its
# periodicity
def z_plus(u):
   if(u <= -L):
      return z_plus(u + 2.0*L)
   if(u >= -L and u <= 0.0):
      return 0.5 * (-f(-u) + G(-u) / v)
   if(u >= 0.0 and u <= L):
      return 0.5 * (+f(+u) + G(+u) / v)
   if(u >= L):
      return z_plus(u - 2.0*L)

# define function z
def z(x, t):
   return z_plus(+x + v*t) - z_plus(-x +
v*t)

# vectorize function z, so argument
# of z can be an array
v_z=np.vectorize(z)


##### Plotting

plt.figure(1)

# for-loop to produce subplots
for t in range(0,9):
   p=plt.subplot(3,3,t+1)
   plt.plot(x,v_z(x, t))
   plt.ylabel('z(t,x)')
   plt.title("t=%s" % t)
   p.set_xlim([0, 10])
   p.set_ylim([-1, 1])

# makes the plot look nicer
plt.tight_layout()

plt.show()
```