Vorbemerkung

Dies ist ein abgegebener Übungszettel aus dem Modul physik321.

Dieser Übungszettel wurde nicht korrigiert. Es handelt sich lediglich um meine Abgabe und keine Musterlösung.

Alle Übungszettel zu diesem Modul können auf http://martin-ueding.de/de/university/bsc_physics/physik321/ gefunden werden.

[disclaimer]

physik321 – Übung 8

Gruppe 8 – Julia Volmer

Martin Ueding
mu@uni-bonn.de

Simon Schlepphorst
s2@uni-bonn.de

2014-07-07

Aufgabe	H 8.1	H 8.2	H 8.3	$\sum$
Punkte	/ 10	/ 20	/ 10	/ 30

H 8.1 potentielle Energie des statischen Magnetfeldes

Wir beginnen mit:

$$P = \int \mathrm{d}V \, \left\langle \boldsymbol{E}, \boldsymbol{j} \right\rangle$$

Wir setzen ein, dass $\boldsymbol{\nabla} \times \boldsymbol{H} - \boldsymbol{j} = 0$ ist.

$$P = \int \mathrm{d}V \, \left\langle \boldsymbol{E}, \boldsymbol{\nabla} \times \boldsymbol{H} \right\rangle$$

Nun benutzen wir die gegebene Identität, allerdings in der richtigen Fassung. Diese lautet $\langle \boldsymbol{\nabla}, \boldsymbol{A} \times \boldsymbol{B} \rangle = \langle \boldsymbol{B}, \boldsymbol{\nabla} \times \boldsymbol{A} \rangle - \langle \boldsymbol{A}, \boldsymbol{\nabla} \times \boldsymbol{B} \rangle$. Dies können wir durch Nachrechnen (siehe unten) überprüfen.

$$P = \int \mathrm{d}V \, \left(\langle \boldsymbol{H}, \boldsymbol{\nabla} \times \boldsymbol{E} \rangle - \langle \boldsymbol{\nabla}, \boldsymbol{E} \times \boldsymbol{H} \rangle \right)$$

Die Rotation des elektrischen Feldes ist gerade $-\dot{\boldsymbol{B}}$.

$$P = \int \mathrm{d}V \, \left(\left\langle \boldsymbol{H}, \dot{\boldsymbol{B}} \right\rangle - \langle \boldsymbol{\nabla}, \boldsymbol{E} \times \boldsymbol{H} \rangle \right)$$

Für den zweiten Summanden wenden wir den Satz von Gauß an. Im ersten Summanden wenden wir an, dass $H = \frac{1}{\mu} B$ ist.

$$P = \frac{1}{2\mu} \int \mathrm{d}V \, \frac{\mathrm{d}}{\mathrm{d}t} B^2 - \int\limits_{\partial V} \mathrm{d}A \, \langle \boldsymbol{\nu}, \boldsymbol{E} \times \boldsymbol{H} \rangle$$

Das Oberflächenintegral ist über den Poyntingvektor, also misst es die Leistung, die durch Wellen nach Außen getragen werden. Falls das Magnetfeld also keine Leistung nach Außen trägt, kann dieses Integral vernachlässigt werden.

$$P = \frac{1}{2\mu} \int \mathrm{d}V \, \frac{\mathrm{d}}{\mathrm{d}t} B^2$$

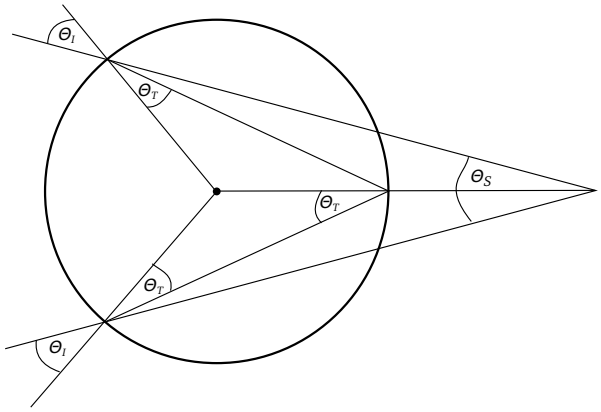

Abbildung 1: Skizze zum Hauptregenbogen

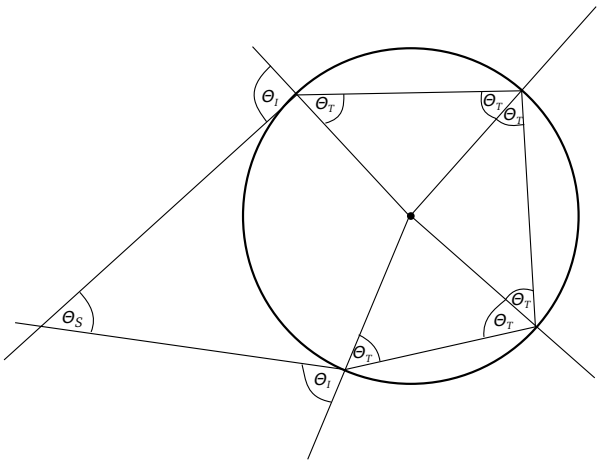

Abbildung 2: Skizze zum Nebenregenbogen

Wir integrieren nach der Zeit und erhalten die Energie im Magnetfeld.

$$U = \frac{1}{2\mu}\int \mathrm{d}V \int_0^t \mathrm{d}t' \, \frac{\mathrm{d}}{\mathrm{d}t'} B^2\left(t'\right)$$
$$U = \frac{1}{2\mu}\int \mathrm{d}V \, B^2(t) = \frac{1}{2}\int \mathrm{d}V \, \langle \boldsymbol{B}, \boldsymbol{H}\rangle$$

Dies ist die gesuchte Relation.

Beweis der Vektoridentität.

$$\begin{aligned}\langle \boldsymbol{\nabla}, \boldsymbol{A}\times\boldsymbol{B}\rangle &= \partial_i \epsilon^i{}_{jk} A^j B^k \\ &= \epsilon^i{}_{jk}\left(\partial_i A^j\right) B^k + \epsilon^i{}_{jk} A^j \partial_i B^k \\ &= B^k \epsilon^i{}_{jk}\left(\partial_i A^j\right) + A^j \epsilon^i{}_{jk}\partial_i B^k\end{aligned}$$

Wir ziehen den nicht differenzierten Vektor vor den $\boldsymbol{\epsilon}$-Tensor und permutieren die Indizes.

$$\begin{aligned}&= B^k \epsilon_k{}^i{}_j \partial_i A^j - A^j \epsilon_j{}^i{}_k \partial_i B^k \\ &= \langle \boldsymbol{B}, \boldsymbol{\nabla}\times\boldsymbol{A}\rangle - \langle \boldsymbol{A}, \boldsymbol{\nabla}\times\boldsymbol{B}\rangle\end{aligned}$$

□

H 8.2 Regenbogen

Wir haben uns für die Aufgabe bei `http://compact.nussnet.at/basiswissen_6/regenbogen.php` inspirieren lassen.

H 8.2.1 Maximum

Analog zur nächsten Aufgabe betrachten wir für den symmetrischen Fall, da wir hier ein Extremum haben. Wir lesen aus der Skizze (Abbildung 1) ab:

$$\frac{1}{2}\Theta_S = \pi - \left(\Theta_I - \Theta_T\right) - \left(\pi - \Theta_T\right)$$

Umgestellt:

$$\Theta_S = -2\Theta_I + 4\Theta_T$$

Wir setzen das Brechungsgesetz ein:

$$\Theta_S = -2\Theta_I + 4\arcsin\left(\frac{\sin\left(\Theta_I\right)}{n}\right)$$

Dann leiten wir ab:

$$\frac{\mathrm{d}\Theta_S}{\mathrm{d}\Theta_I} = \frac{4\cos(\Theta_I)}{n\sqrt{1-\frac{\sin^2(\Theta_I)}{n^2}}} - 2$$

Dies setzen wir gleich 0 und formen nach Θ_I um. Wir erhalten:

$$\Theta_I = \arccos\left(\frac{\sqrt{n^2-1}}{\sqrt{3}}\right)$$

Dann leiten wir noch einmal ab:

$$\frac{\mathrm{d}^2\Theta_S}{\mathrm{d}\Theta_I^2} = 4\left(\frac{\sin(\Theta_I)\cos^2(\Theta_I)}{n^3\left(1-\frac{\sin^2(\Theta_I)}{n^2}\right)^{3/2}} - \frac{\sin(\Theta_I)}{n\sqrt{1-\frac{\sin^2(\Theta_I)}{n^2}}}\right)$$

Wir setzen die Extremstelle Θ_I ein und erhalten:

$$-\frac{32\sqrt{9-n^2}}{27\sqrt{1-\frac{1}{n^2}}n}$$

Zur Vorzeichenbestimmung setzen wir $n = 1.3$ ein und erhalten $-3.85763\,\mathrm{rad}^{-1}$. Es handelt sich also um ein Maximum.

H 8.2.2 Winkelbestimmung

Wir gehen davon aus, dass sich bereits der extremale Winkel eingestellt hat. Somit sind die Reflexionswinkel im Inneren alle gleich sowie – wie auf der Zeichnung auf dem Aufgabenblatt auch dargestellt – der Austrittswinkel gleich dem Eintrittswinkel. Das Fünfeck, das von den Lichtstrahlen aufgespannt wird (siehe Abbildung 2), hat eine Winkelsumme von 3π. Es gilt also:

$$\Theta_S = 3\pi - 2\left(\pi - \Theta_I\right) - 6\Theta_T = \pi + 2\Theta_I - 6\Theta_T$$

H 8.2.3 Extremalbedingung

Zuerst rechnen wir den Winkel um: $\Theta_S = 0.891863$ rad. Dann setzen wir das Brechungsgesetz in die Relation der vorherigen Aufgabe ein und erhalten:

$$\Theta_S = \pi + 2\Theta_I - 6\arcsin\left(\frac{\sin\left(\Theta_I\right)}{n}\right)$$

Die leiten wir ab:

$$\frac{\mathrm{d}\Theta_S}{\mathrm{d}\Theta_I} = 2 - \frac{6\cos\left(\Theta_I\right)}{n\sqrt{1-\left(\frac{\sin\left(\Theta_I\right)}{n}\right)^2}}$$

Und setzen gleich 0 und stellen nach Θ_I um:

$$\Theta_I = \pm\arccos\left(\pm\frac{\sqrt{-1+n^2}}{2\sqrt{2}}\right)$$

Dabei interessiert uns nur die Lösung mit den zwei Pluszeichen. Wir setzen $n_B = 1.334$ und $n_R = 1.331$ ein und erhalten:

$$\Theta_I = 0.937814\,\mathrm{rad}, \quad \Theta_I = 0.879037\,\mathrm{rad}$$

Mit einem Brechungsindex dazwischen würden wir auf die geforderten $\Theta_S = 0.891863\,\mathrm{rad}$ kommen.

Wir leiten noch einmal ab und erhalten:

$$\frac{\mathrm{d}^2\Theta_S}{\mathrm{d}\Theta_I^2} = \frac{6\left(n^2-1\right)\sin(\Theta_I)}{\sqrt{1-\frac{\sin^2(\Theta_I)}{n^2}}\left(n^3-n\sin^2(\Theta_I)\right)}$$

Dort setzen wir die Extremstelle ein und erhalten:

$$\frac{16\sqrt{9-n^2}}{9\sqrt{1-\frac{1}{n^2}n}}$$

Für $n = 1.3$ erhalten wir $5.78644\,\mathrm{rad}^{-1}$, es handelt sich also um ein Minimum.

H 8.2.4 Farbreihenfolge

Da der Winkel ein Minimum und kein Maximum wie beim Primärbogen ist, ist die Farbreihenfolge umgekehrt und somit ist rot innen und blau außen. Außerhalb des Bogens ist es auch heller als innerhalb. Das ganze sehen wir auch an den Ergebnissen der vorherigen Aufgabe, denn rotes Licht wird weniger stark abgelenkt, der rote Bogen ist also kleiner, sprich innen.

H 8.3 potentielle Energie eines Ionenkristalls

Das Durchzählen der Punkte ist relativ simpel realisiert, wir zählen alle Punkte

$$\left\{(x,y,z)\in\mathbb{Z}^3\colon\ \max\left\{x,y,z\right\}\leq a\right\}$$

durch. Dabei ist $a = 60$, wobei der Punkt $(0,0,0)$ ausgeschlossen worden ist. Das Vorzeichen der Ladung an der Stelle (x,y,z) haben wir durch $(-1)^x(-1)^y(-1)^z = (-1)^{x+z+y}$ bestimmt. Somit ist der Ursprung, auf den sich die potentielle Energie bezieht, in einer positiven Ladung zentriert.

Unser Programm für diese Aufgabe haben wir in Python 3 implementiert. Es kann unter `https://github.com/martin-ueding/physik321-08` (`H3.py`) heruntergeladen werden.

```python
#!/usr/bin/python3
# -*- coding: utf-8 -*-

# Copyright © 2012 Martin Ueding <dev@martin-ueding.de>

# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this software and associated documentation files (the "Software"), to deal
# in the Software without restriction, including without limitation the rights
# to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
# copies of the Software, and to permit persons to whom the Software is
# furnished to do so, subject to the following conditions:
#
# The above copyright notice and this permission notice shall be included in
# all copies or substantial portions of the Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
# FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
# AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
# LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
# OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
# SOFTWARE.

"""
Calculates the potential energy in a ion grid. This program is not very fast
since it uses generators instead of simple nested ‘‘for‘‘ loops.
"""

import numpy as np

__docformat__ = "restructuredtext en"

_epsilon_0 = 8.85419e-12
"""
Vacuum permittivity.
"""

_charge = 1.609e-19**2
"""
Elementary charge squared.
"""

_factor = _charge / (4 * np.pi * _epsilon_0)
"""
Factor that comes up in the potential. This saved cycles in the ‘‘potential‘‘
function.
"""

def integers(limit=None):
    """
    Iterator that yields the integers :math:‘\mathbb Z‘ in the order 0, 1, -1,
    2, -2, 3, ....

    :param limit: The largest integer to be returned, inclusively.
    :type limit: int
    :rtype: generator
    """
    i = 0

    yield i
```

```
61
62      while limit is None or i < limit:
63          i += 1
64          yield i
65          yield -i
66
67  def points(limit=None):
68      """
69      Iterator that goes through the three dimensional points around the origin.
70      It will order the points by the uniform norm which is just the maximum of
71      all the coordinates (absolute value).
72
73      :rtype: generator
74      """
75      for x in integers(limit):
76          for y in integers(limit):
77              for z in integers(limit):
78                  yield (x, y, z)
79
80  def potential(sign, R):
81      """
82      Potential energy with charge ``e`` and distance ``R``.
83
84      :param sign: How to count this charge.
85      :type sign: int or float
86      :return: Potential energy.
87      :rtype: float
88      """
89      return sign * _factor / R
90
91  def total_potential(limit=None):
92      """
93      Calculates the total potential including all the points ``(a, b, c)`` where
94      ``max(a, b, c) <= limit`` holds.
95
96      :param limit: Which points to include.
97      :type limit: int
98      :return: Total potential energy
99      :rtype: float
100     """
101     U = 0
102     for x, y, z in points(limit):
103         R = np.sqrt(x**2 + y**2 + z**2)
104
105         if R < 0.1:
106             continue
107
108         sign = (-1)**x * (-1)**y * (-1)** z
109
110         U += potential(sign, R)
111
112     return U
113
114 def main():
115     U = total_potential(60)
116
117     print("U =", U, "J")
118
119 if __name__ == "__main__":
120     main()
```

Die Ausgabe des Programms:

Der Algorithmus hängt leider $\mathcal{O}\left(a^3\right)$ von der Eingabe ab, so dass es ab $a = 60$ mehr als einige Sekunden dauert. Das Ergebnis ändert sich allerdings nur noch wenig für größere a.

Das ganze haben wir auch noch in C implementiert (`H3.c`, auch auf GitHub):

```c
/* Copyright © 2012 Martin Ueding <dev@martin-ueding.de> */

/*
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to
 * deal in the Software without restriction, including without limitation the
 * rights to use, copy, modify, merge, publish, distribute, sublicense, and/or
 * sell copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
 * FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS
 * IN THE SOFTWARE.
 */

/*
 * Calculates the potential energy in a ion grid.
 */

#include <stdio.h>
#include <math.h>

double potential(int limit) {
    double U = 0.0;
    double e2 = pow(1.609e-19, 2);
    double e0 = 8.85419e-12;
    double pi = atan(1)*4;

    int x, y, z;

    short sign;

    for (x = -limit; x <= limit; ++x) {
        for (y = -limit; y <= limit; ++y) {
            for (z = -limit; z <= limit; ++z) {
                if (x == 0 && y == 0 && z == 0) {
                    continue;
                }

                sign = (x + y + z) % 2 == 0 ? 1 : -1;
                U += sign * e2 / (4 * pi * e0 * sqrt(x*x + y*y + z*z));
            }
        }
    }

    return U;
}
```

```
55
56 int main() {
57     double U = potential(60);
58
59     printf("U = %g J\n", U);
60
61     return 0;
62 }
```

Die Ausgabe des Programms:

Es kommt, von Rundungsfehlern abgesehen, das gleiche Ergebnis heraus. Auf GitHub haben wir noch eine Version in Java, die mit 0.1 s noch einigermaßen zügig läuft.