Computerphysik
Hausarbeit 5

Friedrich Hübner 2897111
Fiona Paulus 2909625

25. Juli 2017

Der Differentialgleichungssolver für Runge-Kutta-Verfahren

Für den DGL-Solver wurde das Runge-Kutta-Verfahren mit den Fehlbergkoeffizienten und das klassische Runge-Kutta-Verfahren programmiert. Die Berechnung aller Werte erfolgt genau nach diesen Verfahren. Der Solver wird sowohl in Aufgabe 1 als auch 2 mit den entsprechenden Koeffizienten benutzt.

Programm (abgabe5_runge_kutta.cpp)

In dieser Datei befindet sich das Runge-Kutta-Verfahren analog zur Hausarbeit 4.

In der Datei wird die Klasse RungeKutta definiert, die einen DGL-Solver darstellt. diese hat folgende Funktionen:

- Konstruktor: RungeKutta(f,t_0,y_0, algo): Er initialisiert einen DGL-Solver mit der Funktion f ($y' = f(t, y)$), die Anfangszeit t_0, den Anfangsvektor y_0 und einem Koeffizientenset algo. Dabei ist algo vom Typ struct RungeKuttaAlgorithm, was im wesentlichen eine Sammlung aller Koeffizienten eines Verfahrens ist. Dies ermöglicht eine einfache Auswahl aus vorher schon definierten Koeffizientensets. In der Datei befinden sich die beiden RungeKuttaAlgorithm-Instanzen fehlberg und classic, die jeweils das entsprechende Verfahren darstellen.
- iterate(): berechnet einen neuen Schritt und gegebenenfalls die neue optimale Schrittweite
- setStepSize(h): setzt die aktuelle Schrittweite auf h
- set Precision(p): setzt das ε für die Schrittweitensteuerung auf p

Außerdem werden in der Datei Operatoren für Vektorarithmetik (+,+=,-,*) und für Ausgabe von Vektoren ($<<\dot{}$) überladen. Zusätzlich gibt es noch Funktionen für den Betrag von Vektoren und das Skalar- und Kreuzprodukt.

Aufgabe 1: Randwert

allgemeine Hinweise

Das Programm '1.cpp' wurde unter Linux Mint mit 'g++ -std=c++11 -g -Wall -Wextra 1.cpp -o 1.exe' kompiliert.

a)

Um das gegebene Randwertproblem

$$y''(x) = \frac{2y(x)}{(1+x)^2}, \quad y(0) = 1, \quad y(1) = \frac{1}{2}$$

zu lösen, integriert man die Differentialgleichungen

$$y''(x) = \frac{2y(x)}{(1+x)^2}, \quad y(0) = 1, \quad y'(0) = \alpha$$

$$w''_\alpha(x) = \frac{2w_\alpha(x)}{(1+x)^2}, \quad w_\alpha(0) = 0, \quad w'_\alpha(0) = 1$$

b),c)

Diese Differentialgleichungen werden mit dem Fehlberg-Verfahren (s.o.) integriert um die Randbedingung $y'(0) = \alpha$ zu berechnen. Dazu wird zuerst ein Startwert $\alpha = 2.0$ festgelegt, mit dem die Differentialgleichungen aus a) gelöst werden. Um den Wert von α zu überprüfen wird die Gleichung

$$F(\alpha_n) = y(1, \alpha_n) - y(1) = 0$$

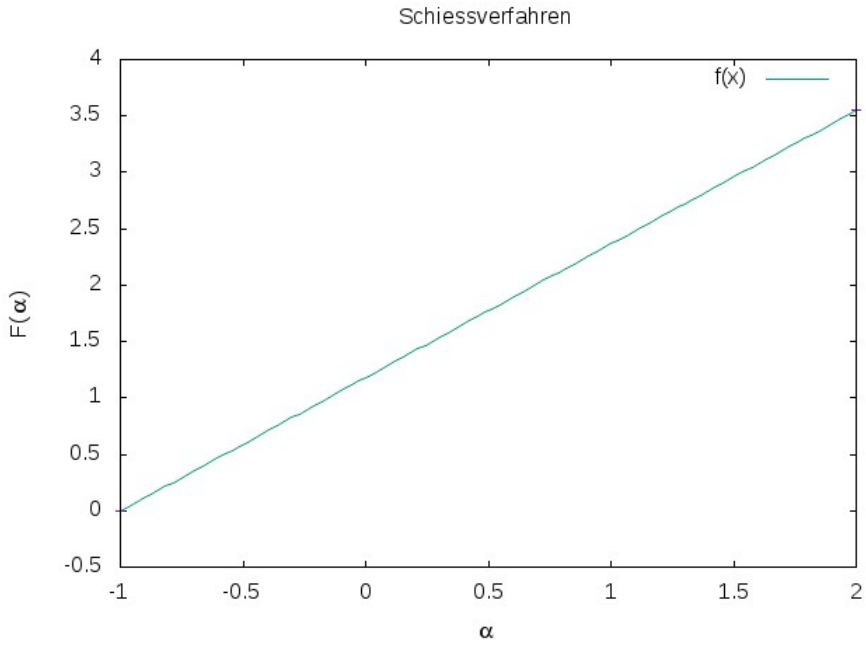
zu überprüfen. Gilt diese Gleichung nicht, wird ein neuer Wert von α berechnet.

$$\alpha_n = \frac{F(\alpha_n)}{F'(\alpha_n)} = \frac{F(\alpha_n)}{w_{\alpha_n}(x)}$$

Dieses Verfahren wird wiederholt bis die Gleichung $F(\alpha_n) = 0$ erfüllt ist. In diesem Fall hat das Verfahren drei Iterationsschritte für α_n, durch die lineare Funktion $F(\alpha)$ werden zwei Schritte erwartet, jedoch wird wird mit dem Fehlbergverfahren nicht exakt bis t=1 integriert, wodurch Abweichungen entstehen.

Iterationsschritt n	α_n	$F(\alpha_n) = y(1, \alpha_n) - y(1)$
0	2.0	$F(\alpha_0) = 3.55008$
1	-0.997482	$F(\alpha_1) = -0.00011$
2	-0.997294	$F(\alpha_2) = -2.71 \cdot 10^{-6}$

Daraus kann man ablesen, dass das Randwertproblem für $\alpha \approx -1$ gelöst ist.



Funktion $F(\alpha_1) = y(1, \alpha_n) - y(1)$ für $\alpha_0 = 2.0$

Sonstige abgegebene Dateien

1.txt

Ausgabedatei des Programms "1.cpp"

1.plt

Plotdatei für Gnuplot für Aufgabe 1

fit.log

Geradenfit für $F(\alpha)$ mit Gnuplot gelöst

Aufgabe 2: Kepler-Orbit

Allgemeine Hinweise

Das Programm '2.cpp' wurde unter Linux Mint mit 'g++ -std=c++11 -g -Wall -Wextra 2.cpp -o 2.exe' kompiliert.

Kepler-Orbit

Die Differentialgleichung

$$\mu\ddot{\vec{r}} = -G\frac{m_1 m_2}{r^2}\hat{e}_r, \quad r_0 = (1,0,0), \quad v_0 = (0,1,0)$$

wird sowohl mit dem klassischen Runge-Kutta-Verfahen (s.o.), als auch mit dem Leapfrog-Verfahren gelöst. Die Lösung mithilfe des Leapfrog-Verfahrens erfolgt ebenfalls durch eine Schrittweise Integration, wobei die schritte für Ort und Geschwindigkeit versetzt berechnet werden.

$$r_{i+1} = r_i + v_{i+\frac{1}{2}}\Delta t$$

$$v_{i+\frac{3}{2}} = v_i + \ddot{r}_{i+1}\Delta t = v_{i+\frac{1}{2}} - G\frac{m_1 m_2}{r\mu}\hat{e}_r\Delta t$$

Wobei bei beiden Verfahren die Schrittweite konstant $\Delta t = 0.01$ ist. Der Anfangswert der Geschwindigkeit wird berechnet durch:

$$v_{\frac{1}{2}} = v_0 + \frac{\Delta t}{2}a(r_0)$$

dadurch muss die Geschwindigkeit bei der Ausgabe wieder zurück verschoben werden um r und v zur gleichen Zeit auszugeben.

$$v_i = v_{i+\frac{1}{2}} - \frac{\Delta t}{2} * a(r_i)$$

Bei jedem Iterationsschritt wird die Energie

$$E = \frac{1}{2}m_1 v_i^2 - G\frac{m_1 m_2}{|r_i|}$$

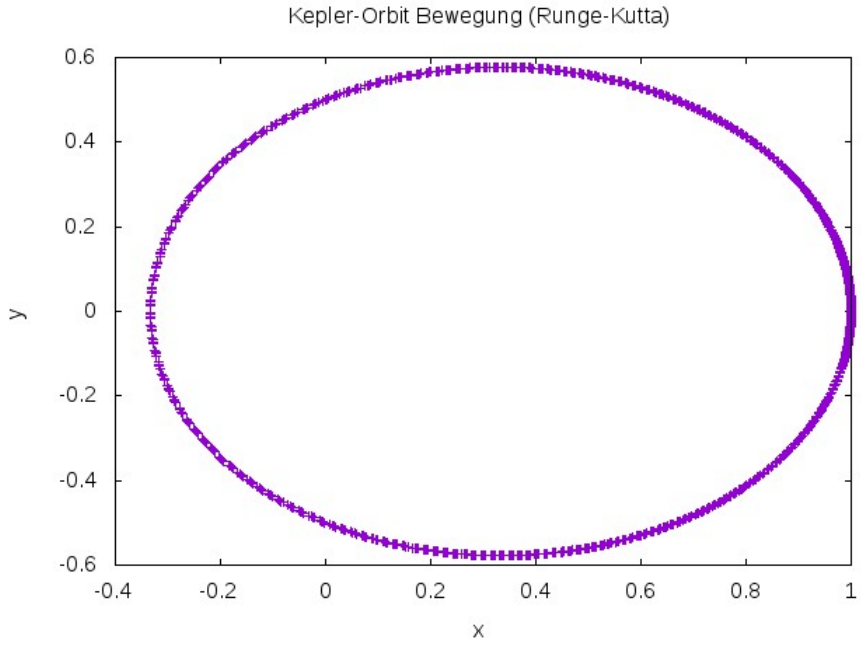
und der Laplace-Runge-Lenz-Vektor

$$\vec{A} = \vec{p}\times\vec{L} - m_1\alpha\hat{e}_{r_i} = m_1\vec{v_i}\times(\vec{r_i}\times m_1\vec{v_i}) - Gm_1^2 m_2\hat{e}_{r_i}$$

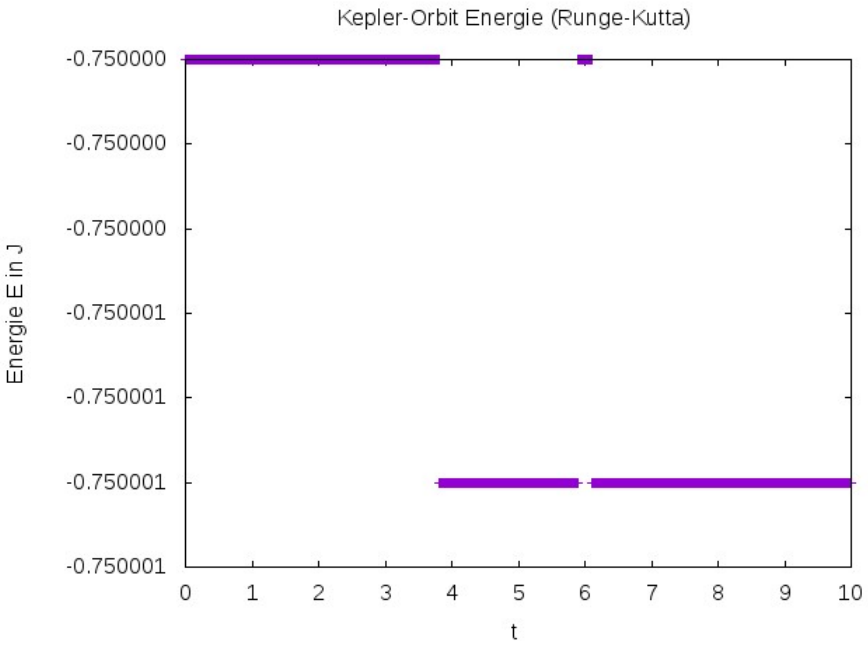
mit

$$\vec{p} = m_1\vec{v_i}, \quad \vec{L} = \vec{r_i}\times\vec{p}, \quad \alpha = Gm_1 m_2$$

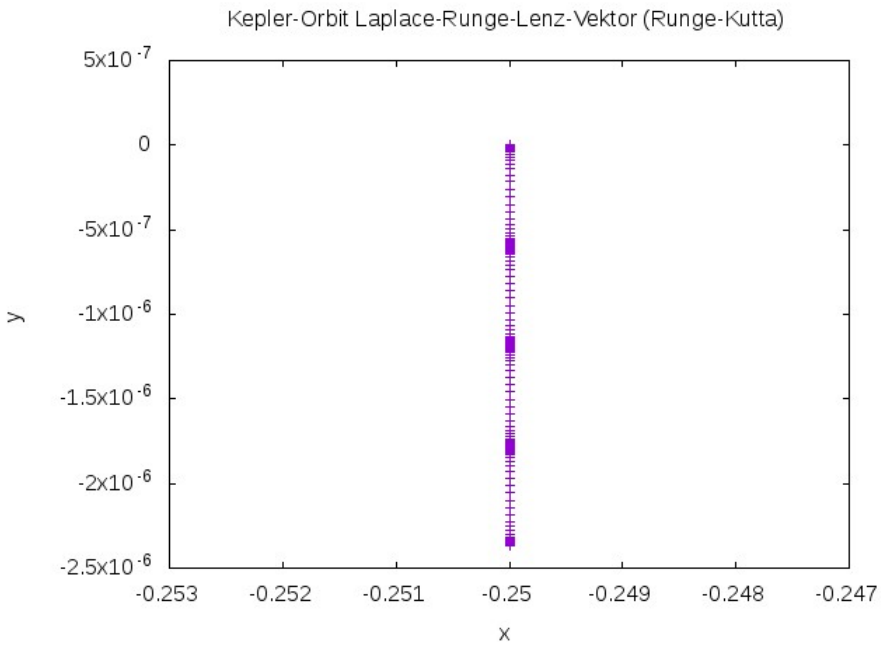
Die Lösung mithilfe des Runge-Kutta-Verfahrens ergibt:



Kepler-Orbit durch Lösung mit Runge-Kutta-Verfahren

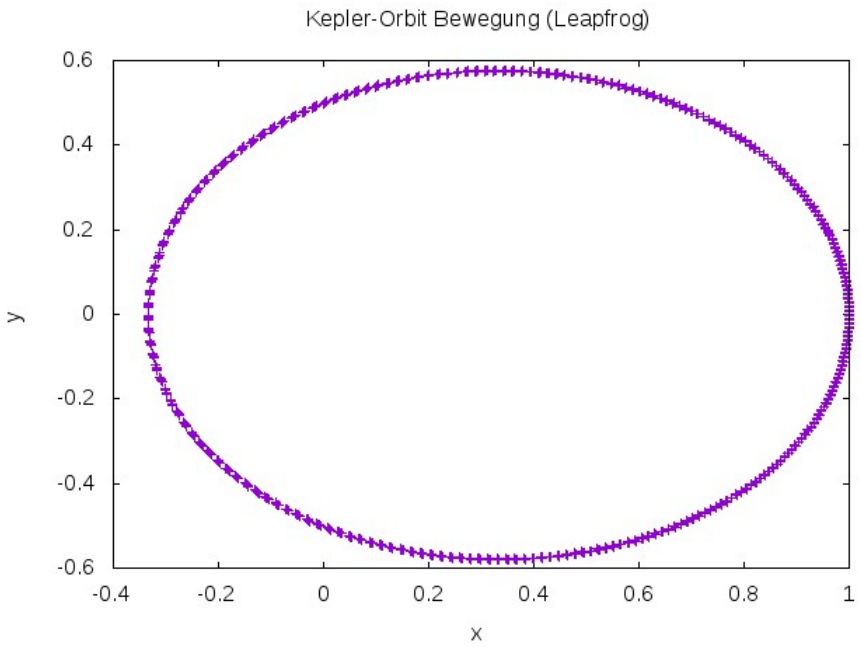


Entwicklung der Energie des Kepler-Orbits durch Lösung mit Runge-Kutta-Verfahren

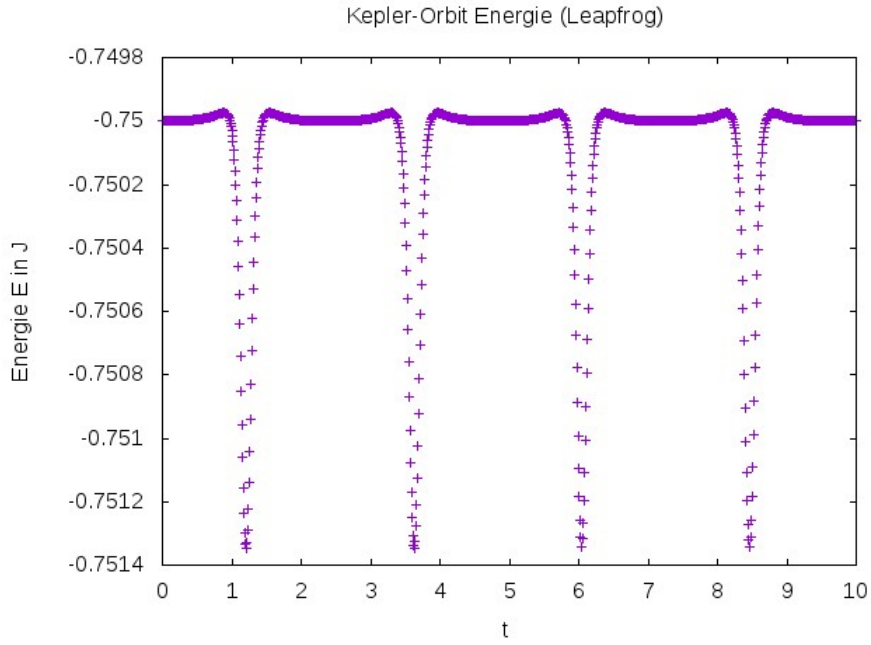


Verhalten des Laplace-Runge-Lenz-Vektors durch Lösung mit Runge-Kutta-Verfahren

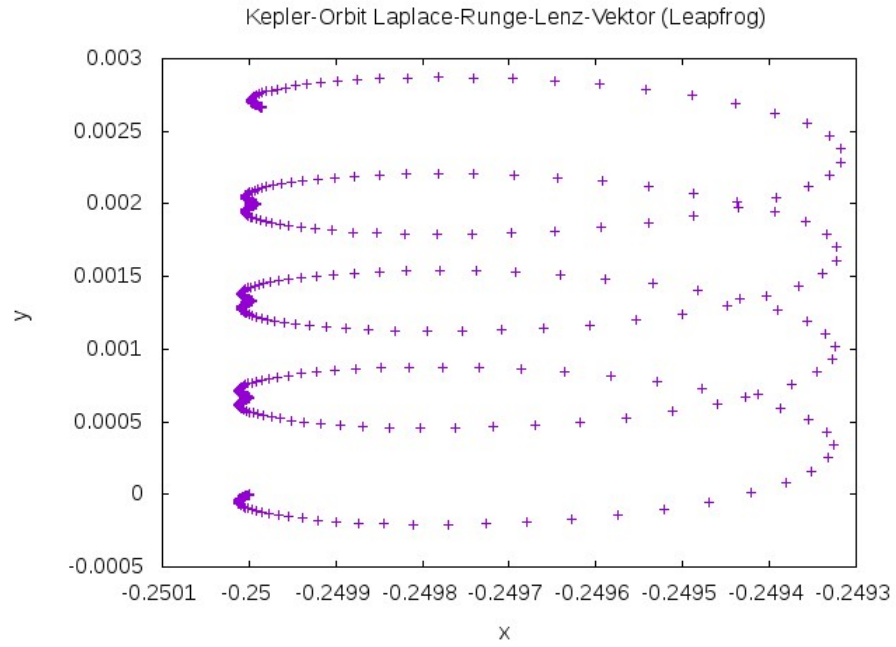
Die Lösung mithilfe des Leapfrog-Verfahrens ergibt:



Kepler-Orbit durch Lösung mit Leapfrog-Verfahren



Entwicklung der Energie des Kepler-Orbits durch Lösung mit Leapfrog-Verfahren



Verhalten des Laplace-Runge-Lenz-Vektors durch Lösung mit Leapfrog-Verfahren

Man kann an den vier Plots erkennen, dass die Energie und der Runge-Lenz-Vektor (und somit auch die Lage der Periapsis) bis auf kleine Abweichungen ($\approx 10^{-6}$ bzw.

$\approx 10^{-3}$) erhalten sind.

Sonstige abgegebene Dateien

2-leapfrog.txt

Ausgabedatei des Leapfrog-Verfahrens aus Programm "2.cpp"

2-runge-kutta.txt

Ausgabedatei des Runge-Kutta-Verfahrens aus Programm "2.cpp"

2.plt

Plotdatei für alle Plots

Aufgabe 3

Allgmeine Hinweise

Das Programm wurde unter Windows 10 mit "g++ -o abgabe55_3.exe -Wall -Wextra -std=c++0x -O2 -static abgabe5_3.cpp" kompiliert.

a) (aufgabe3.cpp)

In der Datei befindet sich die Funktion numerov, die die Schrödingergleichung bei einer Energie e mit 2 Anfangswerten und dem Potential 'potential' löst. Die Funktion gibt ein Array der y-Werte zurück, wobei $y_i = y(h \cdot i)$, h Schrittweite.

Symmetriebetrachtungen

Da die Potentiale in beiden Fällen symmetrisch sind, gibt es nur symmetrische und antisymmetrische Wellenfunktionen. Nummeriert man die Zustände aufsteigend nach ihren Energien durch (beginnend bei 0 für den Grundzustand), so haben alle geraden Zustände symmetrische und alle ungerade Zustände antisymmetrische Wellenfunktionen. Das es nur gerade und ungerade Lösungen gibt, reicht es nur eine Hälfte der Funktion zu berechnen, nämlich $x \in [0, \infty]$.

Anfangswerte

Da die Schrödingergleichung linear ist, reicht es, eine beliebige unnormierte Lösungsfunktion zu einer bestimmten Energie zu finden und später zu normieren. Mit diesen Überlegungen kommt man zu folgenden Anfangsbedingungen:

- symmetrische Lösungen: $y(0) = 1, y'(0) = 0$, bzw. $y_0 = y_1 = 1$. Jede symmetrische Lösung hat Ableitung 0 bei 0. Der Wert $y(0) = 1$ ist vollkommen willkürlich, kann aber später durch normieren nachträglich angepasst werden (siehe b)).
- antisymmetrische Lösungen: $y(0) = 0, y'(0) = 1$, bzw. $y_0 = 0, y_1 = h$, mit h der Schrittweite. Jede antisymmetrische Lösung muss $y(0) = 0$ haben. Der Anstieg $y'(0) = 1$ ist wieder vollkommen willkürlich und wird durch die Normierung nachträglich geändert.

b)

Anmerkung: Alle Energien werden in vielfachen von $\hbar\omega$ angegeben.

Für den harmonischen Oszillator wird als Potential die Funktion 'harmonicOscillator' benutzt. Es wird von 0 bis 10 integriert. Dabei ergibt sich typischerweise folgendes Bild:

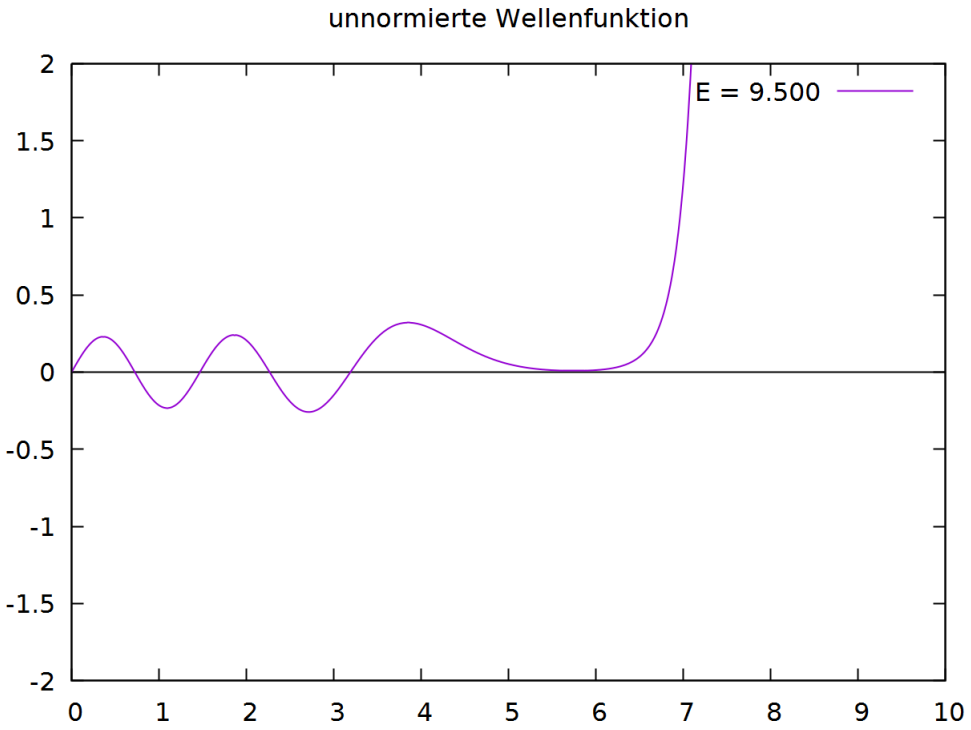


Abbildung 0.1: Unnormierte Wellenfunktion

Man kann erkennen, dass die Funktion gut integriert wird, dann gegen 0 abfällt und dann plötzlich nach ∞ abknickt.

Normierung

Um die Norm der Wellenfunktion zu bestimmen, berechnet man das Integral über das Betragsquadrat der Wellenfunktion. Dies geschieht mit der zusammengesetzten Trapezmethode. Da wir die Funktion nur für positive x kennen, wird das Integral nur für positive x berechnet und danach verdoppelt. Das Integral wird bis zum Abknickpunkt berechnet, da danach die Wellenfunktion nicht mehr realistisch ist. Das Integral geht also von 0 bis zum Abknickpunkt.

Bestimmung des Abknickpunkts

Um den Abknickpunkt zu bestimmen, kann man sich zu nutze machen, dass die n. Eigenfunktion genau Nullstellen hat, bzw. im Intervall $(0, \infty)$ genau $\lfloor n/2 \rfloor$. Danach steigt die Funktion (genauer: deren Betragsquadrat) noch einmal kurz an und fällt dann wie $\exp -x^2$ ab. Beim Abknickpunkt gibt es dann ein lokales Minimum der Funktion.

Der Algorithmus geht jetzt wie folgt vor: Er geht alle berechneten Funktionswerte Schrittweise durch. Der Algorithmus zählt jetzt die Anzahl der Nullstellen (Nulldurchgänge) der Wellenfunktion. Hat er $\lfloor n/2 \rfloor$ Nullstellen gefunden, so sucht er nach dem nächsten lokalen Minimum des Betragsquadrates der Wellenfunktion. An diesem Punkt wird dann die Integration abgebrochen.

Die gesamte Prozedur zur Bestimmung der Norm wird durch die Funktion getNorm(psi, zeros) beschrieben. Dabei ist psi die Wellenfunktion als Array gegeben und zeros die Anzahl der zu erwartenden Nullstellen. Der Rückgabewert ist ein Paar (double, int) mit der Norm und dem Index des Abknickpunktes. In den späteren Graphiken wurde die Wellenfunktion an dem Abknickpunkt abgeschnitten, um eine größere Übersicht zu gewährleisten.

Bestimmung der Grundzustandswellenfunktion

Aus der theoretischen Physik ist bekannt, das die Grundenergie bei 0.5 liegen sollte. In der Umgebung von 0.5 sehen die Wellenfunktionen folgendermaßen aus:

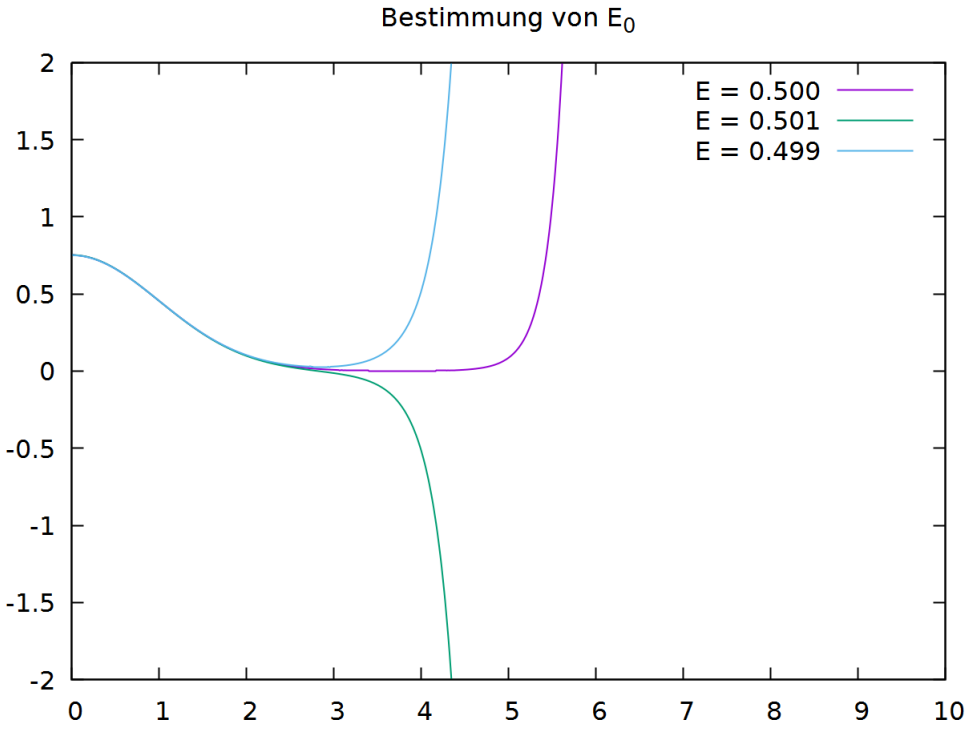


Abbildung 0.2: Bestimmung des Grundzustandes

Ein Energieeigenwert zeichnet sich dadurch aus, dass die Wellenfunktion im unendlichen gegen 0 geht. In dem Bild kann man erkennen, dass die Funktion für $E = 0.5$ gegen unendlich geht und für $E = 0.501$ gegen minus unendlich. Somit muss zwischen beiden Energien eine Lösung der Schrödingergleichung liegen. Also ist die Grundzustandsenergie: $E = 0.5 \pm 0.001$.

Im nächsten Bild ist die Abweichung der Funktion für $E = 0.5$ von der theoretischen Lösung $1/\pi^{(1/4)} \exp -x^2/2$ bis zum Abknickpunkt ($x \approx 3$) dargestellt:

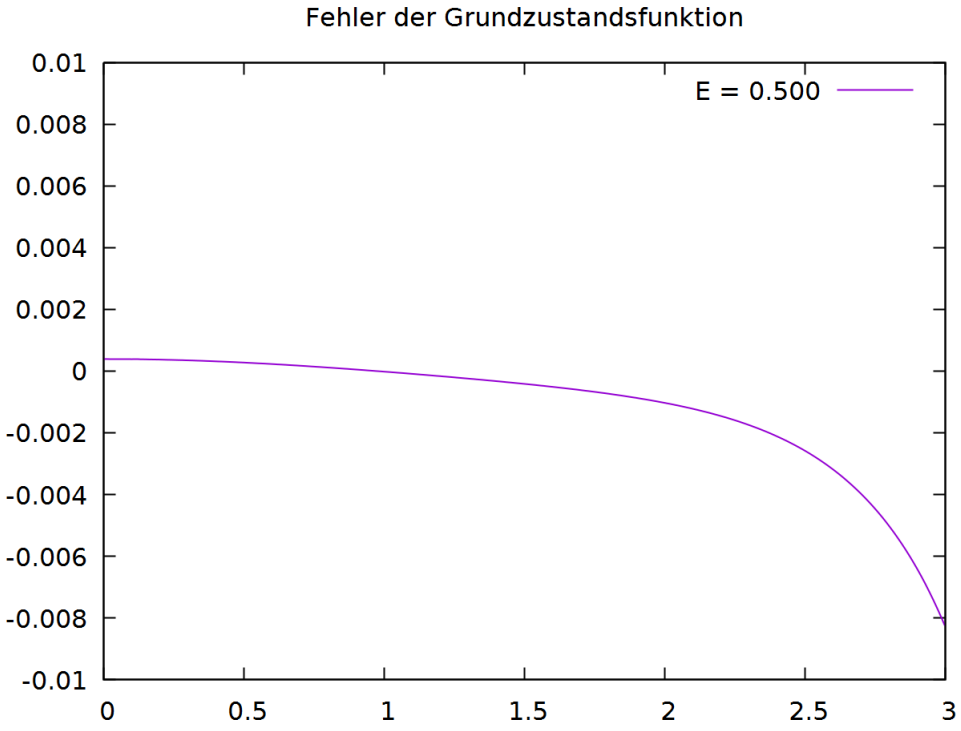


Abbildung 0.3: Fehler des Grundzustandes

Der Fehler ist im Vergleich zu der Funktion recht klein, er wird aber für große x zunehmend größer. Da die Funktion für große x gegen 0 geht, ergibt sich hier ein recht großer relativer Fehler ($\approx 100\%$). Ansonsten scheint das Ergebnis recht gut zu sein.

c)

Man kann erwarten, dass die Energieeigenwerte etwa einen Abstand von $\hbar\omega = 1$ haben. Um die Energien zu bestimmen, geht man immer Energiewerte in kleineren Abständen (hier 0.1) durch und berechnet die zugehörige Wellenfunktion.
Analog zur Bestimmung des Grundzustandes, ist ein Energieeigenwert dadurch gegeben, dass die Funktion im unendlichen gegen 0 geht. Für alle anderen Energien divergieren die Funktionen. Stellt man jetzt bei der Erhöhung der Energie um $E_{i+1} = E_i + 0.1$ fest, dass sich das Divergenzverhalten von $-\infty$ zu ∞ oder andersherum ändert, so liegt im Intervall $[E_i, E_{i+1}]$ ein Energieeigenwert.
Auf diesem Intervall wird dann ein Bisektionsverfahren durchgeführt. Als Vergleichskriterium wird immer der Wert der Wellenfunktion bei 10 genommen. Dessen Vorzeichen bestimmt, ob die Funktion gegen + oder - ∞ geht. Das Bisektionsintervall wird immer so angepasst, dass der linke Rand und der rechte Rand ein unterschiedliches Divergenzverhalten haben. Die Bisektion wird abgebrochen, wenn das Intervall kleiner als 0.001

geworden ist. Somit hat man einen Energieeigenwert auf 0.001 genau bestimmt.

Dieses Verfahren wird durch die Funktion 'sweep()' realisiert. Es gibt ein Array von Paaren (double, bool) zurück. Der erste Wert ist der Energiewert, der zweite Wert ist wahr, wenn es sich um eine symmetrische Funktion handelt, sonst falsch.

d)

Jetzt wird das Potential 'anharmonicOscillator' verwendet. Die Energieeigenwerte sind: 0.885, 3.043, 5.800, 8.929, 12.357, 16.038, 19.939, 24.036, 28.315, 32.758:

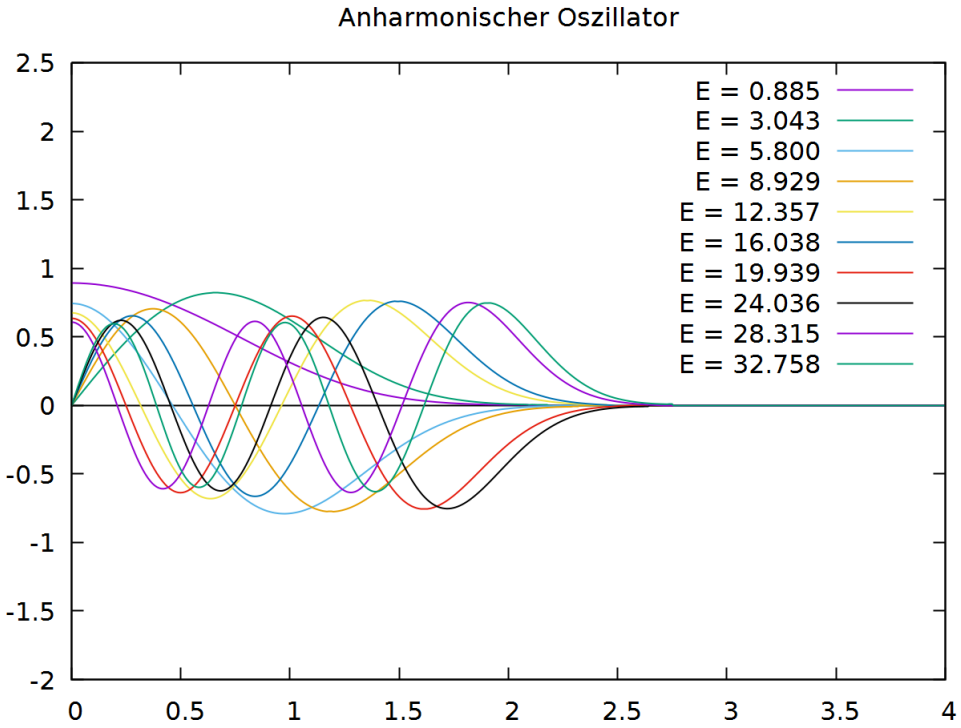


Abbildung 0.4: Die ersten 5 symmetrischen und antisymmetrischen Lösungsfunktionen

Sonstige abgegebene Dateien

plot_harmonic.plt, plot_sweep_perturbation.plt

Die Plot-Dateien für das normale und das gestörte Potential

out_[E].txt

Die Ausgabedateien des Programmes. Entspricht immer einer Wellenfunktion mit der Energie E/1000.

out_sweep_perturbation_[E].txt

Das gleiche wie oben, nur für den gestörten Oszillator