Computerphysik I: Blatt 05

Aurel Müller-Schönau und Leon Oleschko

8. Juli 2022

Aufgabe 6 - Shooting-Methode

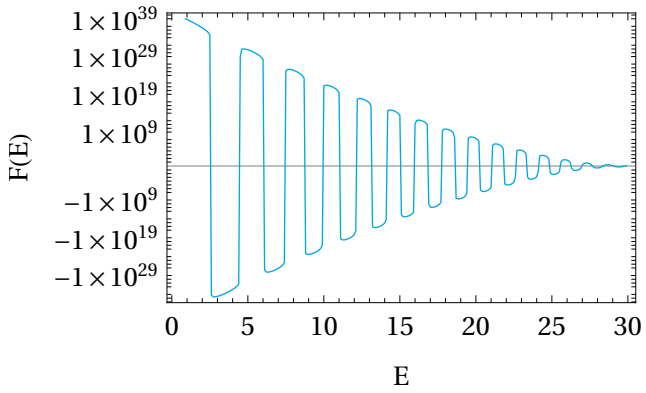

Abbildung 1: Fehlerfunktion für verschiedene Energien

In dieser Aufgabe sollte mit der Shooting Methode die Schrödingergleichung für ein lineares Potential $V(x \geq 0) = x$ gelöst werden. Dabei wurde von $x = \infty$, bzw. X_MAX $= 30$, bis $x = 0$ mit der Numerov-Methode integriert. Für die erste Randbedingung gilt $\psi(\infty \approx \text{X\_MAX}) = 0$, da das Potential unendlich ist. Für die andere Seite gilt $\psi(0) = 0$, da $V(x < 0) = \infty$. Damit diese erfüllt ist, muss die Fehlerfunktion $F(E) = \psi_E(0)$ eine Nullstelle haben.

In Abbildung 1 ist diese Fehlerfunktion gegen die Energie E gezeichnet. Die Daten wurden mit dem Program 1 generiert.

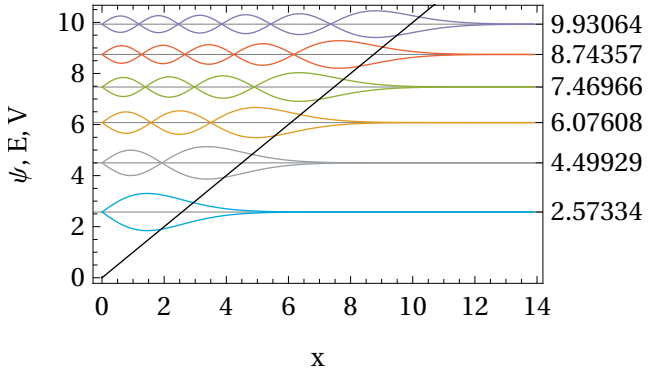

Abbildung 2: Potential mit verschiedenen Wellenfunktionen

Da die Fehlerfunktion stetig ist, wird in Program 2 das Bisektions-Verfahren verwendet, um die Nullstellen zu finden. Dies wird ausgehen von Schätzungen gemacht, die aus dem Abbildung 1 abgelesen wurden. Nachdem das Intervall klein genug ist, wird die Wellenfunktion (im Array psi gespeichert) normiert und abgespeichert.
Da dieser Prozess für alle Schätzungen gemacht werden muss wurde dies parallelisiert.

In Abbildung 2 sind die so erhaltenen ersten 5 Wellenfunktionen und Energien dargestellt.
s Weitere 25 Wellenfunktionen sind in Abbildung 3 zu sehen. Die Eigenenergien sind in der Abbildung 4 dargestellt.

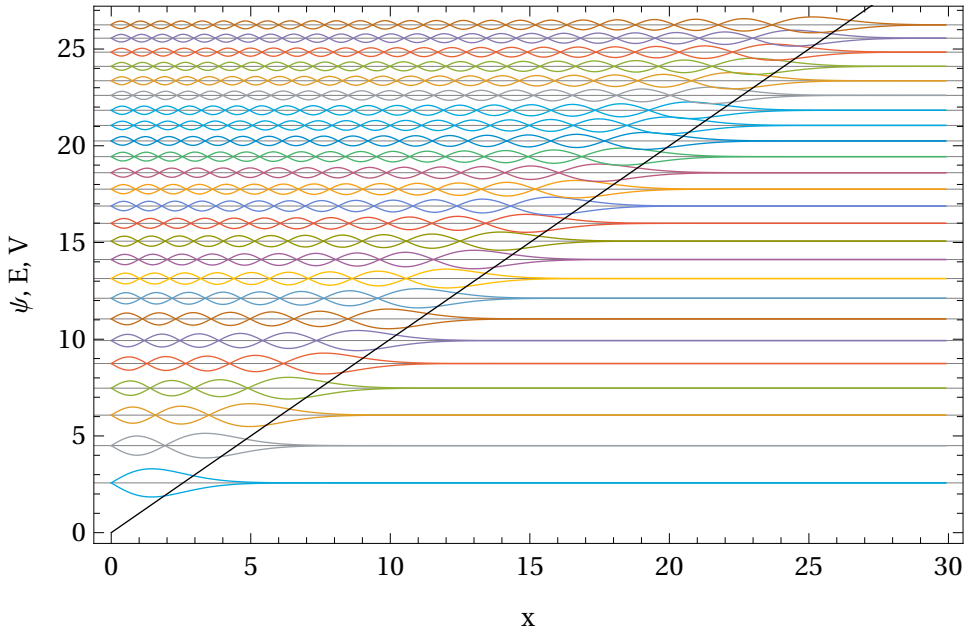

Abbildung 3: die ersten 25 Wellenfunktionen

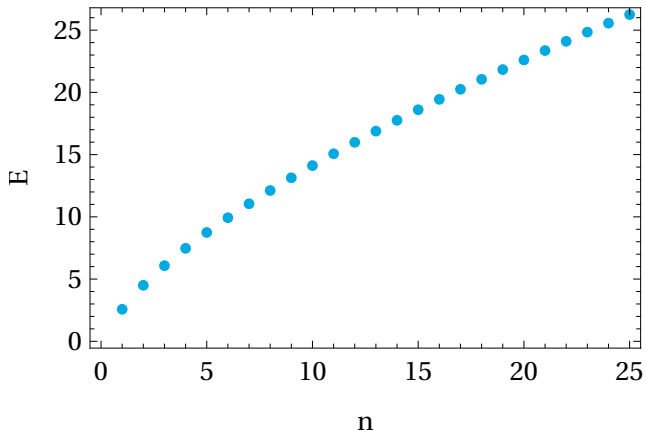

Abbildung 4: Trend der Energieniveaus

Code

6

Listing 1: Programm zum berechnen der Error-Funktion für verschiedene Energien.

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

#define H 1.e-4
#define XMAX 30.

void integ(double E);
void numerov(double *y1, double *y2, double x, double E);
double V(double x);
FILE *file;

int main(){
  file = fopen("errorFkt.dat", "w+");
  for(double E = 0.0; E < XMAX; E += 0.1){
    integ(E);
  }
  fclose(file);
  return 0;
}

void integ(double E){
  double y1 = 0, y2 = H;
  for(double x = XMAX; x > 0; x -= H){
    numerov(&y1, &y2, x, E);
  }
  fprintf(file, "%f %f \n", E, y2);
}

void numerov(double *y1, double *y2, double x, double E){
  double y = 2* *y2*(1 - H*H*5/12*(E-V(x))) - *y1*(1 - H*H/12*(E-V(x-H)))
    ;
  *y1 = *y2;
  *y2 = y;
}

double V(double x){
  return x;
}
```

Listing 2: Programm zum parallelisierten Berechnen der Wellenfunktionen mit der Shooting-;ethode.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <stdbool.h>
#include <pthread.h>

#define H 1.e-4
#define X_MAX 30.
//#define X_MAX 14.
#define N ((int) (X_MAX/H))
#define EXPORT_STEPS ((int) (0.1/H))
#define UNCERT_END 1.e-12

#define UNCERT_GUESS 0.25
double guess[] = {2.5, 4.5, 6, 7.5, 8.5, 10, 11, 12, 13, 14, 15, 16, 17,
    18., 18.5, 19.5, 20., 21, 22, 22.5, 23.5, 24, 25.,25.5, 26.};
//double guess[] = {2.5, 4.5, 6, 7.5, 8.5, 10};

void integ(double E, double* psi);
void numerov(double *y1, double *y2, double x, double E);
void *findAndExport(void *argument);
double V(double x);


int main(){
  const uint NUM_THREADS = sizeof(guess)/sizeof(double);
  int thread_args[NUM_THREADS];
  pthread_t threads[NUM_THREADS];

  // create a thread for each guess
  for (int i = 0; i < NUM_THREADS; i++){
      thread_args[i] = i;
    pthread_create(&threads[i], NULL, findAndExport, &thread_args[i]);
  }

  //wait for each thread to complete
  for (size_t i = 0; i < NUM_THREADS; i++) {
    pthread_join(threads[i], NULL);
  }


  return 0;
```

```
42 }
43
44 // find the root of the error function near the n-th guess and save the
       generated wave function
45 void *findAndExport(void *argument){
46   // get the index of the thread
47   const int n = *((int *)argument);
48
49   // create array for the wave Function
50   // in the heap because the stack is too small
51   double* psi = malloc(N*sizeof(double));
52
53   // bisection to find the root
54   double E0 = guess[n]-UNCERT_GUESS;
55   double E1 = guess[n]+UNCERT_GUESS;
56   double Ex, tmp;
57   do {
58     Ex = (E1+E0)/2.0;
59
60     integ(Ex, psi);
61     tmp = psi[0];
62     integ(E0, psi);
63
64     if(tmp*psi[0]>0)
65       E0=Ex;
66     else
67       E1=Ex;
68   } while (fabs((E1-E0)/E0)>UNCERT_END);
69
70   printf("guess: %.2f E: %f\n", guess[n], E1);
71
72   // normalizing
73   double norm = 0;
74   for(int i=0; i<N; i++){
75     norm += psi[i]*psi[i]*H;
76   }
77   norm = sqrt(norm);
78   for(int i=0; i<N; i++){
79     psi[i] /= norm;
80   }
81
82   // exporting
83   FILE *file;
84   char filename[210];
```

```
85    snprintf(filename, 10, "out%02d.dat", n);
86    printf("saving in: %s\n", filename);
87    file = fopen(filename, "w+");
88    fprintf(file, "%f\n", E1);
89    for(int i=0; i<N; i+=EXPORT_STEPS){
90      fprintf(file, "%f %f\n", i*H, psi[i]);
91    }
92    fclose(file);
93
94    // end the task successfully
95    return 0;
96  }
97
98  // integrating the wave function
99  void integ(double E, double *psi){
100
101   double y1 = 0, y2 = 1e-10;
102   for(int i=0; i <= N; i++){
103     numerov(&y1, &y2, (N-i)*H, E);
104     psi[N-i] = y2;
105   }
106 }
107
108 // numerov step
109 void numerov(double *y1, double *y2, double x, double E){
110
111   double y = 2* *y2*(1 - H*H*5/12*(E-V(x))) - *y1*(1 - H*H/12*(E-V(x-H)))
        ;
112   *y1 = *y2;
113   *y2 = y;
114 }
115
116
117 double V(double x){
118   return x;
119 }
```