

Computerphysik I: Blatt 05

Aurel Müller-Schönau und Leon Oleschko

8. Juli 2022

Aufgabe 5 - Numerov-Verfahren

a)

In Programm 1 ist die Implementierung von Analytisch

$$\text{Numerisch:} \quad -0.039788 \quad (1)$$

$$\text{Analytisch:} \quad -\frac{1}{8\pi} \approx -0.039788 \quad (2)$$

$$\frac{\partial}{\partial r} \phi \propto -\rho(r) \quad (3)$$

$$\Rightarrow \quad \phi(0) = \int_{\infty}^0 -\rho(r) \, dr^2 + \phi(\infty) \quad (4)$$

$$\approx \int_{100}^0 -\rho(r) \, dr^2 \quad (5)$$

$$\approx \sum_{100}^0 -\rho(r) \, dr^2 \quad (6)$$

$$(7)$$

b)

Aufgabe 6 - Shooting-Methode

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 $\psi(\infty \approx X_MAX) = 0$, da das Potential unendlich ist. Für die andere Seite gilt $\psi(0) = 0$,

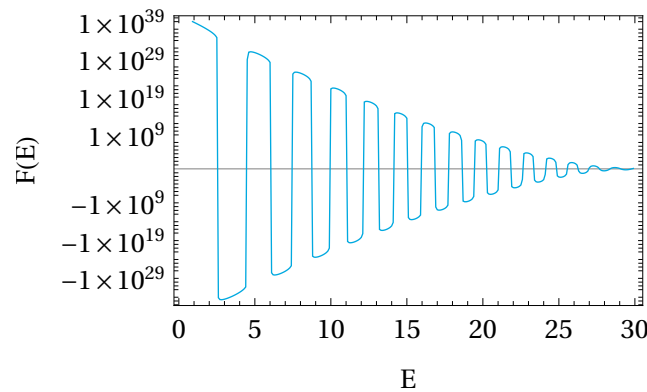


Abbildung 1: Fehlerfunktion für verschiedene Energien

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 2 generiert.

Da die Fehlerfunktion stetig ist, wird in Program 1 das Bisections-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.

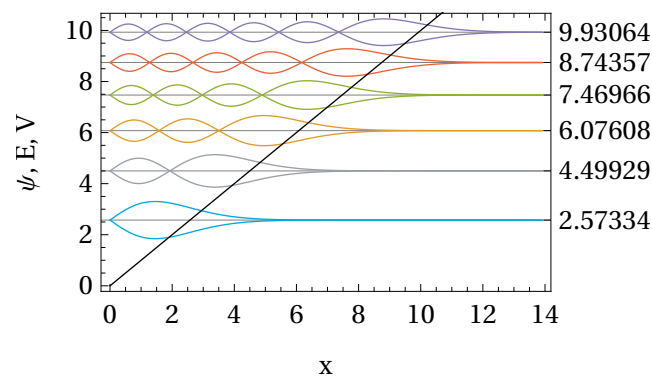


Abbildung 2: Potential mit verschiedenen Wellenfunktionen

In Abbildung 2 sind die so erhaltenen Wellenfunktionen und Energien dargestellt.

In Abbildung 3 sind die ersten 25 Wellenfunktionen dargestellt. Die Eigenenergien sind in der Abbildung 4 dargestellt.

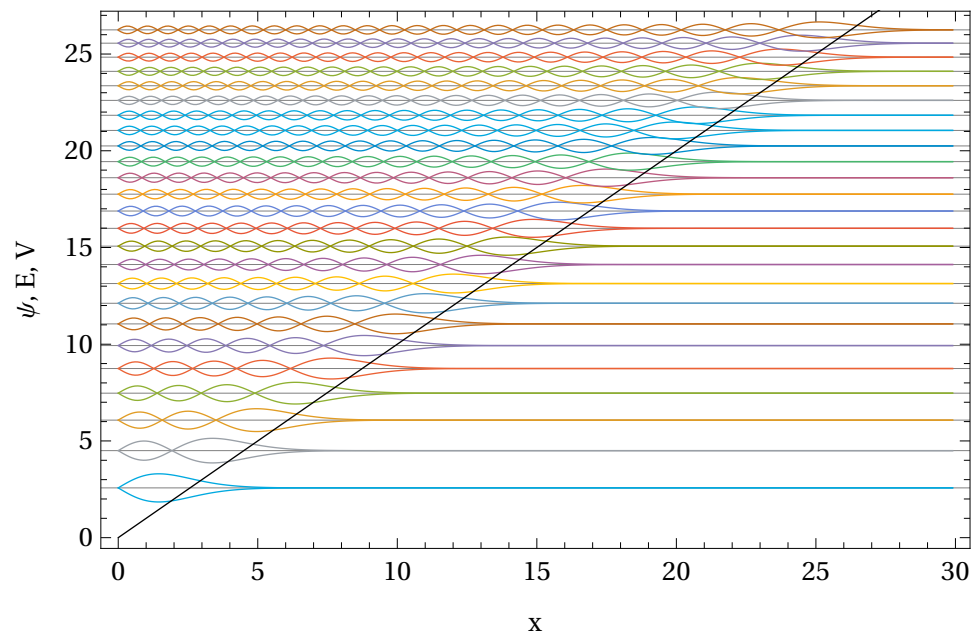


Abbildung 3: die ersten 25 Wellenfunktionen

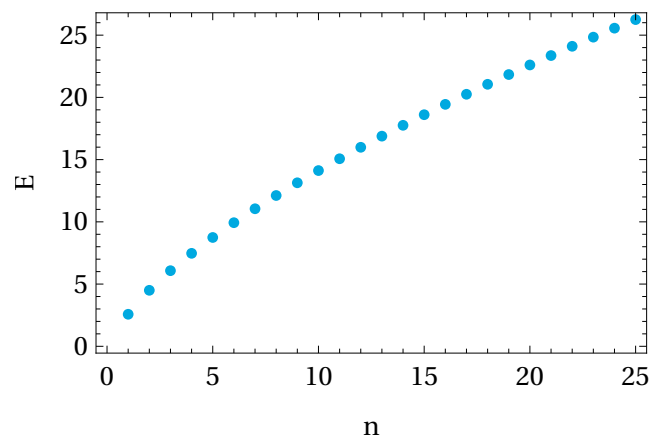


Abbildung 4: Trend der Energieniveaus

Code

5 a)

Listing 1: Programm zum lösen der Poisson-Gleichung mit dem Numerov-Verfahren.

```
1 #include <stdio.h>
2 #include <math.h>
3
4
5 #define H 0.001
6 #define rstart 100
7
8
9 double rho(double r);
10 void numerov(double* y1, double* y2, double r);
11
12
13 int main(){
14
15     double y1=0, y2 = 0;
16
17     for(double r = rstart; r > 0; r -= H){
18         numerov(&y1, &y2, r);
19         printf("%f\n", y2);
20     }
21
22     // Ergebnis: -0.039789
23     // Analytisch: -1/(8 pi) \approx -0.0397887
24
25     return 0;
26 }
27
28
29 double rho(double r){
30     double wert = exp(-r)/(8 * M_PI);
31     return wert;
32 }
33
34 void numerov(double* y1, double* y2, double r){
35     double y = 2 * *y2 - *y1 - H*H/12*(rho(r+H)+10*rho(r)+rho(r-H));
36     *y1 = *y2;
37     *y2 = y;
38 }
```

6

Listing 2: Programm zum aufzeichnen der Error-Funktion für verschiedene Energien.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4
5 #define H 1.e-4
6 #define XMAX 30.
7
8 void integ(double E);
9
10 void numerov(double *y1, double *y2, double x, double E);
11
12 double V(double x);
13
14 FILE *file;
15
16 int main(){
17
18     file = fopen("errorFkt.dat", "w+");
19
20
21     for(double E = 0.0; E < XMAX; E += 0.1){
22         integ(E);
23     }
24
25     //integ(5.0);
26
27
28
29     fclose(file);
30
31     return 0;
32 }
33
34
35 void integ(double E){
36
37     double y1 = 0, y2 = H;
38     for(double x = XMAX; x > 0; x -= H){
39         numerov(&y1, &y2, x, E);
40         //fprintf(file, "%f %f \n", x, y1);
41     }
```

```
42     fprintf(file, "%f %f \n", E, y2);
43 }
44
45 void numerov(double *y1, double *y2, double x, double E){
46
47     double y = 2* *y2*(1 - H*H*5/12*(E-V(x))) - *y1*(1 - H*H/12*(E-V(x-H)
48     ));
49     *y1 = *y2;
50     *y2 = y;
51 }
52
53 double V(double x){
54     return x;
55 }
```

Listing 3: ...

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4 #include <stdbool.h>
5 #include <pthread.h>
6
7 #define H 1.e-4
8 #define X_MAX 30.
9 // #define X_MAX 14.
10 #define N ((int) (X_MAX/H))
11 #define EXPORT_STEPS ((int) (0.1/H))
12 #define UNCERT_END 1.e-12
13
14 #define UNCERT_GUESS 0.25
15 double guess[] = {2.5, 4.5, 6, 7.5, 8.5, 10, 11, 12, 13, 14, 15, 16, 17,
16     18., 18.5, 19.5, 20., 21, 22, 22.5, 23.5, 24, 25., 25.5, 26.};
17 // double guess[] = {2.5, 4.5, 6, 7.5, 8.5, 10};
18
19 void integ(double E, double* psi);
20 void numerov(double *y1, double *y2, double x, double E);
21 void *findAndExport(void *argument);
22 double V(double x);
23
24 int main(){
25     const uint NUM_THREADS = sizeof(guess)/sizeof(double);
26     int thread_args[NUM_THREADS];
27     pthread_t threads[NUM_THREADS];
28
29     // create a thread for each guess
30     for (int i = 0; i < NUM_THREADS; i++){
31         thread_args[i] = i;
32         pthread_create(&threads[i], NULL, findAndExport, &thread_args[i]);
33     }
34
35     // wait for each thread to complete
36     for (size_t i = 0; i < NUM_THREADS; i++) {
37         pthread_join(threads[i], NULL);
38     }
39
40
41     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     sprintf(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     ;
113     *y1 = *y2;
114     *y2 = y;
115 }
116
117 double V(double x){
118     return x;
119 }
```