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

$$\text{Numerisch:} \qquad -0.039788 \tag{1}$$

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

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

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

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

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

$$\tag{7}$$

b)

Aufgabe 6 - Shooting-Methode

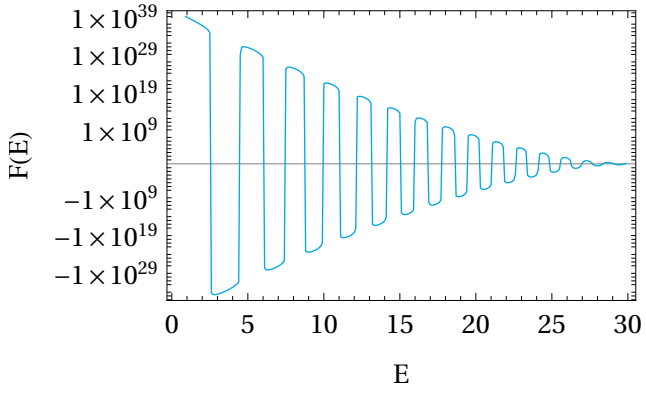

Abbildung 1: Fehlerfunktion für verschiedene Energien

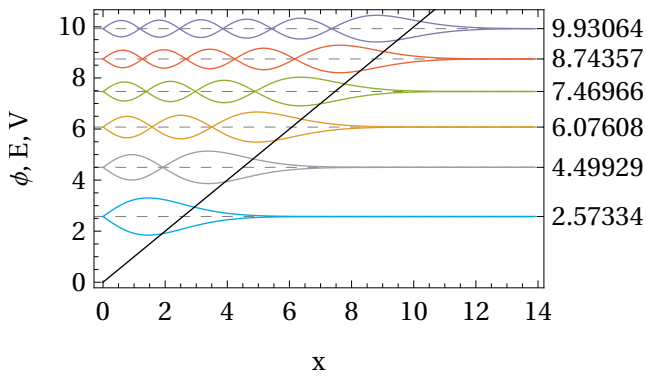

Abbildung 2: Potential mit verschiedenen Wellenfunktionen

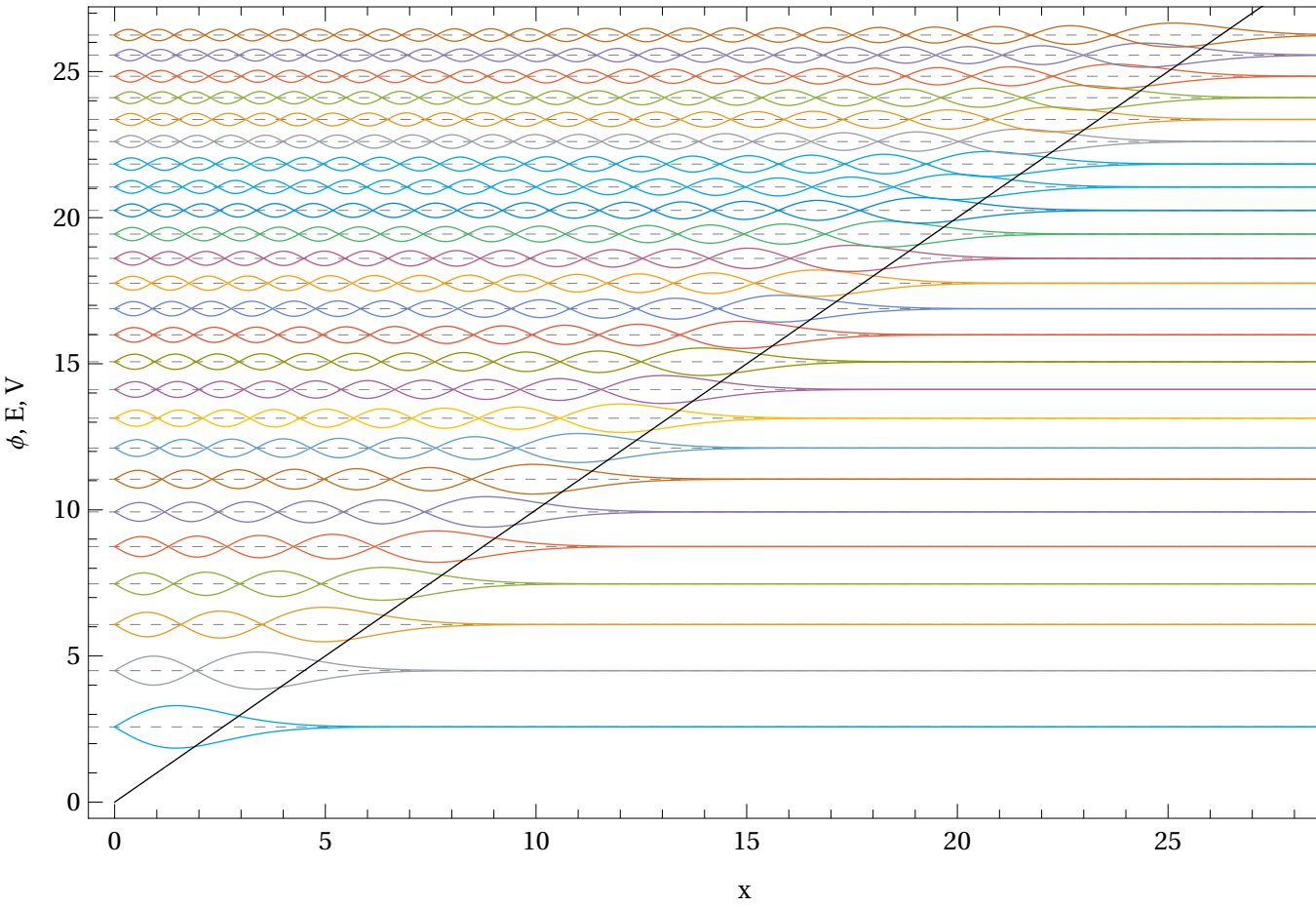

Abbildung 3: die ersten 25 Wellenfunktionen

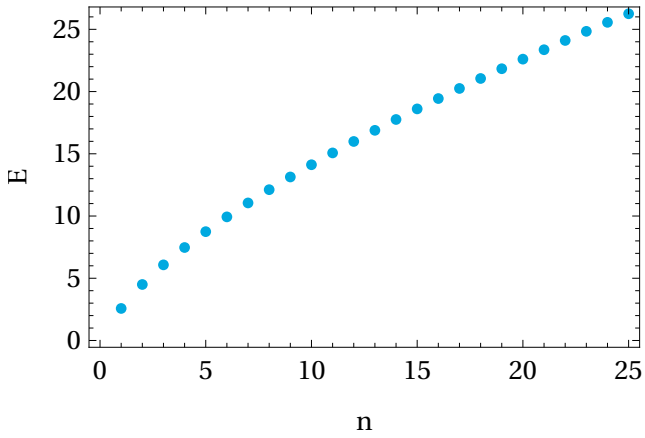

Abbildung 4: Trend der Energieniveaus

Code

5 a)

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

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


#define H 0.001
#define rstart 100


double rho(double r);
void numerov(double* y1, double* y2, double r);


int main(){

    double y1=0, y2 = 0;

    for(double r = rstart; r > 0; r -= H){
        numerov(&y1, &y2, r);
        printf("%f\n", y2);
    }

    // Ergebnis:                          -0.039789
    // Analytisch: -1/(8 pi) \approx      -0.0397887

    return 0;
}


double rho(double r){
    double wert = exp(-r)/(8 * M_PI);
    return wert;
}

void numerov(double* y1, double* y2, double r){
    double y = 2 * *y2 - *y1 - H*H/12*(rho(r+H)+10*rho(r)+rho(r-H));
    *y1 = *y2;
    *y2 = y;
}
```

6

Listing 2: Programm zum aufzeichnen 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);
    }

    //integ(5.0);



    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", x, y1);
    }
```

```
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      *y1 = *y2;
49      *y2 = y;
50  }
51
52
53  double V(double x){
54      return x;
55  }
```

Listing 3: ...

```
#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* phi);
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;
}
```

```
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* phi = 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, phi);
61     tmp = phi[0];
62     integ(E0, phi);
63
64     if(tmp*phi[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 += phi[i]*phi[i]*H;
76   }
77   norm = sqrt(norm);
78   for(int i=0; i<N; i++){
79     phi[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, phi[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 *phi){
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     phi[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 }
```