

# Lösungen Computerphysik, 2. Auflage 2019

## (Kapitel 2-12)

Stand: 17. Juli 2019

### Kapitel 2

#### Aufgabe 2.1

Finden Sie heraus, aus welchen Bestandteilen (CPU, RAM etc.) Ihr Rechner aufgebaut ist, ohne ihn zu zerlegen. Welche sind die wichtigsten Kenngrößen der einzelnen Komponenten?

GUI: KDE Info-Center

Linux-Kommandos (Auswahl): `lscpu`, `lshw`, `hwinfo`, `lspci`, `lsusb`, `df`, `mount`, `free`, `hdparm`

Kenngrößen (Beispiele): CPU-Frequenz (in GHz) und Anzahl Kerne, RAM-Größe (in GByte), Festplattengröße in TByte

#### Aufgabe 2.2

Die **Ausfallwahrscheinlichkeit** einer Festplatte kann mit der Formel

$$p(t) = 1 - e^{-t/MTBF} \quad (1)$$

berechnet werden.  $t$  ist die Laufzeit und  $MTBF$  (*mean time between failures*) die „Zerfallskonstante“. Mit welcher Wahrscheinlichkeit fällt also eine Festplatte mit  $MTBF = 2,5 \cdot 10^6$  Stunden während der Lebensdauer von 5 Jahren aus?

$2,5 \cdot 10^6$  h sind ca 285 Jahre. Damit ist

$$p(t) = 1 - \exp(-5/285) = 0,0174 \approx 1,7\%.$$

Die Ausfallwahrscheinlichkeit innerhalb von 5 Jahren beträgt nach dieser einfachen Formel als ca. 1,7 % bzw. 17 von 1000.

#### Aufgabe 2.3

Welche Betriebssysteme haben Sie schon verwendet? Welche Vor- und Nachteile können Sie nennen?

Auf PCs sind typisch: Windows (7, 8, 10), macOS (10.12, 10.13, 10.14), Linux (Desktop-Distribution)

Auf Servern: Linux (Server-Distribution), Unix, Windows Server

Auf Smartphones: Android, iOS

#### Aufgabe 2.4

Welche sind die aktuellen Versionen der wichtigsten Linux-Distributionen? Welche verwenden Sie?

Reihenfolge sehr individuell (siehe z.B. <http://distrowatch.org/>):

Desktop: Ubuntu 19.04, Linux Mint 19.1, Debian 10, Fedora 30, OpenSUSE 15.1, Arch Linux (Rolling Release), Manjaro 18.0.4

Server: SLES 15 (SP1), RHEL 8.0

## Aufgabe 2.5

Welche Desktop-Umgebung bzw. Fenstermanager verwenden Sie auf Ihrem Rechner? Wechseln Sie auf ein virtuelles Terminal und zurück auf die grafische Oberfläche. Öffnen Sie auch dort ein (Pseudo-)Terminal.

Am verbreitetsten: KDE, Gnome (inkl. MATE und Cinnamon), Unity, XFCE, LXDE

Wechsel in virtuelles Terminal: STRG+F1

Wechsel in grafische Oberfläche: STRG+F7 bzw. F8

Pseudo-Terminal in GUI (z. B.): Konsole, Gnome-Terminal

## Kapitel 3

### Aufgabe 3.1

Öffnen Sie ein Terminal auf Ihrem Linux-System. Ändern Sie die Umgebungsvariable *PS1* und speichern Sie das aktuelle Datum in der Variable *DATE*.

```
$ PS1="X "  
X DATE=$(date)  
X echo $DATE  
Tue Jul 9 ...
```

### Aufgabe 3.2

Wie sieht bei Ihnen die Umgebungsvariable *PATH* aus? Ändern Sie diese Variable, damit auch im jeweils aktuellen Pfad gesucht wird.

```
$ echo $PATH  
/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:...  
$ PATH=".: $PATH"  
$ echo $PATH  
./:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:...
```

### Aufgabe 3.3

Erklären Sie die Ausgabe des Kommandos *ls -l*.

```
-rw-r--r--  1 gerlach admin    129 Dec  8 2018 Datei  
drwxr-xr-x  2 gerlach admin  4096 Mar 20 20:53 Verzeichnis
```

*ls -l* gibt Informationen zu allen Dateien (reguläre Dateien, Verzeichnisse, Links etc.) im aktuellen Verzeichnis aus. Das sind: Rechte(Spalte 1), Links(2), Besitzer(3), Gruppe(4), Größe(5), Änderungsdatum(6) und Name(7).

### Aufgabe 3.4

Verwenden Sie die Kommandos *mkdir*, *cd*, *cp*, *rm* und *rmdir*, um ein Verzeichnis anzulegen, dort Dateien zu kopieren und zu löschen und am Ende aufzuräumen.

```
$ mkdir dir  
$ cd dir  
(dir/$ touch test)  
dir/$ cp test test2
```

```
dir/$ rm test*
dir/$ cd ..
$ rmdir dir
```

### Aufgabe 3.5

Erklären Sie den Unterschied zwischen `cd tmp` und `cd /tmp`. Was machen die Kommandos `cd ..` und `cd ~`?

`cd tmp` wechselt in das lokale Verzeichnis `tmp` (falls vorhanden im aktuellen Verzeichnis) und `cd /tmp` in das globale Verzeichnis `tmp`. `cd ..` wechselt in das tieferliegende Verzeichnis und `cd ~` in das Homeverzeichnis.

### Aufgabe 3.6

Mit dem Kommando `chmod` kann man die Zugriffsrechte von Dateien ändern. Erklären Sie die Syntax dazu und probieren Sie es aus.

Beispiele:

`chmod u+x datei`: Besitzer ((u)ser) erhält Recht zum Ausführen (e(x)ecute) für Datei `datei`.

`chmod g+w datei`: Gruppe ((g)roup) erhält Schreibrecht (w) für Datei `datei`.

`chmod o-r datei`: Andere ((o)thers) wird Leserecht für Datei `datei` entzogen.

### Aufgabe 3.7

Welche Informationen zeigt das Programm `top` in den Kopfzeilen? Was geben die Spalten an?

(Anzeige kann je nach Optionen und Version variieren)

Kopfzeile: Uhrzeit, Laufzeit, Anzahl Benutzer, Load—Anzahl Prozesse nach Status—CPU-Nutzung—Speicherstatistik—Auslagerungsspeicher

Spalten zeigen die Prozessliste an: Prozess-ID, Besitzer, Priorität+Nice-Wert, Speicherverbrauch, Status, CPU- und Speichernutzung, Laufzeit, Aufrufkommando

### Aufgabe 3.8

Starten Sie das Programm `firefox` im Hintergrund einer Shell. Halten Sie das Programm an (SIGSTOP) und lassen Sie es dann weiterlaufen (SIGCONT). Finden Sie die PID des Prozesses heraus und beenden Sie das Programm mit dem Kommando `kill`.

```
$ firefox &
[1] 23604
$ kill -19 %1
[1]+  Stopped  firefox
$ kill -18 %1
$ ps |grep firefox
$ 23604 pts/52    00:00:01 firefox
$ kill 23604
```

# Kapitel 4

## Aufgabe 4.1

Schreiben Sie ein eigenes „Hallo Welt!“-Programm in C, kompilieren und testen Sie es. Quellcode siehe Listing 4.1.

```
$ gcc -o hallo hallo.c
$ ./hallo
Hallo Welt!
```

## Aufgabe 4.2

Schreiben Sie ein C-Programm, das die Größe aller **Datentypen** ausgibt.

```
#include <stdio.h>

int main() {
    printf("char: %d Byte\n", sizeof(char));
    printf("short: %d Byte\n", sizeof(short));
    printf("int: %d Byte\n", sizeof(int));
    printf("long: %d Byte\n", sizeof(long));
    printf("float: %d Byte\n", sizeof(float));
    printf("double: %d Byte\n", sizeof(double));
    printf("long double: %d Byte\n", sizeof(long double));

    return 0;
}

.....

char: 1 Byte
short: 2 Byte
int: 4 Byte
long: 8 Byte
float: 4 Byte
double: 8 Byte
long double: 16 Byte
```

## Aufgabe 4.3

Schreiben Sie ein C-Programm, das die Zahlen von 1 bis 42 mit einer *for*-, *while*- und einer *while-do*-Schleife ausgibt.

```
#include <stdio.h>

int main() {
    const int limit = 42;
    int i;

    for (i = 1; i <= limit; i++)
        printf("%d ", i);
    puts("");

    i = 1;
    while (i <= limit)
        printf("%d ", i++);
    puts("");

    i = 1;
    do
        printf("%d ", i++);
    while (i <= limit);
    puts("");

    return 0;
}
```

## Aufgabe 4.4

Schreiben Sie ein C-Programm, das die **Fakultät**  $n!$  einer eingegebenen Zahl  $n$  berechnet. Erweitern Sie das Programm, sodass es bei Eingabe der Werte  $n$  und  $k$  den **Binomialkoeffizienten**  $\binom{n}{k}$  berechnet. Achten Sie auf alle Sonderfälle. Ändern Sie das Programm nun so, dass es das **Pascal'sche Dreieck** bis zu einem festen  $n$  berechnet und ausgibt.

```
#include <stdio.h>

long fak(int n) {
    return n == 0 ? 1 : n * fak(n-1);
}

long bk(int n, int k) {
    return fak(n)/(fak(k)*fak(n-k));
}

void pascal(int nmax) {
    const int breite = 3;
    int n, k;
    for (n = 0; n < nmax; n++) {
        for (k = 1; k < (nmax-n)*breite; k++)
            printf(" ");
        for (k = 0; k <= n; k++)
            printf("%5ld ", bk(n, k));
        puts("");
    }
}

int main(int argc, char *argv[]) {
    if (argc != 3) {
        printf("Aufruf: %s n k\n", argv[0]);
        return -1;
    }

    int n = atoi(argv[1]);
    int k = atoi(argv[2]);
    if (n < 0 || n > 20 || k < 0 || k > 20 || n < k) {
        printf("n,k < 0 oder n,k > 20 oder k > n nicht erlaubt\n");
        return -2;
    }

    printf("%d! = %ld\n", n, fak(n));
    printf("n ueber k = %ld\n", bk(n, k));

    pascal(n);
}
```

```
.....
$ ./pascal 13 3
13! = 6227020800
n ueber k = 286
```

```

              1
            1 1
          1 2 1
        1 3 3 1
      1 4 6 4 1
    1 5 10 10 5 1
  1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
  1 8 28 56 70 56 28 8 1
    1 9 36 84 126 126 84 36 9 1
      1 10 45 120 210 252 210 120 45 10 1
        1 11 55 165 330 462 462 330 165 55 11 1
          1 12 66 220 495 792 924 792 495 220 66 12 1
```

## Aufgabe 4.5

Schreiben Sie ein C-Programm, das eine Umrechnungstabelle von Celsius in Fahrenheit und Kelvin für 0 bis 100 Grad Celsius ausgibt.

```
#include <stdio.h>

int main() {
    double T_C, T0K = -273.15;
    puts("T in Celsius\tT in Kelvin\tT in Fahrenheit");
    for (T_C = 0.; T_C <= 100.; T_C++) {
        printf("%8g\t%8.2f\t%8.2f\n", T_C, T_C - T0K, T_C * 1.8 + 32.);
    }

    return 0;
}
```

```
.....
T in Celsius      T in Kelvin      T in Fahrenheit
      0           273.15           32.00
      1           274.15           33.80
      2           275.15           35.60
      3           276.15           37.40
      4           277.15           39.20
      5           278.15           41.00
      6           279.15           42.80
      7           280.15           44.60
      8           281.15           46.40
      9           282.15           48.20
     10           283.15           50.00
...

```

## Aufgabe 4.6

Schreiben Sie einen **Makefile**, der automatisch alle bisherigen Programme übersetzt. Legen Sie außerdem ein Git-Repository für das Programmverzeichnis an.

```
CC=gcc
all: types loop pascal temp

%: %.c
    ${CC} -o $@ $<

.....

$ git init
Initialized empty Git repository in .git/
$ git add *.c Makefile
$ git commit -m "Neues Projekt"
[master (root-commit) fc47840] Neues Projekt
5 files changed, 90 insertions(+)
 create mode 100644 Makefile
 create mode 100644 loop.c
 create mode 100644 pascal.c
 create mode 100644 temp.c
 create mode 100644 types.c
$ git hist
* fc47840 2019-07-11 | Neues Projekt (HEAD -> master) [Stefan Gerlach]
```

## Kapitel 5

### Aufgabe 5.1

Schreiben Sie ein Python-Programm, das eine Umrechnungstabelle von Celsius in Fahrenheit und Kelvin für 0 bis 100 Grad Celsius ausgibt.

```
print("T in Celsius\tT in Kelvin\tT in Fahrenheit")
T0K = -273.15
for T_C in range(0, 101):
    print("%8g\t%8.2f\t%8.2f" % (T_C, T_C - T0K, T_C * 1.8 + 32.))
```

```

.....
T in Celsius      T in Kelvin      T in Fahrenheit
    0             273.15           32.00
    1             274.15           33.80
    2             275.15           35.60
    3             276.15           37.40
    4             277.15           39.20
    5             278.15           41.00
...

```

## Aufgabe 5.2

Schreiben Sie ein Python-Programm, das die Differenz zwischen zwei **Zeiten** (z. B. 12:34:12 und 13:17:12) berechnet und ausgibt.

```

from datetime import datetime

t1 = datetime.strptime("12:34:12", "%H:%M:%S")
t2 = datetime.strptime("13:17:12", "%H:%M:%S")

print("Zeitdifferenz:", t2 - t1)
.....
Zeitdifferenz: 0:43:00

```

## Aufgabe 5.3

Schreiben Sie ein Python-Programm, das eine binäre Zahl in eine Dezimalzahl umrechnet. Erweitern Sie das Programm auch für die umgekehrte Umrechnung.

```

bin_str = '101010'
n = int(bin_str, 2)
print(bin_str, ":", n)

dec = 200
print(dec, ":", bin(dec))
.....
101010 : 42
200 : 0b11001000

```

## Aufgabe 5.4

### Vektoren und Matrizen

- (a) Schreiben Sie ein Python-Programm, das den **Betrag** eines reellen Vektors  $\begin{pmatrix} x \\ y \\ z \end{pmatrix}$  berechnet und den normierten Vektor ausgibt.
- (b) Erweitern Sie das Programm, sodass es den **Winkel** zwischen zwei Vektoren berechnet.
- (c) Erweitern Sie das Programm, sodass es einen Vektor um einen beliebigen Winkel und eine der drei Hauptachsen dreht. Überprüfen Sie das Ergebnis, indem Sie den Winkel zwischen den ursprünglichen und dem gedrehten Vektor berechnen.

```

from math import sqrt, acos, pi, sin, cos, degrees
from numpy import array

def sp(v1, v2):
    return v1[0]*v2[0] + v1[1]*v2[1] + v1[2]*v2[2]

def vnorm(v):
    return sqrt(sp(v, v))

def normalize(v):
    return v/vnorm(v)

def angle(v1, v2):
    return acos(sp(v1, v2)/(vnorm(v1)*vnorm(v2)))

# rotate around x axis
def rotate_x(v, angle):
    x = v[0]
    y = cos(angle)*v[1] + sin(angle)*v[2]
    z = -sin(angle)*v[1] + cos(angle)*v[2]
    return array([x, y, z])

v = array([1., 2., 3.])
print("v normiert:", normalize(v))

v1 = array([1, 0, 0])
v2 = array([0, 1, 0])
print("Winkel zw. v1 und v2 (Grad):", degrees(angle(v1, v2)))

rangle = pi/2.
v2r = rotate_x(v2, rangle)
print("v2 um x um ", degrees(rangle), "Grad rotiert:", v2r.round(5))
print("Winkel (Grad):", degrees(angle(v2, v2r)))

.....

v normiert: [0.26726124 0.53452248 0.80178373]
Winkel zw. v1 und v2 (Grad): 90.0
v2 um x um 90.0 Grad rotiert: [ 0.  0. -1.]
Winkel (Grad): 90.0

```

## Kapitel 6

### Aufgabe 6.1

Schreiben Sie ein Python-Programm zur Umrechnung der **Polardarstellung** komplexer Zahlen in die kartesische Darstellung. Implementieren Sie auch die umgekehrte Umrechnung.

```

from cmath import rect, pi, polar

r = 1
phase = pi/2.
z = rect(r, phase)

print("z = %f + I %f" %(z.real, z.imag))

x = 1
y = 1
z = complex(x, y)

print("z = %g e^{I %g}" % polar(z))

.....

z = 0.000000 + I 1.000000
z = 1.41421 e^{I 0.785398}

```



## Aufgabe 6.2

Geben Sie mithilfe eines C-Programms und der GSL-Bibliothek die Werte von zehn selbstgewählten **physikalischen Konstanten** aus. Schreiben Sie eine Funktion, die nur die drei kleinsten dieser Werte ausgibt. Verwenden Sie die GSL-Funktion *gsl\_sort\_smallest*, um das Gleiche zu erreichen.

```
#include <stdio.h>
#include <stdlib.h>
#include <gsl/gsl_const_num.h>
#include <gsl/gsl_const_mksa.h>
#include <gsl/gsl_sort.h>

const int N = 10;

/* int cmp(const void* a, const void* b) {
    double v1 = *((double *) a);
    double v2 = *((double *) b);

    return v1 == v2 ? 0 : (v1 < v2 ? -1 : 1);
}

void smallest(double con[N]) {
    qsort(con, N, sizeof(double), cmp);

    for (int i = 0; i < 3; i++)
        printf("%g\n", con[i]);
}*/

int main() {
    double con[N];

    con[0] = GSL_CONST_MKSA_SPEED_OF_LIGHT;
    con[1] = GSL_CONST_MKSA_PLANCKS.CONSTANT_HBAR;
    con[2] = GSL_CONST_NUM_AVOGADRO;
    con[3] = GSL_CONST_MKSA_BOLTZMANN;
    con[4] = GSL_CONST_MKSA_GRAVITATIONAL.CONSTANT;
    con[5] = GSL_CONST_MKSA_ELECTRON.CHARGE;
    con[6] = GSL_CONST_MKSA_MASS.ELECTRON;
    con[7] = GSL_CONST_MKSA_BOHR.RADIUS;
    con[8] = GSL_CONST_MKSA_VACUUM.PERMITTIVITY;
    con[9] = GSL_CONST_MKSA_ELECTRON.VOLT;

    int i;
    for (i = 0; i < N; i++)
        printf("%g\n", con[i]);

    printf("Kleinste:\n");
    /* smallest(con); */

    double res[3];
    int s = gsl_sort_smallest(res, 3, con, 1, N);
    for (i = 0; i < 3; i++)
        printf("%g\n", res[i]);
}

.....
2.99792e+08
1.05457e-34
6.02214e+23
1.38065e-23
6.673e-11
1.60218e-19
9.10938e-31
5.29177e-11
8.85419e-12
1.60218e-19
Kleinste:
1.05457e-34
9.10938e-31
1.38065e-23
```

### Aufgabe 6.3

Um zwei Dateien zu vergleichen, bzw. den Inhalt einer Datei zu überprüfen, werden sog. **Hashfunktionen** verwendet. Eine weitverbreitete Hashfunktion ist der **MD5**-Algorithmus, der eine 128-Bit-**Prüfsumme** aus dem Inhalt einer Datei berechnet. Wie viele verschiedene MD5-Werte gibt es maximal, d. h., wie hoch ist die Wahrscheinlichkeit, dass zwei Dateien zufällig denselben Hashwert haben?

Mit dem Kommando *md5sum* kann man den MD5-Wert einer Datei berechnen. Welche weiteren Kommandos zur Prüfsummenberechnungen gibt es auf Ihrem System, und wie schnell sind diese im Vergleich mit *md5sum*?

Anzahl an verschiedenen MD5-Werten:  $2^{128} \approx 3,4 \cdot 10^{38}$ . Wahrscheinlichkeit für Kollision:  $1/2^{128}$ , also sehr gering. Zwei Dateien zu erzeugen, die den gleichen Hashwert haben, ist aber mit wenig Rechenaufwand möglich. Daher ist MD5 für sichere Kommunikation nicht mehr zeitgemäß.

Test mit 2 GB Datei:

```
$ time md5sum test.dat
68162933e551e55779504dacdfb0c5ae test.dat
real    0m10.939s
$ time shasum test.dat
e7138032986bfcfa0e5fla8e41e2e9fd1ec94268 test.dat
real    0m15.558s
$ time sha256sum test.dat
cdc3cc3cdda04673f91232d24fbd9ac6521e9309a73b12064f9061d7751825d9 test.dat
real    0m21.247s
$ time sha512sum test.dat
1ac99cfbfcdb4b5cc8d6553dcba354a50ce26851d46639442795fc277d829e0d375a81027450fc62519f6ae17ff8a465ab6a4b5
real    0m14.544s
```

### Aufgabe 6.4

Verwenden Sie die Programme *gzip*, *bzip2* und *xz*, um je eine beliebige Text- und Binärdatei (verlustfrei) zu **komprimieren**. Vergleichen Sie Laufzeit und Komprimierung der Programme.

Überlegen Sie sich, ob es besser ist, ein *tar*-Archiv oder die Dateien im Archiv zu komprimieren.

Textdatei (200 MB):

```
$ time gzip datei.dat
5 s, 43 MB
$ time bzip2 datei.dat
17 s, 32 MB
$ time xz -T 4 datei.dat
60 s, 26 MB
```

Binärdatei (50 MB, exe-Datei):

```
$ time gzip datei.bin
2.6 s, 20 MB
$ time bzip2 datei.bin
4.1 s, 18 MB
$ time xz -T 4 datei.bin
20 s, 15 MB
```

Die Komprimierung von Binärdateien hängt sehr vom Typ der Datei ab. Bereits komprimierte Dateien z. B. lassen sich kaum noch weiter komprimieren.

Ob man erst alle Dateien in einem Verzeichnis oder das *tar*-Archiv komprimiert, kann einen Unterschied ergeben. Das *tar*-Archiv hängt alle Dateien hintereinander und eine Komprimierung des *tar*-Archivs kann damit deutlich besser komprimieren. Außerdem müssen damit nicht alle Dateien im Verzeichnis „angefasst“ werden.

## Aufgabe 6.5

Schreiben Sie ein C-Programm, um die **Bessel-Funktionen** erster Art  $J_0(x)$ ,  $J_1(x)$  und  $J_2(x)$  (aus *math.h*) für  $x = [0, 20]$  auszugeben und stellen Sie die Daten mit Gnuplot analog zu Abb. 6.2 grafisch dar.

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

int main() {
    int i;
    const int N = 1000;
    const double max = 20.;

    for (i = 0; i <= N; i++) {
        double x = max*i/N;
        printf("%g %g %g %g\n", x, j0(x), j1(x), jn(2, x));
    }
}

.....

$ gcc -o bessel bessel.c -lm
$ ./bessel > bessel.dat
$ gnuplot
gnuplot> p "bessel.dat" w l t "j_0", "" u 1:3 w l t "j_1", "" u 1:4 w l t "j_2"
```

## Aufgabe 6.6

### CAS-Übungen

Verwenden Sie z. B. *Mathematica* und lösen Sie folgende Aufgaben:

- (a) Vereinfachen Sie die Ausdrücke

$$a^2 + ab + b^2, \frac{x-1}{x^2-1}, \sin(x) \cos(y) - \cos(x) \sin(y). \quad (2)$$

- (b) Berechnen Sie die erste und zweite **Ableitung** von

$$x(t) = x_0 + v_0 t - \frac{g}{2} t^2, y(t) = x_0 \cos(\omega t) + \frac{v_0}{\omega} \sin(\omega t), f(\varepsilon) = \frac{1}{1 \pm e^{\beta(\varepsilon - \mu)}}. \quad (3)$$

- (c) Bestimmen Sie die **Taylor-Entwicklung** von  $f(x) = \sin(x)$  um  $x_0 = \pi/2$  bis zur zehnten Ordnung und stellen Sie das Ergebnis grafisch dar ( $x = [-2\pi, 3\pi]$ ).

- (d) Berechnen Sie die folgenden **Fresnel-Integrale** und stellen Sie die Ergebnisse grafisch dar für  $x = [0, 5]$ :

$$C(x) = \int_0^x \cos\left(\frac{\pi}{2} t^2\right) dt, S(x) = \int_0^x \sin\left(\frac{\pi}{2} t^2\right) dt. \quad (4)$$

(a)

```
FullSimplify[a^2 + a b + b^2]
```

```
a^2 + a b + b^2
```

```
Simplify[1/(x^2 - 1)]
```

```
1/(1 + x)
```

```
Sin[x] Cos[y] - Cos[x] Sin[y] // Simplify
```

```
Sin[x - y]
```

(b)

$$D\left[x_0 + v_0 t - \frac{g}{2} t^2, t\right]$$

$$D\left[x_0 + v_0 t - \frac{g}{2} t^2, \{t, 2\}\right]$$

$$-g t + v_0$$

$$-g$$

$$D\left[x_0 \cos[\omega t] + \frac{v_0}{\omega} \sin[\omega t], t\right]$$

$$D\left[x_0 \cos[\omega t] + \frac{v_0}{\omega} \sin[\omega t], \{t, 2\}\right]$$

$$v_0 \cos[t \omega] - x_0 \omega \sin[t \omega]$$

$$-x_0 \omega^2 \cos[t \omega] - v_0 \omega \sin[t \omega]$$

$$D\left[\frac{1}{1 + e^{\beta (eps - \mu)}}, eps\right]$$

$$D\left[\frac{1}{1 + e^{\beta (eps - \mu)}}, \{eps, 2\}\right] // FullSimplify$$

$$-\frac{e^{\beta (eps - \mu)} \beta}{(1 + e^{\beta (eps - \mu)})^2}$$

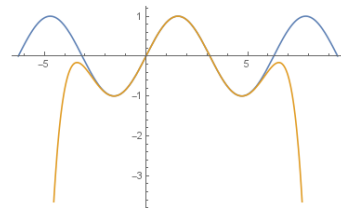
$$2 \beta^2 \operatorname{Csch}[\beta (eps - \mu)]^3 \operatorname{Sinh}\left[\frac{1}{2} \beta (eps - \mu)\right]^4$$

(c)

$$F[x_] = \text{Series}[\sin[x], \{x, \frac{\pi}{2}, 10\}] // \text{Normal}$$

$$1 - \frac{1}{2} \left(-\frac{\pi}{2} + x\right)^2 + \frac{1}{24} \left(-\frac{\pi}{2} + x\right)^4 - \frac{1}{720} \left(-\frac{\pi}{2} + x\right)^6 + \frac{\left(-\frac{\pi}{2} + x\right)^8}{40320} - \frac{\left(-\frac{\pi}{2} + x\right)^{10}}{3628800}$$

$$\text{Plot}[\{\sin[x], F[x]\}, \{x, -2\pi, 3\pi\}]$$



(d)

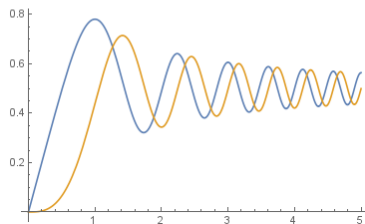
$$f1[x_] = \int_0^x \cos\left[\frac{\pi}{2} x^2\right] dx$$

$$\text{FresnelC}[x]$$

$$f2[x_] = \int_0^x \sin\left[\frac{\pi}{2} x^2\right] dx$$

$$\text{FresnelS}[x]$$

$$\text{Plot}[\{f1[x], f2[x]\}, \{x, 0, 5\}]$$



## Kapitel 7

### Aufgabe 7.1

Finden Sie heraus, wie viele Kerne und welche Vektoreinheiten die CPU Ihres Rechners hat. Das Kommando `cat /proc/cpuinfo` zeigt diese an. Was versteht man unter **Hyperthreading**?

\$ lscpu

```

...
Thread(s) per core: 2
Core(s) per socket: 4
Socket(s): 1
...
$ cat /proc/cpuinfo
... mmx sse sse2 ssse3 sse4_1 sse4_2 avx avx2

```

Hyperthreading bedeutet, dass pro echten Prozessorkern zwei Threads simuliert werden, um die Ressourcen eines Kernes besser zu nutzen. Mit Hyperthreading verdoppelt sich damit die Anzahl der (logischen) verfügbaren Kerne für das Betriebssystem und die Performance kann um bis zu 30 % steigen.

## Aufgabe 7.2

Probieren Sie das „Hallo Welt!“-Programm für **OpenMP** und **MPI** aus. Variieren Sie die Anzahl der verwendeten Rechenkern und lassen Sie die Programme mehrfach laufen. Was fällt Ihnen auf?

```

$ export OMP_NUM_THREADS=2
$ ./hallo-openmp
Hallo Welt!
Hallo Welt!
$ mpirun -n 4 ./hallo-mpi
Prozess 1 von 4 auf tux sagt 'Hallo Welt!'
Prozess 0 von 4 auf tux sagt 'Hallo Welt!'
Prozess 3 von 4 auf tux sagt 'Hallo Welt!'
Prozess 2 von 4 auf tux sagt 'Hallo Welt!'

```

Man erkennt: Die Ausgabe der einzelnen Threads ist nicht sortiert. Die Threads arbeiten also wirklich parallel und sind unabhängig voneinander.

## Aufgabe 7.3

Leiten Sie das **Amdahl'sche Gesetz** (7.1) her. Überlegen Sie sich dazu, wie die parallele Laufzeit mit  $N$  skaliert und berechnen Sie damit den **Speedup** als Verhältnis zwischen serieller und paralleler Laufzeit. Geben Sie den maximalen *Speedup* (Sättigungswert) abhängig von  $p$  an.

$$S = \frac{T_S}{T_P} = \frac{t_S + t_P}{t_S + \frac{t_P}{N}} = \frac{t_S + t_P}{t_S + t_P + \frac{t_P}{N} - t_P} = \frac{T_S}{T_S + t_P(\frac{1}{N} - 1)} = \frac{1}{1 + \frac{t_P}{t_S}(\frac{1}{N} - 1)} = \frac{1}{1 + p(\frac{1}{N} - 1)} \quad (5)$$

mit  $p = t_P/(t_S + t_P) = t_P/T_S$ . Damit:

$$S_{\max} = \lim_{N \rightarrow \infty} S = \frac{1}{1 - p} = \frac{T_S}{t_S}. \quad (6)$$

## Aufgabe 7.4

Was ist die maximal sinnvolle Anzahl  $N$  an Rechenkernen für eine Simulation mit einem **Parallelanteil** von  $p = 90 \%$ , wenn man mit 90 % des maximalen *Speedup* zufrieden ist?

Aus (5) folgt

$$N = \frac{p}{\frac{1}{S} + p - 1} \quad (7)$$

Mit  $p = 0,9$  und  $S = 0,9 * S_{\max} = \frac{0,9}{1-p} = 9$  ergibt sich  $N = 81$ .

## Kapitel 8

### Aufgabe 8.1

Neben dem Binärsystem wird in der Informatik oft auch das **Hexadezimalsystem** mit der Basis 16 (Ziffern: 0123456789ABCDEF) verwendet. Wie viele Binärziffern kann man damit zu einer Ziffer zusammenfassen? Geben Sie Beispiele an.

$16 = 2^4$ , d. h. man kann jeweils 4 Binärziffern zu einer Hexadezimalziffer zusammenfassen. Beispiele:  $1110_2 = 14 = E_{16}$ ,  $01001010_2 = 74 = 4A_{16}$ .

### Aufgabe 8.2

Moderne **Festplatten**, sog. SSD (*solid state drive*), verwenden Halbleiterspeicher anstelle von drehenden Magnetschichten. Aufgrund ihrer Arbeitsweise besitzen die einzelnen Speicherzellen einer SSD jedoch eine begrenzte Anzahl an Schreibzyklen. Für eine typische SSD mit 1,6 TB Kapazität und 5 Jahren Garantie wird die maximale Schreibmenge mit 8,76 PB angegeben. Wie oft pro Tag lässt sich die SSD also über ihre Lebenszeit beschreiben (sog. *drive writes per day*)?

Gesamtzahl an Schreibzyklen:  $8,76 \text{ PB} / 1,6 \text{ TB} \approx 5,5 \cdot 10^3$ .

Schreibzyklen pro Tag:  $5,5 \cdot 10^3 / (5 \cdot 365,25) \approx 3$ . Das entspricht also 4,8 TB pro Tag und ist ausreichend für die meisten Anwendungen.

### Aufgabe 8.3

In einer Simulation sollen dreidimensionale Vektoren auf einem dreidimensionalen Gitter mit 100 Punkten Kantenlänge verwendet werden. Wie viel **Speicherplatz** benötigt man damit, um alle Vektoren in doppelter oder einfacher Genauigkeit zu speichern?

Anzahl an Vektoren:  $100^3$ .

Speicherplatz pro Vektor:  $3 \times \text{sizeof}(\text{double}) | \text{sizeof}(\text{float})$  Byte.

Ergebnis:  $3 \cdot 100^3 \cdot 8 | 4 \text{ Byte} = 2,4 | 1,2 \text{ MByte}$ .

### Aufgabe 8.4

Schreiben Sie ein C-Programm, das bei gegebenem Intervall und Schrittweite die **Anzahl der Schritte** berechnet und die einzelnen Schritte ausgibt. Achten Sie auf Probleme beim Vergleich von Fließkommazahlen.

In Python gibt es dafür die Funktion *arange*. Überprüfen Sie damit die Ergebnisse.

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

int arange(double start, double end, double step) {
    int n = 0;
    while ((end - start) > DBL_MIN) {
        if (fabs(start) < DBL_EPSILON)
            start = 0.;
        printf("%g ", start);
        start += step;
        n++;
    }
}
```

```

    }
    puts("");

    return n;
}

int main() {
    int n = arange(-1, 1, .1);

    printf("%d steps\n", n);
}

.....

from numpy import arange

START=-1.
ENDE=1.
STEP=.1

a = arange(START, ENDE + STEP, STEP)

print(a.size, "Schritte")
print(a.round(5))

.....

$ ./arange
-1 -0.9 -0.8 -0.7 -0.6 -0.5 -0.4 -0.3 -0.2 -0.1 0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1
21 steps
$ python3 arange.py
21 Schritte
[-1. -0.9 -0.8 -0.7 -0.6 -0.5 -0.4 -0.3 -0.2 -0.1 -0.  0.1  0.2  0.3
 0.4  0.5  0.6  0.7  0.8  0.9  1. ]

```

## Kapitel 9

### Aufgabe 9.1

Bestimmen Sie für eine feste Schrittweite die **Ableitung** der Funktion  $f(x) = \sin(x^2)$  mit der Vorwärts- und der zentralen Differenz numerisch. Wann weicht die Näherung deutlich vom exakten Ergebnis ab?

```

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

double f(double x) {
    return sin(x*x);
}

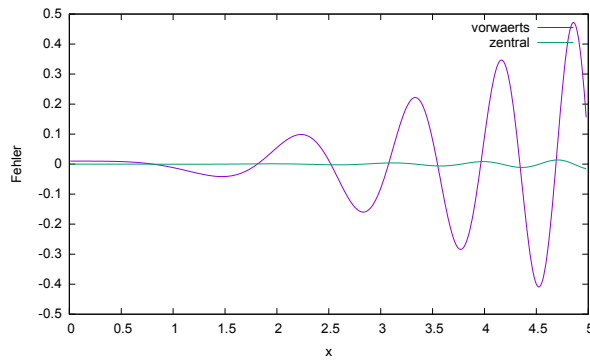
double fs(double x) {
    return 2.*x*cos(x*x);
}

int main() {
    const int N = 500;
    const double h = 0.01;

    double fv[N], fz[N];
    for (int i = 1; i < N-1; i++) {
        double x = i*h;
        fv[i] = (f(x+h)-f(x))/h;
        fz[i] = (f(x+h)-f(x-h))/(2.*h);
    }

    for (int i = 1; i < N - 1; i++)
        /* printf("%g %g %g %g\n", i*h, fs(i*h), fv[i], fz[i]); */
        printf("%g %g %g\n", i*h, fv[i]-fs(i*h), fz[i]-fs(i*h));
}

```



## Aufgabe 9.2

Schreiben Sie in C oder Python ein Programm, um die Nullstelle der Funktion  $f(x) = \cos x - x$  mit dem **Sekantenverfahren** zu berechnen.

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

double f(double x) {
    return cos(x) - x;
}

int main() {
    const int NMAX=100;
    const double EPS=1.e-6;
    double x1 = 0., x2 = 1.;

    int n = 0;
    do {
        double f1 = f(x1), f2 = f(x2);
        double x3 = (f2*x1 - f1*x2)/(f2-f1);
        x1 = x2;
        x2 = x3;
        n++;
    } while (fabs(x2-x1) > EPS && n < NMAX);

    printf(" Nullstelle: %g\n", (x1 + x2)/2.);
}
```

## Aufgabe 9.3

Mithilfe des **Newton-Verfahren** lässt sich die Quadratwurzel einer Zahl  $a$  berechnen, indem man die Funktion  $f(x) = x^2 - a$  betrachtet (babylonisches Wurzelziehen). Wie lautet für  $f(x)$  die (vereinfachte) Newton-Iteration? Programmieren Sie dafür das Newton-Verfahren, um die Wurzel aus 3 zu berechnen. Nach wie vielen Schritten ist die Maschinengenauigkeit erreicht?

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} = x_n - \frac{x_n^2 - a}{2x_n} = \frac{1}{2} \left( x_n + \frac{a}{x_n} \right).$$

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

int main() {
    double x = 1.;
    const int NMAX = 100;
    const double a = 3.;

    int n = 0;
    while ( fabs(x*x - a) > 1.e-15 && n < NMAX ) {
        x = 0.5 * (x + a/x);

        n++;
        printf("%d: %.15g\n", n, x);
    }
}
```



```

    }
}

.....

1: 2
2: 1.75
3: 1.73214285714286
4: 1.73205081001473
5: 1.73205080756888

```

## Aufgabe 9.4

Schreiben Sie ein Python-Programm, dass das **Interpolationspolynom** für drei beliebige Datenpunkte mit der Neville-Aitken-Interpolation findet. Testen Sie Ihr Programm anhand von drei gewählten Punkten und der SciPy-Funktion `scipy.interpolate.lagrange`. Stellen Sie das Ergebnis mit den Datenpunkten grafisch dar.

```

from numpy import linspace
from scipy.interpolate import lagrange

xp=[1,2,3]
yp=[1,3,2]

def P(x, i, j):
    if (i == j):
        return yp[i]
    return ((x - xp[j])*P(x, i, j-1) - (x - xp[i])*P(x, i+1, j))/(xp[j] - xp[i])

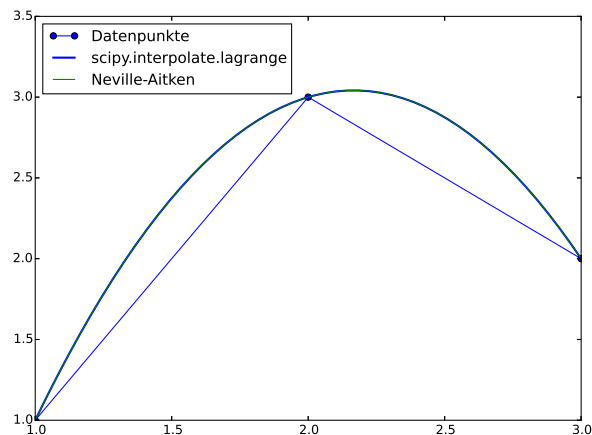
x = linspace(1, 3)
pl.plot(xp, yp, 'bo-', label='Datenpunkte')

print(lagrange(xp, yp))
pl.plot(x, lagrange(xp, yp)(x), label='scipy.interpolate.lagrange', linewidth=2)

pl.plot(x, P(x,0, 2), 'g-', label='Neville-Aitken')

pl.legend(loc='best')
pl.show()

```



## Aufgabe 9.5

Analog zur Fourier-Reihe lässt sich eine Funktion auf dem Intervall  $[-1, 1]$  auch in die orthonormalen **Legendre-Polynome**  $P_l(x)$  (Normierung:  $\int_{-1}^1 P_i(x)P_j(x) dx = \frac{2}{2j+1}\delta_{ij}$ ) entwickeln (s. Abschn. 18.2.2):

$$f(x) = \sum_{l=0}^{\infty} a_l P_l(x), \quad a_l = \frac{2l+1}{2} \int_{-1}^1 f(x) P_l(x) dx. \quad (8)$$

Berechnen Sie die Entwicklungskoeffizienten  $a_l$  durch **numerische Integration** für die Funktion  $f(x) = \sin(\pi x)$  und stellen Sie die Näherungen grafisch dar.

```
from numpy import sin, pi, linspace
from scipy.integrate import quad

def P0(x):
    return 1
def P1(x):
    return x
def P2(x):
    return (3*x**2+1)/2
def f(x):
    return sin(pi*x)

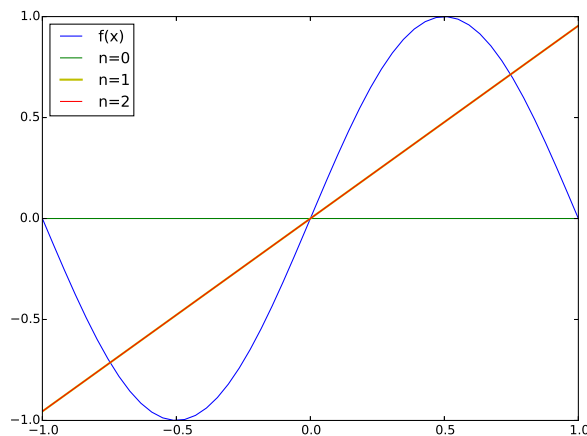
f0 = lambda x: f(x)*P0(x)
f1 = lambda x: f(x)*P1(x)
f2 = lambda x: f(x)*P2(x)

a0 = quad(f0, -1, 1)[0]
a1 = 3/2*quad(f1, -1, 1)[0]
a2 = 5/2*quad(f2, -1, 1)[0]

print(a0, a1, a2)

x = linspace(-1, 1)
pl.plot(x, f(x), 'b-', label='f(x)')
pl.plot(x, a0*P0(x)+0*x, 'g-', label='n=0')
pl.plot(x, a0*P0(x)+a1*P1(x), 'y-', label='n=1', linewidth=2)
pl.plot(x, a0*P0(x)+a1*P1(x)+a2*P2(x), 'r-', label='n=2')

pl.legend(loc='best')
pl.show()
```



## Aufgabe 9.6

Berechnen Sie die Gewichte der **Trapez-** und **Simpson-Regel** in Tab. 9.2 und damit die angegebenen Regeln (9.48) und (9.53). Überprüfen Sie die Gewichte der **2-Punkt-Gauß-Quadratur** mit Formel (9.70).

Trapez,  $N = 1$ :

$$L_0(x) = \frac{x-b}{a-b} \rightarrow w_0 = \frac{1}{b-a} \int_a^b \frac{x-b}{a-b} dx = \frac{1}{2}, \quad (9)$$

$$L_1(x) = \frac{x-a}{b-a} \rightarrow w_1 = \frac{1}{b-a} \int_a^b \frac{x-a}{b-a} dx = \frac{1}{2}. \quad (10)$$

Simpson,  $N = 2$ :

$$L_0(x) = \frac{x - \frac{a+b}{2}}{a - \frac{a+b}{2}} \cdot \frac{x - b}{a - b} \rightarrow w_0 = \frac{1}{b - a} \int_a^b L_0(x) dx = \frac{1}{6}, \quad (11)$$

$$L_1(x) = \frac{x - a}{\frac{a+b}{2} - a} \cdot \frac{x - b}{\frac{a+b}{2} - b} \rightarrow w_1 = \frac{1}{b - a} \int_a^b L_1(x) dx = \frac{2}{3}, \quad (12)$$

$$L_2(x) = \frac{x - a}{b - a} \cdot \frac{x - \frac{a+b}{2}}{b - \frac{a+b}{2}} \rightarrow w_2 = \frac{1}{b - a} \int_a^b L_2(x) dx = \frac{1}{6}. \quad (13)$$

Gauss 2-Punkt:

$$L_0(x) = \frac{x - (-1)}{1 - (-1)} = \frac{x + 1}{2} \rightarrow w_0 = \int_{-1}^1 L_0(x) dx = \frac{1}{2}[x]_{-1}^1 = 1, \quad (14)$$

$$L_1(x) = \frac{x - 1}{-1 - 1} = \frac{1 - x}{2} \rightarrow w_1 = \int_{-1}^1 L_1(x) dx = \frac{1}{2}[x]_{-1}^1 = 1. \quad (15)$$

## Aufgabe 9.7

Das radiale **Beugungsmuster** einer runden Lochblende ist gegeben durch die Intensität  $I(r) = (J_1(kr)/kr)^2$  mit  $k = 2\pi/\lambda$  und den Bessel-Funktionen

$$J_m(x) = \frac{1}{\pi} \int_0^\pi \cos(m\vartheta - x \sin \vartheta) d\vartheta. \quad (16)$$

Schreiben Sie ein Python-Programm zur Berechnung von  $J_0(x)$ ,  $J_1(x)$  und  $J_2(x)$  mit der **Trapezmethode** ( $N = 100$ ). Vergleichen Sie die Ergebnisse mit den entsprechenden Python-Funktionen (s. Abb. 6.2).

Stellen Sie damit das Beugungsmuster einer Lochblende als Dichteplot für verschiedene Wellenlängen dar.

```
import pylab as pl

from numpy import pi, sin, cos, linspace
from scipy.special import j0, j1, jv

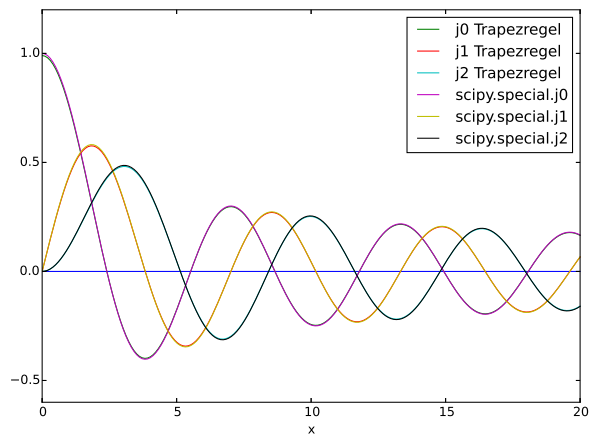
def f(n, t, x):
    return cos(n*t - x*sin(t))

N = 100
def j(n, x):
    t = linspace(0, pi, N)
    h = pi/N

    sum = 0
    for i in t:
        sum += f(n, i, x)
    return h/pi*(sum - (f(n,0,x)+f(n,pi,x))/2)

x=linspace(0, 20, 200)
pl.plot(x,0*x)
pl.plot(x,j(0, x), label="j0 Trapezregel")
pl.plot(x,j(1, x), label="j1 Trapezregel")
pl.plot(x,j(2, x), label="j2 Trapezregel")
pl.plot(x,j0(x), label="scipy.special.j0")
pl.plot(x,j1(x), label="scipy.special.j1")
pl.plot(x,jv(2,x), label="scipy.special.j2")
pl.xlabel('x')

pl.legend(loc='best')
pl.show()
```



```

from numpy import linspace, meshgrid, sqrt, pi, log
from scipy.special import jn
from matplotlib.colors import LogNorm
import matplotlib.pyplot as plt

l = 20          # Wellenlaenge
n = 100         # Anzahl Datenpunkte pro Dimension
x = linspace(-20, 20, n)
y = linspace(-20, 20, n)

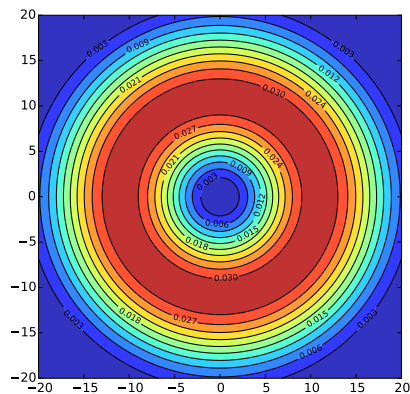
X,Y = meshgrid(x, y)
k = 2 * pi / l  # Wellenvektor
r = sqrt((k * X)**2 + (k * Y)**2)

# Intensitaet: Besselfunktion
I = ((jn(1, r) / r)**2)

plt.contourf(X, Y, I, 10, alpha=.8, norm=LogNorm()) # Dichte plot
C = plt.contour(X, Y, I, 10, colors='black', norm=LogNorm()) # Contour plot
plt.clabel(C, inline=1, fontsize=10) # Contour label

plt.gca().set_aspect('equal', adjustable='box') # Quadratisch
plt.show()

```



mit  $u = x\sqrt{2/(\lambda z)}$  und den **Fresnel-Integralen** (6.3).

Schreiben Sie ein Programm zur numerischen Berechnung der Fresnel-Integrale und damit der Intensität (17) und stellen Sie das Ergebnis für verschiedene Wellenlängen und Abstände zum Schirm grafisch dar.

```
import pylab as pl

from numpy import sin, cos, pi, linspace, vectorize
from scipy.integrate import quad

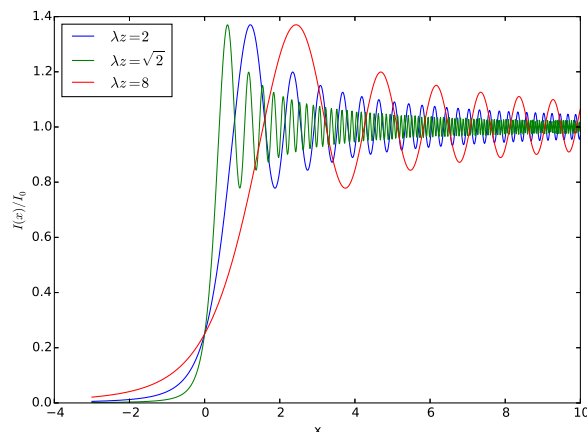
fC = lambda t: cos(pi/2*t**2)
fS = lambda t: sin(pi/2*t**2)

def C(x):
    return quad(fC, 0, x)[0]
def S(x):
    return quad(fS, 0, x)[0]

def I(a, x):
    return 1/8*((2*C(a*x)+1)**2 + (2*S(a*x)+1)**2)
vI = vectorize(I)

x = linspace(-3, 10, 1000)
pl.plot(x, vI(1, x), 'b', label='$\lambda z = 2$')
pl.plot(x, vI(2, x), 'g', label='$\lambda z = \sqrt{2}$')
pl.plot(x, vI(.5, x), 'r', label='$\lambda z = 8$')
pl.xlabel('x')
pl.ylabel('$I(x)/I_0$')

pl.legend(loc='best')
pl.show()
```



## Aufgabe 9.9

Berechnen Sie die Koeffizienten  $a_i$  der reellen **Fourier-Reihe**

$$f_n(x; \mathbf{a}) = \sum_{i=0}^n a_i \cos(k_i x) \quad (18)$$

mithilfe der **Kleinste-Quadrate-Methode**, d. h.

$$\chi^2(\mathbf{a}) = \int (f(x) - f_n(x; \mathbf{a}))^2 dx. \quad (19)$$

Es ergeben sich die bekannten Fourierkoeffizienten (s. Abschn. 9.4), die also die quadratischen Abweichungen der Näherung  $f_n$  von der Funktion  $f$  minimieren.

$$0 = \frac{d\chi^2}{da_i} = \int 2(f(x) - \underbrace{f_n(x; \mathbf{a})}_{=\sum_{j=0}^n a_j \cos(k_j x)}) \underbrace{\frac{df_n(x; \mathbf{a})}{da_i}}_{\cos(k_i x)} dx \quad (20)$$

$$\rightarrow \int f(x) \cos(k_i x) dx = \sum_{j=0}^n a_j \underbrace{\int \cos(k_j x) \cos(k_i x) dx}_{\delta_{ij}} \quad (21)$$

$$= a_i. \quad (22)$$

## Aufgabe 9.10

Bestimmen Sie mit einem Python-Programm die Parameter  $a$  und  $b$  einer **Lineare Regression** für die drei Datenpunkte  $(-1; 0)$ ,  $(0; 2)$  und  $(1; 1)$ . Bestimmen Sie nun mit `scipy.optimize.curve_fit()` die **Nichtlineare Anpassung** an die Datenpunkte.

Berechnen Sie dann das interpolierende Polynom 2. Ordnung mittels **Polynomregression** und nähern Sie das Polynom mit einer linearen Funktion durch **Polynomapproximation** (s. Abschn. 18.2.2) mithilfe der **Legendre-Polynome**. Vergleichen Sie die Approximation mit der Regression bzw. Anpassung.

```
from numpy import linspace, array, ndarray
from scipy import optimize
from scipy.integrate import quad
from scipy.linalg import solve
import pylab as pl

x_data = array([-1, 0, 1])
y_data = array([0, 2, 1])
x = linspace(-2, 3)
pl.scatter(x_data, y_data, label="Data")

def cov(x, y):
    return sum([x[i]*y[i] for i in range(0, len(x))]) - sum(x)*sum(y)

def a(x_data, y_data):
    a = cov(x_data, y_data)/cov(x_data, x_data)
    b = sum(y_data)/len(y_data) - a*sum(x_data)/len(x_data)
    print("Ausgleichsgerade mit Least Squares:")
    print("Steigung: ", a)
    print("y-Achstenabschnitt: ", b)
    print()
    pl.plot(x, a*x + b, label="Least Squares", linestyle="-")

def b(x_data, y_data):
    x_len = len(x_data)
    x_vandermonde = ndarray(shape=(x_len, x_len))
    for i in range(0, x_len):
        for j in range(0, x_len):
            x_vandermonde[i][j] = x_data[i]**j
    a = solve(x_vandermonde, y_data)
    print("Interpolierendes Polynom P(x) = a_2*x**2 + a_1*x + a_0 mit")
    for i in range(0, len(x_data)):
        print("a_"+str(i), " = ", a[i])
    print()
    P = lambda x : sum(a[i]*x**i for i in range(0, x_len))
    pl.plot(x, P(x), label="Interp. Polynom")
    m = 3/2*quad((lambda x: P(x)*x), -1, 1)[0]
    b = 1/2*quad(P, -1, 1)[0]
    print("Mittels Polynomapproximation linearisiertes Polynom $ax + b$ mit:")
    print("a = ", m)
    print("b = ", b)
    print()
    pl.plot(x, m*x + b, label="Polynomapproximation")

def c(x_data, y_data):
```

```

f = lambda x, *p : p[1]*x + p[0]
popt, pcov = optimize.curve_fit(f, x_data, y_data, [0.5, 3])
print("Ausgleichsgerade mit Scipy curve_fit:")
print("Steigung: ", popt[1])
print("y-Achstenabschnitt: ", popt[0])
print()
pl.plot(x, popt[1]*x + popt[0], label="curve_fit", linestyle=":")

a(x_data, y_data)
b(x_data, y_data)
c(x_data, y_data)
pl.grid()
pl.legend(loc='best')
pl.xlim(-2,3)
pl.xlabel('x')
pl.show()

```

.....

Ausgleichsgerade mit Least Squares:

Steigung: 0.5

y-Achstenabschnitt: 1.0

Interpolierendes Polynom  $P(x) = a_2*x**2 + a_1*x + a_0$  mit

$a_0 = 2.0$

$a_1 = 0.5$

$a_2 = -1.5$

Mittels Polynomapproximation linearisiertes Polynom  $ax + b$  mit:

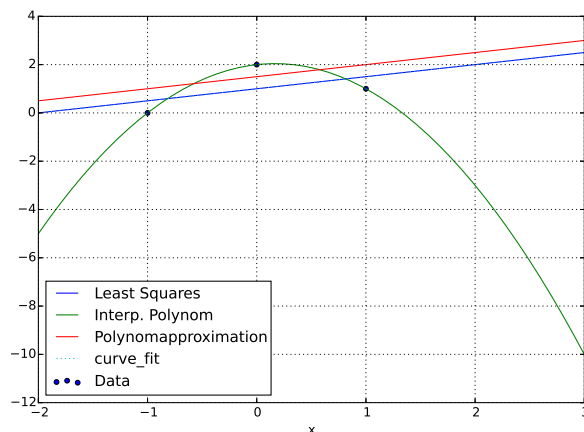
$a = 0.5$

$b = 1.5$

Ausgleichsgerade mit Scipy curve\_fit:

Steigung: 0.49999999999454703

y-Achstenabschnitt: 1.0000000000010902



## Kapitel 10

### Aufgabe 10.1

Leiten Sie die Iterationsvorschrift (10.20) für GDGL der Form

$$f''(x) = F(f(x), x) \quad (23)$$

her. Ersetzen Sie dafür die zweite Ableitung durch die zentrale Differenz (9.6). Woran erkennt man, dass es sich um ein **Mehrschrittverfahren** handelt?

Diskretisierung ( $h = \Delta x$ ):

$$x \rightarrow x_n = x_0 + nh \quad (24)$$

$$f(x) \rightarrow f(x_n) = f_n \quad (25)$$

$$F(f(x), x) \rightarrow F(f_n, x_n) = F_n \quad (26)$$

$$f''(x) \rightarrow \frac{f_{n+1} + 2f_n + f_{n-1}}{h^2} \quad (27)$$

Einsetzen in (23):

$$\rightarrow f_{n+1} = 2f_n + h^2 F(f_n, x_n) - f_{n-1} \quad (28)$$

## Aufgabe 10.2

Die Idee der **Adams-Bashforth-Methode** ist es, anstatt der GDGL

$$f'(x) = F(f(x), x) \quad (29)$$

die vereinfachte GDGL

$$f'(x) = p(x) \quad (30)$$

durch Integration zu lösen.  $p(x)$  ist dabei das **Interpolierende Polynom** für  $F(f(x), x)$ . Finden Sie das lineare interpolierende Polynom (9.13) durch die Punkte  $F(f_{n-1}, x_{n-1})$  und  $F(f_n, x_n)$  und damit durch Integration von (30) die 2-Punkt-Adams-Bashforth-Methode (10.18).

Verwende das interpolierende Polynom mit der alternativen Formel

$$p(x) = F_n + \frac{F_n - F_{n-1}}{h}(x - x_n) \quad (31)$$

und integriere

$$f_{n+1} = f_n + \int_{x_n}^{x_{n+1}} p(x) dx \quad (32)$$

$$= f_n + F_n h + \frac{F_n - F_{n-1}}{h} \underbrace{\int_{x_n}^{x_{n+1}} (x - x_n) dx}_{=h/2} \quad (33)$$

$$= f_n + \frac{h}{2}(3F_n - F_{n-1}). \quad (34)$$

## Kapitel 11

### Aufgabe 11.1

Bestimmen Sie mithilfe der **Gauß-Elimination** die Lösung des LGS

$$\begin{aligned} 2x_1 + x_2 &= 1, \\ x_1 + 2x_2 + x_3 &= 1, \\ x_2 + 2x_3 &= 1. \end{aligned} \quad (35)$$

Schreiben Sie ein C-Programm, um Ihr Ergebnis mit dem **Thomas-Algorithmus** (s. Listing 11.1) zu überprüfen.

$x_1$	$x_2$	$x_3$		
2	1	0	1	I
1	2	1	1	II
0	1	2	1	III
2	3	0	1	2*II-III=II*
4	0	0	2	3*I-II*=I*

Ergebnis:  $x_1 = 0, 5$ ,  $x_2 = (1 - 2x_1)/3 = 0$ ,  $x_3 = (1 - x_2)/2 = 0, 5$ .



```

#include <stdio.h>

const int N = 3;

// simplified Thomas algorithm for diffusion problem (a=c=1)
void solveThomas(double *b, double *d, double *x) {
    int i;
    for (i = 1; i < N; i++) {
        double m = 1.0/b[i-1];
        b[i] = b[i] - m;
        d[i] = d[i] - m*d[i-1];
    }

    x[N-1] = d[N-1]/b[N-1];

    for (i = N - 2; i >= 0; i--)
        x[i] = (d[i] - x[i+1])/b[i];
}

int main() {
    double x[N], b[N], d[N];

    for (int i = 0; i < N; i++) {
        b[i] = 2.;
        d[i] = 1.;
    }

    solveThomas(b, d, x);
    printf("x1 = %g, x2 = %g, x3 = %g\n", x[0], x[1], x[2]);
}

.....
x1 = 0.5, x2 = 0, x3 = 0.5

```

## Aufgabe 11.2

Schreiben Sie ein Python-Programm, das die **Eigenwerte und -vektoren** der Pauli-Matrizen  $\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$ ,  $\sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$  und  $\sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$  bestimmt.

```

from numpy import array
from numpy.linalg import eig

def printEigensystem(name):
    print(name+":")
    print("Eigenwerte:")
    print(w)
    print("Eigenvektoren:")
    print(v)

sx = array([[0, 1], [1, 0]])
w, v = eig(sx)
printEigensystem("sx")

sy = array([[0, -1j], [1j, 0]])
w, v = eig(sy)
printEigensystem("sy")

sz = array([[1, 0], [0, -1]])
w, v = eig(sz)
printEigensystem("sz")

```

```

.....
sx :
Eigenwerte:
[ 1. -1.]
Eigenvektoren:
[[ 0.70710678 -0.70710678]
 [ 0.70710678  0.70710678]]
sy :
Eigenwerte:

```

```
[ 1.+0.j -1.+0.j]
Eigenvektoren:
[[-0. -0.70710678j  0.70710678+0.j]
 [ 0.70710678+0.j  0. -0.70710678j]]
sz :
Eigenwerte:
[ 1. -1.]
Eigenvektoren:
[[1. 0.]
 [0. 1.]]
```

## Aufgabe 11.3

### Eigenwerte mit dem charakteristischen Polynom berechnen

- (a) Schreiben Sie eine C-Funktion, die die **Determinante** einer  $2 \times 2$ -Matrix berechnet. Verwenden Sie diese, um eine weitere C-Funktion zu schreiben, die die Determinante einer  $3 \times 3$ -Matrix berechnet mittels Entwicklung z. B. nach der ersten Zeile.
- (b) Gesucht sind die **Eigenwerte** der Matrix  $\mathbf{A} = \begin{pmatrix} 2 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 2 \end{pmatrix}$ . Definieren Sie das charakteristische Polynom  $f(\lambda) = \det(\mathbf{A} - \lambda \mathbb{1})$  und plotten Sie die Funktion  $f(\lambda)$  für den Bereich  $[0, 4]$ .
- (c) Bestimmen Sie die drei Eigenwerte von  $\mathbf{A}$ , indem Sie die Nullstellen von  $f(\lambda)$  bestimmen. Wählen Sie dazu drei geeignete Intervalle und verwenden Sie jeweils eine **Intervallhalbierung**. Vergleichen Sie die Ergebnisse mit der Ausgabe von Listing 11.4.

```
#include <stdio.h>

double det2(double a[2][2]) {
    return a[0][0]*a[1][1] - a[1][0]*a[0][1];
}

double det3(double a[3][3]) {
    double det=0.0;

    double a1[2][2]={ { a[1][1], a[1][2] }, { a[2][1], a[2][2] } };
    det += a[0][0]* det2(a1);
    double a2[2][2]={ { a[1][0], a[1][2] }, { a[2][0], a[2][2] } };
    det -= a[0][1]* det2(a2);
    double a3[2][2]={ { a[1][0], a[1][1] }, { a[2][0], a[2][1] } };
    det += a[0][2]* det2(a3);

    return det;
}

double ns(double a[3][3], double l1, double l2) {
    double lh;
    while (l2-l1 > 1.e-6) {
        lh=(l1+l2)/2.0;
        double det1, deth;
        a[0][0]=a[1][1]=a[2][2]=2-l1;
        det1=det3(a);
        a[0][0]=a[1][1]=a[2][2]=2-lh;
        deth=det3(a);
        if (det1*deth <= 0.0)
            l2=lh;
        else
            l1=lh;
    }
    printf("%g\n", lh);
}
```

```

}

int main() {
    double a[3][3]={ {2,1,0},{1,2,1},{0,1,2}};
    //printf("%g\n",det3(a));

    double l;
    FILE *f=fopen("eigen.dat","w");
    for (l = 0.0;l <= 4.0;l += 0.1) {
        a[0][0] = a[1][1] = a[2][2] = 2-l;
        fprintf(f,"%g %g\n",l,det3(a));
    }

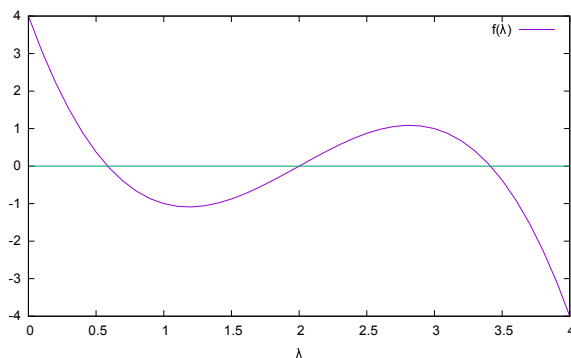
    // Intervallhalbierung
    ns(a,0.0,1.0);
    ns(a,1.5,2.5);
    ns(a,3.0,4.0);
}

```

```

$ ./eigen
0.585786
2
3.41421

```



## Kapitel 12

### Aufgabe 12.1

Betrachten Sie den **linearen Kongruenzgenerator** (12.1) mit  $m = 2^{31}$ ,  $a = 65539$  und  $b = 0$ . Schreiben Sie ein Python-Programm, um damit  $10^4$  Zufallszahlen zu erzeugen und geben Sie die Zahlen in drei Spalten aus. Überprüfen Sie mit einer dreidimensionalen Darstellung der Daten mit Matplotlib, ob Sie das **Hyperebenenverhalten** in Abb. 12.1 reproduzieren können.

```

N=10002
# RANDU
A=65539
B=0
SEED=5
M=2**31

```

```

r=SEED
for i in xrange(N):
    r=(A*r+B) % M
    print r/(M-1.),
    if i%3==2:
        print

```

```

.....

from mpl_toolkits.mplot3d import Axes3D
import pylab as pl
from numpy import genfromtxt

```

```

data = genfromtxt('random.dat', delimiter=' ')

fig = plt.figure()
ax = fig.gca(projection='3d')
ax.scatter(data[:,0], data[:,1], data[:,2], s=4, edgecolor='k', lw=.2)
ax.view_init(elev=10., azim=236)

plt.show()

```

## Aufgabe 12.2

Listing 12.3 zeigt, wie man in Python beliebig viele **Nachkommastellen** von  $\pi$  ausgeben kann. Verändern Sie das Programm, um die **Häufigkeitsverteilung** der Ziffern zu berechnen. Stellen Sie die Häufigkeitsverteilung grafisch dar und überprüfen Sie die Gleichverteilung.

Welche Nachteile könnten die Nachkommastellen von  $\pi$  als Zufallszahlengenerator haben?

```

from numpy import zeros, arange
from mpmath import mp

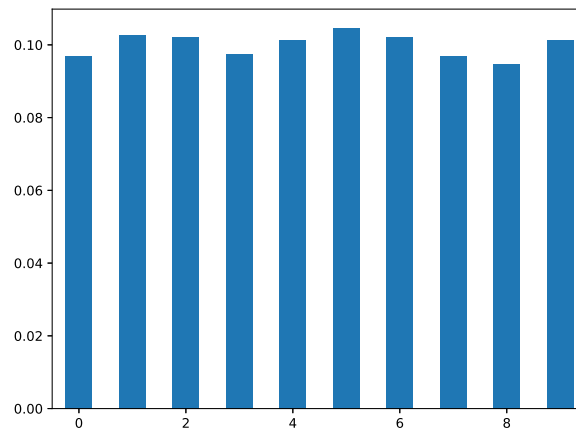
N = 10000 # number of digits
mp.dps = N
s = str(mp.pi)

v = zeros(N)
for i in range(2, N):
    v[i] = int(s[i])

d = .5
plt.hist(v, arange(0, N+1, 1), normed=True, align='left', rwidth=d, label="$N = $ %d" % N)
plt.xlim(-d, 9+d)

plt.show()

```



## Aufgabe 12.3

Die Summe von  $N$  gleichverteilten Zufallszahlen ergibt im Grenzfalle  $N \rightarrow \infty$  eine **Normalverteilung**. Damit kann man also näherungsweise normalverteilte Zufallszahlen erzeugen.

Schreiben Sie ein Python-Programm und bestimmen Sie  $n$  Zufallszahlen  $z$ , indem Sie die Summe von jeweils zehn gleichverteilten Zufallszahlen  $r$  berechnen und den Mittelwert

abziehen:

$$z_i = \sum_{j=1}^{10} r_{i,j} - 5. \quad (36)$$

Stellen Sie die Verteilung der Zufallszahlen  $z_i$  als Histogramm dar. Erzeugen Sie außerdem normalverteilte Zufallszahlen mit  $\mu = 0$  und  $\sigma = 1$  und vergleichen Sie die Verteilungen für verschiedene Mengen  $n$  an Zufallszahlen.

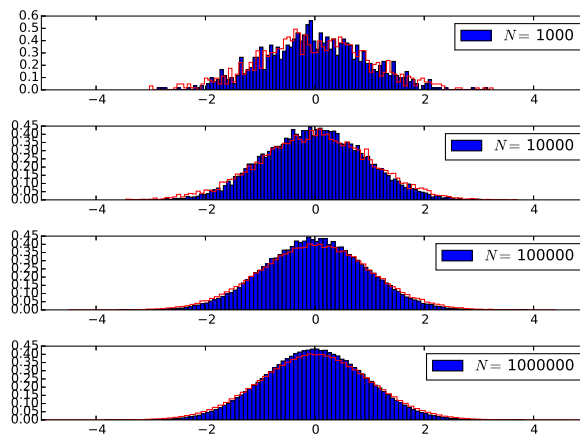
```
import pylab as pl
from numpy import arange, zeros
from random import random, gauss
```

```
BINS=100
```

```
def generate(N):
    a=zeros(N)
    b=zeros(N)
    for i in arange(N):
        sum=0
        for n in arange(10):
            sum += random()
        a[i]=sum-5
    for i in arange(N):
        b[i]=gauss(0.0,1.0)
    pl.hist(a,BINS,normed=True,label="$N = $ %d" % N)
    pl.hist(b,BINS,normed=True,histtype='step', color='red')
    pl.xlim(-5,5)
    pl.legend(loc='best')
```

```
pl.subplot(4,1,1)
generate(1000)
pl.subplot(4,1,2)
generate(10000)
pl.subplot(4,1,3)
generate(100000)
pl.subplot(4,1,4)
generate(1000000)
```

```
pl.show()
```



Computerphysik

Einführung, Beispiele und Anwendungen

Gerlach, S.

2019, XVIII, 290 S. 100 Abb., 71 Abb. in Farbe.,

Softcover

ISBN: 978-3-662-59245-8