Institut für Theoretische Physik
der Universität zu Köln

Prof. Dr. Simon Trebst
Michael Becker, Peter Bröcker

Computerphysik

Übungsblatt 3

SS 2013

Website: http://www.thp.uni-koeln.de/trebst/Lectures/2013-CompPhys.html
Abgabedatum: Montag, 06. Mai 2013 vor Beginn der Vorlesung

Im Gegensatz zum Ableiten ist das analytische **Integrieren** oft trickreich und in vielen Fällen gar nicht möglich. An diesem Punkt helfen uns numerische Methoden, die auf verschiedene Weisen den Flächeninhalt unter einer gegebenen Funktion approximieren. Wir wollen auf diesem Übungsblatt vertiefen, wie man dies am effizientesten anstellt und welche Bedingungen an die zu integrierende Funktion gestellt werden.

13. Das *sys*-Modul in Python

Programmiertechniken

Mittlerweile haben Sie erste Erfahrungen mit der Programmierung von Algorithmen gemacht und sicherlich bemerkt, dass es häufig einen Parameter gibt, den man variieren möchte, um zu schauen wie er das Ergebnis beeinflusst. In den bislang besprochenen iterativen Verfahren könnte dies etwa die Anzahl der maximalen Iterationsschritte oder die geforderte Genauigkeit sein. Um nicht jedes Mal das Programm selbst zu verändern, gibt es die Möglichkeit dem Programm **Startparameter** zu übergeben. Das dazu benötigte Pythonmodule heißt sys. Das folgende Beispielprogramm illustriert die grundlegende Vorgehensweise.

```
1  import sys
2
3  print len(sys.argv)
4
5  for i,arg in enumerate(sys.argv):
6      print "#", i, arg
```

Hat Ihr Programm mehrere Optionen, von denen einige eventuell auch nur optional sind, wird der Programmstart schnell unübersichtlich. Deswegen ist es üblich **flags** zu benutzen um den Programm mitzuteilen, welchen Parameter man als nächstes angeben möchte. Ihr Programmaufruf würde dann etwa folgendermaßen aussehen:

```
python sys_module.py --iteration-steps 100 --discretization 0.1 -w 2
```

Beachten Sie, dass ausgeschriebenen Wörtern meist zwei Striche vorausgehen, während bei kurzen (nur aus einem Buchstaben bestehenden) Argumenten nur ein Strich verwendet wird.

Ihre Aufgabe sei es nun, ein Programm zu schreiben, das zwei Argumente, jeweils in Kurz- und Langform, akzeptiert – nennen wir sie *arg1* und *arg2*. Gibt der Benutzer ein "-h" als Argument, so sollte ihm angezeigt werden, welche Argumente er angeben kann. Beachten Sie, dass Sie sich vorher überlegen müssen, ob ein Parameter optional ist oder nicht und was passiert, wenn er nicht angegeben wird. Am Ende soll die aktuelle Parameterkonfiguration ausgegeben werden.

14. Cache-Effekte

5 Punkte

In der Vorlesung haben wir anhand einfacher physikalischer Überlegungen untersucht, wie schnell sich Informationen im Computer (maximal) ausbreiten können und was dies für die geometrische Anordnung etwa von CPU und Speicher bedeutet.

Illustrieren Sie diese Überlegungen für einen CPU Chip von 3cm Kantenlänge: Wie hoch darf die Taktfrequenz einer solchen CPU höchstens sein, damit die beiden am weitesten voneinander entferntesten Punkte des Chips wähnend eines Taktes noch Informationen austauschen können, also Information von einem Punkt zum anderen und wieder zurück übertragen können? Bedenken Sie dabei, dass die Leiterbahnen auf einem CHIP nicht diagonal sondern in Manhattan-Form (rechtwinklig) verlegt sind.

Um diesen elementaren Flaschenhals zu vermeiden, werden in modernen Chips verschiedene **Speicher-Ebenen** (sogenannte **Cache-Level**) verbaut, die in unmittelbarer Nähe zu den Rechen-Cores auf dem Chip angesiedelt werden.

In dieser Aufgabe wollen wir untersuchen, ob wir mit einem recht elementaren Programm diese Speicherstruktur sichtbar machen können. Untersuchen Sie dazu als erstes den folgenden einfachen C++ Code (welchen Sie allein mit Ihren Python-Kenntnissen verstehen sollten):

```
#include <iostream>   // header needed to write something to the screen
#include <sys/time.h> // header needed for the timing
#include <math.h>     // header needed for the power-function

// specify how many times we want to run the summation
unsigned int runs = 50;

// this is a parameter used in the non-linear access of memory.
int stride = 6;

// the main program
int main() {
    // step through different array sizes and to the summation each time.
    int steps = 100;
    for (int i=0; i<steps; i++) {
          // choose the array size on a logarithmic scale, pow(b, e) gives the number b^e.
          unsigned int array_size = (unsigned int)(pow(2, (double)i*13.0/(double)steps+9.8));

          // allocate memory for array_size integer numbers
          int* big_array = new int[array_size];

          // step through array and fill it with some values
          for (int j=0; j<array_size; j++)
            big_array[j] = j;

          // get the timestamp before the main work begins
          struct timeval t0;
          gettimeofday(&t0, NULL);

          // perform the summation 'runs' number of times to gather some statistical data
          for (int k=0; k<runs; k++) {
            // sum up the values in memory in a non-trivial succession
            long sum = 0;
            for (int j=0; j<array_size; j++)
                  // j << n is a very fast way of calculating j * 2^n
                  // % is the 'modulo' operator to make sure we are always within our array's bounds
                  sum += big_array[(j << stride) % array_size];
          }

          // get a second timestamp after the main work is done
          struct timeval t1;
          gettimeofday(&t1, NULL);

          // release memory (we will allocate a new chunk of memory in the next step)
          delete [] big_array;

          // calculate how many kilobytes of memory our array has used.
          // sizeof(int) returns then umber of bytes which an integer variable uses on this computer
          double kbytes = (double)(array_size * sizeof(int))/1024.0;

          // calculate how many seconds it took to sum up the array (on average)
          double seconds = ((t1.tv_sec-t0.tv_sec)+(t1.tv_usec-t0.tv_usec)*1e-6)/(double)runs;

          // write the results to the screen. Note that we output 'time per kilobyte'.
          std::cout << kbytes << "␣" << seconds/kbytes << std::endl;
    }
}
```

Lesen Sie diesen (stark kommentierten) Quellcode, und beschreiben Sie in Worten den Ablauf des Programms. Skizzieren Sie – ähnlich zu der in der Vorlesung verwandten Notation – einen Flussablauf des Programms.

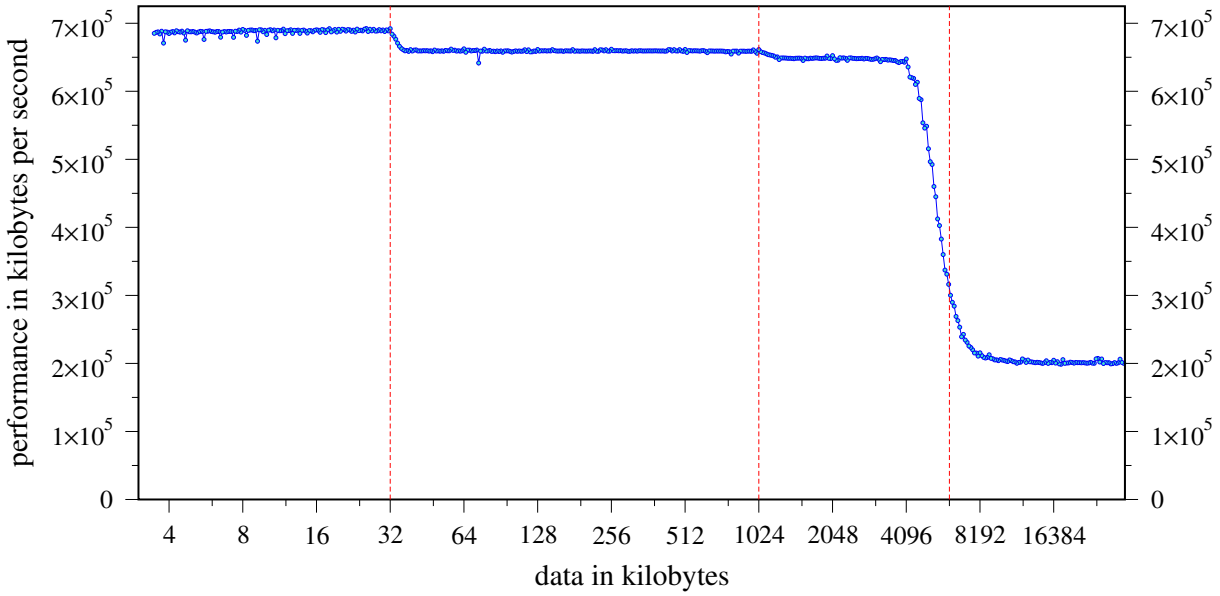

Abbildung 1: Graphische Darstellung der Ausgabe des obigen Programms

In Abbildung 1 ist die Ausgabe des Programms graphisch aufgetragen. Beschreiben Sie den Verlauf der Datenkurve, ihre charakteristischen Merkmale und interpretieren Sie letztere. Können Sie sich erklären, warum in der Summenschleife des Programms der Speicher nicht sequentiell abgearbeitet wird?

Zusatzaufgabe (ohne Punkte).– Kompilieren Sie den Code auf einem Rechner Ihrer Wahl und lassen Sie ihn dort laufen. Unter Linux/Unix/OS X können Sie dies in der Shell mit den folgenden Kommandos machen:

```
g++ cache.cpp -o cache_test
./cache_test
```

Versuchen Sie, mit Hilfe der gewonnenen Daten die Größe der verschiedenen Cache-Speicher auf Ihrem System zu identifizieren. Anschließend vergleichen Sie (wenn möglich) mit den tatsächlichen Systemdaten. Auf einer Linux/Unix-Maschine können Sie letztere über den Befehl

```
cat /proc/cpuinfo
```

ausgeben.

Untersuchen Sie dabei auch, welche Ausmaße die CPU in Ihrem Rechner hat.

15. Bessel-Funktion via numerische Integration *5 Punkte*

Die Mathematik kennt eine Reihe von speziellen Funktionen, die gerade in der Physik eine besondere Relevanz entwickeln. Dazu gehören insbesondere die **Bessel-Funktionen**, welche in Problemen mit sphärischer oder zylindrischer Symmetrie auftreten – etwa in der Elektrodynamik oder der Quantenmechanik.

Die Besselfunktionen wollen wir daher in dieser Übung ein wenig näher numerisch untersuchen. Sie ergeben sich als Lösungen der Differentialgleichung

$$x^2\frac{d^2y}{dx^2}+x\frac{dy}{dx}+(x^2-\alpha^2)y=0\,.$$

Tatsschlich gibt es unendlich viele Lösungen dieser Differentialgleichung, die für ganzzahlige, positive $\alpha = n$ überlicherweise als J_n bezeichnet werden. Ein Weg, die einzelnen Lösungen J_n zu definieren, ist anhand der folgenden Integralform:

$$J_n(x)=\frac{1}{\pi}\int_0^{\pi}\cos(ny-x\sin(y))\,\mathrm{d}y\,.$$

Unser Ziel ist es nun, diese Funktionen numerisch zu berechnen und dabei zwei Integrationsalgorithmen, die Sie in der Vorlesung kennengelernt haben, zu vergleichen. Zuvor sei angemerkt, dass Besselfunktionen und andere spezielle Funktionen in der Bibliothek scipy in dem Modul scipy.special vorhanden sind. Sie können so überprüfen, ob Ihr Code korrekt funktioniert.

Die erste Methode, die Sie kennengelernt haben, ist die **Trapezregel**. Wie der Name schon suggeriert, wird der Flächeninhalt durch eine Reihe von Trapezen approximiert. Abhängig vom Verhalten der Funktion, dem Diskretisierungsschritt und der Größe des Integrationsbereichs kann dies bereits gute Ergebnisse liefern. Genauer ist allerdings die **Simpson Regel**, bei der die Funktion auch diskretisiert, aber dann intervallweise durch eine Parabel genähert wird.

Implementieren Sie ein Programm basierend auf der Beschreibung in der Vorlesung. Integrieren Sie dann die oben genannte Besselfunktion bis $n = 5$ und vergleichen Sie die Ergebnisse. Plotten Sie auch die exakte Lösung, auf die Sie mit scipy.special.jn(n,x) zugreifen können. Wählen Sie einen Integrationsbereich von $(0,50)$ und schauen Sie, wie stabil Ihre Lösung in diesem Intervall ist. Betrachten Sie auch den Integranden und versuchen Sie eventuelle Abweichungen zwischen den beiden Verfahren zu erklären.