

# C++ für Physiker

15.11.2016

Jan Tatz

[jantatz@itp.tu-berlin.de](mailto:jantatz@itp.tu-berlin.de)

# Introduction

Ein Problem ist nicht analytisch lösbar?

A) Ein neues Problem suchen

B) Problem mit cleveren  
Näherungen vereinfachen

C) Aufgeben

D) Numerische Lösung

Mechanik: Vielteilchen Systeme, Chaos

QM: Schrödinger-Gleichung für nicht-langweilige Potentiale

Festkörperphysik: Banddiagramme, Spektren

Hydrodynamik, Molekulardynamik

Biologische Systeme

→ Validierung von physikalisch cleveren Näherungen

# ReadMe

- Literatur:
  - [stackoverflow.com](https://stackoverflow.com)
  - [C++ numerical recipes](#)
  - [wikibooks: C++](#)
- Kurse
  - PC-Pool: [Grundlagen wissenschaftlicher Programmierung](#) (SS)
  - Projektgruppe Praktische Mathematik: [C](#) (WS & SS)
  - Astrophysik: [Numerikum I](#) (WS)

# & More

- Performante Programmiersprache für aufwändiges Problem: C/C++, Fortran
- Scriptsprache: bash, Python
- Auswertung: Mathematica, gnuplot, Matlab, Python-matplotlib, octave, Excel(?)

# Was braucht man zum Programmieren?

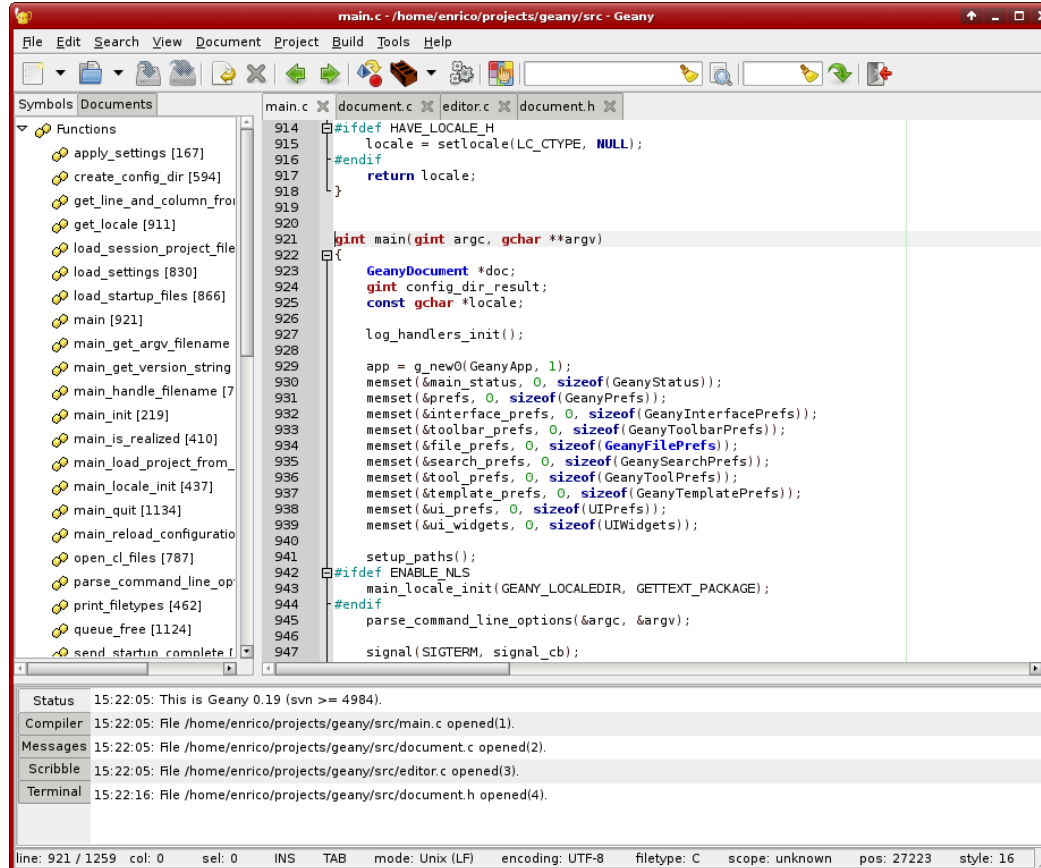
## Linux

1. guten Editor: geany
2. compiler: g++
3. Installation:  
sudo apt-get install g++ geany

## Windows

1. guten Editor: geany
2. compiler: g++ (mingw)
3. Installation guide:  
<http://blog.stijn-dhaese.be/2008/03/how-to-install-mingw-with-geany/>  
(see Appendix)

# Geany Look



# 1. C++ Beispielprogramm

hello\_world.cpp

```
// comment: here be a description of this program

#include <iostream>
using namespace std;

int main(){
    cout << "Hello World!" << endl;
}
```

compile with:

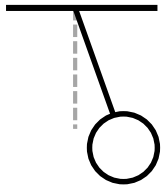
g++ hello\_world.cpp -o hello\_world.exe

# Ubiquitous physical problem: Harmonic Oscillator

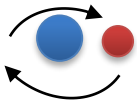
Masse an Feder



Pendel



Elektronenauslenkung



Newtonsche  
Bewegungsgleichung:

$$m\ddot{x} = -kx$$
$$\Leftrightarrow \ddot{x} = -\omega^2 x$$

Variablen /  
Parameter:

$x$  – Auslenkung aus Ruhelage  
 $\omega$  – Eigenfrequenz

1<sup>st</sup> order ODEs  
(good for numerics):

$$\dot{x} = p$$
$$\dot{p} = -\omega^2 x$$

$x$  – position  
 $p$  – momentum



# Numerical solver

(explicit version)

1<sup>st</sup> order ODE:

$$\dot{x} = f(x)$$

finite difference approximation:

$$\frac{x_{t+1} - x_t}{\Delta t} = f(x)$$

rewrite:

$$x_{t+1} = x_t + \Delta t f(x)$$

computer code:

```
xneu = x + dt*( f(x) );
```

# Program Outline

1. Set up variables for numerical simulation
2. Set parameter values
3. Set up initial conditions
4. Solve differential equation
5. Save data

# Complete program I

solver.cpp

```
// comment: here be a description this program
// compile with:
// WIN: g++ -Wall -O3 -static-libstdc++ -static-libgcc -std=c++0x solver.cpp -o solver.exe
// Linux: g++ -Wall -O3 -std=c++0x solver.cpp -o solver.exe

#include <iostream>
#include <fstream>
#include <vector>
using namespace std;
```

# Complete program II

```
int main(){  
    // parameters  
    float dt=0.01;  
    int nsteps=10000;  
    int savesteps=10;  
    int len=nsteps/savesteps;  
    vector<float> x_out(len);  
    vector<float> p_out(len);  
    float xneu, pneu;  
  
    // initial condition  
    float x0=0.0;  
    float p0=0.0;
```

```
    // model parameters  
    float w2=1.0;  
  
    // time integration  
    float x=x0;  
    float p=p0;  
  
    int counter=0;
```

# Complete program III

```
for(int step=0; step<nsteps; step++){  
  
    // status update  
    if(!(step%100)) cout << "step: " << step << endl;  
  
    // save data for output  
    if(!(step%savesteps)){  
        x_out[counter] = x;  
        p_out[counter] = p;  
    }  
  
    // euler step  
    xneu = x + dt*( p );  
    pneu = p + dt*( -w2*x );  
  
    // update state  
    x=xneu;  
    p=pneu;  
}
```

# Complete program IV

```
// save data  
savedata(x_out,p_out,len);  
  
}
```

```
void savedata(vector<float> &x_out, vector<float> &p_out, int len){  
  
    ofstream out("/home/jan/Desktop/output.dat");  
    for(int i=0; i<len; i++) out << x_out[i] << " " << p_out[i] << endl;  
  
}
```

# Windows/Linux: Mathematica

```
data=Import["/home/jan/Desktop/output.dat"];  
Dimensions[data]
```

```
plot=ListLinePlot[Transpose@data,  
    Frame->True,  
    FrameLabel->{„position x", „momentum p"},  
    FrameStyle->Directive[20,FontFamily->"Helvetica",Black],  
    ImageSize->500,  
    AspectRatio->1  
]
```

```
Export["/home/jan/Desktop/pic_ho.png",plot]
```

# Linux: gnuplot

call gnuplot from console: „gnuplot“ or use a bash script

```
#!/bin/bash
file="$HOME/Desktop/output.dat"

#gnuplot call
gnuplot << EOF

# output as png pictures
set term pngcairo enhanced size 640,480 font "Arial"

set out "$HOME/Desktop/pic.png"

# Options
unset key    # no legend
set grid     # grid lines
```

```
# Plotrange
set xrange [-2.0:2.0]
set yrange [-2.0:2.0]

# Axes labels
set xlabel „position x“
set ylabel „momentum p“

# Plot
plot "${file}" u 1:2

EOF
```



# Common sources for C/C++ errors

- “;” am Ende vergessen
- Klammern “{”, “}” vergessen
- falsche Datentypen: float f =3/5 (0), float f=3.0/5.0 (0.6)
- pointer Fehler bei Array Verwendung in Funktionen -> besser STL vector verwenden
- Index Fehler in Arrays ( $\pm 1$ ) -> Speicherzugriffsfehler/segfaults

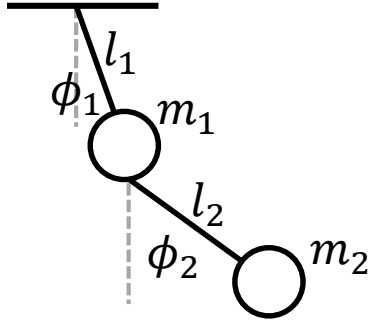
## Debuggen:

- erwartete Variablenwerte im Programm ausgeben:  
`cout << “name: “ << variable << endl;`

# Future steps...

- Implicit/semi-implicit solvers: Leap-frog, Verlet, symplectic
- Higher-order solvers: Runge-Kutta, Adams-Bashforth
- Spatial Discretization: Finite Differences/Finite Elements
- OpenMP: CPU Parallelization
- CUDA: High Performance Computing instead of Gaming
- boost: useful functions: commandline parameters, OS portability, etc
- C++11, 14: new features: auto, lambda expressions, for-each loop, exact time measurement
- git: version control software – never lose code again
- valgrind: check code for memory errors

# Ubiquitous practice problem: Double pendulum



$$L = T - U$$

$$T = T_1 + T_2 = \frac{m_1 + m_2}{2} (l_1 \dot{\phi}_1)^2 + \frac{m_2}{2} [(l_2 \dot{\phi}_2)^2 + 2l_1 l_2 \dot{\phi}_1 \dot{\phi}_2 \cos(\phi_1 - \phi_2)]$$

$$U = U_1 + U_2 = -(m_1 + m_2)gl_1 \cos \phi_1 - m_2 gl \cos \phi_2$$

using ELG:

$$\frac{d}{dt} \frac{\partial}{\partial \dot{q}_i} L - \frac{\partial}{\partial q_i} L = 0$$

equations of motion:

$$0 = \ddot{\phi}_1 + \frac{g}{l_1} \sin \phi_1 + \frac{m_2}{m_1 + m_2} \frac{l_2}{l_1} [\cos(\phi_2 - \phi_1) \ddot{\phi}_2 + \sin(\phi_1 - \phi_2) \dot{\phi}_2^2]$$

$$0 = \ddot{\phi}_2 + \frac{g}{l_2} \sin \phi_2 + \frac{l_1}{l_2} [\ddot{\phi}_1 \cos(\phi_2 - \phi_1) - \sin(\phi_1 - \phi_2) \dot{\phi}_1^2]$$

# Real world problem: Love affair dynamics [1]

problem:

Romeo & Julia lieben sich.

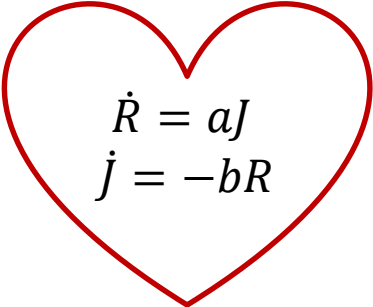
Spannender: Romeo liebt Julia. Aber Julia mag keinen liebestollen Romeo. Sie fühlt sich erst zu ihm hingezogen, wenn er ihr die kalte Schulter zeigt. Sobald sie ihn liebt, steht er auch wieder auf sie.

variables:

$R$  – Romeos Liebe (zu Julia)

$J$  – Julias Liebe (zu Romeo)

ODE system:


$$\begin{aligned}\dot{R} &= aJ \\ \dot{J} &= -bR\end{aligned}$$

$$a, b \in \mathbb{R}^+$$

# Appendix: Installation on Windows

## Part 1. Installing MinGW.

First we are going to install MinGW, the best way to do this, is using the MinGW Auto Installer.

- [Download it](#) and execute it
- Select **download and install** and click **next**
- Agree to the license Agreement
- Chose the current version
- At this step you can chose the components you need, check the **g++ compiler** like in the screenshot, for now this is more then enough, if you need more components in the future, you always can run the installer again and add more components.
- Use the default destination folder (C:\MinGW) (Make sure there are no spaces in the path!)
- A few more clicks on **“next”** and then **“install”**.
- Now, go to start, right click on **“Computer”** > **“Properties”** > **Advance system settings** > **“Advance”** > **“Environment Variables...”** > **“system variables”** > **“Path”** > **“Edit..”**
- In the **“Variable Value”** you add **“;C:\MinGW; C:\MinGW \bin; C:\MinGW\libexec\gcc\mingw32\5.3.0;”**, be sure that the 5.3.0 is the same as your MinGW version.
- Hit **“OK”** a few times and close down all windows.
- In your explorer, go to **“C:\MinGW\libexec\gcc\mingw32\5.3.0”**. Copy all the files in that folder to **“C:\MinGW\bin”**

And that’s all for the first part, go ahead and open a Command line window and enter **“g++ –version”**

# Windows: Install Geany

- Download the win32 build for geany from there [website](#) and install it.
- After the install, run it an File > New “c++ source file”
- Go to Build -> Set Includes and Arguments.
  - Change the Build settings to **g++ -Wall -O3 -static-libstdc++ -static-libgcc -std=c++0x -o "%e" "%f"** [\(fig.5\)](#)

And hit OK.