

# Wissenschaftliches Programmieren für Ingenieure

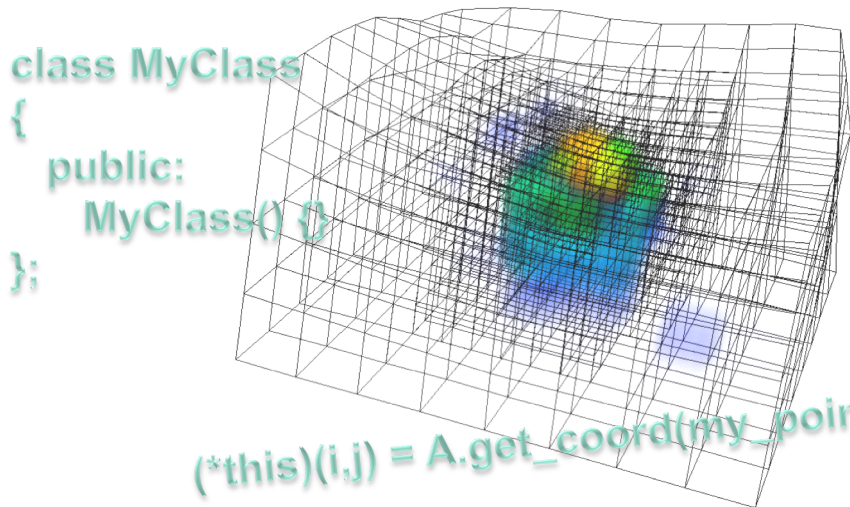
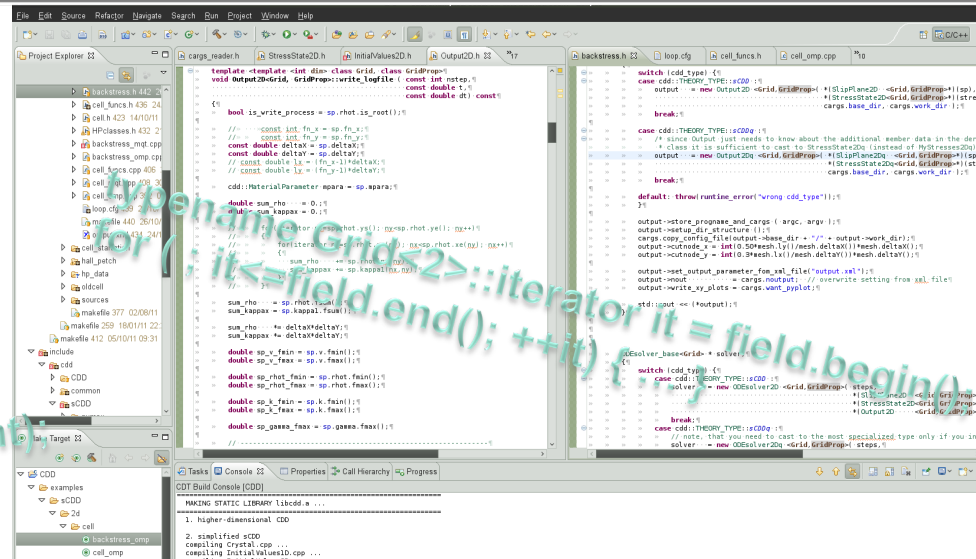
M. Stricker, T. Achkar, A. Trenkle, Dr. D. Weygand

## Übung 4:

### Ein Workflow für wissenschaftliches Rechnen (oder auch gawk, gnuplot und einige Shell-Kommandos)

Institute for Applied Materials – Computational Materials Science (IAM-CMS)

KIT – The Karlsruhe Institute of Technology – University of the State of Baden-Wuerttemberg, GERMANY

# Simulationen starten, auswerten und darstellen

## Info


Am Beispiel einer Parameterstudie des „Game of Life“ wird in dieser Übung die sinnvolle Kombination der bisher gelernten Werkzeuge im wissenschaftlichen Programmieren verdeutlicht.

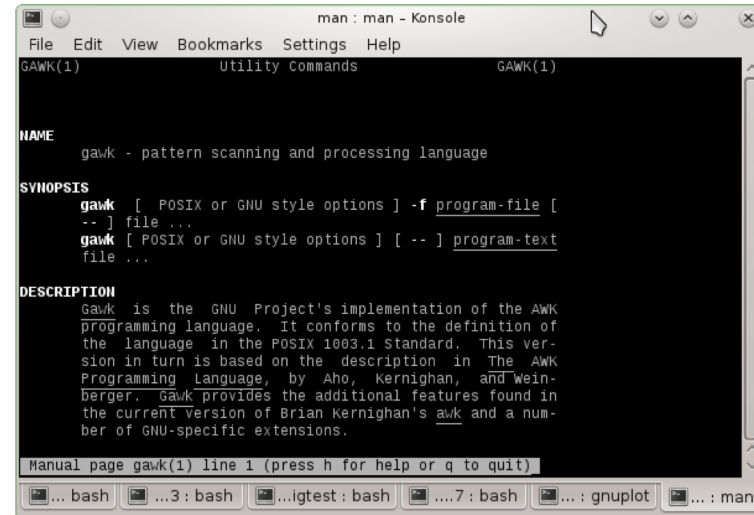
Sie werden heute:

1. Simulationen per Script und Kommandozeilenargument starten
2. Weitere Werte/Ergebnisse aus einzelnen Simulationen extrahieren
3. Diese ermittelten Werte darstellen
4. Eine weitere Bearbeitung der ermittelten Daten vornehmen

# Vorneweg: man-pages von Kommandos

## Info

- (Fast) jedes Programm in der Kommandozeile hat eine sogenannte man-page, eine Art Bedienungsanleitung, die direkt im Terminal angezeigt wird.
- Anzeige der man-page für z.B. gawk erfolgt über `user@comp> man gawk` liefert. 
- Die man-page gibt Auskunft über die Verwendung des Programms und die Reihenfolge der erwarteten Parameter.
- Das Programm `seq`, das im Folgenden verwendet wird, generiert Zahlenfolgen, die innerhalb der Shell z.B. für Schleifen verwendet werden.



# Shell/gawk/gnuplot Kommandos

## Info

Shell-Kommandos (siehe erste Übung) lassen sich auch als sogenannte Scripte (engl. *scripts*) in einer Datei speichern. Vor allem bei mehrfach verwendeten Befehlsketten ist diese Art der Verwendung sinnvoll und zeitsparend.

Die Programmiersprache der Kommandozeile sowie gawk gehören zu den sogenannten Skriptsprachen. Die Syntax beider Sprachen ist ähnlich. Das Ziel dieser Sprachen ist eine schnelle Bearbeitung von textbasierten Daten. Auf die in höheren Programmiersprachen oft geforderte Deklaration von Variablen wird meistens verzichtet, was Zeit spart, aber auch zu Problemen führen kann.

## Todo

Ziel: Vertraut machen mit der Syntax

Zwei einfach Beispiele von Shell-Kommandos (Ausgabe auf der Kommandozeile).

In Konsole:

```
user@comp> echo Hello
```

```
Hello
```

```
user@comp> for i in $(seq 1 3); do echo $i; done
```

```
1
```

```
2
```

```
3
```

# Weitere Beispiele

Todo

Ziel: Vertraut machen mit der Syntax

- Erstellung vieler Ordner per Schleife (in der Konsole)

```
user@comp> for i in $(seq 1 5); do mkdir config$i; done
user@comp> ls
config1 config2 config3 config4 config5
```

- Andere Schrittweite der Schleife (in der Konsole)

```
user@comp> for i in $(seq 1 2 10); do echo $i; done
1
3
5
7
9
```

- Löschen von erstellten Ordnern (in der Konsole)

```
user@comp> for i in $(seq 1 5); do rm -rf config$i; done
```

# Vorbereitung: Kompilieren des Game of Life

Todo

Ziel: Programm vorbereiten/kompilieren

- Herunterladen der Dateien für UB05 von Ilias
- Alle Dateien in einen Ordner `/home/.../UB04/` verschieben (in Konsole `mv`)
- Falls nötig entpacken (in Konsole: `tar -xvcf Archivname.tgz`)
- Game of Life kompilieren (in Konsole: `make`)  
Der Unterschied in dieser Version von Game of Life liegt in der Erzeugung der Zufallszahlen:  
Der Generator wird bei jedem Aufruf mit der Systemzeit initialisiert und liefert dann bei jedem Aufruf andere Zahlen. Siehe:  
`functions_game.cpp`  
`srand(time(NULL));`
- Falls dies nicht hinzugefügt wird, liefert der Zufallsgenerator bei jedem Aufruf des Programms die gleichen Zufallszahlen und damit eine identische „zufällige“ Anfangsverteilung.

# Vorbereitung: Erzeugen der Ordnerstruktur für die Simulationen

Todo

Ziel: Ordnerstruktur vorbereiten

- Pro Anfangsstruktur (Dichteanteile  $\text{frac}=0.3/0.5/0.7$ ) sollen 10 Simulationen auf einem Gitter von  $100 \times 100$  mit unterschiedlichen anfänglichen Zufallsverteilungen gerechnet werden.
- Ordnerstruktur wie oben erklärt einzeln und mit Schleifen erstellen:  

```
./runs/  
    /frac0.3  
    /frac0.5  
    /frac0.7  
        /config1  
        /config2  
        ..  
        /config10
```
- Die Simulationen pro Anfangsdichte werden nun mit einem Script gestartet. Dazu muss im Script `run_sim.sh` zuerst der Pfad zur ausführbaren Datei (siehe nächste Folie) sowie die Berechtigung mit  

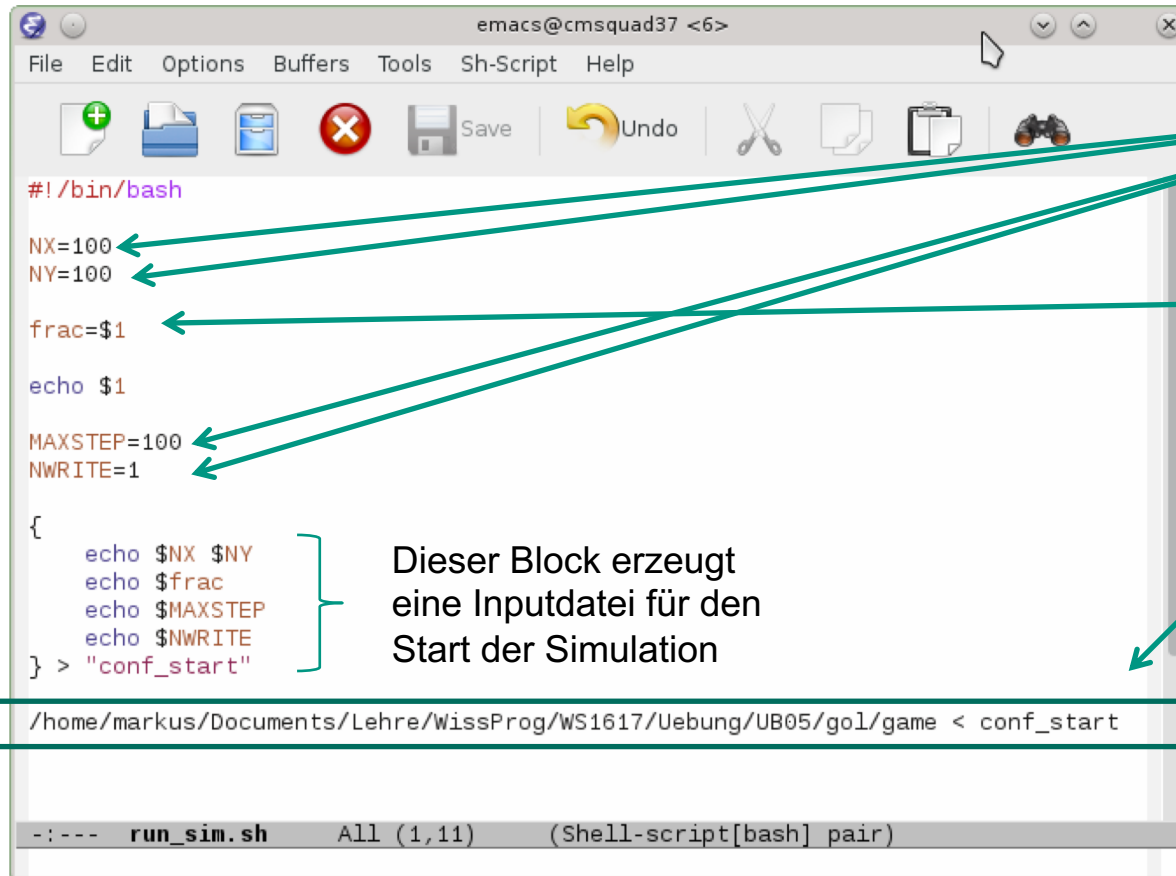
```
user@comp> chmod +x run_sim.sh
```

verändert werden. **Beachten Sie dies bei allen Skripten!!**

# Struktur von `run_sim.sh`

## Info

- Um eine ganze Reihe von Game of Life Simulationen nicht von Hand starten zu müssen, ist das Script `run_sim.sh` bereit gestellt:



```

#!/bin/bash

NX=100
NY=100

frac=$1

echo $1

MAXSTEP=100
NWRITE=1

{
    echo $NX $NY
    echo $frac
    echo $MAXSTEP
    echo $NWRITE
} > "conf_start"

/home/markus/Documents/Lehre/WissProg/WS1617/Uebung/UB05/gol/game < conf_start
  
```

Variablen  
innerhalb des  
Scripts

Variable/Wert wird aus  
Kommandozeile übernommen

Dieser Block erzeugt  
eine Inputdatei für den  
Start der Simulation

Ausführen des Programms mit den  
gewählten Parametern.  
**Diese Zeile muss an den  
eigenen Pfad zum kompilierten  
Game of Life angepasst werden!**



# Start einer einzelnen Simulation per Script

Todo

Ziel: Simulation mit Script starten

- Das Script `run_sim.sh` lässt die Variation der Anfangsdichte mit einem Kommandozeilenargument zu und startet dann die Simulation mit einem Gitter von  $100 \times 100$ .

- Für einen ersten Test, gehen Sie in Ordner

`./runs/frac0.3/config1`

Dort das Script aufrufen:

```
user@comp> /path/to/script/run_sim.sh 0.3
```

Die `0.3`, die nach dem Aufruf steht wird im Script als Variable `frac` verwendet und in die Inputdatei geschrieben.

Das erste Argument (`0.3`) wird im Skript über `$1` angesprochen!!

Danach den Ordner auf Veränderung überprüfen (`ls -l`).

- Nun sollen sich im Ordner 101 Simulationsschritte befinden (`field0.dat` bis `field100.dat`).

# Start vieler Simulationen mit Script:

## Kombination von Kommandozeile mit Script

Todo

Ziel: Kombination von Script und Kommandozeile

- Durch die Kombination der Kommandozeile mit dem `run_sim.sh` Script lässt sich viel Arbeit sparen: Jede Simulationsreihe mit gleicher Anfangsdichte wird nun mit einem Befehl gestartet.
- Gehen Sie in Ordner `./runs/frac0.3/` und führen Sie aus:  

```
user@comp> for i in $(seq 1 10); do cd config$i/  
/path/to/script/run_sim.sh 0.3; cd ..; done
```

### Zerlegung der Befehlskette:

- `for i in $(seq 1 10); do **Befehle**;` done      Schleife
- `cd config$i`      Ordnerwechsel
- `/path/to/script/run_sim.sh 0.3`      Führe Script aus
- `cd ..`      Ordnerwechsel zurück
- Nun haben wir 10 verschiedene Simulationen mit gleicher Anfangsdichte erzeugt.
- Wiederholen Sie das Ganze für die Fälle mit `frac=0.5` und `frac=0.7` in den entsprechenden Ordnern.

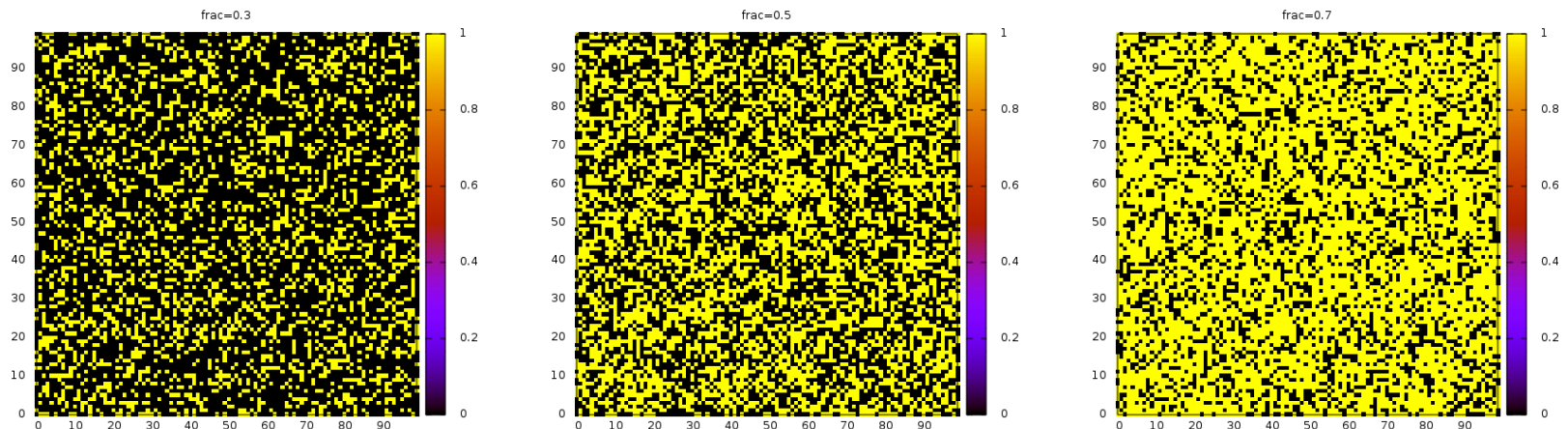
# Überprüfung einzelner Simulationen auf Richtigkeit

Todo

Ziel: Überprüfen der Simulationsparameter

- Suche Sie sich von jeder Anfangsdichte eine Simulation aus und stellen Sie deren Entwicklung mit `gnuplot` dar (siehe letztes Übungsblatt).
- Fragen, die man sich beim ersten Durchschauen stellen kann:
  - Sehe ich einen Unterschied in der Anfangsdichte?
  - Entwickeln sich die unterschiedlichen Populationen verschieden?
  - Gibt es Gemeinsamkeiten?

Anfangsverteilung für die drei Anfangsdichten



## Info

- Während der Simulation wird die Information jedes Kästchens (0/1) für jeden Zeitschritt in eine Datei geschrieben. Die Dateistruktur ist wie folgt:  

```
user@comp> cat field0.dat
```

```
0 0 0
0 1 1
0 2 0
0 3 1 usw.
```
- Die ersten beiden Spalten sind die Position, die dritte Spalte der Wert.
- Bei der Initialisierung der Anfangsstruktur bedeutet die Variable `frac` der Anteil der Kästchen in denen eine 1 steht. Diese Information ist nicht explizit gegeben, lässt sich aber aus den Ergebnissen berechnen.
- Sie berechnen nun für jeden Zeitschritt von allen Simulationen die Variable `frac` und stellen den Verlauf dann über dem Simulationsschritt dar.
- Für die Berechnung der Dichte verwenden wir `gawk`.

# gawk

Todo

Ziel: Vertraut machen mit der Syntax

gawk/awk ist eine Programmiersprache um Textdateien zu bearbeiten und zu manipulieren. gawk liest Textdateien zeilenweise.

Die Syntax von gawk folgt dem Schema

```
user@comp> gawk BEDINGUNG {ANWEISBLOCK}
```

- Gehen Sie in den Ordner einer Simulation, z.B.  
./runs/frac0.3/config1/ und führen Sie folgenden Befehl aus  
user@comp> gawk '(\$3>0){SUM+=1} END{print SUM/NR}'  
field0.dat

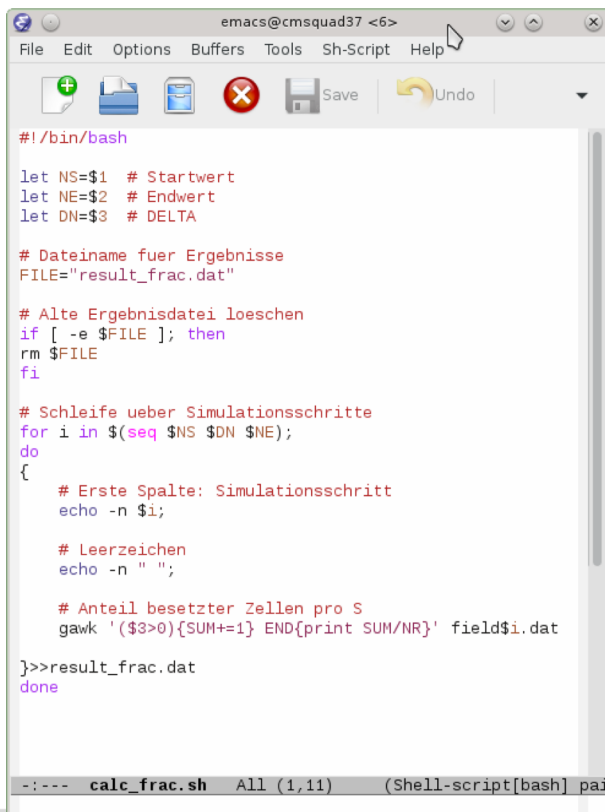
Zerlegung der Befehlskette:

- |                           |                                                                                          |
|---------------------------|------------------------------------------------------------------------------------------|
| ■ gawk ' ... ' field0.dat | Programmaufruf mit Datei                                                                 |
| ■ (\$3>0)                 | Bedingung Spalte 3 > 0                                                                   |
| ■ {SUM+=1}                | Anweisung: Variable SUM + 1                                                              |
| ■ END{print SUM/NR}       | Am Ende der Datei, gib SUM/NR,<br>wobei NR die aktuelle, also letzte<br>Zeilennummer ist |

# Kombination von gawk und Kommandozeile

## Info

- Wir wollen nun nicht jeden einzelnen Schritt einer Simulation anschauen, sondern mit einem Script automatisiert eine Datei erstellen, die uns den Dichteverlauf liefert.
- Dazu kombinieren wir Shell und gawk– siehe Datei `calc_frac.sh`



```
#!/bin/bash

let NS=$1 # Startwert
let NE=$2 # Endwert
let DN=$3 # DELTA

# Dateiname fuer Ergebnisse
FILE="result_frac.dat"

# Alte Ergebnisdatei loeschen
if [ -e $FILE ]; then
rm $FILE
fi

# Schleife ueber Simulationsschritte
for i in $(seq $NS $DN $NE);
do
{
# Erste Spalte: Simulationsschritt
echo -n $i;

# Leerzeichen
echo -n " ";

# Anteil besetzter Zellen pro S
gawk '($3>0){SUM+=1} END{print SUM/NR}' field$i.dat

}>>result_frac.dat
done
```

Kommandozeilenvariablen

gawk Befehl

Schleife über alle  
Ergebnisdateien  
`field$i.dat`

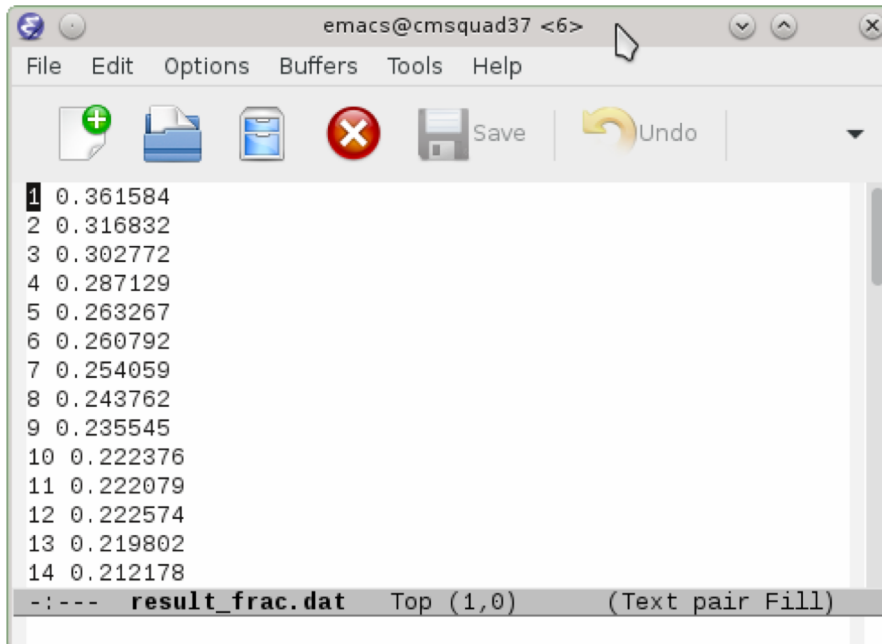
Skript muss ausführbar sein!

# Ausführen von calc\_frac.sh bei einer Simulation

Todo

Ziel: Funktion des Scripts an einer Simulation verstehen

- Gehen Sie nun in Ordner `./runs/frac0.3/config1/` und führen Sie aus  
`user@comp> /path/to/script/calc_frac.sh 0 100 1`
- Es wird nun eine Ergebnisdatei mit Namen `result_frac.dat` erstellt. Diese enthält in der ersten Spalte den Simulationsschritt, in der zweiten Spalte die Dichte:



```
1 0.361584
2 0.316832
3 0.302772
4 0.287129
5 0.263267
6 0.260792
7 0.254059
8 0.243762
9 0.235545
10 0.222376
11 0.222079
12 0.222574
13 0.219802
14 0.212178
```

-: --- result\_frac.dat Top (1,0) (Text pair Fill)

# Ausführen von `calc_frac.sh` bei allen Simulationen

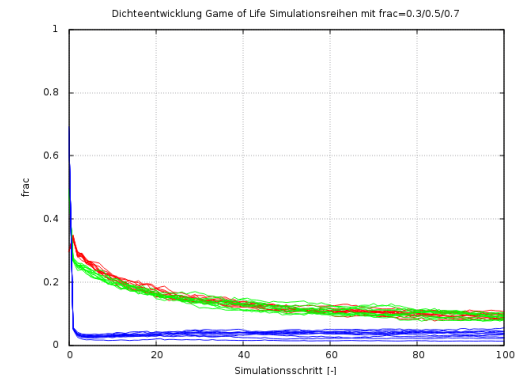
Todo

Ziel: Funktion des Scripts auf alle Simulationen anwenden.

- Gehen Sie nun alle Simulationsreihen durch und erzeugen Sie für jede Simulation den Dichteverlauf.
- Hinweis: Kombination der Kommandozeile sowie Script `calc_frac.sh`, wie in Folie 9.
- Nachdem nun alle Dichteverläufe ausgewertet sind, stellen Sie die Verläufe der unterschiedlichen Reihen in Gnuplot dar. Kopieren Sie Datei `plot_frac_evo.gp` in Ordner `./runs/` und gehen Sie dort hin.

Gnuplot starten und:

```
gnuplot> load 'plot_frac_evo.gp'
```





# Weitere Auswertung (Zusatz, ohne Vorlage)

## Todo

- Der nächste Schritt der Auswertung ist die Mittelung der Simulationsreihen mit gleicher Anfangsdichte.
- Programmieren Sie ein Skript/Kommandozeilenbefehl, der die `result_frac.dat` Dateien einer Anfangsdichte öffnet und dann pro Simulationsschritt über alle Simulationen mittelt. Das Ergebnis soll dann in einer Datei im jeweiligen Grundordner `./runs/frac0.X/result_frac_avrg.dat` gespeichert werden. Die erste Spalte soll den Simulationsschritt angeben, die zweite den Mittelwert über alle 10 Simulationen.
- Die Auswertung des Mittelwerts um die Standardabweichung pro Simulationsschritt ergänzen.
- Die gemittelten Dichteverläufe und Standardabweichung in Gnuplot darstellen.