Computergestütztes wissenschaftliches Rechnen
SoSe 2004

Alexander K. Hartmann, Universität Göttingen

April 19, 2004

1 Einführung

1.1 Warum Computerphysik?

Vorteil analytische Theorie: große Systeme
Nachteil: Nur GANZ wenige Probleme lösbar, insbesondere kaum Dynamik

Computerphysik: gerade umgekehrt → viele (neue) Anwendungen, z.B. kombinatorische Optimerung, Molekularbiologie, Neuronale Netze, Graphen, ...
Algebraische Umformungen (Maple)
Auswertung/Darstellung von Daten
Selbst manche Experimente BRAUCHEN Simulationen (CERN Ereignisbestimmung)

Späteres außeruniversitäres Berufsbild: Überwiegend Informationstechnik, kommt kaum im Studium vor

Praktischer Grund: Pflichvorlesung
Basis für Computerübungen (z.B. dieses Semester QM I)
Schein als Informatik Nebenfach Schein

1.2 Semesterübersicht

Hier wenig Numerische Mathematik → NAM

Programmiersprache: C (Unix/Linux)

Internet:
`www.theorie.physik.uni-goettingen.de/~hartmann/nwgruppe/teaching.html`

14.4 Einführung, Literatur, Populationsdynamik

21.4 Numerische Fehler, Libraries, Nullstellen, DGLs

28.4 Zufallszahlen

5.5 Algorithmen

12.5 Molekulardynamik I

19.5 Molekulardynamik II

26.5 Datenstrukturen, Ereignis-gesteuerte Simulationen

2.6 Perkolation

9.6 Bibliotheken, Datenauswertung

16.6 Monte Carlo I

23.6 Monte Carlo II

30.6 Quanten Monte Carlo `oder` Polymere auf Gittern

7.7 Graphen /-algorithmen

14.7 Optimierungsalgorithmen/ Ausblick

Übungen: Prof. T. Pruschke. Vorbesprechung/Termine: Fr 16.3, 10ct, Cip-Pool.

1.3 Literaturhinweise

- R. Sedgewick, *Algorithms in C*, (Addison-Wesley, Reading (MA) 1990)
- T.H. Cormen, S. Clifford, C.E. Leiserson, und R.L. Rivest, *Introduction to Algorithms*, (MIT Press 2001)
- B.W. Kernighan und R. Pike, *The Practice of Programming*, (Addison-Wesley, Reading (MA) 1999)

  A.K. Hartmann und H. Rieger, *A practical guide to computer simulation* `http://www.theorie.physik.uni-goettingen.de/support/Rechner /Programmierung/simulation_guide.ps.gz`
- W.H. Press, S.A. Teukolsky, W.T. Vetterling, und B.P. Flannery, *Numerical Recipes in C* (Cambridge University Press, Cambridge 1995); see also `www.nr.com`
- W. Kinzel und G. Reents, *Physics by Computer*, (Springer, Berlin-Heidelberg-New York 1999)
- M.P. Allen und D.J. Tildesley, *Computer Simulation of Liquids*, (Clarendon Press, Oxford 1990)
- D.P. Landau und K. Binder, *A Guide to Monte Carlo Simulations in Statistical Physics*, (Cambridge University Press, Cambridge 2000).
- M.E.J. Newman und G.T. Barkema, *Monte Carlo Methods in Statistical Physics* (Clarendon Press, Oxford, 1999)

1.4 Gnuplot

Aufruf `gnuplot` in Shell. Der "Prompt" erscheint: `gnuplot >`
Kommandos eintippen.
Plotten eins x-y-dy file, z.B. `sg_e0_L.dat` (Grundzustands Energy eines Spinglases als Funktion der System Größe). File Format: 1 Punkt pro Zeile, Kommentare (`#`) ignoriert:

```
# ground state energy of +-J spin glasses
# L    e_0   error
  3 -1.6710 0.0037
  4 -1.7341 0.0019
  ...
 14 -1.7866 0.0007
```

```
gnuplot> plot "sg_e0_L.dat" with yerrorbars
```

(short:) `p "sg_e0_L.dat" w e`

→ Window mit Plot erscheint.

Andere Typen möglich, z.B. `with lines`. Plotten von multicolumn Dateien, z.B..

```
gnuplot> plot "test.dat" using 1:4:5 w e
```

Mehr Infos: `help plot` eintippen.

3d Plots: `splot`.

Achsenbezeichnungen:

```
gnuplot> set xlabel "L"
```

und Plotbefehl wieder ausführen, oder `replot`.
Ausgabe als Postscript Datei:

```
gnuplot> set terminal postscript
gnuplot> set output "test.eps"
```

und wieder plotten.

Scripts = ASCII Dateien (z.B. `commands.gp`) mit gnuplot Kommandos. Kann man automatisch generieren. Dann `gnuplot command.gp` aus der Shell.

1.5 Populationsdynamik

Iteration

$$x_{n+1} = 4rx_n(1 - x_n) = f(x_n) \quad n = 0, 1, 2, \ldots \tag{1}$$

Beschreibt Population $x \in [0, 1]$ mit Wachstumsparameter $4r$. Lineares Wachstum $4rx$, begrenzt durch nichlinearen Term $-4rx^2$.
Program `population0.c`

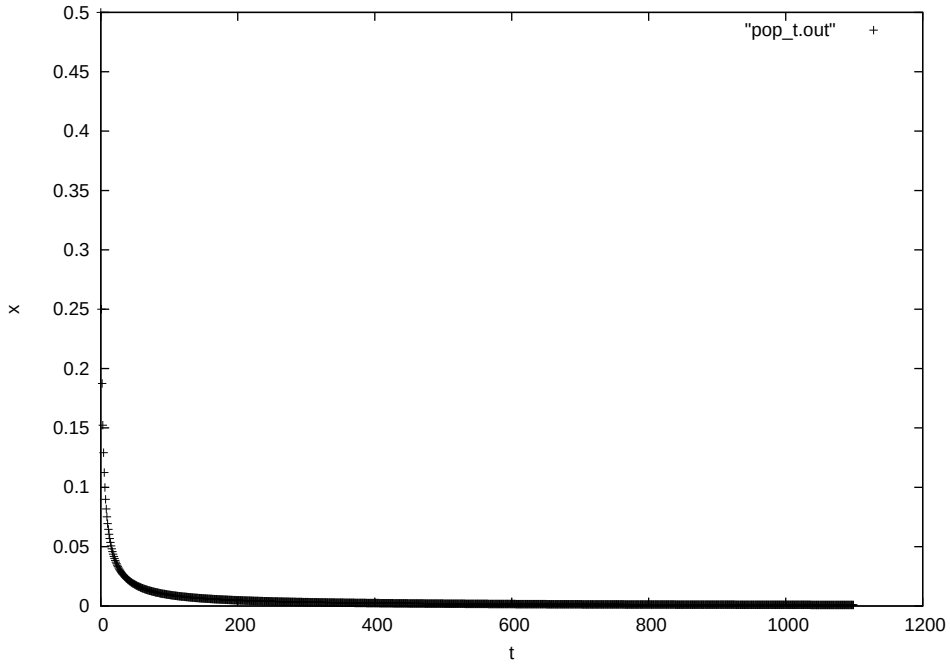
```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
  int iter;      /* iteration counter */
  double x;             /* population */
  double r;          /* growth factor */

  r = 0.249999;
  x = 0.5;
  for(iter=0; iter<1100; iter++)
  {
    printf("%d %e\n", iter, x);
    x = 4*r*x*(1-x);
  }
  return(0);
}
```

```
cc -o population0 population0.c
population0 > pop_t.out
gnuplot
gnuplot> plot  "pop_t.out"
```

ergibt:



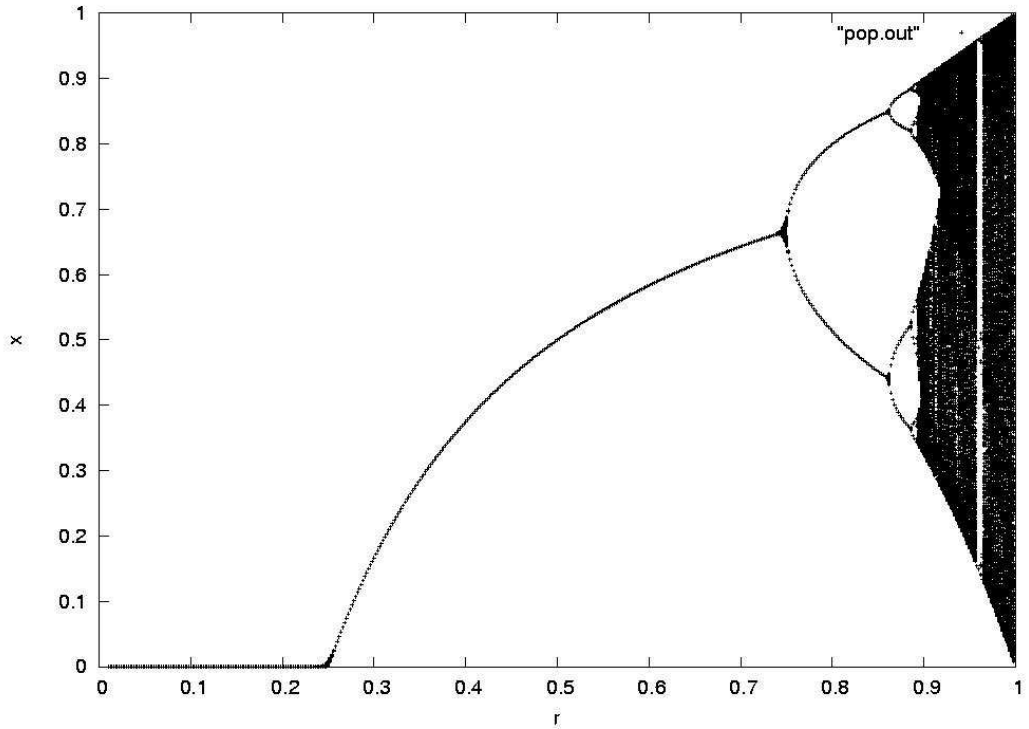
Konvergenzverhalten für verschiedene r. Program `population.c`:

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

int main(int argc, char *argv[])
{
  int iter;      /* iteration counter */
  double x;             /* population */
  double r;          /* growth factor */

  for(r=0.01; r<=1; r+=0.002)
  {
    x = 0.5;
    for(iter=0; iter<1100; iter++)
    {
      x = 4*r*x*(1-x);
      if(iter>100)
        printf("%f %f\n", r, x);
    }
  }
  return(0);
}
```

ergibt (`gnuplot> plot "pop.out" pointsize 0.3`)



Wie können wir das Verhalten verstehen?