

Matlab

- Variable diverser Typen
- Rechenoperationen
- Funktionen
- Graphik, Animation
- Eingabe, Ausgabe von Daten
- Schleifen, Verzweigungen
- Tools
 - Gleichungslöser
 - Optimierer
 - Differentialgleichungslöser
 - Eigenwertproblemlöser

Was möchte man weniger gern?

- mystische Präambeln
- Einbinden von Headern, Bibliotheken
- Deklaration von Variablen
- Bereitstellen von Speicher
- kompilieren, linken

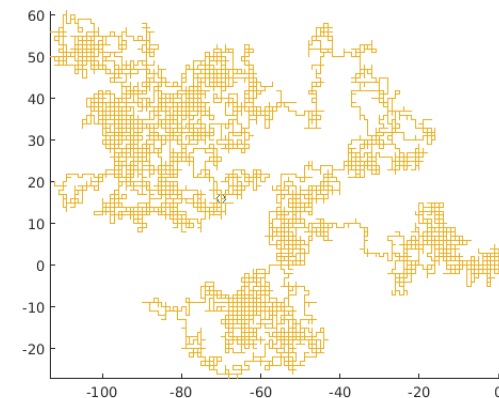
Matlab (und vergleichbare Entwicklungsumgebungen wie Octave, SciLab, julia) ersparen uns weitgehend die Sorge um lästige Nebensachen.

Dafür sind sie meist deutlich langsamer – was bei großen Problemen störend sein kann.

Modellfall: Irrfahrt

Shizuo Kakutani

A drunk man will find his way home, but a drunk bird may get lost forever.



Annahmen

- gesucht ist Weg – Folge von direkt benachbarten Knoten in kardinalem Gitter
- an jedem Knoten wird der Nachfolger fair unter den vier Nachbarn (in der Ebene) ausgelost
- analog wird im D -dimensionalen Fall unter $2D$ Nachbarn gleichverteilt gelost
- im eindimensionalen Fall wird jeweils ein Schritt nach links bzw. rechts gegangen

Bemerkungen

- keine Abhängigkeit von Vorgeschichte
- Verallgemeinerungen: unterschiedliche Wahrscheinlichkeiten für rechts/links, kontinuierliche Winkel und Schrittlängen
- Problem geht zurück auf Arbeiten von Pearson, Pólya, Rayleigh (stochastische Prozesse, Markovketten)

- in 1D und 2D gibt es eine fast sichere Rückkehr zum Ausgangspunkt
- in höheren Dimensionen ist Rückkehr wenig wahrscheinlich
- auch erneute Treffen unabhängig Herumirrender sind in 1D und 2D fast sicher, in 3D und höher nicht

Programmierung (Matlab, 2D)

- Weg wird als Matrix $m \times 2$ repräsentiert
- erste Spalte: x -Koordinate, zweite: y -Werte
- Start (erste Zeile): $[0, 0]$
- Zuwächse: entweder $\Delta x = \pm 1$ und $\Delta y = 0$, oder $\Delta y = \pm 1$ und $\Delta x = 0$
- wiederholen Schritte bis m erreicht, oder Rückkehr zum Ausgangspunkt erfolgt
- Visualisieren des Weges, ggfs. Animation
- Export der Ergebnisse – Speichern von Bild- oder Filmdateien

Implementierung (naiv)

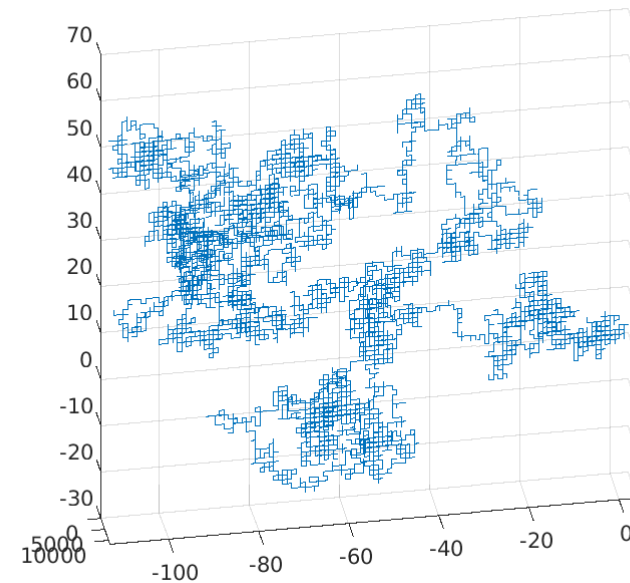
- setze Anfangspunkt
- wiederhole in Schleife
 - wähle mit `rand() < 0.5`, ob in x - oder y -Richtung gegangen werden soll
 - wähle analog, ob Schritt vor- oder rückwärts gehen soll
 - füge neuen Schritt an bisherigen Weg an
 - prüfe Abbruchbedingung
- Ausgabe

% Berechnung des Weges

```
m=12000;           % Festlegen der Weglaenge
o=floor(4*rand(m,1)); % "Wuerfeln" der Richtungswahl,
                   % Optionen 0 bis 3
z=i.^o;           % Nutzen komplexe Zahlen
d=[real(z) imag(z)]; % Zuwaechse
xy=cumsum(d);     % Aufaddierte Zuwaechse
```

% Ausgabe als Trajektorie

```
plot3(1:m, xy(:,1), xy(:,2))
set(gca, 'PlotBoxAspectRatio', [0.1, 1, 1], ...
'Ygrid', 'on', 'Zgrid', 'on')
view(80, 30)
print -dpng dogwalking % Speichern als png-Grafik
comet(xy(:,1), xy(:,2)) % simple Animation
```



Demo

- Zugang
- Vektoren und Matrizen deklarieren
- Standardfunktionen benutzen
- Daten sichern, lesen, plotten
- Hilfe nutzen
- Skripte und Funktionen
- Lineares Gleichungssystem lösen
- Nichtlineare Gleichung lösen
- Differentialgleichung lösen

Spielen mit Matrizen I

```
% vecmateig
% Spielen mit Vektoren und Matrizen
```

```
clear; clc % Workspace loeschen, Konsole saeubern
```

```
x = -3.14:0.1:3.14; y = sin(x);
```

```
% Variable kombinieren
```

```
z = [x y]; % langer Zeilervktor
```

```
z = [x, y]; % das selbe
```

```
z = [x; y]; % zwei Zeilen
```

```
% Matrix eingeben
```

```
A = [1 2 3; 4 5 6; 7 8 9; 10 11 12]
```

```
% als Vektor umordnen
```

```
a = A(:)
```

```
% Format abfragen
size(A)

% Laenge – groessere der Dimensionen
length(A)

% Anzahl der Zeilen (Hoehe)
size(A,1)

% Anzahl der Spalten (Breite)
size(A,2)

% Zeilenlaenge der transponierten Matrix
size(A',2)

length(A')
```

Navigation icons: back, forward, search, etc.

```
% plotting
% Sinus malen auf  $[-\pi + \pi]$ 

clear; clc

% Gitter im Definitionsbereich
%  $x = -3.14:0.1:3.14$ ; % etwas grob)
x=linspace(-pi,pi,63); % etwas besser
y=sin(x); % Funktionsauswertung punktweise
plot(x,y) % Punkte verbinden, Standardfarbe blau

% immer nur ein Zeile ausfuehren: markieren und CTRL+F9
plot(x,y,'r','linewidth',5) % rot und dick
plot(x,y,'g*', 'linewidth',1) % gruene Sternchen
plot(x,y,'k+', 'linewidth',1) % schwarze +Zeichen
```

Navigation icons: back, forward, search, etc.

```
% letzte (vierte) Zeile loeschen
A(4,:)=[];

size(A)

% Eigenwertproblem loesen
lam=eig(A)

% EV und Diagonalmatrix mit EW auf Diagonale
[V,Lam]=eig(A(1:3,:))

% letztes Element
A(end)

% vorletztes Element aus drittletzter Zeile
A(end-2,end-1)
```

Navigation icons: back, forward, search, etc.

```
% transeqntn
% solving  $x/2 = \sin(x)$ 

% nach Banach
x_0=2;
x_1=2*sin(x_0);

while abs(x_1-x_0)>1e-5
    x_0=x_1;
    x_1=2*sin(x_1);
    disp(x_1) % Ausgabe nur fuer Demo
end

% mit matlab-Funktion fzero
x=fzero(@(x) x/2-sin(x),2)
```

Navigation icons: back, forward, search, etc.

```

% lgs
% solving Ax=b, rechteckiger Fall

A= [ 1  2  3  4
      5  6  7  8
      9 10 11 12 ]';

% Vorgabe EINER Loesung
u= [1 1 1]';

b=A*u;

% Loesung nicht eindeutig, Warnung!
x=A\b

% Rang und Kern
rank(A)
N=null(A); N=N/norm(N, inf)

```

```

function eqnarray
disp('demo:_eqnarray')
x=[0;0];
disp(['initial_guess:_ ' num2str(x)' '_and_its_image:_ ' num2str(f(x)')])

disp('matlab_solver')
xstar=fsolve(@f,x,optimset('display','off'));
disp(['sol:_ ' num2str(xstar)',20] '_defect:_ ' num2str(f(xstar)',20))

disp('with_harder_error_bounds:')
xstarstar=fsolve(@f,xstar,optimset('display','off',...
    'tolfun',1e-14,'tolx',1e-14));
disp(['sol:_ ' num2str(xstarstar)',20] ...
    '_defect:_ ' num2str(f(xstarstar)',20))

function y=f(x)
y=x;
y(1)=4*x(1)-x(2)+x(1)^2+x(2)^2-4;
y(2)=exp(x(1))-8*x(2);

```



Differentialgleichung

```

% expogrow
% solving y'=y, y(0)=1
f=@(t,y) y;

t_0=0; t_f=1;
y_0=1;

[t,y]=ode45(f,[t_0 t_f], y_0);

plot(t,y,'b*')
disp(['Eulers_Zahl_ist_etwa:_ ' num2str(y(end),12)])
format long
disp(exp(1))
format short

```



Toolboxen

Je nach Problemstellung können diverse Toolboxen von immensem Nutzen sein.

Wir erwähnen die Optimization, Global Optimization, Statistics, und Symbolic Toolboxen so wie das PDE-Tool und Simulink.

Besonders Funktionen der Optimization Toolbox werden sehr häufig genutzt.

```
% optitest_2.m
% needs and uses obejective_2.m, obejective_2a.m, and restriction_2a.m
% myoptions = optimset('MaxFunEvals',10000);

[xopt,vopt] = fminunc('objective_2', [0 0])
[x0,v0] = fmincon('objective_2a', [0 0], [],[],[],[],[],[],[],...
    'restriction_2a') % ,myoptions)
```

```
function f = objective_2 (x);
f = (2*x(1)-1).^2+(x(2)-1).^2/9+2;
g = (x(1)-3).^2+(x(2)+2).^2-4;
f = f + 100*max(0,g).^2;
```

```
function f = objective_2a (x);
f = (2*x(1)-1).^2+(x(2)-1).^2/9+2;
```

```
function [g,h] = restriction_2a (x);
g = (x(1)-3).^2+(x(2)+2).^2-4;
h = [];
```

```
% muir1980IIa – curvefitting for data from historic paper
```

```
f=@(x,t) x(1)*exp(-x(2)*t)+x(3)*exp(-x(4)*t);
```

```
data=[ 5 1.033
        10 0.830
        15 0.800
        20 0.680
        30 0.555
        60 0.255
        91 0.235
        120 0.220
        240 0.143
        413 0.095    ];
```

```
t=data(:,1); y=data(:,2);
A=2.0; alpha=0.07; B=0.5; beta=0.004;
x0=[A, alpha ,B, beta ];
```

```
optimset('maxfuneval',1e4);
x=lsqcurvefit(f,x0,t,y,[],[], optimset('maxfuneval',1e4))
```

```
tt=linspace(t(1),t(end),101);
plot(t,y,'r+',t,f(x0,t),'b',tt,f(x,tt),'g')
```