Praktikum: Modellierung und Programmierung

Analysis und Numerik

Prof. Dr. K. Frischmuth

Institut für Mathematik, Universität Rostock

Sommersemester 2023

Ziele

Modellierung

Jedes angebotene Thema ist einem Real world problem gewidmet. Ein Untersuchungsgegenstand ist mittels eines Computermodells abzubilden, entsprechende Gleichungen sind aufzustellen und Daten zu beschaffen.

Programmierung

Die aufgestellten Gleichungen sind für die gewählten Situationen und Szenarien mittels eines eigenen Programms zu lösen, die Ergebnisse auszuwerten und im Sinne des Modells wie des Untersuchungsobjekts zu interpretieren.

Dokumentation

Zum erfolgreichen Abschluss eines jeden Projektes gehört die Präsentation und Dokumentation des gesamten Ablaufs von der Problemstellung über die Modellannahmen, die Implementation und die Ergebnisse.
Dies ist als Miniaturausgabe einer BA-Arbeit bzw. einer Abschlussarbeit anzusehen und sollte den einschlägigen Standards genügen.

Phasen

Recherche

Es empfiehlt sich zu jedem Thema zunächst einen Überblick über Literatur und Online-Ressourcen zu gewinnen. Es gibt stets mehrere Herangehensweisen, bei deren Wahl der Betreuer zu konsultieren ist.
Die angebotenen Kurse zum Umgang mit Daten, Texten, Formeln und Grafiken sollen die Auswahl der richtigen Werkzeuge erleichtern.

Implementation

Es ist anzustreben gut strukturierte, verständliche und zwecks Nachvollziehbarkeit kommentierten Programmcode zu erzeugen. Dieser muss lauffähig sein und darüber hinaus zu gewählten Daten korrekte Ergebnisse liefern.
Typischerweise kommt man zu einem positiven Ergebnis im Dialog mit der Entwicklungsumgebung – und dem Betreuer.
Gleiches gilt für die Verarbeitung von Text, Abbildungen und Formeln (LaTeX) wie für die Rechnungen mit Zahlen (Matlab®, Julia, R usw.).

Abschluss

Vortrag

Im Mai/Juni werden in Absprache Termine zur Vorstellung des Bearbeitungsstandes festgelegt. Typischerweise sind bis dahin noch nicht alle Projekte abgeschlossen. Der Betreuer wird entsprechend Hinweise geben, was zu tun bleibt, um eine positive Bewertung zu erlangen.

Bericht

Bis Semesterende ist ein ausführlicher Bericht inklusive Reaktion auf die eventuell konstatierten Mängel einzureichen – auch hier in Dialog und Schleife, bis ein akzeptables Ergebnis erreicht wird. Oder eben nicht.
Für ein positives Endergebnis sind Präsentation und Report inklusive aller Quellen (Programmcode, Eingangsdaten, Grafiken und LaTeX-Dateien) als zip-Archiv einzureichen. Nach Test der Lauffähigkeit/Übersetzbarkeit erfolgt dann gegebenefalls der Eintrag *bestanden* in der Ergebnisliste des ZPA.

Kursangebot

Organisation

Vortragstermine

Termine werden im Mai/Juni zugeteilt.

In der Regel werden drei Vorträge pro Veranstaltung am 28.06, 05. bzw. 12.07.2023 gehalten.

Sprachen

Motivation

Stimuli:

- langwierige Arbeiten
- viele Daten
- lange Rechnungen

Beispiele:

- Volkszählung
- Bilder reduzieren
- Reihen auswerten
- Gleichungssysteme

Viele Sprachen – welche wählen?

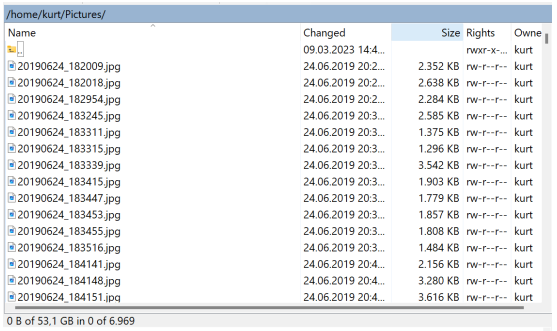
Kriterien:

- Verfügbarkeit
- Kenntnisse
- Vorgaben
- Machbarkeit
- Bequemlichkeit
- Eleganz
- Geschwindigkeit

Shellskript

eine einfache Aufgabe:

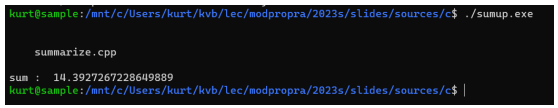
```
mkdir reduced
for i in *.jpg; do
echo $i
convert $i -resize $((5184/4))x$((3456/4))
   reduced/${i%.jpg}.jpeg
done
```



Ein Stück Analysis

diverse Reihen:

```
/*    sumup.cpp
kf2023
*/
#include <stdio.h>

int main ()
{
        printf ("\n\n␣␣␣␣sumup.cpp\n\n");
        long long int n=1e6;
        double S=0.0;
        for (long long int i=1;i<=n;i++)
        S+=1.0/i;
        printf("sum␣:␣%20.16f\n", S);
        return 0;
}
```



Das selbe in Matlab

```
n=1E+9; S=0; for i=1:n; S=S+1/i; end; S
```

geht, aber ... ist das so wirklich Matlab?!

```
s=sum(1./(1:n))
```


Das selbe in julia

```
n=1E+9; S=0; for i=1:n; S=S+1/i; end; S
```

hier muss ein Leerzeichen rein:

```
s=sum(1 ./(1:n))
```


Andere Exponenten

```
n=1E+3; S=0; for i=1:n; S=S+1/i^2; end; S
```

Wer kennt das Ergebnis?

Matlab

Was braucht man?

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

Was möchte man weniger gern?

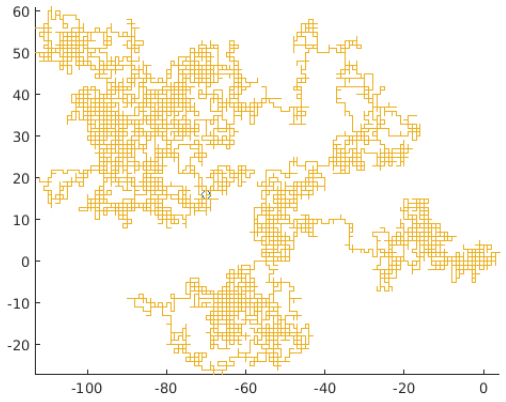
- mystische Präambeln
- Einbinden von Headern, Bibliotheken
- Deklaration von Variablen
- Bereitstellen von Speicher
- kompilieren, linken

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.



Modellierung

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)

Ergebnisse

- in 1D und 2D gibt es eine fast sichere Rückkehr zum Ausgangspunkt
- in höhren 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

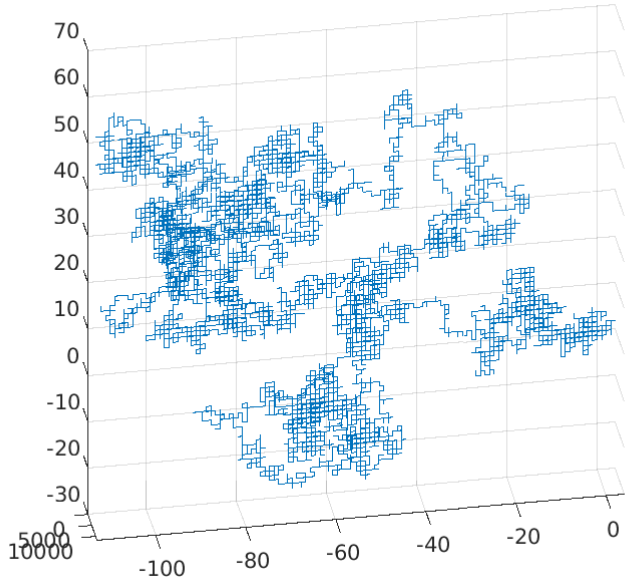
Implementierung (naiv)

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

Implementierung (weniger naiv)

```
% 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
```

Trajektorie



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(:)
```


Spielen mit Matrizen II

```
% 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')
```

Spielen mit Matrizen III

```
% 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)
```


Plotten

```
% 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
```


Nichtlineare Gleichung

```
% 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)
```


Lineares Gleichungssystem

```
% 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)
```

Nichtlineares Gleichungssystem

```
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.

Optimieren

```
% optitest_2.m

% needs and uses obejective_2.m, objejective_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 = [];
```

Curvefitting

```
% muir1980lla - 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')
```


Julia

Motivation

- frei
- offen
- leicht
- schnell
- dynamisch

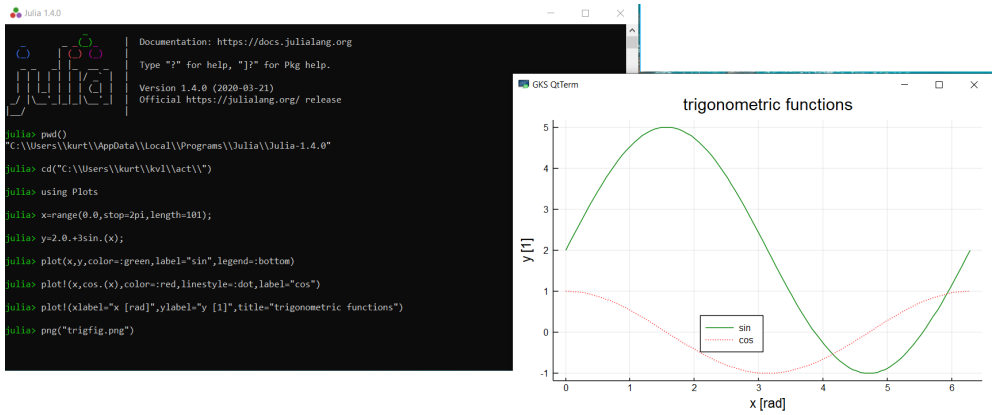
Getting started

- windows, linux, macOS
  download installer/archive from julialang.org
  unter ubuntu: `sudo apt-get julia` gibt UraltVersion!
  aktuelle stabile Version: 1.9.0 (Mai 2023)
- packages
  Pkg.add(name), using name
  Beispiel: `add Plots; using Plots; plot(sin,-pi,pi)`
  inzwischen wird nach `using Plots;`
  automatisch Installation angeboten – sofern nötig

Arbeitsweisen

- REPL-Modus (inklusive help und pkg)
- Skripte mit `include`
- GUI dank VisualStudioCode, vorher: atom (hackable editor)
- Notebooks unter Jupyter mit IJulia

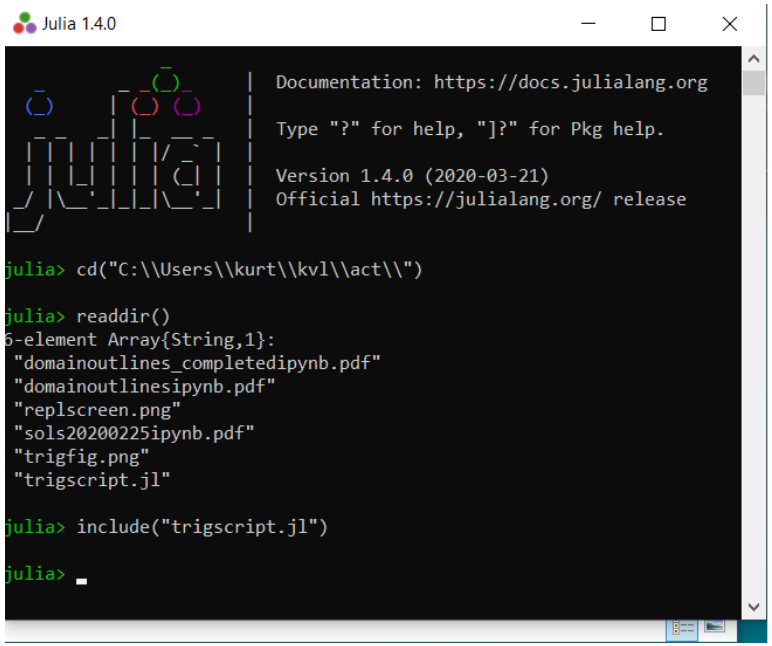
REPL



Remarks zu REPL

- REPL steht für: read-eval-print loop
- Ausführung mit Enter
- Hilfe mit ?
- package-mode mit] (add Plots etc.)
- copy und paste ggfs. mit rechter Mausetaste

Include



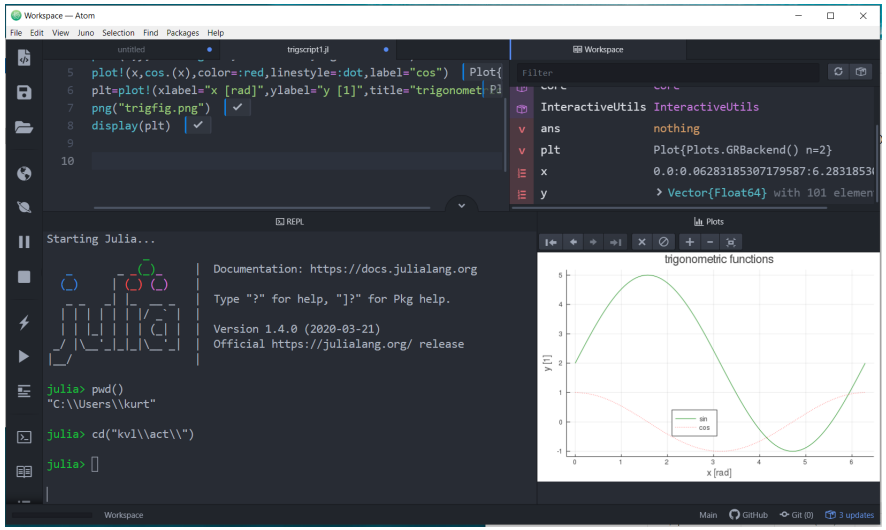
Remarks zu include

- im REPL Skripte mit include ausführen
- Ausgaben, u.a. Plots, werden nur auf Wunsch gezeigt
- Verzeichnis wird nicht automatisch gewechselt

Julia – Example

```
using Plots

x=range(0.0,stop=2pi,length=101);
y=2.0.+3sin.(x);

plot(x,y,color=:green,label="sin",legend=:bottom)
plot!(x,cos.(x),color=:red,linestyle=:dot,label="cos")

plt=plot!(xlabel="x [rad]",ylabel="y [1]",
title="trigonometric functions")

png("trigfig.png")
display(plt)
```


atom



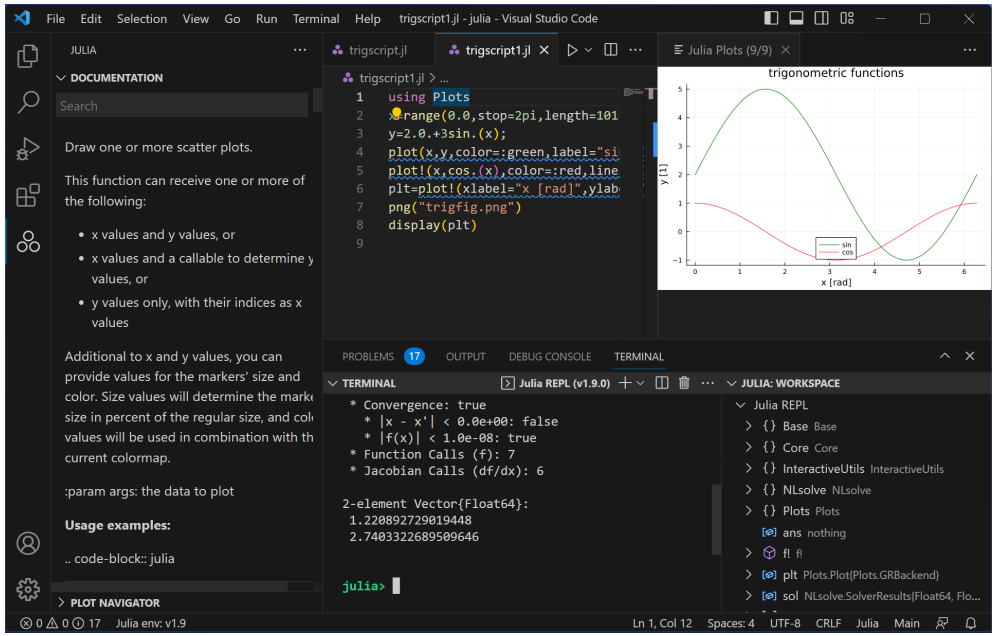
Remarks zu atom

- unter atom wird Ausführung von Skript blockweise (Shift+Enter) oder gesamt (Crtl+Shift+Enter)
- Ausgaben ggfs. hinter jedem Block
- Plots docked in speziellem Unterfenster, zum durchblättern
- Workspace einsehbar
- es gibt keinen clear-Befehl

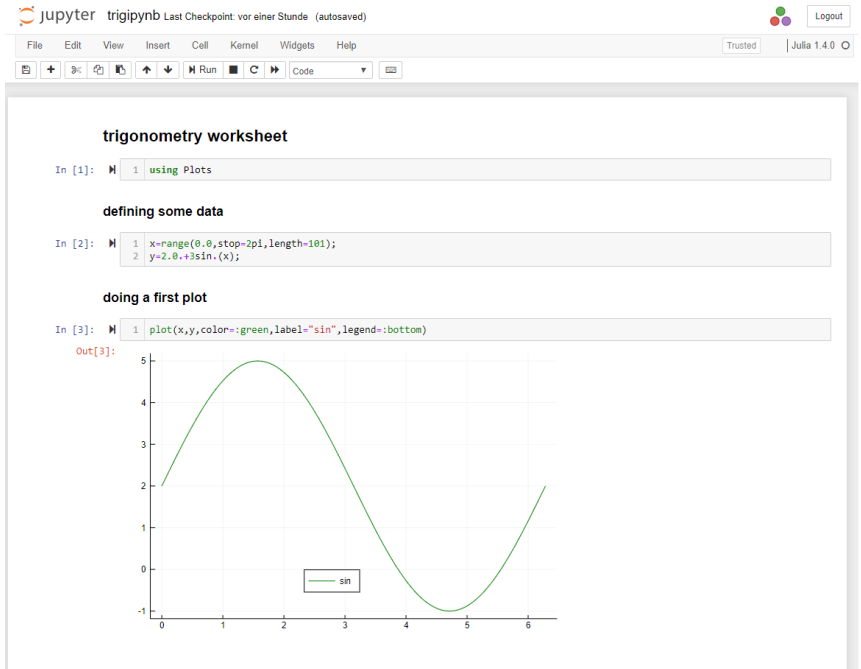
Remarks zu Visual Studio Code

- löst atom ab
- nicht zu verwechseln mit C++Entwicklungsumgebung von MS
- `code.visualstudio.com`
- Plugins für julia und jupyter holen
- Workspace, PlotHistorie, Probleme, Doku ...
- vielfältige Nutzung, u.a. python, LaTeX...

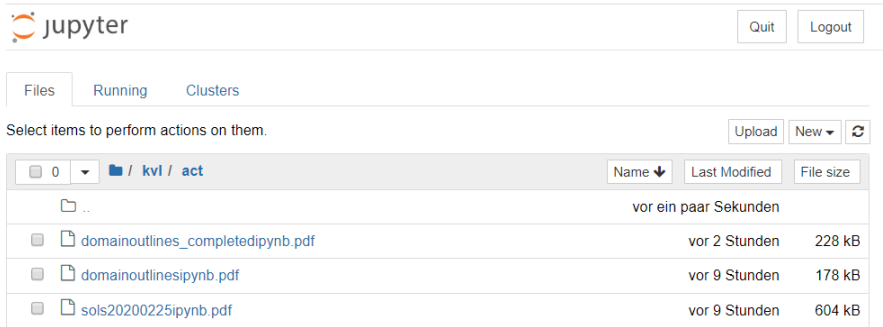
VSC-Screenschot



Jupyter



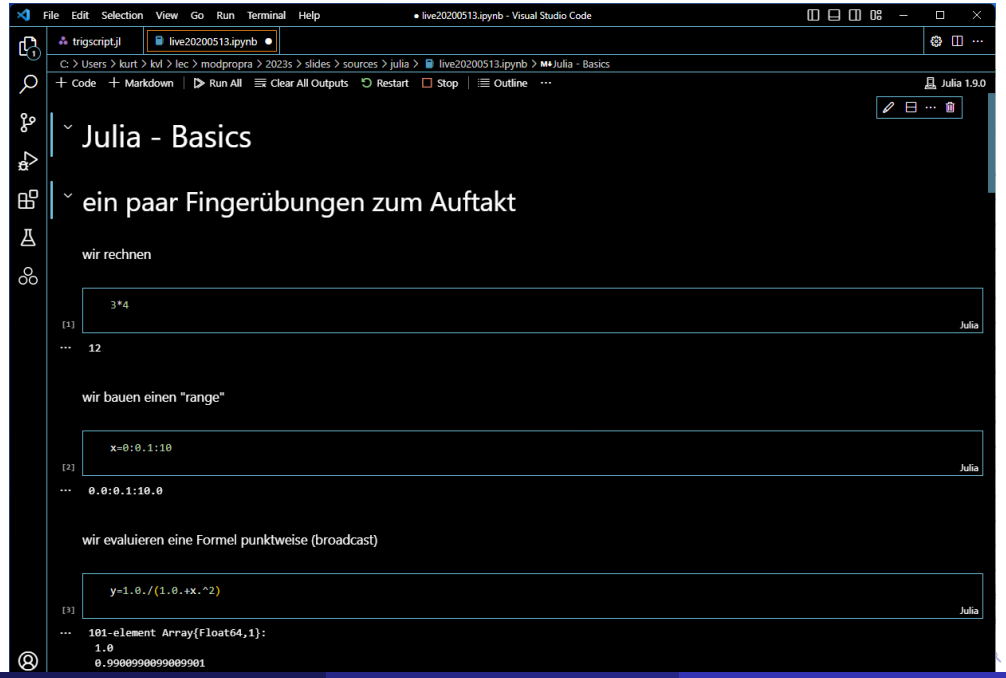
Jupyter – FileManager



Remarks zu IJulia

- Start mit: `using IJulia; notebook()`
- Maple-ähnliche Oberfläche im Standardbrowser
- neues oder existierendes notebook öffnen
- Text (inklusive LaTeX) und Input eingeben
- Output im Worksheet
- Ctrl+P zwecks Ausdruck

Jupyter in VSC



Unterschiede zu Matlab

- Klammern
  `x=[1;2;3], x[2]+=7`
- Ranges sind keine Vektoren
  `y=1:3; y[3]`
  `y[3]=9`
- Vektoren sind keine Matrizen
- Operatoren müssen broadcasted werden
  `x.+y`
- Pakete müssen geladen werden
  `eigvals([1 2; 3 4])` erst nach: `using LinearAlgebra`
- Funktionen sehr user-friendly defined
  `f(x)=sqrt(x)-2x^3`

Erste Fingerübungen

- Plots
  (2d, contour, 3d, Animation)
- Nichtlineare Gleichungen lösen
  (Gleichgewichtszustand)
- LGS Lösen
  (zum Beispiel Kräftegleichgewicht)
- Minimum finden (stationäre Lösung)
- ODE lösen
  (Oszillator)
- Bewegungsgleichungen aufstellen
  (symbolisches Rechnen, Formelmanipulation)

Alternativen bei Graphik

many packages – As you like it

- Plots
  Pkg.add(Plots)
  using Plots
- PyPlot
  best for Python fans
- Gaston
  based on Gnuplot
- Winston
  looks pretty poor
- GR
  ist der Standard bei Plots

Bemerkung: Man kann Frontend wie Backend wechseln, also etwa Plots so nutzen, als hätte man PyPlot geladen.

Alternativen bei ODEs

- DifferentialEquations
  Maß der Dinge
- ODE
  wishlist, still growing project
- OrdinaryDifferentialEqs
- Sundials
  wraps libraries od Lawrence Livermore National Laboratory
  SUite of Nonlinear and DIfferential/Algebraic equation Solvers

Angestaubte Variante

```
using Sundials, Winston
# a well-known right-hand side
function f(t,y,ydot)
   mu = 2.5
   ydot[1] = y[2]
   ydot[2] = mu*(1-y[1]^2)*y[2]-y[1]
ydot
end
# time range
t = [0:.01:10.0;]
# initial state
y0 = [1.0, 3.0]
# call of solver
res = Sundials.cvode(f, y0, t)
# plotting results
plot(res[:,1],res[:,2])
```


Moderne Variante – skalare ODE (Zerfall)

```
using DifferentialEquations, Plots;

# Funktion mit Rueckgabe der Ableitung
f(y,p,t)=p*y;

# Zusammenstellen des Cauchyproblems (AWP)
deqprbl=ODEProblem(f,3.0,(0.0,12.0),-0.33)

# Loesung des Problems
solu=solve(deqprbl)

# Visualisierung
plot(t->solu(t),0,12,legend=false,title="Loesung des AWP")
scatter!(solu.t,solu.u,xlabel="Zeit t",ylabel="Menge y(t)")
```

Bemerkung: Übergabe des Parameters, Markierung des vom Solver gewählten Schritte.

Achtung: Reihenfolge der Argumente bei `ODEProblem` und der rechten Seite

Moderne Variante – System (Pendel)

```
function f!(yp,y,p,t)
    g=p[1]; l=p[2];
    yp[1]=y[2];
    yp[2]=-g*sin(y[1])/l;
    return;
end;

y0=[0.0,6.26]; tspan=(0.0,12.0); p=(9.81,1.0);
pendprob=ODEProblem(f!,y0,tspan,p)

pendsol=solve(pendprob,reltol=1.0e-4)

plot(t->pendsol(t)[1],0,12,label="Winkel",
xlabel="Zeit t")
plot!(t->pendsol(t)[2],0,12,label="Spin",
ylabel="Winkel und Spin")
plot(t->pendsol(t)[1],t->pendsol(t)[2],0,12,leg=false,
title="Phasenportrait",xlabel="Winkel",ylabel="Spin")
```

Bemerkung: Hauptaufwand ist Aufstellen der rechten Seite, zwecks Lösung reicht ein knappes `solve`, der meiste Text dient der grafischen Darstellung der Lösung.

Unterschiede zu Matlab

- packages müssen deklariert werden, zuvor einmalig installiert Grafik, ODE-solver, Optimierung, Statistik . . .
- Kommentare mit `#`
- Matrixelemente mit `x[i]` statt $x(i)$
- Deklaration von Variablen-Typen
- Arrays zuweisen mit `x=copy(y);` sonst zeigen `x` und `y` auf das selbe Object
- Befehl *meshgrid* fehlt (anfangs)
- *sparse* ist anders umgesetzt