Einführung in Matlab und Python - Einheit 7

Grafik-Handle, Validierung, GUI

Jochen Schulz

Georg-August Universität Göttingen

Aufbau

Grafik-Objekte

(engl. `properties`).

- Die Eigenschaften der Objekte sind in sogenannten *handles* gespeichert. Sie liegen dort als `double` (Gleitkommazahl) vor.
- Mit Hilfe dieser Handle können die Eigenschaften existierender Grafik-Objekte geändert werden.

Hierachische Struktur von Grafik-Objekten

Level 1	Root	Wurzel-Objekt. Gesamter Darstellungsbereich. Es wird automatisch erzeugt und es gibt nur eins.
Level 2	Figure	Grafik-Fenster.
Level 3	Axes, Uicontrol, Uimenu, Uicontextmenu	Benutzerdefinierte Grafik-Interfaces. Die Axes Objekte definieren eine Region im Grafikfenster und ordnen ihre Kinder in dieser Region an.
Level 4	Image, Line, Text, Surface,...	Die eigentlichen Grafiken. Sie sind Kinder der Axes Objekte.

Umgang mit dem Grafik-Handle

- Konstruktion einer Grafik

```
x = 0:0.2:2*pi;
plot(x,sin(x))
```

- Abfragen und Bedeutung der Handles aller Objekte

```
h = findobj
```

```
h =
         0
    1.0000
  100.0015
    3.0016
```

```
get(h,'type')
```

```
ans =
    'root'
    'figure'
    'axes'
    'line'
```

Umgang mit dem Grafik-Handle

- Momentane Einstellung des 'Axes'-Objekts

```
set(h(3))
```

```
ActivePositionProperty: [position | {outerposi
ALim
ALimMode: [ {auto} | manual ]
AmbientLightColor
        ...
```

- Eigenschaften des 'Line'-Objekts

```
get(h(4))
```

```
DisplayName: {}
Color: {}
LineStyle: {5x1 cell}
LineWidth: {}
        ...
```

Umgang mit dem Grafik-Handle

- Ändern des Markers:

```
set(h(4),'Marker','s','MarkerSize',4)
```

- Ändern der Einheiten auf der x-Achse

```
set(h(3),'XTick',[0 pi/2 pi 2*pi])
set(h(3),'XtickLabel','0|pi/2|pi|2pi')
```

- gca, gcf und gco sind die Handle für die aktuelle *Axes*, die aktuelle *Figure* und das aktuelle *Objekt* des 4. Levels.

```
set(gcf,'Name','Tolle Abb.')
set(gca,'Fontsize',15)
```

```
l = legend('sin(x)');
set(l,'FontSize',20);
```

Hierachie

- Informationen zu den zugeordneten Kindern

```
a = get(l,'Children'), get(a,'type')
```

- Information zu den Eltern

```
d = get(l,'Parent'), get(d,'type')
```

- Ändern der Kindeigenschaften

```
set(a(3),'Color',[1 0 0])
```

Beispiel

```
%-----------------------------------
% current_figure.m
%
% Aendert die Uebrschriften der Figures so ab,
% das jeweis das aktuelle Fenster die Ueberschrift
% 'aktuell' hat und die anderen 'nicht aktuell'
%-----------------------------------

% Handle aller Figures
a = get(0,'children')

% Beschrifte alle Figures als 'nicht aktuell'
for i = 1:length(a)
    set(a(i),'name','nicht aktuell')
end

% Ueberschreibe den Namen des aktuellen Fensters
set(gcf,'name','aktuell')
```

Umgang mit Objekten

- Löschen von Objekten:

```
delete(handle)
```

- Kopieren von Objekten:

```
copyobj(handle,new_parent)
```

  Hängt das Objekt mit Handle `handle` an andere Eltern `new_parent` an.

- Finden von Objekten mit bestimmten Eigenschaften:

```
findobj(Eigenschaft, Spezifikation)
```

Defaulteinstellungen

- Ansehen der Defaultwerte

```
a = get(0,'Factory')
```

- Ändern der Defaultwerte

```
set(0,'DefaultLineLineWidth',3)
set(0,'DefaultFigureColor','g')
set(0,'DefaultAxesFontSize',20)
```

- Einstellungen gelten immer auch für alle Kinder und Kindes-Kinder.

Defaulteinstellungen II

- Löschen der Defaulteinstellung

```
set(0,'DefaultAxesFontSize','remove')
```

- Die Defaulteinstellungen können in der Datei `startup.m` (Linux/Unix) bzw. `matlabroot\toolbox\local` (Windows) abgelegt werden. Sie werden so beim Start von MATLAB automatisch eingeladen.
- Unter Linux muss die Datei durch den Suchpfad `path` erreichbar sein (oder einfach dort hinlegen: `~/.matlab/startup.m`)

Ein Beispiel

```
% This example shows how the properties of a graphic can
    be modified
% Generate Grid
x = linspace(-2,2,30);
[X,Y] = meshgrid(x,x);

% Function Values
Z = exp(-X.^2-Y.^2).*sin(pi*X.*Y);

% Plot graphic
h = surf(X,Y,Z);

k = get(h,'Parent'); % Handle for the graphics object
Angles = get (k,'View');

for l = 1:360
    set(k,'View',Angles+l);
    pause(1/60)
end
```

Aufbau

GUIs in MATLAB

- Das Graphical User Interface(GUI) ermöglicht das Steuern von Programmen mit Hilfe der grafischen Oberfläche.
- Programme können von Usern ohne MATLAB-Kenntnisse genutzt werden.
- Hilfreich selbst für den Implementierer von numerischen Algorithmen.
- Steuerung der GUIs durch Grafik-Objekte der Typen
  - `uimenu`
  - `uicontextmenu`
  - `uicontrol`
  - `uipanel`

  (Gleiche Hierachie-Ebene wie `axes`-Objekte)
- Grafische Oberfläche zur Erstellung von GUIs:

```
guide
```

Vordefinierte GUIs für Dialogboxen

- `helpdlg`: Hilfebox
- `msgbox`: Eine beliebige Nachricht
- `warndlg`: Anzeige einer Warnung
- `inputdlg`: Abfragen einer Größe
- `questdlg`: Frage

Beispiel:

```
h1 = warndlg('NameFenster','Nachricht')
h2 = errordlg('NameFenster','Nachricht')
ans = questdlg('NameFenster','Nachricht')
ant = inputdlg({'Frage 1','Frage 2','Frage3'},...
  'NameFenster',[1 2 3], {'defAnt1','defAnt2','defAnt3'})
```

Grafik-Objekte und GUIs

- uicontrol: Interaktive grafische Objekte, die Aktionen steuern oder bestimmte Optionen setzen.
- uimenu: Benutzerdefinierte Menüführung in einer `figure` (wird nicht behandelt).
- uicontextmenu: Ein Pop-up Menü, das erscheint, wenn der Benutzer mit der rechten Maustaste ein grafisches Objekt anklickt (wird nicht behandelt).
- uipanel: Container-Objekt zum Gruppieren von anderen Elementen.

Arten von UiControls

`'checkbox'`	Wahl von Zuständen an/aus
`'edit'`	Editierbare Texteingabe
`'popup'`	Auswahl aus Liste bei Anklicken
`'listbox'`	Auswahl aus einer skrollbaren Liste
`'pushbutton'`	Starten eines Events
`'radio'`	Auswahl einer Option
`'toggle'`	Wahl des Zustandes: an/aus
`'slider'`	Auswahl aus Wertebereich
`'text'`	Anzeige von Text in einer Box

Eigenschaften von UiControls

Style	Art des UiControls
CallBack	Durch Anklicken des Users auszuführende Aktion
Position	Lage des Uicontrol-Objekts in der `figure` Eingabe: [links unten Breite Höhe] Einheiten werden durch die `Units`-Eigenschaft gesteuert.
String	Textdarstellung. Bei mehreren Optionen durch Eingabe `string={'opt1';'opt2'}` oder `string='opt1|opt2'`
Units	Einheit zur Bestimmung der Lages des UiControls, Werte: `pixels` (Default), `centimeters`, `normalized` (Werte in $[0, 1]$)
Tag	String zum Auffinden des Objekts durch `findobj`

Erzeugen von UiControls

UiControl-Objekte können durch `uicontrol` erzeugt werden. Aufruf:

```
handle = uicontrol('<eig1>',<wert1>,.., '<eign>',<wertn>)
```

Beispiel:

```
u = uicontrol('style','listbox',...
'string','Option1|Option2|Option3',...
'units','centimeters','position',[0 0 3 3])
```

CallBack

- Sobald ein GUI-Objekt aktiviert wird, führt es MATLAB Code aus. Dies wird *CallBack* genannt.
- *CallBack* ist eine Eigenschaft von UiControl.
- Ein GUI-Objekt wird in der Regel durch einen Mausklick oder ähnliches aktiviert.
- CallBack kann eine Funktion oder ein String, der durch `eval` ausführbar ist, sein.
- Syntax der Callback-Funktion (durch `guide` erstellt)

CallBack - Syntaxvarianten

- `function myfile`
  `set(h, 'CallBack', 'myfile')`
- `function myfile(obj, event)`
  `set(h, 'CallBack', @myfile)`
- `function myfile(obj, event, arg1, arg2)`
  `set(h, 'CallBack', {'myfile', 5, 6})`
- `function myfile(obj, event, arg1, arg2)`
  `set(h, 'CallBack', {@myfile, 5, 6})`
- GUIDE

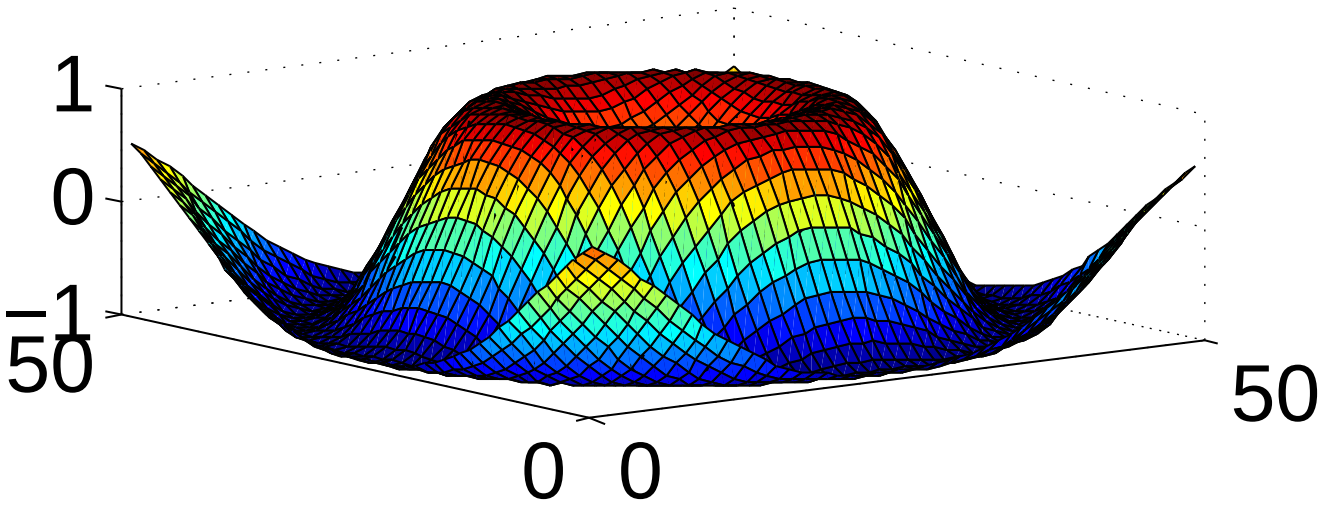
```
function varargout = <objectTag>_Callback(h,ev_data,
    handles, varargin)}}
```

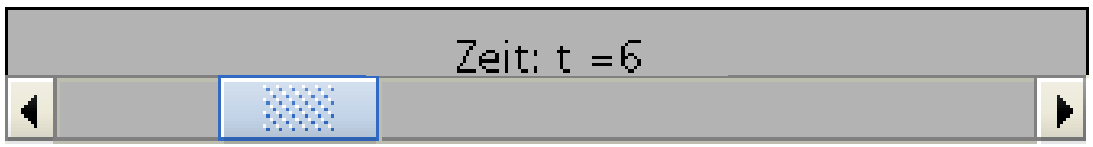
  - `objectTag`: ist der Name, der im Tag des Objects gespeichert wird.
  - `h`: ist der Handle des Objekts, dass die CallBack-Funktion aufruft.
  - `ev_data`: event-data (normalerweise unnötig).
  - `handles`: Struktur aller Objekte in der GUI
  - `varargin`: ist eine Liste von Argumenten, die an die Funktion übergeben wird.

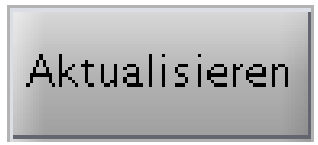
Hinweise

- Callback-Funktionen werden immer im Haupt-Workspace gestartet.
- Mit Hilfe der `figure`-Eigenschaft `UserData` können Daten an das `figure`-Objekt gehängt werden.
- Der User kann grafische Objekte per Hand schließen. Dies kann zu falschen Aufrufen führen.
- Man achte darauf, dass man die richtigen Objekte anspricht. Tipp: Eigenschaft `Tag`.
- `hold (<axes_handle>)`: Verhindern, dass Axen-Einstellungen durch nachfolgende plot-Befehle überschrieben werden.

Beispiel einer GUI







bild_funktion.m

```
function han = bild_funktion()
x = linspace(-1,1,50);
y = linspace(-1,1,50);
t = 0:1:30;
A = erzDaten(x,y,t);
han = erzGUI(A);
end
%-------------------- Grafische Oberflaeche erzeugen
function han = erzGUI(A);
delete(findobj('tag','figGUI'));
fig = figure('name','Beispiel GUI','UserData',A,'tag','
    figGUI');
han.pushbutton = uicontrol(fig,'Parent',fig,'Style',...
  'pushbutton','units','normalized','position',...
  [0.8 0.2 0.15 0.15],'String','Aktualisieren',...
  'Callback','darstGrafik');
han.grafikachse = axes('Position',[0.1 0.5 0.6 0.3],'tag'
    ,'axesGUI');
han.grafik = surf(A(:,:,1));
```

bild_funktion.m

```
han.frame1 = uicontrol(fig,'style','frame', 'units',...
  'normalized','position',[0.1 0.2 0.6 0.1]);
han.slider = uicontrol(fig,'style','slider','sliderstep',
    ...
  [0.2 0.2],'min',0,'max',30, 'units','normalized',...
  'position', [0.1 0.2 0.6 0.05],'tag','slider',...
  'Callback','darstGrafik');
han.text1 = uicontrol(fig,'style','text', 'tag',...
  'text1','units','normalized','position', ...
  [0.3 0.25 0.1 0.03],'String','Zeit t = 0');
han.frame2=uibuttongroup('units','normalized','tag','
    radio',...
  'position',[0.8 0.5 0.15 0.3]);
han.text2=uicontrol(fig,'style','text','parent',
    han.frame2,...
  'units','normalized','position', [0.1 0.6 0.8 0.3],...
  'String','Darstellung');
```

bild_funktion.m

```
han.radio1=uicontrol(fig,'style','radio','parent',
    han.frame2,...
  'tag','r1', 'units','normalized',...
  'position', [0.1 0.45 0.8 0.15],'String','Surf');
han.radio2=uicontrol(fig,'style','radio','parent',
    han.frame2,...
  'tag','r2','units','normalized',...
  'position', [0.1 0.25 0.8 0.15],'String','Mesh');
han.radio3=uicontrol(fig,'style','radio','parent',
    han.frame2,...
  'tag','r3','units','normalized',...
  'position', [0.1 0.05 0.8 0.15],'String','Contour');
end
function A = erzDaten(x,y,t)
[X,Y,T] = meshgrid(x,y,t);
A = cos(pi*T.^0.5.*exp(-X.^2-Y.^2));
end
```

darstGrafik.m

```
function darstGrafik()
% Callback-Funktion fuer die GUI, die durch
    bild_funktion.m erstellt wurde
A = get(findobj('tag','figGUI'),'UserData');
t = round(1+get(findobj('tag','slider'),'Value'));
set(findobj('tag','text1'), 'string', strcat('Zeit: t = '
    ,num2str(t-1)) );
selection = findobj('tag','radio');
switch get(get(selection,'SelectedObject'),'tag')
  case 'r1'
    surf(A(:,:,t));
  case 'r2'
    mesh(A(:,:,t));
  case 'r3'
    contour(A(:,:,t));
  otherwise
    error('Keines oder zuviele entsprech. GUIs geoeffnet'
        );
end
```

Aufbau

Verschiedene Ansätze

- (Enthought) Traits (Widgets: wx)
- wxPython (Widgets: wx)
- PyQT (Widgets: QT)
- tKinter (Widgets: Tk)

Reihenfolge subjektiv nach Präferenz und den Wünschen eines Wissenschaftlers.

Traits

Zusatz-Eigenschaften zu Python-Objekten:

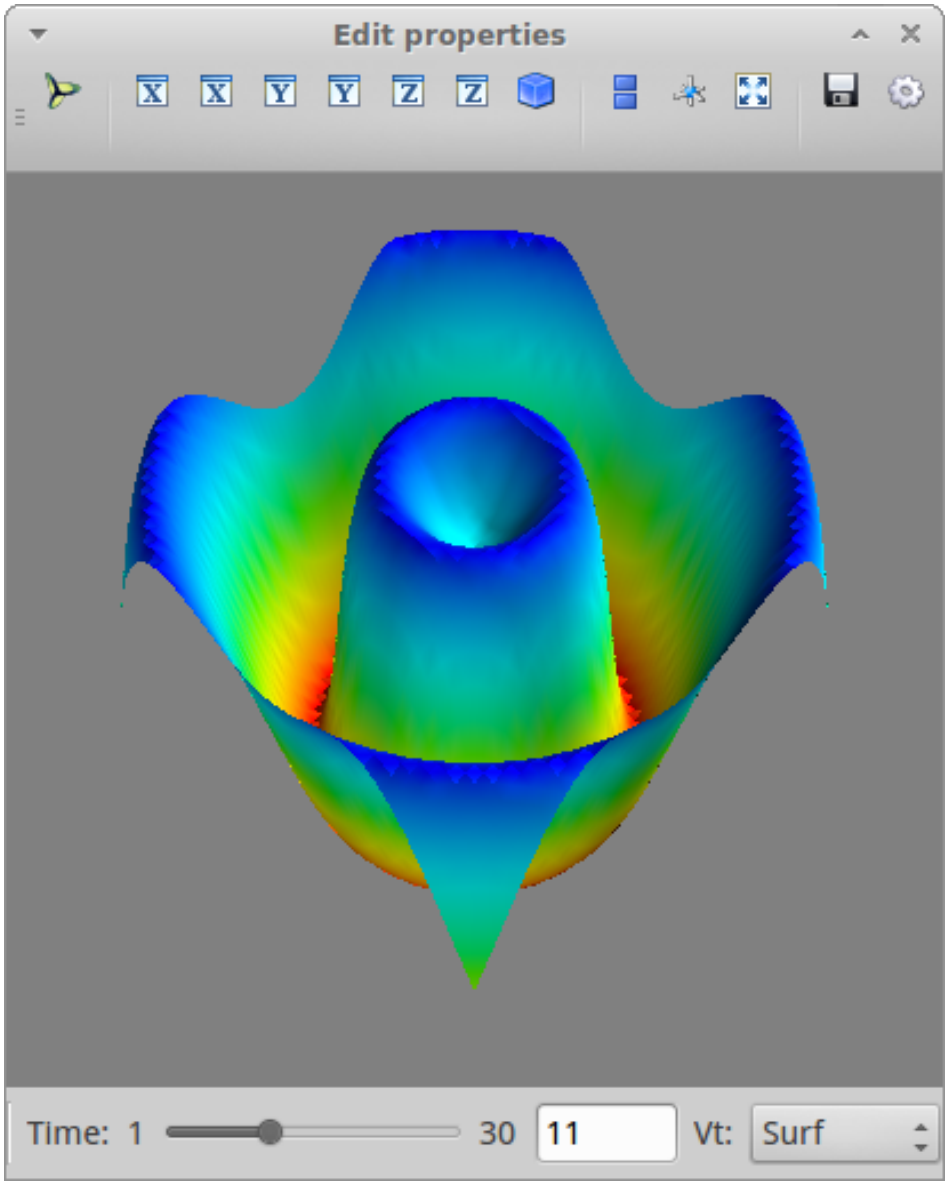
- Initialisierung
- Validierung
- Visualisierung
- Benachrichtigung
- Dokumentation

Vorteile:

- Datenorientiert
- einfache Möglichkeit Daten zu visualisieren

GUI-Beispiel

Wir bauen eine ähnliche GUI wie das Matlab-Beispiel.



Erzeugung

Imports und Klassendefinition (Kochbuch)

```
from traits.api import HasTraits, Range, Enum
class Visualization(HasTraits):
```

Nötige Variablen mittels Traits erzeugen:

```
        time = Range(1, 30, 1) # Range-slider
        vt = Enum ('Surf','Mesh','Contour') #Enumeration
```

Auswahl Basis-Typen

Trait	**Python Type**	**Default**
Bool	Boolean	False
Complex	Complex number	0+0j
Float	Floating point number	0.0
Int	Plain integer	0
Str	String	”

Auswahl erweiterte Typen

Name	type
Array	Array of values
Button	Button of some form and style
Dict	Dictionary ()
Enum	Enum of values
Instance	Klassen-Instanz (Kochbuch :-))
Range	Von-bis in einem Slider

(komplette Liste und weitere Erklärung:

`http://docs.enthought.com/traits/traits_user_manual/defining.html#other-predefined-traits`)

Grafische Repräsentation - TraitsUI

Traits UI und Mayavi einbinden (Kochbuch)

```
from traitsui.api import View, Item, Group
from tvtk.pyface.scene_editor import SceneEditor
from mayavi.tools.mlab_scene_model import MlabSceneModel
from mayavi.core.ui.mayavi_scene import MayaviScene
```

View erzeugen; simple Anordnungen nutzen (Item + Gruppe)

```
scene = Instance(MlabSceneModel, ()) # Kochbuch
view = View(Item('scene', editor=SceneEditor(scene_class=
    MayaviScene), height=250, width=300, show_label=False
    ), #first Item
            Group( #second Item
                  'time','vt'
                  orientation='horizontal',layout='normal
                      '),
            kind='live', title='simple GUI'
            )
```

Views

View ist eine Ansicht der Traits. Es enthält

- Items
- Groups (Gruppen von Items)

```
View (<itemORgroup>[, <itemORgroup>, <settings>])
```

settings

- height: Höhe des Fensters
- width: Breite des Fensters
- title : Titel des Fensters
- kind: Art des Fensters
  - 'live' : normales Fenster
  - 'wizard': Wizard; updated erst bei 'Finish'
  - 'panel': Einbettung in andere Fenster

View: Items

Ein Item stellt ein Trait dar und wird daher direkt mit diesem assoziiert:

```
Item (<traitname>[, <settings>])
```

settings

- height: Höhe des Widgets
- width: Breite des Widgets
- padding: Extraplatz um das Widget herum
- tooltip : generiert einen Tooltip
- show_label: Anzeige eines Labels

View: Groups

Eine Group ist eine visuelle oder logische Einheit. Es kann selbst wieder Items und Groups enthalten.

```
Group(<item>[,<item>,<settings>])
```

settings

- orientation : Orientierung der Unterelemente
- layout: Art der Gruppierung
  - 'normal': Hintereinanderdarstellung.
  - 'flow': Hintereinander, aber kann wrappen.
  - ' split ': Split-Balken zwischen den Elementen.
  - 'tabbed': Elemente in Tabs aufgeteilt.

Callbacks

Initialisierung der Daten und der Klasse. Initiale Grafik wird in der letzten Zeile erzeugt.

```
x,y,t = np.mgrid[-1:1:(2.0/50),-1:1:(2.0/50),1:31]
Z = cos(pi*t**0.5*exp(-x**2-y**2))
def __init__(self):
     HasTraits.__init__(self)
     self.plot = self.scene.mlab.surf(self.x[:,:,0],
        self.y[:,:,0],self.Z[:,:,0])
```

Änderung des Kamera-viewpoints, beim ersten Aktivieren (z.T. Kochbuch)

```
@on_trait_change('scene.activated')
def create_plot(self):
     self.scene.mlab.view(45,210)
```

Mit @on_trait_change('variable' wird eine Funktion dann ausgeführt, wenn die angegebene Variable sich geändert hat.

Callbacks II

Funktion zum Plot-update, wenn sich eine der Variablen ändert (z.T. Kochbuch)

```
    @on_trait_change('time,vt')
    def update_plot(self):
        self.plot.remove() # remove last image
        if self.vt == 'Surf':
            self.plot = self.scene.mlab.surf(self.x
                [:,:,0],self.y[:,:,0],self.Z[:,:,self.
                time-1])
        elif self.vt == 'Mesh':
            self.plot = self.scene.mlab.surf(self.x
                [:,:,0],self.y[:,:,0],self.Z[:,:,self.
                time-1],representation='wireframe')
        elif self.vt == 'Contour':
            self.plot = self.scene.mlab.contour_surf(self
                .x[:,:,0],self.y[:,:,0],self.Z[:,:,self.
                time-1],contours=15)
        else:
            print "Plot-Auswahl fehlgeschlagen"
```

Literature

- NumPy, SciPy SciPy developers (http://scipy.org/),
- Writing a graphical for scientific programming using TraitsUI (http://code.enthought.com/projects/traits/docs/html/tutorials/traits_ui_scientific_app.html)
- Building applications using mayavi (http://docs.enthought.com/mayavi/mayavi/building_applications.html)
- Traits user manual (http://docs.enthought.com/traits/traits_user_manual/index.html)
- Traits API reference (http://code.enthought.com/projects/files/ets_api/enthought.html)
- Traits: building interactive dialogs (http://scipy-lectures.github.io/packages/traits/index.html)

Aufbau

Motivation

Validierung (und Auswertung)

- Validierung von entscheidender Bedeutung für die Numerik
- Als Werkzeug dient als wichtiger Teil die Visualisierung

Visualisierung (und Auswertung)

- Nutze die Intuition und alle Sinne!
- Verbindung zur realen Welt

Strategien

- Aufteilung des Problems in möglichst kleine einzelne Module (Funktionen/Klassen)
- Analytische Lösungen numerisch verifizieren
- Initial möglichst anschauliche/einfache Probleme berechnen
- Benchmarks nutzen: klar definierte Referenzprobleme nutzen
- Konsistents-Tests (z.B.):
  - Lösung einer Gleichung Rückwärts nutzen.
  - Grössen, Strukturen, Orientierungen und Formate von allen Daten/Matrize/Vektoren prüfen.
  - Raumzugehörigkeiten prüfen (Falls möglich).
- Kondition von Matrizen beachten (Vorkonditionierung, Vermeidung)
- Laufzeit-Ausgaben
- Dokumentation
- Visualisierung nutzen (Intuition!)

Subjektive Auswahl von Software/Tools

- python-like (matplotlib) (script)
- VTK, Paraview (gui)
- yt (script)
- Matlab (gui)
- gnuplot (script)
- Visit (gui)