Einführung in Sage - Einheit 9

Strings, interaktive Grafiken, Wärmeleitungsgleichung, Sage-Code, geogebra ?, interface zu anderen

Jochen Schulz

Georg-August Universität Göttingen

Aufbau

Klausur

- Zeit: 01.03.2010 von 10:00 - 12:00
- Ort: HS1 (A bis J) und AudiMax (K bis Z)
- Hilfsmittlel: Papier, Schreibgerät(e) und Unterlagen in Papierform
- Studenten-Ausweis mitbringen

Aufbau

Strings

- Zeichenketten (engl. strings) sind eine geordnete Aneinanderreihung von Zeichen. Zeichen sind z.B. Buchstaben, Ziffern, Sonderzeichen,...
- Mit ihnen kann man in Sage Texte gestalten. Sie sind wichtig für die Ausgabe der Ergebnisse.
- Sie haben den Datentyp `str`.
- Sie werden innerhalb von Hochkommas oder Anführungszeichen angegeben.

Beispiele für Strings

```
text1 = 'Dies ist ein String.'; text1
```

```
'Dies ist ein String.'
```

```
text2 = "Dies ist noch ein String."; text2
```

```
'Dies ist noch ein String.'
```

```
type(text1)
```

```
<type 'str'>
```

Zugriff

- Mit dem Indexoperator [ ] können einzelne Zeichen einer Zeichenkette extrahiert werden.

```
text1[0], text1[3], text1[4]
```

```
('D', 's', ' ')
```

- Ersetzungen innerhalb des Strings:

```
text1.replace('Dies','Das')
```

```
'Das ist ein String.'
```

Operationen für Strings I

- Zusammenhängen von Strings

```
A='Letzte '; B='Vorlesung'; A+B
```

```
'Letzte Vorlesung'
```

- `len` gibt die Anzahl der Zeichen in einer Zeichenkette an.

```
a=len(A+B); a
```

```
16
```

Operationen für Strings II

- neue Zugriffsmöglichkeit

```
(A+B)[-1:]
```

```
'g'
```

- Beliebige Sage-Objekte können durch `str()` in einen String verwandelt werden.

```
str(x^2+2), str([1,2,3])
```

```
('x^2 + 2', '[1, 2, 3]')
```

Operationen für Strings III

- Aufspalten von Texten

```
text = 'Dies ist ein Satz, und ein Nebensatz'
text.split()
```

```
['Dies', 'ist', 'ein', 'Satz,', 'und', 'ein', '
    Nebensatz']
```

```
text.split('i')
```

```
['D', 'es ', 'st e', 'n Satz, und e', 'n Nebensatz']
```

Vertiefung print und Formate

```
print 'Text %<format> und %<format> ... ' % (x,y,...)
```

`<format>`:

```
%[<flag>][<minwidth>][.<precision>]converter
```

- `flag`: 0 für das Auffüllen mit Nullen
- `minwidth`: Minimale Breite der Darstellung
- `precision`: Genauigkeit (Nachkommastellen)
- `converter`:
  - `'i'` ganze Zahl mit Vorzeichen
  - `'e'` Gleitkommazahl mit exponentialformat (kleingeschrieben).
  - `'f'` Gleitkommazahl im Dezimalformat.
  - `'g'` Gleitkommazahl. Benutzt kleingeschriebene Exponentialform wenn der Exponent kleiner als -4 oder der Genauigkeit ist, ansonsten Dezimalformat.

print - Beispiele

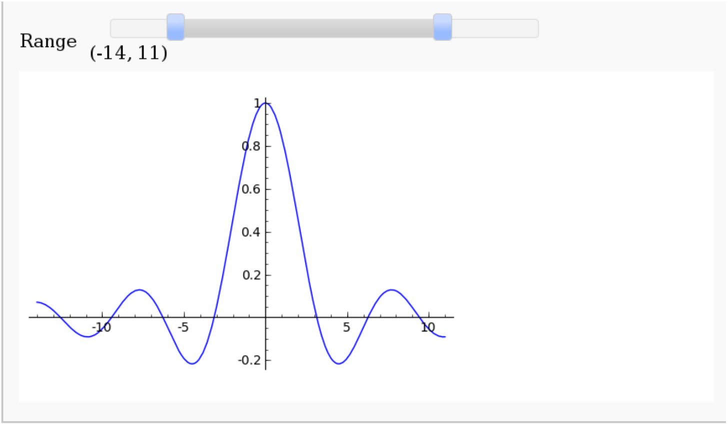
```
print '%5s | %7s' % ('Index','Wert')
for k in srange(1,25,9.55):
    print '%5i | %07.3f' % (k,k^2)
```

```
Index |    Wert
    1 | 001.000
   10 | 111.303
   20 | 404.010
```

Aufbau

interact

```
@interact
def _(b = range_slider(-20, 20, 1, default=(-19,3), label
    ='Range')):
     plot(sin(x)/x, b[0], b[1]).show(xmin=b[0],xmax=b[1])
```



interact

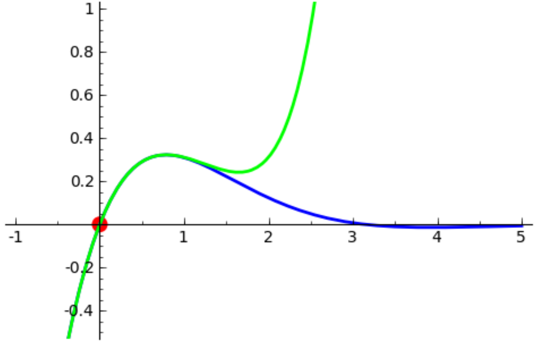
- ```
  u=slider(vmin, vmax=, step_size=1, default=, label=)
  ```
  Regler mit entsprechenden Werten
- ```
  u=range_slider(vmin, vmax=, step_size=1, default=,
      label=)
  ```
  Regler eines Intervalles
  ```
  u=checkbox(default=True, label=)
  ```
  Eine Ankreuzfeld
- ```
  u=selector(values, label=, nrows=, ncols=, buttons=
      False)
  ```
  Ein Aufklappmenü oder Knöpfe (Knöpfe wenn nrows, ncols, oder buttons gesetzt ist, sonst Aufklappmenü)
- ```
  u=text_control(value='')
  ```
  Ein Textblock

interact - Taylor

order 1

$$f(x) = e^{(-x)} \sin(x)$$

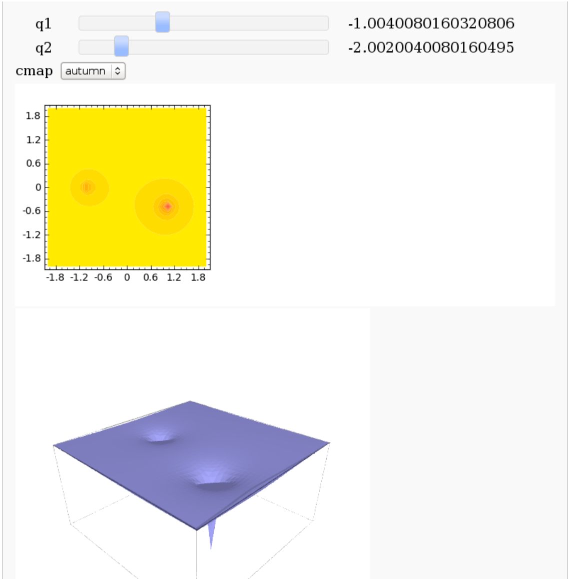
$$\hat{f}(x;0) = \frac{1}{90}x^6 - \frac{1}{30}x^5 + \frac{1}{3}x^3 - x^2 + x + O(x^7)$$



interact - Taylor

```
var('x')
x0  = 0
f   = sin(x)*e^(-x)
p   = plot(f,-1,5, thickness=2)
dot = point((x0,f(x=x0)),pointsize=80,rgbcolor=(1,0,0))
@interact
def tayl(order=(1..12)):
    ft = f.taylor(x,x0,order)
    pt = plot(ft,-1, 5, color='green', thickness=2)
    html('f(x)\;=\;%s'%latex(f))
    html('\hat{f}(x;%s)\;=\;%s+\mathcal{O}(x^{%s})'%(x0,
        latex(ft),order+1))
    show(dot + p + pt, ymin = -.5, ymax = 1)
```

interact - Kontur und 3D-Plot von einer Abstandsfunktion



interact - Kontur und 3D-Plot von einer Abstandsfunktion

```
@interact
def _(q1=(-1,(-3,3)), q2=(-2,(-3,3)),
      cmap=['autumn', 'bone', 'cool', 'copper', 'gray', '
          hot', 'hsv',
            'jet', 'pink', 'prism', 'spring', 'summer', '
                winter']):
    x,y = var('x,y')
    f = q1/sqrt((x+1)^2 + y^2) + q2/sqrt((x-1)^2+(y+0.5)
        ^2)
    C = contour_plot(f, (x,-2,2), (y,-2,2), plot_points
        =30, contours=15, cmap=cmap)
    show(C, figsize=3, aspect_ratio=1)
    show(plot3d(f, (x,-2,2), (y,-2,2)), figsize=5,
        viewer='tachyon')
```

Aufbau

Größter gemeinsamer Teiler (ggT)

Berechnung des ggT von natürlichen Zahlen a und b mit Hilfe des euklidischen Algorithmus.

Idee: Es gilt:

1. $ggT(a, b) = ggT(a, b - a)$ für $a < b$.
2. $ggT(a, b) = ggT(b, a)$.
3. $ggT(a, a) = a$.

Algorithmus:
Wiederhole, bis $a = b$

- Ist $a > b$, so $a = a - b$.
- Ist $a < b$, so $b = b - a$

ggT - Implementierung

```
def ggT(a,b):
    """Bestimme den ggT von a und b"""
    while a<>b:
        if a>b:
            a = a-b
        else:
            b = b-a
    return a
ggT(6,9)
```

3

Berechnung von Primzahlzwillingen

```
T = []; anz = 0
for i in [2..100]:
    if (is_prime(i) and is_prime(i+2)):
        anz += 1
        T.append([i,i+2])
print('Anzahl = %s' % anz);T
```

```
Anzahl = 8
[[3, 5], [5, 7], [11, 13], [17, 19], [29, 31], [41, 43],
    [59, 61], [71,
73]]
```

Betrag (noch sehr unschoen)

```
def betrag(a):
    if type(a) == Integer or type(a) == Rational or
        is_RealNumber(a):
         if a>0:
             y = a
         else:
             y = -a
    elif is_ComplexNumber(a):
         y = sqrt(real(a)^2+imag(a)^2)
    else:
         print("Falscher Eingabetyp");
    return y
betrag(2+I*4)
```

Gültigkeit von Variablen

- Mit der Gültigkeit von Variablen ist die Bestandsdauer von Variablen bzw. der Werten dieser Variablen gemeint.
- Beim interaktiven Gebrauch von Sage sind alle Variablen global, d.h. die den Variablen zugewiesenen Werte bleiben für die gesamte Laufzeit vom jeweiligen worksheet erhalten bis sie geändert werden. Man kann auf die Variablen jederzeit zugreifen und die Werte der Variablen ändern.
- Daneben gibt es aber auch lokale Variablen, die nur innerhalb einer Prozedur/Funktion gültig sind. Nach Beenden der Prozedur werden diese Variablen wieder gelöscht.

Mandelbrot-Menge

Die Mandelbrot-Menge ist die Menge von Punkten $c \in \mathbb{C}$ bei denen die Folge $(z_n)_n$, die durch

$$z_0 := c, \qquad z_{n+1} = z_n^2 + c, \quad n \in \mathbb{N}$$

definiert ist, beschränkt ist.

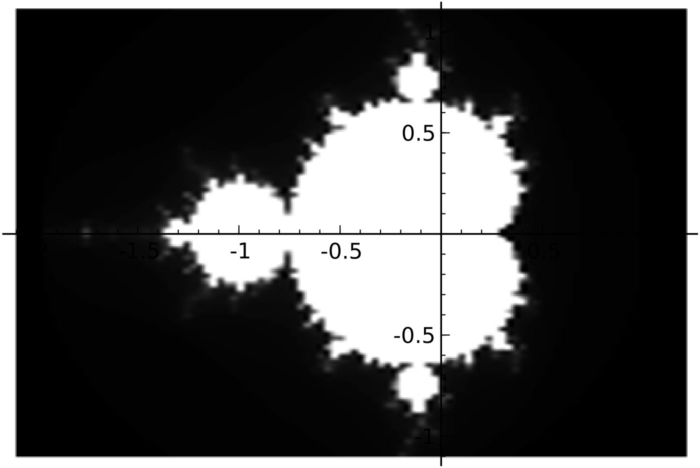
Programm - Mandelbrot

```
def mandel(x,y):
    c = (x + I*y).n()
    z = c
    it = 0
    max_it = 150
    while abs(z)<2 and it<max_it:
        z = z^2 + c
        it += 1
    return float(it/max_it)
```

Die Funktion `mandel` gibt zu $x + iy$ die relative Anzahl der Iterationsschritte zurück.

Plot - Mandelbrot

```
density_plot(mandel, (-2.1,1.2), (-1.1,1.1), plot_points
    =100)
```



Programmierregeln ??

- Programme vollständig kommentieren. Das heißt zum einen das eine Kommentarzeile zu Beginn steht, was das Programm macht und wieviele und welche Eingabeparameter es erhalten darf und was die Prozedur zurückgibt. Zusätzlich sollten auch alle wesentlichen Operationen kommentiert werden.

Letztes Beispiel I

```
Gadisch:=proc(x,basis)
  /*---------------------------------------
   Berechnung der  Darstellung
   einer natuerlichen Zahl x zur Basis b
   Rueckgabe des Ergebnis als Liste!
  ----------------------------------------*/
  local T,T_r,i;  /* lokale Variablen*/
  begin /* Beginn lokale Prozedur */
  /* Abfangen der Eingabe */
    if not testtype(x,Type::PosInt)
       then return(procname(args()));
    end_if;
    if (not testtype(basis,Type::PosInt)) or basis=1
       then return(procname(args()));
    end_if;
```

Letztes Beispiel II

```
  T:=[];  /* leere Liste */
  /* Beginn Schleife */
  while x>0 do
    T:=[x mod basis].T;
    print(Unquoted,expr2text(x)." : "
      .expr2text(basis)." = "
      .expr2text(x div basis)." Rest "
      .expr2text(x mod basis));
    x:=(x div basis);
  end_while;
  /* Rueckgabe der Liste */
  return(T);
end_proc:
```

Allerletztes Beispiel: Kochsche Kurven I

- Seien y_1, y_2 zwei Punkte im $\mathbb{R}^2$.
- Betrachte die Strecke mit Endpunkten y_1 und y_2.
- Ersetze diese Strecke durch 4 Strecken $\overline{y_1 z_1}$, $\overline{z_1 z_2}$, $\overline{z_2 z_3}$, $\overline{z_3 y_2}$ mit Endpunkten

$$
\begin{aligned}
z_1 &= \tfrac{2}{3} y_1 + \tfrac{1}{3} y_2 \\
z_2 &= \tfrac{\sqrt{3}}{6} \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} (y_1 - y_2) + \tfrac{1}{2}(y_1 + y_2) \\
z_3 &= \tfrac{1}{3} y_1 + \tfrac{2}{3} y_2
\end{aligned}
$$

- Dieses Prozedere wird nun für jede einzelne Teilstrecke wiederholt.

Allerletztes Beispiel II

```
koch:=proc(y1,y2,lev)
  local z1,z2,z3;
  begin
    if (lev = 0)
      then Listelinien:=append(Listelinien,
        plot::Line2d([y1[1],y1[2]],[y2[1],y2[2]]));
    else
      /* Definieren der neuen Punkte */
      z1 := 2/3 * y1 + 1/3 * y2;
      z3 := 1/3 * y1 + 2/3 * y2;
      z2 := sqrt(3)/6*matrix([[0, 1],[ -1, 0]])*
              (y1-y2) + 1/2 * ( y1 + y2);
      /* Definieren der 4 Strecken */
      koch(y1, z1, lev-1);
      koch(z1, z2, lev-1);
      koch(z2, z3, lev-1);
      koch(z3, y2, lev-1);
    end_if;
  end_proc:
```

Allerletztes Beispiel III

```
/* Einfacher Fall einer Linie */
plotKoch1:=proc(lev)
           begin
             Listelinien := [];
             y1 := matrix([0,0]);
             y2 := matrix([1,0]);
             koch(y1,y2,lev);
             plot(Listelinien, Axes = None);
           end_proc:
```