Symbolisch und Numerisch Rechnen mit Python

Carsten Knoll

TU Dresden, Institut für Regelungs- und Steuerungstheorie

Interdisziplinärer Python "'Sommerkurs"'

20. Juni 2011

Vorbemerkungen

- Ziel: Überblick, für welche Art von Problemen Werkzeuge vorhanden sind
- keinesfalls vollständig
- Aufbau:

  numpy/scipy
  - Wiederholung numpy arrays und ipython
  - „Broadcasting“
  - Lineare Algebra, Gleichungssysteme
  - Interpolation,
  - Statistik
  - DGLn-Löser,

  sympy
  - Grundlegendes
  - Rechnen
  - Wichtige Funktionen / Datentypen
  - Substituieren
  - Formeln numerisch auswerten

  Übungsaufgaben

Vorbemerkungen

- Ziel: Überblick, für welche Art von Problemen Werkzeuge vorhanden sind
- keinesfalls vollständig
- Aufbau:

  numpy/scipy
  - Wiederholung numpy arrays und ipython
  - „Broadcasting“
  - Lineare Algebra, Gleichungssysteme
  - Interpolation,
  - Statistik
  - DGLn-Löser,

  sympy
  - Grundlegendes
  - Rechnen
  - Wichtige Funktionen / Datentypen
  - Substituieren
  - Formeln numerisch auswerten

  Übungsaufgaben

Gliederung

Wiederholung numpy arrays

- Wichtigster (Container-)Datentypen für numerische Daten
- Beliebig viele Dimensionen (üblich: 1,2,3-dim.)
- Elementweise Operationen (auch Vergleiche!)
- `min, max, mean, median, abs, clip, sin, cos, ...`
  Schnelle Berechnungen (C- oder Fortran-Code im Hintergrund)
- Slicing (Teil-arrays herauslösen)

Wiederholung ipython *beispiel01.py*

- Ausführlich: Vortrag 02 „Interaktive Lernmethoden“
- `ipython`: Verbesserte Python Shell (interaktive Eingabeaufforderung)
- Warum bessonders nützlich für *Berechnungen*?
  - Lösungsweg oft nicht vorher im Detail klar
    Interaktivität $\Longleftrightarrow$ schrittweises Vorgehen
  - Schneller Zugriff auf docstrings
  - Unicode printing (grieschische Buchstaben, Indizes)
  - Bedienung ohne Maus
  - Intelligente History
  - Ausgaben Zwischenspeicher (_, __, ___)
  - ...

Wiederholung ipython *beispiel01.py*

- Ausführlich: Vortrag 02 „Interaktive Lernmethoden“
- `ipython`: Verbesserte Python Shell (interaktive Eingabeaufforderung)
- Warum bessonders nützlich für *Berechnungen*?
  - Lösungsweg oft nicht vorher im Detail klar
    Interaktivität $\Longleftrightarrow$ schrittweises Vorgehen
  - Schneller Zugriff auf docstrings
  - Unicode printing (grieschische Buchstaben, Indizes)
  - Bedienung ohne Maus
  - Intelligente History
  - Ausgaben Zwischenspeicher (_, __, ___)
  - ...
  - **Lässt sich einfach in eigene Scripte einbinden**
    $\rightarrow$ beispiel01.py

„Broadcasting“ *beispiel01.py*

- numpy’s Umgang mit arrays mit unterschiedlichen Abmessungen
- Trivialbeispiel: 2d-array + Skalar → Skalar wird aufgeblasen
- anderes Beispiel: 2d-array * 1d-array
- Regel: Die Größe entlang der letzten *Achsen* beider Operanden müssen übereinstimmen oder eine von beiden muss eins sein.
- Beispiel:

Image	(3d array):	256	256	3	A	(4d array):	8	1	6	1
Scale	(1d array):			3	B	(3d array):		7	1	5
Result	(3d array):	256	256	3	Result	(4d array):	8	7	6	5

- Führt manchmal zu Verwirrung/Problemen:

```
ValueError: shape mismatch: objects cannot be
broadcast to a single shape
```

→ Am besten (interaktiv) ausprobieren

„Broadcasting“ *beispiel01.py*

- numpy’s Umgang mit arrays mit unterschiedlichen Abmessungen
- Trivialbeispiel: 2d-array + Skalar → Skalar wird aufgeblasen
- anderes Beispiel: 2d-array * 1d-array
- Regel: Die Größe entlang der letzten *Achsen* beider Operanden müssen übereinstimmen oder eine von beiden muss eins sein.
- Beispiel:

Image	(3d array):	256	256	3	A	(4d array):	8	1	6	1
Scale	(1d array):			3	B	(3d array):		7	1	5
Result	(3d array):	256	256	3	Result	(4d array):	8	7	6	5

- Führt manchmal zu Verwirrung/Problemen:

```
ValueError: shape mismatch: objects cannot be
broadcast to a single shape
```

→ Am besten (interaktiv) ausprobieren

„Broadcasting“ *beispiel01.py*

- numpy’s Umgang mit arrays mit unterschiedlichen Abmessungen
- Trivialbeispiel: 2d-array + Skalar → Skalar wird aufgeblasen
- anderes Beispiel: 2d-array * 1d-array
- Regel: Die Größe entlang der letzten *Achsen* beider Operanden müssen übereinstimmen oder eine von beiden muss eins sein.
- Beispiel:

Image	(3d array):	256	256	3	A	(4d array):	8	1	6	1
Scale	(1d array):			3	B	(3d array):		7	1	5
Result	(3d array):	256	256	3	Result	(4d array):	8	7	6	5

- Führt manchmal zu Verwirrung/Problemen:

```
ValueError: shape mismatch: objects cannot be
broadcast to a single shape
```

→ Am besten (interaktiv) ausprobieren

Lineare Algebra, Gleichungssysteme

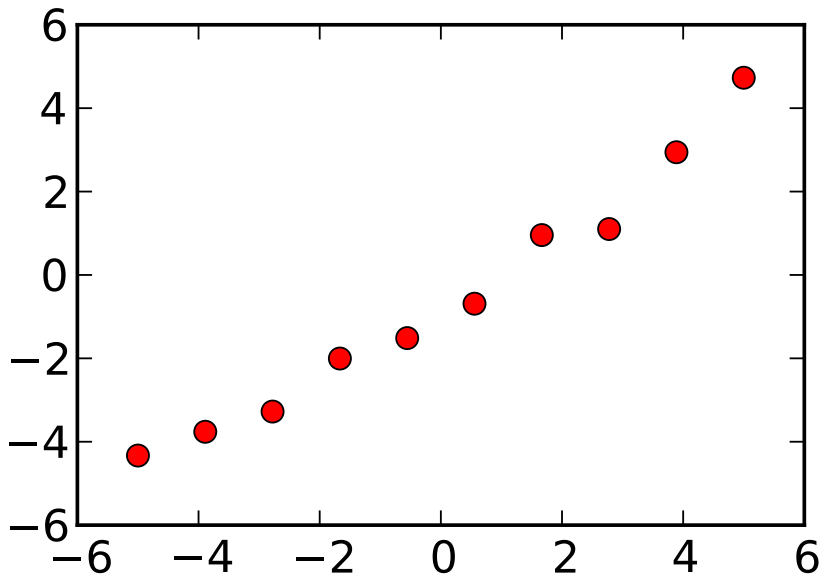
- „The only thing, mathematicians really understand is linear algebra“
- Matrixmultiplikation: `numpy.dot`
- Paket `numpy.linalg`:
  `det, inv, solve, leastsq, eig, svd, ...`
- Im Hintergrund schneller Fortran-Code (lapack)

```
# Solve ``3 * x0 + x1 = 9`` and ``x0 + 2 * x1 = 8``:

A = np.array([[3,1], [1,2]])
b = np.array([9,8])
x = np.linalg.solve(A, b) # -> array([ 2.,  3.])

# Check that the solution is correct:

(np.dot(A, x) - b) # -> array([ 0.,  0.])
```

Regression, Interpolation

beispiel03.py

- Kurven auf Daten „fitten“
- Linear oder Polynome h. O.



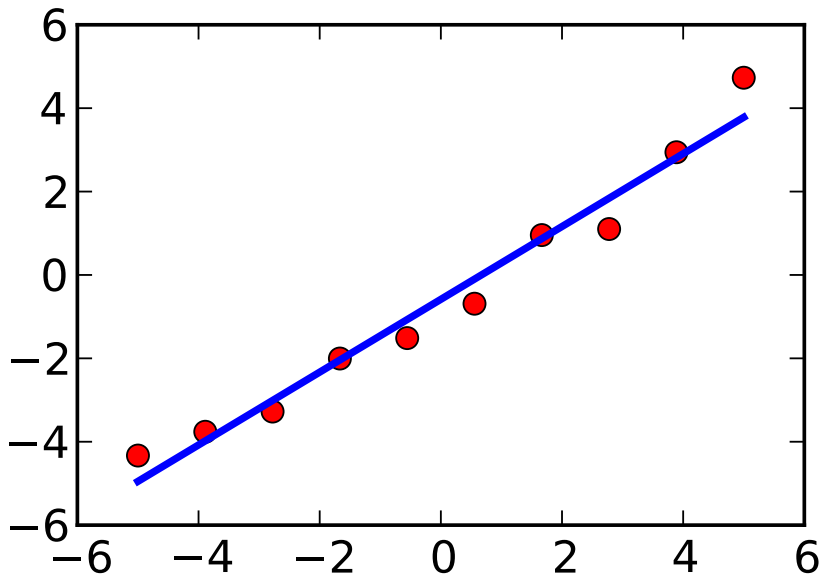
- Interpolation
- Stückweise polynomial (spline)
- (beliebiger Ordnung)

- Mischform:
- „geglätter Spline“

Regression, Interpolation

beispiel03.py

- Kurven auf Daten „fitten“
- Linear oder Polynome h. O.



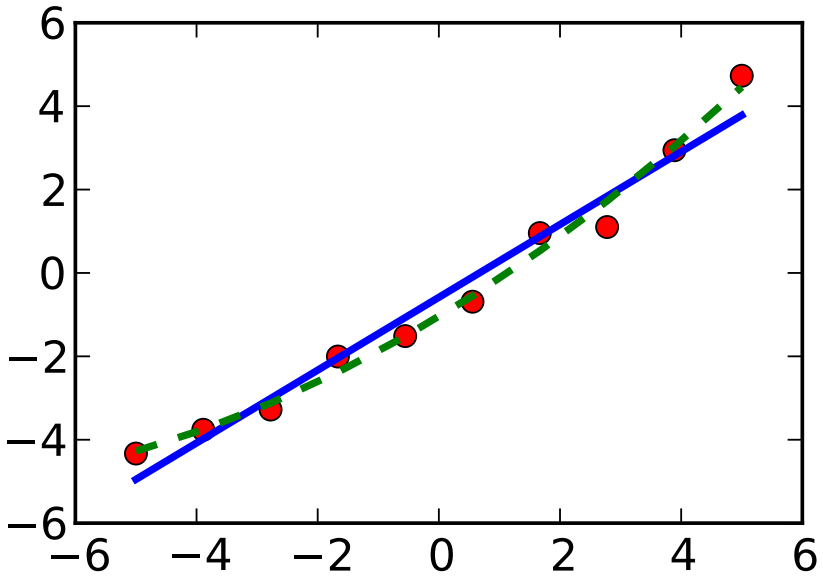
- Interpolation
- Stückweise polynomial (spline)
- (beliebiger Ordnung)

- Mischform:
- „geglätter Spline“

Regression, Interpolation

beispiel03.py

- Kurven auf Daten „fitten“
- Linear oder Polynome h. O.



- Interpolation
- Stückweise polynomial (spline)
- (beliebiger Ordnung)

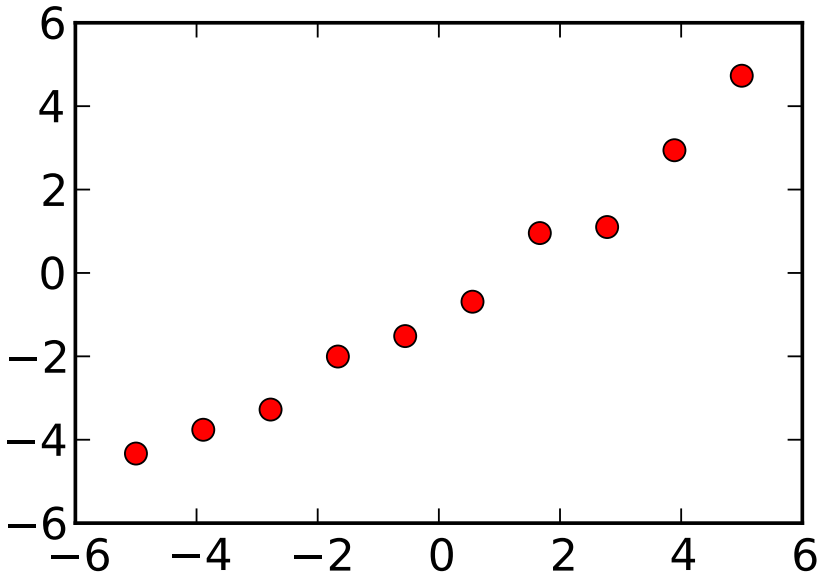
- Mischform:
- „geglätter Spline“

Regression, Interpolation *beispiel03.py*

- Kurven auf Daten „fitten“
- Linear oder Polynome h. O.

- Interpolation
- Stückweise polynomial (spline)
- (beliebiger Ordnung)

- Mischform:
- „geglätter Spline“



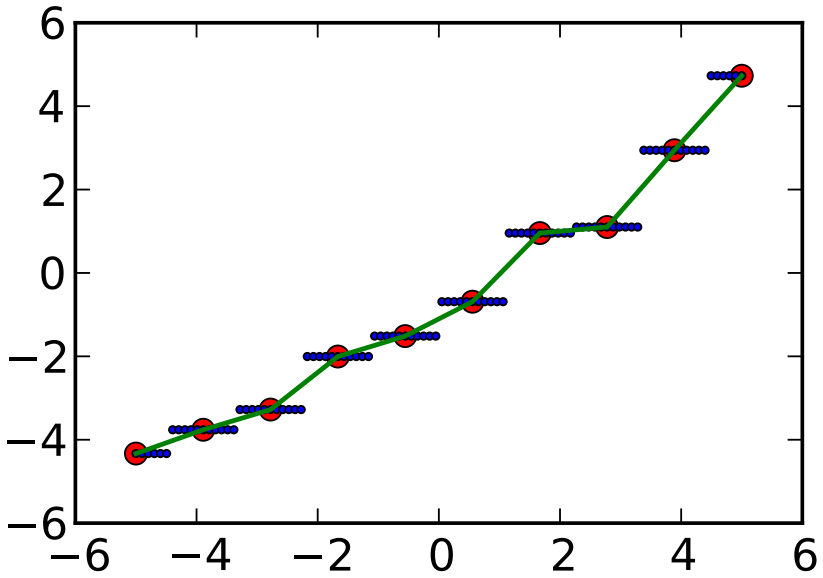
Regression, Interpolation

beispiel03.py

- Kurven auf Daten „fitten“
- Linear oder Polynome h. O.

- Interpolation
- Stückweise polynomial (spline)
- (beliebiger Ordnung)

- Mischform:
- „geglätter Spline“



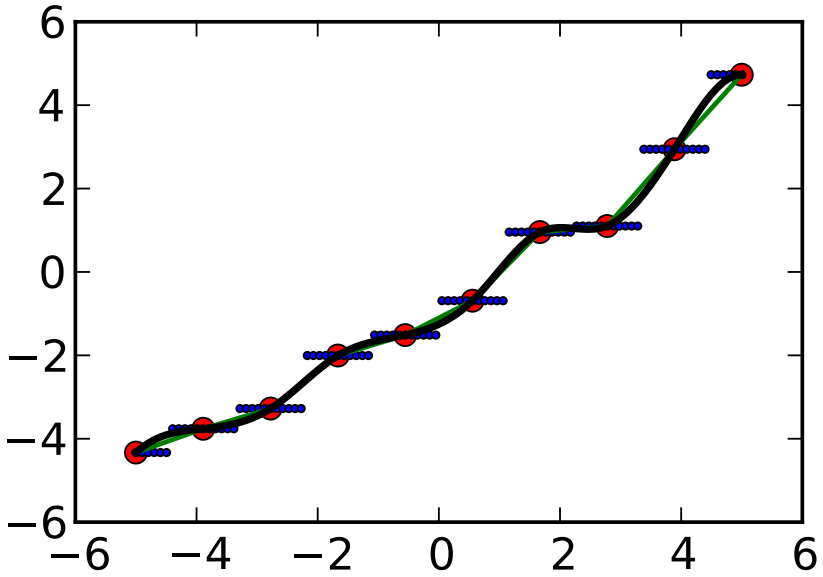
Regression, Interpolation

beispiel03.py

- Kurven auf Daten „fitten“
- Linear oder Polynome h. O.

- Interpolation
- Stückweise polynomial (spline)
- (beliebiger Ordnung)

- Mischform:
- „geglätter Spline“

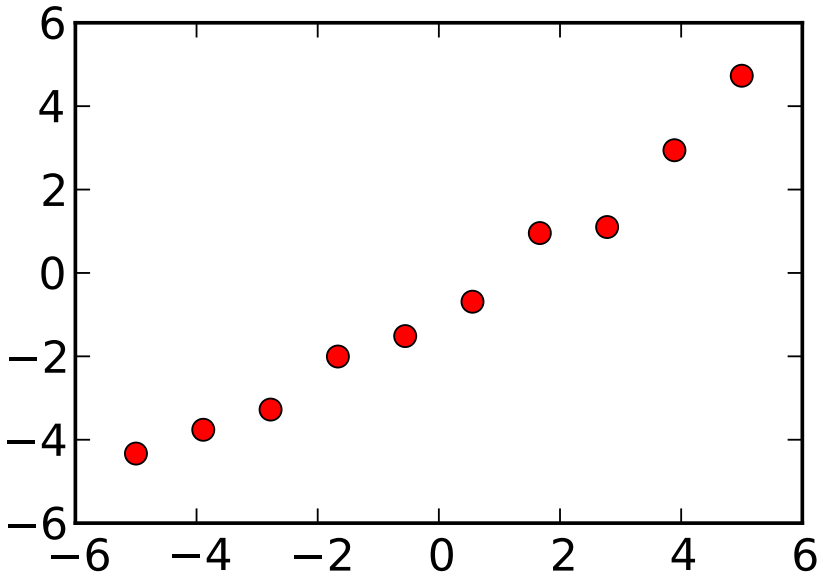


Regression, Interpolation *beispiel03.py*

- Kurven auf Daten „fitten“
- Linear oder Polynome h. O.

- Interpolation
- Stückweise polynomial (spline)
- (beliebiger Ordnung)

- Mischform:
- „geglätter Spline“



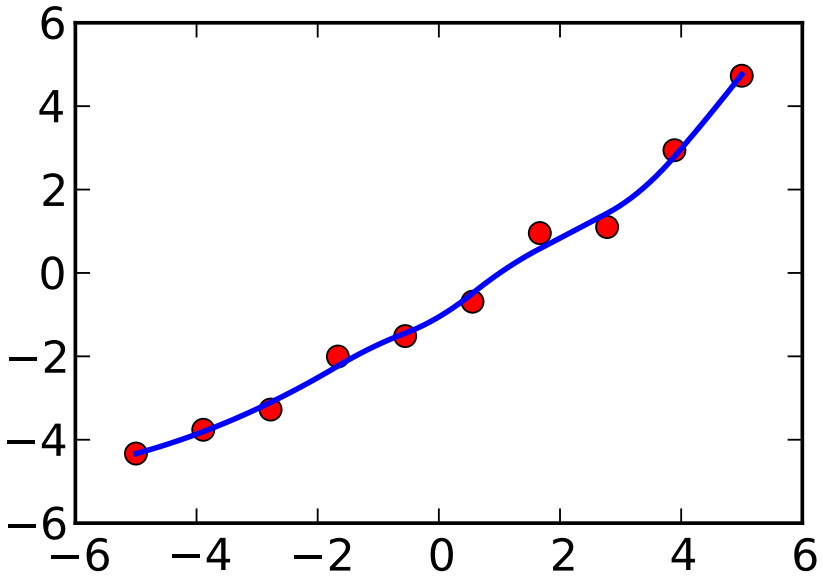
Regression, Interpolation

beispiel03.py

- Kurven auf Daten „fitten“
- Linear oder Polynome h. O.

- Interpolation
- Stückweise polynomial (spline)
- (beliebiger Ordnung)

- Mischform:
- „geglätter Spline“



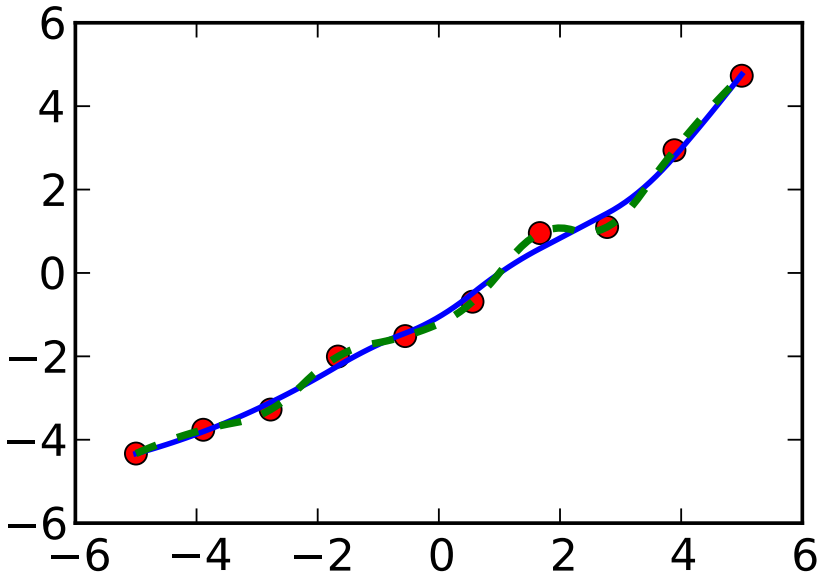
Regression, Interpolation

beispiel03.py

- Kurven auf Daten „fitten“
- Linear oder Polynome h. O.

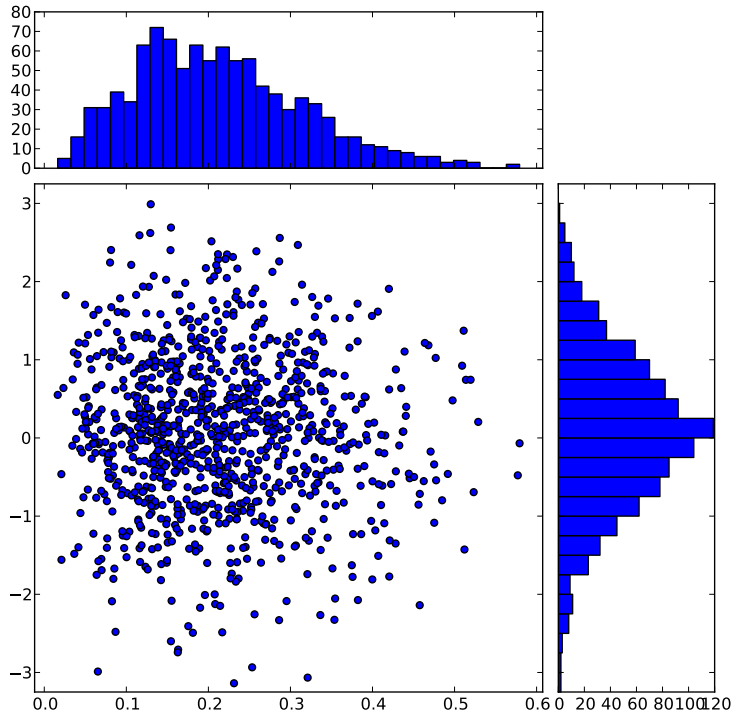
- Interpolation
- Stückweise polynomial (spline)
- (beliebiger Ordnung)

- Mischform:
- „geglätter Spline“



Statistik *beispiel04.py*

- Paket: `scipy.stats`
- Verteilungen (Gauss, Student-T, χ^2, Weibull, ..., Binomial, Hypergeom, ...)
- Für jede (kontinuierl.) Verteilung:
  - Dichtefunktion (`pdf`), Verteilungsfunktion (`cdf`)
  - Quantile
  - Entsprechender Zufallsgenerator
  - Mittelwert, Varianz, ...
- `histogram`
- Harmonisches u. geometrisches Mittel, Schiefe, ...



Numerische Integration *beispiel05.py*

Paket `scipy.integrate`

- DGL-Systeme in Zustandsdarstellungen lösen:

  $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$
- Beispiel harmonischer Oszillator: $\ddot{y} + 2\delta\dot{y} + \omega^2 y = 0$

```
def rhs(x,t):
    """ rhs = 'right hand side [function]' """

    x1_dot=x[1]
    x2_dot=-(2*delta*x[1]+omega_2*x[0])

    return [x1_dot, x2_dot]

t=np.arange(0,100,.01) # unabhaengige Variable (Zeit)
x0=[10,0]   # Anfangszustand fuer x, und x_dot
x=odeint(rhs, x0, t) # Aufruf des Integrators
```

- Außerdem: Trapezregel, Quadraturverfahren

Numerische Integration *beispiel05.py*

Paket `scipy.integrate`

- DGL-Systeme in Zustandsdarstellungen lösen:

  $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$

- Beispiel harmonischer Oszillator: $\ddot{y} + 2\delta\dot{y} + \omega^2 y = 0$

```
def rhs(x,t):
    """ rhs = 'right hand side [function]' """

    x1_dot=x[1]
    x2_dot=-(2*delta*x[1]+omega_2*x[0])

    return [x1_dot, x2_dot]

t=np.arange(0,100,.01) # unabhaengige Variable (Zeit)
x0=[10,0]   # Anfangszustand fuer x, und x_dot
x=odeint(rhs, x0, t) # Aufruf des Integrators
```

- Außerdem: Trapezregel, Quadraturverfahren

Gliederung

Grundkonzept

- Python-Bibliothek für symbolische Berechnungen („mit Buchstaben rechnen“)

→ Backend eines Computer Algebra Systems (CAS)

- Frontend: python-shell, ipython, eigenes skript

- Vorteil: CAS-Funktionalität zusammen mit richtiger Programmiersprache

- aktuelle Version: 0.7rc2 (→ aktiv in Entwicklung)
- Unterstützung durch das Google-Summer-of-Code Projekt

Rechnen *beispiel06.py*

```
import sympy as sp
x = sp.Symbol('x')
a,b,c = sp.symbols('a b c') # verschiedene Wege Symbole zu erz.

# Zusammenfassen
z = a*b*x*b + b**2*a*x - c*b*(2*a/c*x*b-1/(b*2))
print z # -> -b*c*(-1/(2*b) + 2*a*b*x/c) + 2*a*x*b**2

# Ausmultiplizieren:
print z.expand() # -> c/2

# Funktionen:
y = sp.sin(x)*sp.exp(x)*sp.sqrt(a)
print y # -> a**(1/2)*exp(x)*sin(x)

# Vereinfachungen:
print sp.trigsimp(sp.sin(x)**2+sp.cos(x)**2) # -> 1

# Differenzieren
f = sp.sinh(5*x**2)
print sp.diff(f, x) # -> 10*x*cosh(5*x**2)
```

Wichtige Datentypen / Methoden / Funktionen

- Subklassen von `Expr`: `Mul, Add, Pow, ...`
- Reihenentwicklung: `series(var)`
- Grenzwerte: `limit(var, value)`
- Integration: `integrate(f, x)`
- Vereinfachungen: `powsimp`, `radsimp`, `logcombine` `together`, `cancel`
- Zähler-Nenner-Aufspaltung: `to_num_denom`
- Lösen von LGS u. algebraischen Gl. `solve`

- Polynome: `Polynomial(x**7+a*x**3+b*x+c, x, domain='EX')`
- Stückweise definierte Funktionen `Piecewise(...)`
- Matrizen: `Matrix([[x, a+b], [c*x, sp.sin(x)]])`

- Pretty printing: `pprint(sp.pi**2/4)`
- Datentypen-Anpassung: `sympify`

Wichtige Datentypen / Methoden / Funktionen

- Subklassen von `Expr`: `Mul, Add, Pow, ...`
- Reihenentwicklung: `series(var)`
- Grenzwerte: `limit(var, value)`
- Integration: `integrate(f, x)`
- Vereinfachungen: `powsimp`, `radsimp`, `logcombine` `together`, `cancel`
- Zähler-Nenner-Aufspaltung: `to_num_denom`
- Lösen von LGS u. algebraischen Gl. `solve`

- Polynome: `Polynomial(x**7+a*x**3+b*x+c, x, domain='EX')`
- Stückweise definierte Funktionen `Piecewise(...)`
- Matrizen: `Matrix([[x, a+b], [c*x, sp.sin(x)]])`

- Pretty printing: `pprint(sp.pi**2/4)`
- Datentypen-Anpassung: `sympify`

Wichtige Datentypen / Methoden / Funktionen

- Subklassen von `Expr`: `Mul, Add, Pow, ...`
- Reihenentwicklung: `series(var)`
- Grenzwerte: `limit(var, value)`
- Integration: `integrate(f, x)`
- Vereinfachungen: `powsimp`, `radsimp`, `logcombine` `together`, `cancel`
- Zähler-Nenner-Aufspaltung: `to_num_denom`
- Lösen von LGS u. algebraischen Gl. `solve`

- Polynome: `Polynomial(x**7+a*x**3+b*x+c, x, domain='EX')`
- Stückweise definierte Funktionen `Piecewise(...)`
- Matrizen: `Matrix([[x, a+b], [c*x, sp.sin(x)]])`

- Pretty printing: `pprint(sp.pi**2/4)`
- Datentypen-Anpassung: `sympify`

Substitutionen *beispiel07.py*

- Vergleichbar mit `str.replace(alt, neu)`
- Nützlich für:
  - Manuelle Vereinfachungen,
  - Koordinatentransformationen
  - (partielle) Funktionsauswertungen

```
term1 = a*b*sp.exp(c*x)

term2 = term1.subs(a, 1/b)

print term2 # -> exp(c*x)
print term1.subs(a, a+1)# -> b*(1 + a)*exp(c*x)

num_werte = {a:2.5, b:0.4, c:-8, x:.125}
print term1.subs(num_werte) # -> exp(-1.0)
```

Numerische Formelauswertung *beispiel07.py*

- Gegeben: Formel, Werte der einzelnen Variablen
- Gesucht: numerisches Ergebnis
- Prinzipiell möglich: `expr.subs(num_werte).evalf()`
- Besser (bzgl. Geschwindigkeit): `lambdify`
- Erzeugt eine Python-Funktion, die man dann mit den Argumenten aufrufen kann

```
f = a*x*sp.atan(b*x)*sp.exp(c+a*x)
f_xa = f.diff(x).diff(a)  # beispielhafte Rechnung

f_xa_fnc = sp.lambdify((a,b,c,x), f, modules='numpy')

XX = np.linspace(-2, 2, 25)
print f_xa_fnc(0.4, 1.7, 3.2, XX) # -> [ 11.33..., ... ]
```

Gliederung

Was es sonst noch so gibt

numpy

- Fast-Fourier-Trafo (FFT)
- `np.vectorize`,
- Index-Tricks,(`r_`, `c_`),
- Integer-Array zum Indizieren nutzen
- Paket `scipy.optimize`
  Minimieren: `fmin, leastsq`
  NL-Gleichungen lösen: `fsolve`
- ...

sympy

- Code-Export (C, Fortran, Latex)
- `Assumptions` (Kommutativität, ...)
- „Pattern-Matching“
- Spezialfunktionalität für Quantenmechanik
- ...

- stdlib-Paket: `itertools`
  - `combinations`
  - `product` (Kartesisches Produkt, „jeder mit jedem“)

Was es sonst noch so gibt

numpy

- Fast-Fourier-Trafo (FFT)
- `np.vectorize`,
- Index-Tricks,(`r_`, `c_`),
- Integer-Array zum Indizieren nutzen
- Paket `scipy.optimize`
  Minimieren: `fmin, leastsq`
  NL-Gleichungen lösen: `fsolve`
- ...
  - stdlib-Paket: `itertools`
    - `combinations`
    - `product` (Kartesisches Produkt, „jeder mit jedem“)

sympy

- Code-Export (C, Fortran, Latex)
- `Assumptions` (Kommutativität, ...)
- „Pattern-Matching“
- Spezialfunktionalität für Quantenmechanik
- ...

Was es sonst noch so gibt

numpy

- Fast-Fourier-Trafo (FFT)
- `np.vectorize`,
- Index-Tricks,(`r_`, `c_`),
- Integer-Array zum Indizieren nutzen
- Paket `scipy.optimize`
  Minimieren: `fmin, leastsq`
  NL-Gleichungen lösen: `fsolve`
- ...

sympy

- Code-Export (C, Fortran, Latex)
- `Assumptions` (Kommutativität, ...)
- „Pattern-Matching“
- Spezialfunktionalität für Quantenmechanik
- ...

- stdlib-Paket: `itertools`
  - `combinations`
  - `product` (Kartesisches Produkt, „jeder mit jedem“)

Vorschläge für Übungsaufgaben

1. Die Beispiele durchgehen, anpassen und (interaktiv) Herumprobieren
2. Die DGL $\ddot{x} + \dot{x} + x^3 + ax$ für $a \in \{1, 0, -1\}$ lösen und die Lösung $x(t)$ grafisch darstellen.

   Hinweise: `sc.integrate.odeint`
3. Den mathematischen Ausdruck $x \sin(x)$ als Funktion im Intervall $[-1, 1]$ grafisch darstellen. Dann, die Ableitung und die Stammfunktion ebenfalls darstellen.

   Hinweis: Entweder numerisch (`np.diff, np.cumsum`) oder symbolisch (`sp.diff, sp.integrate, sp.lambdify`)
4. Eine symmetrische 3×3-Matrix erzeugen, deren Einträge entsprechend nummerierte Symbole sind (z.B. a11, a12, ...)

   Hinweis: Der Konstruktor von `sp.Matrix` akzeptiert als 3. Argument eine Funktion von i, j.

Links

- `http://www.scipy.org/Tentative_NumPy_Tutorial`
- `http://www.scipy.org/EricsBroadcastingDoc`
- `http://www.scipy.org/NumPy_for_Matlab_Users`
- `http://www.scipy.org/Cookbook`

- `http://docs.sympy.org/dev/tutorial.html`
- `http://docs.sympy.org/dev/gotchas.html` (Fallstricke)

- Generell meist sehr erfolgreich:
  Suchanfragen á la `numpy eigenvalues`