Mit Berechnung

SCaVis – Werkbank für technisch-wissenschaftliche Berechnungen und Visualisierungen mit Java und Jython

Klaus Rohe

SCaVis ist eine vollständig in Java implementierte Software für numerische und symbolische Mathematik sowie die zwei- und dreidimensionale Datenvisualisierung.

SCaVis ist unter den in [SCaVisFea] genannten Bedingungen lizenziert, Teile des Quellcodes sind nach den Bedingungen der GPL verfügbar (s. [SCaVis]). Die interaktive Durchführung von Berechnungen und Visualisierungen geschieht über eine integrierte Entwicklungsumgebung. Sie ermöglicht auch die Automatisierung der Arbeitsvorgänge mit dem Python-Dialekt Jython, der BeanShell oder einer „MATLAB-ähnlichen" Skriptsprache. Die Entwicklung von SCaVis wurde 2005 von Dr. Sergei Chekanov am DESY in Hamburg gestartet. Es wurde seitdem als Gemeinschaftsprojekt von ihm weitergeführt, während er am CERN an Experimenten im Bereich der Hochenergiephysik arbeitete. Es beinhaltet über 50 Java-Open-Source-Projekte (s. [Chek10], Abschnitt 1.5).

SCaVis hatte ursprünglich den Namen jHepWork, wobei das „Hep" in dem Namen die Abkürzung für High Energy Physics ist. Die Umbenennung in SCaVis (*Scientific Computation and Visualization Environment*) soll reflektieren, dass es für alle Arten von technischen und wissenschaftlichen Berechnungen und Visualisierungen sehr gut geeignet ist. Im Folgenden wird ein Überblick über dieses interessante Werkzeug gegeben.

Installation

SCaVis kann als zip-Datei von der in Referenz [SCaVis] angegebenen Webseite heruntergeladen werden. Die zip-Datei entpackt man in ein beliebiges Verzeichnis. Dort entsteht dann ein Unterverzeichnis mit dem Namen „SCaVis", welches alle notwendigen Komponenten enthält. Zum Starten von SCaVis muss man auf Windows-Betriebssystemen die Batch-Datei „SCaVis.bat" ausführen und auf UNIX- und LINUX-Systemen das Shell-Skript „SCaVis.sh". Die Skripte setzen die von SCaVis benötigten Umgebungsvariablen, wie den CLASSPATH und die Parameter für die JVM. Die aktuelle Version ist SCaVis 1.1 (Juni 2013). Sie setzt voraus, dass Java 1.7.x auf dem Rechner installiert ist.

Die integrierte Entwicklungsumgebung

Die integrierte Entwicklungsumgebung von SCaVis ist in Abbildung 1 mit ihren sieben Hauptteilen zu sehen. Links im Bild ist der File-Explorer geöffnet, der zur Verwaltung der Dateien und Navigation im Dateisystem dient. Im gleichen Bereich gibt es noch den Code-Explorer, der die Struktur des Quellcodes (Klassen, Methoden, Funktionen) anzeigt, der rechts im Editor geöffnet ist. Unter dem Editor befinden sich drei verschiedene Shells und eine Systemkonsole. Die Shells kann man für die interaktive Arbeit mit den unterstützten Skriptsprachen nutzen.

In Abbildung 1 ist die Shell für Jython geöffnet. Sie gestattet das interaktive Arbeiten mit Jython innerhalb von SCaVis. Die „Bean Shell" steht für BeanShell Scripting zur Verfügung [BeanShell] und die „jMathLab Shell" unterstützt das Scripting in einer mit „MATLAB/Octave" vergleichbaren Skriptsprache [JMathLab]. „jMathLab" basiert auf dem Computeralgebrasystem „jasymca", welches von Professor Helmut Dersch an der Hochschule Furtwangen entwickelt wird [Dersch09]. Eine weitere Möglichkeit, symbolische Mathematik zu nutzen, bietet das Python-Framework „sympy" [sympy], welches ebenfalls in SCaVis enthalten ist.

Abbildung 2 zeigt die SCaVis-Menüleiste. Die Menüs „File", „Edit" und „Search" bieten die üblichen Funktionen zum Verwalten und Editieren von Dateien sowie entsprechende Such- und Ersetzungsfunktionen. Mit „View" kann man das „Syntax Coloring" einstellen und mit der Funktion „CodeView" eine HTML-Datei des Quellcodes erzeugen, der sich gerade im Editor befindet. Diese HTML-Dateien werden im Unterverzeichnis „cachedir" von SCaVis gespeichert.

Mit „Run" kann man Skripte (Jython, BeanShell, MATLAB/Octave) ausführen, aber auch Java-Quellcode kompilieren und jar-Dateien erzeugen. Das Menü „Plots" bietet Funktionen, um zwei- oder dreidimensionale Plots aus Daten zu erzeugen, die in Dateien gespeichert sind. Außerdem wird eine

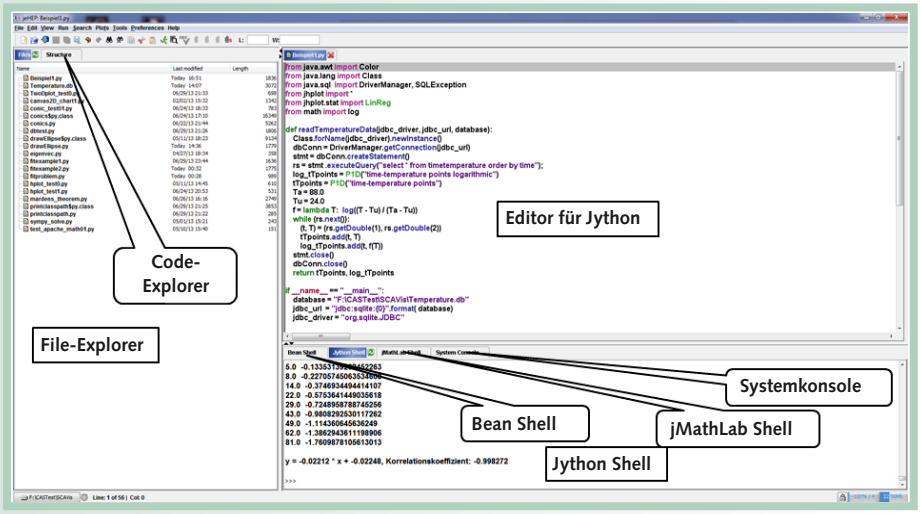

Abb. 1: Ansicht der SCaVis-Entwicklungsumgebung auf Windows

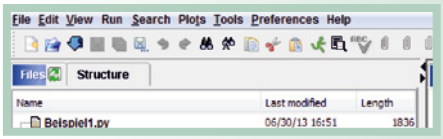

Abb. 2: Menüleiste von SCaVis

Funktion angeboten, welche ein Werkzeug startet, mit dem man Funktionen einer reellen Variablen definieren kann, um diese dann für einen bestimmten Wertebereich grafisch darzustellen.

Das Menü „Tools“ bietet eine Reihe von Werkzeugen an, unter anderem: einen Bildeditor, einen LateX-Editor, ein Zeichenprogramm, eine einfache Tabellenkalkulation, ein interaktives Periodensystem der chemischen Elemente und die Möglichkeit zur Umrechnung von physikalischen und technischen Größen in verschiedene Einheitensysteme. Mit „Preferences“ kann man unter anderem die Einstellungen für den Editor, für die Art der Darstellung der Entwicklungsumgebung (Windows, Motif, …) und das Projektverzeichnis ändern. Mit „Help“ kann man auf die Dokumentation zugreifen und neue Updates installieren. Die beschriebenen Funktionen der Entwicklungsumgebung spiegeln sich in der Komponentenstruktur von SCaVis wieder, die in Abbildung 3 dargestellt ist.

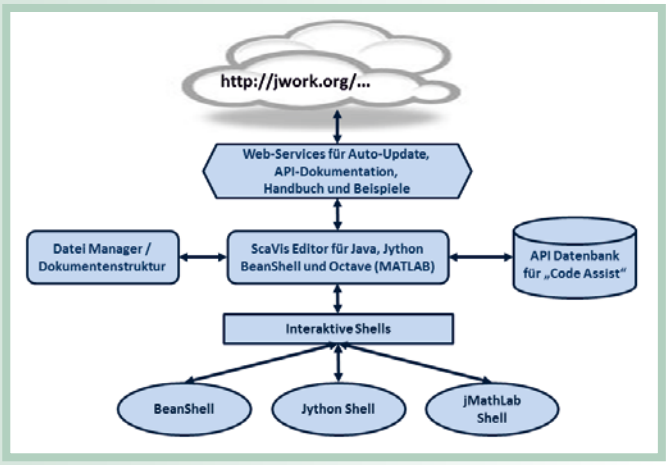

Abb. 3: Komponentenstruktur von SCaVis

Zu ergänzen ist noch die Aufgabe von „Code Assist“. Es gibt dem Entwickler die Möglichkeit, sich in den Konsolen der Shells und den Editoren die Dokumentation der Methoden von Klassen anzeigen zu lassen, die gerade instanziiert wurden. Details dazu sind in [Chek10], Seite 21 zu finden.

Im Folgenden wird anhand eines nicht trivialen, aber nicht zu umfangreichen Beispiels gezeigt, wie man mit SCaVis und Jython arbeitet und wie Java-Bibliotheken von Jython aus genutzt werden können. Eine gute Einführung in die Grundlagen von Jython wird im Kapitel 2 von [Chek10] gegeben. Eine ausführliche Darstellung von Jython im Web ist [JBNSW10].

Beispiel

In dem Beispiel geht es darum, eine Funktion an Messdaten anzupassen. Gemessen wurde die Abkühlung von 250 cm³ Kaffee als Funktion der Zeit. Diese Abkühlungsfunktion hat die Form

$$T(t) = T_u + (T_a - T_u) * \exp(-kt)$$

Dabei ist T_u die Umgebungstemperatur, T_a die Anfangstemperatur, *exp* die natürliche Exponentialfunktion und k eine Konstante, welche die Abkühlgeschwindigkeit bestimmt, t ist die Zeit. Die Temperaturen wurden in C° gemessen, die Zeit in Minuten. Die Konstante k hat damit die Einheit 1/Minuten.

```
 1 # -*- coding: cp1252 -*-
 2 from java.awt import Color, Font
 3 from java.lang import Class
 4 from java.sql  import DriverManager
 5 from jhplot import *
 6
 7 def lese_tTDaten(jdbc_driver, jdbc_url):
 8   Class.forName(jdbc_driver).newInstance()
 9   dbConn = DriverManager.getConnection(jdbc_url)
10   stmt = dbConn.createStatement()
11   rs = stmt .executeQuery(
       "select * from zeittemperatur order by t")
12   tT = P1D()
13     while (rs.next()):
14       tT.add(rs.getDouble(1), rs.getDouble(2))
15    stmt.close()
16    dbConn.close()
17    return tT
18
19 if __name__ == "__main__":
20   db = "F:\Temperatur.db"
21   jdbc_url  = "jdbc:sqlite:{0}".format(db)
22   jdbc_driver = "org.sqlite.JDBC"
23
24   tT = lese_tTDaten(jdbc_driver, jdbc_url)
25   tT.setColor(Color.red)
26   tT.setTitle("Messwerte")
27   tT.setPenWidth(4)
28
29   fitter = HFitter("leastsquares")
30   fitter.setFunc("Abkfunk", 1, "Tu + (Ta - Tu) * exp(-k * x[0])",
       "k, Ta, Tu")
31   fitter.setPar("k", 0.02)
32   fitter.setPar("Ta", 90.0)
33   fitter.setPar("Tu", 20.0)
34   fitter.setRange(0, 135)
35   fitter.fit(tT)
36   abkfunk = fitter.getFunc()
37   fitres = fitter.getResult()
38   k = fitres.fittedParameter("k")
39   Ta = fitres.fittedParameter("Ta")
40   Tu = fitres.fittedParameter("Tu")
41
42   c = HPlot("Canvas", 1000, 1000)
43   c.visible()
44   c.setLegendFont(Font("Arial", Font.BOLD, 20))
45   c.setGrid(0, 1)
46   c.setGrid(1, 1)
47   c.setGTitle("Abkühlung von 250 cm³ Kaffee als Funktion der Zeit")
48   c.setNameX("t [Minuten]")
49   c.setNameY("T [C°]")
50
51   max_X = max(tT.getArrayX())
52     fstr = "T(t) = {0:.0f} C° + {1:.0f} C° * exp(-{2:.3f}/min * t)"
        .format(Tu, Ta - Tu, k)
53   f = F1D(fstr, abkfunk, 0, max_X)
54   f.setColor(Color.blue)
55   f.setPenWidth(3)
56   c.setRange(-5, 135, 20, 100)
57   c.draw(tT)
58   c.draw(f)
59   c.export("F:\Abb4.png")
```

Listing 1: Jython-Skript Abkühlung.py

Die Messwerte wurden in einer Tabelle in einer SQLite-Datenbank gespeichert. Das Jython-Skript „Abkühlung.py“ in Listing 1 erledigt die hier beschriebene Aufgabe. Zeile 1 bewirkt, dass man auch Umlaute und andere Nicht-ASCII-Zeichen in den Strings verwenden kann. In den Zeilen 2 – 4 werden die vom AWT und die für JDBC benötigten Java-Klassen importiert. In Zeile 5 werden die Java-Klassen der SCaVis-Komponente *jhplot* importiert. Dies sind Klassen zum Anpassen von Funktionen an Messdaten, zum Auswerten von Funktionen und zum Plotten.

In den Zeilen 7 – 17 ist die Jython-Funktion `lese_tTDaten` definiert, welche die Messdaten per JDBC aus einer Tabelle `zeittemperatur` ausliest, die sich in einer SQLite-Datenbank befindet. Damit dies funktioniert, muss die Umgebungsvariable `CLASSPATH` den Pfad zum SQLite-JDBC-Treiber enthalten. In den Zeilen 8 – 10 wird die Verbindung zur Datenbank aufgebaut und in den Zeilen 10 und 11 wird ein SQL-SELECT-Befehl ausgeführt. Dabei werden die Messdaten aus der Tabelle in die Variable `rs` übertragen, die vom Typ `JDBC ResultSet` ist. In Zeile 12 wird eine Variable `tT` als Klasse `P1D` initialisiert. `P1D` ist ein Container-Typ aus jhplot. Er wird zum Plotten der Messwerte und zur Anpassung der oben beschriebenen Funktion benutzt. Der Container wird in der `while`-Schleife mit der `add`-Methode mit den Daten aus `rs` gefüllt. Die Methode wird hier mit zwei Parametern aufgerufen: Zeit (`rs.getDouble(1)`) und Temperatur (`rs.getDouble(2)`).

Die Funktion gibt in Zeile 17 die Variable `tT` an ihren Aufrufer zurück. In der `main`-Funktion des Jython-Skripts wird in den Zeilen 20 – 22 die Verbindung zur konkreten SQLite-Datenbank festgelegt. In Zeile 24 wird die Funktion `lese_tTDaten` mit den entsprechenden Parametern aufgerufen. Das Ergebnis wird in der Variablen `tT` vom Typ `P1D` gespeichert. In den Zeilen 25 – 27 wird definiert, mit welcher Farbe, Strichstärke und Titel die Messwerte geplottet werden. In den Zeilen 29 – 40 wird die oben diskutierte Abkühlungsfunktion, welche die Parameter k, T_a und T_u enthält, an die Messwerte angepasst. Dies geschieht mit der Klasse `HFitter` aus jhplot. Sie wird in Zeile 29 mit dem Parameter `leastsquares` initialisiert, was festlegt, dass zur Anpassung die Methode der kleinsten Fehlerquadrate benutzt wird.

In Zeile 30 wird die Funktion angegeben, die angepasst werden soll, sowie die Parameter, die zum Fitten zu benutzen sind. Das geschieht mit der Methode `setFunc` der Klasse `HFitter`. Der erste Parameter der Methode ist ein Name für die Funktion, der zweite gibt die Anzahl der unabhängigen Variablen der Funktion an, der dritte ist ein String, der die anzupassende Funktion definiert, und der letzte Parameter ist ein String, der durch Kommata getrennt die Namen der Parameter enthält, die zum Fitten benutzt werden sollen. In den Zeilen 31 – 33 werden geschätzte Startwerte für die anzugleichenden Parameter gesetzt. Mit der Methode `setRange` wird der zu beachtende Wertebereich der unabhängigen Variablen angegeben. Dies ist hier die Zeit t. Das eigentliche Fitten geschieht mit der Methode `fit` in Zeile 35, die als Parameter das Containerobjekt `tT` vom Typ `P1D` übergeben bekommt. Die angepasste Funktion wird dann mit der Methode `getFunc` geholt. In den Zeilen 38 – 40 werden die Werte für die angepassten Parameter geholt. In der Zeile 42 wird eine Variable `c` vom Typ `HPLOT` mit Namen „Canvas“ und der Größe von 1000 x 1000 Pixeln instanziiert und in den folgenden Zeilen mit entsprechenden Eigenschaften versehen, wie Titel, Hilfsgitter sowie Name der X- und Y-Achse. In Zeile 49 wird das Maximum der t-Werte der Variablen `max_X` zugewiesen. In Zeile 51 wird eine String-Variable `fstr` konstruiert, welche die gefittete Funktion beschreibt. Diese werden in Zeile 52 zum Anlegen der Variable `f` vom Typ `F1D` benutzt.

`F1D` ist eine Klasse aus jhplot und dient zur effizienten Handhabung der X-Y-Werte von Funktionen einer unabhängigen Variablen und ihrer grafischen Darstellung. Der Konstruktor von `F1D` bekommt als weitere Parameter die angepasste Funktion und die Grenzen ihres betrachteten Wertebereichs. `F1D` ist ein Container, der die Definition einer Funktion einer unabhängigen Variablen plus ihrer Werte über einem bestimmten diskreten Wertebereich enthält. In jhplot gibt es noch die Funktionscontainer `F2D` und `F3D` für Funktionen von zwei respektive drei unabhängigen Variablen (s. dazu [Chek10], Abschnitt 10.11 und 10.12).

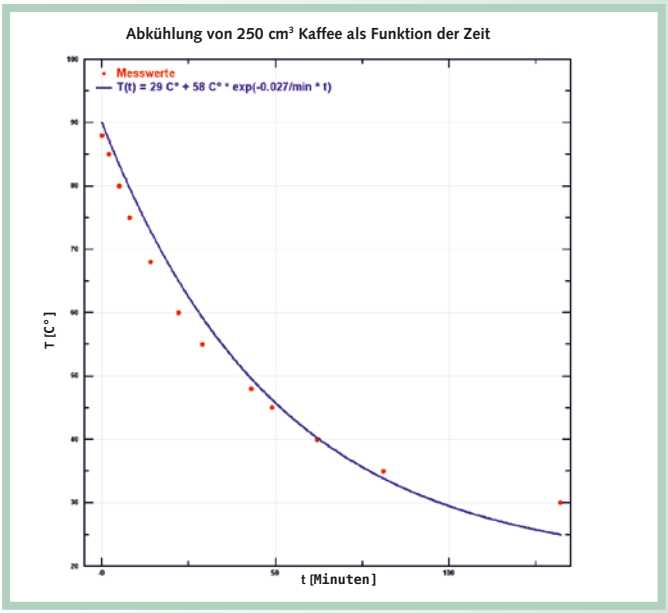

Abb. 4: Grafische Ausgabe des Jython-Skripts aus Listing 1, die angepasste Funktion ist blau dargestellt und die Messwerte rot

Es werden dann noch die Zeichenfarbe und Strichstärke für die grafische Darstellung der Funktion gesetzt. In den Zeilen 57 – 59 wird die eigentliche grafische Ausgabe der Messwerte und der gefitteten Funktion durchgeführt. In Zeile 54 werden die Randwerte für die X- und Y-Achse gesetzt, dann werden in Zeile 57 die Messwerte grafisch dargestellt und in Zeile 58 die angepasste Funktion. Die letzte Zeile in Listing 1 speichert die Grafik in einer png-Datei, die in Abbildung 4 zu sehen ist.

In dem Beispiel konnte nur ein sehr kleiner Ausschnitt der Möglichkeiten von SCaVis dargestellt werden. SCaVis bietet noch mächtige Funktionen für numerische Analysis und numerische lineare Algebra sowie für statistische Analysen. Wer sich tiefer mit SCaVis befassen möchte, sollte das in [Chek10] genannte Buch zur Hand nehmen und sich die umfangreiche Dokumentation anschauen.

Zusammenfassung

SCaVis ist ein sehr leistungsfähiges System für technische und wissenschaftliche Berechnungen und deren Visualisierung. Es unterstützt sowohl numerisches als auch symbolisches Rechnen. Da SCaVis in reinem Java implementiert ist, kann es auf allen Betriebssystemen eingesetzt werden, für welche die geforderte Java-Version verfügbar ist. Eine Portierung von SCaVis auf Android gibt es ebenfalls [SCaVisAndr].

Die Funktionalität von SCaVis lässt sich sowohl über die unterstützten Skriptsprachen als auch über Java erweitern. Dadurch ist SCaVis auch sehr einfach mit anderen Applikationen, die zum Beispiel REST-Schnittstellen besitzen, integrierbar. Die Anbindung an Datenbanken wurde exemplarisch in dem Beispiel gezeigt. Ebenso ist es natürlich kein Problem, SCaVis mit JMS-Systemen zu verbinden oder andere Java-Applikationen über RMI anzusprechen. Auch im Bereich „Big Data“ ist SCaVis als Auswertungs- und Visualisierungswerkzeug sehr gut geeignet.

Zum Schluss bedanke mich herzlich bei Herrn Dr. Sergei Chekanov vom Argonne National Laboratory für seine sehr hilfreiche Beantwortung meiner Fragen und dafür, dass er mir das Architekturschaubild (Abb. 3) zur Verfügung gestellt hat.

Literatur und Links

[BeanShell] Lightweight Scripting for Java, http://www.beanshell.org/
[Chek10] S. V. Chekanov, Scientific Data Analysis using Jython Scripting and Java, Springer Verlag, 2010
[Dersch09] H. Dersch, Jasymca 2.0 - Symbolischer Rechner für Java, 2009, http://webuser.hs-furtwangen.de/~dersch/jasymca2/Jasymca2de/Jasymca2de.html, http://webuser.hs-furtwangen.de/~dersch/
[JBNSW10] J. Juneau, J. Baker, V. Ng, L. Soto, F. Wierzbicki, The Definitive Guide to Jython, 2010, http://www.jython.org/jythonbook/en/1.0/
[JMathLab] (J) MathLab – A multiplatform computational platform, http://jwork.org/jmathlab/
[SCaVis] Scientific Computation and Visualization Environment,Webseite mit Möglichkeiten zum Download, http://jwork.org/scavis/
[SCaVisAndr] http://jwork.org/scavis/android
[SCaVisFea] SCaVis features and license, http://jwork.org/scavis/features
[sympy] Python library for symbolic mathematics, http://code.google.com/p/sympy/

Klaus Rohe beschäftigt sich mit Softwaretechnik und Integrationstechnologien.
E-Mail: klaus-rohe@t-online.de