

Handbuch zu pyvLab

Christoph Würstle

17. Mai 2006

Inhaltsverzeichnis

1	Einleitung	3
2	Installation	3
3	Unterstützte Messgeräte	5
4	Features	6
4.1	Aufbau der Oberfläche	6
4.2	Einstellen einer Messung	7
4.3	Einfaches Beispiel	8
4.4	Komplexeres Beispiel	8
4.5	Graph	9
4.6	Speichern	9
4.7	VisaTester	10
5	Messgeräte hinzufügen	10
5.1	Einbinden einfacher Geräte	11
5.2	Beispiel	12
5.3	Einbinden komplexer Geräte	14
5.4	Heizer	16
5.5	Spektrum	17
5.6	pseudoVisa	18

1 Einleitung

Um verschiedenste Messungen an den Proben effizient auszuführen wurde das Messprogramm pyvLab geschrieben. PyvLab steht für Python+Visa+Labor(-atory). Realisiert wurde es, wie der Name sagt, in der Sprache Python und unter Verwendung des National Instruments VISA Treibers. Eine Installationsanleitung ist in Kapitel 2 zu finden. Eine Beschreibung der Features von pyvLab folgt in Kapitel 4. Die standardmäßig unterstützten Messgeräte sind in Kapitel 3 aufgeführt. Das Programm ist so aufgebaut, dass leicht neue Instrumente hinzugefügt werden können, siehe dazu Kapitel 5.

2 Installation

Dieses Kapitel beschreibt die Installation der von pyvLab benötigten Bibliotheken. Diese stehen für Windows (ab Windows 98) und Linux zur Verfügung, was die freie Wahl des Betriebssystems ermöglicht. Vorausgesetzt wird nur, dass eine GPIB-Karte

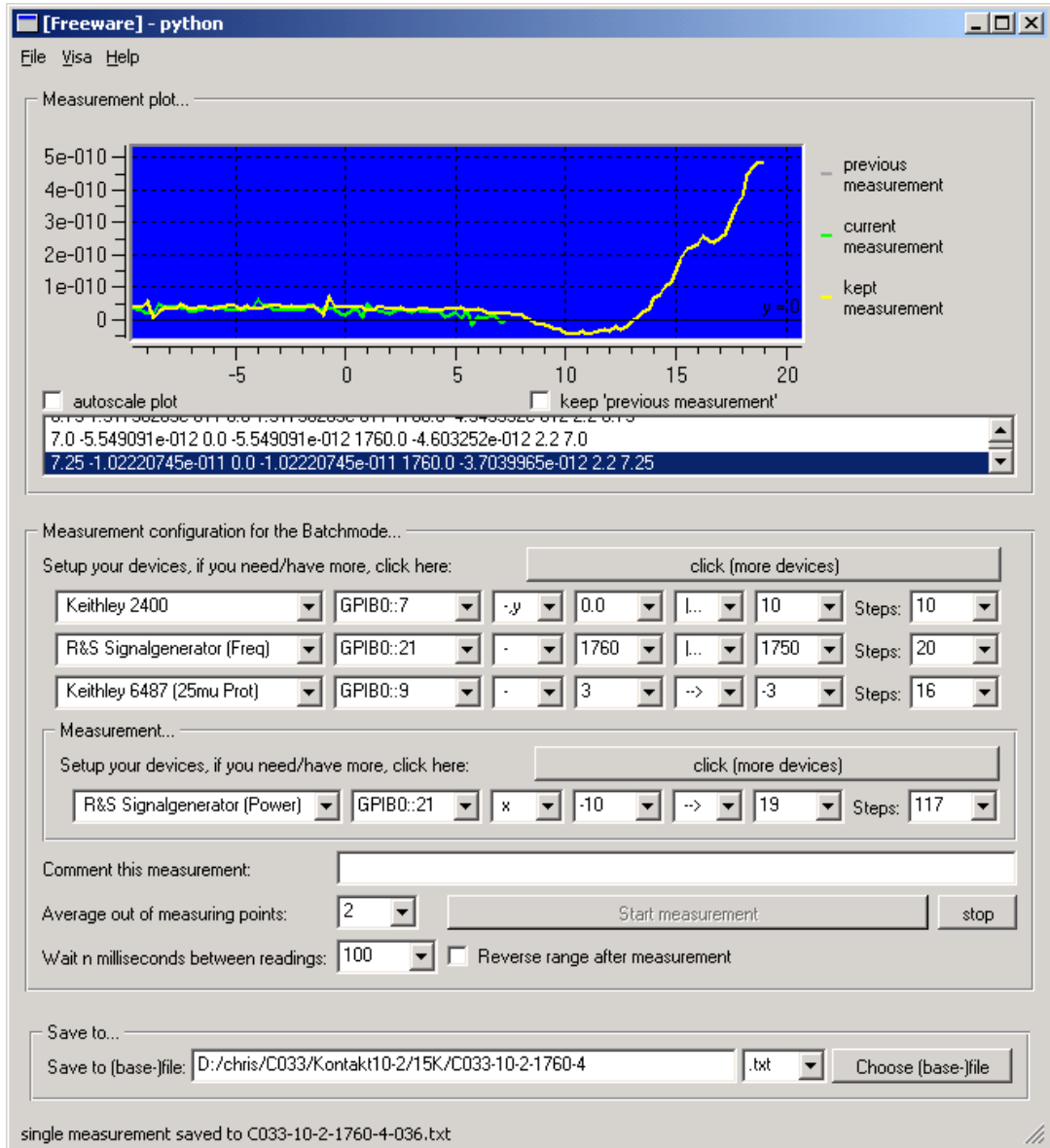


Abbildung 1: Screenshot des Messprogramms pyvLab.

(oder ein GPIB-USB-Konverter) bereits installiert ist, welche/r mit dem National Instruments VISA Treiber zusammenarbeitet. Letzterer ist auf <http://www.ni.com/visa> zu finden und sollte meist bereits mit dem Kartentreiber installiert worden sein.

PyvLab benötigt verschiedene Programme, welche unter Windows alle mit einem normalen Aufrufen des Setups installiert werden können. Ein Linuxanwender sollte die meisten schon bei einer Standardinstallation seiner Distribution installiert bekommen haben. Folgende Pakete müssen installiert werden; die Links in der folgenden Auflistung beziehen sich auf die jeweiligen Windows Bibliotheken:

1. Python ab Version 2.3, z.B. von <http://www.python.org/ftp/python/2.3.5/Python-2.3.5.exe>, nötig um Python Programme auszuführen
2. Numeric, z.B. von <http://prdownloads.sourceforge.net/numpy/Numeric-23.7.win32-py2.3.exe?download>, erweitert Python um numerische Klassen
3. Qt Version 2.3 für Windows, von <ftp://ftp.trolltech.com/qt/non-commercial/QtWin230-NonCommercial.exe>, Toolkit für die Programmoberfläche
4. PyQt, von <http://pyqwt.sourceforge.net/support/PyQt-win-nc-msvc-3.13.exe>, Python-Anbindung an die QT Bibliothek
5. pyQwt, von <http://prdownloads.sourceforge.net/pyqwt/PyQwt.Qt230-4.2.win32-py2.3.exe?download>, Plotbibliothek zur Darstellung der Messdaten
6. ctypes, von <http://sourceforge.net/projects/ctypes>, benötigt von pyVisa
7. pyVisa ab Version 0.9.6, von pyvisa.sf.net, Python-Anbindung an den NI-VISA Treiber

Anschließend lässt sich pyvLab (pyvlab.sf.net) ohne Installation durch Ausführen von *pyvlab.py* starten. Ein Screenshot ist in Abbildung 1 zu sehen.

3 Unterstützte Messgeräte

Tabelle 1 listet die bereits von pyvLab unterstützten Messgeräte auf. Daneben stehen noch die in Tabelle 2 aufgezählten virtuellen Instrumente zur Verfügung. Neue Instrumente lassen sich leicht hinzufügen. Eine detaillierte Erklärung hierzu findet sich in Kapitel 5.

Tabelle 1: Von pyvLab unterstützte Messgeräte

Gerät	Hinweise
Keithley 213	Spannungspports unabhängig steuerbar unter :fetch? compatible device nur Spannungssteuerung implementiert benötigt Reset nach dem Einschalten, siehe 4.7 benötigt Reset nach dem Einschalten, siehe 4.7 Strom- und Spannungssetzen implementiert sowohl Rohde&Schwarz wie auch Agilent Auslesen des Spektrums, wie auch der Marker verschiedene Modi verfügbar
Keithley 2000	
Keithley 2400	
Keithley 6485	
Keithley 6487	
Agilent E3646A	
Signalgeneratoren	
R&S Network Analyzer ZVC	
Neocera Temp. Controler	
Lock-In Ithaco 3961b	
Lock-In Stanford SR510	-

4 Features

PyvLab eignet sich für alle Messungen, bei denen ein oder mehrere Parameter von Messgeräten unabhängig voneinander verändert werden. Die maximale Anzahl der verwendeten Geräte ist nur durch den GPIB-Bus beschränkt.

4.1 Aufbau der Oberfläche

Die Oberfläche ist dreigeteilt. Der oberste Bereich beinhaltet den Plotbereich. Hier werden ausgewählte Werte während der Messung aufgetragen. Im unteren Teil der Programmoberfläche gibt man an, wo die Messdaten gespeichert werden. Dazwischen ist der Bereich, in dem man Messgeräte auswählt und ihre Konfiguration setzt.

Dieser ist ebenfalls grob dreigeteilt:

Die eigentliche Messung wird dort im mittleren Bereich eingestellt. Details hierzu folgen. Der obere Bereich kann benutzt werden um nach Beendigung der eigentlichen Messung einen Parameter einen Schritt weiter zu setzen und die eigentliche Messung (vom mittleren Teil) zu wiederholen. Dieser Modus wird im Folgenden Batchmode genannt. Im unteren Bereich lassen sich verschiedene Messparameter konfigurieren. Darunter fällt die Anzahl der Messwerte, über die gemittelt werden soll, wie auch der zeitliche Abstand, in dem diese ausgelesen werden.

Tabelle 2: Virtuelle Geräte, die pyvLab unterstützt

Gerät	Hinweise
Standby Time	Wartet die angegebene Zeit (Sekunden), bis zum nächsten Messschritt gegangen wird.
Constant	Liefert an jedem Messwert die angegebene Konstante zurück.
Counter	Zählt die Messschritte ab einem angegebenen Startwert mit.
Stop at yValue	Beendet die Messung, sobald der auf der y-Achse geplottete Wert den hier angegebenen erreicht.
Integral	Liefert an jedem Messwert das (numerisch berechnete) Integral des Graphen zurück.
Stop at Integral	Beendet die Messung sobald das Integral des geplotteten Graphen den hier angegebenen Wert erreicht.

4.2 Einstellen einer Messung



Abbildung 2: Kontrolleiste zur Steuerung eines Messgeräts.

Eine Messung lässt sich folgendermaßen einstellen, vgl. hierzu Abbildung 2: ein ansteuerbares Messgerät wird in der Dropdownbox (1) ausgewählt, daneben die korrekte VISA Adresse (2) eingetragen. Fährt man mit der Maus über (3) erscheint ein Tooltip (siehe Abbildung 3), der auch die Rückgabewerte des gewählten Messgeräts anzeigt. Der erste und/oder zweite Parameter kann geplottet werden. Hierfür wählt man die passenden Achsen in (3) aus. Nun kann der Parameter des gewählten Geräts zwischen zwei Werten in einer gewählten Schrittzahl variiert werden: von einem Startwert (4) bis zu einem Zielwert (6) in einer Anzahl von Schritten (7). Mittels (5) lassen sich die Werte von (4) bis (6) variieren oder auch fest auf (4) setzen, was bei Verwendung mehrerer Messgeräte Bedeutung erlangt. Für jedes Messgerät wird eine Kontrolleiste benötigt. Mittels des Klickbuttons lässt sich eine neue Zeile im jeweiligen Bereich hinzufügen.

Bei Geräten, die mehrere Parameter besitzen (z.B.: Signalgeneratoren), existiert für jeden ein eigener Eintrag in der Liste der Geräte (z.B.: Signalgenerator Power und Signalgenerator Frequenz). Falls nur ein Parameter verändert werden soll, wählt man nur diesen aus der Liste aus und setzt den anderen am Messgerät oder man wählt auch den anderen aus und setzt diesen Parameter auf einen festen Wert (in der dazugehörigen Dropdownbox (5)).

Zwei Beispiele veranschaulichen die Bedienung:

4.3 Einfaches Beispiel

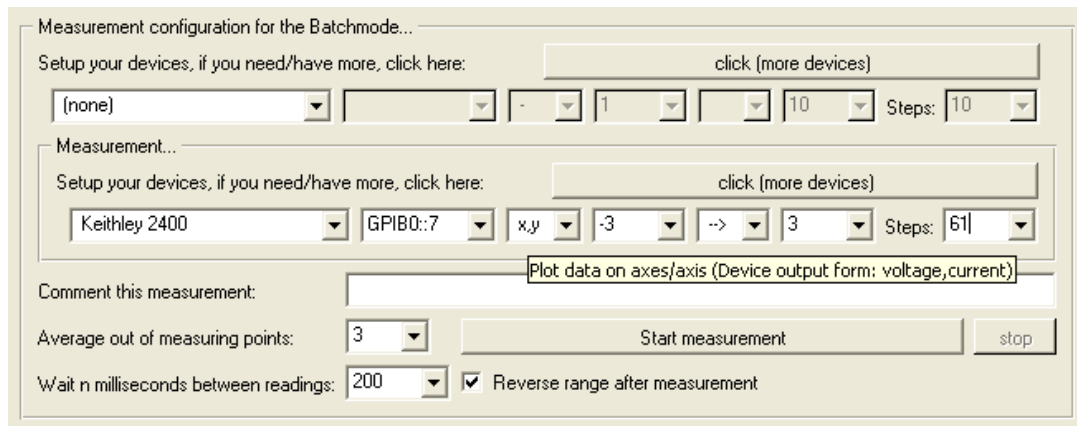


Abbildung 3: Einfaches Beispiel zur Steuerung der Spannung eines Keithley 2400 von -3V bis +3V in 61 Schritten.

In Abbildung 3 ist eine einfache Einstellung zu sehen. Ein Keithley 2400 mit der GPIB-Adresse 7 variiert die Spannung von -3V bis +3V in 61 Schritten. Hierbei werden an jedem Spannungswert drei Strommessungen mit mindestens 200 Millisekunden Abstand genommen und darüber gemittelt. Wie in dem Tooltip zu sehen ist, gibt das Keithley zwei Werte zurück: die angelegte Spannung und den fließenden Strom. Diese sollen geplottet werden. Hierfür wurde in der dritten Dropdownbox *x,y* ausgewählt - das bedeutet erster Wert (Spannung) auf die x-Achse und zweiter Wert (Strom) auf die y-Achse.

Nach Beendigung der Messung werden bei aktiviertem *reverse range after measurement* die Start- und Zielwerte vertauscht. Ist unter *repeat measurement* eine Zahl größer als eins eingetragen, so wird die Messung die angegebene Anzahl mal wiederholt, bei aktiviertem *reverse range after measurement* abwechselnd vor- und rückwärts.

4.4 Komplexeres Beispiel

Abbildung 4 zeigt ein komplexeres Beispiel. Hier werden für 27 verschiedene Spannungen am Keithley 2400 jeweils die Leistung eines Signalgenerators variiert. Bei einer Spannung von -1.3V am Keithley und einer festen Frequenz von 1986MHz wird die Leistung des Signalgenerators in 53 Schritten von -10dBm bis 16dBm erhöht. Bei jedem Schritt werden 3 Auslesevorgänge aller Messgeräte vorgenommen. Geplottet wird auf

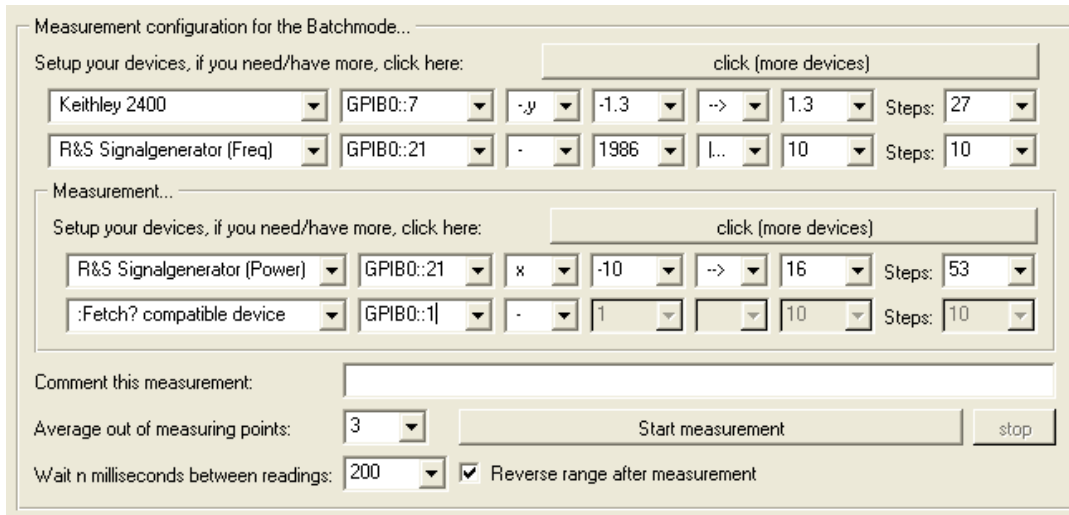


Abbildung 4: Komplexeres Beispiel zur Steuerung verschiedener Messgeräte.

die x-Achse die Leistung des Signalgenerators und auf die y-Achse der zweite Wert des Keithley 2400, der Strom. Anschließend wird die Spannung auf dem 2400er um einen Schritt erhöht und die Messung wiederholt. Dies geschieht solange, bis der Endwert erreicht ist.

4.5 Graph

Bei einer laufenden Messung werden die beiden ausgewählten Variablen (x,y) geplottet. Weiter wird auch der vorherige Graph grau angezeigt und wahlweise ein zuvor festgehaltener gelb. Diese lassen sich ein- bzw. ausblenden, indem man auf den Legendeneintrag klickt. Der aktuelle Graph lässt sich automatisch skalieren. Alternativ kann man die Maus hierfür benutzen. Unter dem Graph werden die aktuellen Zahlenwerte aufgelistet.

4.6 Speichern

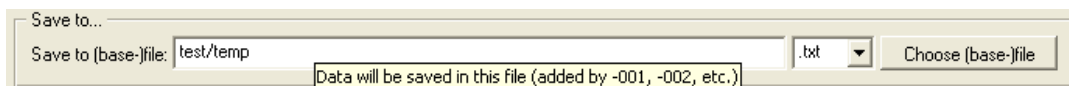


Abbildung 5: Konfiguration zum Speichern der Messdaten.

Jede Messung wird in einer eigenen CVS-Datei gespeichert, wobei der eingegebene Dateiname um eine laufende Nummer erweitert wird (siehe Abbildung 5). Neben allen

Messdaten werden auch die Geräteeinstellungen, Uhrzeit und Messkommentar gespeichert. Daneben werden Gnuplot-Dateien angelegt mit, denen die Messungen für einen ersten Überblick geplottet werden können. Gnuplot wird dabei angewiesen für jede Messung einen eigenen Plot zu erzeugen und zusätzlich einen Plot, der alle Messungen mit dem gleichen (eingeebenen) Dateinamen beinhaltet.

4.7 VisaTester

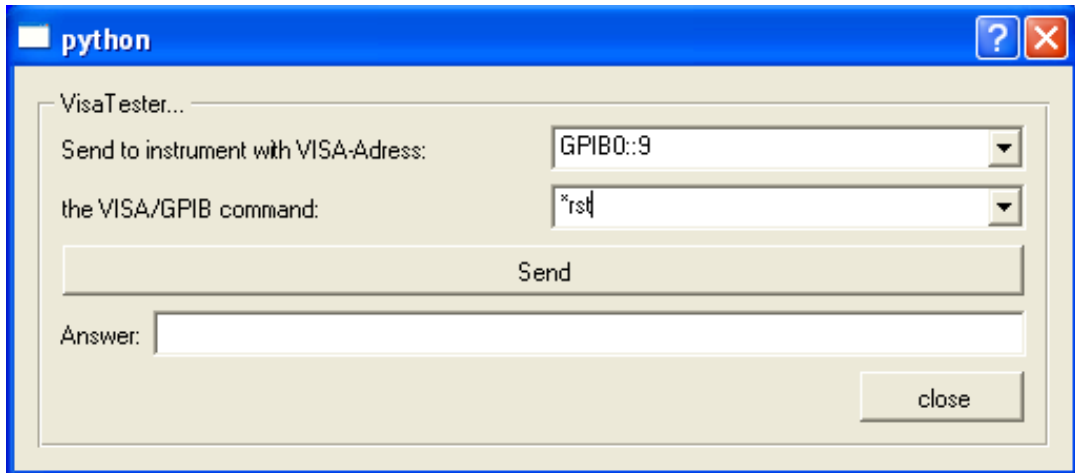


Abbildung 6: VisaTester schickt einzelne Befehle an Messgeräte.

Um einzelne Befehle an ein Gerät zu schicken, gibt es das Hilfsmittel VisaTester (siehe Abbildung 6). Dieses lässt sich über den Menüpunkt VISA aufrufen. Hier lässt sich nun die GPIB-Adresse eines Gerätes auswählen und ein dazugehöriger Befehl eingeben. Beinhaltet dieser ein Fragezeichen, so wird auch eine Antwort des Gerätes erwartet. Die jeweiligen Befehle finden sich im Handbuch des Geräts. Damit ist es zum Beispiel möglich Messgeräte zu testen oder zu reseten.

Das Keithley 4685 und 4687 benötigt nach dem Einschalten einmalig den Resetbefehl (*rst). Dieser kann über den VisaTester geschickt werden. Da hierbei die angelegte Spannung weggenommen wird, wurde darauf verzichtet diese Geräte automatisch zu reseten.

5 Messgeräte hinzufügen

Alle Informationen über die unterstützten Messgeräte sind in der Datei *devicesconfig.py* gespeichert. Um ein Weiteres hinzuzufügen muss man dessen Eigenschaften in

dieser Datei speichern. Hierfür legt man ein neues Klassenobjekt an. Am einfachsten kopiert man den Code von einem ähnlichen Gerät und modifiziert dieses dann.

5.1 Einbinden einfacher Geräte

PyvLab unterscheidet verschiedene Typen von Instrumenten (gespeichert in der Variable *self.typ*), siehe Tabelle 3.

Tabelle 3: Typen von Instrumenten die pyvLab unterscheidet

Typ	Beschreibung
w	Geräte, welche nur gesetzt werden (z.B. Signalgeneratoren)
r	Geräte, die nur ausgelesen werden (z.B. Keithley 2000)
rw	Geräte, an die Befehle geschickt und auch ausgelesen werden (z.B. Keithley 2400)
v	virtuelle Geräte

Nach dem Kopieren eines Geräts gleichen Typs muss dessen Code angepasst werden. Dies bedeutet Variablen an das neue Gerät anzupassen. Tabelle 4 zeigt die hierfür relevanten.

Tabelle 4: Wichtige Variablen einer Geräteklasse

Variable	Beschreibung
typ	Siehe Tabelle 3
classname	Klassenname, muss identisch mit dem echten Klassennamen sein und darf keine Leer- oder Sonderzeichen enthalten
init	Liste der Befehle zum Initialisieren des Geräts vor dem Start einer Messung
finish	Liste der Befehle, die nach Beendigung einer Messung aufgerufen werden
name	Name, unter dem das Gerät in der Dropdownliste erscheint
resultform	Beschreibung der Rückgabewerte des Instruments
description	Beschreibung des Geräts bzw. dessen Konfiguration
canAveraging	True oder False, je nachdem ob es Sinn macht das Gerät öfters abzufragen und dann zu mitteln

Die Variablen *init* und *finish* erwarten eine Python-Liste von Befehlen. Für den Fall, dass nur ein Befehl gesendet werden soll, wird ein Komma benutzt um Python zu zeigen, dass der Befehl in einer Liste gespeichert werden soll (z.B.: *self.init=(":init",)*). Falls kein Befehl geschickt werden soll, lässt sich dies so angeben: *self.finish=()*. Beim

Stoppen einer Messung werden diese Befehle nicht gesendet.

Anschließend muss noch die Funktion *read* angepasst werden. Zum Auslesen wird diese aufgerufen und erwartet als Rückgabe eine Zeichenkette oder eine Liste an Abfragebefehlen. Hierbei muss das Gerät aber auf jeden Befehl antworten.

Falls das Gerät auch gesetzt werden soll (Typ *w* oder *rw*), muss dies noch konfiguriert werden. Hierfür wird eine Funktion mit der Bezeichnung *getStep* benötigt. Diese liefert in der Variable *value* den zu setzenden Wert mit und erwartet als Rückgabe die passenden GPIB-Befehle als Liste (z.B. ("*SOUR:VOLT "+str(value),*)"). In der Tabelle 5 sind die Funktionen nochmals schematisch aufgeführt.

Für die meisten Geräte ist man nun fast fertig. Das Gerät muss nur noch in die Liste der vorhandenen Geräte aufgenommen werden. Hierfür muss der Klassenname in die Variable *self.devices* aufgenommen werden (am Anfang von *devicesconfig.py*).

5.2 Beispiel

Anhand der Implementierung für das Keithley 2400 soll die Vorgehensweise veranschaulicht werden.

```
#More complex device. Set a source voltage, read current.
#The data read out (e.g. "1.0,0.54365A") is splitted by the getdata-function (of abstractDevice) automatically
class keithley2400(abstractDevice):
    def __init__(self):
        abstractDevice.__init__(self)
        self.typ="rw" #Device reads and writes
        self.classname="keithley2400" #:SENS:CURR:PROT 0.000001",
        self.init=(":SOUR:FUNC VOLT",":SOUR:CLE:AUTO OFF",[gekürzt],":OUTP:STAT ON")
        self.finish={}
        self.name="Keithley 2400"
        self.resultform="voltage,current"
        self.description="Sets source voltage, reads current"
        self.canAveraging=True

    #Change source voltage to value
    def getStep(self,value):
        s=(":SOUR:VOLT "+str(value),)
        return s

    def read(self):
        return ":read?"
```

Abbildung 7: Code zur Implementation eines Keithley 2400.

Im Folgenden soll der Code zur Implementation eines Keithley 2400 beschrieben werden (siehe Abbildung 7). Die ersten beiden Zeilen sind Kommentare. Diese beginnen in Python immer mit dem Rautenzeichen. Danach beginnt die Klasse *keithley2400*, die alle Eigenschaften von *abstractDevice* erbt. In Python gibt es kein *begin* oder geschweifte Klammern um zusammenhängende Blöcke zu kennzeichnen, stattdessen wird einfach eingerückt. Als erste Funktion folgt der Konstruktor (*init*). Er wird als erstes beim Erzeugen der Klasse aufgerufen. Dort wird zuerst der Konstruktor der Eltern-

Tabelle 5: Übersicht wichtiger Funktionen einer Geräteklasse

getStep(value)

Funktion wird aufgerufen um die Befehle zum Setzen des Gerätes auf einen Wert zu erhalten.

value: Integer, hält den aktuell zu setzenden Schrittwert an;
Rückgabvariable: Liste an Strings, welche alle Befehle beinhaltet um das Gerät auf den aktuellen Schrittwert zu setzen;

read()

Funktion wird aufgerufen um die Befehle zum Auslesen des Gerätes zu erhalten.

Rückgabvariable: Liste an Strings, welche alle Befehle beinhaltet um das Gerät auszulesen;
alternativ: String, welcher den Befehl beinhaltet um das Gerät auszulesen;

isDataOk(data)

Funktion wird nach dem Auslesen aufgerufen um der Geräteklasse die Möglichkeit zu geben den kompletten Datensatz (aller ausgelesenen Geräte) zu verwerfen. Sinnvoll zum Beispiel, falls Soll- und Ist-Temperatur eines Heizers zu stark voneinander abweichen. Wird 0 (=Erfolg) zurückgegeben, so wird die Antwort in der Listenvariable *rawdata* angehängt.

data: String, welcher die Antwort des Gerätes nach einmaligem Auslesen (aller) Auslesebefehle (dieses Gerätes) enthält;
Rückgabvariable: Integer, 0 für alles OK (Standard), ansonsten eine Wartezeit in Millisekunden, nach der der Auslesevorgang wiederholt werden soll.

getData()

Diese Funktion konvertiert die Antworten des Messgeräts in eine verschachtelte Liste, welche pyvLab interpretieren kann (siehe Abbildung 8). Diese Antworten (gespeichert in *rawdata*) müssen nun verarbeitet werden. Standardmäßig kann mit vielen Geräten umgegangen werden, bei speziellen Geräten muss jedoch eine Anpassung erfolgen. Details hierzu sind im Text verfasst.

Rückgabvariable: verschachtelte Liste, siehe Abbildung 8.

klasse aufgerufen, der alle Variablen auf Standardwerte setzt. Die davon abweichenden werden daher im Folgenden neu gesetzt. Darunter der *typ* auf *rw* und den Klassennamen *classname* auf den richtigen Klassennamen (*keithley2400*). Ansonsten werden noch die Initialisierungsbefehle aus dem Handbuch eingetragen und die Beschreibungstexte gesetzt.

Anschließend folgt die Funktion zum Setzen der Spannung. Diese liefert in *value* den numerischen Wert und gibt die Befehlszeichenkette zurück. Der Befehl zum Auslesen ist der Rückgabewert der letzten Funktion (*read*).

5.3 Einbinden komplexer Geräte

Um komplexere Geräte einbinden zu können folgt nun eine etwas technischere Beschreibung der Funktionsweise. Im Anschluss wird diese ebenfalls an zwei Beispielen erläutert. Die Elternklasse *abstractDevice* jedes Geräts besitzt noch weitere Funktionen, die je nach Gerät sinnvollerweise durch angepassten Code ersetzt werden. Dies betrifft insbesondere die Funktionen *getData* und *isDataOK*. Letztere gibt einer Geräteklasse die Möglichkeit anhand ihrer Messdaten den Auslesevorgang für alle Messgeräte wiederholen zu lassen. Erstere Funktion (*getData*) dient der Konvertierung der Antworten eines Geräts zu einer Liste mit numerischen Messdaten. Falls ein Gerät mehrmals ausgelesen wurde und darüber gemittelt werden soll, muss der Mittelwert ebenfalls hier berechnet werden.

Nachdem *pyvLab* den Abfragebefehl von *read* erhalten hat, schickt es ihn an das entsprechende Gerät. Die Antwort wird als ein Element in der Listenvariable *rawdata* der jeweiligen Geräteklasse festgehalten. Bei mehr als einem Abfragebefehl werden die Geräteantworten, getrennt mit *,*, zu einer Zeichenkette zusammengefügt und ebenfalls als ein einzelnes Element gespeichert. Somit ist die Anzahl, über die gemittelt werden soll nach dem Auslesen identisch mit der Elementzahl von *rawdata*, sofern die Variable *canAveraging* dieser Klasse *True* ist.

Anschließend wird die Funktion *isDataOK* aufgerufen. Hier lässt sich zum Beispiel für einen Heizer die IST- und SOLL-Temperatur vergleichen und bei zu großem Abstand eine Wartezeit zurückgeben. Für diesen Fall werden die gelesenen Daten aller Messinstrumente verworfen und nach einer gewünschten Wartezeit wird erneut gemessen. Als Parameter wird die Zeichenkette der Abfragebefehle eines Auslesevorgangs mitgegeben. Wenn die Daten in Ordnung sind, wird Null als Rückgabewert erwartet, andernfalls die Wartezeit in Millisekunden, bis der Auslesevorgang wiederholt werden soll. Standardmäßig wird immer Null zurückgegeben.

Nun wird *getData* aufgerufen. Diese muss die Ausgabe des Messgerätes in eine Liste der gemessenen Werte konvertieren. Im einfachsten Fall ist dies nur ein Zahlenwert.

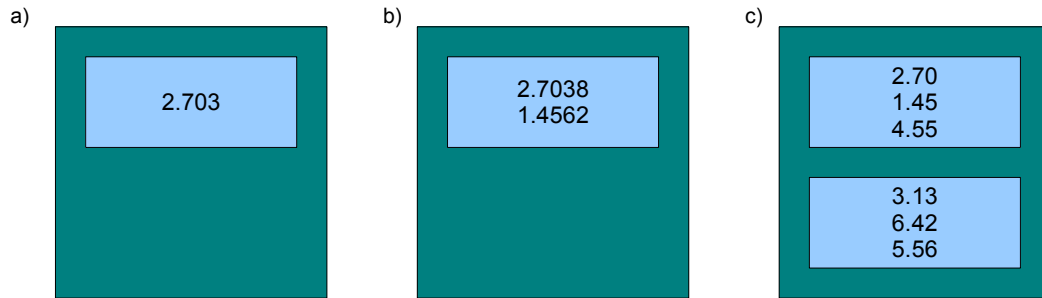


Abbildung 8: Schema der Rückgabewerte der Funktion `getData`. a) zeigt den einfachsten Fall: ein Messgerät gibt einen Wert zurück. Dieser wird in eine Liste gepackt, diese wiederum in eine Liste. In b) gibt das Instrument zwei Werte zurück, wie z.B. Spannung und Strom; diese werden in eine Liste gepackt und wie in a) wiederum in eine Liste; Bei c) gibt das Messgerät verschiedene Sätze von Messdaten zurück, z.B. ein ganzes Spektrum; die Werte zu einer Frequenz werden jeweils in einer Liste gespeichert und alle Listen in die übergeordnete Liste eingefügt;

Dieser wird in einer Liste mit nur diesem Zahlenwert als Element gespeichert. Diese Einelementliste wird wiederum in einer übergeordneten Liste gespeichert und ist der Rückgabewert der Funktion (siehe Abbildung 8a). In einem komplexeren Fall, wie dem Spektrum eines Netzwerkanalysators, sieht es folgendermaßen aus: für jede Frequenz gibt er den dazugehörigen Real- und Imaginärwert zurück. Diese Werte werden nun für jede Frequenz in einer Liste gespeichert. Die übergeordnete Liste enthält nun alle Listen für die einzelnen Frequenzen (siehe Abbildung 8c).

Standardmäßig wird `getData` der Elternklasse aufgerufen. Diese kann meist mit den Fällen a) und b) der Abbildung 8 umgehen. Genauer gesagt trennt die Hilfsfunktion `splitData` die Daten auf und `getData` braucht nur noch deren Rückgabewert als einelementige Liste weiter zurückgeben.

`splitData` geht dabei folgendermaßen vor: zuerst trennt sie die einzelnen Antworten des Geräts (`rawdata`). Dann trennt sie die separierten Antworten an einem eventuell vorhandenen Komma oder Semikolon. Nach dem Entfernen eventueller Buchstaben enthält sie dann anschaulich folgenden Datensatz:

Antwort-1-Zahl-1, Antwort-1-Zahl-2, ... , Antwort-1-Zahl-n, Antwort-2-Zahl-1, Antwort-2-Zahl-2, ... , Antwort-2-Zahl-n, ... , Antwort-m-Zahl-1, Antwort-m-Zahl-2, ... , Antwort-m-Zahl-n

m ist hierbei die Anzahl der Auslesevorgänge (über die gemittelt werden soll) und n ist die Anzahl der Rückgabewerte des Geräts (z.B. Strom und Spannung). Anschließend mittelt sie über die jeweils zusammengehörigen Werte, also z.B. *Antwort-1-Zahl-1, Antwort-2-Zahl-1, ... , Antwort-m-Zahl-1*.

Die meisten Messgeräte geben die Daten in einer für *splitData* verständlichen Form zurück. Wird jedoch z.B. ein ganzes Spektrum zurückgegeben, d.h. viele einzelne Wertepaare, so muss *getData* überschrieben werden und die Messdaten müssen mittels eigenem Code konvertiert werden (siehe Kapitel 5.5). Oft ist dies jedoch nicht nötig, sondern es reichen kleine Anpassungen, wie z.B. beim Lock-In Stanford SR510 (siehe *deviceconfig.py*). Dort sind zwei Auslesebefehle nötig, da zwei Messgrößen von Interesse sind (Amplitude und Phase) und beide nicht zusammen abgefragt werden können. Die Antworten des Geräts werden in einem *rawdata*-Element gespeichert. Um die Antworten des Geräts unterscheidbar zu machen, werden diese automatisch mit einem Doppelkomma separiert (,,). In unserem Fall würde das erste Element von *rawdata* z.B. folgenden String beinhalten: *A 1.34,,P 34.7*. Hier ist das Doppelkomma ausnahmsweise unerwünscht, denn standardmäßig würde *splitData* eine Null zwischen die beiden Werte einfügen (*1.34,0,34.7*). Aus diesem Grund überschreibt man die *getData*-funktion, ersetzt in jedem Element das Doppelkomma durch ein einfaches und ruft dann die Hilfsfunktion *splitData* auf.

Alle wichtigen Funktionen sind in der Tabelle 5 nochmals aufgeführt. Zwei Beispiele veranschaulichen diese nun:

5.4 Heizer

```
#Test if Data is OK, e.g if heater temperature already reached,
#if not a new reading will be made and asked again
def isDataOK(self,data):
    #print "New Data: ",data," Steppos: ",self.steppos
    b=data.split(",")
    temp2=b[1].strip()
    temp2=temp2[:-2]
    diff=abs(float(temp2)-self.steppos)
    print "Diff: ",diff," Temp2: ",temp2
    #Weniger als 10% Abweichung?
    if diff<0.1*self.steppos:
        return 0
    else:
        return 10000
    #return 0
```

Abbildung 9: Code zur Kontrolle der IST- und SOLL-Temperatur eines Heizers.

Hier soll das Verhalten von *isDataOK* behandelt werden, siehe Abbildung 9. Der Heizer (Neocera Temperature Controller) unterstützt zwei Temperatursensoren. Einer befindet sich im Innenraum eines Kryostaten, der zweite an der zu messenden Probe. Beide Temperaturen werden abgefragt ("*QSAMP?1;*", "*QSAMP?2;*"). Relevant für den Temperaturvergleich ist nur die Temperatur an der Probe. Die Variable *data* hat zum

Beispiel folgenden Inhalt: " 4.034K;,, 4.467K;". Mittels der beiden Kommas werden die beiden Temperaturen wieder getrennt. Der zweite beinhaltet nun die Probertemperatur. Nun werden die Leerzeichen und die beiden Zeichen am Schluss entfernt. Anschließend wird diese Temperatur mit der Solltemperatur, gespeichert in *steppos*, verglichen. Bei Übereinstimmung im gewählten Rahmen wird Null zurückgegeben, andernfalls eine Wartezeit von 10 Sekunden.

5.5 Spektrum

```
#Here we get not just one measurement point, but the whole spectrum
#So we have to split the data into a list with all data points
def getData(self):
    if len(self.rawdata[0])==0:
        print "deviceConfig: ERROR: not data (in networkanalyzerspectrum getData)"
        return False

    #canAverage is false, so just Data in rawdata[0]
    #Y-Values then ,, then X-Values
    a=self.rawdata[0] # "1,2,3,,4,5,6"

    #Two read commands are sperated by ,,
    b=a.split(",,") # "(1,2,3)","(4,5,6)"

    #Split x measurement points (frequencies)
    xaxis=b[1].split(",") #X-measure points
    tmp=b[0].split(',') # Real and imaginary points

    i=0
    ix=0
    data=[]
    while i+1<len(tmp):
        #calculate of real and imaginary part magnitude, phase etc...
        try:
            magn=20*math.log10(math.hypot(float(tmp[i+1]),float(tmp[i])))
            phase=math.tan(float(tmp[i+1])/float(tmp[i]))
        except:
            magn=0.0
            phase=0.0

        res=[]
        #append results to ONE data set
        res.append(str(xaxis[ix]))
        res.append(str(magn))
        res.append(str(phase))
        res.append(str(tmp[i+0])) #real
        res.append(str(tmp[i+1])) #imaginary

        data.append( res )
        i+=2
        ix+=1

    #Clear list of rawdata
    self.rawdata=[]
    return data # List with data sets!
```

Abbildung 10: Code, der die Antwort des R&S Netzwerkanalysators ZVC konvertiert.

Im Folgenden Beispiel wird die Konvertierung des Spektrums eines Netzwerkanalysators erklärt, siehe Abbildung 10. Mittels zweier Befehle werden die Daten abgefragt. Der erste gibt eine kommasetrennte Liste der Frequenzen zurück und der zweite eine ebenfalls kommasetrennte Liste der Real- und Imaginärteile, d.h. diese enthält doppelt soviele Werte wie die erste. Da Mitteln hier keinen Sinn macht, enthält *rawdata* nur ein Element. Zuerst werden in diesem ersten Element die Antworten des Instruments getrennt, anhand des doppelten Kommas. Anschließend werden zwei Listen erstellt, *xaxis* enthält die Frequenzen und *tmp* die Real- und Imaginärteile. Für jede Frequenz wird nun der Betrag und die Phase berechnet und alle Werte in einer Liste (*res*) gespeichert. Diese wird bei jeder Frequenz in die übergeordnete Liste *data* kopiert. Zuletzt wird noch *self.rawdata* geleert, damit sie mit neuen Daten gefüllt wird.

5.6 pseudoVisa

Zum Testen enthält pyvLab eine pseudoVisa genannte Schnittstelle. Um diese zu aktivieren kommentiert man die Zeile *from visa import ** in *visacontrol.py* und *visates-ter.py* mittels Raute aus und aktiviert stattdessen die Zeile *from pseudovisa import ** in den beiden Dateien. Nun werden alle Befehle, die an Messgeräte gesendet würden, stattdessen in der Konsole angezeigt und Zufallszahlen zurückgeschickt. Für ein spezielleres Verhalten lässt sich die Datei *pseudovisa.py* selbstklärend anpassen.