

## **Übung “Pervasive and Embedded Systems”, 2013S**

### **PROJECT SIGNALVERARBEITUNG (AUDIO-, IMAGE), I/O, FILTER**

**Dipl.-Ing. Dr. Andreas Riener, Dipl.-Ing. Gerold Hölzl**  
Johannes Kepler University Linz  
Institute for Pervasive Computing  
Altenberger Strasse 69, A-4040 Linz



## Embedded System 2013S: Schedule (tentative, as of April 16, 2013)

Lecture

No.	Date	Lecture contents	Exercise contents	Hand-out	Return
1	05. 03. '13	Introduction	Intro/Organizational		
2	12. 03. '13	Sensors I	Phidgets (Sensors I)	PE1/PE2	
3	19. 03. '13	Sensors II	Phidgets (Sensors II)	PE2/PE1	
	<b>26. 03. '13</b>	***** <b>No classes (Easter break)</b> *****	*****		PE1+PE2
	<b>02. 04. '13</b>	***** <b>No classes (Easter break)</b> *****	*****		
4	09. 04. '13	Sensors III	Phidgets (Sensors III)	PE3/PE4	
5	16. 04. '13	Communic., Positioning, Actuators	Phidgets (Sensors IV)	PE4/PE3	
6	23. 04. '13	Dependability	Project Introduction	<b>PR</b>	PE3+PE4
7	30. 04. '13	Time + models, clock, clock sync.	-	-	-
8	07. 05. '13	Model-based design (discrete, cont.)	Dependability	TA1	-
9	14. 05. '13	Concurrent models of execution	Petrinets	TA2	TA1
	<b>21. 05. '13</b>	***** <b>No classes (Pentecost)</b> *****	*****		
10	28. 05. '13	Scheduling (introduction)	-	-	TA2
11	04. 06. '13	Energy efficient design	Scheduling I	TA3	-
12	11. 06. '13	Scheduling (optimality)	Scheduling II	-	-
13	18. 06. '13	RTOS (implementation issues)	<b>Project presentation</b>		<b>PR, TA3</b>
	<b>25. 06. '13</b>	<b>Written Exam</b>			

**Red highlight: compulsory attendance!**

PR...Practical exercises, TA...Theoretical Assignments  
Each assignment counts 24 points; PR counts 3x24 points

# Signalverarbeitung :: Morsecode

Projektarbeit läuft **parallel** zum regulären Übungsbetrieb (und den theoretischen Übungen)

- Vollständiger Signalverarbeitungskreis zur Übertragung von Morsecode
- Zählt wie 3 Übungen (3x24=72 Punkte) (min **18 Punkte** müssen erreicht werden)
- Teilaufgaben
  - Morse-Codierung (24 Punkte)
  - Morse-Decodierung (16 Punkte)
  - Öffnen/Aufnehmen, Filtern, Ausgeben/Speichern (24 Punkte)
  - Vorführung/Präsentation: (8 Punkte)
- Keine Vorgabe der Entwicklungsumgebung/Programmiersprache (empf.: JAVA + Octave)

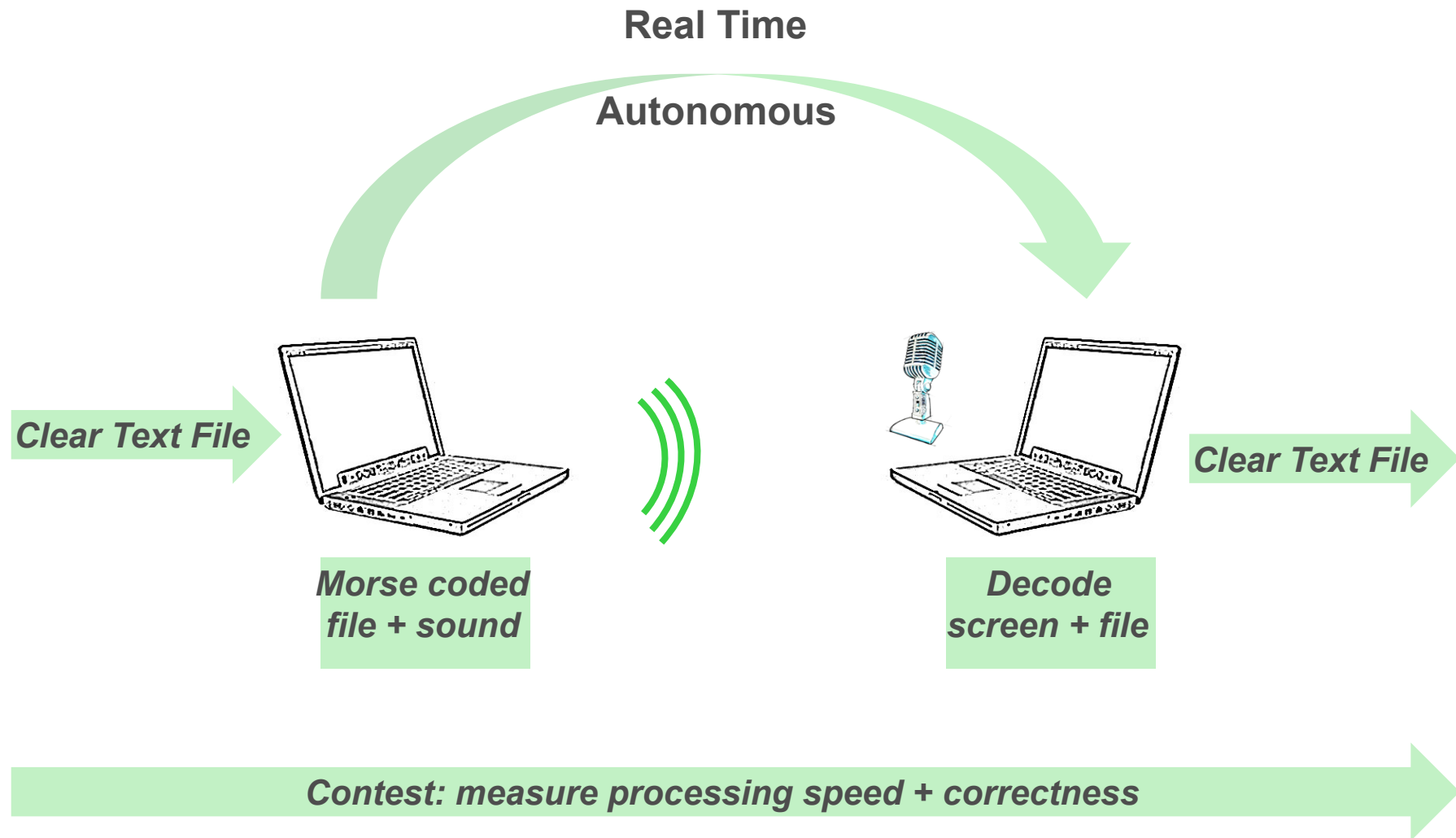
- Bearbeitbar im **2er-Team**

- Teamfindung über Forum
- bis 25.04.2013 15:00 (alle die bis zu diesem Zeitpunkt keine(n) PartnerIn gefunden haben, werden zusammengelost)
- Übungsupload nur einmal pro Gruppe

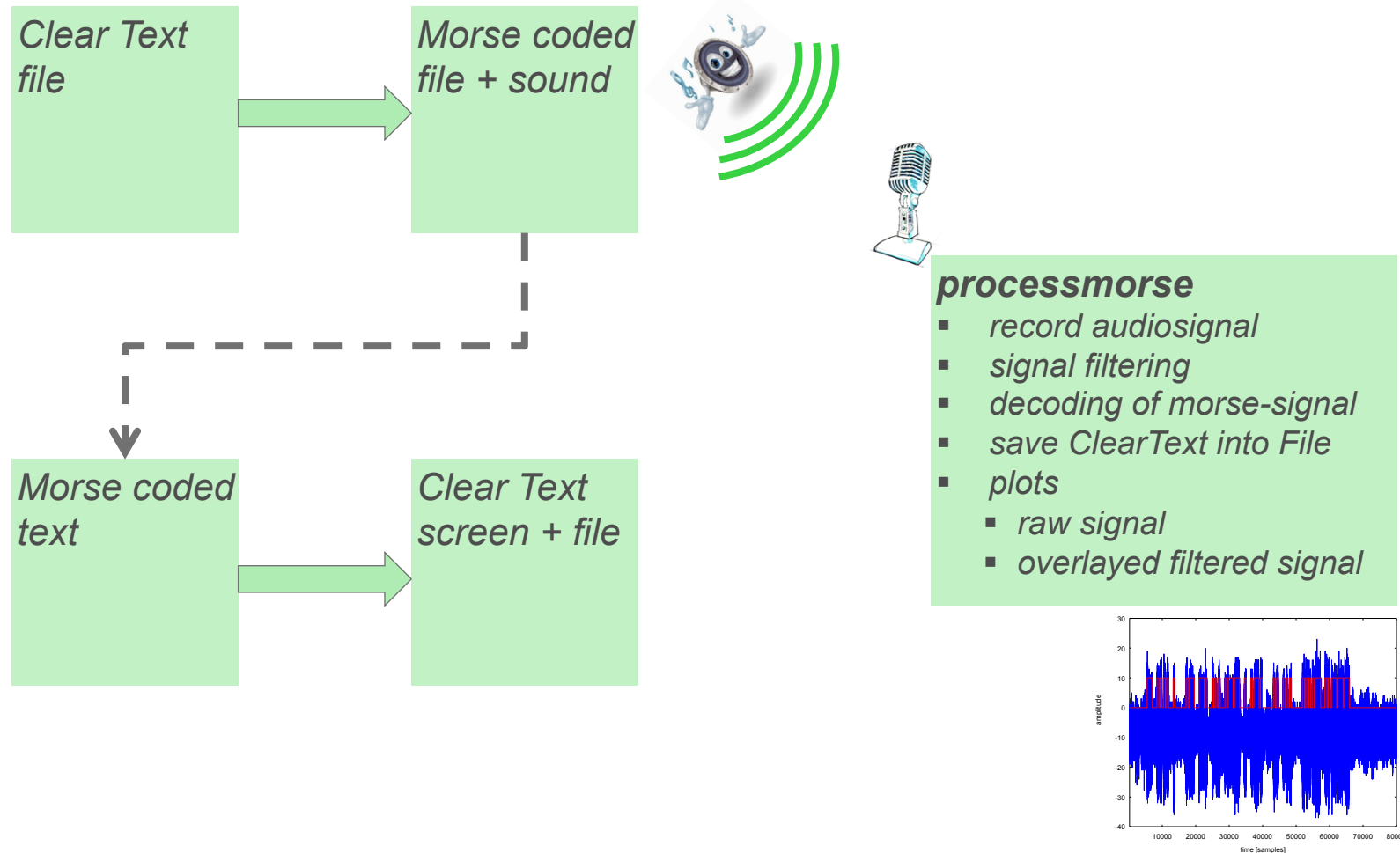
Subject	Posts	Started By	Last Post
► Teamfindung für PROJEKT Delete Thread • Move Thread • Merge Thread	1	hoelzl	04/19/2013 08:07AM Last Post by hoelzl

- Präsentation mit Livevorführung aller Teams **am Di. 18.06.2013**
  - **HS 19, 17.15 – 19.45 Uhr, gemeinsam für alle Übungsgruppen**
  - Contest/challenge: **Schnellste fehlerfreie Lösung gesucht**

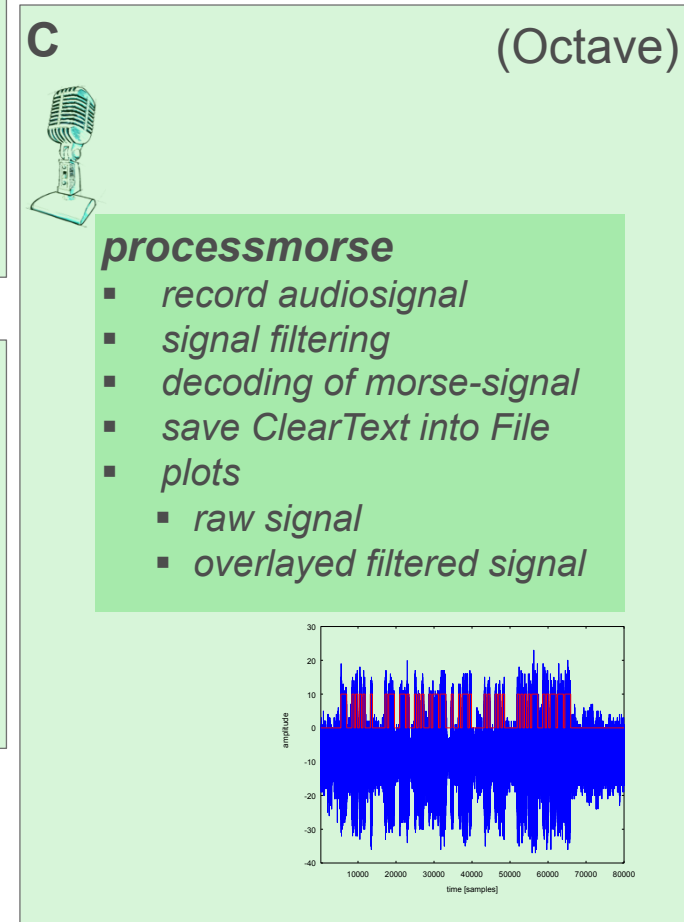
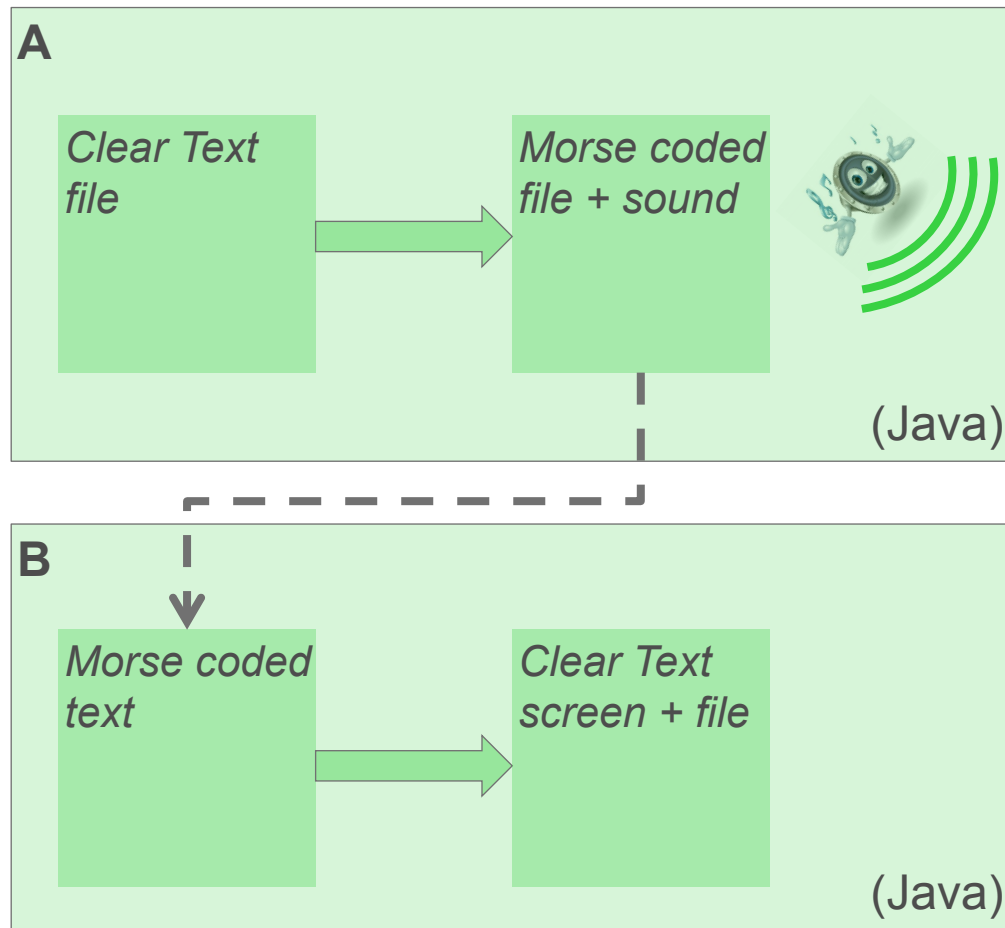
# Signalverarbeitung :: Morsecode :: Challenge



# Signalverarbeitung :: Architecture



# Signalverarbeitung :: Architecture



# MORSECODIERUNG: Allgemeines

## Charakteristik

- Ein- und Ausschalten eines konstanten Signals
- Beliebiges Medium mit dem zwei verschiedene Zustände eindeutig und in der zeitlichen Länge variierbar dargestellt werden können, z.B.
  - Tonsignal (Ton, kein Ton)
  - Funksignal
  - Elektrischer Puls
  - Mechanisch
  - Optisch (blinkendes Licht)
- "Erstversion" 1833 durch *Samuel Morse* (Schreibtelegraf)
- Standardisierung auf dem *Internationalen Telegraphenkongress* (Paris, 1865); später Normierung durch die ITU (*Internationale Fernmeldeunion*)

# MORSECODIERUNG: Allgemeines (2)

## Vorteile

- Basiert auf einfachem, unmodulierten Signal
  - Vorteil: Benötigt niedrige Bandbreite bzw. Sendeleistung
- Geringe Anforderungen an Hardware zum Senden und Empfangen
- "Robustes" Verfahren (arbeitet auch bei schlechtem Signal-Rausch-Verhältnis noch sehr zuverlässig – siehe Übungsblatt!)

## Zeitliches Verhalten

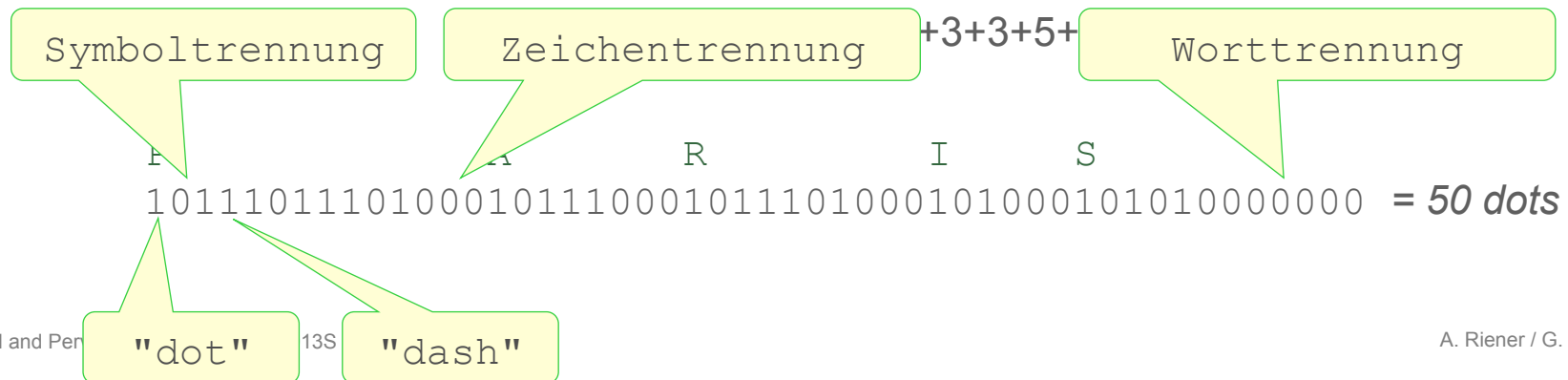
- Es gibt "nur" drei unterschiedliche Symbole
  - " ."    Punkt    *Dot*    (*Dit*)
  - " – "    Strich    *Dash*    (*Dah*)
  - " "    Pause    *Gap*



## MORSECODIERUNG: Allgemeines (3)

## Zeitliches Verhalten

- Basiseinheit ist 1 Punkt (*dot*)
  - Ein Strich "–" ist 3-mal so lang wie ein Punkt "."
  - Zwischen 2 Symbolen wird eine Pause von einem *dot* (Punkt) eingelegt
  - Buchstaben in einem Wort werden durch eine Pause von einem *dash* (Strich) = 3 *dots* getrennt
  - Pause zwischen 2 Wörtern beträgt 7 *dots*
- Die Sendegeschwindigkeit wird bestimmt durch die Länge eines Punktes (*dot*, *dit*):
  - Angabe in WPM (Wörtern pro Minute), seltener auch in BPM (Buchstaben pro Minute)
  - Basis ist das Wort "PARIS" (besteht aus "kurzen" und "langen" Buchstaben)



# MORSECODIERUNG: Allgemeines (4)

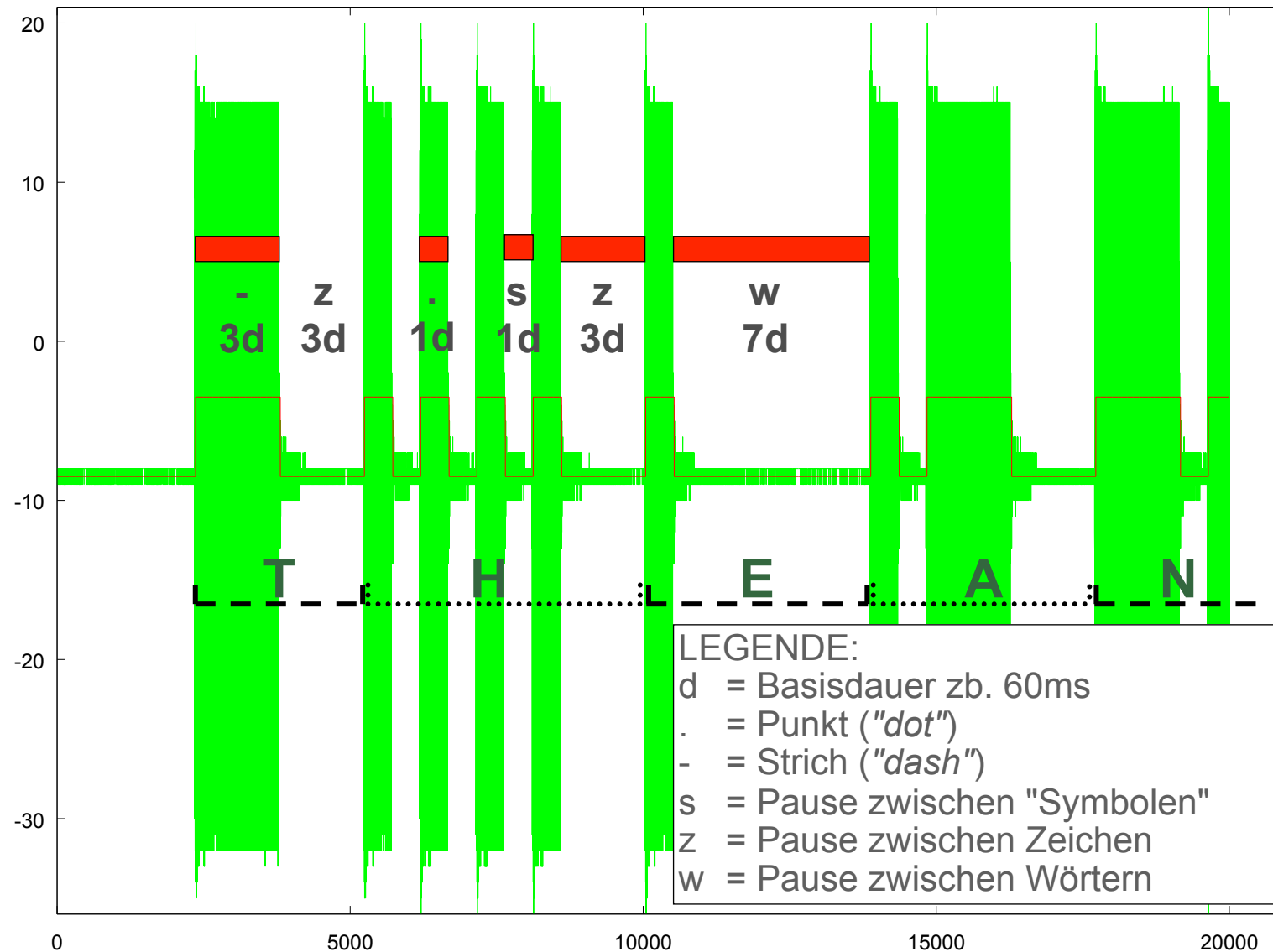
## Morsecode-Tabelle

- Keine Unterscheidung zwischen Groß- und Kleinbuchstaben
- Standardisiert für
  - Lateinische Buchstaben
  - Ziffern
  - Sonder- und Satzzeichen
  - Signale (SOS, etc.)

→

T —	M — —	O — — —	CH — — — —
			Ö — — — .
	N — .	G — — .	Q — — . —
			Z — — . .
		K — . —	Y — . — —
			C — . — .
E .	A . —	D — . .	X — . . —
			B — . . .
		W . — —	J . — — —
			P . — — .
	I . .	R . — .	Ä . — . —
			L . — . .
		U . . —	Ü . . — —
			F . . — .
		S . . .	V . . . —
			H . . . .

# SIGNALFILTERUNG: Morse-Signal in Octave filtern



# TÖNE IN JAVA: Ausgabe eines 2kHz-Sinussignales

```
import javax.sound.sampled.*;

public class Tone {
    public static void main(String[] args) {
        try {
            // AudioFormat(sample rate in Hz, sample size in bits, number of channels,
            // are samples signed?, are samples in big endian format?)
            AudioFormat format = new AudioFormat(16000, 16, 1, true, true);
            SourceDataLine line = AudioSystem.getSourceDataLine(format);
            line.open(format, 4096); // open line with 4 KB buffer
            for (int i = 0; i < 16000; i++) { // play 1 sec tone (16000 samples)
                // start to play when buffer is 3/4 filled (avoids buffer underrun)
                if (line.available() < 1024) line.start();
                // v(i) = sin( 2 * pi * frequency * i/sampleRate ) * amplitude
                short v = (short) (Math.sin(2 * Math.PI * 2000 * i / 16000) *
                    (Short.MAX_VALUE));
                // write short v in big endian format to line. blocks if buffer full
                line.write(new byte[] { (byte) (v >> 8), (byte) v }, 0, 2);
            }
            line.drain(); // wait for playback to end
            line.close(); // close line
        } catch (LineUnavailableException e) {
            System.err.println(e.getMessage());
        }
    }
}
```

16000 samples per sec,  
16 bit samples, mono channel,  
samples signed, samples in  
big endian byte order (high  
byte first)

# **OCTAVE:** Introduction, Installation Guide

**Dipl.-Ing. Dr. Andreas Riener, Dipl.-Ing. Gerold Hölzl**  
Johannes Kepler University Linz  
Institute for Pervasive Computing  
Altenberger Strasse 69, A-4040 Linz



# OCTAVE: Einführung

## ■ Was ist Octave?

- Umfangreiches **numerisches** "Mathematiksystem" (im Gegensatz zu algebraischer/symbolischer Software wie **Mathematica**)
- ähnlich zu MatLab (größtenteils Syntaxkompatibel), jedoch frei verfügbar unter GNU General Public License
- unterstützte Systeme: GNU/Linux, BSD, OS X, Windows (Cygwin, MinGW)
- schnelle Darstellung von Ergebnissen/Daten als 2D/3D Diagramm
- Kontrollkonstrukte wie in anderen Programmiersprachen möglich
- zahlreiche Erweiterungen stehen in sogenannten "Toolboxes" zur Verfügung
- besonders bei Herstellern **reaktiver Systeme** eingesetzt (zB. in der Auto-, Luft-, Raumfahrt-, Kommunikationstechnik)

## ■ Typische Anwendungsgebiete

- Technisches Rechnen
- Messtechnik
- Steuerungs- und Regelungstechnik
- Bildverarbeitung
- Signalverarbeitung und Kommunikation
- Finanzrechnung
- etc.

## OCTAVE: Windows-Installation

### ■ Für den Übungsbetrieb wird (unter Windows) benötigt

- UNIX-Umgebung **Cygwin** (POSIX-Layer) als Host-System für Octave
- **GNU-Octave** für Windows (aktuelle Version 3.6.1 vom 22.2.2012)
- **MATCOMPAT-Paket** (Matlab-Erweiterungen für Octave)
- weitere Pakete (nach Bedarf) aus **Octave-Forge** downloadbar  
(zentrales Repository für Erweiterungen für GNU Octave)

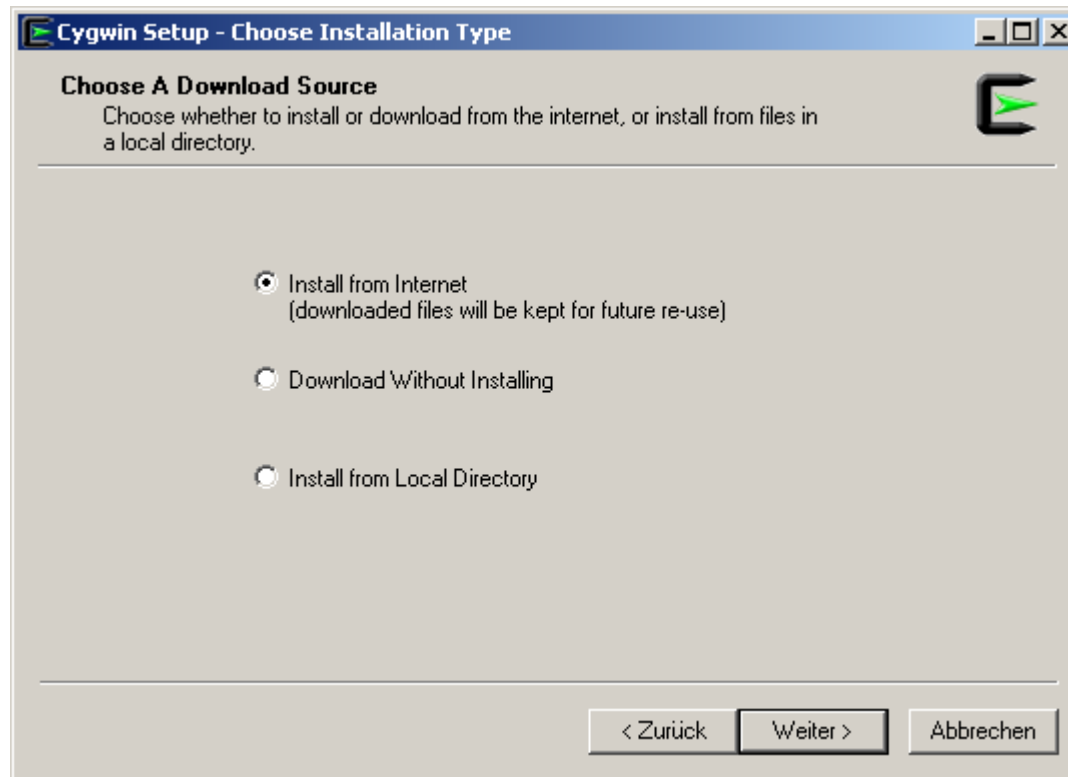
### ■ Link-Sammlung

- <http://cygwin.com/> (Cygwin Home)
- <http://www.gnu.org/software/octave/> (Octave Home)
- <ftp://ftp.octave.org/pub/gnu/octave/> (Binaries, ältere Versionen, etc.)
- <http://octave.sourceforge.net/> (Windows installer, Mac OS X App.)
- [www.gnuplot.info](http://www.gnuplot.info) (Gnuplot Home)
- <http://users.powernet.co.uk/kienzle/octave/index.html>  
(Matlab/Octave compatibility packages)
- Sonstige Erweiterungen (sog. „Packages“) werden verwaltet unter  
<http://octave.sourceforge.net/packages.php>

## OCTAVE: Windows-Installation

### ■ Installation / Konfiguration von Cygwin

- Download des Installationsprogrammes (`setup.exe`, 681kB) unter <http://cygwin.com>
- Aktuelle Version 1.7.17-1 (20.10.2012)
- Installationsprogramm ausführen, Option 1 "Install from Internet" wählen

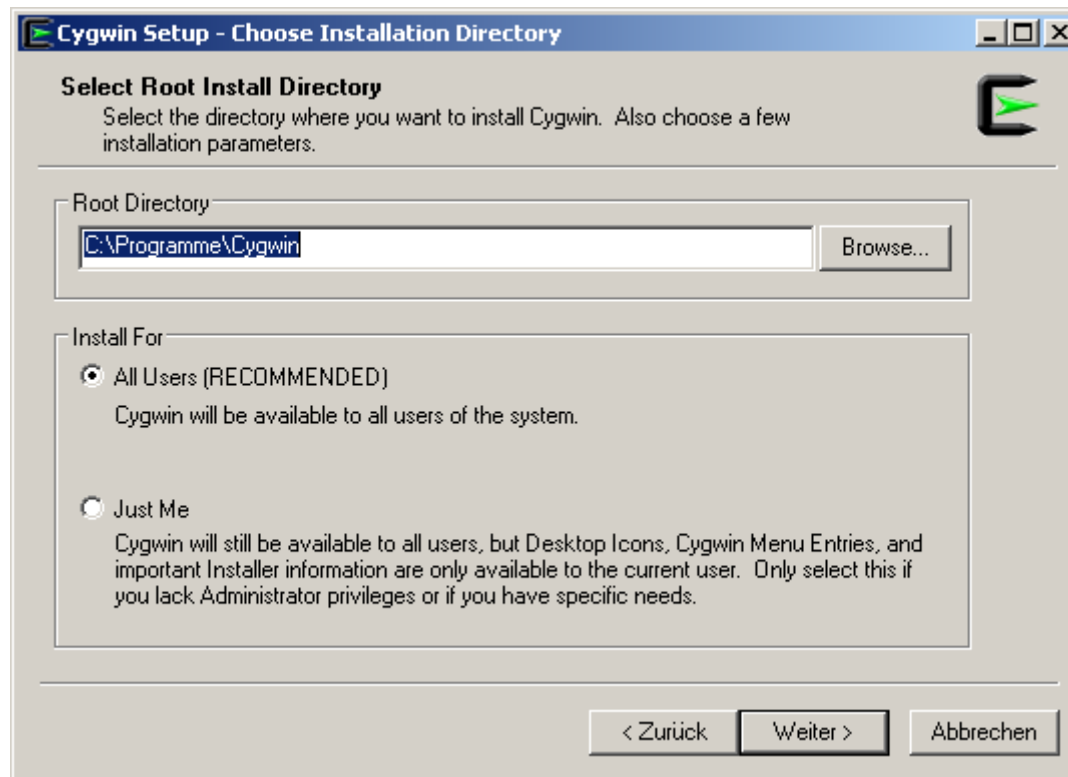




## OCTAVE: Windows-Installation

### ■ Installation / Konfiguration von Cygwin

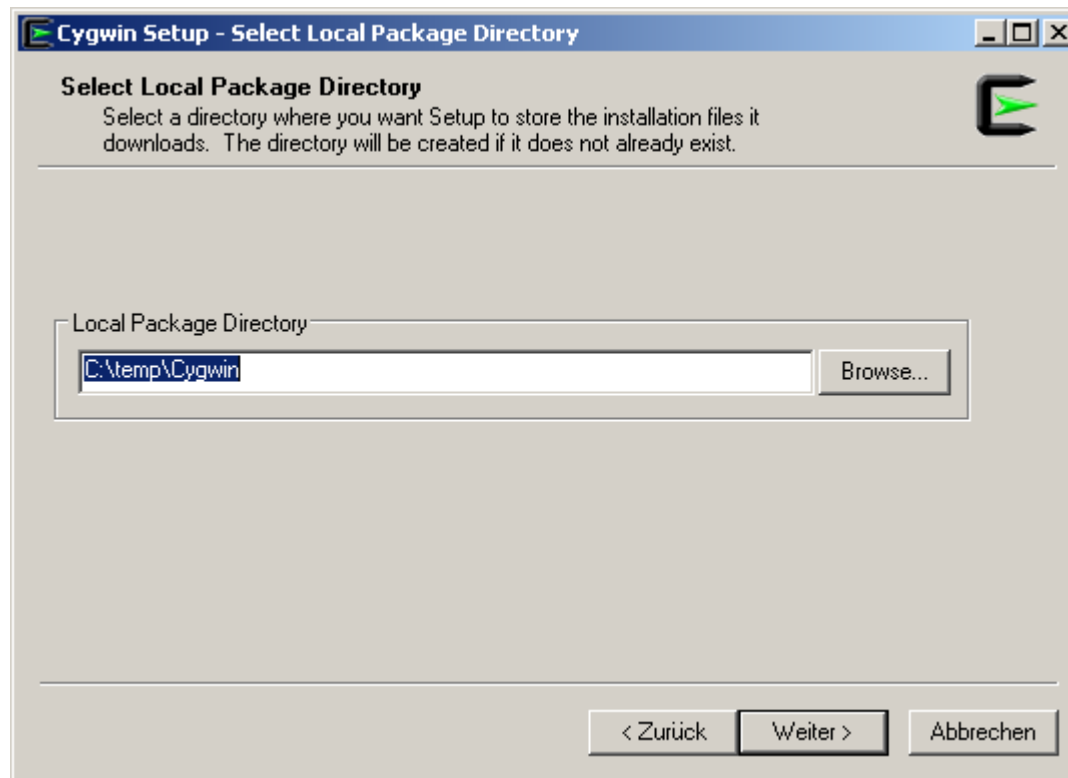
- Installationsordner wählen, zb. C:\Programme\Cygwin



## OCTAVE: Windows-Installation

### ■ Installation / Konfiguration von Cygwin

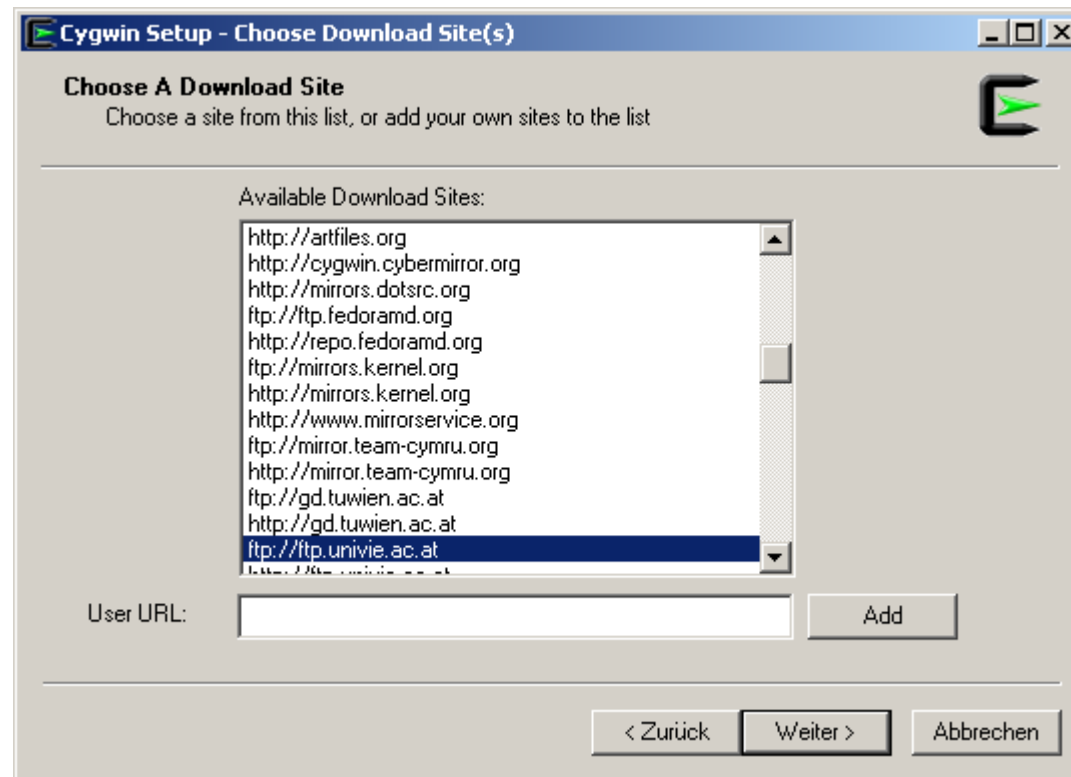
- Auswahl eines temporären Verzeichnisses für die Installationsdateien (ca. 170MB - je nach gewähltem Installationsumfang).



## OCTAVE: Windows-Installation

### ■ Installation / Konfiguration von Cygwin

- Download-Server (Mirror) auswählen, zb. `ftp://ftp.univie.ac.at`



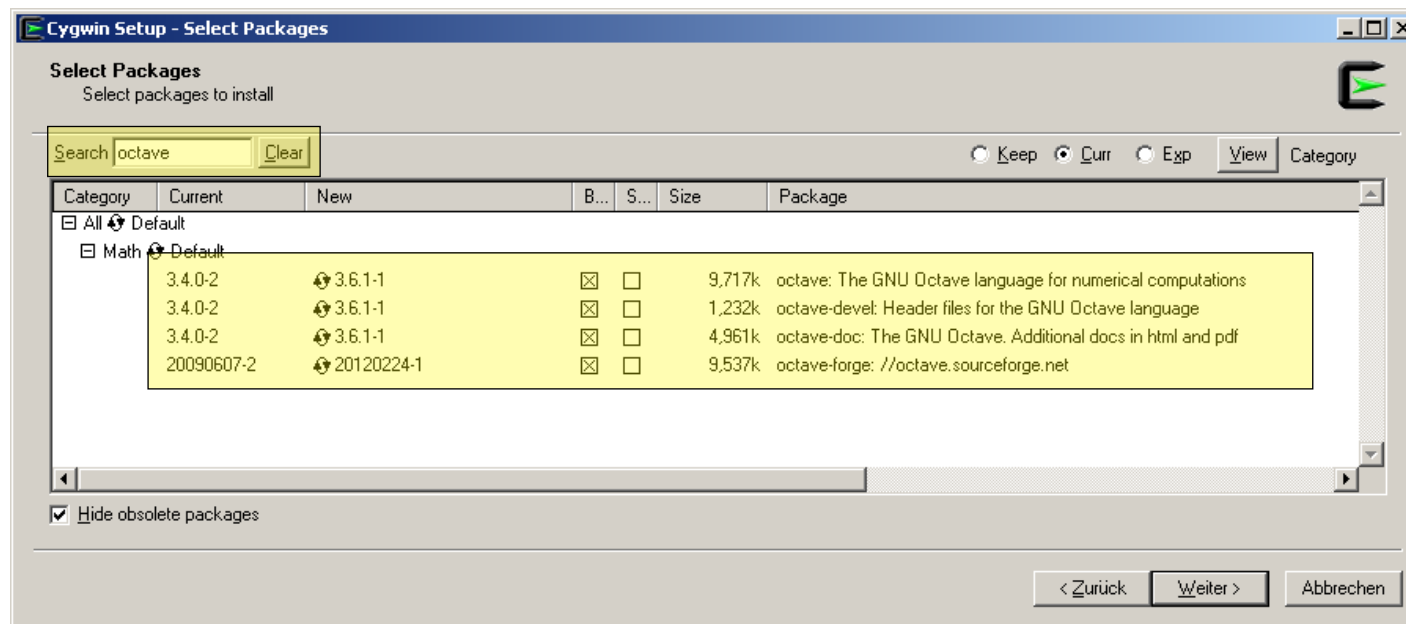
# OCTAVE: Windows-Installation

## ■ Installation / Konfiguration von Cygwin

- Auswahl der zu herunterladenden/installierenden Pakete
- Für die Übung wird empfohlen **cygwin** mit **octave** und **xserver** herunterzuladen, das sind insbesondere folgende Packages (jeweils default-Einstellung):

octave, octave-doc, octave-forge, gnuplot, xorg-server, xterm, x-start-menu-items

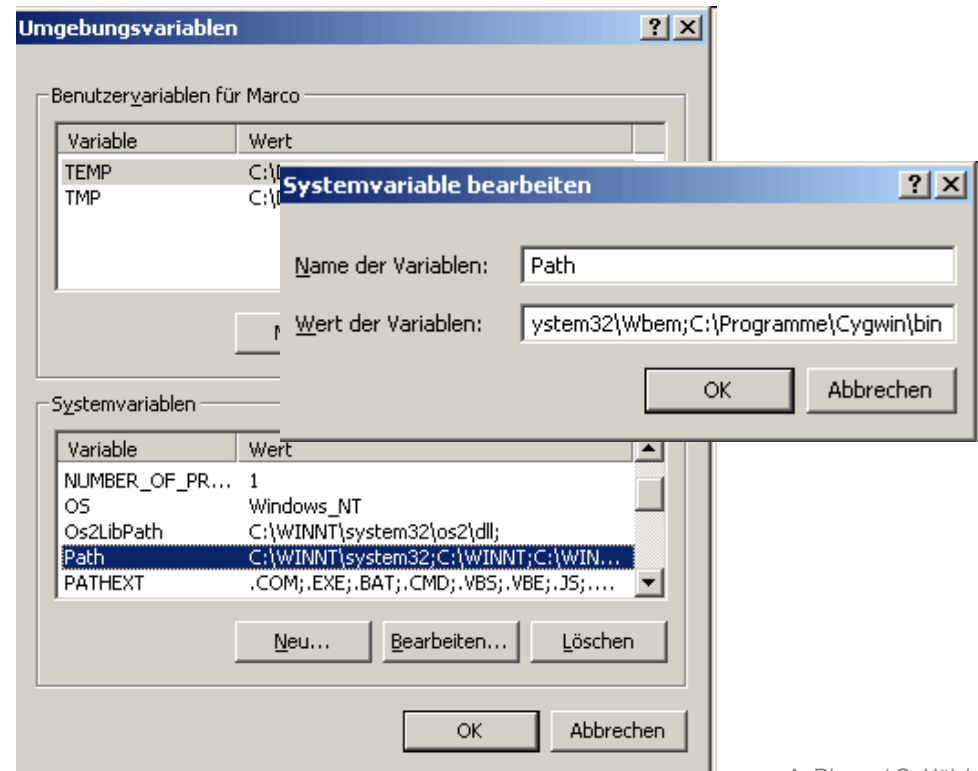
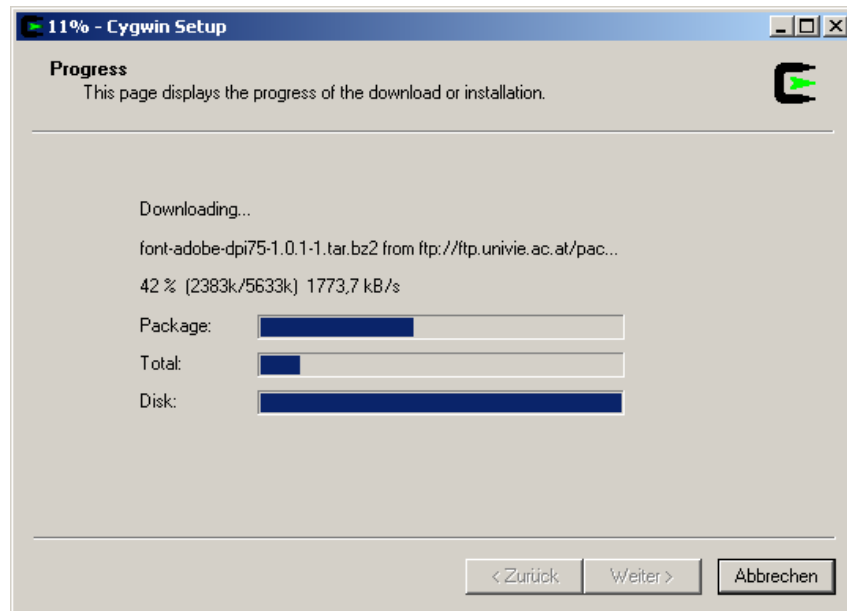
- Mit "Weiter" wird der Downloadvorgang fortgesetzt und anschließend die Installation gestartet...



# OCTAVE: Windows-Installation

## ■ Installation / Konfiguration von Cygwin

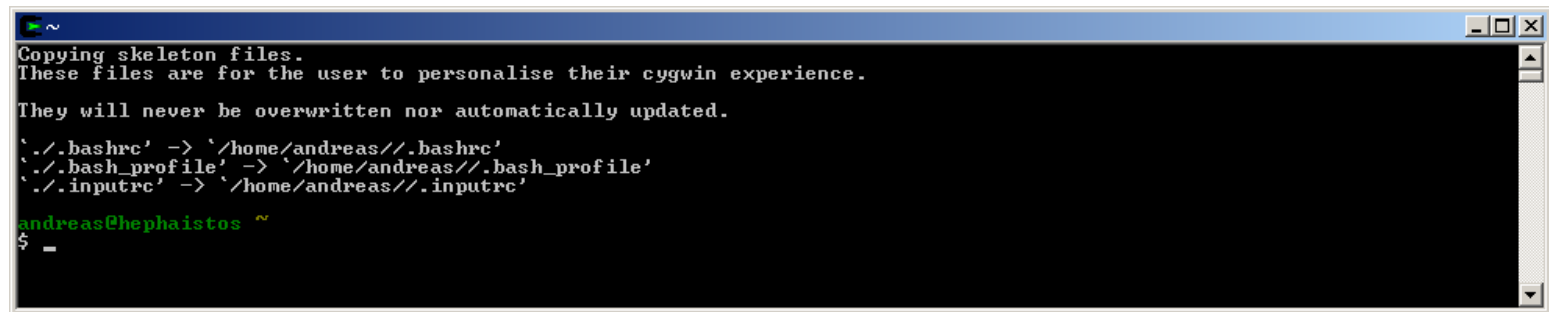
- Abschluss der Installation - Setzen der Umgebungsvariablen.
- „Systemsteuerung“ → „System“ → Registerkarte „Erweitert“ → „Umgebungsvariablen“ → „Systemvariablen“ → „Path“
- Das „bin“-Verzeichnis der Cygwin-Installation eintragen, zb.



# OCTAVE: Windows-Installation

## ■ Installation / Konfiguration von Cygwin

- Erster Start von Cygwin, wahlweise:
  - > Startmenü
  - > Desktop-Symbol
  - > Direkt aus dem Verzeichnis, z.b. `C:\programme\cygwin\cygwin.bat`

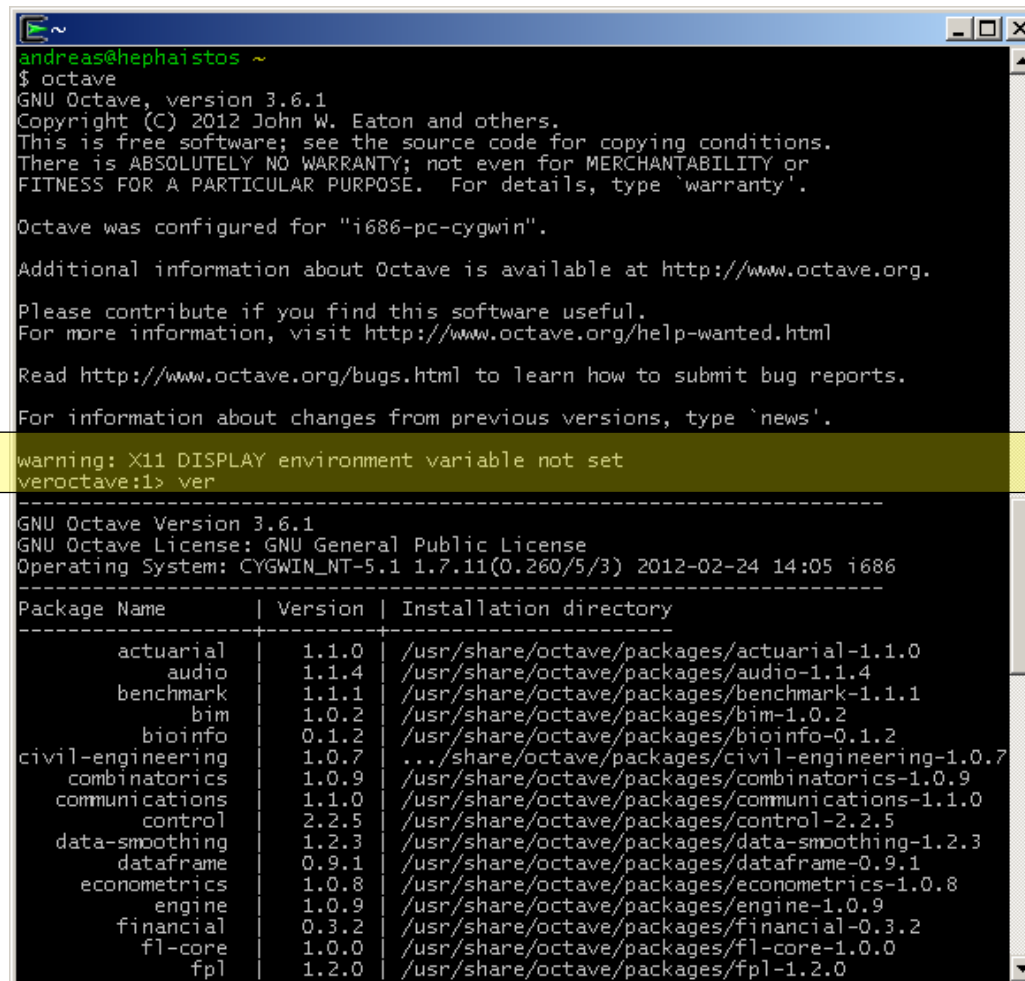


```
~  
Copying skeleton files.  
These files are for the user to personalise their cygwin experience.  
They will never be overwritten nor automatically updated.  
`./bashrc' -> '/home/andreas//.bashrc'  
`./bash_profile' -> '/home/andreas//.bash_profile'  
`./inputrc' -> '/home/andreas//.inputrc'  
andreas@hephaistos ~  
$ -
```

# OCTAVE: Windows-Installation

## ■ Installation / Konfiguration von Cygwin

- In Cygwin kann GNU Octave mit dem Kommando "octave" gestartet werden



```
andreas@hephaistos ~
$ octave
GNU Octave, version 3.6.1
Copyright (C) 2012 John W. Eaton and others.
This is free software; see the source code for copying conditions.
There is ABSOLUTELY NO WARRANTY; not even for MERCHANTABILITY or
FITNESS FOR A PARTICULAR PURPOSE. For details, type `warranty'.

Octave was configured for "i686-pc-cygwin".

Additional information about Octave is available at http://www.octave.org.

Please contribute if you find this software useful.
For more information, visit http://www.octave.org/help-wanted.html

Read http://www.octave.org/bugs.html to learn how to submit bug reports.

For information about changes from previous versions, type `news'.

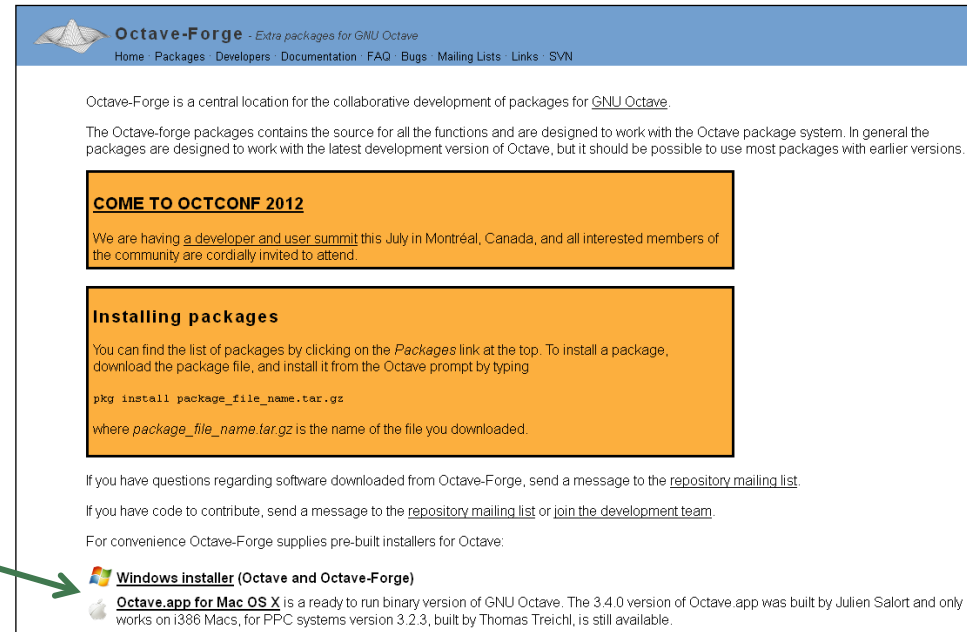
warning: X11 DISPLAY environment variable not set
ver octave:1> ver
-----
GNU Octave Version 3.6.1
GNU Octave License: GNU General Public License
Operating System: CYGWIN_NT-5.1 1.7.11(0.260/5/3) 2012-02-24 14:05 i686
-----
Package Name | Version | Installation directory
-----
actuarial    | 1.1.0   | /usr/share/octave/packages/actuarial-1.1.0
audio        | 1.1.4   | /usr/share/octave/packages/audio-1.1.4
benchmark   | 1.1.1   | /usr/share/octave/packages/benchmark-1.1.1
bim          | 1.0.2   | /usr/share/octave/packages/bim-1.0.2
bioinfo      | 0.1.2   | /usr/share/octave/packages/bioinfo-0.1.2
civil-engineering | 1.0.7   | .../share/octave/packages/civil-engineering-1.0.7
combinatorics | 1.0.9   | /usr/share/octave/packages/combinatorics-1.0.9
communications | 1.1.0   | /usr/share/octave/packages/communications-1.1.0
control      | 2.2.5   | /usr/share/octave/packages/control-2.2.5
data-smoothing | 1.2.3   | /usr/share/octave/packages/data-smoothing-1.2.3
dataframe    | 0.9.1   | /usr/share/octave/packages/dataframe-0.9.1
econometrics | 1.0.8   | /usr/share/octave/packages/econometrics-1.0.8
engine       | 1.0.9   | /usr/share/octave/packages/engine-1.0.9
financial     | 0.3.2   | /usr/share/octave/packages/financial-0.3.2
fl-core      | 1.0.0   | /usr/share/octave/packages/fl-core-1.0.0
fpl          | 1.2.0   | /usr/share/octave/packages/fpl-1.2.0
```

X11 DISPLAY environment:  
Wird zum Plotten benötigt,  
führt auf Windows-Systemen  
immer wieder zu Problemen...

## OCTAVE: Windows-Installation

### ■ Stand-alone Installation von GNU Octave (mit mingW32)

- Mit der Cygwin-Installation wurde (bei entsprechender Auswahl) das Programmpaket "Octave" mitinstalliert
- Dieses funktioniert jedoch häufig (je nach Windows-Version, etc.) nicht vollständig (fehlende/inkompatible Pakete/Funktionen, Abhängigkeiten mit anderen Paketen bzw. Software; "plot" und "record" sind häufig betroffene Funktionen)
- Lösung: GNU Octave (stand-alone Version) installieren (i.e., Archiv extrahieren; am besten in ein Verzeichnis innerhalb der cygwin-Installation)



The screenshot shows the Octave-Forge website. At the top is a blue header with the Octave-Forge logo and navigation links: Home, Packages, Developers, Documentation, FAQ, Bugs, Mailing Lists, Links, SVN. Below the header, the text states: "Octave-Forge is a central location for the collaborative development of packages for GNU Octave." and "The Octave-Forge packages contains the source for all the functions and are designed to work with the Octave package system. In general the packages are designed to work with the latest development version of Octave, but it should be possible to use most packages with earlier versions." There are two orange boxes. The first box is titled "COME TO OCTCONF 2012" and contains text about a developer and user summit in July in Montréal, Canada. The second box is titled "Installing packages" and contains instructions on how to install a package from the Octave prompt, including a code snippet: `pkg install package_file_name.tar.gz`. Below these boxes, there is text about mailing lists and a link to the development team. At the bottom, there is a section for pre-built installers. A green arrow points from the left towards the "Windows installer (Octave and Octave-Forge)" link in this section.

**Octave-Forge** - Extra packages for GNU Octave  
Home · Packages · Developers · Documentation · FAQ · Bugs · Mailing Lists · Links · SVN

Octave-Forge is a central location for the collaborative development of packages for GNU Octave.

The Octave-Forge packages contains the source for all the functions and are designed to work with the Octave package system. In general the packages are designed to work with the latest development version of Octave, but it should be possible to use most packages with earlier versions.

**COME TO OCTCONF 2012**

We are having a developer and user summit this July in Montréal, Canada, and all interested members of the community are cordially invited to attend.

**Installing packages**

You can find the list of packages by clicking on the *Packages* link at the top. To install a package, download the package file, and install it from the Octave prompt by typing


```
pkg install package_file_name.tar.gz
```


where *package\_file\_name.tar.gz* is the name of the file you downloaded.

If you have questions regarding software downloaded from Octave-Forge, send a message to the [repository mailing list](#).

If you have code to contribute, send a message to the [repository mailing list](#) or [join the development team](#).

For convenience Octave-Forge supplies pre-built installers for Octave:

 **Windows installer (Octave and Octave-Forge)**

 **Octave.app for Mac OS X** is a ready to run binary version of GNU Octave. The 3.4.0 version of Octave.app was built by Julien Salort and only works on i386 Macs, for PPC systems version 3.2.3, built by Thomas Treichl, is still available.



## OCTAVE: Windows-Installation

### ■ Start und Funktionstest (Optionen)

- (1) `cygwin` starten, daraus dann `octave` starten
- (2) standalone Version von `octave` starten (Icon)
- (3) `cygwin` starten, zum Verzeichnis der standalone-`octave` Version wechseln, `octave` im bin-Verzeichnis starten
- [(4) das komplette Verzeichnis der standalone-Version von `octave` nach `cygwin` kopieren, dann weiter bei (1) ]

### Funktionstest:

```
octave.exe:1> mysong = record(5,5000)
Please hit ENTER and speak afterwards!

5+0 records in
5+0 records out
25000 bytes (25 kB) copied, 7.125 s, 7.7 kB/s

octave.exe:2> plot(mysong)
```

**Funktionieren beide Kommandos ist alles in Ordnung!**

# 01 INTRODUCTION OCTAVE, GNUPLOT

20.03.2012

Dipl.-Ing. Dr. Andreas Riener, Dipl.-Ing. Gerold Hölzl  
Johannes Kepler University Linz  
Institute for Pervasive Computing  
Altenberger Strasse 69, A-4040 Linz



# OCTAVE: Hilfesystem

- **Befehl help**

- Listet alle Funktionen

- **Befehl help <funktionenname>**

- Beispiel

```
octave-3.6.1.exe:1> help for
*** for:
-- Keyword: for I = RANGE
    Begin a for loop.
        for i = 1:10
            i
        endfor
    See also: do, while.
...

```

## OCTAVE: Hilfesystem (2)

- Suchen nach einem Befehl dessen Name nicht genau bekannt ist:

`lookfor [-all] <suchstring>`

- Beispiel:

```
octave-3.6.1.exe:2> lookfor -all while
break      Exit the innermost enclosing do, while or for loop.
continue   Jump to the end of the innermost enclosing do,
           while or for loop.
end         Mark the end of any 'for', 'if', 'do', 'while', or
           'function' block.
endwhile    Mark the end of a while loop.
while       Begin a while loop.
...
```

# OCTAVE: Der Workspace

## ■ Anzeige/Löschen von Variablen

- Informationen über alle definierten Variablen erhält man mit den Befehlen `who` bzw. `whos` (Parameter/Optionen: `help who`, `help whos` bzw. `doc who/whos`).

Beispiele:

```
octave-3.6.1.exe:3> whos
```

Variables in the current scope:

```
NEWVLA    a          ans          b          s
```

```
octave-3.6.1.exe:4> whos
```

Attr	Name	Size	Bytes	Class
====	====	====	=====	=====
	NEWVLA	1x45	45	char
	a	1x3	24	double
	ans	1x45	45	char
	b	1x1	8	double
	s	1x8	8	char

# OCTAVE: Der Workspace

## ■ Formatierung der Anzeige

- Ausgabe von `whos` wird festgelegt durch einen Formatstring in der Variable `whos_line_format`  
Optionen siehe `help whos_line_format`

```
octave-3.6.1.exe:5> whos_line_format  
ans = %a:4; %ln:6; %cs:16:6:1; %rb:12; %lc:-1;
```

Attr	Name	Size	Bytes	Class
====	====	====	=====	=====
	NEWVLA	1x45	45	char
	a	1x3	24	double
	s	1x8	8	char

## ■ Löschen von Variablen

- Löschen nicht mehr benötigter Variablen bzw. des gesamten Workspaces  
`octave-3.6.1.exe:6> clear a`  
`octave-3.6.1.exe:7> clear all`

# OCTAVE: Datentypen (1)

## ■ Zahlen

- Zahlen werden in Octave immer als **Double-Array** behandelt!

Beispiele:

```
octave-3.6.1.exe:8> a = 5
```

```
a = 5
```

```
octave-3.6.1.exe:9> b = 1.23456789
```

```
b = 1.2346
```

```
octave-3.6.1.exe:10> whos
```

Variables in the current scope:

Attr	Name	Size
====	====	====
	a	1x1
	b	1x1
	path	1x22

Total is 24 elements using 38 bytes

Selbst einfache (int-) Zahlen werden als 1x1 Double Array abgelegt!

Bytes	Class
====	====
8	double
8	double
22	char

# OCTAVE: Datentypen (2)

## ■ Matrizen

- Matrizen werden in Octave ebenfalls als **Double-Array** behandelt!

Beispiele:

```
octave-3.6.1.exe:11> a = 5  
a =  
    5
```

Matrizen lassen sich jederzeit erweitern. Felder, deren Wert noch nicht spezifiziert wurde, werden mit 0 aufgefüllt

```
octave-3.6.1.exe:12> a(1,3) = 6  
a =  
    5    0    6
```

Indizes werden in runden Klammern und durch Beistriche getrennt angegeben (Reihe, Spalte). Achtung: Indizes beginnen bei 1!

```
octave-3.6.1.exe:13> b = [1 2 ; 3 4]  
b =  
    1    2  
    3    4
```

Ganze Matrizen lassen sich in eckigen Klammern angeben. Spalten werden durch Leerzeichen, Reihen durch Semikolon getrennt



## OCTAVE: Datentypen (3)

### ■ Strings

- Strings werden als **Character-Array** betrachtet.

Beispiele:

```
octave-3.6.1.exe:14> s = 'embedded'
s = embedded
```

```
octave-3.6.1.exe:15> whos
```

Attr	Name	Size	Bytes	Class
====	====	====	=====	=====
	s	1x8	8	char

```
octave-3.6.1.exe:16> s = ['e' 'm' 'b' 'e' 'd' 'd' 'e' 'd']
s = embedded
```

```
octave-3.6.1.exe:17> whos
```

Attr	Name	Size	Bytes	Class
====	====	====	=====	=====
rwd	s	1x8	8	char

Strings werden in einfache Hochkomma eingeschlossen

Identes Ergebnis wie  
s = 'embedded'

# OCTAVE: Datentypen (4)

## ■ Komplexe Zahlen

- $\sqrt{-1}$  wird als Konstante "i" bezeichnet (imaginäre Zahl).

Beispiele:

```
octave-3.6.1.exe:18> a = 3.01 + 2i  
a = 3.0100 + 2.0000i
```

Attr	Name	Size	Bytes	Class
====	====	====	=====	=====
	a	1x1	16	double

```
octave-3.6.1.exe:19> real(a)  
ans = 3.0100
```

```
octave-3.6.1.exe:20> imag(a)  
ans = 2
```

Die Funktion `real()` liefert den Real-Anteil einer komplexen Zahl

`imag()` liefert den imaginären Anteil

# OCTAVE: Datentypen (5)

## ■ Strukturen

- Enthalten mehrere beliebig benannte Datenfelder

Beispiele:

```
octave-3.6.1.exe:21> a.name = 'Riener'
```

```
a =
```

```
{  
  name = Riener  
}
```

Felder werden mit Punkt indiziert  
und können beliebige Datentypen  
enthalten

```
octave-3.6.1.exe:22> a.forename = 'Andreas';
```

```
octave-3.6.1.exe:23> a.legs = 2
```

```
a =
```

```
{  
  name = Riener  
  forename = Andreas  
  legs = 2  
}
```

Ein Strichpunkt unterdrückt die  
sofortige Ergebnis-Ausgabe

# OCTAVE: Datentypen (6)

## ■ Cell-Arrays

- Jedes Feld kann unterschiedliche Datentypen enthalten

Beispiele:

```
octave-3.6.1.exe:24> a = {2.5 'Riener' [1 2 ; 3 4] }
```

a =

```
{  
  [1,1] = 2.5000  
  [1,2] = Riener  
  [1,3] =  
    1    2  
    3    4  
}
```

Cell Arrays werden in geschweiften Klammern angegeben

Indizierung von Cell Arrays mittels geschweiften Klammern

```
octave-3.6.1.exe:25> a{3}
```

ans =

```
1    2  
3    4
```

# OCTAVE: Arbeiten mit Matrizen (1)

## ■ Colon-Operator

- Mit dem Doppelpunkt lassen sich Reihen in der Form **Anfang:Ende** oder **Anfang:Schrittweite:Ende** erstellen

Beispiele:

```
octave-3.6.1.exe:26> 2:3:10
```

ans =

2	5	8
---	---	---

Eine Reihe (oder Vektor) beginnend mit dem Wert "2", Schrittweite "3" und bis zum Maximalwert "10" erstellen

```
octave-3.6.1.exe:27> M = [1:3 ; 4:6 ; 7:9]
```

M =

1	2	3
4	5	6
7	8	9

Matrix aus 3 Reihen mit den Werten "1" bis "3", "4" bis "6" und "7" bis "9" erstellen

# OCTAVE: Arbeiten mit Matrizen (2)

## ■ Indizierung von Reihen und Spalten

- Mit dem **Colon-Operator** können auch komplette Spalten und Reihen indiziert werden.

Beispiele:

(Reihe, Spalte)  
octave-3.6.1.exe:28> M(:,1)

ans =  
1  
4  
7

Ausgabe der 1.  
Spalte der Matrix M

octave-3.6.1.exe:29> M(2,:)

ans =  
4      5      6

Ausgabe der 2.  
Reihe der Matrix M

# OCTAVE: Arbeiten mit Matrizen (3)

## ■ Indizierung mit Matrizen

- Mit **Matrizen als Index** lassen sich mehrere Spalten/Reihen auf einmal auswählen.

Beispiele:

```
octave-3.6.1.exe:30> M([1 2], :)
```

```
ans =
```

1	2	3
4	5	6

Ausgabe von Reihe 1  
und 2 der Matrix M

```
octave-3.6.1.exe:31> M([1 3], end-1:end)
```

```
ans =
```

2	3
8	9

Die beiden letzten Spalten  
von Reihe 1 und 3.

**"end" bezeichnet den  
maximalen Index**

# OCTAVE: Arbeiten mit Matrizen (4)

## ■ Indizierung mit logischen Matrizen

- Mit **logischen Matrizen** lassen sich jene Elemente finden, die eine bestimmte Bedingung erfüllen.

Beispiele:

```
octave-3.6.1.exe:32> C = M<5
```

C =

1	1	1
1	0	0
0	0	0

Erstellen einer logischen Matrix. **Elemente die die Bedingung erfüllen sind "1", alle anderen sind "0"**

```
octave-3.6.1.exe:33> M(C)
```

ans =

1
4
2
3

Indizierung der Matrix M mittels logischer Matrix C. **Alternative mit gleichem Ergebnis: M(M<5)**



# OCTAVE: Matrixoperation (1)

## ■ Grundrechnungsarten (+,-,/,\*)

- Werden grundsätzlich als Matrix-Operationen angesehen.

Beispiele:

```
octave-3.6.1.exe:34> A=[1 2;3 4],B=[5 6; 7 8]
```

A =

1	2
3	4

B =

5	6
7	8

Matrixmultiplikation (Spaltenanzahl der linken muss mit der Zeilenanzahl der rechten Matrix übereinstimmen)

```
octave-3.6.1.exe:35> A*B
```

ans =

19	22
43	50

Multiplikation der einzelnen Felder, zb.

$$19 = 1*5 + 2*7$$

$$43 = 3*5 + 4*7$$

(Berechnung: Skalarprodukt auf Paare aus einem Zeilenvektor der ersten und einem Spaltenvektor der zweiten Matrix)

## OCTAVE: Matrixoperation (2)

### ■ Grundrechnungsarten (+,-,/,\*)

- Um die Operation elementweise anzuwenden, ist dem Operator ein Punkt voranzustellen.

Beispiele:

```
octave-3.6.1.exe:36> A.*B
```

```
ans =
```

```
5    12
21   32
```

$$\begin{aligned} 5 &= 1*5 \\ 21 &= 3*7 \end{aligned}$$

```
octave-3.6.1.exe:37> A^2
```

```
ans =
```

```
7    10
15   22
```

**Achtung:**  
 $A^2$  ist ungleich  $A.^2$   
 $7 = 1*1 + 2*3$   
 $15 = 3*1 + 4*3$

```
octave-3.6.1.exe:38> A.^2
```

```
ans =
```

```
1     4
9     16
```

**Achtung:**  
 $A^2$  ist ungleich  $A.^2$   
 $1 = 1*1$   
 $9 = 3*3$

## OCTAVE: Matrixoperation (3)

### ■ Inverse, Transponierte

- Inverse einer Matrix  $M = \text{inv}(M)$ 
  - > Matrix-Rechtsdivision  $A / B = A * \text{inv}(B)$
  - > Matrix-Linksdivision  $A \setminus B = \text{inv}(A) * B$
- Transponierte von  $M = M'$

Beispiele:

```
octave-3.6.1.exe:39> A=[1 2;3 4];
```

```
octave-3.6.1.exe:40> A\[13;17]
```

```
ans =
```

```
-9
```

```
11
```

```
octave-3.6.1.exe:41> A(:,1)'
```

```
ans =
```

```
1    3
```

Lösung des folgenden linearen  
Gleichungssystems:

$$x + 2y = 13$$

$$3x + 4y = 17$$

$$\mathbf{x = -9, y = 11}$$

Transponieren der  
1. Spalte der Matrix A

## OCTAVE: Matrixoperation (4)

### ■ Anwendung: Lösen linearer Gleichungssysteme

- 3 unabh. Gleichungen mit 3 Unbekannten:

$$\begin{aligned}x_1 - 12x_2 - 4x_3 &= -5 \\ -20x_1 + 3x_2 - 5x_3 &= -119 \\ -14x_1 - 3x_2 - 17x_3 &= -53\end{aligned}$$

Beispiel:

```
octave-3.6.1.exe:42> A=[1 -12 -4;-20 3 -5; -14 -3 -17];
```

```
octave-3.6.1.exe:43> b=[-5; -119; -53];
```

```
octave-3.6.1.exe:44> A\b
```

```
ans =
```

```
7.0000
```

```
2.0000
```

```
-3.0000
```

Lösung:

$$x_1 = 7$$

$$x_2 = 2$$

$$x_3 = -3$$

# OCTAVE: Ausgabeoperationen

## ■ Semicolon

- Sofern die Ausgabe nicht mit ";" unterdrückt wird, wird jedes Ergebnis ausgegeben.

Beispiele:

```
octave-3.6.1.exe:45> 'Hallo Welt'  
ans = Hallo Welt
```

## ■ Explizite Ausgaben

```
octave-3.6.1.exe:46> disp('Hallo Welt')  
Hallo Welt
```

```
octave-3.6.1.exe:47> a=2;  
octave-3.6.1.exe:48> fprintf('\nHallo Welt %d \n',a);
```

```
Hallo Welt 2
```

Mit **fprintf** können  
formatierte Texte  
ausgegeben werden  
(im Stil von "C")

# OCTAVE: Kontrollstrukturen (1)

## ■ IF-Statement

```
if (Bedingung) Befehl elseif (Bedingung) Befehl else Befehl endif  
octave-3.6.1.exe:49> help if ...
```

Beispiele:

```
octave-3.6.1.exe:50> a = 2;  
octave-3.6.1.exe:51> if (a==5)  
    disp('a ist 5')  
elseif (a==6)  
    disp('a ist 6')  
else  
    disp('a ist nicht 5')  
    disp('und auch nicht 6')  
endif
```

Ausgabe:

```
a ist nicht 5  
und auch nicht 6
```

# OCTAVE: Kontrollstrukturen (2)

## ■ FOR- und WHILE-Statement

- Struktur: **for** Variable = Matrix Befehl **end**  
          **while** (Bedingung) Befehl **end**
- "for" iteriert über alle Elemente der Matrix

Beispiel:

```
octave-3.6.1.exe:52> for i = [0.1 0.5 1 2.5 5]
                      fprintf('Aktueller Wert: %d\n', i);
                      endfor
```

Ausgabe:

```
Aktueller Wert: 0
Aktueller Wert: 0
Aktueller Wert: 1
Aktueller Wert: 2
Aktueller Wert: 5
```

## OCTAVE: Kontrollstrukturen (3)

```
octave-3.6.1.exe:53> for i = 1:4
                        fprintf('Aktueller Wert: %d\n', i);
                    endfor
```

Ausgabe:

```
Aktueller Wert: 1
Aktueller Wert: 2
Aktueller Wert: 3
Aktueller Wert: 4
```

```
octave-3.6.1.exe:54> a = 0;
                    while (a<3)
                        a = a+1
                    end
```

Ausgabe:

```
a = 1
a = 2
a = 3
```

Ausgabe wieso?  
kein **fprintf**-Befehl,  
jedoch Ausgabe nicht  
durch ";" unterdrückt!



# MATLAB: Kontrollstrukturen (4)

## ■ SWITCH-Statement

- Struktur: **switch** Variable **case** Konstante Befehl **otherwise** Befehl **end**

Beispiele:

```
octave-3.6.1.exe:55> methode = 'linear';
```

```
switch methode
    case {'linear','bilinear'}
        disp('Methode ist linear')
    case 'constant'
        disp('Methode ist konstant')
    otherwise
        disp('Unbekannte Methode')
end
```

Ausgabe:

```
Methode ist linear
```

# OCTAVE: Eingebaute Funktionen, Konstanten (1)

## ■ Konstanten

- Octave kennt die üblichen mathematischen Konstanten wie `pi`, `e`, etc. (diese sind **nicht** als Variablen im Workspace ersichtlich)

**Achtung:** Konstanten können überschrieben werden!

```
octave-3.6.1.exe:56> pi
```

```
ans = 3.1416
```

```
octave-3.6.1.exe:57> pi=5
```

```
ans = 5
```

```
octave-3.6.1.exe:58> x = 2 * pi
```

```
x = 10
```

## ■ Eingebaute Funktionen

Umfangreiche Informationen zu den Funktionen liefert das Kommando `help`.

### - (a) Mathematische Funktionen:

In Octave sind alle gängigen mathematischen Funktionen implementiert, zb. `sin`, `cos`, `tan`, `log2`, `log10`, etc.

> Ruft man Funktionen auf Matrizen auf, so werden die Funktionen auf die einzelnen Elemente angewandt!

# OCTAVE: Eingebaute Funktionen, Konstanten (2)

## ■ Eingebaute Funktionen

- Beispiel für die Hilfefunktion `help`

```
octave-3.6.1.exe:58> help log10
-- Mapping Function:  log10 (X)
    Compute the base-10 logarithm for each element of X.
    See also: log, log2, logspace, exp.
```

Overloaded function:

```
flog10 (fixed scalar, ...)
flog10 (fixed matrix, ...)
flog10 (fixed complex, ...)
flog10 (fixed complex matrix, ...)
```

```
log10 is a built-in function
.
.
.
```

# OCTAVE: Eingebaute Funktionen, Konstanten (3)

## ■ Eingebaute Funktionen

- (a) Mathematische Funktionen: Bsp. zu elementweiser Funktionsanwendung

```
octave-3.6.1.exe:59> a = (pi/2 : pi/3 : 2 * pi)'
```

```
a =
```

```
1.5708
```

```
2.6180
```

```
3.6652
```

```
4.7124
```

```
5.7596
```

```
octave-3.6.1.exe:60> b = sin(a)
```

```
b =
```

```
1.00000
```

```
0.50000
```

```
-0.50000
```

```
-1.00000
```

```
-0.50000
```

# OCTAVE: Eingebaute Funktionen, Konstanten (4)

## ■ Eingebaute Funktionen

### - (b) Matrix- bzw. Vektorfunktionen:

Octave stellt eine Reihe von nützlichen Funktionen für die Matrix- bzw. Vektormanipulation zur Verfügung.

### - Beispiele:

```
octave-3.6.1.exe:61> length(vect)
```

...gibt die Länge des Vektors 'vect' zurück.

```
octave-3.6.1.exe:62> size(matr)
```

...gibt die Grösse der Matrix 'matr' als Zeilenvektor [Zeilen, Spalten] zurück.

```
octave-3.6.1.exe:63> eye(m,n)
```

...erzeugt eine m x n Grosse Einheitsmatrix.

```
octave-3.6.1.exe:64> zeros(m,n) bzw. ones(m,n)
```

...erzeugt jeweils eine m x n grosse Matrix, im ersten Fall eine Nullmatrix, im zweiten Fall eine Matrix mit 1ern.

```
octave-3.6.1.exe:65> rand(m,n)
```

...erzeugt eine Zufallsmatrix mit gleichverteilten Einträgen [0,1[.

# OCTAVE: Eingebaute Funktionen, Konstanten (4)

## ■ Eingebaute Funktionen

- **(c) Binomialkoeffizient:** `bincoeff(n, k)`  $\binom{n}{k} = \frac{n!}{k! * (n-k)!}$   
`nchoosek(n, k)`

Die Funktion `nchoosek` ist meist schneller als `bincoeff` (bei kleinen, ganzzahligen Skalaren) und warnt zudem vor Genauigkeitsverlust (bei großen `n`, `k`).

Falls `n` ein Vektor ist werden alle Kombinationen der Elemente aus `n` (jeweils `k` Werte) zurückgegeben (eine Zeile pro Kombination).

- Beispiele:

```
octave-3.6.1.exe:66> nchoosek(5, 3)  $\binom{5}{3} = \frac{5!}{3! * (5-3)!} = \frac{120}{6 * 2} = 10$   
ans = 10
```

```
octave-3.6.1.exe:67> nchoosek(1:4, 3)
```

```
ans =
```

```
1 2 3
```

```
1 2 4
```

```
1 3 4
```

```
2 3 4
```

- Was ist die Ausgabe von `nchoosek(1:45, 6)` ?

# OCTAVE: Programmierung (1)

## ■ Scripts

- Können mit einem beliebigen ASCII-Texteditor geschrieben werden (der Editor Notepad++ wird mit der Windows-Version von Octave mitinstalliert und kann mit `edit` gestartet werden)
- Sind in einer Datei mit der Erweiterung `.m` abzulegen. Der Dateiname lautet gleich dem späteren Aufrufnamen!
- Damit Octave ein Script findet muss es sich im aktuellen Arbeitspfad oder im Octave-Suchpfad befinden (zb. im Root-Verzeichnis von Octave).
- Aufruf in der Kommandozeile mit ihrem Namen.
- Vermeide Dateinamen die gleichlautend mit eingebauten Funktionen sind.
- Kommentare werden durch vorangestellte `%`-Zeichen oder `##` gekennzeichnet.
- Der erste zusammenhängende Kommentarblock am Beginn des Scripts kann mit dem `help`-Befehl als Beschreibung/Hilfe ausgegeben werden.

Beispiel `edit embedded1.m`

Kommentare beginnen  
mit einem `%`-Zeichen

```
% Das erste Skript im Rahmen von Embedded Systems
a = (pi/2 : pi/3 : 2 * pi)';
b = sin(a);
```

# OCTAVE: Programmierung (2)

## ■ Scripts

- Aufruf von `embedded1.m`

```
octave-3.6.1.exe:68> embedded1
```

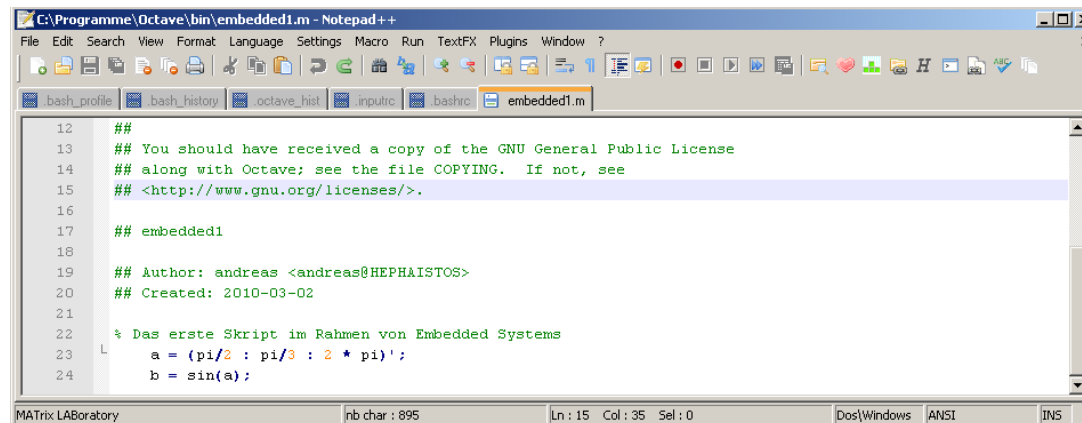
```
octave-3.6.1.exe:69> help embedded1
```

```
embedded1 is the file .\embedded1.m
```

Das erste Skript im Rahmen von Embedded Systems

```
octave-3.6.1.exe:70>
```

- Nach dem Aufruf sind die durch das Skript definierten Variablen im Workspace von Octave bekannt und können weiterverwendet werden.



The screenshot shows a Notepad++ window titled 'C:\Programme\Octave\bin\embedded1.m - Notepad++'. The window contains the following text:

```
12  ##
13  ## You should have received a copy of the GNU General Public License
14  ## along with Octave; see the file COPYING. If not, see
15  ## <http://www.gnu.org/licenses/>.
16
17  ## embedded1
18
19  ## Author: andreas <andreas@HEPHAISTOS>
20  ## Created: 2010-03-02
21
22  % Das erste Skript im Rahmen von Embedded Systems
23  L   a = (pi/2 : pi/3 : 2 * pi)';
24     b = sin(a);
```

The status bar at the bottom indicates 'MATrix LABORatory', 'nb char : 895', 'Ln : 15 Col : 35 Sel : 0', 'Dos/Windows', 'ANSI', and 'INS'.



# OCTAVE: Programmierung (3)

## ■ Selbstgeschriebene Funktionen

- Ähnlich wie Scripts, zusätzlich jedoch mit Ein- und/oder Ausgabeparametern
- Spezifikation der Funktion in der ersten Zeile der .m-Datei

Struktur:

```
function [ergebnis1,ergebnis2,...] =  
    funktionsname(parameter1,parameter2,...)  
  
% FUNKTIONSNAME Kurzhilfetext  
% FUNKTIONSNAME(par1,par2,...) akzeptiert ... Parameter  
% und gibt ... Werte zurück  
%  
% Sie dient nur zu Demonstrationszwecken.  
...
```

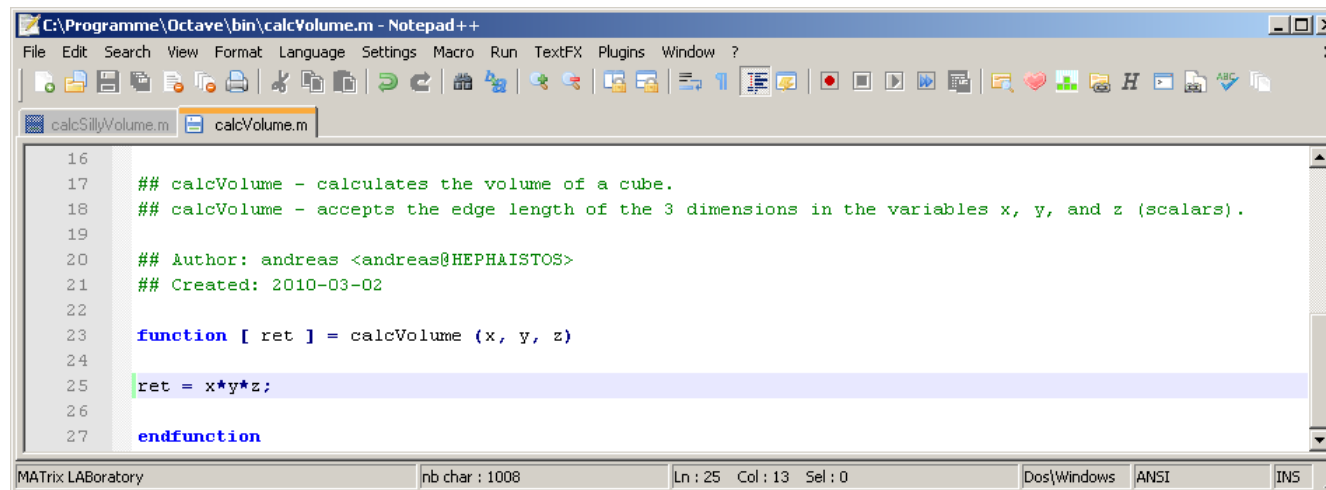
Aufruf in Octave:

```
octave-3.6.1.exe:71> [a,b,...] = funktionsname(1,2,...)
```

# OCTAVE: Programmierung (4)

## ■ Selbstgeschriebene Funktionen

### - Beispiel



```
C:\Programme\Octave\bin\calcVolume.m - Notepad++
File Edit Search View Format Language Settings Macro Run TextFX Plugins Window ?
calcSillyVolume.m calcVolume.m
16
17 ## calcVolume - calculates the volume of a cube.
18 ## calcVolume - accepts the edge length of the 3 dimensions in the variables x, y, and z (scalars).
19
20 ## Author: andreas <andreas@HEPHAISTOS>
21 ## Created: 2010-03-02
22
23 function [ ret ] = calcVolume (x, y, z)
24
25 ret = x*y*z;
26
27 endfunction
MATrix Laboratory nb char : 1008 Ln : 25 Col : 13 Sel : 0 Dos\Windows ANSI INS
```

### -Aufruf in Octave:

```
octave-3.6.1.exe:72> help calcVolume
```

```
'calcVolume' is a function from the file C:\Programme\Octave\bin\calcVolume.m
calcVolume - calculates the volume of a cube.
calcVolume - accepts the edge length of the 3 dimensions in the variables x, y,
and z (scalars).
```

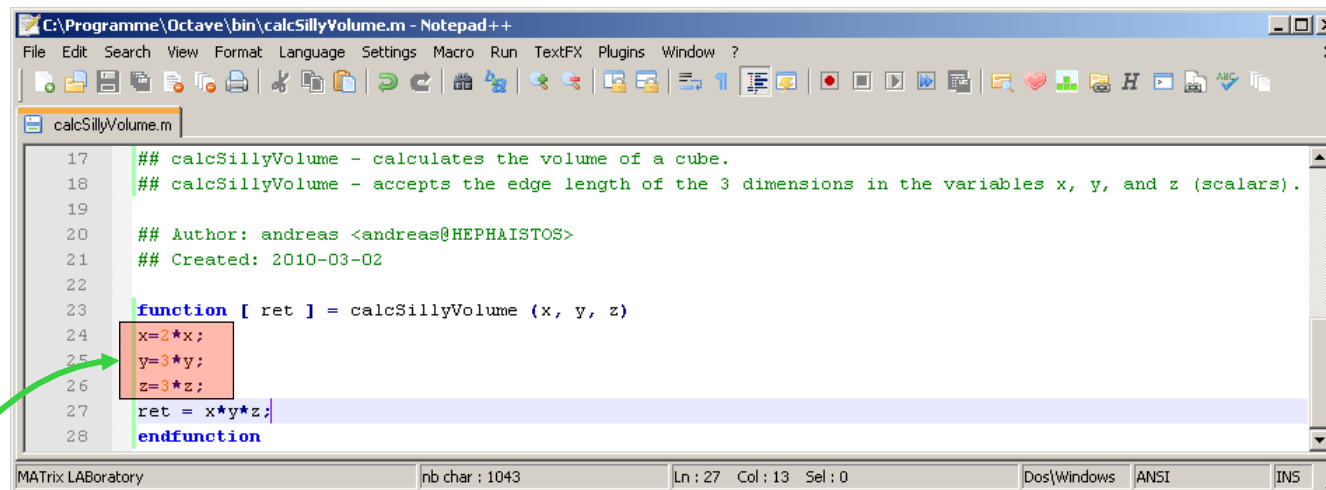
```
octave-3.6.1.exe:73> calcVolume(1,2,3)
```

```
ans = 6
```

# OCTAVE: Programmierung (5)

## ■ Unterschied von Funktionen und Scripts

- Beispiel



```
17  ## calcSillyVolume - calculates the volume of a cube.
18  ## calcSillyVolume - accepts the edge length of the 3 dimensions in the variables x, y, and z (scalars).
19
20  ## Author: andreas <andreas@HEPHAISTOS>
21  ## Created: 2010-03-02
22
23  function [ ret ] = calcSillyVolume (x, y, z)
24      x=2*x;
25      y=3*y;
26      z=3*z;
27      ret = x*y*z;
28  endfunction
```

-Aufruf in Octave:

```
octave-3.6.1.exe:74> x=1; y=1; z=1; erg=1;
octave-3.6.1.exe:75> [ret] = calcSillyVolume(x,y,z)
ret = 18
octave-3.6.1.exe:76> x,y,z
```

```
x=1
y=1
z=1
```

# OCTAVE: Programmierung (6)

## ■ Unterschied von Funktionen und Scripts

- Die Eingangsparameter werden durch Aufruf von Funktionen nicht verändert (**call-by-value**), die Variablen in der Funktion sind nur in der Funktion bekannt.
- Eine Änderung der Variablen des aufrufenden Workspace ist lediglich über die Ausgabeparameter der Funktion möglich.
- In der Funktionsdatei steht in der ersten Zeile die Funktionsdeklaration (wie oben beschrieben).
- Funktionen können im Gegensatz zu Scripts mit Argumenten aufgerufen werden.
- Variablen die in einer Skriptdatei und auf Kommandozeilenebene definiert werden sind global, Variablen innerhalb einer Funktion sind lokal.

# OCTAVE: (Einfache) Grafische Ausgaben (1)

## ■ 2-dimensionale Plots

- Befehl: `plot(x-vector, y-vector, style)`

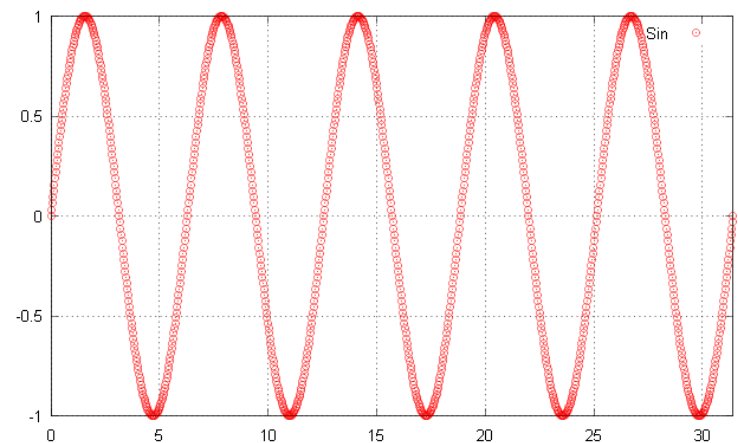
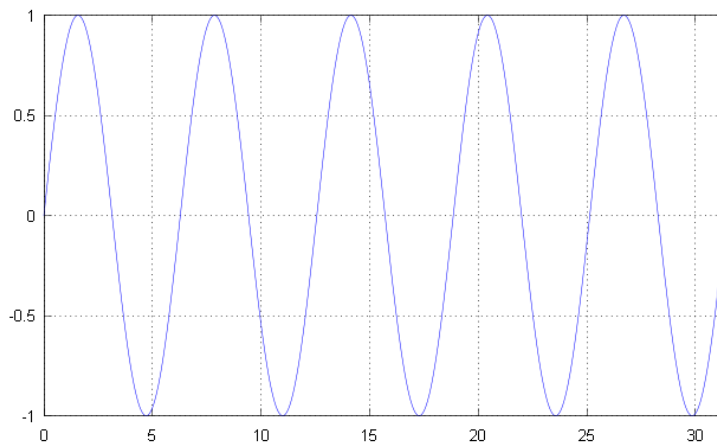
- Beispiel Ausgabe einer Sinuskurve:

```
octave-3.6.1.exe:77> a = (0 : pi/25 : 10*pi);
```

```
octave-3.6.1.exe:78> plot(a, sin(a))
```

- Art, Stil durch den 3. Parameter, zb. Kreise (o) in rot (r) und Beschriftung 'Sin'

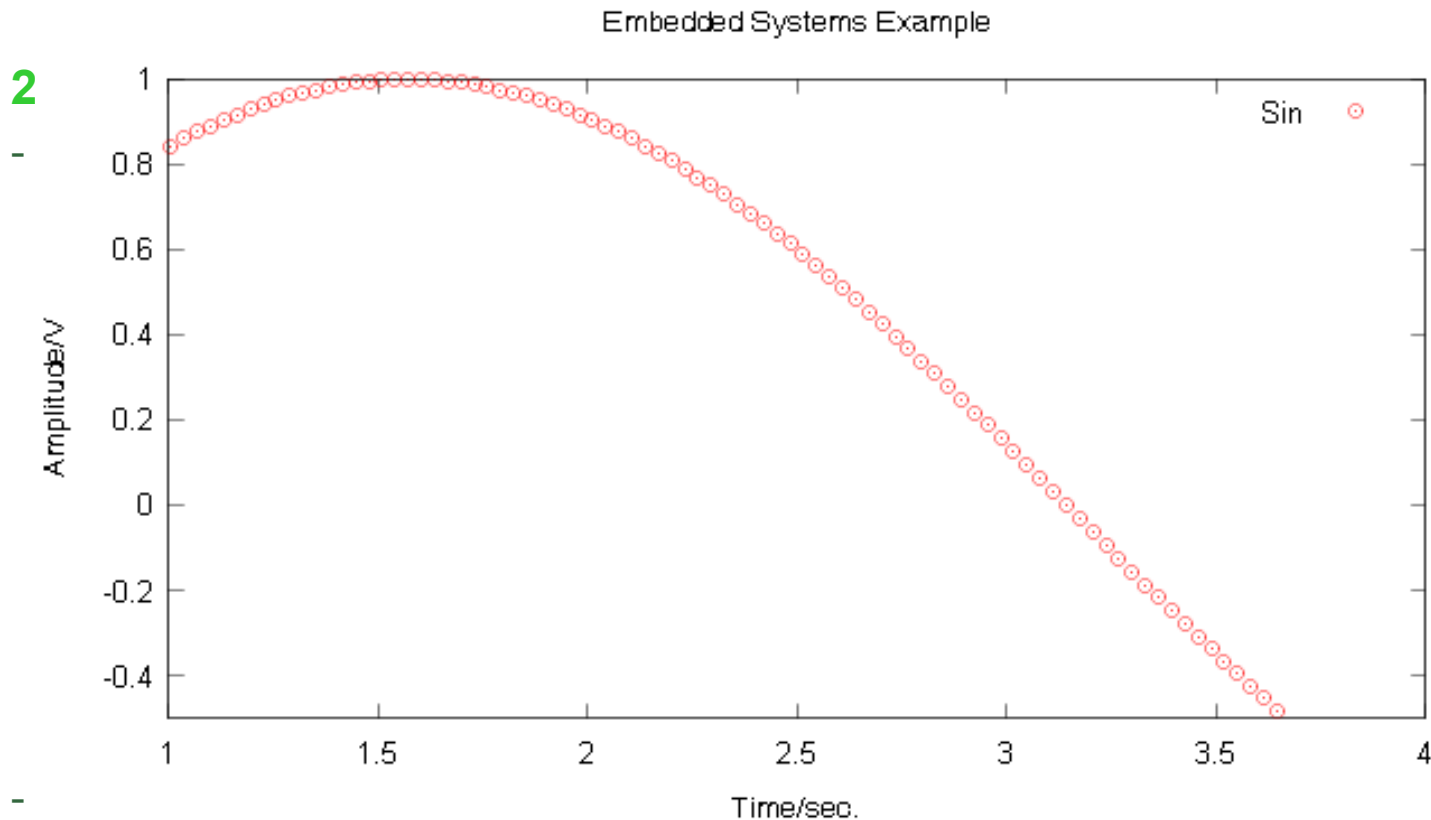
```
octave-3.6.1.exe:79> plot(a, sin(a), "or;Sin;")
```



- Weitere Parameter bzw. Details zur Funktion: `help plot`

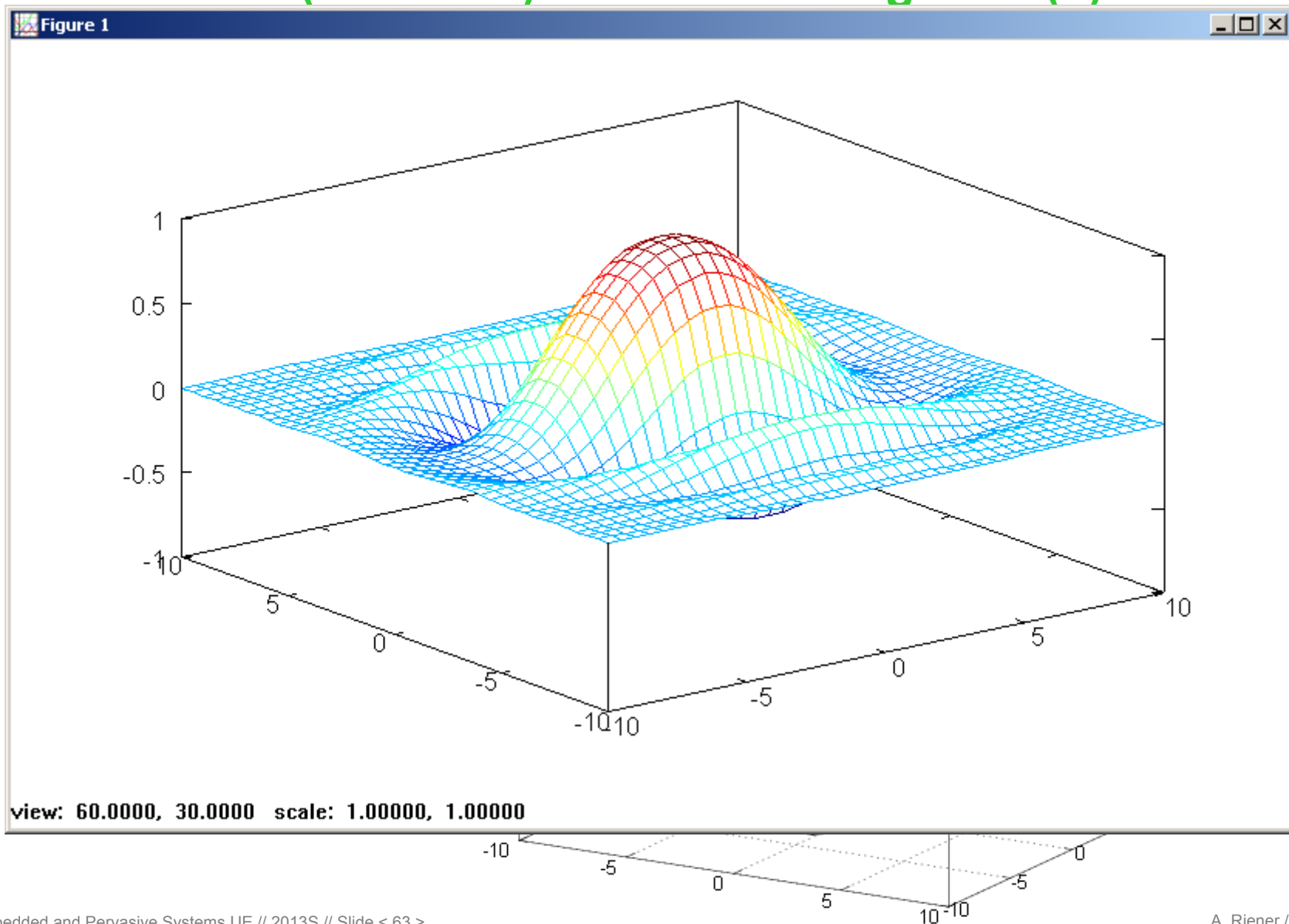
## OCTAVE: (Einfache) Grafische Ausgaben (2)

■ 2



```
octave-3.6.1.exe:80> figure
octave-3.6.1.exe:81> plot(a, sin(a), "or;Sin;")
octave-3.6.1.exe:82> xlabel("Time/sec.")
octave-3.6.1.exe:83> ylabel("Amplitude/V")
octave-3.6.1.exe:84> title("Embedded Systems Example")
octave-3.6.1.exe:85> axis([1,4,-0.5, 1])
```

## OCTAVE: (Einfache) Grafische Ausgaben (3)



## Octave-Erweiterungen



# OCTAVE: Erweiterungen

## Matlab "Toolboxes", Octave "Packages"

- Matlab stellt eine Reihe **kostenpflichtiger** sog. Toolboxes bereit um die Funktionalität des Programmes zu erweitern
- Manche dieser Toolboxes wurden für Octave nachimplementiert (zb. Digitalfilter, etc.) und stehen kostenlos zur Verfügung (**Matlab/Octave compatibility packages**)
- Eine ständig aktualisierte Liste der Programmkompatibilität einzelner Funktionen zwischen Octave und Matlab steht unter  
<http://users.powernet.co.uk/kienzle/octave/matcompat/index.html>
- Zudem gibt es auch Erweiterungen für Octave die unter Matlab nicht unmittelbar vorhanden sind (siehe Octave-Forge, nächste Folie) u. U.

# OCTAVE: Erweiterungen

## Octave <-> Matlab Kompatibilitätsdatenbank

- Paul Kienzle stellt auf seiner Homepage eine umfangreiche Datenbank mit Funktionen aus Matlab für Octave zur Verfügung: <http://users.powernet.co.uk/kienzle/octave/matcompat-2001.02.25.tar.gz>
- Das sog. MATCOMPAT-Paket wird in der Folge auch in den Übungen verwendet (Details, Installationsanleitung später)
  - Unter Windows-Rechnern wird ein installiertes Cygwin ("UNIX-Emulation" für Windows) vorausgesetzt, Installationsanleitung und Download unter <http://www.cygwin.com>
  - Eine weitere ausführliche Installationsanleitung: <http://www.tu-harburg.de/~matjz/work/octave/windows/#cygwin>

# OCTAVE: Erweiterungen (2)

## Octave-Forge

- Stellt zentrales Repository für Erweiterungen für GNU Octave dar
- Auszug verfügbarer Erweiterungspakete (> 80 packages)
  - Audio (audio recording, processing and playing tools)
  - Video (avifile, addframe, aviread, etc.)
  - Time (date/time manipulation tools)
  - Communications (digital communication, Error Correcting Codes ECC, etc.)
  - Image (read, write and process images)
  - Information Theory (theory definition, source coding)
  - Signal (signal processing tools, including filtering and display functions)
  - Statistics (additional statistical tools)

## Installation

- Herunterladen der Pakete (.tar.gz) in ein beliebiges Verzeichnis (einzeln, gesamt)
- Am besten Kopieren bzw. Verschieben in das Octave-Verzeichnis (`c:\programme\octave`) oder ein Unterverzeichnis davon
- Octave starten, in das Verzeichnis dass das Paket enthält wechseln (`cd...`)
- Installieren mit `pkg install package_file_name.tar.gz` (korrekten Paketnamen einsetzen)

# OCTAVE: Erweiterungen (3)

## Verwaltung von Erweiterungspaketen

- Die Packages werden abgelegt im Octave-Verzeichnis unter `/share/octave/packages/package_name/`

## Package laden

- Damit die Funktionen eines Paketes verwendet werden können, muss es zunächst in Octave geladen werden:  
`pkg load package_name`  
bzw. um alle installierten Pakete zu laden:  
`pkg load all`
- Damit alle Pakete immer gleich beim Start von Octave zur Verfügung stehen kann das Kommando "`pkg load all`" auch in die Konfigurationsdatei "`.octaverc`" (liegt in `/share/octave/3.2.4/m/startup/`) eingetragen werden

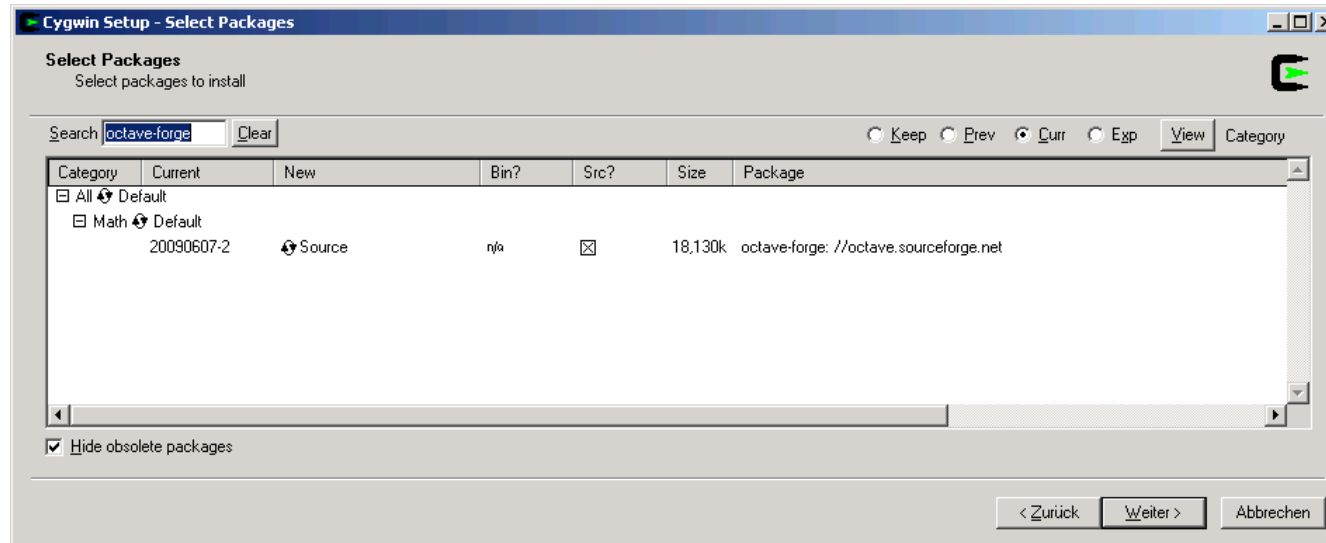
## Package deinstallieren

- `pkg uninstall package_name` (korrekten Paketnamen einsetzen)

# OCTAVE: Erweiterungen (4)

## Installation / Konfiguration via Cygwin Setup

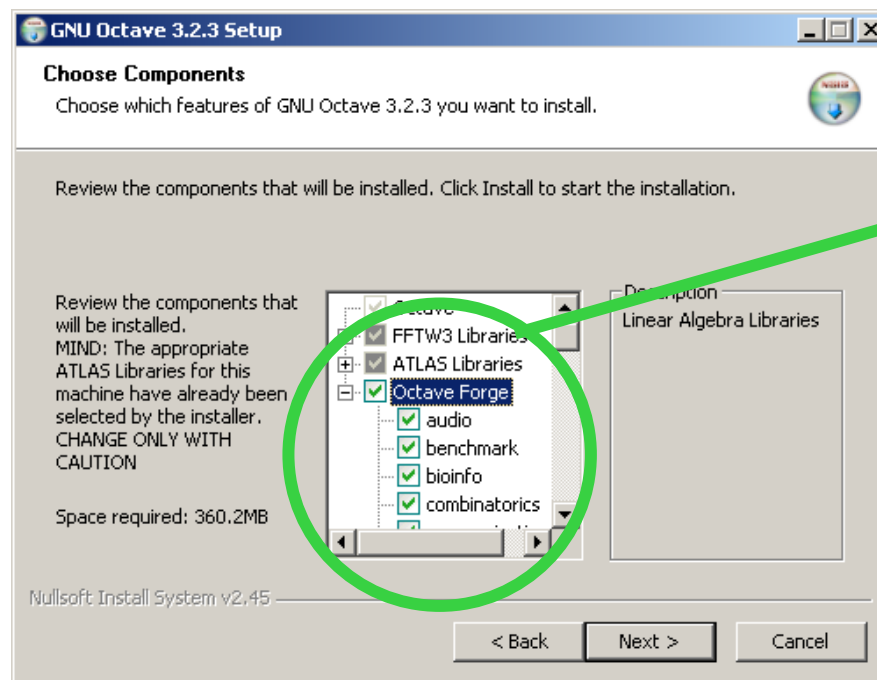
- In der Suchmaske Paketauswahl einschränken („octave-forge“)  
octave-forge
- Paket **octave-forge** zur Installation auswählen (es werden alle enthaltenen Pakete installiert)
- Installation mit „Weiter“ starten und abschließen
- Pakete werden installiert, müssen in Octave aber (trotzdem) manuell geladen werden (siehe vorige Folie!)



# OCTAVE: Erweiterungen (5)

## Installation für „Windows-Octave“

- Bei der Installation der Windows „standalone“-Version von Octave, die Komponente „Octave Forge“ vollständig auswählen
- Die (ausgewählten) Packages werden installiert und beim Start geladen (kein `pkg load package_name` nötig!)



Am besten alles aus  
„Octave Forge“  
mitinstallieren!

**Signalverarbeitung**  
*(unter Verwendung von Erweiterungspaketen)*

# OCTAVE: Signalverarbeitung

## Grundlagen

- Definieren eines Sinus-Signals  $s(t) = 5\sin(2\pi t/25)$

```
octave-3.2.4.exe:1> for t=1:256;s(t)=5*sin(2*pi*t/25);end
```

- Generieren von Rauschen (Gaussian Noise)

```
octave-3.2.4.exe:1> n=2*randn(1,256);
```

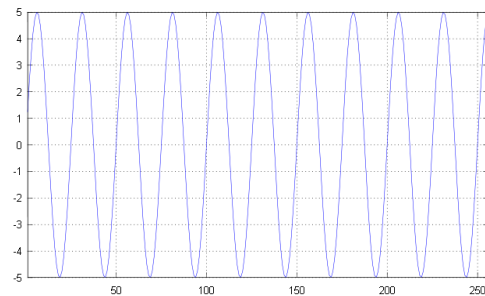
- Anwenden des Rauschens auf das Sinussignal

```
octave-3.2.4.exe:1> x=s+n;
```

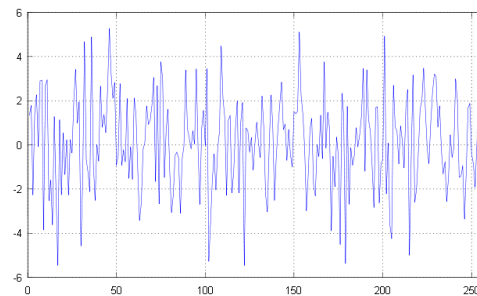
- Ausgabe

```
octave-3.2.4.exe:1> plot(s); plot(n); plot(x);
```

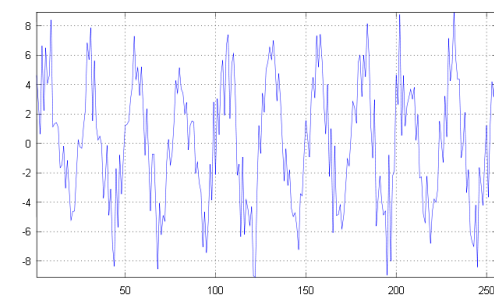
```
%octave-3.2.4.exe:2> plot(s,'r', n,'g', x,'b')
```



Signal



Noise



Signal+Noise



# OCTAVE: Signalverarbeitung

## Grundlagen (2)

- Filtern des Signals (Tiefpass-Filter)

```
octave-3.2.4.exe:1> b=fir1(10,0.3,'low');
```

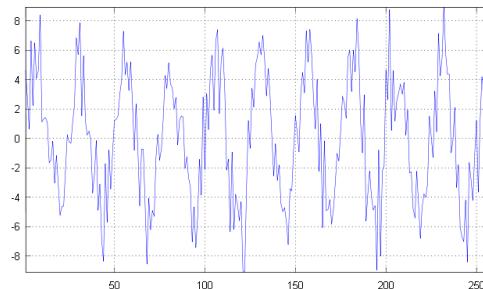
- Anwenden des Filters

```
octave-3.2.4.exe:1> y=filter(b,1,x);  
%x...verraushtes Sinussignal
```

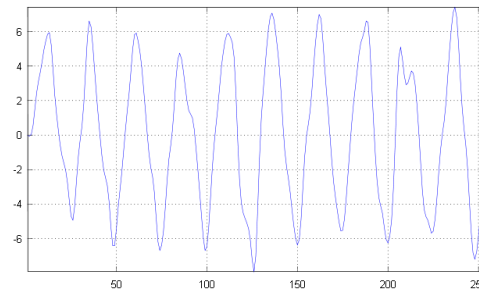
- Ausgabe

```
octave-3.2.4.exe:1> plot(x); plot(y); plot(x,'r',y,'b');
```

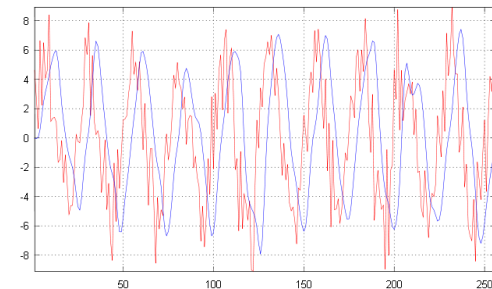
FIR...Finite impulse response  
stabiler Filtertyp, problemlos  
realisierbar; hohe Ordnung  
Param: Ord.,Grenzfreq.,Typ



Signal+Noise



Filtered Signal+Noise



Original+Filtered (Overlay)

**Audio**

# OCTAVE: Erweiterung "Audio"

## Audioverarbeitung

- Octave kann Audiodateien einlesen und verarbeiten. Bspw. können (einfache) Ausgangssignale eines A/D-Wandlers eingelesen und verarbeitet werden
- Charakterisiert werden die Signale anhand der Sample-Rate und der Wortlänge je Sample (Sample = einzelner Wert des A/D-Wandlers)
- Octave kann nur mit linear- und mu-law-kodierten Signalen arbeiten. Mu-law ist das internationale Format für Telefon-Verschlüsselung (Europa: A-law oder ITU) Umwandlung in das jeweils andere Format mittels der Funktion `lin2mu` bzw. `mu2lin`
- Ausgabe von Audiosignalen ("Abspielen")
  - Öffnen, Abspielen und Darstellung einer Audiodatei (mu-coded)

```
song = loadaudio('nothingelsematters','au',8);  
playaudio(song);  
plot(song);
```
- Eingabe von Audiosignalen ("Aufzeichnen/Recording")
  - Mit der Funktion `record` können Audiosignale über die Soundkarte des PCs (bzw. das Mikrofon oder auch die TV-Karte) aufgenommen werden
  - Parameter: Aufnahmedauer in Sekunden, Sample-rate in Hertz

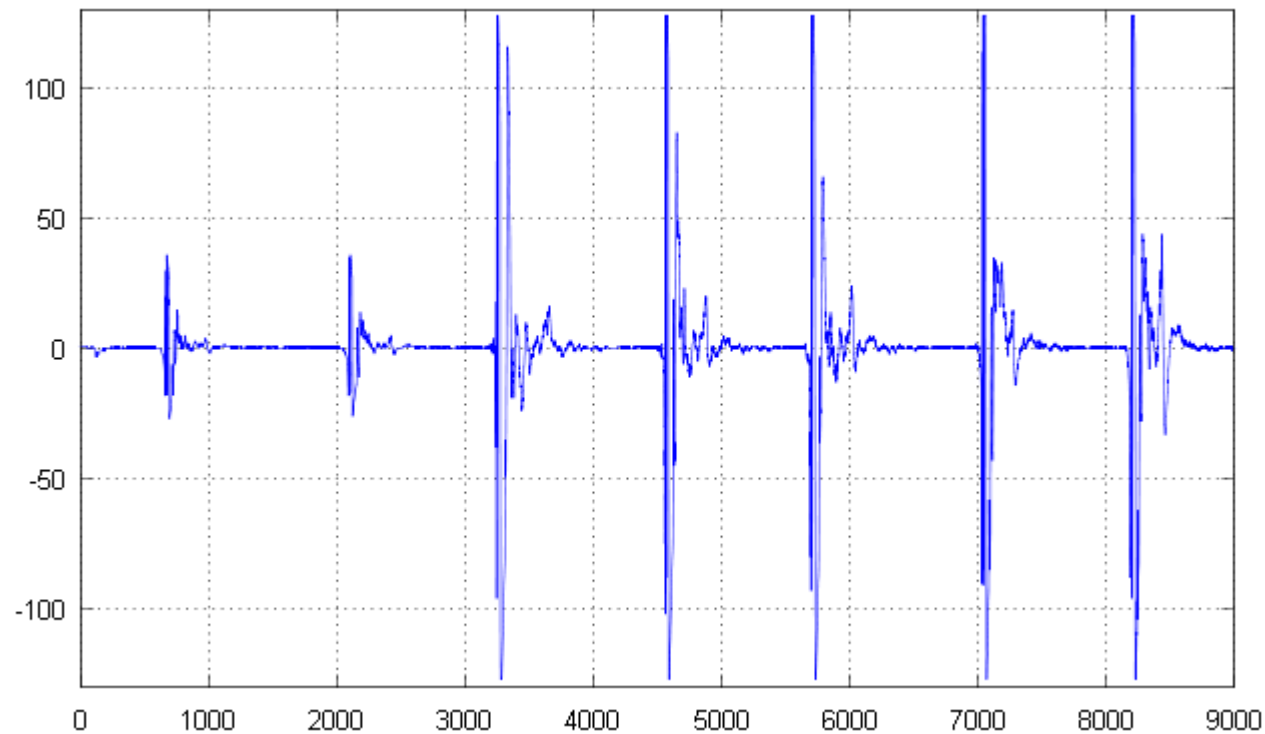
```
myvoice=record(8,11000); %8 Sekunden @ 11kHz
```

# OCTAVE: Erweiterung "Audio"

## Audioverarbeitung

- Darstellung (Plot)

```
plot(myvoice);
```



# OCTAVE: Erweiterung "Audio"

## Audioverarbeitung

- Speichern und Laden

```
saveaudio ('myvoice',mysong,'lin',8);  
x = loadaudio ('myvoice','lin',8);
```

- (Normierte) Darstellung von Audiosignalen

```
N = length(x); % Anzahl der Punkte  
b = 1:N;       % Behälter  
Ts = 1/8000;   % Abtastintervall in Sekunden  
fs = 1/Ts;     % Abtastfrequenz in Hertz  
ts = Ts*(b-1); % Abtastzeit  
plot(ts,x);    % Zeichnen des Signals
```

# OCTAVE: Erweiterung "Audio"

## Audioverarbeitung

- Weiterverarbeitung von Signalen
  - Aufgenommene Signale können als Datei gespeichert und geladen werden (Vektor)
  - Signal-Processing-Tools können zur Analyse verwendet werden

- Beispiel: Spektrum (DFT) eines Audiosignals

```
X = fft(x);           % DFT von x
pwr = X.*conj(X)/N;    % Leistung des Signals
frs = (b-1)/N*fs;      % Frequenzen
plot(frs,pwr);
```

**Bilder**

# OCTAVE: Erweiterung "Image"

## Bildverarbeitung (Image Processing)

- Octave kann Bilder als Matrizen einlesen, weiterverarbeiten und umgekehrt Matrizen wiederum als Bilder darstellen (mittels der Funktion `loadimage`)
- Zum Anzeigen von Bildern muss Octave mit dem X-Window System verwendet werden (`xloadimage` bzw. `xv` muss installiert sein); Manipulation von Bildern ist auch ohne dem X-System möglich
- Octave kann nur mit seinem eigenen Dateiformat (2 Matrizen, eine mit Bilddaten und eine mit Farbdaten) arbeiten

## Funktionsübersicht

- `[x, map] = loadimage(file)`  
Laden einer Bilddatei (Bildinhalt, Farbtabelle). Bild muss im Octave-Format gespeichert sein
- `saveimage (file,x)`  
Speichern einer Matrix `x` in Octave's Bildformat (die aktuelle Farbtabelle wird automatisch mitgespeichert)
- `saveimage (file,x,'ppm')`: Speichern im PPM statt Standard Octave-Format.  
`saveimage (file,x,'ps')`: Speichern im Postscript-Format (Hinweis: Bilder im `.ps`-Format können nicht mehr in Octave zurückgeladen werden!)



# OCTAVE: Erweiterung "Image"

## Funktionsübersicht (2)

- `[r, g, b] = ind2rgb(x)`  
Konvertieren eines indizierten Bildes in seine RGB-Farbkomponenten. Ohne Angabe einer Farbtabelle `[r, g, b] = ind2rgb(x, map)` wird die aktuelle für die Konversion verwendet
- `[x, map] = rgb2ind(r, g, b)`  
Konvertiert ein RGB-Bild in ein indiziertes Octave-Bild
- `ocean(n)`  
Generiert eine Farbtabelle mit n Farben (ohne Angabe von n: n=64)
- `imshow(x)`  
Zur Anzeige von indizierten Octave-Bildern unter Verwendung der aktuellen Farbtabelle  
`imshow(x, map)` verwendet die angegebene Farbtabelle  
`imshow(i, n)` für Bilder in Grauskala  
`imshow(r, g, b)` für die Anzeige von Bildern im RGB-Format
- etc.



**Input/Output**

# OCTAVE: Input and Output

## Grundlagen der Ein-/Ausgabe

- Zwei Klassen von Ein- und Ausgabeoperationen:
  - Angelehnt an den Befehlsvorrat von MATLAB
  - Entsprechend der Syntax der Standard I/O-Bibliothek der Sprache "C" (flexibler)

- Sondervariable "ans": Enthält den Wert der letzten Berechnung

```
octave-3.2.4.exe:1> 3*3 + 5^2;  
octave-3.2.4.exe:1> ans  
ans = 34
```

- Funktion `disp(x)` gibt den Wert von x aus:

```
octave-3.2.4.exe:1> disp ("Der Wert von e ist:", disp(e));
```

```
Der Wert von e ist:  
2.7183
```

- Kommando `format` bietet umfangreiche Formatierungsvarianten in Octave, zb. "short", "long", "bank", "hex", "bit", etc.

```
octave-3.2.4.exe:1> format long  
octave-3.2.4.exe:1> disp ("Der Wert von e ist:", disp(e));
```

```
Der Wert von e ist:  
2.71828182845905
```

# OCTAVE: Input and Output

## Terminal Input

- Funktion `input(prompt)` gibt einen Text aus und wartet auf Eingabe vom Benutzer

```
octave-3.2.4.exe:1> input("Ihre Eingabe: ")
```

```
Ihre Eingabe: 12
```

```
ans = 12
```

```
Ihre Eingabe: x=5*4 % Eingabe wird ausgewertet
```

```
ans = 20 % Variable x wird auf den Wert 20 gesetzt
```

```
octave-3.2.4.exe:1> input("Ihre Eingabe: ","s")
```

```
Ihre Eingabe: x=5*4 % Eingabe wird NICHT ausgewertet
```

```
ans = x=5*4
```

- Kommando `x=kbhit()` liest ein Zeichen von der Tastatur und weist es der Variable `x` zu. **Vorsicht: Tastaturpuffer!** - Verwenden von `fflush(stdout)`

```
octave-3.2.4.exe:83> x = kbhit()
```

```
12
```

```
x = 1
```

```
octave-3.2.4.exe:84> x = kbhit()
```

```
x = 2
```

# OCTAVE: Input and Output

## Einfache Dateiein-/ausgabe

- Die Kommandos `save` und `load` können verwendet werden um Daten in verschiedenen Formaten abzuspeichern und zu laden
- Standardeinstellungen beim Speichern können festgelegt werden durch die (eingebauten) Variablen `default_save_format` und `save_precision`  
`default_save_format`: "ascii", "binary", "float-binary", "mat-binary"  
`save_precision`: Anzahl der Stellen die gespeichert werden sollen (def=17)
- Octave kann keine Strukturen und benutzerdefinierten Datentypen laden/speichern
- Abspeichern der Variablen `var1`, `var2`, `var3` in die Datei mit Namen `myfile`:  
`save [options] myfile var1 var2 var3`
- Laden der Variablen `var1`, `var3` aus der Datei `myfile`:  
`load [options] myfile var1 var3`  
Wichtigste Option: `-force` verhindert das Überschreiben gleichnamiger Variablen
- Beispiel:  

```
octave-3.2.4.exe:1> result = 999;  
octave-3.2.4.exe:1> save myfile result  
octave-3.2.4.exe:1> result = 10;  
octave-3.2.4.exe:1> load myfile  
octave-3.2.4.exe:1> result  
ans = 999
```

# OCTAVE: Input and Output

## Dateiverarbeitung im "C"-Stil

- Octave kennt die "Standard streams" `stdin` (file id 0), `stdout` (file id 1) und `stderr` (file id 2)
- Öffnen von Dateien:  
`[fid, msg] = fopen (name, mode, arc)`  
`mode`: `read`, `write`, `read-only`, `append`, etc.  
`arc`: IEEE big endian, IEEE little endian, native, Cray-format, etc.  
`fid`: Referenz auf den Dateinamen (oder -1 im Falle eines Fehlers)  
`msg`: Enthält die zugehörige Fehlermeldung des Systems (falls `fid = -1`)
- Schliessen einer Datei:  
`fclose(fid)` schliesst die Datei mit der Referenz `fid`  
Im Falle eines Fehlers wird eine Fehlermeldung ausgegeben und die Funktion liefert den Rückgabewert 0, andernfalls Rückgabewert 1

# OCTAVE: Input and Output

## Ausgabe im "C"-Stil

- Basisfunktionen:

`fputs(fid, str)`: Unformatierte Ausgabe des Strings `str` in die Datei mit der Referenz `fid`

`puts(str)`: Ausgabe des Strings `str` auf die Standardausgabe

- Formatierte Ausgabe:

Die folgenden Funktionen können zur formatierten Ausgabe verwendet werden. Die Syntax ist jener der Programmiersprache "C" angelehnt (jedoch nicht zu 100% ident)

`printf(template, ...)`

`fprintf(fid, template, ...)`

`sprintf(template, ...)`

Beispiel:

```
octave-3.2.4.exe:1> percent=37;
```

```
filename = "foo.txt";
```

```
printf ("Processing of '%s' is %d%% finished.\nPlease be  
patient.\n", filename, percent);
```

```
Processing of `foo.txt' is 37% finished.
```

```
Please be patient.
```

```
octave-3.2.4.exe:2>
```

# OCTAVE: Input and Output

## Eingabe im "C"-Stil

- Basisfunktionen:

`fgetl(fid, len)`: Lesen von Zeichen aus der Datei mit der Referenz `fid`. Der Lesevorgang wird nach `len`-Zeichen oder einem "Newline" beendet. Die gelesenen Zeichen werden als String zurückgegeben (ohne "Newline"). Returnwert -1 wenn das Dateiende vor Ende des Lesevorganges erreicht wird

`fgets(fid, len)`: Funktion grundsätzlich wie `fgetl`, bei `fgets` wird bei Auftreten von "Newline" dieses Zeichen in den Rückgabestring inkludiert

- Formatierte Eingabe:

Die folgenden Funktionen können zur formatierten Eingabe im "C-Stil" verwendet werden (Syntax ist auch hier nicht zu 100% C-kompatibel)

`[val, count]=fscanf(fid, template, size)`

`[val, count]=sscanf(fid, template, size)`

`[val, count]=scanf(template, size)`

`[v1, v2, ..]=fscanf(fid, template, "C")`

`size`: Gibt die zu lesende Datenmenge an (optional)

`template`: String der angibt wie die gelesenen Daten zu interpretieren sind (Dezimalzahl, Gleitkommazahl, Zeichen, etc.)



**Filter**

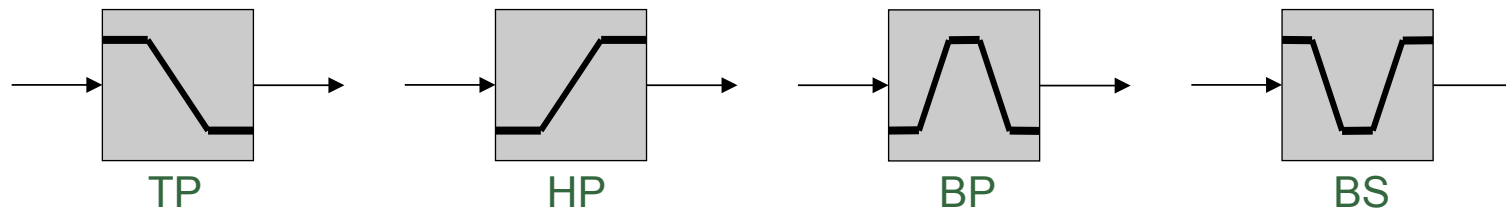
# OCTAVE: Filter

## Charakterisierung

- Filter dienen der gezielten Manipulation von Signalen
- Frequenzselektive Filter: Bereiche des Frequenzspektrums sind möglichst gut durchzulassen, andere Teile möglichst gut zu sperren
- Filtertypen: Tiefpass (TP), Hochpass (HP), Bandpass (BP), Bandsperre (BS)
- "Sondertyp" Allpass: Lässt alle Frequenzen durch, ändert aber die Phase des Signales (Anwendung: Kompensation störender Phasenverschiebungen)
- Realisierungsformen: analoge Filter (aktiv, passiv), digitale Filter

## Filter-Blockschaltbilder

- Zeigen den Amplitudengang des Filters
- Durchlassbereich: Frequenzen werden (theoretisch) ohne Änderung durchgelassen
- Sperrbereich: Frequenzen werden (theoretisch) auf Null-Amplitude unterdrückt
- Übergangsbereich verläuft (theoretisch) senkrecht



# OCTAVE: Filter

## Filtertypen, Eigenschaften

Jeder Filter ist durch seinen Durchlass- bzw. Sperrbereich charakterisiert

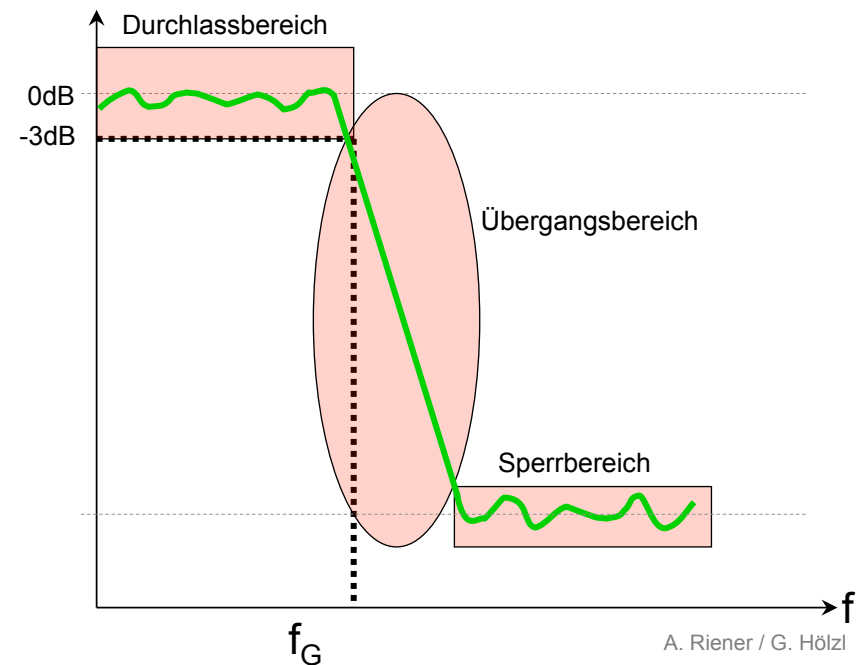
- Tiefpass (TP):
  - Lässt niedrige Frequenzen durch, sperrt hohe Frequenzen
  - Anwendung: Anti-Aliasing, Entfernung von hochfrequentem Rauschen aus Signalen
- Hochpass (HP):
  - Lässt hohe Frequenzen durch, sperrt (entfernt) niedrige Frequenzen
  - Anwendung: Entfernung von Gleichanteil im Signal, Unterdrücken von langsamen Drift im Signal; meist-verwendeter Filter
- Bandpass (BP):
  - Lässt nur einen bestimmten Frequenzbereich durch
  - Anwendung: Frequenzmodulierte Signale
- Bandsperre (BS):
  - Sperrt einen definierten Frequenzbereich
  - Anwendung: Unterdrückung der Störung durch das 50Hz Signal aus dem Versorgungsnetz

# OCTAVE: Filter

## Filtereigenschaften, Entwurf

Reale Filter: Kompromiss zwischen Steilheit und Welligkeit des Signales (*Je steiler die Flanke, desto welliger*)

- Filteruntersuchung im Frequenzbereich
  - Grenzfrequenz  $f_G$ : Jener Punkt, bei dem die Amplitude um 3dB bzw. das 0,707-fache des Wertes im Durchlassbereich zurückgeht
  - Filterordnung: Je höher die Ordnung, desto besser der Filter;
  - Durchlassbereich: möglichst flach
  - Übergangsbereich: möglichst steil
  - Sperrbereich: flach, hohe Dämpfung
  - Filtertyp: TP, HP, BP, BS, Allpass



# OCTAVE: Filter

## FIR, IIR-Filter

- **FIR-Filter (Finite Impulse Response)**
  - Ausgangswert wird aus unterschiedlich gewichteten, addierten früheren Eingangswerten zusammengesetzt. Maximale Anzahl an vergangenen Werten = Ordnung
  - Merkmale von FIR-Filtern:
    - (i) Immer stabil
    - (ii) Linearer Phasengang → keine Phasenverzerrungen
    - (iii) Ordnungen oft im zwei- bis dreistelligen Bereich
    - (iv) Problemlos realisierbar
- **IIR-Filter (Infinite Impulse Response)**
  - Alle früheren **Ausgangswerte** werden in die Berechnung des aktuellen Ausgangswertes miteinbezogen. Antwort auf einen Eingangsimpuls kann damit theoretisch unendlich lang werden - jeder Ausgangswert wird auf den Eingang rückgekoppelt
  - Merkmale von IIR-Filtern:
    - (i) Instabilität durch Rundungsfehler → "Aufschaukeln" durch Rückkopplung → schwierigere Realisierung
    - (ii) Kein linearer Phasengang → verschiedene Frequenzanteile werden durch den Filter unterschiedlich lange verzögert → "Verzerrung"
    - (iii) Vorteil gegenüber FIR: Um einen gewünschten Frequenzgang zu erzielen ist eine sehr viel niedrigere Ordnung ausreichend → weniger Rechenaufwand

# OCTAVE-Erweiterung: Filter

## Digitalfilter

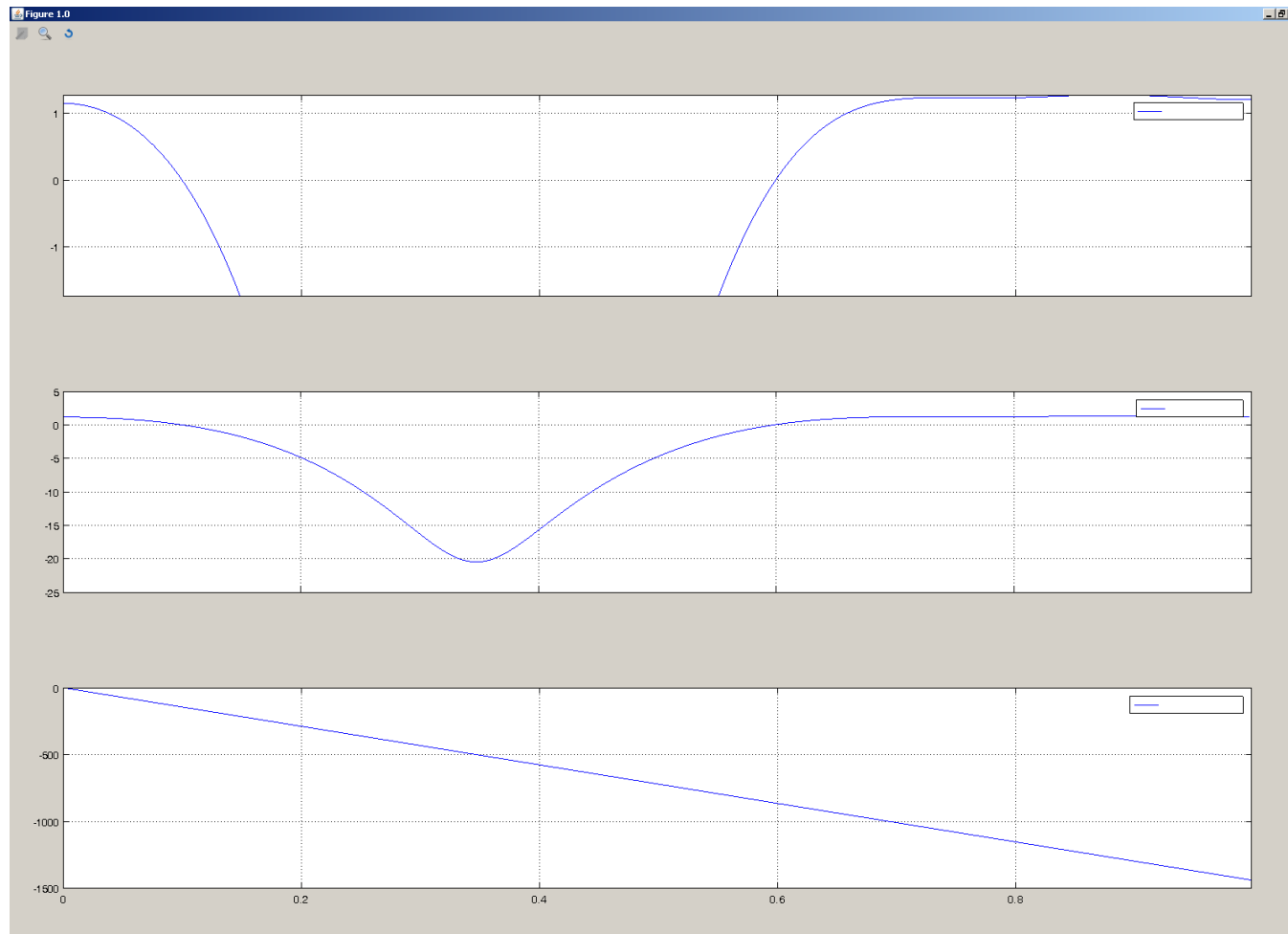
- Entwurf bzw. Verarbeitung von Digitalfiltern in MATLAB durch die mächtige "Signal Processing Toolbox" (kostenpflichtig)
- Ähnliche Funktionen in Octave durch das Programmpaket MATCOMPAT. Dieses muss zusätzlich installiert werden und enthält eine Reihe von Filterfunktionen
- `fir1` und `fir2` sind Funktionen zur Berechnung der Koeffizienten eines Digitalfilters nach entsprechender Spezifikation:

## Filterfunktionen

- `b = fir1(n, w [, type])`
  - `n`...Anzahl der Filterkoeffizienten (1 weniger als die Länge des Filters)
  - `w`...Grenzfrequenz(en) aus dem Intervall [0, 1]
    - > Einelementige Menge für einfachen Filter (Hochpass, Tiefpass) oder
    - > Vektor-Paar für Bandpass oder Bandsperr-Filter
  - `type`...Filtertyp
    - > 'high' für Hochpass, 'low' für Tiefpass, Grenzfrequenz bei `w`
    - > 'stop' für Bandsperrfilter, Grenzen bei `w = [lo, hi]`
- Detaillierte Beschreibung der Filterfunktionen in der Octave-Hilfe (`help fir1`)

# OCTAVE-Erweiterung: Filter

**Bsp:** `freqz(fir1(40,0.3));`  
`freqz(fir1(15,[0.2, 0.5], 'stop'));` # zero-crossing at 0.1



# OCTAVE-Erweiterung: Filter

## Beispiel Filterentwurf

- Spezifikation: Filterordnung 10, Grenzfrequenzen 6 bzw. 10kHz

## Lösung (Auszug)

```
fA = 44.0e+03; % Abtastfrequenz 44kHz (Audiosignal)
fGU = 6.0e+03; % Untere Grenzfrequenz
fGO = 10.0e+03; % Obere Grenzfrequenz
N = 10; % Ordnung des Filters
L = 2048; %
% Berechnung der Filterkoeffizienten
FIRKoeff = fir1(N,[fGU/fA*2.0 fGO/fA*2.0],'stop');
x = zeros(1,L); % Eingangsimpuls (Vektor = 0)
x(1) = 1.0; % 1. Wert auf 1
y = filter(FIRKoeff,1,x); % Berechnung der Impulsantwort des Filters
Y = fft(y); % Transformation der Impulsantwort mittels FFT in den
            % Frequenzbereich (Darstellung Dämpfungsverlauf)
f = [0:fA/(L-1):fA];
            % Grafikausgabe (Raster und Parametrisierung, Einzeichnen fGU, fGO)

grid;
axis([0 1.5 -80 0]);
hold on;
plot([fGU/10000,fGU/10000],[-80 10], 'b', [fGO/10000,fGO/10000],[-80,10], 'b');
hold off;
```



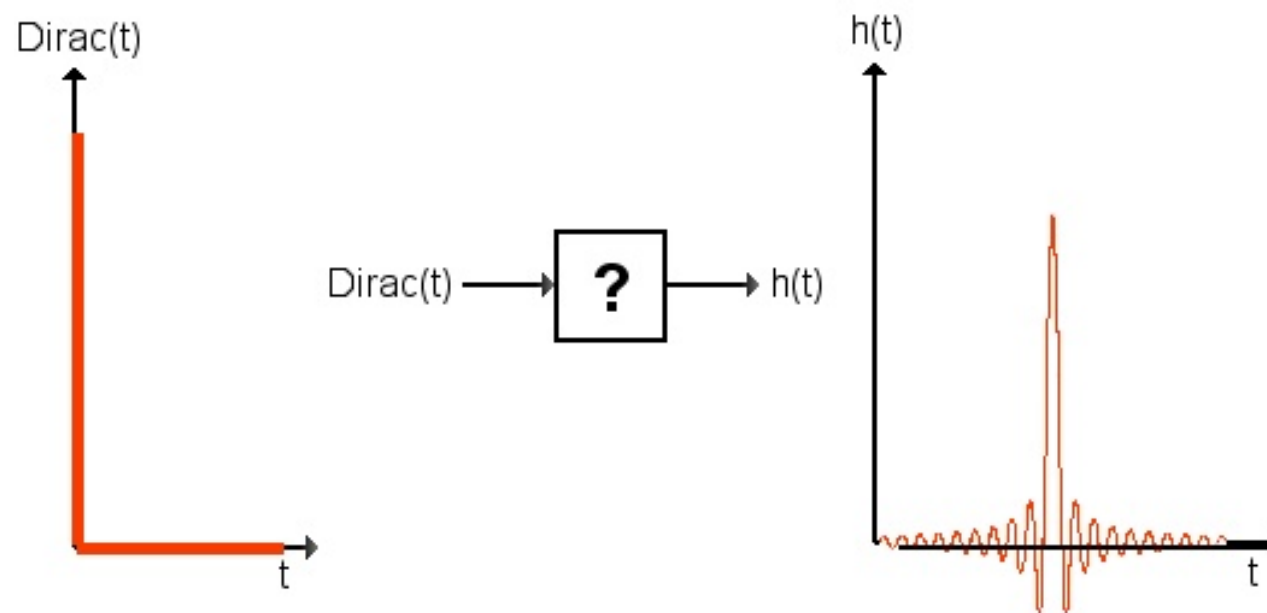
## **Filterentwurf mit Octave: Beispiele**

Teile der Inhalte zu Filterentwurf sind entnommen aus:  
VO Digitale Signalverarbeitung, Bernd Edler, WS08/09, Leibniz-Universität Hannover

# OCTAVE-FILTERENTWURF: Grundlagen

## Impulsantwort

- In Octave kann die Impulsantwort<sup>1</sup> mit '`impz`' generiert werden
- Die **Impulsantwort** ist das Ausgangssignal eines Systems bei dem am Eingang ein sog. Dirac-Impuls zugeführt wird

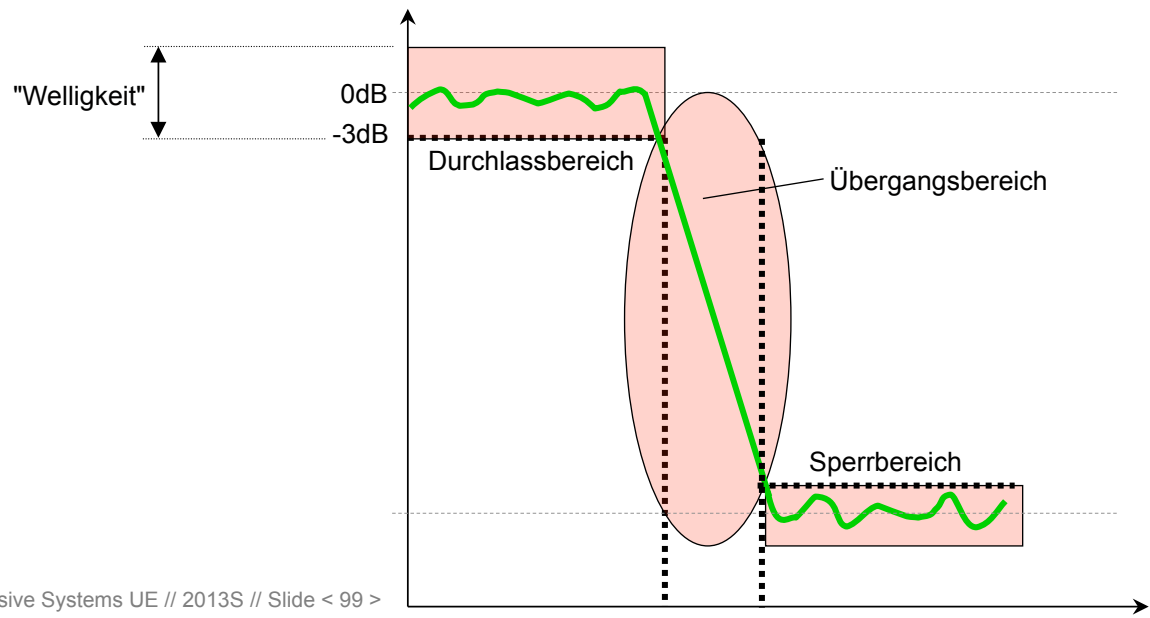


<sup>1</sup> Weiterführende Information zur Impulsantwort: <http://de.wikipedia.org/wiki/Impulsantwort>

# OCTAVE-FILTERENTWURF: Grundlagen

## Darstellung von Frequenz und Phase

- Frequenz- und Phasengang können in Octave mit der Funktion '`freqz`' dargestellt werden
  - "Pass band": (Durchlassbereich). Frequenzanteile die den Filter ohne Dämpfung (bzw. bis -3dB) passieren
  - "Stop band": (Sperrbereich). Frequenzanteile die der Filter nicht durchlässt (bzw. nur mit Dämpfung höher eines bestimmten Niveaus)
  - "Phase": Verschiebung der Phase zwischen Ein- und Ausgang des Filters



## **A) Infinite Impulse Response (IIR)-Filter**

# OCTAVE-FILTERENTWURF: Butterworth-Filter

## Butterworth IIR Filterdesign

### ■ Beispiel:

```
octave-3.2.4.exe:1> wd = 1/8;  
octave-3.2.4.exe:1> ws = 1/6;  
octave-3.2.4.exe:1> dd = 1;  
octave-3.2.4.exe:1> ds = 40;  
octave-3.2.4.exe:1> [n,wc] = buttord(wd,ws,dd,ds);  
octave-3.2.4.exe:1> [b,a] = butter(n,wc);
```

### ■ Ergebnis:

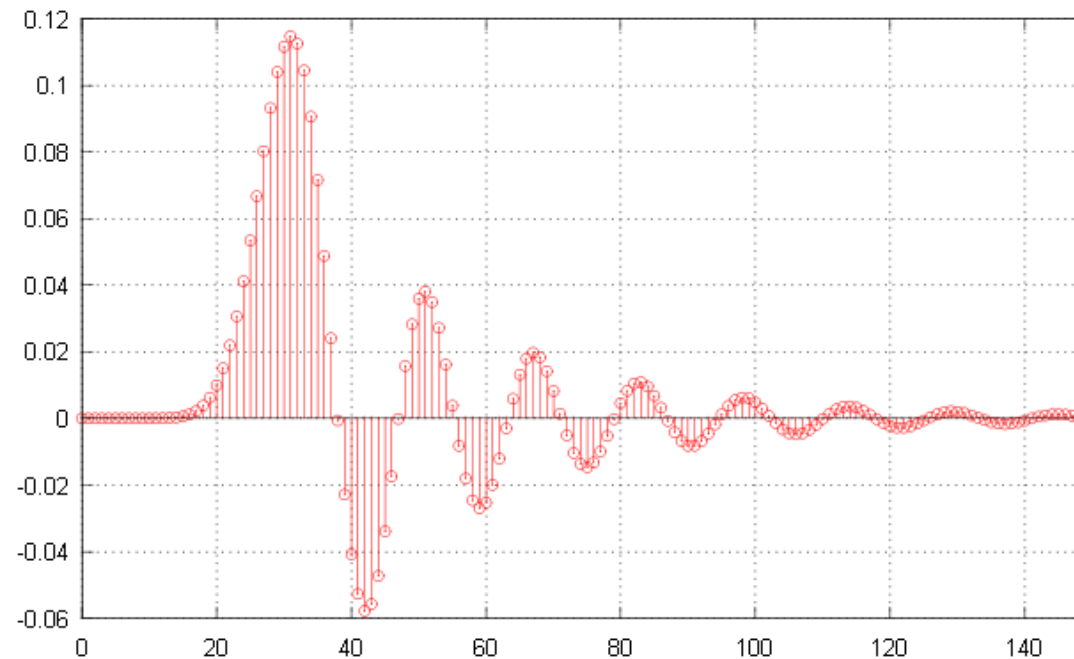
- '**buttord**' berechnet Filterordnung und cutoff für einen Butterworth-Filter
  - n ist die Ordnung/der Grad des Filters (hier: 18)
  - a, b sind die Koeffizienten des Zähler- (a) bzw. Nennerpolynoms (b)
- '**butter**' erzeugt einen Butterworth-Filter
- Filtertypen: Tiefpass, Hochpass, Bandpass, Bandstop

# OCTAVE-FILTERENTWURF: Butterworth-Filter

## Butterworth IIR Filterdesign (2)

- Impulsantwort mit `'impz'`, Darstellung als "stem"-Diagramm

```
octave-3.2.4.exe:1> L = 150;  
octave-3.2.4.exe:1> [x,t] = impz(b,a,L);  
octave-3.2.4.exe:1> stem(t,x);
```

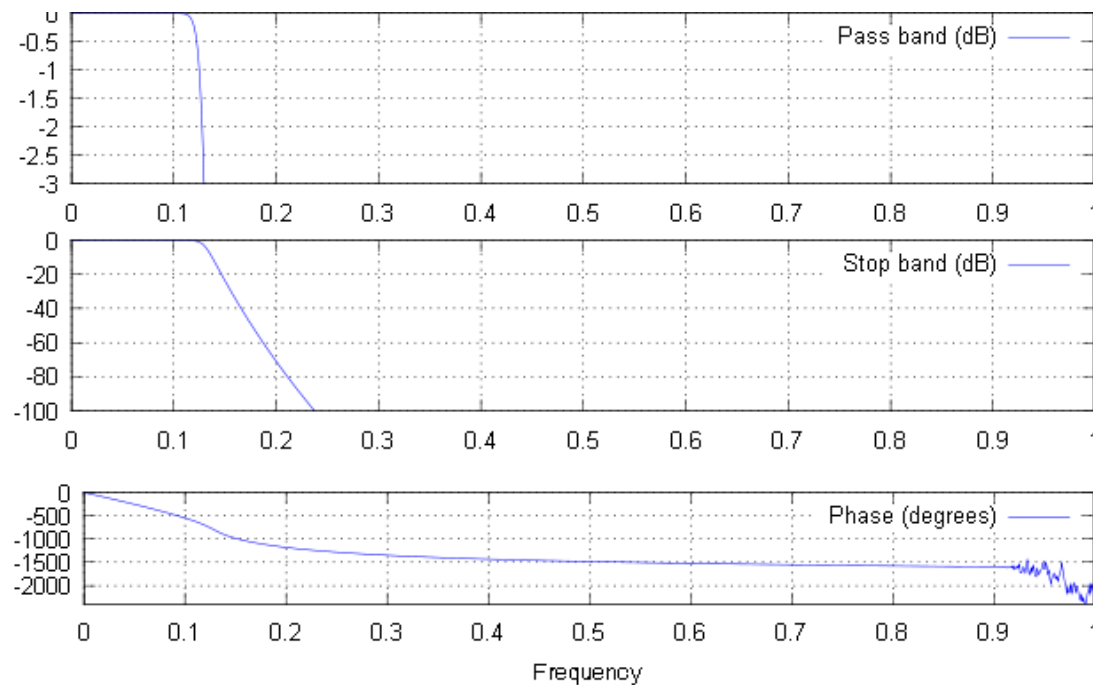


# OCTAVE-FILTERENTWURF: Butterworth-Filter

## Butterworth IIR Filterdesign (3)

- Darstellung von Frequenz- und Phasengang mit '`freqz`'

```
octave-3.2.4.exe:1> FREQRES = 1024;  
octave-3.2.4.exe:1> freqz(b,a,FREQRES);
```



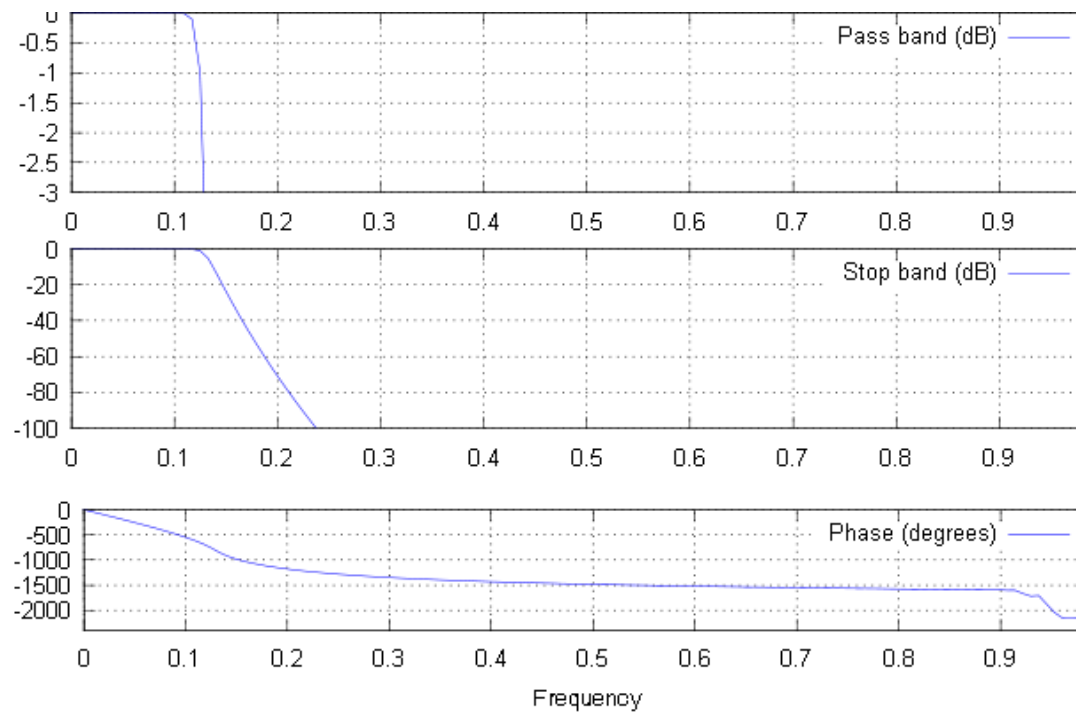
# OCTAVE-FILTERENTWURF: Butterworth-Filter

## Butterworth IIR Filterdesign (3)

- Darstellung von Frequenz- und Phasengang mit '`freqz`'

```
octave-3.2.4.exe:1> FREQRES = 128;
```

```
octave-3.2.4.exe:1> freqz(b,a,FREQRES);
```





# OCTAVE-FILTERENTWURF: Tschebyscheff-Filter

## Tschebyscheff IIR Filter: Tiefpass Typ 1 (Filtergrad n=8)

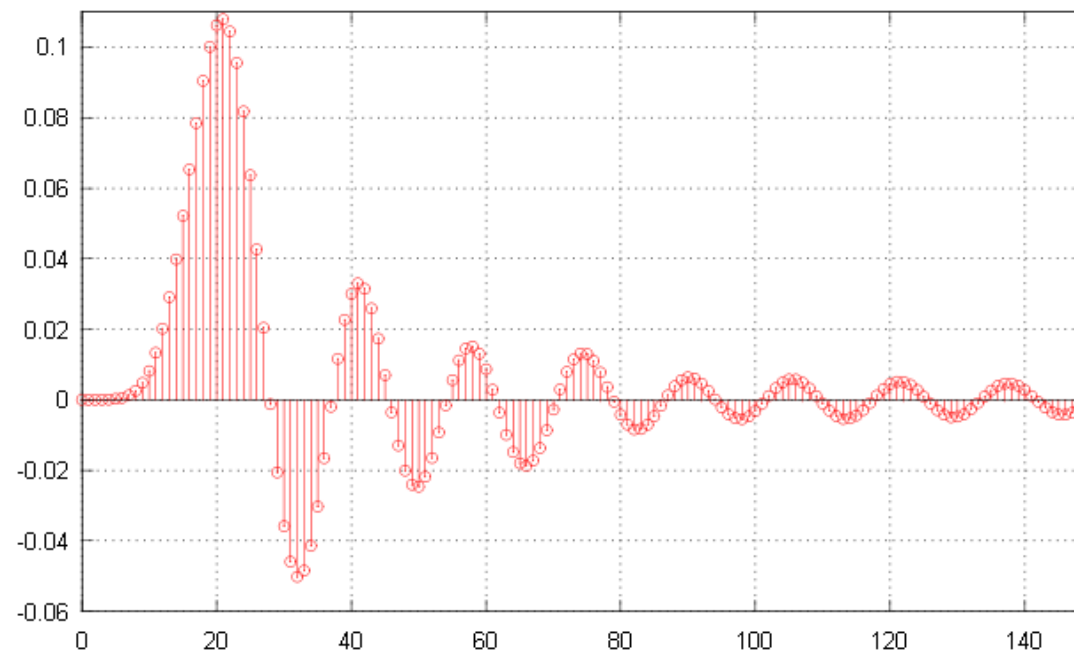
```
octave-3.2.4.exe:1> [n,wc] = cheblord(wd,ws,dd,ds);
```

```
octave-3.2.4.exe:1> [b,a] = cheby1(n,dd,wc);
```

- Generierung der Impulsantwort mit '**impz**', Darstellung als "stem"-Diagramm

```
octave-3.2.4.exe:1> L = 150; [x,t] = impz(b,a,L);
```

```
octave-3.2.4.exe:1> stem(t,x);
```



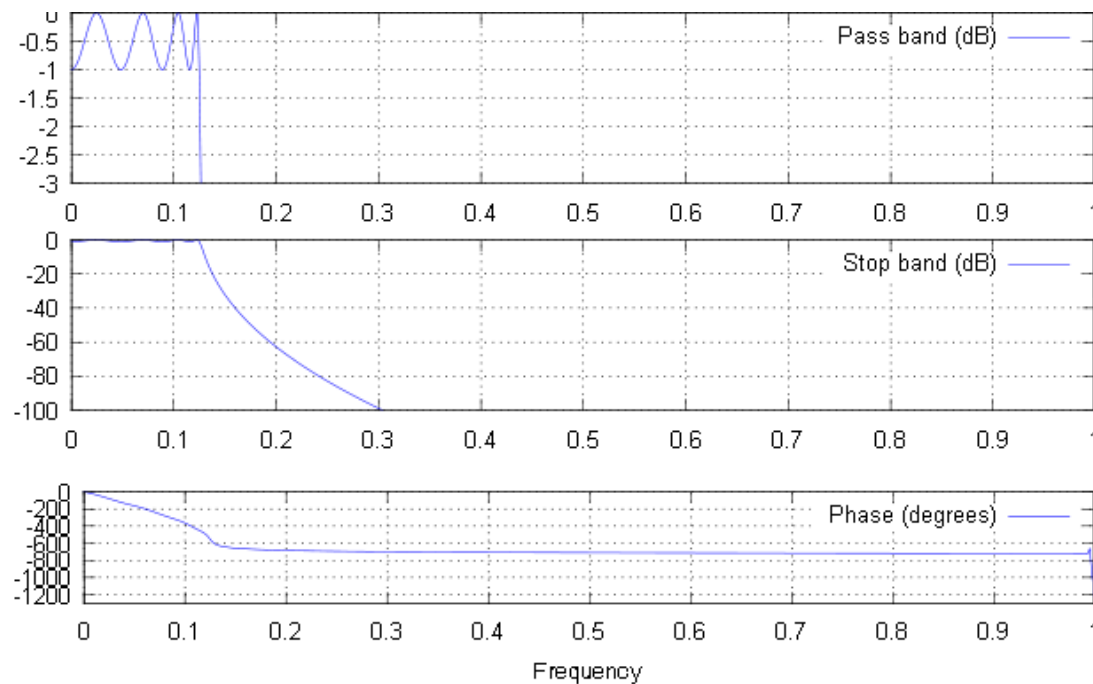
# OCTAVE-FILTERENTWURF: Tschebyscheff-Filter

## Tschebyscheff IIR Filter: Tiefpass Typ 1 (2)

- Darstellung von Frequenz- und Phasengang mit 'freqz'

```
octave-3.2.4.exe:1> FREQRES = 1024;
```

```
octave-3.2.4.exe:1> freqz(b,a,FREQRES);
```



# OCTAVE-FILTERENTWURF: Tschebyscheff-Filter

## Tschebyscheff IIR Filter: Tiefpass Typ 2 (Filtergrad n=8)

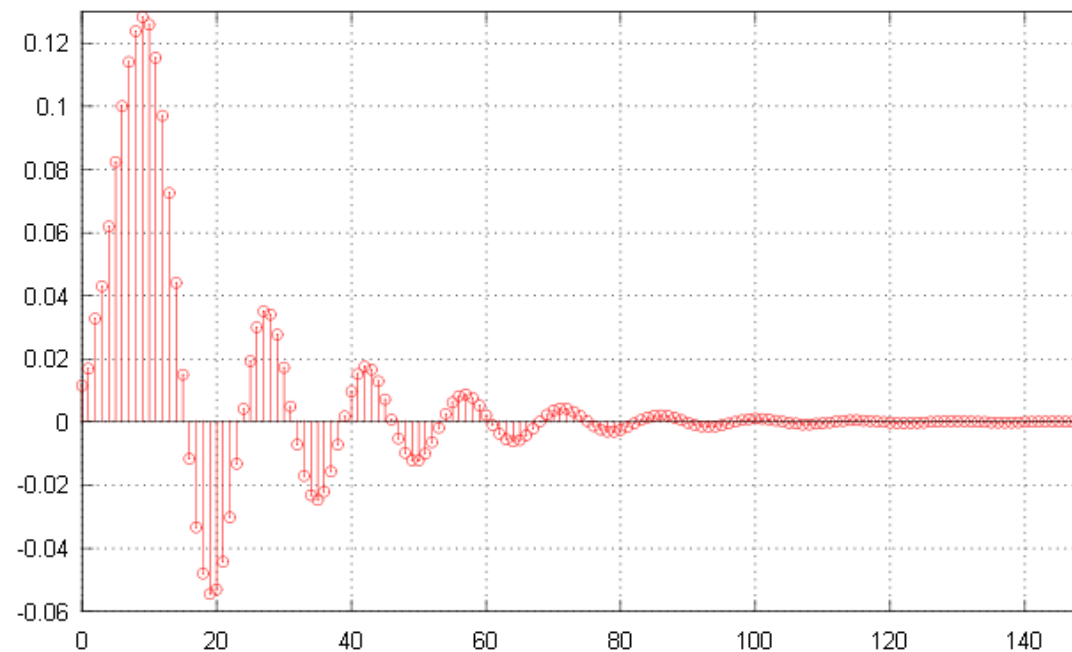
```
octave-3.2.4.exe:1> [n,wc] = cheb2ord(wd,ws,dd,ds);
```

```
octave-3.2.4.exe:1> [b,a] = cheby2(n,ds,wc);
```

- Generierung der Impulsantwort mit `'impz'`, Darstellung als "stem"-Diagramm

```
octave-3.2.4.exe:1> L = 150; [x,t] = impz(b,a,L);
```

```
octave-3.2.4.exe:1> stem(t,x);
```



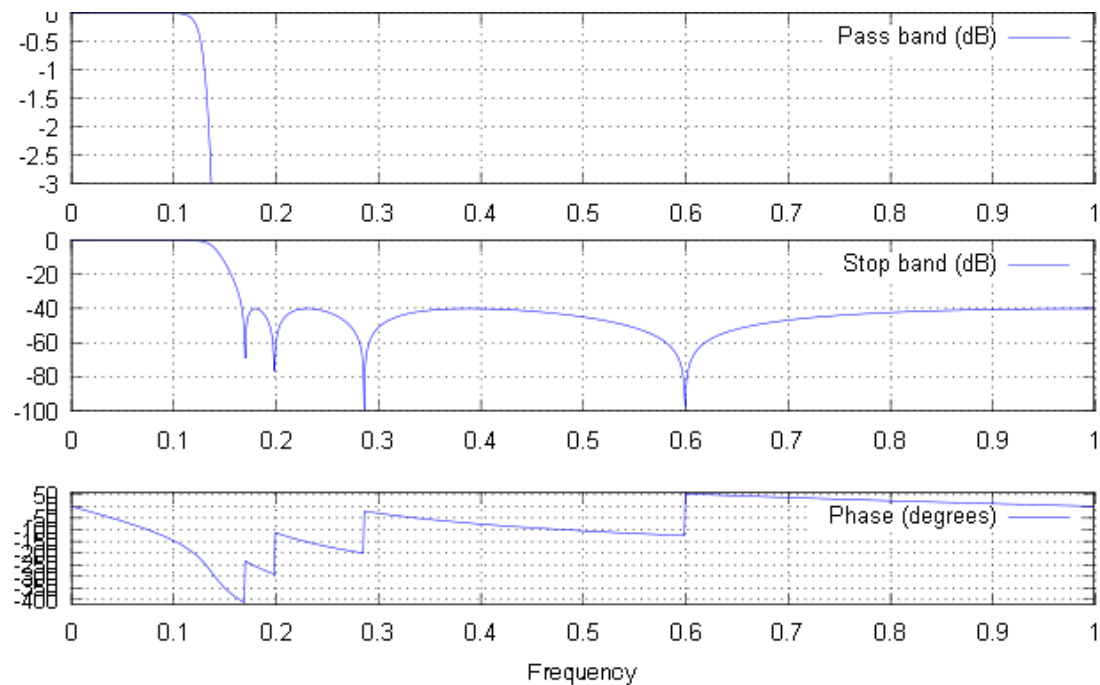
# OCTAVE-FILTERENTWURF: Tschebyscheff-Filter

## Tschebyscheff IIR Filter: Tiefpass Typ 1 (2)

- Darstellung von Frequenz- und Phasengang mit 'freqz'

```
octave-3.2.4.exe:1> FREQRES = 1024;
```

```
octave-3.2.4.exe:1> freqz(b,a,FREQRES);
```



# OCTAVE-FILTERENTWURF: Cauer-Tiefpass

## Cauer IIR Filter: Tiefpass (Filtergrad n=5)

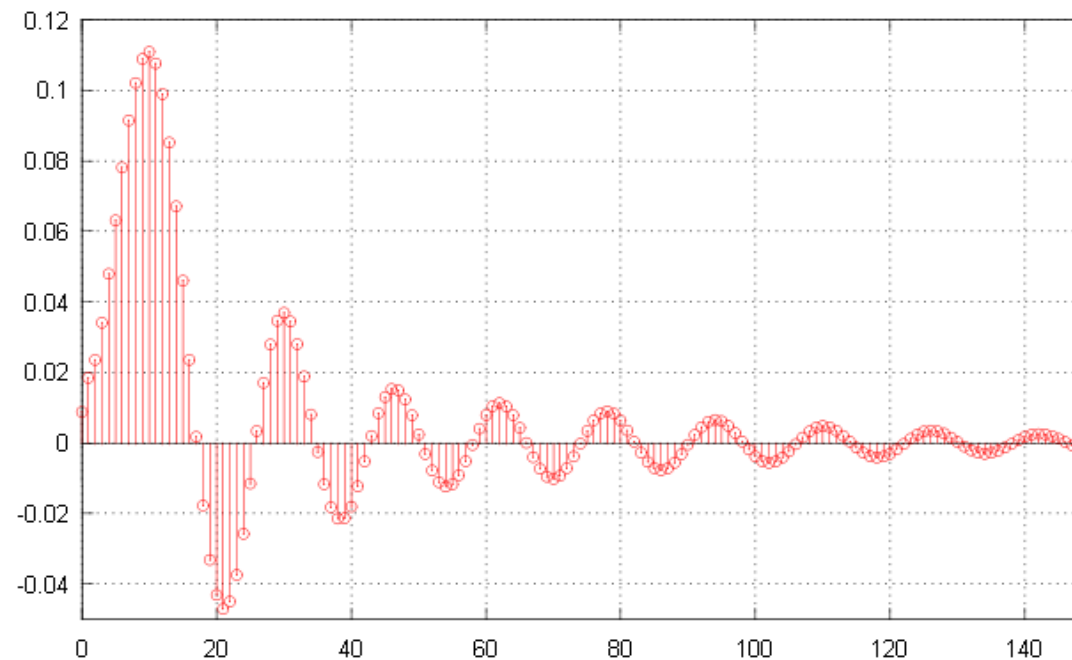
```
octave-3.2.4.exe:1> [n,wc] = ellipord(wd,ws,dd,ds);
```

```
octave-3.2.4.exe:1> [b,a] = ellip(n,dd,ds,wc);
```

- Generierung der Impulsantwort mit `'impz'`, Darstellung als "stem"-Diagramm

```
octave-3.2.4.exe:1> L = 150; [x,t] = impz(b,a,L);
```

```
octave-3.2.4.exe:1> stem(t,x);
```

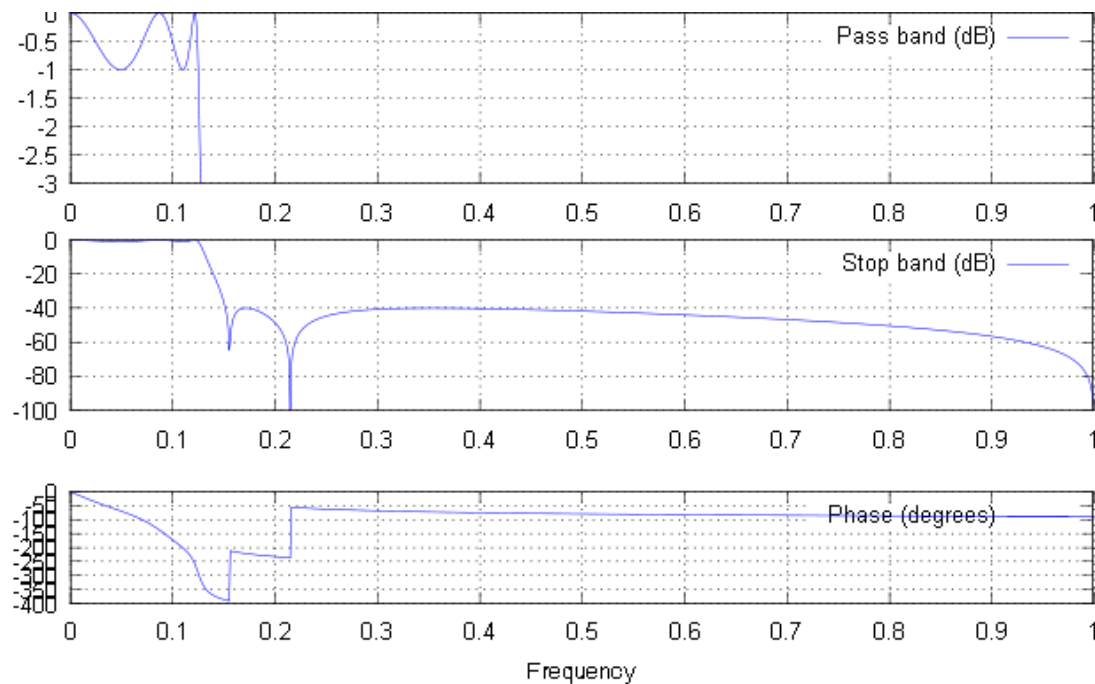


# OCTAVE-FILTERENTWURF: Tschebyscheff-Filter

## Cauer IIR Filter: Tiefpass (Filtergrad n=5)

- Darstellung von Frequenz- und Phasengang mit 'freqz'

```
octave-3.2.4.exe:1> FREQRES = 1024;  
octave-3.2.4.exe:1> freqz(b,a,FREQRES);
```



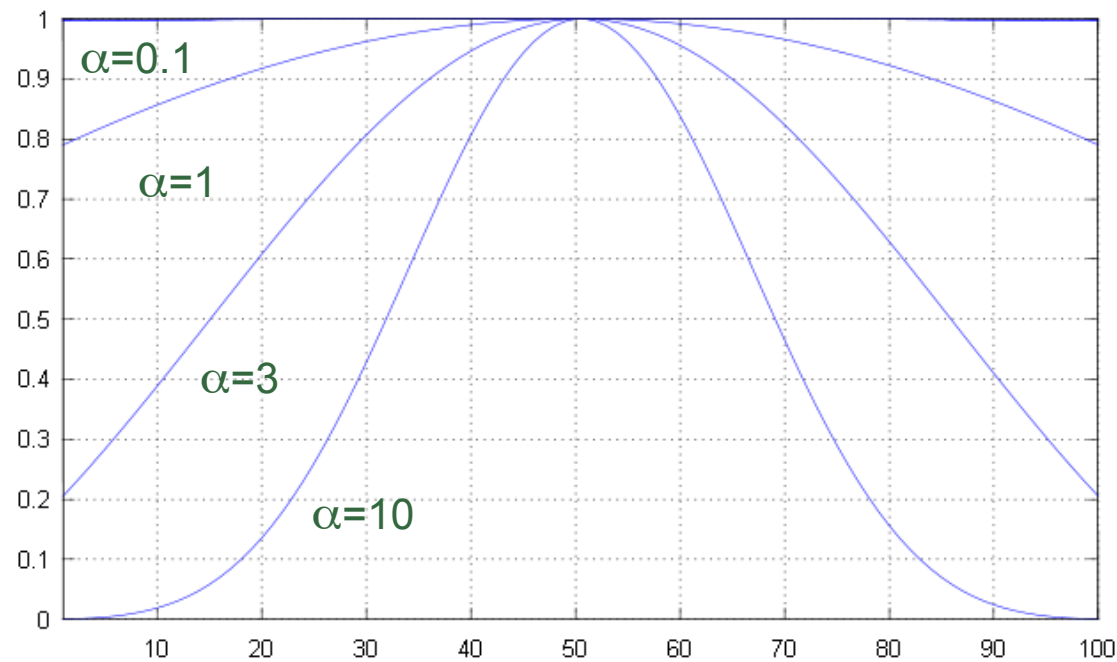
## **B) Finite Impulse Response (FIR)-Filter**

# OCTAVE-FILTERENTWURF: FIR-Filter

## Alternativen

- Entwurf mit "Fensterfunktion" (inv. Fouriertransformation IFFT)
- Entwurf mit Tschebyscheff-Approximation

Kaiser-Fenster für verschiedene  $\alpha$  (Fensterbreiten):





# OCTAVE-FILTERENTWURF: Entwurf mit Fensterfkt.

## Vorgehensweise: Entwurf eines Tiefpass-Filters

- Vorgabe eines Toleranzschemas
- Bestimmung der benötigten Filterlänge
- Bestimmung von  $\alpha$  ("Breite des Fensters")

- Beispiel:

```
octave-3.0.3.exe:1> wp = 1/8;  
octave-3.0.3.exe:1> ws = 1/6;  
octave-3.0.3.exe:1> delta = 0.01;  
octave-3.0.3.exe:1> [n, w, alpha, ftype] =  
                        kaiserord([wp,ws],[1,0],[delta,delta]);  
octave-3.0.3.exe:1> a = fir1(n,w,kaiser(n+1,alpha),ftype);
```

- Ergebnis:

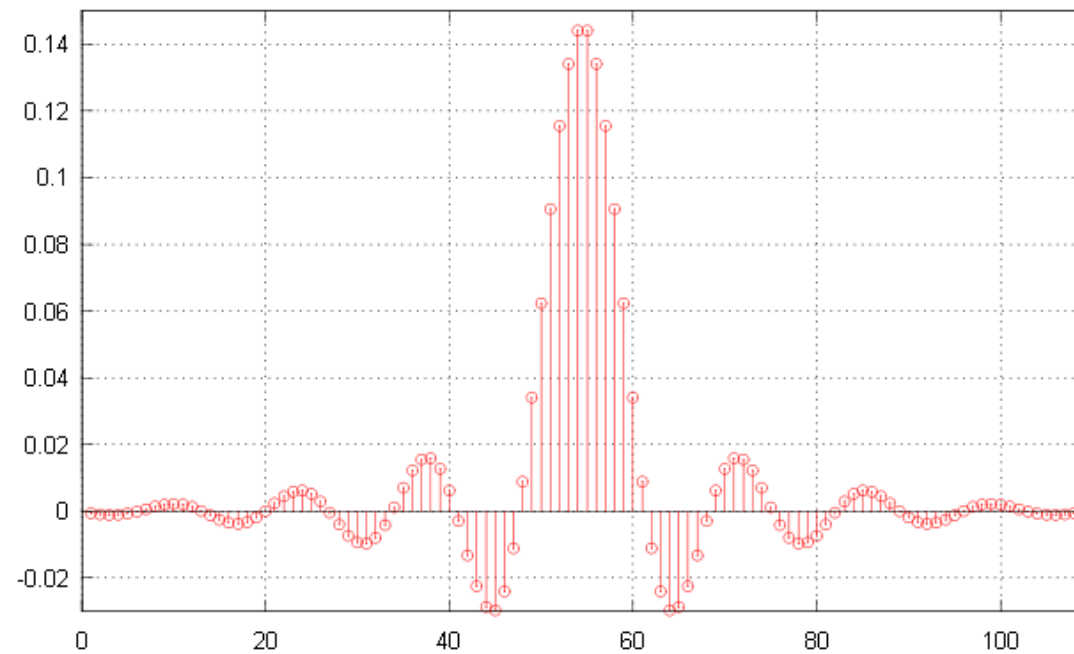
- $\alpha = 3.3953$
- $L = n+1 = 108$
- ftype = "low"

# OCTAVE-FILTERENTWURF: Entwurf mit Fensterfkt.

## Tiefpass-Filter mit Kaiserfenster

- Darstellung als "stem"-Diagramm

```
octave-3.2.4.exe:1> stem(a);
```

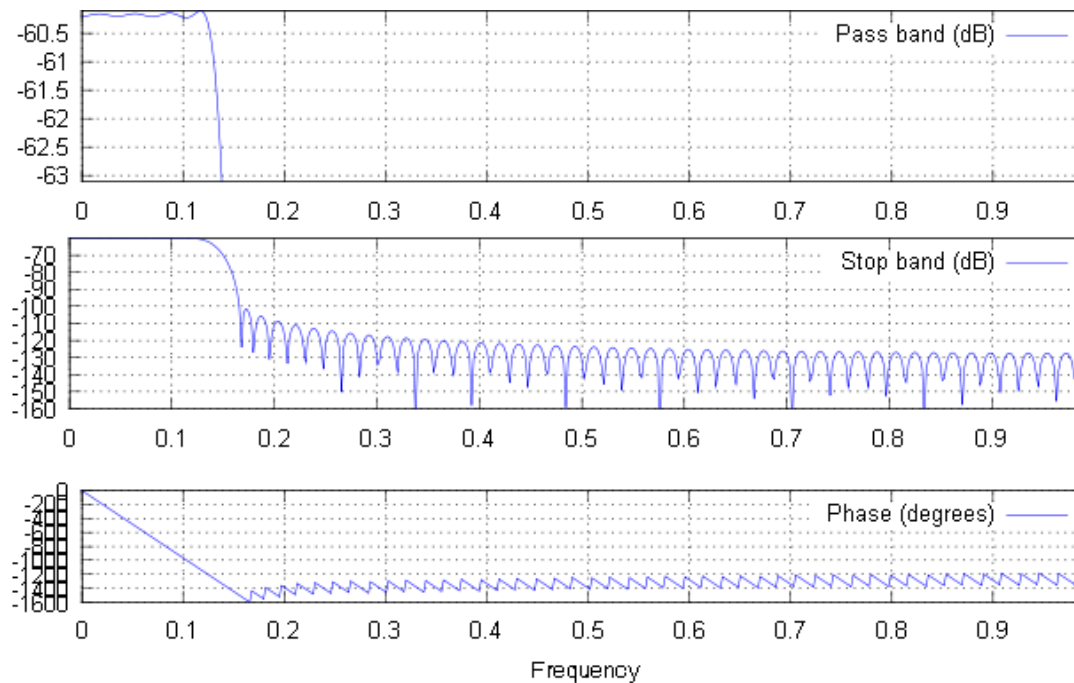


# OCTAVE-FILTERENTWURF: Entwurf mit Fensterfkt.

## Tiefpass-Filter mit Kaiserfenster (Filtergrad $n=108$ )

- Darstellung von Frequenz- und Phasengang mit 'freqz'

```
octave-3.2.4.exe:1> FREQRES = 1024;  
octave-3.2.4.exe:1> freqz(a,FREQRES);
```



# OCTAVE-FILTERENTWURF: Tschebyscheff-Approx.

## Ansatz

- Bestmögliche Ausnutzung des Toleranzschemas
- Numerische Optimierung: Minima, Maxima:
  - "Remez-Exchange"
  - "Parks-McClellan"

- Beispiel:

```
octave-3.2.4.exe:1> wp = 1/8;  
octave-3.2.4.exe:1> ws = 1/6;  
octave-3.2.4.exe:1> d1 = 0.05;  
octave-3.2.4.exe:1> d2 = 0.01;  
octave-3.2.4.exe:1> FREQRES = 1024;  
octave-3.2.4.exe:1> n=ceil((-10*log10(d1*d2)-13)/  
    (2.324*pi*(ws-wp))); % Approximation!!!  
octave-3.2.4.exe:1> a=remez(n,[0 wp ws 1],[1 1 0 0],[1 d1/d2]);
```

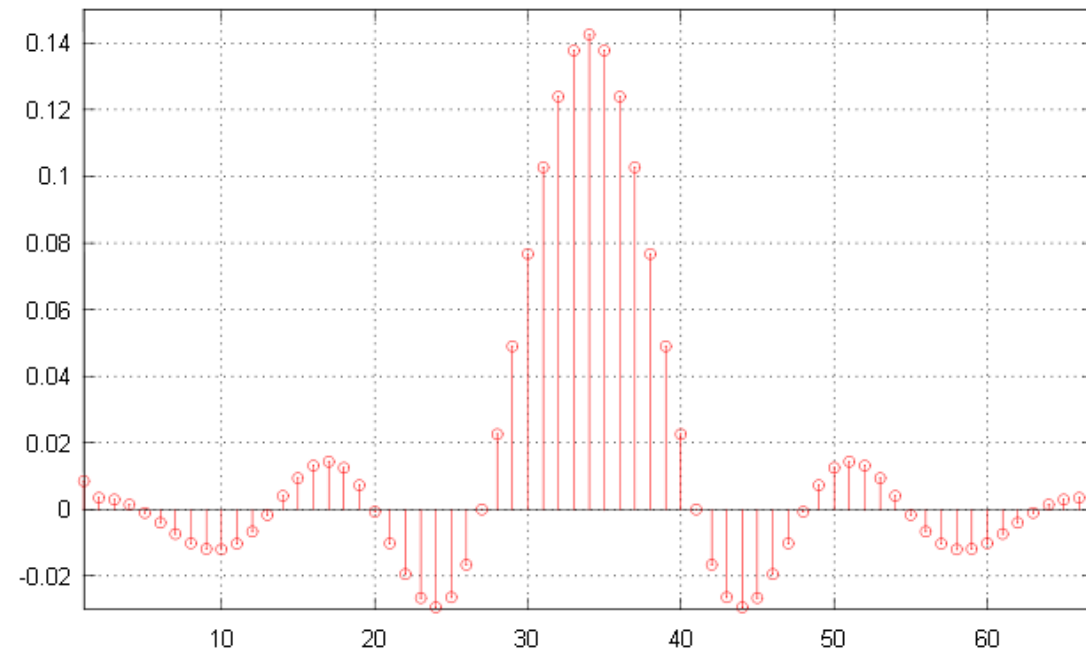
- Hier: Filterordnung  $n = 66$

# OCTAVE-FILTERENTWURF: Tschebyscheff-Approx.

## Filter mit Tschebyscheff-Approximation

- Darstellung als "stem"-Diagramm

```
octave-3.2.4.exe:1> stem(a);
```

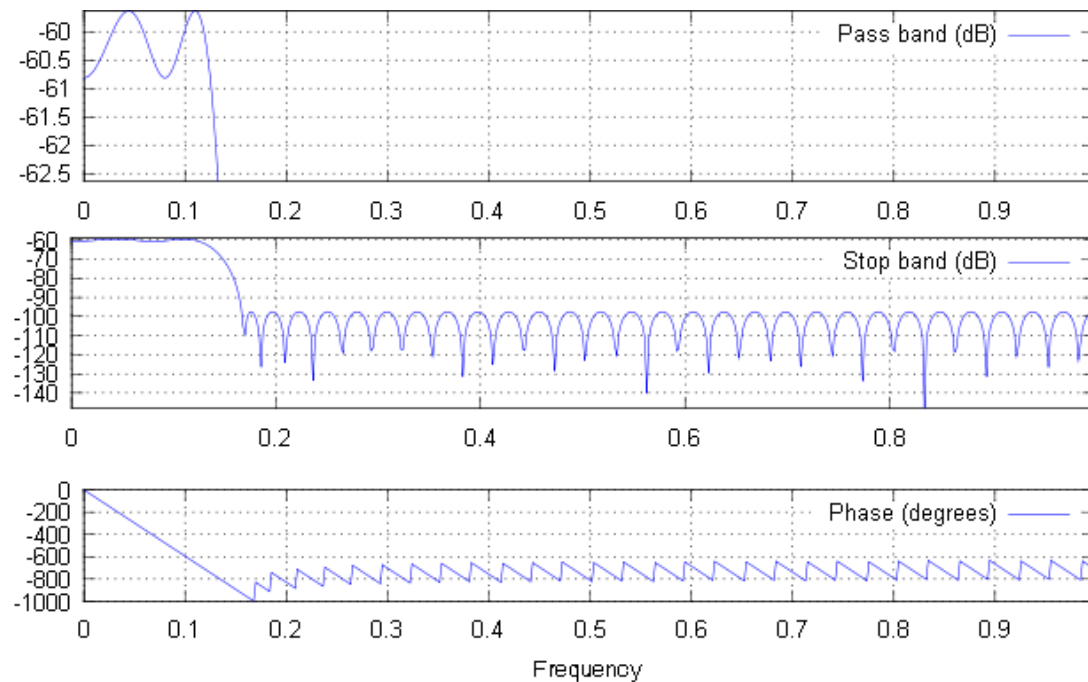


# OCTAVE-FILTERENTWURF: Tschebyscheff-Approx.

## Filter mit Tschebyscheff-Approximation (Filtergrad n=66)

- Darstellung von Frequenz- und Phasengang mit 'freqz'

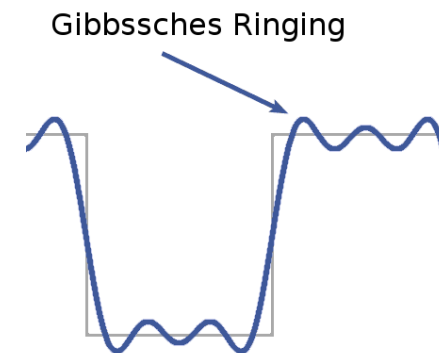
```
octave-3.2.4.exe:1> FREQRES = 1024;  
octave-3.2.4.exe:1> freqz(a,FREQRES);
```



# OCTAVE-FILTERENTWURF: Entwurf mit Fensterfkt.

## Grundidee

- Vorgabe eines "Wunschfrequenzgangs"
- Berechnung einer Impulsantwort durch inverse Fouriertransformation (IFFT)
- Begrenzung auf "endliche Länge" durch Multiplikation mit einer Fensterfunktion
- Problem bei Unstetigkeiten des Wunschfrequenzgangs:
  - Der „Gibbs-Effekt“ (oder „Ringing“) bezeichnet das typische Verhalten von Fourierreihen in der Nähe von Sprungstellen; an den Unstetigkeitsstellen ergeben sich Unter/Überschwinger (ca. 10% bei einem Rechteckfenster; bleibt auch, wenn man die Reihe „verlängert“); dieses Phänomen sollte jedoch nicht mit dem Überschwingen von Signalen verwechselt werden!
  - Verbesserungsmöglichkeit: Verwendung besserer Fensterfunktionen
- Häufig wird das sogenannte "Kaiser-Fenster" für den Filterentwurf verwendet



# OCTAVE-FILTERENTWURF: Kaiserfenster

**[n, Wn, beta, ftype] = kaiserord(f, m, dev [, fs])**

- Die Funktion `kaiserord` berechnet die notwendigen Werte zur Parametrisierung eines FIR-Filters (`fir1`) unter Verwendung eines entspr. Kaiserfensters
  - n: Filterordnung (Länge des Filters minus 1)
  - Wn: Grenzfrequenz(en) für den FIR-Filter
  - beta: Parameter für ein Kaiserfenster mit Länge n+1
  - ftype: Filtertyp (Bandpass, Bandsperre, ...)
- f: Trennfrequenzen, paarweise angegeben (1 Paar für Hochpass/Tiefpass, 2 Paare für Bandpass/Bandsperre); die 2 Werte dienen zur Angabe der Steilheit des Übergangsbereiches (obere, untere Grenze)
- m: Filterklasse; [1 0] für Tiefpass, [0 1] für Hochpass, [0 1 0] für Bandpass, [1 0 1] für Bandsperre
- dev: Zulässige Abweichung (Toleranz) im Bereich 0...1, je ein Wert pro Band (2 Werte für Hoch-/Tiefpass, 3 Werte für Bandpass/-sperre)
- fs: Abtastfrequenz; wird verwendet um Frequenzen auf den Bereich [0, 1] zu skalieren (1 repräsentiert dabei die sog. "Nyquist-Frequenz"  $fs/2$ )



# OCTAVE-FILTERENTWURF: Kaiserfenster (2)

**[n, Wn, beta, ftype] = kaiserord(f, m, dev [, fs])**

## Hinweise

- Octave verwendet gleiche Toleranzen pro Band (der geringste angegebene Wert) – damit genügt auch die Angabe eines Skalars!
- Experimentieren Sie mit verschiedenen Toleranzen; je geringer die Toleranz desto höher die Filterordnung – was allerdings nicht heissen muss, dass damit das Ergebnis besser ist (speziell beim am stärksten verrauschten Signal in Übung 4 kann das festgestellt werden...)

### ■ Beispiel 1:

```
octave-3.2.4.exe:1> b = fir1(n,Wn,kaiser(n+1,beta),ftype, 'noscale');
```

### ■ Beispiel 2:

```
octave-3.2.4.exe:1> [n, w, beta, ftype] = kaiserord([1000 1200], [1 0],  
[0.05 0.05], 11025);  
octave-3.2.4.exe:1> freqz(fir1(n,w,kaiser(n+1,beta),ftype,'noscale'),  
1,[],11025);
```

# OCTAVE-FILTERENTWURF: Entwurf mit Fensterfkt.

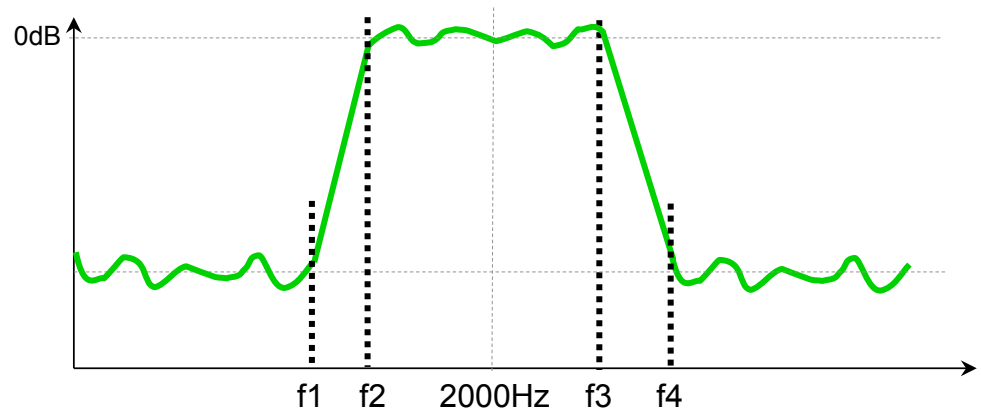
## Zu Übung 4: Entwurf eines Bandpass-Filters mit Kaiserfenster zur Entfernung von Artefakten

- Signalträger: Audiosignal mit 2000Hz (siehe Angabe Ü3) bei geg. Abtastrate  $sr$ 
  - Grenzfrequenzen  $f_1$ ,  $f_2$ ,  $f_3$ ,  $f_4$
- Vorgabe eines Toleranzschemas ( $t_1$ ,  $t_2$ ,  $t_3$ ) (experimentell ermitteln!)
- Signalbasis nach 0 verschieben
- Anwendung FIR-Filter
- Mögliche Lösung:

...

```
[n,Wn,beta,ftype] = kaiserord([f1 f2 f3 f4],[0 1 0],[t1 t2 t3],sr);  
bpsignal=filter(firl1(n,Wn,ftype,kaiser(n+1,beta),'noscale'),1,x);
```

...



# OCTAVE-FILTERENTWURF: Entwurf mit Fensterfkt.

## Zu Übung 4: Entwurf eines Tiefpass-Filters mit Kaiserfenster zur Signalglättung/Demodulation

- Demodulation: Tiefpass-Filter bei dot-Frequenz; dot-Länge 60ms → Tiefpass bei  $1000/60=16,6\text{Hz}$  → Grenzfrequenzen  $f_1, f_2$
- Vorgabe eines Toleranzschemas ( $t_1, t_2$ ) (experimentell ermitteln!)
- Anwendung FIR-Filter
- Mögliche Lösung:

...

```
[n,Wn,beta,ftype] = kaiserord([f1 f2],[1 0],[t1 t2],sr);  
tpsignal=filter(fir1(n,Wn,ftype,kaiser(n+1,beta),'noscale'),1,bpsignal);
```

...

## **Übung “Pervasive and Embedded Systems”, 2013S**

### **PROJECT SIGNALVERARBEITUNG (AUDIO-, IMAGE), I/O, FILTER**

**Institute for Pervasive Computing**  
Johannes Kepler University  
Dipl.-Ing. Dr. Andreas Riener  
riener@pervasive.jku.at

