



Einführung in Matlab

5. Funktionen

Prof. Dr. Christiane Zarfl, Dipl.-Inf. Willi Kappler, Prof. Dr. Olaf
Cirpka



- wie Sie durch Skripte Befehlsfolgen wiederverwendbar machen.
- wie Sie häufig vorkommende Berechnungen in Algorithmen formulieren.

Wie kann ich eigene Funktionen für verschiedene Eingaben erstellen und damit Berechnungsschritte wiederverwenden?



- können Sie interaktive Eingaben anfordern und Ausgaben erstellen.
- können Sie eigene Funktionen definieren und z.B. in Skripten aufrufen/wiederverwenden.



- Ausgabe eines Textes im Command-Window:
 - `disp` steht für “display”
 - Text-Strings sind von Hochkommata umschlossen
 - `disp` versteht auch Vektoren: `disp(['He' 'llo' 'World'])`
- Interaktive Eingabe:
 - `a = input('Bitte Halbwertszeit [Tage]
für Photoabbau eingeben: ')`
 - erzeugt den Text auf dem Bildschirm und wartet, bis eine Eingabe abgeschlossen ist (Return beendet die Eingabe).
 - Der eingegebene Wert steht dann in der Variable `a`.



- Beispiel:

- `jn = input('Abbruch? (j/n) ','s')`
- Zusatzargument 's' erklärt, dass Ergebnis als Text-String zu lesen ist (selbst wenn der String aus Ziffern besteht).
- Umgang mit Fehlern der Nutzer \Rightarrow typische Schleife, bis richtige Antwort kommt:

```
1  jn = ''; % Initialisierung
2  while jn~='j' & jn~='n'
3      jn = input('Abbruch?_(j/n)_','s')
4  end
5  disp('Ihre_Eingabe:')
6  disp(jn)
```



Überlagerung von zwei Sinusschwingungen (schwing.m)

- Darzustellende Funktion:
- $y(x) = a_1 \sin(fx + \phi_1) + a_2 \sin(fx + \phi_2)$
- Interaktive Eingabe:
 - gemeinsame Frequenz f
 - Amplitude der ersten Komponente a_1
 - Amplitude der ersten Komponente a_2
 - Phasenwinkel der ersten Komponente ϕ_1
 - Phasenwinkel der zweiten Komponente ϕ_2
- Graphische Ausgabe im Intervall $[-\pi, +\pi]$



- “Idee” eines Skriptes: ersetzt manuelle Eingabe einer Befehlsfolge:
 - Variablen stehen auch nach Ausführung des Skriptes zu Verfügung.
 - Wenn Variablenwerte im Skript verändert werden, sind sie auch außerhalb des Skriptes verändert.
 - Ein Skript kann ein anderes Skript aufrufen.
 - mit gemeinsamer Nutzung des Arbeitsspeichers.
 - Nachteile eines Skriptes:
 - Allgemeiner Arbeitsspeicher wird mit Zwischengrößen “zugemüllt”.
 - Variablenbezeichnung muss über alle Skripte hinweg konsistent sein (**Vorsicht!** Seiteneffekte).



- “Idee” einer **Funktion**: “Input \Rightarrow Output”-Beziehung:
 - Von der Außenwelt ist nur das bekannt, was als Eingabeargument(e) übergeben wird.
 - Der Außenwelt wird lediglich das Ergebnis (Ausgabeargument(e)) mitgeteilt.
 - Innenwelt der Funktion ist von außen nicht einsehbar.
 - Eine Funktion hat ihren eigenen Speicherbereich.



- Modularisierung:
 - Unterteile ein großes Projekt in viele Einzelschritte und stelle Methoden für die Einzelschritte zur Verfügung.
 - Eigener Namensraum, in verschiedenen Funktionen kann es die lokale Variable i oder t geben.
- Wiederverwendbarkeit:
 - Identischer Einzelschritt kann mehrfach, auch in anderen Projekten, verwendet werden.
- Einkapselung:
 - Wenn eine Funktion funktioniert, interessiert mich nicht ihr Innenleben, und ich will nicht mit Zwischenergebnissen belästigt werden.
 - **Black Box** Prinzip (s.a. [Information Hiding](#))



Funktionen - Definition in Matlab:

- Definiert in der ersten Zeile
- `function [out1,out2] = myfunction(in1,in2)`
 - Erstes Wort = Schlüsselwort `function`
 - Liste der **Eingabeparameter** (in eckigen Klammern, also als Vektor).
 - Name der Funktion mit Liste der **Ausgabeparameter** in runden Klammern.
- Die Schnittstelle ist definiert über die Reihenfolge der Argumente, nicht über die Namen.
 - **Vorteil:** Variablennamen im Innern können sich von den Variablennamen außen unterscheiden.
 - **Nachteil:** Man muss sich die Reihenfolge merken.
 - Voraussetzung für allgemeine Wiederverwendbarkeit.



- Kommentarzeilen direkt unter der Kopfzeile werden bei `help Funktionsname` ausgegeben (z.B. `help myfunction`):

```
1 function [out1,out2] = myfunction(in1,in2)
2     % Loest die Weltformel
3     % Verwendung:
4     % [out1,out2] = myfunction(in1,in2)
5     % in1: erster Eingabeparameter [Einheit]
6     % in2: zweiter Eingabeparameter [Einheit]
7     % out1: ...
8     % written by Max Mustermann,
9     % Maerz 30, 2016
10 end
```



Bogenmaß in Grad umwandeln:

- Funktion `phi_deg = r2d(phi)` soll einen Winkel, der in Bogenmaß ($0 - 2\pi$) gegeben ist, in Gradmaß ($0-360^\circ$) umwandeln.



Mittelwert und Standardabweichung:

- Funktion `[m,s] = mittel_standardabw(x)` soll Mittelwert und Standardabweichung des Vektors `x` berechnen. Verwenden Sie für Ihre Funktion einfach die in Matlab implementierten Befehle `mean` und `std`.
- Schreiben Sie zusätzlich ein Skript, das **interaktiv** nach der Anzahl `n` normal verteilter Zahlen mit vorgegebenem Mittelwert `m_init` und Standardabweichung `s_init` fragt, diese generiert und die Funktion `mittel_standardabw` aufruft und ausgibt.
- Tipp: normal verteilte Zufallszahlen generieren: `x = m_init + s_init * randn(n,1);`



- Abstand zwischen einem Punkt (x_0, y_0) zu einem (oder vielen) Punkt(en) (X, Y) :

$$r = \sqrt{(x - x_0)^2 + (y - y_0)^2}$$

- Funktionskopf: `r = dist(x0, y0, X, Y)`
- Schreiben Sie die Funktion `dist` in eine Datei `dist.m`.
- Schreiben Sie ein Skript, in welchem Sie:
 - den Punkt (x_0, y_0) interaktiv eingeben.
 - die Matrizen **X** und **Y** auf einem regelmäßigen Gitter erzeugen (`meshgrid`).
 - Ihre Abstandsfunktion `dist` aufrufen.
 - Das Ergebnis als `contour`-Grafik darstellen.



- Wie kann ich eigene Funktionen nutzen, um Differentialgleichungen mit Matlab zu lösen?