

Übung 1

Modellierung, Simulation und Optimierungsverfahren II

Dr. Colin W. Glass

Institut für Höchstleistungsrechnen

09.04.2014

Übung 1: Strategie

Ein frischgebackener, berufstätiger Ingenieur ist auf Wohnungssuche in Timbuktu. Der Ingenieur kann maximal 10 Wohnungen besichtigen und jeweils:

- Die Güte der Wohnung quantifizieren (eine reelle Zahl)
- Die Wohnung nehmen oder verwerfen (er ist ja ein berufstätiger Ingenieur)

Frage: Wie soll der Ingenieur vorgehen (Strategie bzw. Verfahren), um die Wahrscheinlichkeit, die beste Wohnung zu wählen, zu maximieren?

Übung 1: Strategie

Ein frischgebackener, berufstätiger Ingenieur ist auf Wohnungssuche in Timbuktu. Der Ingenieur kann maximal 10 Wohnungen besichtigen und jeweils:

- Die Güte der Wohnung quantifizieren (eine reelle Zahl)
- Die Wohnung nehmen oder verwerfen (er ist ja ein berufstätiger Ingenieur)

Frage: Wie soll der Ingenieur vorgehen (Strategie bzw. Verfahren), um die Wahrscheinlichkeit, die beste Wohnung zu wählen, zu maximieren?

Wichtig:

- Die Güte erlaubt nur einen Vergleich zwischen Wohnungen. Der absolute Wert hat keine Bedeutung (Kein Vorwissen).
- Eine Wohnung kann nur direkt nach der Besichtigung genommen werden - eine verworfene Wohnung ist endgültig weg.

Bonusaufgabe Übung 1: Strategie

Frage: Wie soll der Ingenieur vorgehen (Strategie bzw. Verfahren), um eine möglichst gute Wohnung zu kriegen?
(Ziel: Erwartungswert des Güte-Rangs minimieren.)

Matlab/Octave

Extrem mächtig! Optimal für Leute die quantitativ arbeiten wollen, denn:

- Code direkt auf der Command line ausführbar (kein Compilieren usw. erforderlich)
- Sehr intuitiv, z.B. müssen Variablen nicht initialisiert werden.
- Mächtiges Set an built-in Funktionen, direkt einsetzbar
- Eigene Funktionen und Skripte erzeugen
- Spielend leicht Plots und sonstige Visualisierungen generieren

Im Vergleich zu anderen Programmiersprachen sehr leicht zu lernen ...

Matlab

- GUI
- Super Debugger
- Simulink
- Toolboxes
- Generell teuer(!), für Studenten meist umsonst

Octave

- Opensource, freeware (GNU)
- Reduzierte Funktionalität
- GUI sehr mühsam einzubinden

Aber: Jederzeit (und überall) einsetzbar (und installierbar)!

Erste Schritte

Wenn ihr das Programm startet, könnt ihr sofort loslegen mit der **Command Line**.

- Interaktiv (Befehle ausführen)
- Volle Funktionalität
- Jede neue Zuweisung (an eine Variable) wird angezeigt, ausser der Befehl endet mit ";"
- Rückgabevariable nicht definiert: Variable "ans" wird verwendet
- Funktionen, Skripte usw werden im aktuellen Ordner und in allen Ordnern in "path" gesucht.
- Bei Unsicherheiten: help Name
- z.B. help path

Variablen

Es gibt verschiedene Arten von Variablen:

- Zahlen (int8, int16 ... double ...). Default: double
- Text (char, string)
- Logical (true, false)
- NaN (Not A Number)

P.s: Variablen (und Funktionen) müssen nicht deklariert werden.

Zahlen

Zahlen können in beliebig grosse Matrizen beliebiger Dimension abgelegt werden:

- $a = 1$
- $a = [1,2,3]$ oder $[1 \ 2 \ 3]$ (Spaltenvektor)
- $a = [1;2;3]$ (Reihenvektor)
- $a = [1,2,3 ; 4,5,6]$ (2D Matrix)
- $a = \text{ones}(2,2,2)$ (3D Matrix, Länge 2 pro Dimension)

Index

Sämtliche Datenstrukturen haben Dimensionen, die durch Indizes adressiert werden können.

- $a(:)$ → alle Elemente (Reihen, Kolonnen)
- $a(:, :)$ → alle Elemente
- $a(:, 3)$ → die ganze dritte Kolonne
- $a(1:2:end)$ → jedes zweite Element

Indexspielereien sind das Herzstück der Matlab/Octave Programmierung.

Wichtige Operatoren

Elementweise:

- Logische Operatoren: | (oder), & (und), ! (nicht)
- Vergleiche: > (größer), < (kleiner), <=, >=, != (nicht gleich), == (gleich)
- Inkrementieren: ++, --
- Ausserdem lassen sich z.B. Matrixmultiplikationen elementweise durchführen mit einem Punkt vor dem Operator (.* oder .^)

"Short Circuit": für logisches und/oder gibt es Operatoren, die alle Elemente auswerten und nur wenn alles zutrifft "wahr" zurückgeben:

- Logische Operatoren: || (oder), && (und)

Anweisungen (Kontrollfluss)

- Logische Unterscheidung:

if (Bedingung)

 "Was machen"

elseif (Bedingung)

 "Was anderes machen"

else

 "Noch was anderes machen"

end

- Schleifen:

for zaehlVar

 "Was machen"

end

while (Bedingung)

 "Was machen"

end

Buchstaben

Buchstaben werden in so genannten "strings" gespeichert:

- `a = "H"`
- `a = "Hallo Welt"`
- `a = ["Hallo " "Welt"]`
- `a = ["Hallo " num2str(1) "ste Welt"]`

Wir sehen hier schon: `[ ]` ist eine Funktion, nämlich `cat`. Später mehr. `num2str` verwandelt Zahlen zu Buchstaben.

Weitere Datentypen

- `a = cell (N, M)`
- `a = cellstr (STRING)`
- `a = struct ("field", VALUE, "field", VALUE, ...)`

`cell` und `cellstr` haben verschiedene Felder, jedes Feld kann eine Matrix bzw. ein String sein.

Ein Element kriegen: `a{x,y}`

`struct` hat verschiedene Felder, die jeweils irgendwas sein können.

Ein Element kriegen: `a.field`

Wichtige Funktionen

Viele Funktionen sind bereits vorhanden (eine Matlab Toolbox ist z.B. in erster Linie eine Sammlung von Funktionen):

- Erzeugen: ones, zeros, rand, randn, ...
- Manipulieren: find, sort, max, min, diff, perm, ...
- Runden: floor, ceil, round ...
- Visualisieren: plot, scatter, hist, ...
- Eigenschaft prüfen: isempty, isnan, isstr, ismember, ...
- Konvertieren: str2num, num2str, ...
- Datenmanagement: print, load, fprintf, ...
- ...

Dies ist gewissermassen der Wortschatz. Es gibt noch VIEL mehr ...

Funktionen verwenden

Eine Funktionen wird aufgerufen durch:

$$[\text{Out1}, \text{Out2}, \dots] = \text{NameFunc} (\text{In1}, \text{In2}, \dots)$$

Den meisten Funktionen können verschieden viele Inputs und Outputs übergeben werden. Das Verhalten kann sich dadurch ändern!

Funktionen verwenden

Sich änderndes Verhalten (bei nur einem Output kann [] weggelassen werden), z.B:

`Out1 = find (In1)`

liefert die Indizes aller Werte in `In1` $\neq 0$, 1D durchnummeriert.
Folglich: `In1(Out(3))` gibt den dritten Wert $\neq 0$

`[Out1, Out2] = find (In1)`

liefert die Indizes aller Werte in `In1` $\neq 0$, 2D durchnummeriert (Zeilen, Spalten).
Folglich: `In1(Out1(3),Out2(3))` gibt den dritten Wert $\neq 0$

Funktionen verwenden

Wird eine Funktion aufgerufen, ohne explizit die Outs zu benennen, verhält sich die Funktion so, als wäre nur ein Output festgelegt worden.

Dies erlaubt es, Funktionen zu schachteln. Out1 der inneren Funktion wird auf In1 der äusseren gepiped. Z.B:

```
Wieviele = length(find(In1))
```

verhält sich gleich wie:

```
temp = find(In1)
```

```
Wieviele = length(temp)
```

Indexspielereien

$\text{IndNonZero} = \text{find}(A)$

IndNonZero ist ein Vektor mit den Indices der Werte. Mit den Indices können nun die "gefundenen" Werte (also $\neq 0$) in A adressiert werden:

$A(\text{IndNonZero}) += 1$

(Achtung: $+=$ geht nicht in allen Versionen von Matlab und Octave.)
Hier werden also alle Werte $\neq 0$ um 1 hochgezählt.

Ausserdem ist IndNonZero selber ein Vektor ...:

$A(\text{IndNonZero}(2:5)) += 1$

Hier wird der zweite bis fünfte Wert $\neq 0$ um 1 hochgezählt.

Eigene Funktionen

Eigene Funktionen zu schreiben ist sehr leicht.

- Textdatei: "NameFunc.m"
- erste Zeile:
function [Out1,Out2,...] = NameFunc (In1,In2,...)

P.s.: Funktionen haben eine lokale Sicht auf Variablen. Nach Beendigung der Funktion sind alle Variablen, die nicht zurückgegeben werden, weg.

Funktionen schreiben...

Eine Funktion (oder ein Skript) ist ganz gewöhnlicher Text und kann folglich in jedem Text Editor geschrieben werden.

```
function [out1, out2] = doStuff (in1, in2)
% In dieser Funktion produzieren wir Output auf Basis
% des Inputs. Dies ist keine wirklich sinnvolle Funktion.
if in1  $\geq$  in2
    out1 = in1 * in2;
    out2 = in1 - in2;
else
    out1 = in1/in2;
    out2 = in2 - in1;
end
```

Funktionen: Kommentieren

Generell sollte man Code kommentieren. Dies ist vorzugsweise direkt bei der entsprechenden Codezeile anzubringen. Ausserdem:

```
function [out1, out2] = doStuff (in1, in2)
```

Die übergebenen Variablen (*in1*, *in2*, *out1*, *out2*) können irgendwas sein. Einzelner Wert, Vektor, Matrix bestehend aus Integer, Double, Boolean usw. Für gewöhnlich schreibt man direkt unter der Deklaration ein Block Kommentar, in dem das alles erklärt wird. Also:

```
% in1, in2, out1, out2 sind einzelne Integer. in1 ist *Erklärung* ...
```

Dieser erste Kommentarblock wird zurückgeliefert, wenn man die Hilfsfunktion aufruft: `help doStuff`

Eigene Skripte

Skripte verhalten sich exakt gleich, wie wenn die enthaltenen Befehle konsekutiv in die Commandozeile eingegeben werden.

- Textdatei: "NameSkript.m"
- Befehle eintippseln

Nomenklatur

P.s: Versucht NIE einer eigenen Funktion, Skript oder Variable einen bereits vergebenen Namen zuzuweisen (ausser ihr wollt sie overloaden - eher die Ausnahme).

Endet meist in falschen Daten, verpassten Deadlines und Tränen.

Beispiel: Benford'sche Zahlenverteilung

% Dies ist ein Skript

Werte = 100 * ones(10000,1);

Prozente = 1 + (1+rand(10000,1)*2) / 100;

% Prozente ist im Bereich [1.01, 1.03]

for ind = 1:1000

Werte = Werte .* Prozente;

end

res = zeros(10000,1);

for ind = 1:10000

temp = num2str(Werte(ind));

res(ind) = str2num(temp(1));

end

Beispiel: Benford'sche Zahlenverteilung

hist (res,9)

