Kurze Einführung in MATLAB

Markus Grasmair

Computational Science Center,
Universität Wien

8. März 2010

Übersicht

Was ist MATLAB

- MATLAB = MATrix LABoratory.
- Programmiersprache und -umgebung für technische Berechnungen.
- Verbreitet in numerischer Mathematik, wissenschaftlichem Rechnen, Simulation, Datenanalyse, Computergrafik.

Wie bekommt man MATLAB?

- Installiert in den Computerräumen der Mathematik (Linux!).
- Studentenversionen über ZID der TU Wien erhältlich.
- Octave: Freie open-source Software (GPL), großteils kompatibel mit MATLAB. Erhältlich unter `http://www.gnu.org/software/octave/`

Was kann MATLAB (nicht)

Positiv:

- Berechnungen mit Matrizen und Vektoren.
- Einfache graphische Ausgabe.
- Einfach (intuitiv) zu programmieren.

Negativ:

- Größere Berechnungen langsam (Programme werden nicht kompiliert).
- Schlechte Unterstützung für symbolisches Rechnen.

Übersicht

Definition von Variablen

- Variable = Platzhalter für Daten, etwa: x, y, *temp*.
- Variablen werden durch Zuweisung eines Wertes erzeugt:
  - `x = 1;` — Typ: Zahl
  - `y = [0.5 -0.5; 1 2];` — Typ: Matrix
  - `name = 'Ausgabetext'` — Typ: Text
- Zusammenfassung mehrerer Befehle in einer Zeile möglich. Trennung mit , (Ausgabe) oder ; (keine Ausgabe).
  - `x=1, y=2`

    ```
    x = 1
    y = 2
    ```
  - `x=1; y=2;`

Vektoren

Zeilenvektoren:

```
x = [0.5 -1]
x =
0.5000 -1.0000
```

Spaltenvektoren:

```
x = [0.5; -1]
x =
0.5000
-1.0000
```

Regelmäßige Vektoren mittels `Anfang:[Inkrement:]Ende`

```
x = 1:4
x =
1   2   3   4
x = 1:0.3:2
x =
1.0000   1.3000   1.6000   1.9000
```

Matrizen

Eingabe wie Vektoren:

```
A = [1 2;3 4]
A =
1   2
3   4
```

Spezielle Matrizen:

- `eye(m,n)` ... $m \times n$ Matrix mit Einsern auf Hauptdiagonale. `eye(n)` = `eye(n,n)`.
- `zeros(m,n)` ... $m \times n$ dimensionale Nullmatrix.
- `ones(m,n)` ... $m \times n$ dimensionale Matrix, alle Einträge 1.
- `rand(m,n)` ... $m \times n$ dimensionale Matrix, Einträge zwischen 0 und 1 gleichverteilte Zufallszahlen.

Dimensionen von Variablen

Länge von Vektoren mittels `length` bestimmbar:

```
x = 1:3; length(x)
ans =
3
```

Dimension von Matrizen mittels `size`:

```
A = [1 2 3; 4 5 6]; size(A)
ans =
2  3
```

Wird `length` auf eine Matrix angewandt, erhält man die größte Dimension: `A = [1 2 3; 4 5 6]; length(A)`

```
ans =
3
```

Zugriff auf Elemente

Ist `x` ein Vektor, so bezeichnet:

- `x(i)` ... den i-ten Eintrag von `x`,
- `x(i:j)` ... den Vektor der Einträge von i bis j,
- `x(i:end)` ... den Vektor der Einträge ab i,
- `x([i j])` ... die Einträge i und j.

Ist `A` eine Matrix, so bezeichnet:

- `A(i,j)` ... den (i,j)-ten Eintrag von `A`,
- `A(i,:)` ... die i-te Zeile von `A`,
- `A(:,j)` ... die j-te Spalte von `A`.

Einfache Operationen

Addition, Subtraktion und Multiplikation (auch skalare Multiplikation) mittels $+$, $-$, $*$, Transposition mit .' ($+$ komplexe Konjugation mit '), Potenzieren mit ^.

```
A=[1 2; 3 4]; B=[0 1; 3 2];
```

```
A+B
ans =
1   3
6   6
```

```
A-B
ans =
1   1
0   2
```

```
A*B
ans =
 6   5
12  11
```

```
A'
ans =
1   3
2   4
```

```
B^2
ans =
3   2
6   7
```

```
-A
ans =
-1  -2
-3  -4
```

```
2*A
ans =
2   4
6   8
```

Komponentenweise Operationen

Wird den Operationen `*`, `^` und `/` (für Division) ein `.` vorangestellt, so werden die entsprechenden Operationen *elementweise* statt auf die ganze Matrix angewendet:

```
A=[1 2; 3 4]; B=[0 1; 3 2];
```

```
A.*B
ans =
0  2
9  8
```

```
A./B
ans =
Inf  2
1    2
```

```
A.^2
ans =
1   4
9  16
```

```
A.^B
ans =
1    2
27  16
```

Elementare Funktionen

Elementare Funktionen wie Sinus, Cosinus, Exponentialfunktion und Logarithmus sind in MATLAB bereits implementiert. Unvollständige Listen mittels: `help elfun`, `help specfun`.

Anwendung der Funktionen auf eine Zahl gibt das erwartete Ergebnis, Anwendung auf Matrizen ist *immer* komponentenweise, die Funktion `expm` liefert das Exponential einer quadratischen Matrix (also $\sum_i A^i/i!$):

```
exp(1)
ans =
2.7183
```

```
exp([1 2; 3 4])
ans =
 2.7183   7.3891
20.0855  54.5982
```

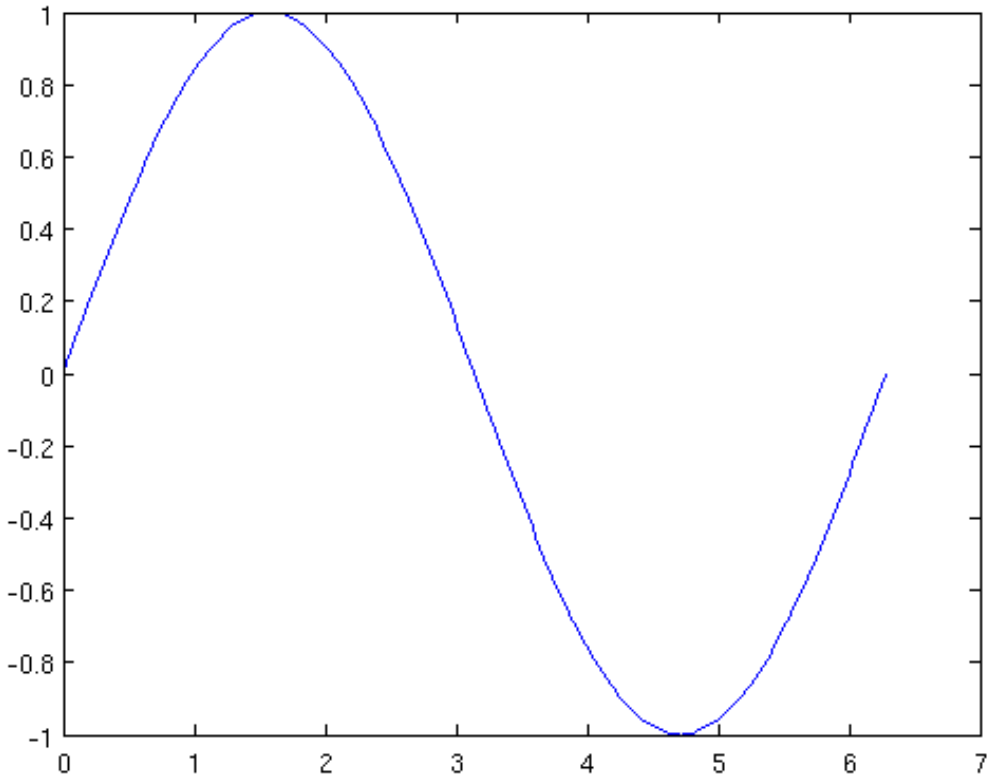
```
expm([1 2; 3 4])
ans =
 51.9690   74.7366
112.1048  164.0738
```

Beispielgrafik

Als Beispiel plotten wir den Sinus im Intervall $[0, 2\pi]$.

- Diskretisierung der x-Achse: `x = 0:pi/100:2*pi;`
- Definition der Funktionswerte: `y = sin(x);`
- Plot der Funktion: `plot(x,y);`

Beispielgrafik — erster Versuch

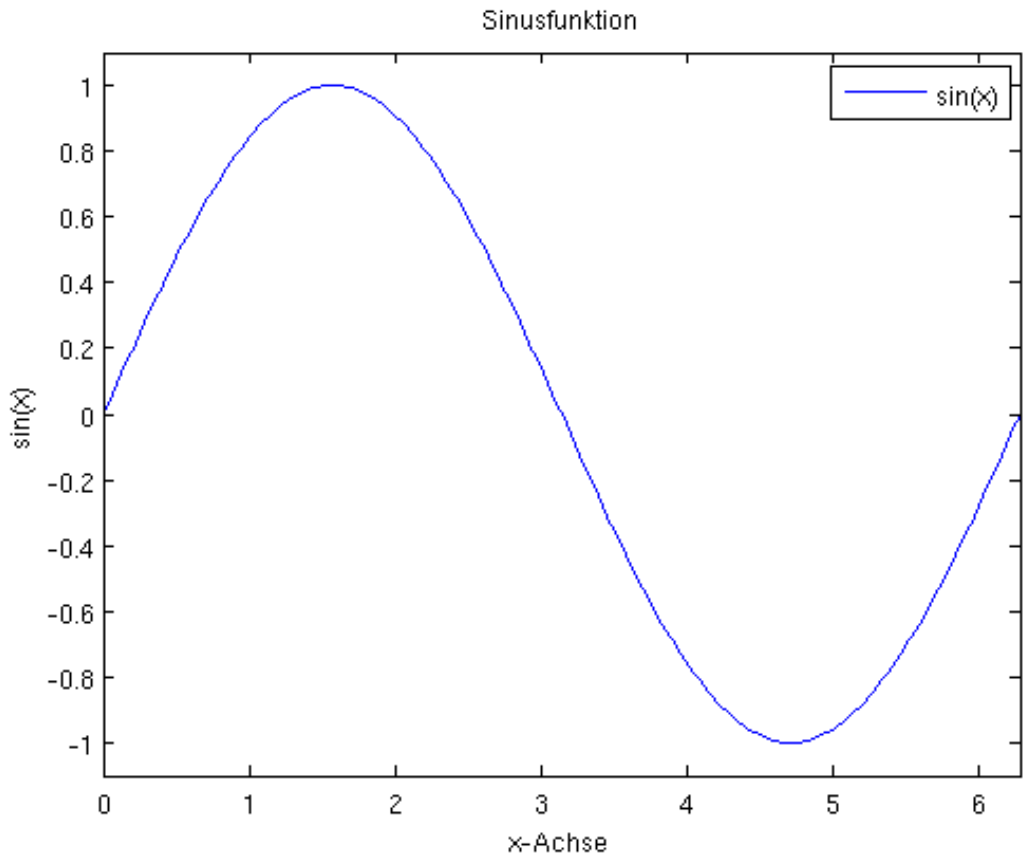


Beispielgrafik — Verfeinerung

- Diskretisierung der x-Achse: `x = 0:pi/100:2*pi;`
- Definition der Funktionswerte: `y = sin(x);`
- Plot der Funktion: `plot(x,y);`
- Verbesserung des Plots mit Titel, Achsenbeschriftung, Achsenskalierung,...

```
title('Sinusfunktion')
xlim([0 2*pi]);ylim([-1.1 1.1]);
xlabel('x-Achse');ylabel('sin(x)');
legend('sin(x)','Location','NorthEast');
```

Beispielgrafik — zweite Version



Übersicht

m-files

Erstellung eigener Programme in (Text)Dateien mit der Endung `.m`.
Beispiel: einfaches Programm, das die Summe zweier Variablen berechnet.

- Öffnen und Erstellen der Datei:
  ```
  edit mysum.m
  ```
- In der Datei `mysum.m` wird die Funktion definiert, etwa:
  ```
  function result = mysum(x,y)
  % Dies ist ein Kommentar
  result = x+y;
  ```
- Aufruf der Funktion in MATLAB liefert:
  ```
  mysum(1,2)
  ans =
  3
  ```

Aufgabenstellung

Die folgenden Beispiele zeigen, wie einfache Programme in MATLAB aussehen könnten.

Als Beispiele wurden Mittelwert und Varianz eines Vektors gewählt, also für $x = (x_1, \ldots, x_n)$

$$\text{mean}(x) := \frac{1}{n} \sum_{i=1}^{n} x_i \,,$$

$$\text{var}(x) := \frac{1}{n} \sum_{i=1}^{n} \big(x_i - \text{mean}(x)\big)^2 \,.$$

Programmierung über Schleifen

Beispielcode:

```
function output = mymean_loop(x)
% mymean_loop(x) berechnet den Mittelwert von x
output = 0;
for i = 1:length(x)
  output = output+x(i);
end
output = output/length(x);
```

Vektorisierung

In MATLAB ist es fast immer zu empfehlen, Schleifen wenn möglich zu vermeiden.

Im Beispiel des Mittelwertes ist dies etwa möglich mittels der Funktion `sum`:

```
function output = mymean(x)
% mymean(x) berechnet schnell den Mittelwert von x
output = sum(x)/length(x);
```

Laufzeitvergleich

Die Laufzeit eines Prozesses können wir mit den Befehlen `tic` (Beginn des timers) und `toc` (Ende des timers) messen:

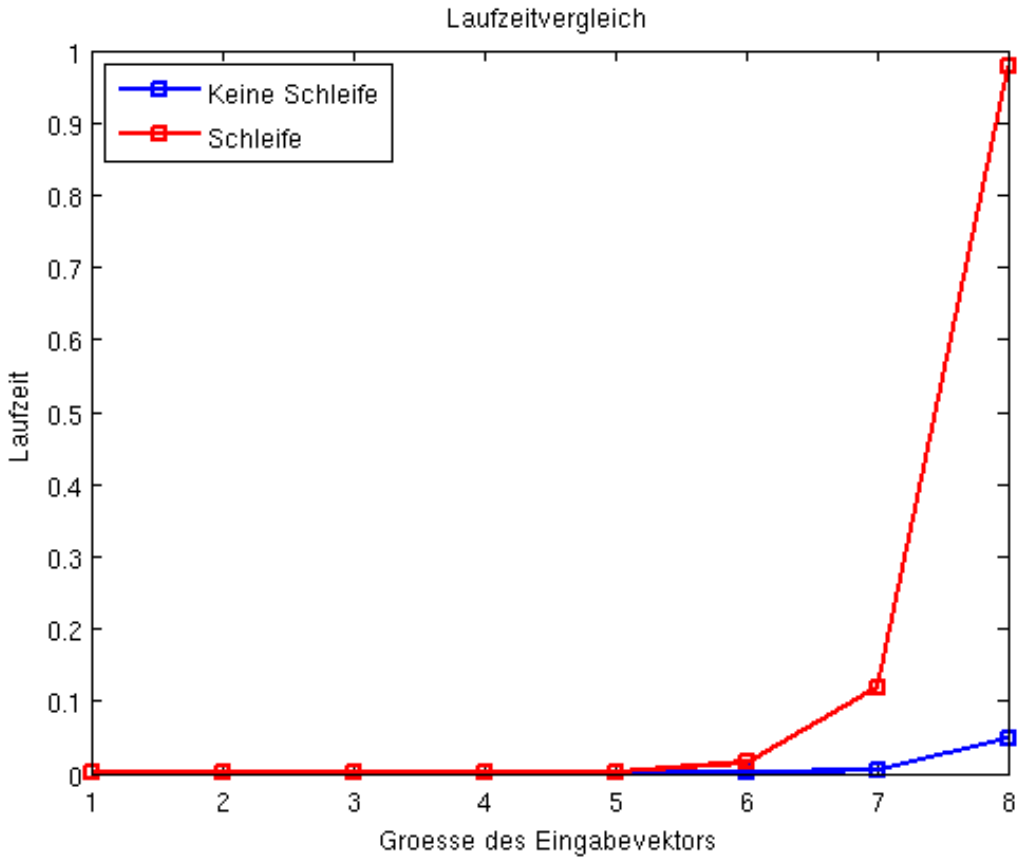
```
time_loop = zeros(1,8);
time_noloop = zeros(1,8);
for i=1:8
  x = 1:(10^i);
  tic;mymean_loop(x);time_loop(i)=toc;
  tic;mymean(x);time_noloop(i)=toc;
end
```

Plot der Ergebnisse

Eine einfache Grafik liefert:

```
plot(1:8,time_noloop,'-rs',...
    'LineWidth',2,'Color','b');
hold on;
plot(1:8,time_loop,'-rs',...
    'LineWidth',2,'Color','r');
xlabel('Groesse des Eingabevektors');
ylabel('Laufzeit');
legend('Ohne Schleife','Schleife','Location','NorthWest');
title('Laufzeitvergleich');
hold off;
```

Plot der Ergebnisse

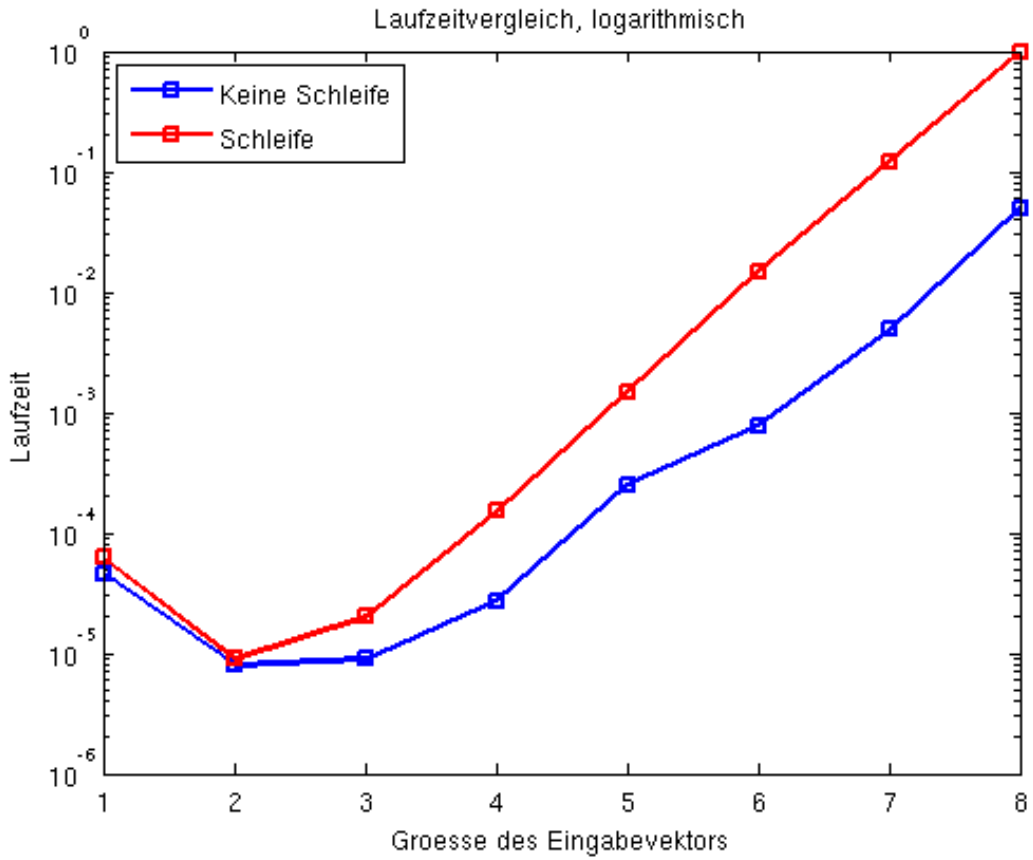


Besserer Plot der Ergebnisse

Um die Sichtbarkeit zu verbessern, verwenden wir eine logarithmische Darstellung:

```
semilogy(1:8,time_noloop,'-rs',...
    'LineWidth',2,'Color','b');
hold on;
semilogy(1:8,time_loop,'-rs',...
    'LineWidth',2,'Color','r');
xlabel('Groesse des Eingabevektors');
ylabel('Laufzeit, logarithmisch');
legend('Ohne Schleife','Schleife','Location','NorthWest');
title('Laufzeitvergleich');
hold off;
```

Plot der Ergebnisse



Programmierung über Schleifen

Berechnung der Varianz eines Vektors:

```
function output = myvar1(x)
% Varianz mit Schleife
output = 0;
m = mean(x);
for i = 1:length(x)
    output = output+(x(i)-m)^2;
end
output = output/length(x);
```

Zwei vektorisierte Programme

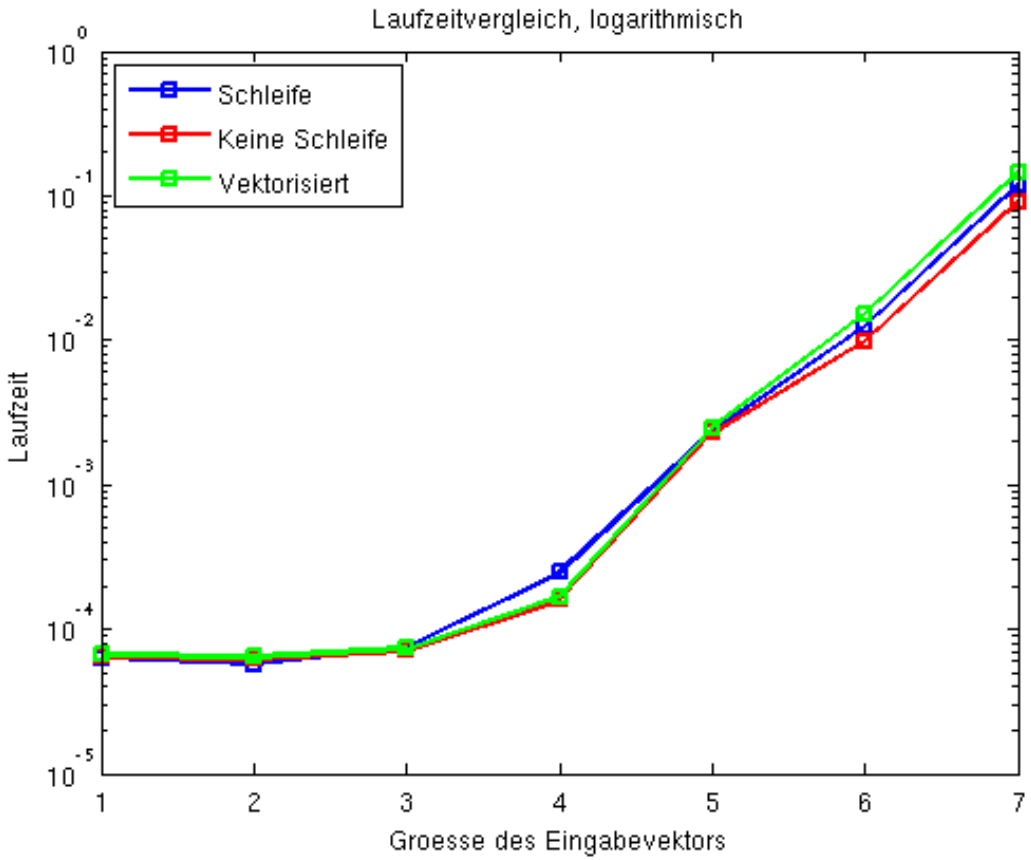
Analog zum Mittelwert:

```
function output = myvar2(x)
% Varianz ohne Schleife
m=mean(x);
output = sum((x-m).^2)/length(x);
```

Auch denkbar wäre:

```
function output = myvar3(x)
% Varianz voll vektorisiert
m=mean(x);
output = ((x-m)*(x-m)')/length(x);
```

Laufzeitvergleich



Sicherheitsabfragen

Um dem Anwender es möglichst schwer zu machen, das Programm falsch zu verwenden, sollte man zusätzliche Abfragen einbauen:

```
function output = myvar4(x)
% Varianz ohne Schleife
if(~isvector(x))
  error('Eingabe muss ein Vektor sein')
end
if(length(x) <= 1)
  error('Der Eingabevektor muss mindestens zwei
Elemente beinhalten')
end
m=mean(x);
output = sum((x-m).^2)/length(x);
```

Übersicht

MATLAB-Dokumentation

In MATLAB erhält man Hilfe über:

- `help` +Befehl . . . Dokumentation des Befehls.
- `help matlab/general` . . . wenn man ganz verloren ist.
- `doc` . . . selbiges in einem Hilfefenster.
- `demos` . . . wie der Name sagt: Demos.

Außerdem gibt es:

- Cleve Moler, Numerical Computing with MATLAB: `http://www.mathworks.com/moler/chapters.html`
- Harald Führer, Tutorium zu Numerische Mathematik: Dienstag, 11:00 (im Anschluss an die Vorlesung), Computerraum C2.04.

Typische Fehler

- ??? Undefined function or variable 'mytest'.
  - Die eigenen Funktionen werden nur dann gefunden, wenn sie im *Current Folder* stehen $\Longrightarrow$ Pfad wechseln.
  - Variablen, die in einer Funktion definiert werden, sind *lokal*, die Kommandozeile weiß also nichts von ihrer Existenz. Umgekehrt kennen die Funktionen die Variablen, die in der Kommandozeile definiert wurden, nicht.
- ??? Error using ==> mtimes
  Inner matrix dimensions must agree.
  - Auch MATLAB kann Matrizen nur dann multiplizieren, wenn die Dimensionen kompatibel sind.

Tips

- Kommentieren Sie Ihre Programme! Das erleichtert die Fehlersuche und das spätere Editieren (durch Sie oder andere) deutlich.
- Testen Sie Ihre Programme zunächst anhand von einfachen Beispielen, bei denen Sie das Ergebnis bereits kennen.
- Vermeiden Sie unnötige Schleifen, die das Programm verlangsamen.
- Vermeiden Sie jedenfalls *Endlos*schleifen, die das Programmen *sehr stark* verlangsamen.
- Versuchen Sie, möglichst viele Benutzerfehler schon am Beginn des Programms abzufangen.
- Bedenken Sie immer die Existenz von Rundungsfehlern.