

Mathematik A für die Molekulare Biotechnologie, WS 2002/3

SCILAB/MATLAB Kurs Übungsblatt

Öffnen Sie den SCILAB Kursaccount im CIP-Pool, indem Sie den Usernamen `simwerk02` und das Passwort `12bock34` eingeben. Öffnen Sie dann ein Terminal, indem Sie unten mitte-links auf das Terminalzeichen klicken, so dass eine sogenannte UNIX-Shell erscheint. Gehen Sie, wenn Sie am ersten Kurs teilnehmen, mit dem Befehl `cd scilab-2.6/poolXX` in das Unterverzeichnis, das Ihrem Rechnernamen entspricht (im zweiten Kurs bitte durch `cd scilab-2.6/POOLXX`), und öffnen Sie mit `scilab` eine SCILAB-Benutzeroberfläche (die man auch manchmal "SCILAB-Shell" nennt). Der Name SCILAB steht für "Science Laboratory" und ist dafür gedacht, alle erdenklichen Computerberechnungen schnell und einfach durchzuführen. SCILAB ist einem kommerziellen Programm namens MATLAB ganz direkt nachempfunden, aber im Gegensatz zu diesem frei verfügbar. Es wurde in einem Forschungsinstitut in Paris entwickelt. Alle in diesem Kurs besprochenen SCILAB Kommandos sind identisch auch in MATLAB ausführbar; in Ausnahmefällen wird der äquivalente MATLAB-Befehl extra angegeben (Wollen Sie MATLAB im CIP-Pool starten, tippen Sie statt `scilab` einfach `matlab`, um eine MATLAB-Benutzeroberfläche zu starten).

Vektoren und vordefinierte Funktionen:

1. Tippen Sie in der Benutzeroberfläche `a=1.2e3` ein. Dadurch wird eine Variable namens `a` und mit dem Eintrag `1.2e3` angelegt. Diese Variable kann man sich wieder angucken, indem man einfach `a` eingibt.
2. Tippen Sie nun `a=1.3;` ein. Was ist der Unterschied zu vorher? Welchen Wert hat die Variable `a` nun? *Das Semikolon am Ende unterdrückt die Bildschirmausgabe und ändert sonst nichts.*
3. Was passiert, wenn man `neuevariable1=a` eingibt? *Variablenamen können beliebig lang sein und Ziffern enthalten. Sie müssen nur mit einem Buchstaben anfangen.*
4. Tippen Sie nun z.B. `b=[1, 2.5 15];` ein. Was entsteht? Die so erzeugte Liste von Zahlen nennt man einen Vektor. Gucken Sie sich den Vektor an, indem Sie `b` tippen. *Vektoren werden in eckige Klammern eingeschlossen, und das Komma trennt dabei die Einträge. Das Komma kann auch durch Leerzeichen ersetzt werden.*
5. Greifen Sie mit `b(2)` auf das zweite Element des Vektors zu.
6. Mit `length(b)` erhalten Sie die Länge des Vektors, also die Zahl seiner Elemente.
7. "Transponieren" Sie `b` durch die Operation `c=b'`. Dadurch wird nur die Orientierung des Vektors geändert. (Vektoren können "liegen" oder "stehen". Das wird erst im Zusammenhang mit Matrizen wichtig.)
8. Was erhält man mit `c(2)`?
9. Erzeugen Sie einen liegenden Vektor `[1,2,...,20]` durch Eingabe von `d=[1:20]`
10. Erzeugen Sie nun einen Vektor `[1,3,5,...,19]` durch Eingabe von `d=[1:2:19]`. Der Wert in der Mitte bezeichnet die Schrittweite, mit der vom linken zum rechten Wert gegangen werden soll.

11. Welchen Wert erhält man mit `length(d)`?
12. Erzeugen Sie einen Vektor `t=[0:0.1:10]`.
13. Was erzeugt `f=sin(t)`? *Viele SCILAB Funktionen (exp, log, abs, sqrt, sign, round, ...) wirken **einzel**n auf die Vektorelemente.*
14. Tippen Sie `help sin` ein. Mit `help` können Sie jederzeit Hilfe über eine beliebige SCILAB-Funktion bekommen.
15. Plotten Sie den Sinus, indem Sie `plot(t,f)` eingeben. Was erzeugt `plot(f)`? Wo liegt der Unterschied?
16. Was erzeugt `plot([0,1,2], [1,-1, 1])`?
17. Wenn Sie Befehle wiederholen möchten, können Sie einfach die "Cursor nach oben"-Taste drücken. Führen Sie z.B. `plot([0,1,2], [1,-1, -3])` aus, indem Sie die letzte Zahl der obigen Zeile löschen. Dies ist eine sehr praktische Einrichtung.
18. Mit `whos -type constant` (MATLAB: `whos`) können Sie ansehen, welche Variablen Sie noch im Speicher haben. (Es werden noch einige von SCILAB vordefinierte Variable mit gezeigt)
19. Mit `clear` können Sie alle von Ihnen definierten Variablen wieder löschen.
20. Tippen Sie `disp('hallo')`. Was passiert?
Bemerkung: Der Befehl `disp('...')` druckt Text aus. Textteile, sog. "strings", werden immer in Hochkommata eingeschlossen, um sie von Variablennamen zu unterscheiden.

Ein erstes Programm:

SCILAB bietet die Möglichkeit, festabgespeicherte Befehlssequenzen zu definieren, die einfachste Form von Programmen. Diese müssen erst als einfache Text-Files erzeugt werden, und können dann von SCILAB ausgeführt werden. Text-Files erzeugt man mit einem Editor. Ein sehr guter Editor ist z.B. `emacs`, der allerdings einige Einarbeitung erfordert. Deshalb können Sie heute z.B. erstmal den Editor `nedit` benutzen, der sich fast von selbst erklärt.

21. Öffnen Sie den Editor Ihrer Wahl, z.B. durch Eingabe von `nedit &` in der Shell, geben Sie zwei beliebige SCILAB-Zeilen ein (z.B. `a=20` und `b=30`) und speichern Sie das Dokument unter dem Namen `myscript.sci`, in Ihrem Verzeichnis. (MATLAB: unter dem Namen `myscript.m`)
22. Gehen Sie nun mit der Maus auf den Menüpunkt "File" links oben im SCILAB Fenster, und dort auf den Unterpunkt "File operations". Es erscheint eine Liste der "Files" (Dokumente) in Ihrem Verzeichnis `scilab-2.6/poolXX`. Klicken Sie dann auf Ihr File `myscript.sci`, und danach ganz unten rechts auf "exec" (execute=ausführen). Nun passiert genau dasselbe, wie wenn Sie die Zeilen, die in Ihrem File `myscript.sci` stehen, direkt in die SCILAB-Benutzeroberfläche eingetippt hätten. Dies ist eine einfache Möglichkeit, häufig wiederholte Schritte abzuspeichern. Eine so abgespeicherte Befehlsfolge nennt man auch ein "Skript". (MATLAB: Gehen Sie nach Abspeichern des Files unter dem Namen `myscript.m` wieder in die MATLAB-Oberfläche und starten Sie das Skript, indem Sie einfach `myscript` eingeben.)
23. Vergewissern Sie sich, dass die Ausführung Ihres Skriptes Spuren hinterlassen hat (z.B. indem Sie sich den Wert von `a` ansehen)

24. Gehen Sie nun in den Editor, löschen Sie die alten Zeilen und tippen Sie die Zeilen

```
summe=0
for i=1:10
    i
    summe=summe+i
end
```

ein, und speichern Sie dies wieder unter dem Namen `myscript.sci` ab. (MATLAB: `myscript.m`)

25. Gehen Sie wieder wie in Aufgabe 22 vor, um das Skript auszuführen. Welchen Wert hat danach die Variable `sum`? Welchen Wert hat `i`?

Mit `for` und `end` wird eine sogenannte Schleife definiert, d.h. die Zeilen, die zwischen `for` und `end` stehen, werden mehrmals hintereinander ausgeführt, wobei `i` die Werte von 1 bis 10 durchläuft. Der Übersichtlichkeit halber rückt man alle Zeilen im Innern der Schleife meist ein bisschen ein.

26. Löschen Sie nun die erste Zeile Ihres Skriptes (`summe=0`), speichern Sie es wieder ab und lassen es ausführen. Welchen Wert hat die Variable `summe` nun? Starten Sie nochmals, und gucken Sie sich wieder den Wert von `summe` an. Was ist passiert?

Selbstdefinierte Funktionen:

Eine SCILAB-Funktion ist fast dasselbe wie ein Skript, nur dass Sie mit einer Zeile

```
function [output]=myfunction(input)
```

startet. Eine Funktion bekommt eine Variable (hier z.B. `input`) führt dann Rechnungen damit aus, und gibt am Ende das Ergebnis in Form einer neuen Variablen (hier z.B. `output`) aus. Der Wert der Ergebnis-Variablen muss irgendwo im Programmcode definiert werden (`output=...`). Dafür können auch neue Zwischenvariable definiert werden. Im Gegensatz zu einem Skript hat eine SCILAB-Funktion nur sogenannte lokale Variable, d.h. sie kennt nur `input` und die Variablen, die in der Funktion selbst definiert werden.

27. Schreiben Sie eine Funktion, z.B.

```
function [b]=mysquare(a)
b=a*a;
```

und speichern Sie diese z.B. unter dem Namen `mysquare.sci` ab. (MATLAB: Nennen Sie die Funktion `mysquare.m`. Achtung: Im Gegensatz zu SCILAB muss der Filename in MATLAB **immer** mit dem Funktionsnamen übereinstimmen.)

28. Um die Funktion für SCILAB verfügbar zu machen, muss sie zunächst mit dem Befehl `getf` geladen werden. Gehen Sie dafür wieder auf den Menüpunkt "File", "File operations", und klicken Sie auf den Filenamen `mysquare.sci` und dann auf "Getf" (get function). Dies lädt die Funktion in SCILAB ein. (MATLAB: Dieser Schritt entfällt. Dafür kann man in MATLAB nur Funktionen aufrufen, deren File mit dem entsprechenden Namen im aktuellen Verzeichnis liegt.)
29. Den **Start** einer Funktion erreicht man einfach durch Aufruf in der SCILAB-Shell, also z.B. durch `mysquare(4)`.

30. Sogenannte Kommentare stehen im Rest einer Zeile nach // (in MATLAB nach %). Sie werden nicht ausgeführt, aber dienen Leuten, die das Programm (die Funktion) später einmal lesen wollen, als wertvolle Hilfe.

31. Die bedingte Ausführung einer Programmpartie kann mit `if`, `end` formuliert werden. Dabei sind die Bedingungen durch Zahlen dargestellt und gelten immer als WAHR, wenn sie nicht 0 sind. (Ein Trick: will man eine längere Programmregion auskommentieren (also ungültig machen, ohne sie zu löschen) kann man am Beginn einfach `if 0` und am Ende `end` schreiben.)

Vergleiche sind möglich z.B. durch `==` (gleich). Achtung, hier niemals das einfache `=` tippen, denn dann wird einfach nur ein Wert zugewiesen. Man kann auch mit `<=`, `<`, und `~=` (ungleich) vergleichen. Ein Beispiel wäre

```
if a<4
    disp('a ist kleiner als vier')
end
```

was sie gerne in Ihre Funktion einarbeiten können. D.h. Eintippen, Abspeichern, Laden mit "Getf", Ausführen – ein Tipp ist, mit der Cursor-nach-oben Taste den (automatisch generierten) "getf" Befehl wiederzufinden und wieder auszuführen statt immer wieder per Mausklick über das Menü zu gehen.

32. Logische Verknüpfungen von Ausdrücken sind durch "und": `(...)&(...)`, durch "oder": `(...) | (...)` und durch "nicht": `~(...)` möglich. Mit `else` kann man noch etwas ausführen, wenn die Bedingung **nicht** erfüllt ist. Testen Sie z.B.

```
if (a<4) & (a>1)
    disp('a ist kleiner als vier und groesser als 1')
else
    disp('a ist nicht kleiner als vier oder nicht groesser als 1')
end
```

in Ihrer Funktion.

33. Schreiben Sie nun eine Funktion, die das n -te Glied der Folge $a_n = (1 + \frac{1}{n})^n$ erzeugt. Z.B.

```
function [a]=eulerfolge(n)
a=(1+1/n)^n
```

Speichern Sie die Funktion und aktivieren Sie sie mit "Getf".

34. Führen Sie nun die Funktion aus, indem Sie sie z.B. mit `eulerfolge(10)` aufrufen. Testen Sie auch `eulerfolge(1000)`. Vergleichen Sie mit `exp(1)` (Bei `eulerfolge(1e15)` bekommt SCILAB langsam Schwierigkeiten...)

35. Eine Funktion kann auch einen ganzen Vektor als Output haben. Schreiben Sie nun eine Funktion, die die ersten n Glieder der Folge $(1 + \frac{1}{n})^n$ erzeugt, z.B. indem Sie die alte Funktion abwandeln:

```
function [a]=eulerfolge(n)
for k=1:n
    a(k)=(1+1/k)^k
end
```

Speichern und aktivieren Sie diese Funktion, und rufen Sie sie mit `[a]=eulerfolge(20)` auf. Welchen Wert hat nun `a`? Plotten Sie `a` mit `plot(a)`.

36. Wer noch Kraft hat, sollte nun noch versuchen, eine Funktion zu schreiben, die zu gegebenem n die ersten n Partialsummen $\sum_{i=1}^k \frac{1}{i}$ der harmonischen Reihe berechnet (also für $k = 1, 2, \dots, n$). Dabei kann man ausnutzen, dass immer ein Glied durch Aufaddieren von $\frac{1}{k}$ aus dem vorhergehenden erhalten wird. Plotten Sie die ersten tausend Glieder.
37. Wer seine Fähigkeiten noch weiter testen möchte, kann versuchen, π zu berechnen. Die Zahl π kann nämlich durch die Reihe

$$\pi = 4 \left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \dots \right)$$

dargestellt werden. Berechnen Sie eine Näherung für π , indem Sie die ersten 100 Partialsummen auswerten. Gucken Sie sich die Partialsummenfolge mit `plot` an. Berechnen Sie auch die zehntausendste Partialsumme. Wie genau ist dieser Wert?

38. Für Fortgeschrittene: eine Funktion kann auch mehrere Inputs und mehrere Outputs haben. Sie wird dann begonnen mit

```
function [output1, output2, ...]=myfunction(input1,input2,...)
```

Dabei sind alle Inputs und Outputs Matrizen, und die Outputs müssen wieder irgendwo im Programmcode definiert werden.

Matrizen und ihre Speicherung:

Eine Matrix ist nichts anderes als eine Tabelle von Zahlen. Man sagt z.B. "eine $n \times m$ -Matrix" ("n kreuz m Matrix"), und meint damit eine Tabelle mit n Zeilen und m Spalten. Matrizen sind in SCILAB die wichtigsten Variablen, und Vektoren werden eigentlich nur als Sonderfall einer Matrix behandelt (nämlich als $1 \times n$ - bzw. als $n \times 1$ -Matrix (liegend bzw. stehend)). Wie man mit Matrizen rechnet, werden wir bald in der Vorlesung behandeln. Um bereits jetzt ein bisschen mit SCILAB-Matrizen vertraut zu werden, können Sie noch die folgenden Übungen machen.

39. Tippen Sie z.B. `b=[1, 2 , 3; 4 5 6]` ein. *Matrizen werden auch in eckige Klammern eingeschlossen. Das Komma oder Leerzeichen trennt dabei, wie zuvor, die Spalten, und das Semikolon trennt die Zeilen.*
40. Greifen Sie mit `b(2,1)` auf das erste Element in der zweiten Spalte zu.
41. Was erhält man durch `size(b)`?
42. Transponieren Sie `b` durch die Operation `c=b'`. Transponieren heisst einfach nur, aus Zeilen Spalten zu machen, und umgekehrt.
43. Spezielle Matrizen kann man erzeugen durch die Kommandos `zeros(5,2)`, `ones(5,2)`, `eye(5,2)`, `rand(2,3)`. Was erzeugen diese jeweils?
44. Gewöhnliche **Matrixoperationen** haben in SCILAB die einfachst mögliche Form: Für `A=[1,2; 3, 4]` und `B=[0, 1; 1, 1]`, kann man z.B. einfach `A+B`, `10*A`, `A*B`, `A-B`, oder `A^2` ausführen.
45. Speichern Sie die Matrizen `A` und `B` in Ihrem Verzeichnis ab, indem Sie `save('filename', A, B)` eingeben (MATLAB: `save 'filename' A B`)
46. Der Befehl `clear` löscht wieder alle von Ihnen bis jetzt erzeugten Variablen.

47. Mit `load('filename')` können Sie die Daten wieder laden.

Beispiellösung zu Aufgabe 36:

```
function [a]=harmonischereihe(n)
a(1)=1
for k=2:n
    a(k)=a(k-1)+1/k
end
```