Praktikum: “Numerik am Computer” FS 2018
Prof. M. Grote
R. Brügger, R. Croce, A. Heinimann, M. Nagy, G. Talarico Universität Basel

Beilage zur Serie 2

Debugging in MATLAB

Um Fehler, sogenannte ‘Bugs’, in Programmen zu finden, gibt es in MATLAB eine sehr nützliche Funktion: den Debugger. Um mit ihm umzugehen, sollte man allerdings erst einmal die am häufigsten auftretenden Fehlermeldungen in MATLAB kennen. Dazu hier eine kleine Übersicht:

1. Arithmetische Fehler: Dies sind Fehlermeldungen, die auftreten, wenn Operationen wie `+,-,*,/,^,\` auf Vektoren und Matrizen angewendet werden, die unpassende Dimensionen haben, z.B.

```
>> [12,13,14] + [0,1,2,3]
Error using  +
Matrix dimensions must agree.
```

Die Vektoren $(12, 13, 14)$ und $(0, 1, 2, 3)$ könenn nicht miteinander addiert werden, weil sie nicht dieselben Dimensionen haben.

```
>> [5 6 7; 8 9 10]*[1 0 0; 0 0 1]
Error using  *
Inner matrix dimensions must agree.
```

Die Matrizen $\mathbf{A} = \begin{pmatrix} 5 & 6 & 7 \\ 8 & 9 & 10 \end{pmatrix}$ und $\mathbf{B} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ können nicht miteinander multipliziert werden, weil sie nicht dieselben Dimensionen haben. Je nachdem, was die eigentliche Absicht war, kann dies folgendermassen behoben werden:

1. Die Matrizen sollten eintragsweise multipliziert werden (d.h. der Eintrag $a_{i,j}$ soll nur mit dem Eintrag $b_{i,j}$ multipliziert werden). In diesem Fall kann man `"*"` einfach durch `".*"` erstzen.

2. Es sollte tatsächlich ein Matrixprodukt berechnet werden. Dann muss eine der Matrizen transponiert werden, indem man zum Beispiel `B` durch `B'` ersetzt. Dadurch würde eine 2×3-Matrix mit einer 3×2-Matrix multipliziert werden, was mathematisch Sinn ergibt.

3. Die Matrizen sind falsch aufgeschrieben. Da muss man die Lösung selbst finden...

```
>> [0,1,3;5,7,9]^4
Error using  ^
Inputs must be a scalar and a square matrix.
To compute elementwise POWER, use POWER (.^) instead.
```

Hier wurde versucht, eine nicht-quadratische Matrix zu potenzieren. Falls man eintragsweise potenzieren will, muss man `"^"` durch `".^"` ersetzen.

2. Indexierungsfehler: Diese Fehler treten auf, wenn man auf nicht existierende oder ungültige Matrixeinträge zugreifen will.

```
>> A = [10,11];
>> A(3)
Index exceeds matrix dimensions.
```

Hier ist **A** definiert als Vektor der Länge 2, man versucht aber, auf den dritten Eintrag zuzugreifen. Dieser existiert nicht, weswegen MATLAB eine Fehlermeldung ausgibt.

```
>>A(-1)
>>A(i)
>>A(1.5)
>>A(0)
Subscript indices must either be real positive integers or logicals.
```

Versucht man, auf Matrixeinträge mit negativen, rationalen oder komplexen Indizes zuzugreifen, gibt dies auch einen Fehler - Matrixindizes dürfen nur positive ganze Zahlen sein! Beachte: Im Gegensatz zu anderen Programmiersprachen fängt Matlab bei der Indexierung mit 1 anstelle von 0 an (damit die Indexierung konsistent mit mathematischer Notation ist). Deswegen tritt auch eine Fehlermeldung auf, wenn man versucht, auf `A(0)` zuzugreifen.

3. Zuweisungsfehler: Will man inkompatible oder ungültige Zuweisungen machen, treten diese Fehler auf.

```
>> a = 2;
>> if a = 3
 if a = 3
      |
Error: The expression to the left of the equals sign
is not a valid target for an assignment.
```

`if` erwartet einen Wahrheitswert, hier wird aber versucht, a den Wert 3 zuzuweisen. Richtig wäre `"if a == 3"`.

```
>> A = [1,2,3;4,5,6;7,8,9];
A(3,:) = [10,11];
Subscripted assignment dimension mismatch.
```

Hier wird versucht, alle Einträge der dritten Zeile von $\mathbf{A} = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$ durch den Vektor $(10, 11)$ zu ersetzen. Das funktioniert nicht, weil diese Zeile drei Einträge hat. Möglich wären zum Beispiel `A(3,[1,2]) = [10,11]` oder `A(3,:) = [10,11,12]`.

4. Syntaxfehler: Dies sind Fehler, die gegen die "Satzbauregeln" von MATLAB verstossen.

```
>> A(1
 A(1
    |
Error: Expression or statement is incorrect--possibly
unbalanced (, {, or [.
```

Irgendwo wurde eine Klammer vergessen.

```
>> A(1))
 A(1))
     |
Error: Unbalanced or unexpected parenthesis or bracket.
```

Irgendwo ist eine Kammer zu viel.

```
>> A(1::)
 A(1::)
     |
Error: Unexpected MATLAB operator.
```

Ein Zeichen ist an der falschen Stelle.

```
>> x = 4+5+
 x = 4+5+
         |
Error: Expression or statement is incomplete or incorrect.
```

Eine Eingabe wurde nicht richtig beendet.

```
>> x+$
 x+$
   |
Error: The input character is not
valid in MATLAB statements or expressions.
```

Ein MATLAB unbekanntes Zeichen wurde verwendet.

5. Fehler bei Funktionsaufrufen: Dies sind Fehler, die gegen die "Satzbauregeln" von MATLAB verstossen.

```
>> x = cos()
Error using cos
Not enough input arguments.
```

Eine Funktion wird mit zu wenig Input-Argumenten aufgerufen.

```
>> x = cos(3,4,5)
Error using cos
Too many input arguments.
```

Eine Funktion wird mit zu vielen Input-Argumenten aufgerufen.

```
>> [x,y] = cos(2)
Error using cos
Too many output arguments.
```

Das gleiche geht auch mit zu vielen/zu wenigen Output-Argumenten...

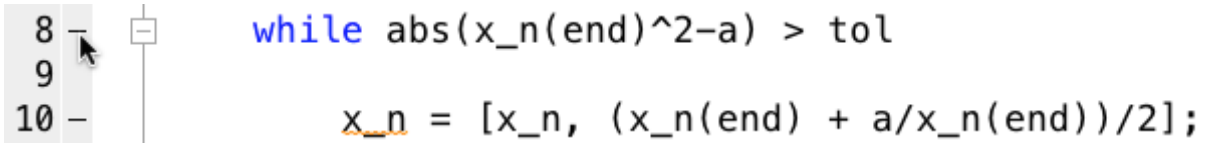
```
>> x = cso(3)
Undefined function or variable 'cso'.
```

Eine nicht definierte Funktion wird aufgerufen. Meistens geschieht dies durch Tippfehler.

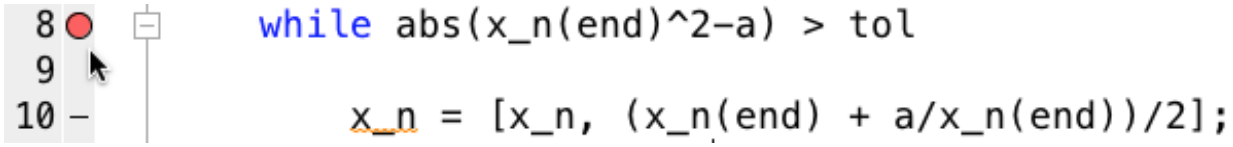
Der Debugger

Der Debugger ist ein in MATLAB eingebautes Tool, mit dem man Schritt für Schritt den eigenen Code durchgehen kann, um nachzuvollziehen, welche Befehle welchen Output erzeugen, oder welcher Wert zu einem bestimmten Zeitpunkt in einer Variable steht. Dadurch kann man sehr leicht Fehler identifizieren und beheben.

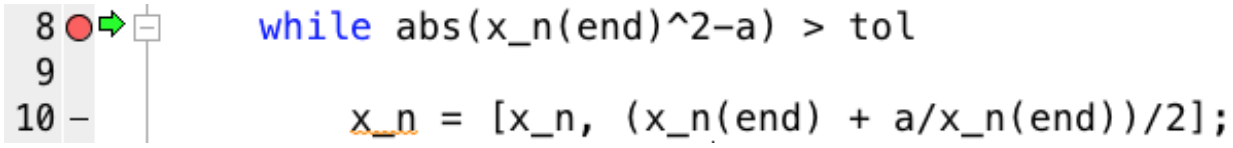
Um den Debugger zu aktivieren, sucht eine Zeile in eurem Code aus, in der ihr gerne überprüfen möchtet, was passiert, und klickt auf den schwarzen Strich neben der Zeilenzahl ganz links.



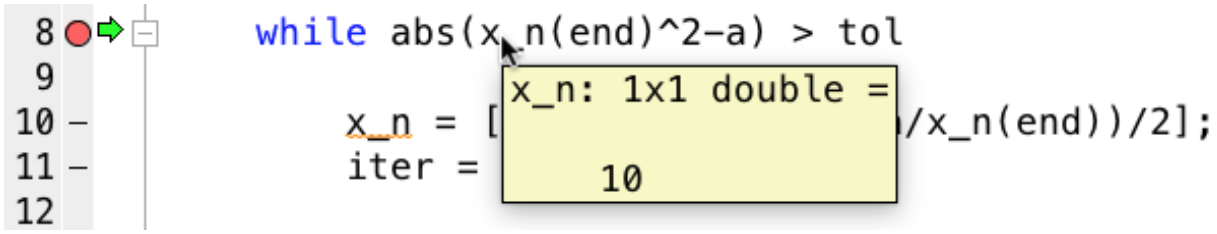
Danach erscheint dort ein roter Punkt, ein sogenannter "Breakpoint". **Achtung:** Ein `clear all` entfernt je nach MATLAB-Version auch alle Breakpoints - ersetzt daher jedes `clear all` durch ein einfaches `clear`!



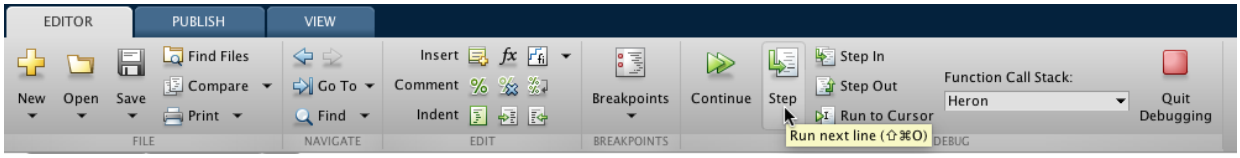
Wird nun das Programm ausgeführt, hält es an jedem Breakpoint an. Beachtet, dass die Zeile noch nicht ausgewertet wurde. Das passiert erst, wenn man das Programm weiterlaufen lässt.



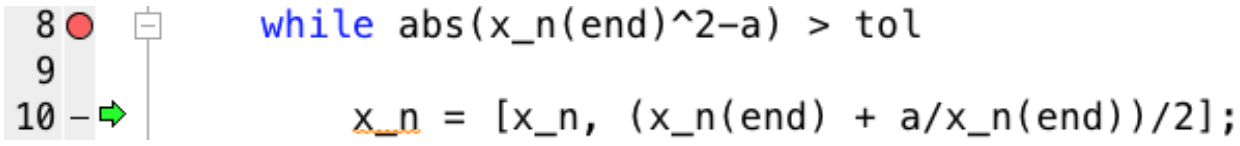
Man kann dann mit dem Mauszeiger über Variablennamen fahren, um deren Werte zu sehen, oder sie im Workspace Window genauer betrachten.



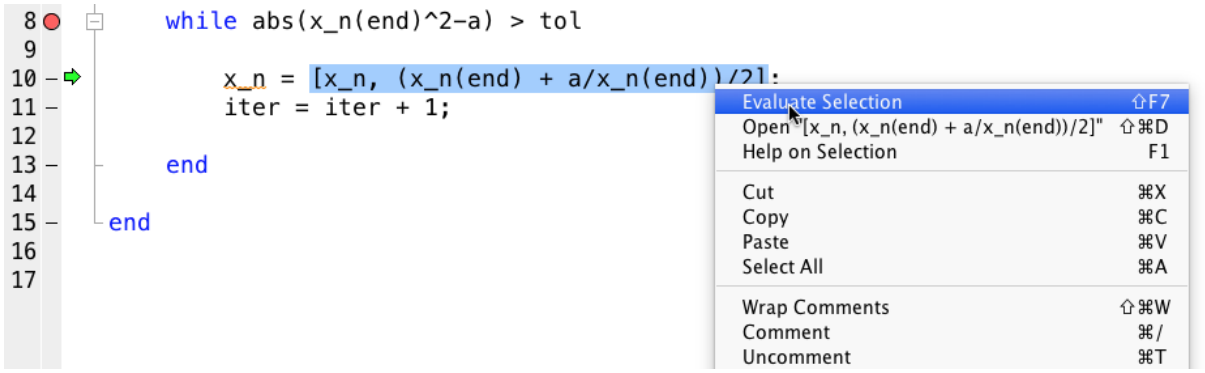
Mit dem Button "Step" oben in der Toolbar könnt ihr die aktuelle Zeile ausführen lassen und zur nächsten Zeile im Programmablauf springen.



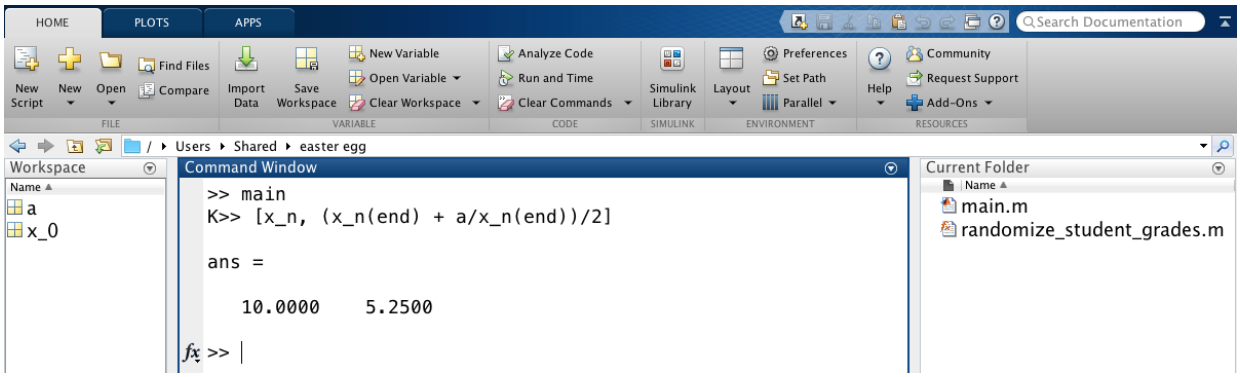
Der grüne Pfeil links zeigt auch an, in welcher Zeile ihr euch gerade befindet.



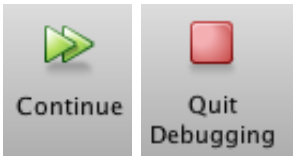
Um Fehler zu finden, kann es hilfreich sein, nur einen Teil einer Zeile auszuführen. Dies könnt ihr tun, indem ihr den gewünschten Teil markiert, darauf rechtsklickt, und dann "Evaluate Selection" auswählt.



Im Command Window seht ihr dann, was in dem ausgewählten Stück Code passiert.



Mit dem Button "Continue" läuft das Prgramm weiter, bis es entweder den nächsten Breakpoint oder das Ende erreicht hat. Mit dem Button "Quit Debugging" beendet ihr den Debug-Modus und das Programm wird nicht weiter ausgeführt.



Newton-Interpolation

Für die $n+1$ paarweise verschiedenen Stützpunkte (x_i, y_i), $i = 0, 1, \ldots, n$, betrachten wir statt der Lagrange-Interpolation (siehe Serie 1) die Newton-Interpolation. Dann hat p die Form

$$p(z) = \sum_{j=0}^{n} c_j \, \omega_j(z),$$

wobei ω_j das j-te Newton-Polynom ist, d.h.

$$\omega_j(z) = \prod_{k=0}^{j-1}(z - x_k).$$

Auch hier werden die Koeffizienten mit Hilfe der Interpolationbedingung $p(x_i) = y_i$, $i = 0, \ldots, n$ bestimmt. Das daraus entstehende lineare Gleichungssystem können wir schreiben als

$$\begin{aligned}
p(x_0) &= c_0 &&= y_0,\\
p(x_1) &= c_0 + c_1(x_1 - x_0) &&= y_1,\\
p(x_2) &= c_0 + c_1(x_2 - x_0) + c_2(x_2 - x_0)(x_2 - x_1) &&= y_2,\\
&\ldots
\end{aligned}$$

Die Koeffizienten können (wie in der Vorlesung gezeigt wurde) mit dem Schema der dividierten Differenzen berechnet werden. Mit den Notationen aus der Vorlesung gilt

$$c_n = \delta^n\, y[x_0, x_1, \ldots, x_n].$$

Damit dies auch für $n = 0$ stimmt, setzen wir $\delta^0\, y[x_0] := y[x_0]$. Als Pseudocode lautet dieser Algorithmus:

for $k = 0, 1, \ldots, n$ **do**
  $c_k = y_k$
end for
for $j = 1, 2, \ldots, n$ **do**
  for $k = n, n-1, \ldots, j$ **do**
    $c_k = \dfrac{c_k - c_{k-1}}{x_k - x_{k-j}}$
  end for
end for

Das Interpolationspolynom p kann dann an der Stelle z wie im folgenden Pseudocode effizient ausgewertet werden:

$p = c_n$
for $k = n-1, n-2, \ldots, 0$ **do**
  $p = c_k + (z - x_k) \cdot p$
end for

Allgemeine Informationen zum Praktikum befinden sich auf der Webseite
`https://tinyurl.com/UnibasFS18PraktikumII`