Praktikum II „Numerik am Computer“ FS 2024
Prof. Dr. H. Harbrecht Universität Basel
M. Dujella, A. Heinimann, L. Mehrle, M. Morini, M. Ramm, D. Yusofy

---

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 diesem umgehen zu können, ist es nützlich, erst einmal die am häufigsten auftretenden Fehlermeldungen in MATLAB zu 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 +
Arrays have incompatible sizes for this operation.
```

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

```
>> [5 6 7; 8 9 10] * [1 0 0; 0 0 1]
Error using *
Incorrect dimensions for matrix multiplication.
```

Die Matrizen

$$\mathbf{A} = \begin{pmatrix} 5 & 6 & 7 \\ 8 & 9 & 10 \end{pmatrix} \text{ und } \mathbf{B} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

können nicht miteinander multipliziert werden, weil sie nicht die richtigen Dimensionen dazu haben. Je nachdem, was die eigentliche Absicht war, kann dies folgendermassen behoben werden:

a) 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 muss „`*`“ durch „`.*`“ ersetzt werden.

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

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

```
>> [0, 1, 3; 5, 7, 9]^4
Error using ^
Incorrect dimensions for raising a matrix to a power.
```

Hier wurde versucht, eine nicht-quadratische Matrix zu potenzieren. Um diese Matrix eintragsweise zu potenzieren, muss „`^`“ durch „`.^`“ ersetzt werden.

2. Indexierungsfehler: Diese Fehler treten auf, wenn versucht wird, auf nicht existierende oder ungültige Matrixeinträge zuzugreifen.

```
> > vec = [10, 11];
> > vec(3)
Index exceeds the number of array elements.
```

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.

```
> > vec(-1)
> > vec(i)
> > n = 1.5;
> > vec(n)
> > vec(0)
Array indices must be positive integers or logical values.
```

Wird versucht auf Matrixeinträge mit negativen, rationalen oder komplexen Indizes zuzugreifen, gibt dies auch einen Fehler - Matrixindizes dürfen nur positive ganze Zahlen sein! Beachte, dass MATLAB im Gegensatz zu anderen Programmiersprachen bei der *Indexierung mit 1 anstelle von 0* startet, damit die Indexierung konsistent mit mathematischer Notation ist. Deswegen tritt eine Fehlermeldung auf, wenn versucht wird, auf `vec(0)` zuzugreifen.

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

```
> > a = 2;
> > if a = 3
if a = 3
     ↑
Incorrect use of ' = ' operator. Assign a value to a variable using
' = ' and compare values for equality using ' = = '.
```

Der Befehl `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 (und nicht zwei) 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
   ↑
Invalid expression --possibly unbalanced (, {, or [.
```

Irgendwo wurde eine Klammer vergessen.

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

Irgendwo ist eine Klammer zu viel.

```
> > A(1::)
A(1::)
    ↑
Invalid use of 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: Invalid text character.
```

Ein in 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 wenigen Input-Argument aufgerufen.

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

Eine Funktion wird mit zu vielen Input-Argument aufgerufen.

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

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

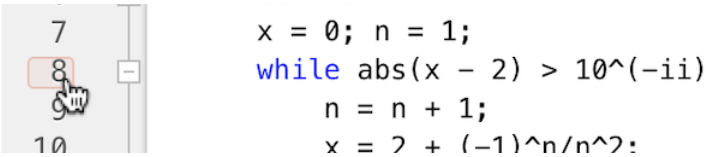
```
> > x = cso(3)
Undefined function or variable 'cso'.
```

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

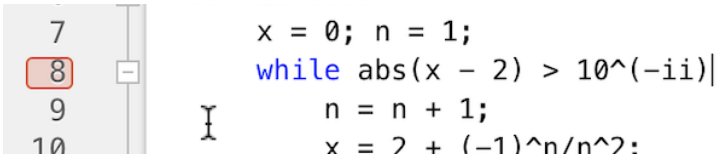
Der Debugger

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 bestimmten Variable steht. Dadurch kann man sehr leicht Fehler identifizieren und beheben.

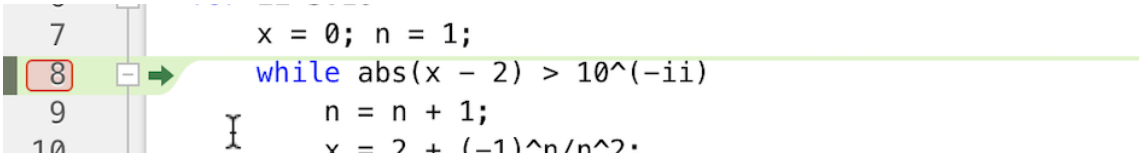
Um den Debugger zu aktivieren, sucht eine Zeile in eurem Code aus, in der ihr gerne überprüfen möchtet, was passiert, und klickt auf die Zeilenzahl ganz links.



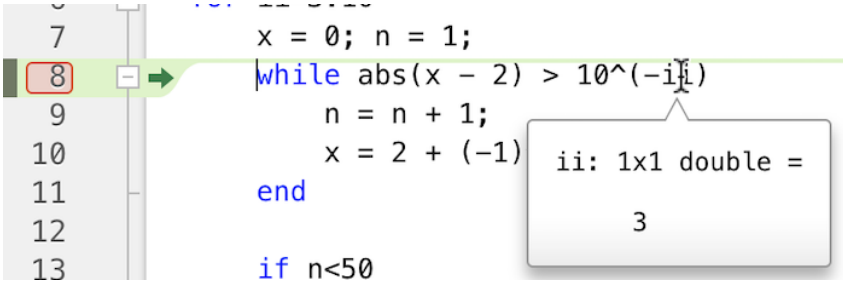
Danach wird die Zeilenzahl rot eingefärbt, es entsteht ein sogenannter „Breakpoint“. **Achtung:** Der Befehl `clear all` entfernt auch alle Breakpoints, der Befehl `clear` jedoch nicht.



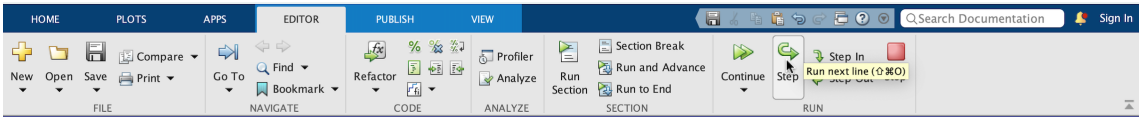
Wird nun das Programm ausgeführt, hält es an jedem Breakpoint an. Beachtet, dass die Zeile noch nicht ausgewertet wurde. Das passiert erst, wenn man das Programm weiterlaufen lässt.



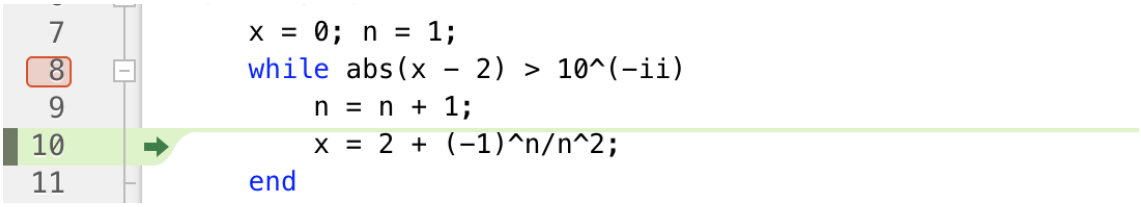
Man kann dann mit dem Mauszeiger über Variablennamen fahren, um deren Werte zu sehen, oder sie im Workspace Window genauer betrachten.



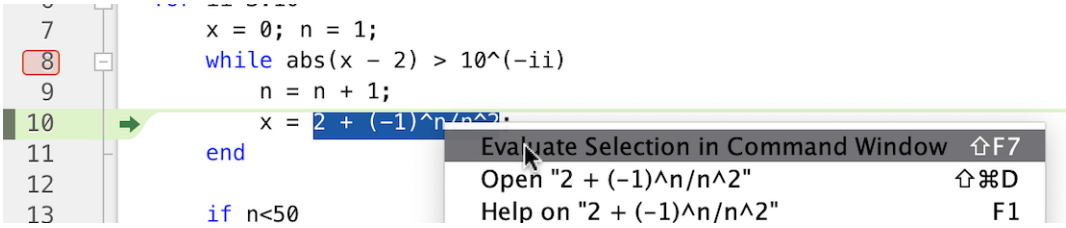
Mit dem Button „Step“ oben in der Toolbar könnt ihr die aktuelle Zeile ausführen lassen und zur nächsten Zeile im Programmablauf springen.



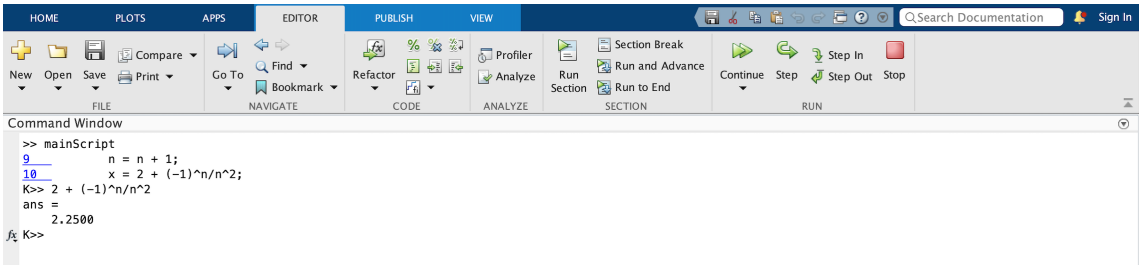
Der grüne Pfeil links zeigt auch an, in welcher Zeile ihr euch gerade befindet.



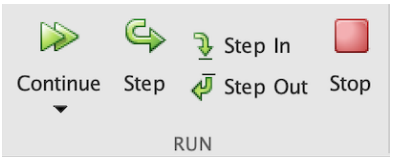
Um Fehler zu finden, kann es hilfreich sein, nur einen Teil einer Zeile auszuführen. Dies könnt ihr tun, indem ihr den gewünschten Teil markiert, darauf rechtsklickt, und dann „Evaluate Selection in Command Window“ auswählt.



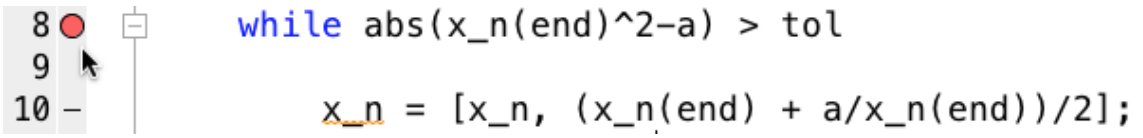
Im Command Window seht ihr dann, was in dem ausgewählten Stück Code passiert.



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



Bemerkung: In MATLAB Versionen, die älter sind als die 2021 Version, sieht das Debuggen etwas anders aus. Die Funktionsweise ist aber dieselbe. Ein Breakpoint wird auf den schwarzen Strich neben der Zeilenzahl gesetzt, dort erscheint dann ein roter Punkt. Die restlichen Bedienelemente lassen sich dann auf gleiche Weise wie oben dargestellt benutzen.



Numerische Differentiation

Um die erste Ableitung einer Funktion f an der Stelle x numerisch zu bestimmen, benutzen wir die *Differenzenquotienten*

$$D_h^+ f(x) = \frac{f(x+h) - f(x)}{h} \text{ und}$$
$$D_h^- f(x) = \frac{f(x) - f(x-h)}{h}.$$

Nach Definition der Ableitung gilt dann

$$f'(x) = \lim_{h \to 0} D_h^+ f(x) = \lim_{h \to 0} D_h^- f(x).$$

Die beiden Differenzenquotienten D_h^+ und D_h^- werden *rechtsseitige-* bzw. *linksseitige Differenz* genannt.

Beispiel: Seien (x_i, y_i), $i = 0, 1, ..., n$, Tupel aus äquidistanten Stützpunkten und Funktionsauswertungen $y_i := f(x_i)$ für alle $i = 1, ..., n$. Sei $h := x_1 - x_0$ der Abstand zwischen zwei aufeinanderfolgenden Stützpunkte. Dann gilt für $0 < i < n - 1$

$$f'(x_i) \approx D_h^+ f(x_i) = \frac{y_{i+1} - y_i}{h}.$$

Zur Approximation der zweiten Ableitung wird der Differenzenquotient

$$D_h^2 f(x) \coloneqq \left(D_h^- \circ D_h^+\right) f(x) = \frac{1}{h} D_h^- \left[f(x+h) - f(x)\right] = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}$$

verwendet.

Bemerkung: Man kann nachprüfen, dass

$$D_h^2 = D_h^+ \circ D_h^- = D_h^- \circ D_h^+$$

gilt.

Sparse Matrix Format

In MATLAB können Matrizen in zwei verschiedenen Formaten gespeichert werden: Einerseits im `full`-Format und andererseits im `sparse`-Format.

↪ Das `full`-Format speichert alle Einträge.

↪ Im Gegensatz dazu speichert das `sparse`-Format nur die nicht-null Einträge.

Da MATLAB die Formate intern umrechnet, kann man ohne Probleme eine `sparse`- und eine `full`-Matrix miteinander verrechnen. Die meisten MATLAB-Funktionen für Matrizen im `full`-Format kann man auch auf `sparse`-Matrizen anwenden. Daneben gibt es aber auch Funktionen, welche nur mit Matrizen im `sparse`-Format funktionieren.

Wann sollte also welches Format verwendet werden? Sobald man grosse, dünn besetzte (d.h. wenige Einträge sind ungleich 0) Matrizen hat, sollte das `sparse`-Format bevorzugt werden. Dadurch laufen Programme meistens schneller und brauchen weniger Speicherplatz. Falls jedoch häufig die Struktur der Matrix geändert wird, indem zum Beispiel einzelne Einträge hinzugefügt werden, ist das `full`-Format vermutlich schneller.

Zur Umwandlung vom einen ins andere Format gibt es die Befehle `sparse` und `full`. Einige wichtige Befehle im Zusammenhang mit `sparse`-Matrizen sind `spalloc`, `speye`, `spones`, `spdiags` und `spy`. Genauere Infos dazu (und zu weiteren Befehlen) sind in der MATLAB-Hilfe zu finden.