HANSER

Leseprobe

Ulrich Stein

Programmieren mit MATLAB

Programmiersprache, Grafische Benutzeroberflächen, Anwendungen

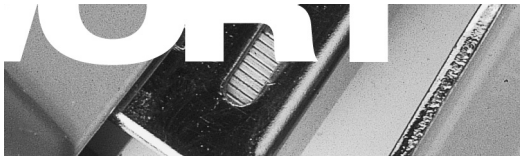
ISBN (Buch): 978-3-446-44299-3

ISBN (E-Book): 978-3-446-44391-4

Weitere Informationen oder Bestellungen unter

http://www.hanser-fachbuch.de/978-3-446-44299-3

sowie im Buchhandel.



Vorwort zur 5. Auflage

Dieses Lehrbuch wendet sich an Studentinnen und Studenten der Ingenieurwissenschaften in den ersten Semestern und orientiert sich in der Stoffauswahl an der Vorlesung „Angewandte Informatik“ am Department „Maschinenbau und Produktion“ der Hochschule für Angewandte Wissenschaften, am Berliner Tor in Hamburg.

Unser Weg durch die Informatik beginnt mit den elementaren Prinzipien der Datenverarbeitung und vermittelt dann anhand der in MATLAB® integrierten Programmiersprache systematisch die Grundkenntnisse des Programmierens. Darauf aufbauend vertiefen Anwendungen das Wissen und führen in die weitergehenden numerischen Verfahren von MATLAB ein. Wo für spezielle Probleme das Basismodul von MATLAB nicht mehr ausreicht, wird für eine intensivere Nutzung auf die MATLAB-Toolboxen™ verwiesen – deren Installation zum Verständnis dieses Buches jedoch nicht notwendig ist.

Anwendungsbeispiele, zahlreiche Abbildungen und Übungsaufgaben zum Stoff fördern das Verständnis. Das Werk ist sowohl für Programmierneulinge als auch für Umsteiger von anderen Programmiersprachen wie C geeignet.

In diesem Buch wird in MATLAB programmiert. Im Vordergrund steht dabei aber nicht primär das Programm MATLAB, sondern die Vermittlung von Programmierkenntnissen, wie sie auch für andere Sprachen nützlich sind. Das bedeutet, dass nicht immer Wert darauf gelegt wurde, den Programm-Code in Bezug auf die Stärken von MATLAB zu optimieren. Auch ist dies kein Referenz-Handbuch für MATLAB. Die MATLAB-Funktionen werden oft nur so weit vorgestellt, wie es für die aktuelle Aufgabenstellung nötig ist. Für eine vollständige Definition der Funktionen sei auf die MATLAB-Hilfe verwiesen.

Warum wird das Programmieren bei uns nicht anhand von C vermittelt?

In der Industrie habe ich einige Jahre Programme in C beziehungsweise C++ geschrieben. Nahezu alles, was das Programmiererherz begehrt, ist mit C möglich. Aber diese Sprache enthält auch viele, böse Fallen. Um sich einigermaßen fehlerfrei in C ausdrücken zu können, muss man lange damit gearbeitet haben, besonders wenn man grafische Oberflächen und Grafiken erstellen will oder sich im Bereich COM/OLE tummeln möchte. Die letzten beiden Punkte konnte ich deshalb in meiner zweisemestrigen C-Vorlesung höchstens erwähnen. Und Hand aufs Herz – nur ein sehr kleiner Teil der Ingenieur-Absolventen hat heutzutage im Beruf noch mit C zu tun. MATLAB jedoch wird in vielen Bereichen eingesetzt, sowohl im Fortgeschrittenen-Studium, zum Beispiel in der Regelungstechnik oder in der Simulation von Motoren, als auch in Industrie und Forschung.

Vor einigen Jahren habe ich den Versuch gestartet, Programmieren anhand von MATLAB zu lehren, mit ähnlichen Lernzielen und Inhalten wie vorher mit C. Und es hat geklappt. Besser noch – mit MATLAB war vieles einfacher. Ich kam schneller voran und konnte mehr Inhalte bringen. Und es machte Spaß, sowohl den Studenten als auch mir.

Dieses Buch soll das Programmieren lehren, aber nicht zum Informatiker ausbilden. Deshalb fehlen Themen wie Automatentheorie, formale Sprachen, Petrinetze, Maschinensprache, Netzwerke und auch die beliebten Umwandlungen vom Dezimal- zum Dualsystem. So auf Du und Du mit Prozessor, Speicher oder serieller Schnittstelle sind Ingenieure heute meist nicht mehr. Leider mussten auch die verketteten Listen wegfallen – dazu fehlen in MATLAB die Zeiger. Informationen zu diesen Themen finden Sie in Büchern zur Informatik, wie im „Informatik Handbuch“ von P. Rechenberg und G. Pomberger.

Frau Dipl.-Ing. Erika Hotho machte mir vor sechs Jahren den Vorschlag, ein Buch über das Programmieren mit MATLAB zu schreiben. Vielen Dank für Ihr Engagement und Dank auch an Frau Franziska Jacob, M.A und Frau Mirja Werner, M.A., die mich inzwischen kompetent betreuen, und an Frau Dipl.-Ing. (FH) Franziska Kaufmann, die mir beim Layout zur Seite stand. Dank auch an alle Kollegen, die mich zu diesem Projekt ermutigten und mir hilfreiche Tipps gaben, wie Prof. Dr.-Ing. habil. Jürgen Dankert, Prof. Dr.-Ing. Bernd Kost, Prof. Dr.-Ing. Wolfgang Schulz, Prof. Dr. rer. nat. Bernd Baumann, Prof. Dr.-Ing. Thomas Frischgesell. Nicht zu vergessen die Mitarbeiter des Rechenzentrums Berliner Tor, die mir in Fragen der MATLAB-Installation zur Seite standen.

2007 erschien die erste Auflage dieses Buches. Danke für die vielen Zuschriften und für die nahezu einhellig positive Reaktion der Leser. Für die fünfte Auflage habe ich mehrere Bereiche vollständig überarbeitet, zum Beispiel die neuen Methoden der objektorientierten Programmierung eingebaut und für die Prozess-Kommunikation den Kontakt zum Internet ausgewählt. Auch wurden einige weitere numerische Verfahren aufgenommen. Damit das Buch dadurch nicht noch dicker und teurer wird, habe ich im Gegenzug längere Programm-Listings entfernt und auf meine Homepage gepackt.

Die im Buch beschriebenen und abgebildeten Abläufe beziehen sich auf die Bedieneroberfläche der Version **MATLAB 2014**. Andere MATLAB-Versionen präsentieren sich dem Anwender zum Teil mit einer leicht abgewandelten Oberfläche. Lassen Sie sich deshalb nicht verwirren. Die vorgestellten Programme wurden jedoch mit verschiedenen Versionen von MATLAB 7.0.1 bis MATLAB 2014 getestet. Erweiterungen und die Lösungen der Aufgaben finden Sie auf meiner Homepage

www.Stein-Ulrich.de/Matlab/

Ich wünsche den Lesern, dass Ihnen das Programmieren neben der Lernarbeit auch Spaß macht und dass Ihnen möglichst viel vom hier präsentierten Stoff im wirklichen Leben bei Problemlösungen nützt. Und nicht verdrängen oder vergessen: Informatik kann auch Schaden anrichten. Deshalb sollte jeder, der programmiert, sich überlegen, ob er sein Tun verantworten kann und will.

Hamburg, im Januar 2015
Ulrich Stein

Inhalt

2

Programmstrukturen

2 Programmstrukturen

In Kapitel 1 haben wir ein wenig in MATLAB herumgeschnüffelt, und Sie haben auch bereits einige Möglichkeiten des Programms kennen gelernt. Nun, in Kapitel 2, sollen systematisch die zentralen Bereiche der MATLAB-Programmiersprache vorgestellt werden. Für eine vollständige Beschreibung aller Möglichkeiten, die MATLAB bietet, reicht der Umfang dieses Buches jedoch bei Weitem nicht aus. Deshalb wurden für dieses Kapitel speziell die Aspekte ausgewählt, die typischerweise auch in anderen Programmiersprachen, wie in C oder Java, die Basis der Programmierung bilden. Dazu gehören

- die zentralen Ablauf-Konstrukte: Funktion, Verzweigung, Schleife
- die Kommunikation mit der Außenwelt: Ein- und Ausgabe von Zahlen und Text, Grafikerstellung, Dateioperationen
- die erweiterten Datentypen: Feld, Struktur, String.

2.1 Funktionen

2.1.1 Eine Black Box

Sie (ich spreche mal die männlichen Leser an) besuchen Ihre Freundin und bringen Eier, Milch, Mehl, Zucker, Butter und Salz mit. Ihre Liebste verschwindet mit den Sachen in der Küche, während Sie im Zimmer warten müssen. Nach einiger Zeit kommt Ihre Freundin mit frischen Pfannkuchen zurück.

Diesen Vorgang kann man als „Black Box“ ansehen – eine „schwarze Schachtel“, in der abgedunkelt etwas passiert, was Sie von außen nicht beobachten können.

Abbildung 2.1 Black Box Pfannkuchen

Die Black Box hat *Eingangswerte*, alles was in die Box hineingelangt, in unserem Fall Eier, Milch, Mehl, Zucker, Butter und Salz – und *Rückgabewerte*, die Pfannkuchen, die in der Box auf geheimnisvolle Art entstehen (der Zutritt zur Küche wurde Ihnen nach leidvollen Erfahrungen bei der letzten Party verboten).

Auch Funktionen sind so eine Art Black Box. Nur beschickt man sie nicht mit Eiern, sondern mit Zahlen oder Texten, also unseren Daten. Aus dem Mathematik-Unterricht sollten Ihnen bereits mathematische Funktionen bekannt sein, wie Sinus, Kosinus oder die Wur-

zelfunktion, die man in Programmiersprachen oft mit *sqrt* (Square Root) bezeichnet. Wie bei der Black Box für die Pfannkuchen gibt es auch bei den Funktionen Eingangs- und Rückgabewerte – beispielsweise kann man die Zahl 9 als Eingabe an die sqrt-Funktion geben, die dann als Rückgabe hoffentlich die Zahl 3 liefert.

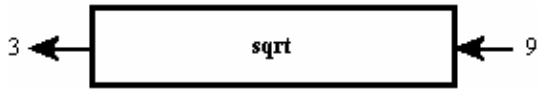


Abbildung 2.2 Black Box sqrt-Funktion

In MATLAB lautet dieser Aufruf, wie wir bereits gesehen haben, folgendermaßen:

```
>> sqrt( 9 )
ans = 3
```

oder falls man sich das Ergebnis in der Variablen *y* merken möchte:

```
>> y = sqrt( 9 )
y = 3
```

Wie MATLAB oder eine andere Programmiersprache die Wurzel von 9 berechnet, das wird dem Anwender im Allgemeinen nicht mitgeteilt. Auch hier dürfen wir die Küche der Programmierer nicht betreten und müssen darauf vertrauen, dass MATLAB einen zuverlässigen Algorithmus zur Wurzelberechnung verwendet hat.

2.1.2 Eingangs- und Rückgabeparameter

Bleiben wir noch etwas bei den Eingangs- und Rückgabewerten bzw. den dafür vorgesehenen formalen Parametern. Die meisten Funktionen haben eine fest vorgegebene Zahl von Eingangs- und Rückgabeparametern, bei der sqrt-Funktion sind es ein Eingangs- und ein Rückgabeparameter. Daneben sind die *Reihenfolge* der Parameter und deren Datentyp wichtig. Es gibt aber auch Funktionen, bei denen die Zahl und der Datentyp der Eingangswerte (Argumente) variabel ist und erst beim Aufruf der Funktion, also zur Laufzeit, bestimmt wird – durch den so genannten *varargs-Mechanismus*.

Daneben gibt es Funktionen, die gar *keine Eingangsparameter* haben – die Analogie wäre der Fall des Pfannkuchenbackens, wobei Sie selbst keinerlei Zutaten mitbringen, sondern auch das der Freundin überlassen. Diese Black Box hätte dann nur den Rückgabeparameter *Pfannkuchen*. Analog gibt es Funktionen, die *keine Rückgabeparameter* haben, was auf den ersten Blick kurios klingt: Was soll eine Funktion, die nichts zurückgibt. Diese Funktion wird aber normalerweise trotzdem etwas tun, aber wir, als Auftraggeber, bekommen nichts zurück – die Analogie wäre zum Beispiel der Auftrag, während unseres Urlaubs die Haustiere zu füttern, wovon wir selbst jedoch keine Rückmeldung bekommen. In zuverlässigen Projekten sollte aber jede Funktion etwas zurückgeben, zum Beispiel eine E-Mail, dass die Aufgabe erledigt wurde – Vertrauen ist gut, Kontrolle des Vorgangs besser.

2.1.3 Funktionen in MATLAB

In MATLAB haben Funktionen folgende allgemeine Form:

```
% H-Line (Hilfe zur Funktion)
function y = fname( x )
  % Anweisungen, Verarbeitung der Eingangsparameter
  ... x
  % Zuweisung des Rückgabewerts an den Rückgabeparameter
  y = ...
```

Zeilen im Funktions-Code, die mit einem *%-Zeichen* beginnen, sind *Kommentare*, die bei der Programmausführung nicht berücksichtigt werden. Sie dienen alleine als Hilfe für die Programmierer. Von spezieller Bedeutung ist dabei eine Kommentarzeile, die vor der Funktionsdeklaration steht. Diese so genannte *H-Line* enthält einen Hilfetext zur Funktion und wird von MATLAB angezeigt, wenn Sie im Command-Window die Anweisung „*help fname*“ eintippen.

In der H-Line dürfen vor dem %-Zeichen keine Leerzeichen stehen! Ansonsten kann die H-Line auch mehrere Zeilen lang sein, solange sie nicht durch eine Leerzeile geteilt ist. Die H-Line darf auch unterhalb der function-Zeile stehen.

Der eigentliche Funktions-Code beginnt in MATLAB mit dem Schlüsselwort *function*. Diese Zeile, der *Funktionskopf*, enthält der Deklaration der Funktion. Hier werden der Name der Funktion und die Reihenfolge der Rückgabe- und Eingangsparameter festgelegt. Der *Funktionsname* fname muss mit einem Buchstaben beginnen. Darauf kann jede beliebige Kombination von Buchstaben, Zahlen oder Unterstrichen („_“) folgen. Gibt es mehrere *Eingangsparameter* (Funktionsargumente), dann werden diese durch Kommas getrennt, zum Beispiel (*x*,*y*,*z*,…). Bei mehreren *Rückgabeparametern* verwendet man anstelle der einen Variablen *y* eine MATLAB-Liste, zum Beispiel [*a*,*b*,*c*,…], wie wir sie bereits von den Zeilenvektoren kennen. Gibt es keine Rückgabeparameter, dann entfällt der vordere Teil „*y* = ...“ in der Funktionsdeklaration, oder man ersetzt *y* durch eine leere Liste, also beispielsweise „[] = fname(...)“.

Unterhalb der Zeile mit dem Funktionskopf folgt der *Funktionskörper*. Hier beginnt eine beliebige Anzahl von Anweisungen, in denen die Eingangsparameter verarbeitet werden. Hat die Funktion einen Rückgabewert, muss in der Funktion irgendwo eine Anweisung erscheinen, die den Rückgabewert setzt, also zum Beispiel in der Form „*y* =…“. Der Programmablauf innerhalb einer Funktion erfolgt *sequentiell*, das heißt, der Programm-Code wird Zeile für Zeile nacheinander abgearbeitet – außer man trifft auf einen Verzweigungspunkt (Auswahl), eine Schleife (Iteration) oder auf eine Unterfunktion.

Für jede selbst geschriebene MATLAB-Funktion, die Sie vom Command-Window aus aufrufen möchten, erzeugen Sie im Arbeitsverzeichnis eine separate Datei. Diese muss den **gleichen Namen** tragen wie die Funktion selbst und die Endung „.m“ besitzen. Der so genannte *M-File* kann nach der Hauptfunktion zusätzlich noch weitere Funktionen (*private Funktionen*) enthalten, die aber nur innerhalb dieser Datei bekannt sind. Von außen, zum Beispiel vom Command-Window aus, können private Funktionen eines M-Files nicht aufgerufen werden.

Zum *Ausführen* Ihrer MATLAB-Funktion tippen Sie den Namen der Funktionsdatei (ohne die Endung „.m“) im Command-Window ein. In Klammern folgen die Parameter, zum Beispiel „*y* = sqrt(9)“. Wie bereits erwähnt, kann man einer Funktion als *Argumente* beliebige *mathematische Ausdrücke* übergeben, in denen auch Variablen erlaubt sind. Diesen Variablen müssen Sie vorher natürlich einen Wert zugewiesen haben:

```
>> sqrt( 4 + 5 )
ans = 3
>> a = 15;
>> sqrt( a + 1 )
ans = 4
```

Wenn Sie sich den *Rückgabewert* einer Funktion merken wollen, darf auf der linken Seite des Gleichheitszeichens jedoch nur eine *Variable* (L-Value) stehen, aber kein Ausdruck:

```
>> r = sqrt( 4 + 5 )
r =  3
>> s + 1 = sqrt( 4 + 5 )
??? s + 1 = sqrt( 4 + 5 )
          |
Error: The expression to the left of the equals sign
            is not a valid target for an assignment.
```

Im Funktionskopf „function y = fname(*x*)“ wurde die Variable *x* als Übergabeparameter festgelegt. Diese Variablen im Funktionskopf bezeichnet man als „Formalparameter“, um sie von den Werten (Argumenten) zu unterscheiden, mit denen die Funktion später aufgerufen wird. Die Argumente beim Aufruf nennt man „Aktualparameter“.

Programmieren besteht hauptsächlich darin, Funktionen zu schreiben. Die Funktionen dienen zur Verarbeitung von Daten, zum Beispiel dem Addieren von Zahlen oder der Ausgabe von Text auf dem Bildschirm. Doch woher kommen diese Daten bzw. wohin werden sie nach der Bearbeitung geschickt?

Man kann Daten direkt in der Funktion definieren, etwa durch die Zuweisung $x = 5$, die den Speicherplatz *x* mit der Zahl 5 belegt. Oder man übergibt die Daten beim Aufruf an die Funktion mittels Eingangsparameter, wie die Zahl 9 beim Aufruf *y* = sqrt(9). Daten können

auch zur Laufzeit vom Benutzer abgefragt werden, beispielweise von der Tastatur. Dieses Thema werden wir im Abschnitt Ein- und Ausgabe behandeln.

Zur Frage, wohin die Daten gehen: Daten können von einer Funktion zurückgegeben werden, zum Beispiel an die Variable *y* beim Aufruf *y* = sqrt(9). Oder man schickt Daten an ein Ausgabemedium, wie den Bildschirm, den Drucker, oder speichert sie in einer Datei auf der Festplatte.

Ein Computerprogramm, ganz gleich wie lange es auch sein mag, besteht aus solchen einzelnen Operationen, die durch Verzweigungen und Schleifen im Ablauf strukturiert werden. Eine größere Aufgabe zerlegt man dabei in kleinere Häppchen, typischerweise in der Größe von einer Textseite Programm-Code. Theoretisch könnte man ein großes Projekt auch als lange Abfolge einzelner Anweisungen anlegen, so genannten „*Spaghetti-Code*" programmieren, folgende Überlegungen sprechen jedoch dagegen:

- Einzelne, kurze Funktionen sind besser zu testen.
- Einzelne Funktionen können in weiteren Projekten (wieder-)verwendet werden.
- Nur die Zerlegung eines Projektes in Funktionen erlaubt ein sinnvolles Concurrent-Engineering, das Zusammenarbeiten mehrerer Mitarbeiter im Projekt.

Das Anlegen einer Funktion als Black Box ist insbesondere beim *Concurrent-Engineering* wichtig – man definiert nur Eingangs- und Rückgabeparameter und die gewünschte Funktionalität. Der Mitarbeiter, der die Funktion programmiert, hat dann relativ freie Hand bei deren Gestaltung. In der Systemdokumentation ist der Funktionsablauf zwar exakt beschrieben, die meisten Anwender der Funktion werden sich für die Einzelheiten jedoch nicht interessieren – genauso wenig, wie es für uns wichtig war, mit welchem Algorithmus der MATLAB-Programmierer die sqrt-Funktion implementiert hat.

2.1.4 Funktionsbeispiel: Umfang

Jetzt wird es aber Zeit für ein lauffähiges Beispiel einer MATLAB-Funktion. Dazu brauchen wir einen Funktionsnamen, die Eingangs- und Rückgabeparameter und natürlich die Beschreibung, was die Funktion tun soll:

- *Aufgabe* der Funktion: Die Funktion soll nach folgender Formel aus dem Kreisradius den Umfang berechnen: *Umfang* = 2 * π * *Radius*.
- *Name* der Funktion: Umfang.
- *Eingangsparameter*: *r*, der Radius, eine reelle Zahl.
- *Rückgabeparameter*: *u*, der berechnete Umfang.

Als *Hilfetext* wählen wir:

```
u = Umfang(r): Berechnung des Kreisumfangs u aus dem Kreisradius r
```

Der Funktions-Code muss in eine ASCII-Datei geschrieben werden. Hierzu verwenden wir den in MATLAB integrierten *Editor*, der im MATLAB-Menü unter „File+New+Script" gestartet wird. Tippen Sie dort hinein den Programm-Code unserer Funktion Umfang:

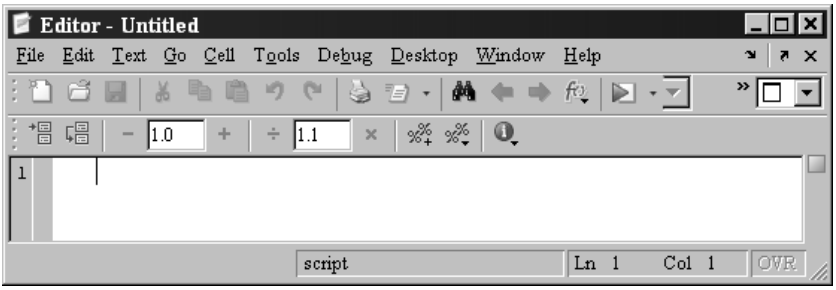


Abbildung 2.3 Der MATLAB-Editor

Listing 2.1 function Umfang

```
% u = Umfang(r):
%   Berechnung des Kreisumfangs u aus dem Kreisradius r
function u = Umfang( r )
   erg = 2*pi*r;     % Berechnung des Umfangs aus Eingangswert r
   u   = erg;        % Zuweisung an Rückgabewert u
```

Speichern Sie den M-File mit dem Programm-Code unter dem Namen *Umfang.m*.

In der ersten Zeile steht der Hilfetext, dann folgt der Funktionskopf mit der Deklaration des Funktionsnamens und der Parameter. Die dritte Zeile enthält die erste ausführbare Anweisung des Programms: „*erg* = 2**pi***r*;“. Hier steht rechts der Ausdruck (2**pi***r*) und links die Variable *erg*. Diese *Wertzuweisungen* werden von *rechts nach links* interpretiert. Zuerst wird der Ausdruck (2**pi***r*) auf der rechten Seite berechnet, dies jedoch in der Reihenfolge von links nach rechts. Die Zahl 2 wird also mit der MATLAB-Konstanten *pi* multipliziert und dann das Ergebnis mit dem übergebenen Radius aus der Variablen *r* multipliziert. Dieser berechnete Wert wird anschließend der neu definierten Variablen *erg* auf der linken Seite zugewiesen. Das Semikolon am Zeilenende dient dazu, dass MATLAB bei der Programmausführung die Protokollierung unterdrückt.

In der letzten Zeile „*u* = *erg*;“ greift die Funktion wieder auf den in der Variablen *erg* gespeicherten Wert zu und kopiert ihn in den Rückgabeparameter *u*. Ohne diese Zeile würde die Funktion zwar den Wert des Umfangs berechnen, aber der Aufrufer der Funktion bekäme ihn nicht zu sehen.

Um die einzelnen Operationen, Berechnung und Rückgabe, klarer herauszuarbeiten, wurden sie in diesem Beispiel in getrennten Zeilen ausgeführt. Sie können die beiden Zeilen aber auch zu einer einzigen *zusammenfassen* und auf die Zwischenvariable *erg* verzichten:

```
% Berechnung und Rückgabe des Umfangs
u = 2*pi*r;
```

Im Command-Window von MATLAB *testen* wir die neue Funktion:

```
>> rad = 1.0;
>> umf = Umfang( rad );
>> umf
umf = 6.2832
```

Die Semikolons am Ende der beiden ersten Zeilen dienen wieder dazu, die Kontrollausgabe zu unterdrücken. Durch die dritte Zeile wird der Wert von *umf* ausgegeben.

Als Formalparameter für die Übergabe haben wir in der Funktionsdefinition von Umfang die Variable *r* verwendet. Beim Aufruf wurde das Argument *rad* mit dem Wert 1.0 als Aktualparameter an die Funktion übergeben. Wir hätten die Funktion auch direkt mit dem Wert 1.0 aufrufen können, also „*umf* = Umfang(1.0);“. Für die Rückgabe wählten wir ebenfalls unterschiedliche Namen für die formalen und die aktuellen Variablen.

2.1.5 Stack, Funktionsparameter

Es macht also keine Probleme, wenn wir beim Funktionsaufruf für Eingangs- und Rückgabevariablen (hier *rad* und *umf*) *andere Namen* verwenden, als im Funktionskopf des M-Files Umfang.m angegeben sind. Dies ist eine typische Eigenschaft für alle Funktionen. Für den *Funktionsaufruf* (hier „>> *umf* = Umfang(*rad*);“) brauchen Sie weder zu wissen, wie der Umfang in der Funktion berechnet wird, noch welche Namen der Programmierer innerhalb der Funktion für die Eingangs- und Rückgabeparameter verwendet hat. Und Sie müssen nicht notwendigerweise Variablen für die Eingangsparameter der Funktion anlegen, sondern können die Daten auch direkt übergeben.

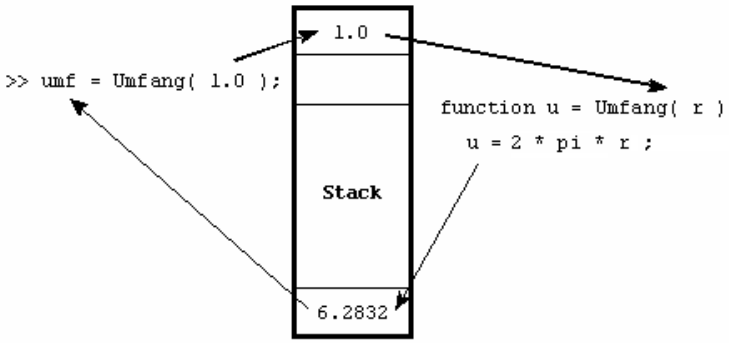


Abbildung 2.4 Stack zur Wertübergabe

Beim Funktionsaufruf werden zur Übergabe der Eingangsparameter an die Funktion gar keine Variablen verwendet. Die Daten werden stattdessen in der angegebenen Reihenfolge nacheinander in einen speziellen Speicherbereich gepackt – den *Stack*. Die aufgerufene Funktion liest die Daten aus dem Stack aus, verarbeitet sie und schreibt das Ergebnis der

Berechnungen ebenfalls in einen bestimmten Teil des Stacks zurück, von wo der Rückgabewert nach dem Funktionsaufruf abgeholt werden kann.

Wenn Sie von MATLAB aus andere MATLAB-Funktionen aufrufen, müssen Sie sich um die Art der Parameterübergabe nicht kümmern, solange Sie die korrekten Datentypen verwenden. Wie bereits erwähnt, ist MATLAB nicht typsicher. Es gibt also keinen automatischen Check, der verhindert, dass Sie beim Funktionsaufruf beispielsweise einem double-Parameter einen String zuweisen. Mehr noch – der Datentyp eines Funktionsparameters wird bei der Funktionsdefinition nicht einmal vereinbart.

Sie können von den meisten Programmierumgebungen aus auch Funktionen aufrufen, die in einer anderen Programmiersprache, beispielsweise in C, Java oder FORTRAN, geschrieben sind und als übersetzte Bibliotheks-Funktionen vorliegen (siehe Abschnitt 4.9 „MEX – C in MATLAB“). Hierbei müssen Sie sich aber ein paar mehr Gedanken zur Datenübergabe machen, insbesondere im Fall von komplexeren Datentypen wie mehrdimensionalen Arrays oder Strukturen.

In der Funktion Umfang haben wir eigene Variablen erzeugt, zum Beispiel die Variable *erg* bei der Berechnung des Umfangs. Kommt es da nicht zu Problemen, wenn auch in anderen Funktionen oder im Workspace eine Variable *erg* verwendet wird? Nein, denn auf die Variablen einer Funktion, so genannte „lokale Variablen“, kann man nur innerhalb dieser Funktion zugreifen. Ihr „*Scope*“ ist die Funktion. Eine andere Variable *erg*, die zum Beispiel im Workspace definiert ist, hat einen eigenen Speicherbereich und wird von MATLAB separat adressiert und verwaltet. Für die Funktionsparameter, hier *r* und *u*, gilt eine analoge Aussage. Ihr Scope ist ebenfalls auf die Funktion beschränkt. Hat das Programm eine Funktion verlassen, sind die Variablen und die Parameter der Funktion nicht mehr verfügbar. Deshalb können Sie Werte, die Sie in einer Funktion berechnet haben, nur dann außerhalb der Funktion weiter verwenden, wenn Sie diese über Rückgabeparameter aus der Funktion „exportieren“, in unserem Bespiel durch die Zuweisung des Wertes in *erg* an den Rückgabeparameter *u*, der dann über den Stack an die Variable *umf* im Workspace gelangt. Eine Ausnahme gibt es allerdings: Sie können eine Variable als *global* definieren, wie es beispielsweise im Spielprojekt im Kapitel 4 geschieht. So eine globale Variable kann dann in beliebigem Kontext verwendet werden, führt aber zu einem langsameren Programmablauf.

2.1.6 Ablaufprotokoll

Um den Ablauf unserer Funktion genauer zu verstehen, wollen wir die Zeilen der Funktion einmal einzeln durchgehen und unser Augenmerk darauf richten, wo welche Daten anfallen und gespeichert werden. Dazu dient das Ablaufprotokoll, das protokolliert, wie ein Computer unser Programm ausführen würde – nacheinander, Zeile für Zeile. Zur Orientierung haben wir die Zeilen des Funktions-Codes mit Nummern versehen. Der Aufruf der Funktion mit dem Wert 1.0 wurde der Vollständigkeit halber als Zeile 0 hinzugefügt, da hier die Übergabe an den Stack stattfindet.

Index

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z