OTH Regensburg

Studiengang
IN, IT B. Sc.
Prof. Dr. Martin Weiß
Sommersemester 2019

Computerarithmetik und Rechenverfahren

2. Praktikum: Grundlagen MATLAB, Zahldarstellungen

2.1. *** TEX/LATEX-Code aus MATLAB erzeugen

Start kann folgendes Code-Fragment sein:

```
A = zeros(5,3);
v = 1:15;
A(:) = v;
disp(A);

% latex ist ein Kommando der Symbolic Toolbox und erwartet
% Datentyp class 'sym', selbst wenn keine Symbole in der Matrix
% enthalten sind
s1 = latex(sym(A))

% Symbole deklarieren
% äquivalent: x = sym('x'); y = sym('y'); z = sym('z');
syms x y z
% Funktion als class sym-Objekt
f = sin(x)+ y*x^3
s2 = latex(f)
% partielle Ableitungen ausrechnen; es wird der MAPLE-Kern aufgerufen
fx = diff(f, 'x')       % wenn kein Symbol x deklariert ist
fx = diff(f, x)
fy = diff(f, y)
s3 = latex(fx)

B = [1 x x^2 x^3;
     1 y y^2 y^3;
     1 z z^2 z^3];
s4 = latex(sym(B))

C = A/2
latex(sym(C))           % MATLAB 2012: wohl nicht das gewünschte Ergebnis
% latex(sym(rats(C)))   % Problem: string to sym
% latex(sym(rat(C)))    % Problem: rat to sym
```

Sie können auch TEX/LATEX-Kommandos zur Beschriftung verwenden:

```
set(groot, 'defaultAxesTickLabelInterpreter','latex');
set(groot, 'defaultLegendInterpreter','latex');
figure(1);clf
x = 0:0.01:2*pi;
f1 = sqrt(x);
f2 = 3/2*sin(10*pi*x);
plot(x,f1,x,f2);
str = '$$ \frac 3 2\sin(10 \pi x_1) $$';
text(2.25,2,str,'Interpreter','latex','FontSize',20)
legend('$\sqrt{x_1}$', '$\frac 3 2\sin(10 \pi x_1)$')
% einfache Indizes funktionieren auch so
xlabel('x_1');ylabel('x_2 ');
shg
```

2.2. *** TEX/LATEX-Code aus MATLAB erzeugen: selber machen

Erzeugen Sie mit LATEXdie Formeln:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \qquad \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} \\ \frac{1}{2} & \frac{1}{4} & \frac{1}{5} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \end{bmatrix}, \qquad \frac{\partial^2}{\partial x \partial y} f(x,y) = \frac{\partial^2}{\partial y \partial x} f(x,y), \qquad \int x^2 \,\mathrm{d}x = \frac{x^3}{3} + C$$

Verwenden Sie den LATEX-Rahmen `smybolic.tex` aus dem GRIPS / K-Laufwerk, getestet mit IDE TeXworks (Zeichensatz UTF-8).

MATLAB kann eine kleine Teilmenge der gängigsten LATEX/TeX-Konstrukte verstehen, und so Beschriftungen für Graphen usw. professioneller setzen:

2.3. Debuggen in der MATLAB-IDE

Debuggen heisst Fehler suchen und bereinigen. Die wichtigsten Tasten in MATLAB sind:

- F12: "breakpoint" setzen oder löschen
- F10: "step over" aktuelle Zeile ausführen. Wenn die Zeile einen Funktionsaufruf enthält, wird *nicht* in die Funktion gesprungen. Bei MATLAB entspricht eine Zeile u.U. weniger als einem Kommando, etwa nur dem Aufbau einer Zeile einer Matrix statt der Zuweisung an die ganze Matrix.
- F11: "step into" aktuelle Zeile ausführen. Wenn die Zeile einen Funktionsaufruf enthält, wird in die Funktion gesprungen.
- F5: "run" Programm weiter ausführen bis zum nächsten breakpoint oder zum Ende
- (F9: "run selection" den im Editor markierten Bereich ausführen)

Läuft Ihr Programm auf einen breakpoint, ändert sich der prompt in der Shell auf `K»`. Sie können jetzt Variableninhalte prüfen und ändern, über den *function call stack* auch in anderen Funktionen der Aufrufhierarchie. Mit *Quit Debugging* können Sie den debug-Modus verlassen.

2.4. Ägyptische Multiplikation

Implementieren Sie eine MATLAB-Funktion zur "Russischen Bauernmultiplikation" (auch ägyptisches Multiplizieren, Abessinische Bauernregel oder Verdopplungs-Halbierungs-Methode) nichtnegativer ganzer Zahlen. Verwenden Sie einen der Datentypen `uint8, uint16, uint32, uint64`.

Algorithmus 1 Multiplikation im Binärsystem: berechne $p := a * b$

Eingabe: $a, b \in \mathbb{N}_0$
  $p := 0$
  while $b \neq 0$ **do**
    if b ungerade **then**
      $p := p + a$
    end if
    $a := a * 2$
    $b := b \div 2$
  end while

Machen Sie sich die Funktionsweise des Algorithmus anhand eines Beispiels klar:

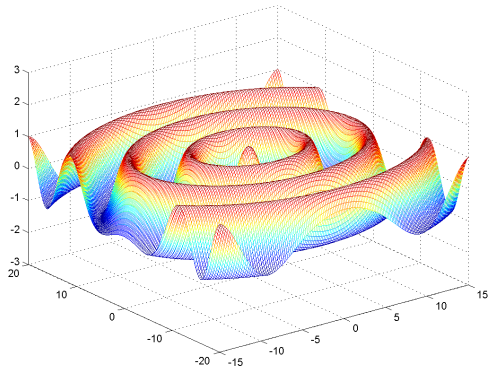
```
Binär bestätigt: 42*13 = 546
     101010 * 1101 =
     ------------
      101010
       101010
        000000
+        101010
     ------------
     1000100010
```

Interpretation:
Multiplikation zurückgeführt auf Addition und Multiplikation mit 2 (shift)

Ägyptische, abessinische, … Multiplikation

Hinweis: Funktionen `bitand`, `bitshift`

2.5. Erzeugen Sie folgende Graphik der Funktion $f : \mathbb{R}^2 \to \mathbb{R}$, $(x, y) \mapsto \cos(\sqrt{x^2 + y^2})$ *ohne* Verwendung von Schleifen:



Hinweise: komponentenweise Operatoren wie `.*`, Kommandos `meshgrid`, `mesh`, `rotate3d`, `surf`, `surface`.

Ändern Sie die Funktion so ab, dass ein "schwarzes Loch"entsteht, oder die Wellen wie bei einem ins Wasser geworfenen Stein nach außen abklingen!

2.6. Schreiben Sie MATLAB-Funktionen zur Umwandlung in das IEEE 754-Format für einfache und doppelte Genauigkeit:

```
[Signum, Characteristic, Mantissa, String] = IEEE754Float(x)
[Signum, Characteristic, Mantissa, String] = IEEE754Double(x)
```

Der Rückgabewert `String` soll die Bit-Codierung als Zeichenkette liefern. Schreiben Sie eine Basis-Funktion, die durch Angabe der bit-Anzahl für Charakteristik und Mantisse auf die gewünschten Funktionen spezialisiert werden kann.

2.7. Lokale und geschachtelte Funktionen

Bisher hatten wir immer genau eine Funktion in einem `.m`-File (oder ein Skript, das gar keine Funktion enthält). Man kann aber in MATLAB auch mehrere Funktionen in einer Datei definieren. Davon ist die erste bzw. die äußerste die Hauptfunktion, die von der Shell aus oder für andere Funktionen sichtbar ist, alle anderen Funktionen sind lokal.

Rufen Sie mit `doc function` die Hilfeseite zu Funktionen auf. Lesen Sie die Weiterleitung zu "local functions" und "nested functions", ganz unten im Hilfetext. Bestätigen Sie, dass nur die Hauptfunktion jeweils von der Shell aus aufrufbar ist.

Erweitern Sie das Beispiel zu "nested functions" um Variablen mit gemeinsamem Sichtbarkeitsbereich:

```
function parent(x)
y = x*x;
disp('This is the parent function')
disp(sprintf('of course I know about x=%g and y=%g', x, y));
nestedfx

   function nestedfx
      disp('This is the nested function')
      disp(sprintf('I also know about x=%g and y=%g', x, y));
   end

end
```

Seit MATLAB 2017 möglich: `.m`-Datei beginnt als Skript, enthält lokale Funktionen am Ende.

2.8. Lokale und geschachtelte Funktionen: selber machen

Schreiben Sie eine Funktion, die die Graphen der Funktionen sin, cos, arctan in den Farben rot, grün, blau mit `LineWidth` 2 im Bereich $x \in [-2\pi, \pi]$ darstellt, auf der y-Achse soll die Darstellung den Bereich $[-1.5, 1.5]$ umfassen.

Schreiben Sie eine Hauptfunktion und eine lokale Hilfsfunktion

`function plotCol2(x, y, col)`

die die Daten wie gefordert darstellt.

Bemerkung: Funktionen als Argumente anderer Funktionen werden in der nächsten Aufgabe erklärt.

2.9. Mehr MATLAB

Erarbeiten Sie den Code `moreMATLAB.m`, `packdata.m`, `unpackdata.m` im K-Laufwerk unter `K:\Wma\CR\Praktikum_2`.

2.10. ***

Bestimmen Sie in MATLAB für Zahlobjekte einfacher und doppelter Genauigkeit:

a) die relative Maschinengenauigkeit, indem Sie die betraglich kleinste Potenz von 2 ermitteln, für die in IEEE-754-Arithmetik gilt:

```
1 + x ~= 1
```

b) die betragsmässig größte und kleinste darstellbare Zahl.

Vergleichen sie mit den Werten, die MATLAB über die Funktionen `eps`, `realmin`, `realmax` liefert. Bestimmen Sie über den Logarithmus zur Basis 2

$$\log_2(x) = \frac{\log(x)}{\log(2)}$$

den Exponenten der Darstellung $x = 2^k$ mit $k = -23$ bzw. $k = -52$ für `float` und `double`. Bestätigen Sie so die IEEE 754-Darstellung. Vergleichen Sie mit den Werten, die Sie für verschiedene Datentypen einer höheren Programmiersprache Ihrer Wahl (C, C++, Java) erhalten.

In C++ können Sie die Funktionen aus `std::numeric_limits` verwenden, siehe `http://www.cplusplus.com/reference/limits/numeric_limits/`. In C gibt es kein standardisiertes Äquivalent.

In Java gibt es die Standard-Bibliotheksfunktion `java.lang.Math.ulp`, aber das IEEE-Format ist hier ohnehin Sprachstandard, und die Werte $\varepsilon_{\text{mach}} = 2^{-23}$ und $\varepsilon_{\text{mach}} = 2^{-52}$ können als gegeben angesehen werden.

In MATLAB liefert `eps` die Maschinengenauigkeit bei double precision, alternativ kann der Datentyp mit `eps('single')` oder `eps('double')` angegeben werden.

Mit `eps(x)` erhält man den Abstand von `x` zur nächsten IEEE-Zahl, `eps` ist äquivalent zu `eps(1.0)`.

Typische Abragen zur Gleichheit von Fließkommazahlen werden in der Form

```
if abs(xs - x) < C*eps(x)
   ...
```

formuliert: Das ist durch die Abschätzung

$$\frac{|\text{rd}(x) - x|}{|x|} < \varepsilon_{\text{mach}}$$

motiviert: Betrachte Zahlen als identisch, die sich nur um ein kleines Vielfaches der Maschinengenauigkeit unterscheiden, etwa $C = 10$ für "gleich bis auf 10-faches der Maschinengenauigkeit".