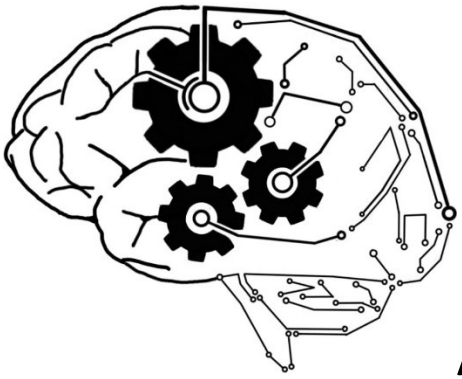


AI Engineering

Einführung in MATLAB

Open Educational Resource

Autor(en)

Uwe Knauer, Hochschule Anhalt

Aufbau und Funktionsweise des Programms MATLAB

Das Programm MATLAB ist eine Arbeits- und Entwicklungsumgebung, die Studierenden vieler Universitäten und Hochschulen für die Dauer des Studiums kostenfrei zur Verfügung steht. Dazu muss ein Account bei der Fa. MathWorks unter Verwendung der Hochschul-E-Mailadresse erstellt werden. Sie können die Software MATLAB dann herunterladen und auf Ihrem persönlichen Rechner installieren. Weiterhin kann ein großer Teil der Funktionen von MATLAB auch ohne Installation im Browser (über die Webseite der Firma Mathworks) genutzt werden. Die Software ist nur in englischer Sprache verfügbar.

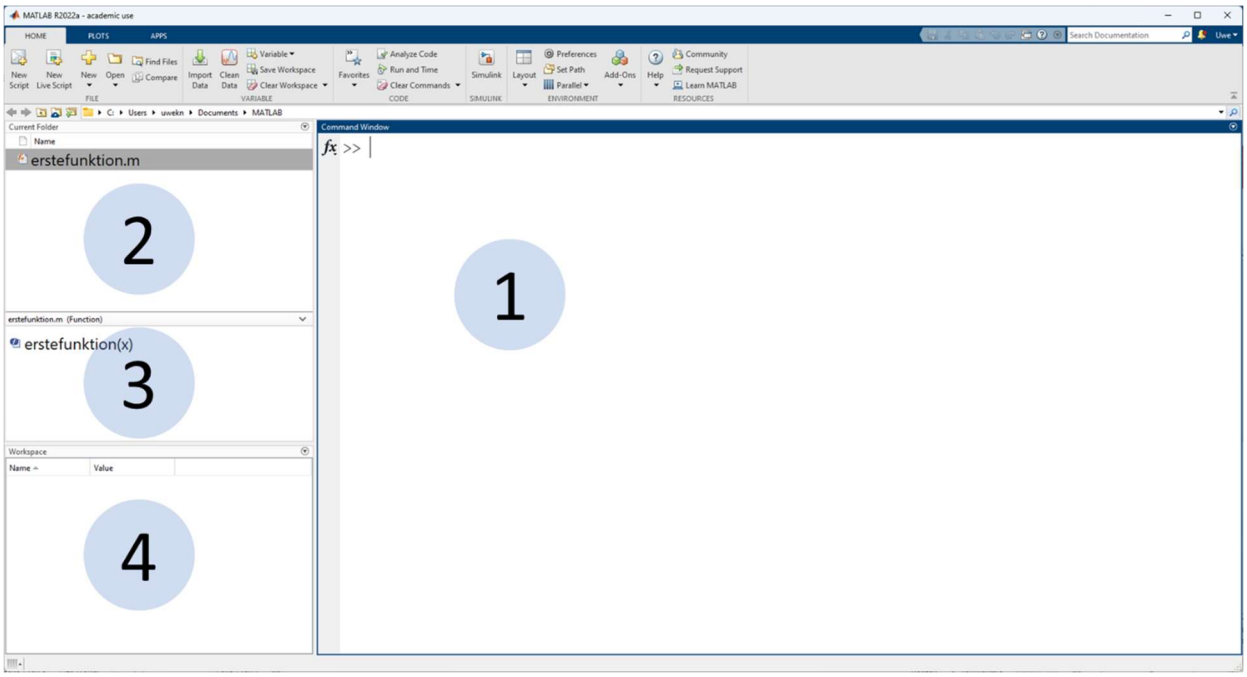
Informationen zur Verfügbarkeit von MATLAB an Ihrer Hochschule finden Sie hier:

https://de.mathworks.com/products/matlab/student.html

Die Software ermöglicht die Erstellung und Durchführung komplexer Datenanalysen und darüber hinaus die Erstellung eigener Programme. Sie sollen MATLAB als Werkzeug kennenlernen und verwenden, wiederholbar und anpassbar Datenanalysen und Datenvisualisierungen zu erstellen.

Dabei lernen Sie Grundkonzepte der Programmierung kennen, die Sie auch in R, Python und anderen Programmiersprachen wiederfinden. Damit soll Ihnen künftig der leichtere Einstieg in andere Umgebungen ermöglicht werden.

Nach dem Start von Matlab öffnet sich die Programmoberfläche.



In der Abbildung sind vier Bereiche der Programmoberfläche gekennzeichnet. Die Zahl 1 kennzeichnet das sogenannte Command Window. Hier können Anweisungen eingegeben und die Ergebnisse der Verarbeitung eingesehen werden.

Probieren wir es aus:

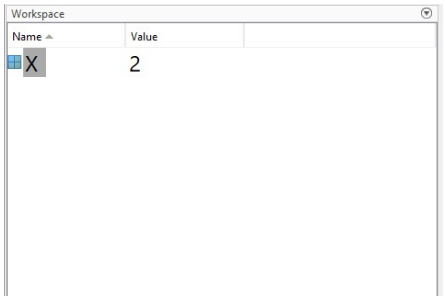
Die Eingabe 1+1 liefert das Ergebnis 2, das direkt angezeigt wird. Wir können MATLAB also wie einen Taschenrechner benutzen.

Unsere Ergebnisse können wir auch in sogenannten Variablen (können verschiedene Werte annehmen) speichern.

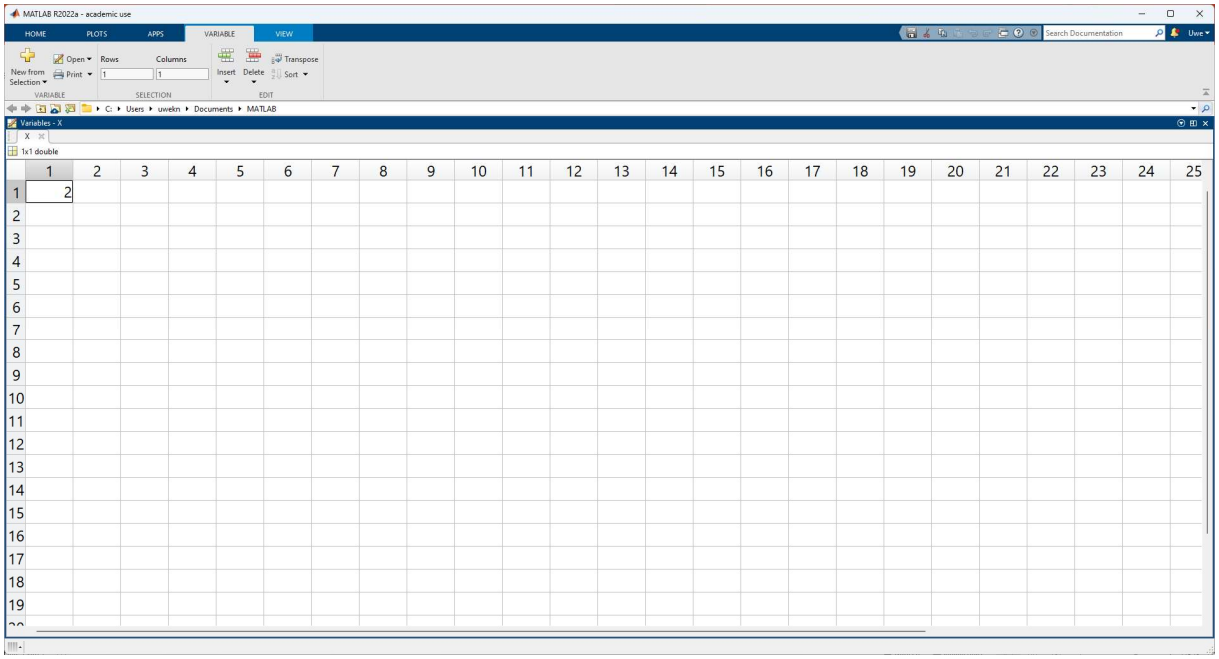
Probe:

```
X=1+1
```

Speichert das Ergebnis 2 in der Variablen X. Diese taucht nun im Bereich 4 auf, dem sogenannten Workspace. Dort werden die aktuell von MATLAB im Hauptspeicher des Computers (RAM, Random Access Memory) angelegten Variablen angezeigt.



Wählt man dort eine Variable durch Doppelklick aus, öffnet sich ein Variablen-Editor mit dem die Werte der Variablen eingesehen und geändert werden können.



Im Bereich 2 sieht man Dateien im aktuellen Verzeichnis. Der Pfad zu diesem Verzeichnis ist als Zeile oberhalb der Bereiche 1 und 2 angezeigt. Dort kann man das aktuelle Verzeichnis (Arbeitsverzeichnis) auch ändern.

Im Bereich 3 werden Zusatzinformationen angezeigt, wenn man im Bereich 2 eine Datei auswählt.

Abschließend die Bemerkung, dass das Erscheinungsbild der MATLAB-Programmoberfläche angepasst werden kann. Die verschiedenen Bereiche können mit der Maus neu angeordnet werden, ausgeblendet werden und es können auch weitere Elemente hinzugefügt werden.

Datentypen in MATLAB

MATLAB steht für *Matrices Lab*. Einen grundlegenden Datentyp stellen damit Matrizen dar. Darunter verstehen wir ein- oder mehrdimensionale Felder elementarer Datentypen.

Einzelne Werte werden als Skalar in einer 1x1 Matrix gespeichert.

Zeilenvektoren sind 1xN Matrizen.

Spaltenvektoren sind Nx1 Matrizen.

NxM Matrizen haben N Zeilen und M Spalten.

Matlab kann darüber hinaus auch mit höherdimensionalen Matrizen arbeiten.

Beispiele:

Die Zuweisung M1=1.2 erzeugt eine 1x1 Matrix, die nur den Wert 1.2 enthält.

Die Zuweisung M2=[1 2 3] erzeugt eine 1x3 Matrix mit den Werten 1, 2 und 3. Gleichzeitig handelt es sich um einen Zeilenvektor.

Die Zuweisung M3=[1; 2; 3] erzeugt eine 3x1 Matrix mit den Werten 1, 2 und 3. Gleichzeitig handelt es sich um einen Spaltenvektor.

Die Zuweisung M4=[1 2 3; 4 5 6; 7 8 9; 10 11 12] erzeugt eine 4x3 Matrix mit den Werten von 1 bis 12, die in 4 Zeilen und 3 Spalten angeordnet sind.

Die Zuweisung M5=[ ] erzeugt eine leere Matrix.

Der Standarddatentyp ist double, also eine reelle Zahl mit doppelter Genauigkeit (64 Bit). Die erste Zuweisung eines Wertes bestimmt den Datentyp der Matrix.

Elementare Operationen in MATLAB

Die arithmetischen Operationen können auf Matrizen angewendet werden.

M6 = M1 + 0.8 addiert zum Wert 1.2 den Wert 0.8. In der 1x1 Matrix M6 ist nach der Operation der Wert 2 gespeichert.

M7 = M2 – 1 subtrahiert von allen Werten in M2 den Wert 1. M7 ist daher 1x3 Matrix mit den Werten 0, 1 und 2.

M8 = M2 – [1 2 3] führt eine elementweise Subtraktion durch. M8 ist daher eine 1x3 Matrix mit den Werten 0, 0 und 0.

M9=M2 * 2 führt eine Multiplikation durch. In M9 sind die Werte 2, 4 und 6 gespeichert.

M10= M2 * [1 ; 1 ; 0] führt eine Matrixmultiplikation zwischen einem Zeilen- und einem Spaltenvektor durch. Das Ergebnis ist das Skalarprodukt beider Vektoren. M10 hat den Wert 3.

M11 =M3* [1 2 3] führt eine Matrixmultiplikation zwischen einem Spalten- und einem Zeilenvektor durch. Das Ergebnis ist eine 3x3 Matrix.

1 2 3

2 4 6

3 6 9

Bei den Grundrechenarten +,-,* und / wird zwischen Matrixoperationen, wie der eben beschriebenen, und elementweisen Operationen unterschieden.

M12=M3.*M3 multipliziert M3 elementweise mit sich selbst. Jedes Element der Matrix M12 ist also das Quadrat des gleichen Elements in M3.

Die Exponentialfunktion wird durch ^ realisiert. 2^8 liefert das Ergebnis 256.

Die Berechnungsergebnisse werden in der Variablen gespeichert, der sie zugewiesen wurden. Erfolgte keine Zuweisung, z.B. wurde M2 – 1 statt M7=M2-1 eingegeben, dann wird das letzte Berechnungsergebnis in der Variablen ans (Abkürzung für answer) gespeichert.

MATLAB-Skripte

Alle Befehle, die in der Konsolenansicht eingegeben und ausgeführt werden, können auch in Skripten zusammengefasst und nacheinander ausgeführt werden. Skripte sind Textdateien mit der Dateiendung .m

Skripte nutzen die aktuelle Arbeitsumgebung. Auf die im aktuellen Workspace vorhandenen Variablen kann zugegriffen und diese können durch das Skript verändert werden. Nach Abarbeitung eines Skripts steht der aktuelle Zustand des Workspace für weitere Operationen zur Verfügung.

Skripte werden durch Eingabe ihres Namens ohne die Dateiendung gestartet.

Um ein Skript anzulegen, kann der eingebaute Editor verwendet werden.

Durch Eingabe von

edit aufraeumen.m

wird ein neues Skript angelegt und es öffnet sich der Editor. Nun notiert man in der Datei aufraeumen.m alle Befehle, die nacheinander ausgeführt werden sollen.

In unserem Beispiel sind das die folgenden beiden Befehle:

```
clear

clc
```

Der Befehl clear löscht alle Variablen im Workspace und der Befehl clc löscht den bisherigen Inhalt des Command Window.

Damit man sich dies besser merken kann, nutzen wir die Möglichkeit, Skripte zu kommentieren. Darunter versteht man die Eingabe von Texten im Skript, die aber nicht als Befehle interpretiert werden. Hierfür dient das Prozent-Zeichen:

```
clear %Dieser Befehl löscht alle Variablen

clc %Dieser Befehl löscht die Ein- und Ausgaben im Command Window
```

Speichern Sie das Skript und führen Sie es durch Eingabe seines Namens aufraeumen aus!

MATLAB-Funktionen

Im Unterschied zu Skripten können Funktionen nicht auf die Variablen im Workspace, also der aktuellen Arbeitsumgebung zugreifen. Funktionen bekommen alle Eingaben immer beim Aufruf übergeben und liefern definierte Ergebnisse zurück. Alle Zwischenergebnisse, die beim Abarbeiten

einer Funktion entstehen, werden nach Ausführung des letzten Befehls und Beenden der Funktion automatisch gelöscht.

Funktionen werden genauso wie Skripte als Dateien mit der Erweiterung .m angelegt.

Der Befehl edit visualisiereSinus.m legt eine neue Datei an. Um kenntlich zu machen, dass es sich um eine Funktion und kein Skript handelt, muss im Editor am Anfang der Datei das Schlüsselwort

function

gefolgt vom Namen der Funktion eingetragen werden.

function visualisiereSinus

Die angesprochenen Eingabewerte (Argumente) der Funktion legen wir in Klammern fest. Da wir eine eigene Funktion erstellen, können wir diese Liste von Argumenten frei nach unseren Bedürfnissen gestalten.

function visualisiereSinus(winkel)

bedeutet, dass die Funktion beim Aufrufen die Übergabe einer Variablen oder eines Wertes für den Winkel erwartet. Später also die Eingabe von visualisiereSinus(0) oder visualisiereSinus([0 : 0.1 :2*pi]) das gewünschte Ergebnis liefern soll. In beiden Fällen wird nur eine Variable übergeben!

visualisiereSinus(0) die 1x1 Matrix mit dem Wert 0

visualisiereSinus([0:0.1:2*pi]) die 1xN Matrix mit den Werten von 0 bis 6.28 in Schritten von 0.1.

Als ersten Befehl in unserer Funktion soll der Sinus für die Werte in der Variablen winkel berechnet werden. Dazu greifen wir auf die in Matlab bereits vorhandene Funktion sin zurück.

S=sin(winkel)

liefert den Winkel zurück und speichert ihn in der Variablen S. Da diese Variable nur während des Aufrufs der Funktion visualisiereSinus existiert und danach verschwindet, nennen wir sie eine lokale Variable oder eine Variable mit lokaler Sichtbarkeit (nur innerhalb der Funktion kann auf sie zugegriffen werden).

Ein Testaufruf unserer Funktion schreibt im Command Window alle berechneten Werte aus. Da wir das oft nicht möchten, können wir am Ende eines jeden Befehls ein Semikolon setzen. Dieses unterdrückt die Anzeige, führt den Befehl ansonsten genauso aus. Manchmal ist es aber hilfreich, während der Erstellung einer Funktion auf das Semikolon zu verzichten, um die Zwischenergebnisse zu sehen und nach Fehlern zu suchen.

S=sin(winkel);

Nun soll die Funktion auch eine grafische Ausgabe des Sinus erzeugen. Immerhin haben wir sie visualisiereSinus genannt.

In der nächsten Zeile ergänzen wir deshalb den Befehl plot. Auch das ist eine vorhandene MATLAB-Funktion.

plot(winkel,S)

verwendet die Werte in der Variablen winkel als X-Werte und die Werte in der Variablen S als Y-Werte. In einem Diagramm zeichnet er die Wertepaare ein und verbindet sie durch eine Gerade.

Durch weitere Argumente beim Funktionsaufruf der plot-Funktion kann das Aussehen des Graphen angepasst werden.

Eine schnelle Möglichkeit bietet die Angabe einer Formatierungszeichenkette als drittes Argument der Funktion.

plot(winkel,S,‘r:‘)

zeichnet eine rote gestrichelte Linie anstelle der blauen durchgezogenen Linie.

Rufen Sie mit doc plot aus dem Command Window eine Übersicht aller Optionen der Plot-Funktion auf, um sich einen Überblick über die Möglichkeiten der Formatierung zu verschaffen.

Zusammenfassend sieht die erstellte Funktion nun wie folgt aus:

function visualisiereSinus(winkel)

S=sin(winkel);

plot(winkel,S,‘r:‘);

Wir ergänzen einen Kommentarblock gleich nach der ersten Zeile mit dem Schlüsselwort function

```
function visualisiereSinus(winkel)
%Die Funktion visualisiereSinus wurde im Informatik-
Praktikum der Hochschule Anhalt erstellt und %zeichnet zu
den übergebenen Winkeln im Bogenmaß die zugehörigen Werte
der Sinus-Funktion.
% Aufruf: visualisiereSinus(WINKEL)
% Winkel ist ein Zeilen- oder Spaltenvektor mit den
Winkeln im Bogenmaß
% Beispiel: visualisiereSinus( [0:0.1:2*pi] );
S=sin(winkel);
plot(winkel,S,'r:');
```

Wenn nun im Command Window help visualisiereSinus eingegeben wird, so wird der Text des Kommentarblocks als Hilfetext ausgegeben. Dies bietet eine einfache Möglichkeit, die Nutzung eigener Funktionen zu erklären und zu dokumentieren.

Testen wir nun die Funktion mit verschiedenen Aufrufen:

visualisiereSinus (0:2*pi)

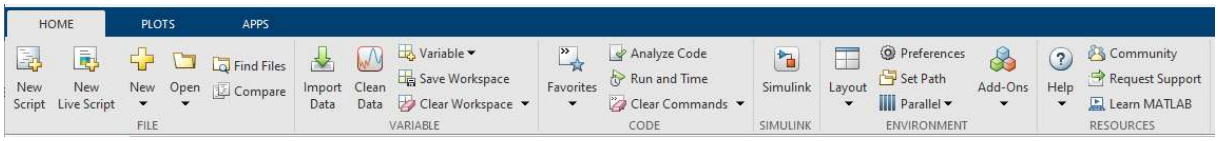
visualisiereSinus(0:0.1:2*pi)

visualisiereSinus(0:0.1:4*pi)

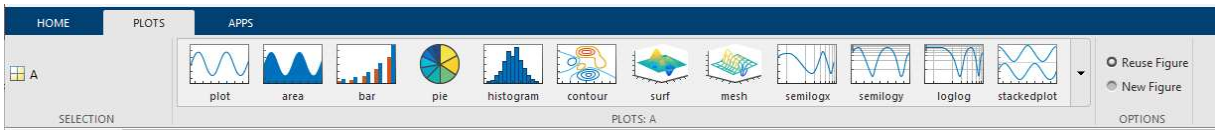
Bemerkung: In den Beispielaufrufen haben wir das Schlüsselwort **pi** verwendet. Dahinter verbirgt sich die Konstante $\pi \approx 3.14$. Diese Konstante (der Wert kann nicht durch den Nutzer oder Programmierer geändert werden) ist Bestandteil von MATLAB.

Datenvisualisierung mit MATLAB

Sicher ist Ihnen die Menüleiste im oberen Bereich der Programmoberfläche aufgefallen. Hier sind verschiedene Funktionen und MATLAB-Apps direkt zugänglich.



Skripte und Funktionen können hier auch per Mausklick (ohne Aufruf von edit) angelegt werden.

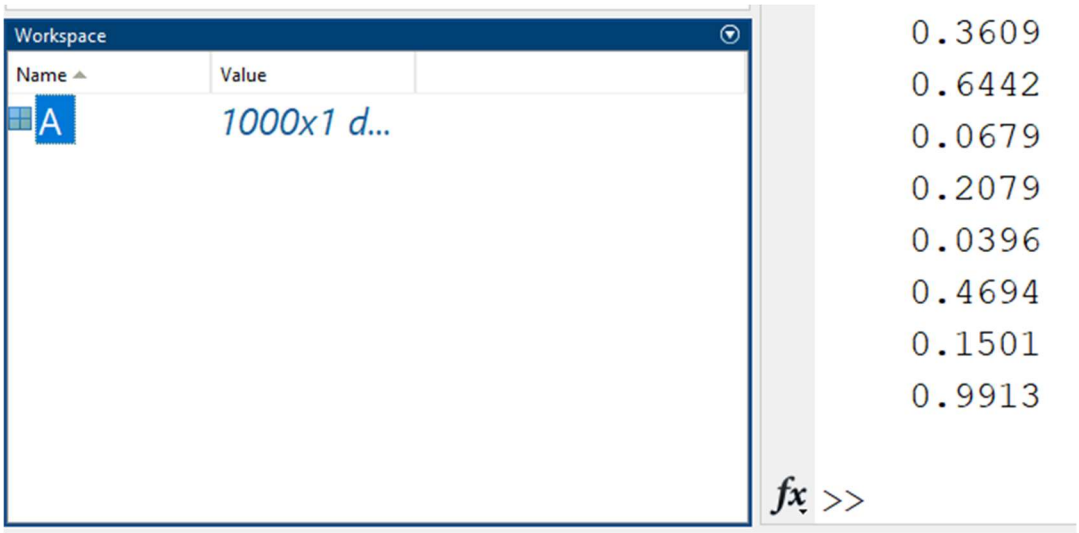


Im Menü-Reiter PLOTS können Daten direkt visualisiert werden. Dazu muss im Bereich 4 der Programmoberfläche (siehe erste Abbildung) eine Variable ausgewählt werden.

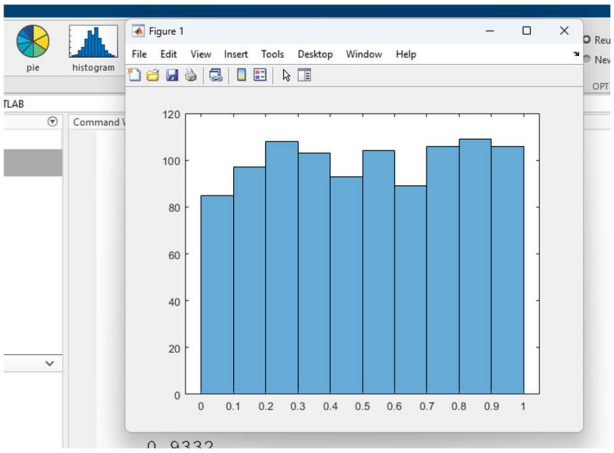
Erzeugen wir zunächst eine große Folge von Zufallszahlen durch die folgende Anweisung:

```
A=rand(1000,1)
```

Wählen dann A aus



Und erzeugen durch Auswahl des Diagrammtyps Histogram die folgende Grafik der Häufigkeitsverteilung:

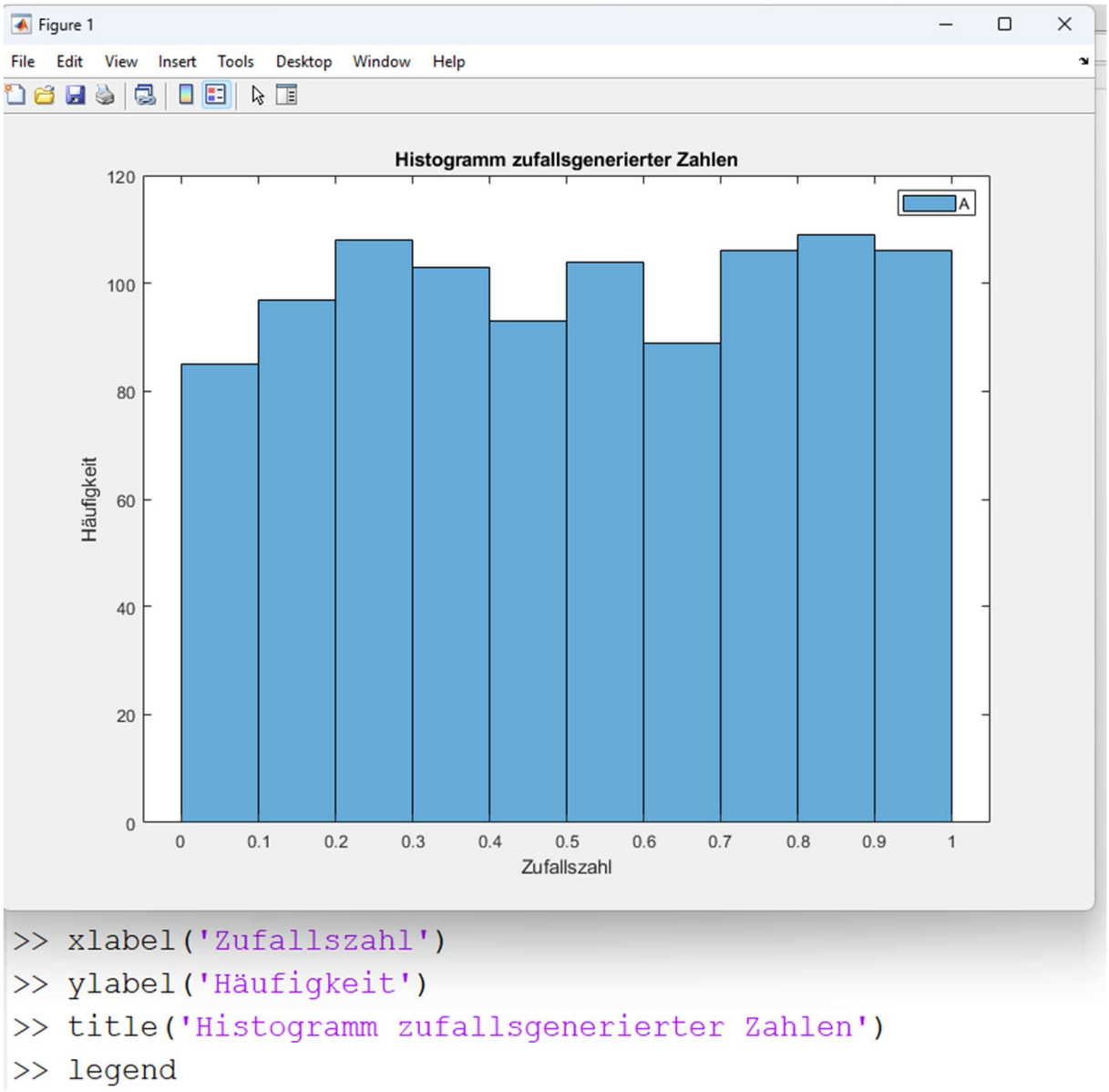


Im Command Window sehen wir die Befehlszeile, die unsere Auswahl ausgelöst hat:

histogram(A)

Wir können diese Zeile auch direkt eingeben oder in Skripten und Funktionen verwenden.

Mit den folgenden Befehlen kann die Grafik weiter gestaltet werden:



Schriftart und -größe sind auch nachträglich anpassbar, so dass eine große Flexibilität bei der Gestaltung besteht.

Kontrollstrukturen in Skripten und Funktionen

Mit Funktionen und Skripten haben wir Hilfsmittel kennengelernt, um Anweisungen zusammenzufassen und wiederholt ausführen zu können. Im Fall von Funktionen sorgen Aufrufe mit unterschiedlichen Argumenten für Flexibilität, ein und dieselbe Funktion für unterschiedliche Aufgabenstellungen zu nutzen.

Oft ist es jedoch sinnvoll, während des Ablaufs einer Funktion oder eines Skripts unterschiedliche Operationen auszuführen oder eine Gruppe von Anweisungen mehrmals zu wiederholen.

Dafür sehen Programmiersprachen (und auch MATLAB) Kontrollstrukturen für den Programmablauf, wie Schleifen und die IF-Anweisung, vor.

Als Beispiel verwenden wir die Visualierung von Daten in einer Streudiagrammmatrix mit einer eigenen Funktion.

Mit dem Befehl **edit streudiagrammmatrix** legen wir eine neue Datei an und öffnen den Editor.

Wir beginnen die Funktion mit dem Schlüsselwort function, gefolgt vom Namen der Funktion und den Argumenten:

function streudiagrammmatrix(D)

Wird diese Funktion später aufgerufen, muss ihr eine Variable übergeben werden, auf die innerhalb der Funktion mit dem Variablennamen D zugegriffen werden kann.

D wird später eine Matrix sein, deren Spalten einzelnen Variablen entspricht (z.B. Temperatur, Niederschlag) und deren Zeilen die Beobachtungen (Messung zu einer bestimmten Uhrzeit pro Zeile) darstellen.

Die for-Schleife erlaubt es, einen Block von Anweisungen mehrfach auszuführen. Wie oft er ausgeführt wird, bestimmen die Argumente für die for-Schleife.

```
for i=1:10



end
```

führt alle Anweisungen zwischen den Zeilen for i=1:10 und end genau zehnmal aus. Dabei wird bei jedem Durchlauf des Blocks die Variable i um den Wert 1 erhöht.

```
X=0
for i=1:10
   X=X+2;
end
```

addiert zum Wert X in jedem Durchlauf den Wert 2. Daher hat X nach dem Ende der for-Schleife den Wert 20.

In unserem Beispiel nutzen wir die for-Schleife, um verschiedene Streudiagramme zu erzeugen.

for i=1:size(D,2) wiederholt den folgenden Anweisungsblock für die Anzahl der Spalten von D. Diese Anzahl wird mit size(D,2) ermittelt. Der Aufruf size(D,1) würde die Zahl der Zeilen von D zurückliefern.

Als Besonderheit verwenden wir eine zweite for-Schleife

```
for i=1:size(D,2)
    for j=1:size(D,2)



    end
end
```

Für jeden Durchlauf der äußeren Schleife wird die innere mehrfach durchlaufen, abhängig von der Zahl der Spalten von D.

Hat D zwei Spalten wird der innere Block mit Anweisung 2x2=4-mal durchlaufen.

In diesem inneren Block legen wir eine Diagrammmatrix an:

subplot(size(D,2), size(D,2),1)

Damit stehen z.B. 2x2 Diagrammfelder im Ausgabefenster zur Verfügung. Der letzte Parameter legt fest, welches Diagramm gerade aktiv ist.

Mit der Formel

index=(i-1)*size(D,2)+j wandert dieses aktive Diagramm immer um ein Feld weiter.

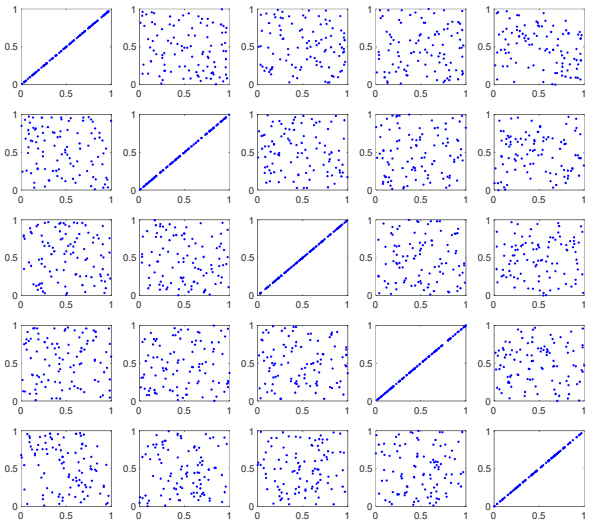
```
function streudiagrammmatrix(D)

for i=1:size(D,2)
    for j=1:size(D,2)
        index=(i-1)*size(D,2)+j
     subplot(size(D,2),size(D,2),index)
     plot(D(:,i), D(:,j), 'b.');
    end
end
```

Rufen wir diese Funktion nun mit einer Matrix aus Zufallszahlen auf:

```
D=rand(100,5);

streudiagrammmatrix(D);
```



Anhand der Ausgabe erkennen wir, dass in jedem Durchlauf der inneren Schleife ein neues Diagramm hinzugefügt und gezeichnet wird. Wenn die Zählvariable i den gleichen Wert wie j hat, also i=j=1 oder

i=j=2 werden die Werte in einer Spalte der Matrix D gegeneinander abgetragen, so dass sich eine Gerade ergibt. Die anderen Felder zeigen den zufälligen Charakter der Daten, wenn zwei Spalten der Matrix D gegeneinander abgetragen werden.

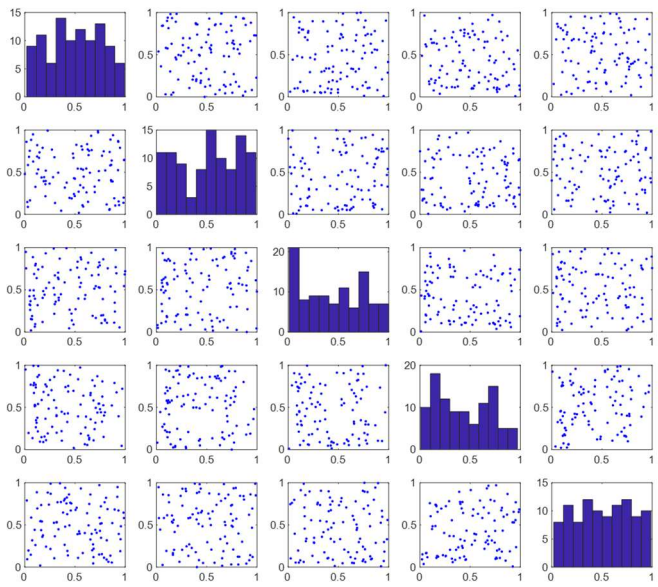
Wir nutzen nun die IF-Funktion, um die Hauptdiagonale anders zu gestalten. Dort soll künftig ein Histogramm der Daten in einer Spalte der Matrix D angezeigt werden.

```
function streudiagrammmatrix(D)


for i=1:size(D,2)
    for j=1:size(D,2)
        index=(i-1)*size(D,2)+j
     subplot(size(D,2),size(D,2),index)
     if i==j
            hist(D(:,i));
        else
            plot(D(:,i), D(:,j), 'b.');
        end
    end
end
```

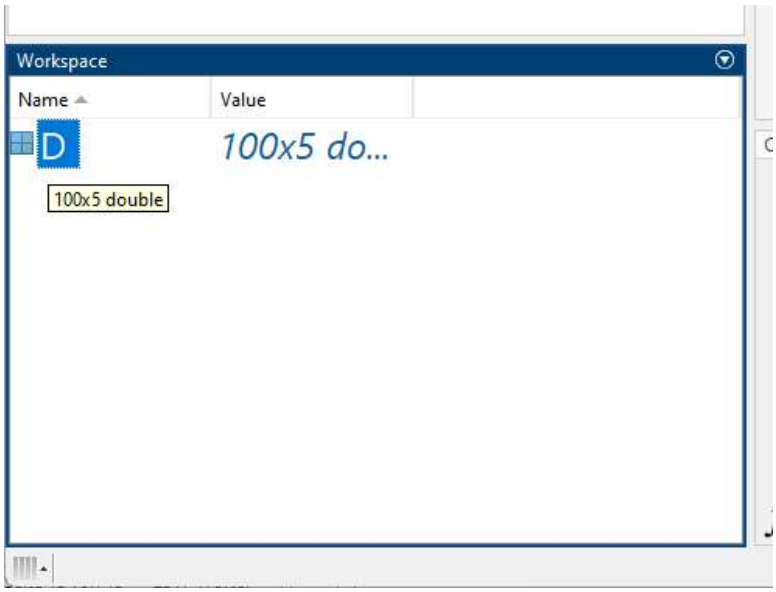
Dazu fügen wir hinter dem Schlüsselwort **if** eine Bedingung hinzu. Im Beispiel den Vergleich zweier Variablen auf Gleichheit der Wert i==j

Ist die Bedingung erfüllt, gilt also i==j, dann wird die folgende Anweisung ausgeführt und ein Histogramm gezeichnet. Sind die Werte ungleich, werden also zwei unterschiedliche Spalten von D gegeneinander abgetragen, so wird wie bisher die Funktion plot verwendet.

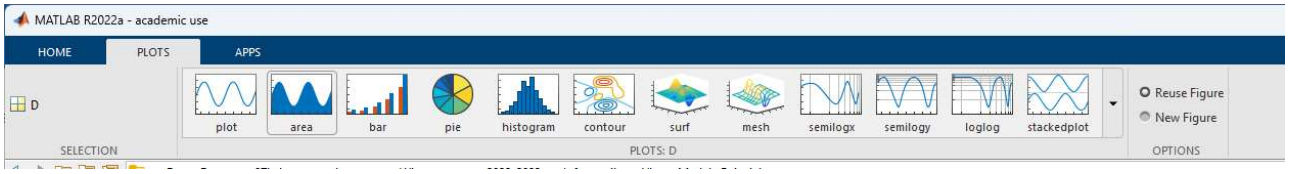


Im Beispiel haben wir gesehen, wie vorhandene MATLAB-Funktionen (plot, hist) in einer eigenen neuen Funktion miteinander kombiniert werden können. Eine Stärke von MATLAB ist es, dass es bereits viele fortgeschrittene Auswerte- und Visualisierungsfunktionen enthält.

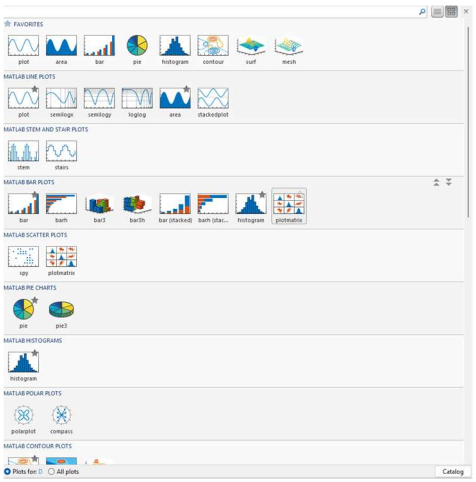
Wir können bspw. die Variable D auch in der MATLAB-Oberfläche mit der Maus auswählen.



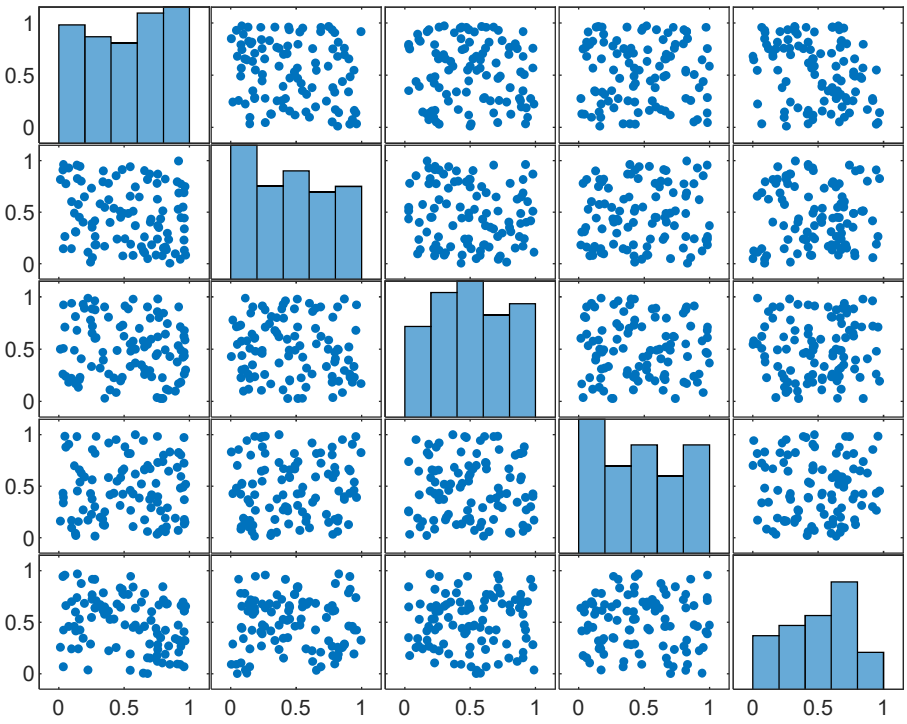
Dann können wir im Menü (im Reiter PLOTS) aus einer Reihe von Diagrammarten wählen.



Es werden eine Reihe für die Matrix D geeigneter Diagramme angezeigt.



Wir können den Diagrammtyp plotmatrix auswählen und erhalten ebenfalls die Streudiagrammatrix.



Gleichzeitig zeigt MATLAB im Command Window auch die verwendete Funktion und wie sie aufgerufen wurde an:

```
>> plotmatrix(D)
>>
```

Wie wir schon bei der Histogramm-Funktion gesehen haben, ist das sehr hilfreich, denn wir erfahren auf diese Weise, wie wir das Diagramm zukünftig in eigene Programme einbauen können und wie der Aufruf ohne den Umweg über das Menü funktioniert.

Nutzung der Dokumentation und der Matlab-Hilfe

Im vorangegangenen Beispiel wurden Matlab-Hilfe und die Matlab-Dokumentation bereits eingeführt.

Die Eingabe von doc ruft die Startseite der MATLAB-Dokumentation auf.

doc <Funktionsname> ruft die Dokumentation einer MATLAB-Funktion auf, z.B. doc plot

help und help <Funktionsname> rufen die textbasierte MATLAB-Hilfe auf.

Fazit

In dieser Einführung wurde MATLAB vorgestellt. Es wurde ein Überblick über grundlegende **Funktionen, wie sie sich in vielen Programmiersprachen finden** gegeben. Wir haben gemeinsam eine erste Funktion erstellt, die eine Streudiagrammmatrix anzeigt. Diese Funktion kann nun immer wieder genutzt oder in komplexeren MATLAB-Programmen verwendet werden.