A Anhang

A.1 Eigenständige Applikationen (Apps)

A.1.1 Erzeugung einer MATLAB App

Im Folgenden wird beschrieben, wie eine eigenständige Applikation erzeugt wird. Da hierfür das Add on MATLAB Compiler erforderlich ist, muss dieses Produkt in der MATLAB Version, die benutzt wird, lizensiert sein. Die Erzeugung von Programmen, die außerhalb MATLAB eingesetzt werden, wird „Deployment“ genannt. Der MATLAB Compiler kann von „Command Window“ mit der Anweisung

```
deploytool
```

gestartet werden. Um eine eigenständige Applikation (Standalone application) zu erzeugen, ist anschließend „Application Compiler“ auszuwählen (Bild A.1). Alternativ ist ein Aufruf in der Symbolleiste im Register „APPS“ über das Icon „Application Compiler“ möglich (Bild A.2).

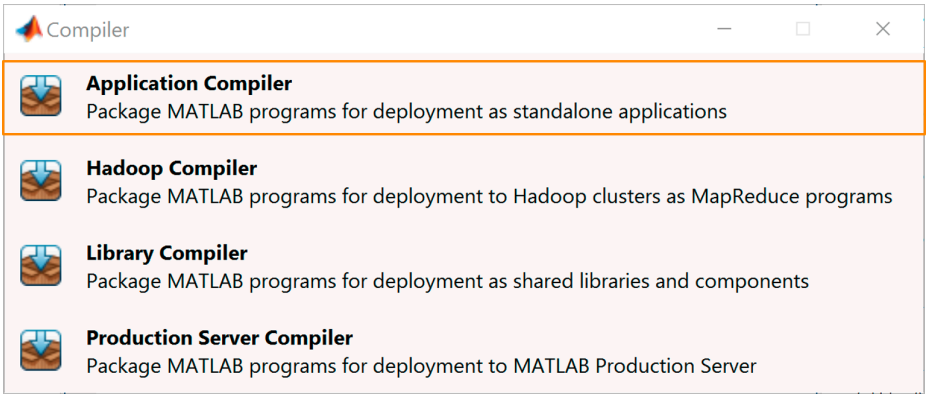


Bild A.1 Auswahl für eigenständige Applikation (Standalone application)

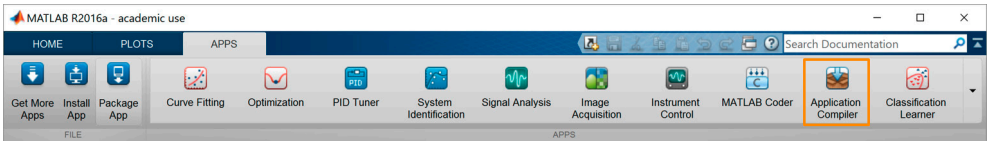


Bild A.2 Auswahl für eigenständige Applikation (Standalone application)

Die Fenster für die Einstelllungen des Compiles zeigt Bild A.3. Die Funktion der markierten Bereiche ist die folgt:

❶ Der Filename für den eine App erstellt werden soll, ist im Browser auszuwählen.

❷ Eingabe des Namens für die App. Die Einstellungen werden auch unter diesem Namen gespeichert. Die Extension des Filenamens ist ☐.prj.

❸ Eingabebereiche für Informationen zur App

❹ Auswahl von Files, die für die App benötigt werden und nicht automatisch erkennbar sind (z. B. Datenfiles, ☐.mat). Automatisch erkennbare Scripts oder Functions können hier auch eingefügt werden. Dies ist jedoch nicht zwingend erforderlich.

❺ Files, die mit der App installiert werden sollen.

❻ Falls bei der Ausführung der App ein Fehler auftritt, kann dieser optional in einem logfile (Logbuch) dokumentiert werden. Falls dies gewünscht wird, ist dies auszuwählen.

❼ Es gibt zwei Möglichkeiten, die MATLAB Compiler Runtime dem Anwender zur Verfügung zu stellen:

- Sie wird bei der Erstellung einer eigenständigen Applikation mit der Anwendung in einem Paket zusammengefasst.
- Sie wird vom Anwender bei der Installation der eigenständigen Applikationen von der MathWorks Homepage heruntergeladen. Hierfür gibt es einen Web-Installer.

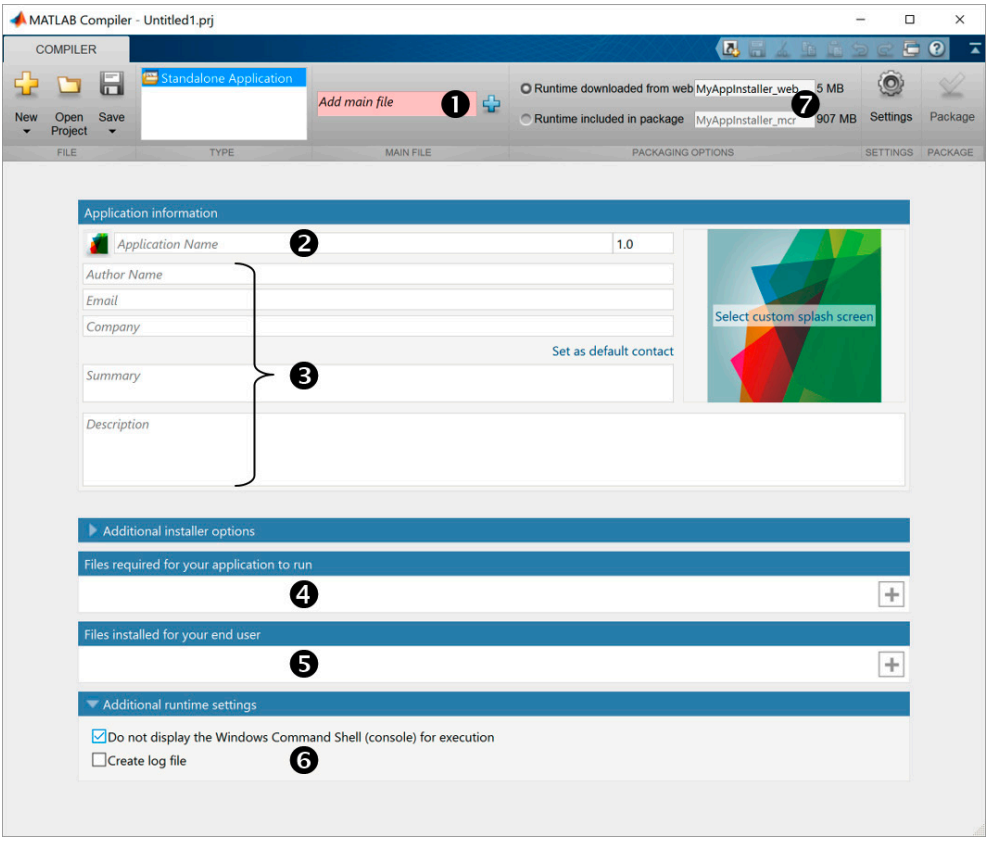


Bild A.3 Compiler Einstellungen

Die Einstellungen des Compilers zur Erzeugung einer App zeigt Bild A.4 beispielhaft für die Übung 4. Der Name der erzeugten App ist Uebung4. Die Compiler Einstellungen werden in Uebung4.prj im aktuellen Folder gespeichert. Anstatt der Standard-Icons wurde an allen Stellen das Logo der Hochschule Rosenheim eingesetzt. Für die MATLAB Compiler Runtime wurde die Installation aus dem Web ausgewählt. Die Packungsgröße ist dadurch deutlich kleiner, als würde die Compiler Runtime mitverpackt (5 MB im Vergleich zu 907 MB in diesem Beispiel ohne Berücksichtigung der Daten der Applikation). Nur bei erstmaliger Installation der App auf dem Zielrechner ist der unter Umständen zeitaufwändige Web-Installationsaufwand der MATLAB Compiler Runtime erforderlich. Bei weiteren Apps hat man immer den Vorteil der kleinen Packungsgröße. Die Erzeugung der eigenständigen Applikation wird mit dem Icon „Package“ gestartet. Der Fortschritt wird anschließend angezeigt (Bild A.5).

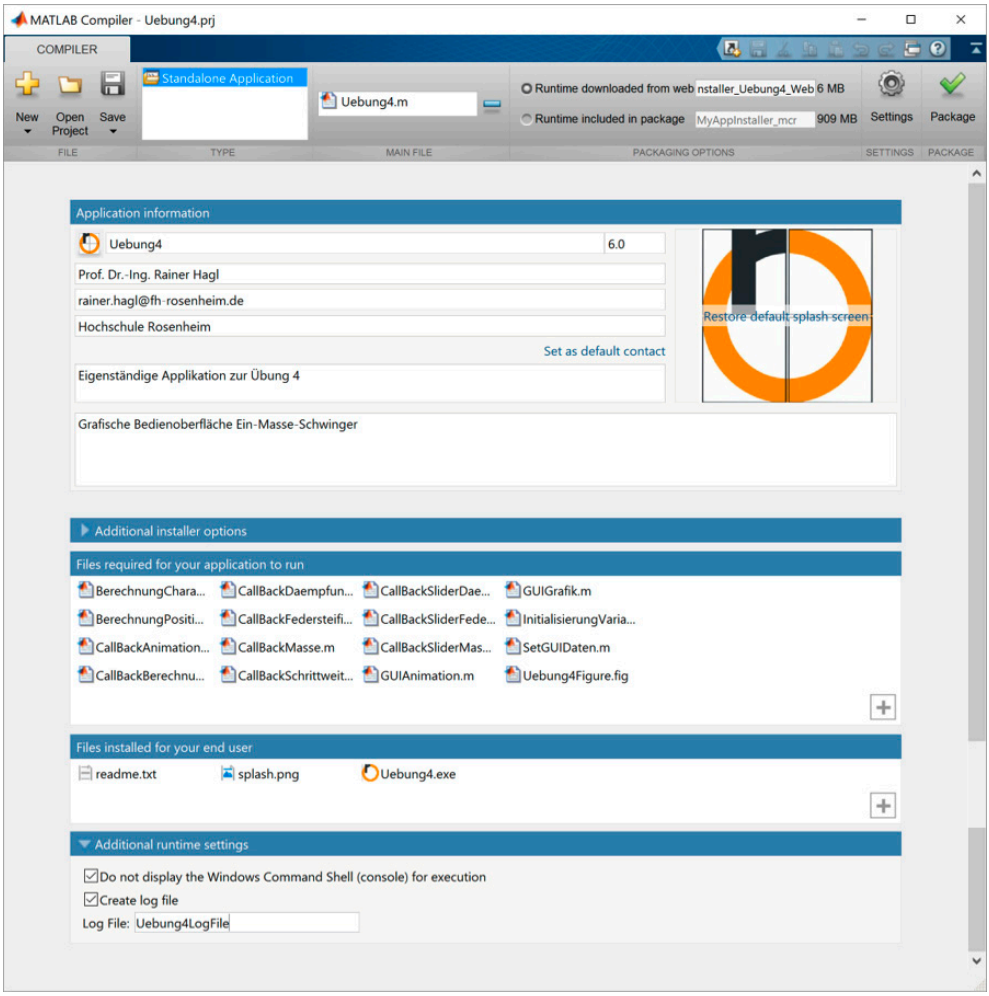


Bild A.4 Compiler Einstellungen Übung 4

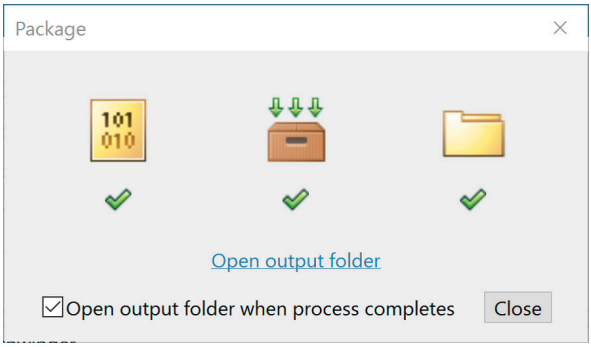


Bild A.5 Fortschrittsanzeige des „Package“ (Nach erfolgreichem Abschluss)

A.1.2 Hinweise zur Installation einer MATLAB App

Die folgenden Installationshinweise werden beispielhaft für eine eigenständige Applikation gezeigt. Bei der Applikation handelt es sich um die grafische Bedienoberfläche, die in der Übung 4 erstellt wird.

0 Stellen Sie sicher, dass Ihr Computer während der Installation eine Verbindung zum Internet hat. Bei der Installation wird automatisch eine Verbindung zu MathWorks hergestellt, die für die Softwareinstallation erforderlich ist.

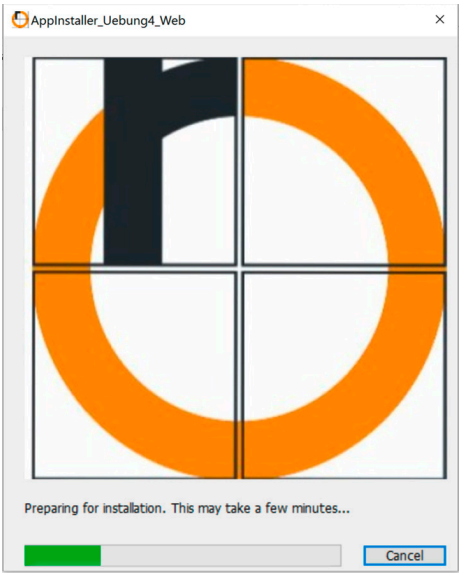
1 Starten Sie den Installer für die App, für das Beispiel

AppInstaller_Uebung4_Web.exe

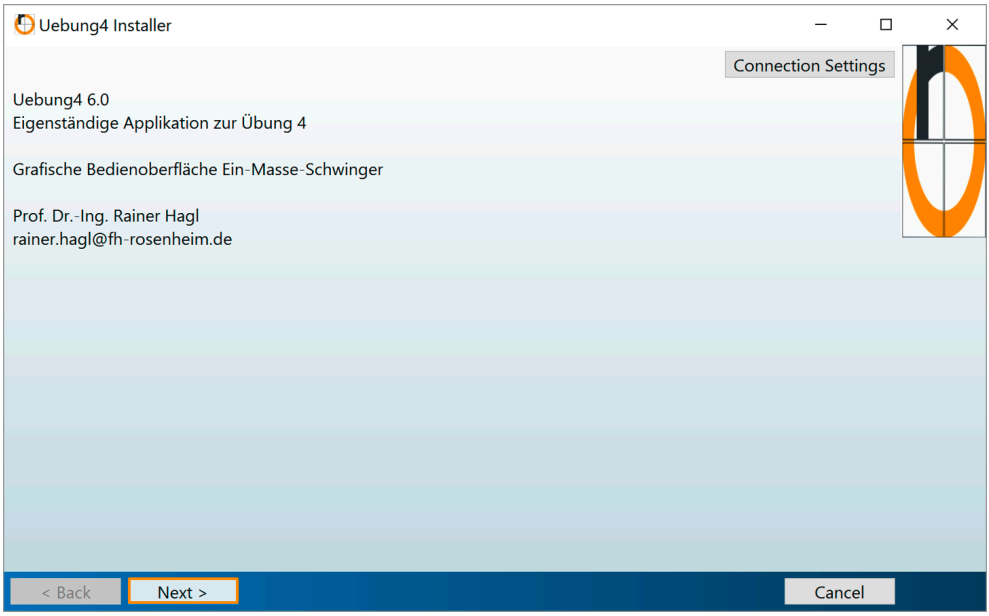
Der Installer ist bei den Programmbeispielen zum Lehrbuch im folgenden Unterverzeichnis abgelegt: Eigenständige Applikationen/Uebung4

2 Unter Umständen blockiert die Benutzerkontosteuerung zunächst den Download. Sie müssen Änderungen an Ihrem Computer zulassen, damit die Installation fortgeführt wird.

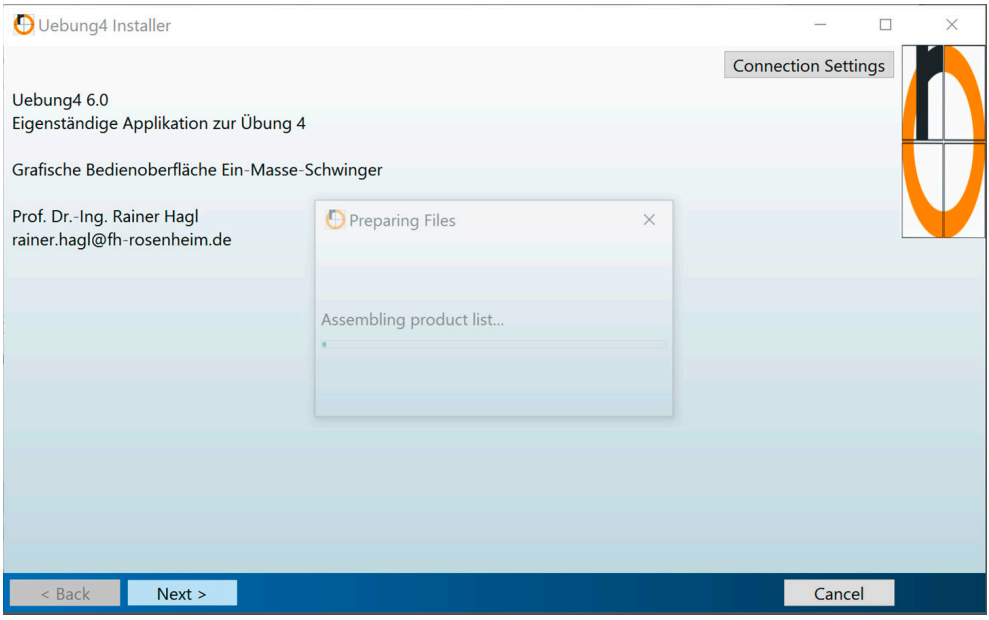
3 Der Download startet dann automatisch und es erscheint zunächst folgende Darstellung:



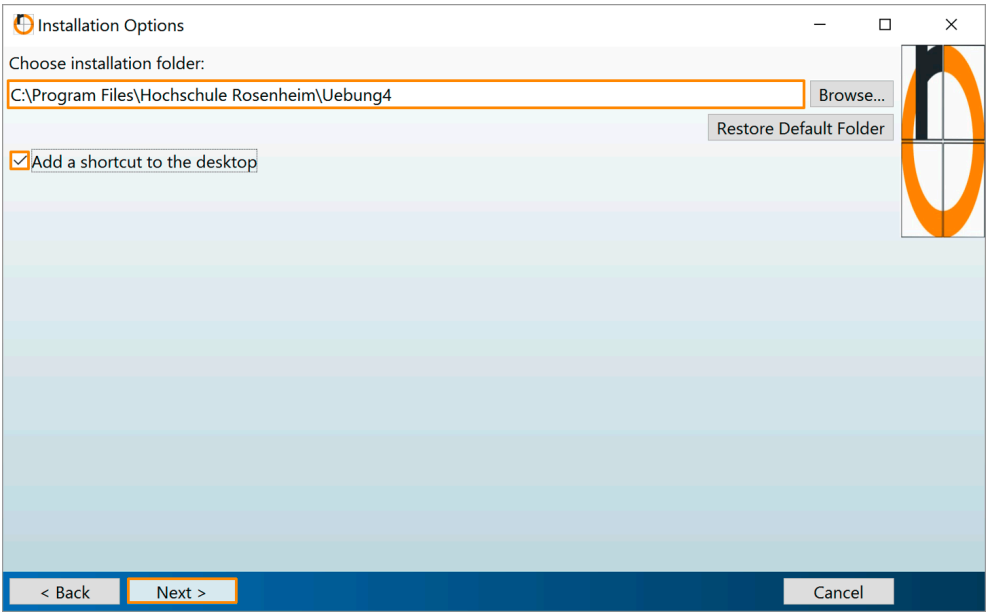
❹ Nach einiger Zeit erscheint folgendes Fenster:



❺ Nach dem Drücken des Push Buttons „Next>“ erfolgt automatisch eine Kontaktaufnahme mit MathWorks und die Files werden vorbereitet.

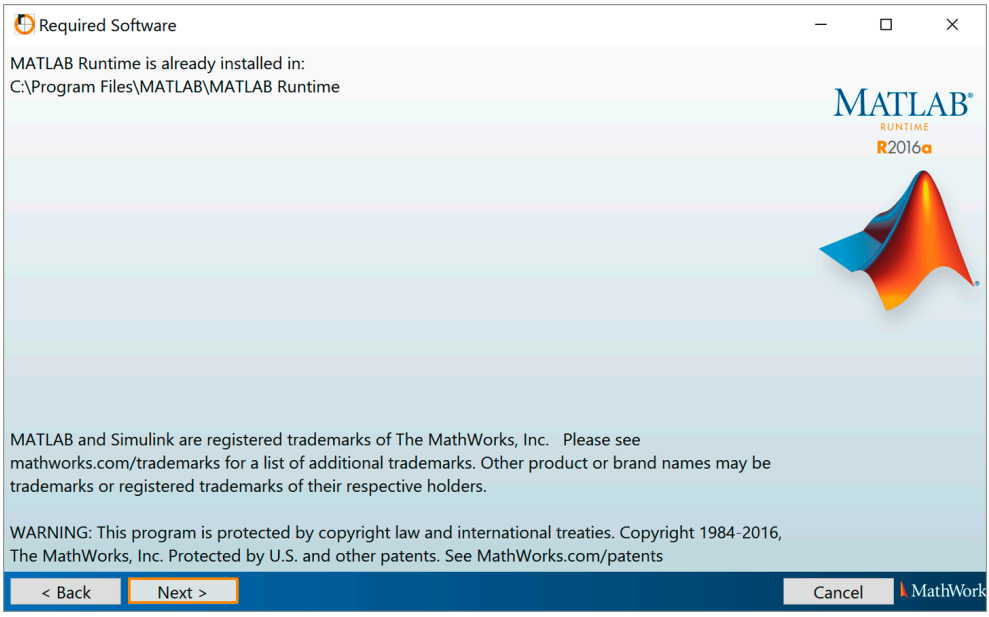


❻ Legen Sie den Ordner fest, in dem die App installiert werden soll oder verwenden Sie die Standardeinstellung. Selektieren Sie „Add a shortcut to the desktop“ (siehe unten). Wenn das Häkchen nicht gesetzt ist, erscheint im Desktop nach der Installation kein Icon für die App.



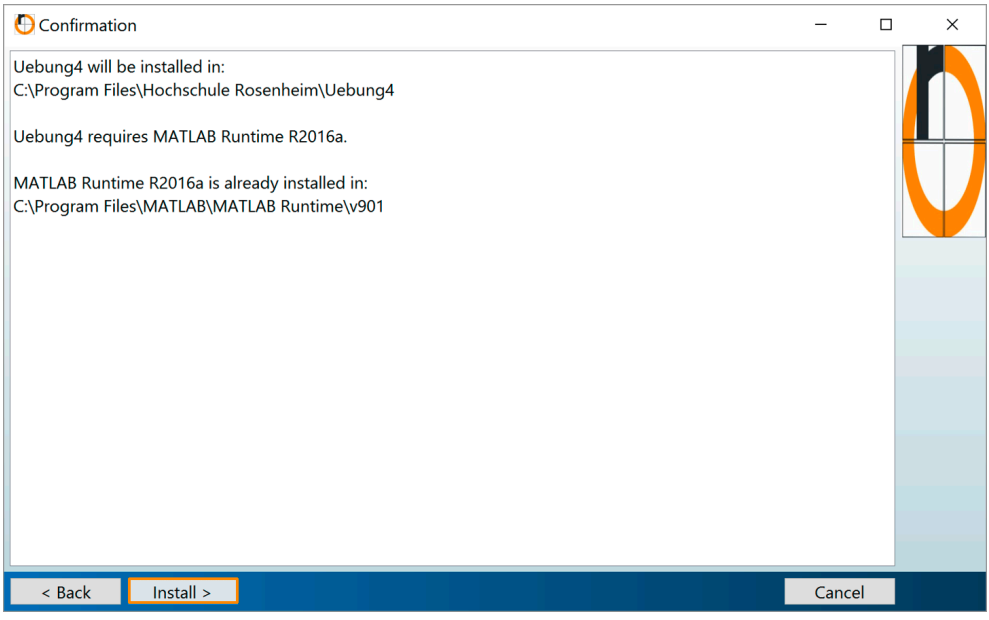
Drücken des Push Buttons „Next>“.

❼ Anschließend erscheint ein Fenster für „Required Software“, das bei der Erstinstallation einer MATLAB App von der unten dargestellten abweicht.



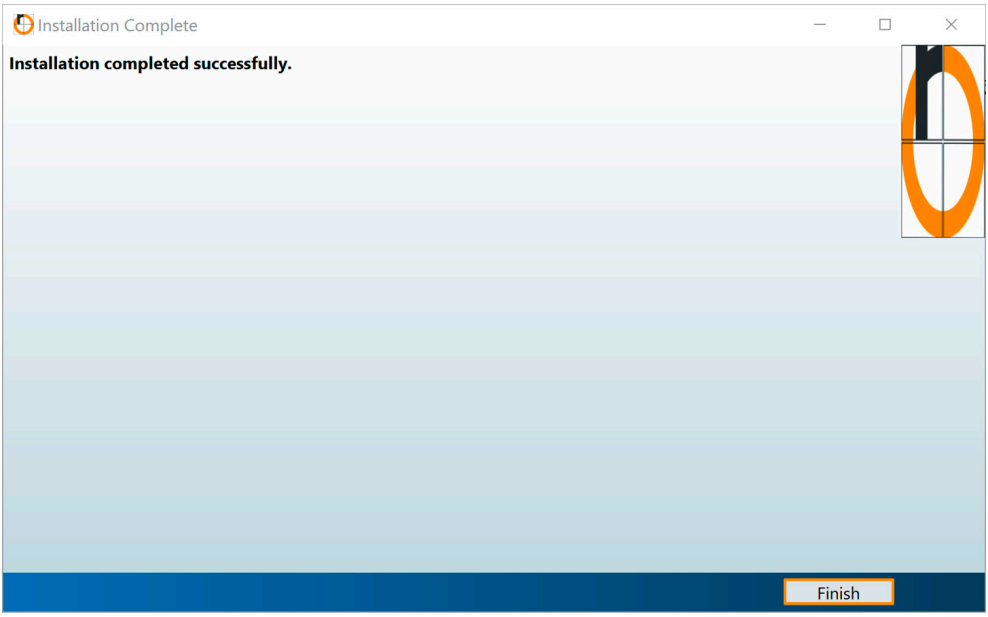
Drücken des Push Buttons „Next>“.

❽ Bestätigen Sie, dass Sie die MATLAB App installieren wollen durch Drücken des Push Buttons „Install>“.

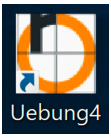


Die Softwareinstallation auf Ihrem Rechner startet. Bei der Erstinstallation einer MATLAB App auf dem Rechner muss einmalig die MATLAB Compiler Runtime (MCR) aus dem Internet von MathWorks geladen werden. Abhängig von der zur Verfügung stehenden Internet-Bandbreite kann dies einige Zeit dauern. Bitte haben Sie Geduld.

❾ Nach Abschluss des Software Downloads erscheint das unten gezeigte Fenster. Drücken Sie den Push Buttons „Finish>“, um die Installation zu vervollständigen.



❿ Auf Ihrem Desktop erscheint folgendes zusätzliche Icon. Damit können Sie die App starten.



Automatisch erscheint im Desktop beim Start der App auch ein Icon mit dem Namen Uebung4LogFile. Dieses Icon ist die Verknüpfung zum Logbuch der App. Der Textfile kann mit einem Editor gelesen werden.

Das Aussehen der eigenständigen Applikation bei der Ausführung zeigt Bild A.6.

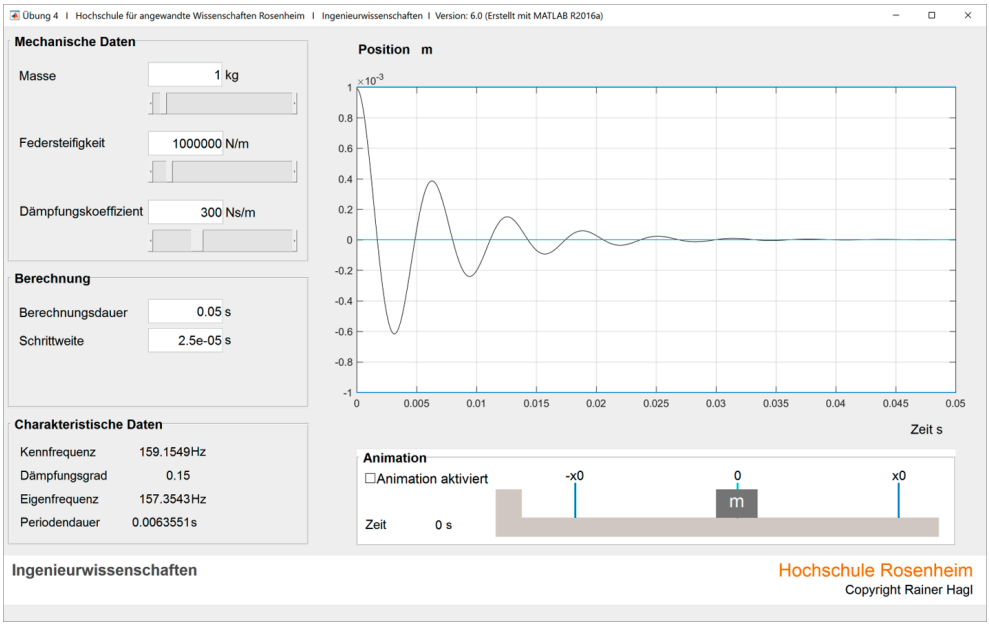


Bild A.6 Aussehen der eigenständigen Applikation zur Übung 4

A.2 Übungen

Übung 1 – Erste Schritte

Es soll die kinetische Energie einer bewegten Masse bestimmt werden. Bekanntermaßen gilt:

$$E = \frac{1}{2} m v^2 \tag{Ü.1}$$

E	Kinetische Energie	*Kinetic energy*	J
m	Bewegte Masse	*Moved mass*	kg
v	Geschwindigkeit	*Velocity*	m/s

Bestimmen Sie die kinetische Energie für folgende Fälle:

a) Die bewegte Masse ist 300 kg, und die Geschwindigkeit beträgt 60 m/min. Nutzen Sie dabei das „Command Window“ nur als Taschenrechner.

b) Definieren Sie die Masse und die Geschwindigkeit als Variablen (Variablenname: m und v) und ordnen diesen die Werte aus Teilaufgabe a) zu. Führen Sie die Berechnung mit den Variablen durch, und speichern das Ergebnis in einer Variablen für die kinetische Energie (Variablenname: E). Nutzen Sie zur Ergebnisprüfung den „Workspace Browser“.

c) Die kinetische Energie soll für Geschwindigkeiten von 0 m/min bis 60 m/min in Geschwindigkeitsschritten von 10 m/min berechnet werden. Nutzen Sie dabei den Vorteil von MATLAB, dass die Vektorverarbeitung stark vereinfacht wird. Definieren Sie einen Vektor („Zahlenreihe“) für die Geschwindigkeit mit folgenden Werten:

   ```
   [0 10 20 30 40 50 60]
   ```

   und weisen diesen der Variablen für die Geschwindigkeit (Variablenname: v) zu. Welche Veränderungen ergeben sich dadurch im Workspace und im „Variable Editor“?

   Berechnen Sie anschließend die kinetische Energie für die sieben unterschiedlichen Geschwindigkeiten unter Nutzung der Vektorverarbeitung von MATLAB. Für eine elementweise Multiplikation oder elementweises Quadrieren der Geschwindigkeit gibt es in MATLAB folgende Operatoren:

   - Elementweises Multiplizieren v.*v
   - Elementweises Quadrieren v.^2

   Überprüfen Sie das Ergebnis der Berechnung für die kinetische Energie im „Variable Editor“.

d) Stellen Sie die berechnete kinetische Energie der bewegten Masse über der Geschwindigkeit grafisch, wie in Bild Ü.1 gezeigt, dar. Hierfür benötigen Sie folgende MATLAB Functions.

   - Für den Kurvenverlauf

     ```
     plot
     ```

- Für die Achsbeschriftungen

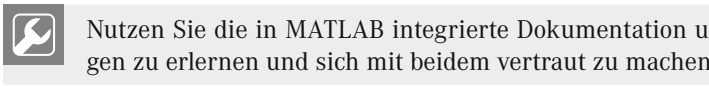
  `xlabel` und `ylabel`

- Für die Überschrift

  `title`

- Für die Gitternetzlinien

  `grid`



Nutzen Sie die in MATLAB integrierte Dokumentation und Hilfe, um Anweisungen zu erlernen und sich mit beidem vertraut zu machen.

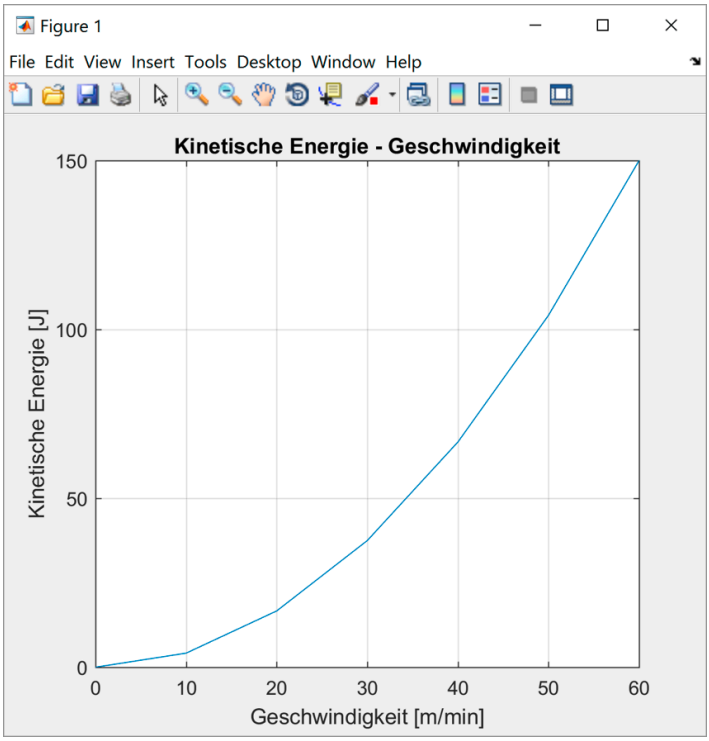


Bild Ü.1 Grafische Darstellung der kinetischen Energie

Übung 2 – Einführung

Für das im Bild Ü.2 dargestellte System sollen der

- Positions-Zeit Verlauf
- Geschwindigkeits-Zeit Verlauf
- Beschleunigungskraft-Zeit Verlauf
- Energie-Zeit Verlauf (kinetische Energie der bewegten Masse)

berechnet und grafisch dargestellt werden. Die Masse des Systems ist $m = 300$ kg.

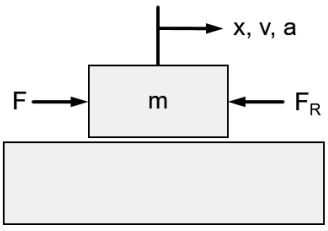


Bild Ü.2 Linearbewegung

Die Reibkraft (F_R) ist konstant 50 N. An der Masse greift eine abschnittsweise konstante Kraft an:

$$F(t) = \begin{cases} 200\,N, & 0 \le t < 0{,}5 \text{ s } (\textit{Zeitabschnitt 1}) \\ 75\,N, & 0{,}5 \text{ s} \le t \le 1{,}5 \text{ s } (\textit{Zeitabschnitt 2}) \end{cases} \tag{Ü.2}$$

Die Berechnung und die grafische Darstellung soll für den Zeitraum 0 s $\le t \le$ 1,5 s erfolgen. Die Werte sind alle 0,05 s zu berechnen. Nutzen Sie bei der Programmierung die MATLAB Operatoren zur Vektorverarbeitung. Das Programm ist so zu erstellen, dass es auch für andere Zeiten und Kräfte einsetzbar ist.

Alle vier Zeitverläufe sind in einem Grafikfenster, wie in Bild Ü.3 gezeigt, darzustellen. Hierfür benötigen Sie die MATLAB Function „subplot“.

Nutzen Sie die MATLAB Dokumentation und Hilfe für die Beschreibung der Anweisung.

Alle vier Grafiken sind mit

- Achsbeschriftungen, inklusive Einheiten
- Überschriften
- Gitternetzlinien

zu versehen.

Erstellen Sie Ihr Programm als MATLAB Script im Editor. Gehen Sie dabei bitte schrittweise vor (Tabelle Ü.1).

- Programm für Zeitabschnitt 1 ($0 \leq t \leq 0{,}5$ s), inklusive der für diesen Zeitbereich dazugehörigen Grafik.
- Ergänzen Sie das Programm für Zeitabschnitt 2 ($0{,}5 \leq t \leq 1{,}5$ s).

Tabelle Ü.1 Schrittweises Vorgehen zur Erstellung der Übung

❶	Programm für Zeitabschnitt 1 ($0 \leq t \leq 0{,}5$ s), inklusive der für diesen Zeitbereich dazugehörigen Grafik.
❷	Ergänzen Sie das Programm für Zeitabschnitt 2 ($0{,}5 \leq t \leq 1{,}5$ s).

Nutzen Sie zur Fehlersuche den Debugger.

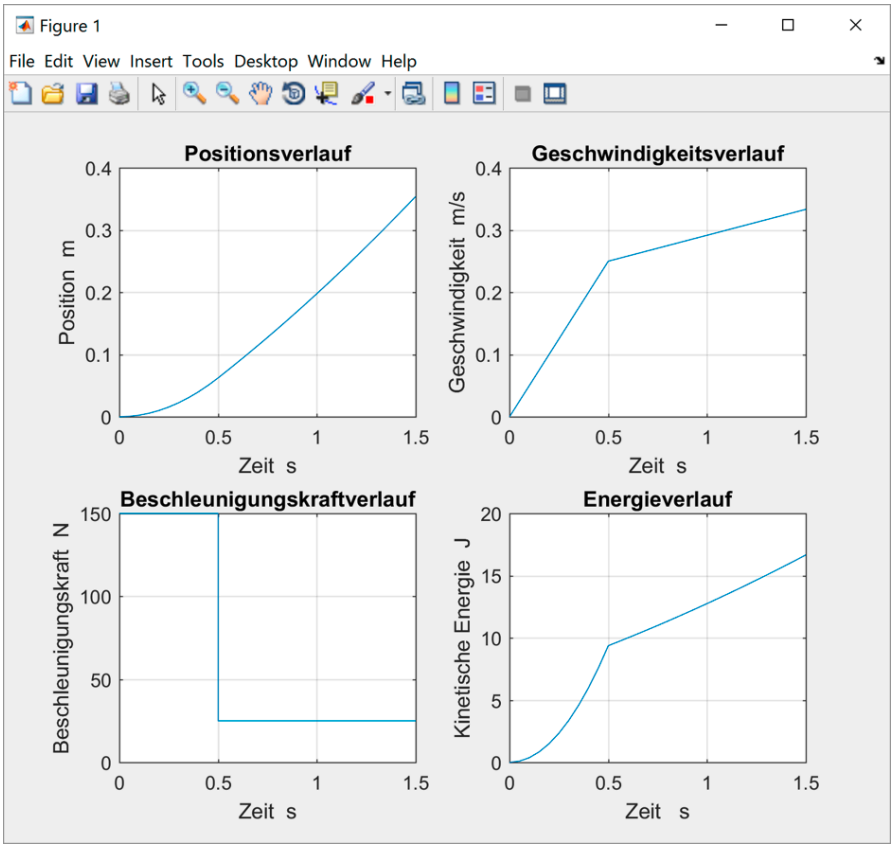


Bild Ü.3 Grafische Darstellung

Welchen Wert hat die Energie nach 0,25 s?

Nutzen Sie zur Werteermittlung den „Workspace Browser“.

Testen Sie Ihr Programm, ob es auch bei anderen Zeiten und Kräften, ohne Änderung des Berechnungs- und Grafikteils, richtige Ergebnisse liefert, z. B. für:

$$F(t) = \begin{cases} 200\,N,\ 0 \le t < 1\ \text{s} & (\textit{Zeitabschnitt}\,1) \\ -50\,N,\ 1 \le t \le 2\ \text{s} & (\textit{Zeitabschnitt}\,2) \end{cases} \tag{Ü.3}$$

Das Ergebnis für diesen beispielhaften Testfall zeigt Bild Ü.4.

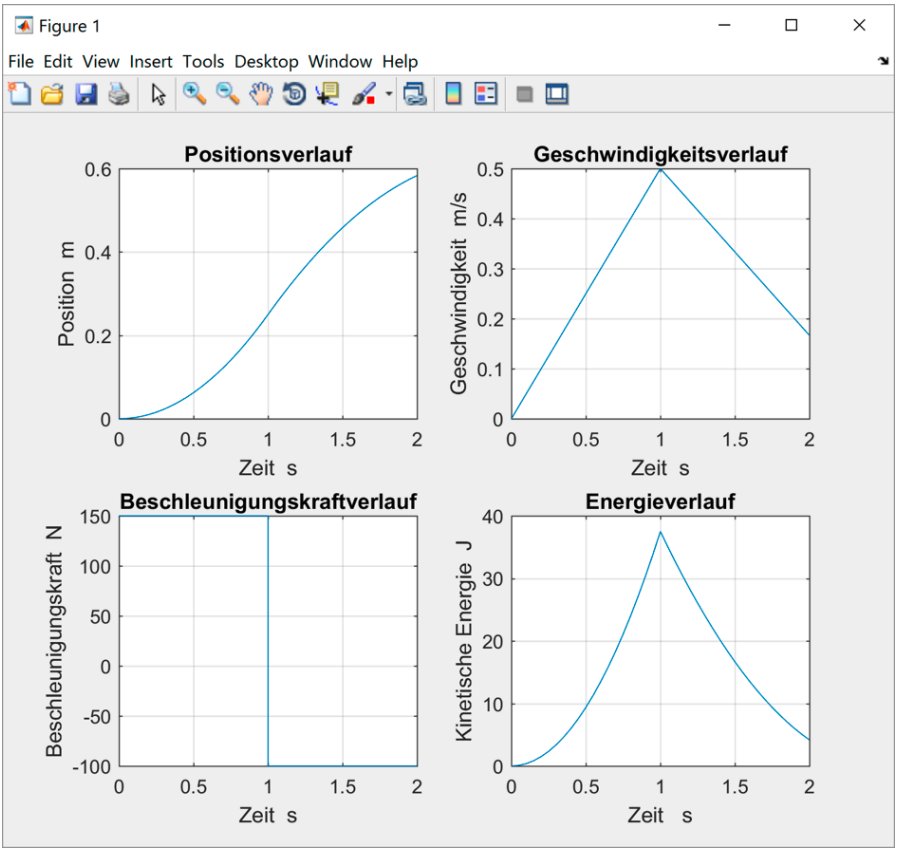


Bild Ü.4 Beispielhafter Testfall

Übung 3 – Grundlagen der Programmierung

Für diese Übung benötigen Sie einige Files die vorgefertigt sind. Diese Files sind bei den Programmbeispielen zum Lehrbuch im folgenden Unterverzeichnis abgelegt: Uebungen/Uebung3

3.1 Debugger

Der Drehmomentverlauf an einem Pendel soll in einem Winkelbereich von 0 Grad (Pendel in vertikaler Ausrichtung) bis 90 Grad (Pendel in horizontaler Ausrichtung) berechnet und visualisiert werden. Die bekannte Drehmomentgleichung lautet:

$$M(\varphi) = m\,g\,r\,\sin(\varphi) \tag{Ü.4}$$

M	Drehmoment	Nm
m	Masse	kg
g	Erdbeschleunigung	m/s^2
r	Abstand des Schwerpunktes zum Drehpunkt	m
m	Drehwinkel	rad

Das folgende Script für eine Masse von 10 kg und einem Abstand des Schwerpunkts zum Drehpunkt von 0,1 m hat einen offensichtlichen Fehler, wie an der erzeugten Grafik (Bild Ü.5) zu erkennen ist. Sie erhalten keine Fehlermeldung, da das Programm von der Syntax korrekt ist. Verwenden Sie den Debugger, um den Fehler zu finden (selbst wenn Sie den Fehler bei der Programmdurchsicht gleich erkennen).

a) Gehen Sie im Debugmodus Zeile für Zeile des Programmes durch und überprüfen Sie in jeder Zeile im „Workspace Browser“, ob das Programm bis zu dieser Zeile korrekt funktioniert.

b) Sobald Sie den Fehler lokalisiert haben, setzen Sie einen Breakpoint in die Zeile direkt nach der fehlerhaften Zeile und starten das Script im Debugmodus nochmals. Nachdem der Debugger in der Zeile pausiert, korrigieren Sie manuell die fehlerhafte Anweisung im „Command Window“. Danach lassen Sie die Programmausführung weiterlaufen und überprüfen, ob das Ergebnis nun korrekt ist.

c) Korrigieren Sie die Fehler im Script, so dass das Programm fehlerfrei abläuft.

Script Ü.1 Fehlerhaftes Script für Drehmomentverlauf (Uebung3_1.m)

```
%% Initialisierung
% Workspace löschen
clear
% Masse
m = 10;
% Erdbeschleunigung
g = 9.81;
% Radius
r = 0.1;
% Winkelpunkte
phi = 0:90;
%% Berechnung des Drehmomentes
x = 2*pi*phi;
M = m*g*r*sin(x);
%% Visualisierung
plot(phi,M)
xlabel('Winkel Grad')
ylabel('Drehmoment Nm')
grid on
```

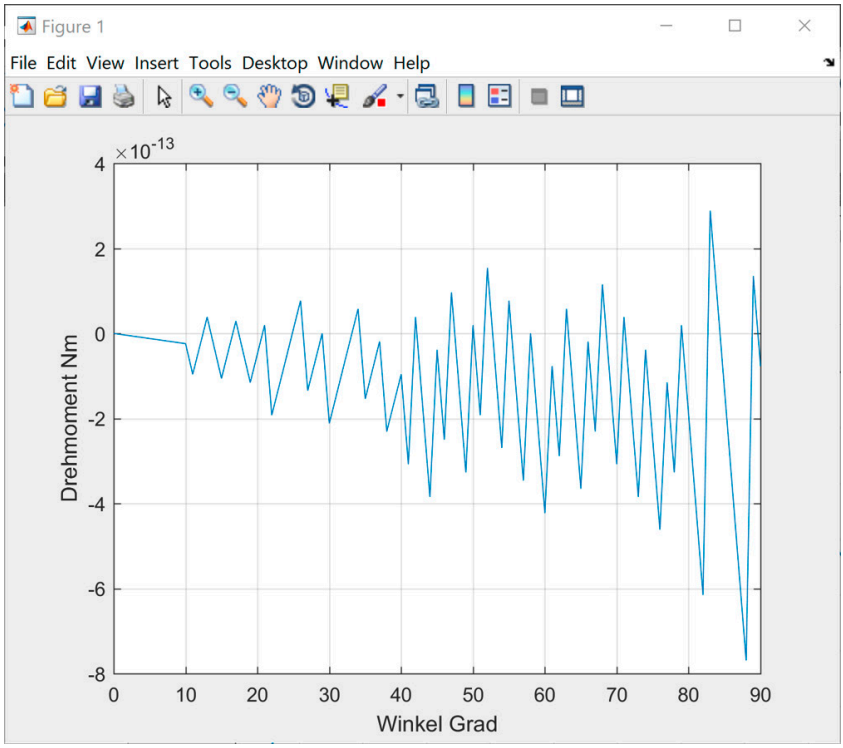


Bild Ü.5 Grafik des fehlerhaften Programms

3.2 Erzeugen von Vektoren und Matrizen

Einer der aufgeführten fünf Ausdrücke lässt sich nicht erzeugen. Welcher ist es und warum kann man ihn nicht erzeugen?

Erzeugen Sie folgende Vektoren bzw. Matrizen und prüfen Sie das Ergebnis im „Command Window“.

a) $e1 = [1 \quad 3 \quad 5]$

b) $e2 = \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix}$

c) $e3 = [1 \quad 3 \quad 5 \quad 7 \quad 9 \quad 11 \quad 13 \quad 15 \quad 17 \quad 19 \quad 21 \quad 23 \quad 25 \quad 27 \quad \ldots \quad 101]$

d) $e4 = [10 \quad 0.99 \quad 0.98 \quad 0.97 \quad \ldots \quad -0.01]$

e) $e5 = \begin{bmatrix} 10 & 0.9 & 0.8 & 0.7 & \ldots & 0 & 10 & 0.9 & 0.8 & 0.7 & \ldots & 0 \\ 1 & 2 & 3 & 4 & \ldots & 101 & 2 & 4 & 6 & 8 & \ldots & 202 \end{bmatrix}$

3.3 Vektoren und Matrizen

Für die Übung sind zunächst folgende Matrizen und Vektoren zu definieren.

$$A = \begin{bmatrix} 2 & 5 & 7 \\ 6 & 8 & 6 \\ 2 & 2 & 7 \end{bmatrix} \quad B = \begin{bmatrix} -2 & -6 \\ 4 & 4 \\ -5 & 8 \end{bmatrix} \quad c = \begin{bmatrix} 3 \\ 5 \\ 1 \end{bmatrix} \quad d = \begin{bmatrix} 1 \\ -1 \\ 2 \end{bmatrix} \quad e = \begin{bmatrix} 2 & 4 & 6 \end{bmatrix}$$

Schreiben Sie nach und nach alle Anweisungen zur Lösung dieser Aufgabe in ein MATLAB Script mit dem Namen Uebung3_3. Benutzen Sie die angegebenen Variablennamen für die Ergebnisse. Überprüfen Sie unmittelbar nachdem Sie die Anweisung(en) für eine Aufgabe „eingebaut“ haben im „Workspace Browser“ oder „Variables Editor“, ob diese korrekt arbeitet. Für einige Lösungen sind die angegebenen MATLAB Functions erforderlich bzw. effizient. Informationen zu den Functions, insbesondere zur Syntax, finden Sie in der MATLAB integrierten Dokumentation.

a) Berechnen Sie folgende Ausdrücke:

1. Addition von Vektoren

   e1 = c + d

2. Multiplikation von Vektoren

   E2 = c e

3. Elementweise Multiplikation von Matrizen

   E3 = A A

4. Addition von Matrizen

   E4 = A + E3

5. Muliplikation von Matrizen

   E5 = A B

6. Transponieren einer Matrix

   BT = B^T

7. Invertieren einer Matrix (MATLAB Function „inv“)

   AINV = A^{-1}

8. Muliplikation einer Matrix mit deren Invertierten

   E8 = A AINV

Das analytische Ergebnis ist die Einheitsmatrix. Abweichungen hiervon resultieren aus der begrenzten Rechengenauigkeit eines Computers.

b) Bauen Sie durch Teilentnahmen aus den Matrizen A und B folgende Matrix C auf:

$$C = \begin{bmatrix} 7 & 5 & -6 \\ 6 & 8 & 4 \\ 7 & 2 & 8 \end{bmatrix}$$

c) Finden Sie die Indices der Elemente in der Matrix C aus Teilaufgabe b), die größer als 5 sind, und speichern diese in einer Variablen mit dem Namen „Indices“. Benutzen Sie hierfür die MATLAB Function „find“, deren Syntax Sie in der MATLAB integrierten Dokumentation finden. Beachten Sie hierbei insbesondere die zu Grunde liegende Nummerierung der Indices. Prüfen Sie manuell, ob die Werte, die zu den Indizes gehören, tatsächlich alle größer als 5 sind.

d) Ersetzen Sie alle Indices, die Sie in Teilaufgabe c) ermittelt haben, mit einer einzigen Anweisung durch die Zahl 77 und weisen das Ergebnis einer Matrix mit dem Namen C5_77 zu.

e) In dieser Teilaufgabe soll auf das Speichern der Indices verzichtet werden, um alle Elemente der Matrix C aus Teilsuafgabe c), die größer als 5 sind, durch 77 zu ersetzen. Dies soll mit einer einzigen Anweisung, ohne Nutzung der MATLAB Function „find“ erreicht werden, in dem eine logische Indizierung für die zu ersetzenden Elemente der Matrix C verwendet wird.

f) Erzeugen Sie eine Zufallsmatrix mit 4 Zeilen und 3 Spalten, deren Werte zwischen -3 und $+3$ liegen. Benutzen Sie hierfür die MATLAB Function „rand“, deren Syntax Sie in der MATLAB integrierten Dokumentation finden.

3.4 Vektoroperationen

Berechnen Sie die unten aufgeführten Ausdrücke unter Nutzung der MATLAB Function „sum“. Das Ergebnis ist mit einer einzigen Zuweisung, zu den angegebenen Variablennamen, zu programmieren. Verwenden Sie als Namen für das MATLAB Script Uebung3_4.

a) $e1 = \sum_{i=0}^{i=99} i$

b) $e2 = \sum_{i=0}^{i=99} i^2$ (Nutzen Sie hierfür elementweise MATLAB Operatoren)

c) $e3 = \sum_{i=0}^{i=99} \sin\left(\frac{i \cdot \pi}{11}\right)$

Lösungen: a) 4950, b) 328350 c) 6.9552

3.5 Function und Schleifen

Die Aufgabe ist eine Function zu erstellen, die die kleinsten Primzahlen bis zu einer vorgebbaren Anzahl an Primzahlen bestimmt. Die Lösung soll als Function mit dem Namen „PrimeCheck“ realisiert werden. Das erste Ausgabeargument der Function soll die Zahl sein, bei der die vorgegebene Anzahl an Primzahlen erreicht wird. Das zweite optionale Ausgabeargument sollen die Primzahlen sein.

Lesen Sie vor der Bearbeitung erst alle Teilaufgaben durch.

a) Erstellen Sie eine Funktionsmaske für Function „PrimeCheck“, welche kurz die Aufgabe sowie alle Ein- und Ausgabeargumente beschreibt. Mit dem Befehl help PrimeCheck soll diese Maske im „Command Window“ erscheinen. Überprüfen Sie dies, nachdem Sie die Funktionsmaske erstellt haben.

b) Ergänzen Sie die Function „PrimeCheck“ mit den erforderlichen Anweisungen. Nutzen Sie dabei zur Auswertung, ob eine Zahl eine Primzahl ist, die MATLAB Function „isprime“. Die Primzahlen soll nur gespeichert werden, wenn die Function mit zwei Ausgabeargumenten aufgerufen wird.

c) Erstellen Sie ein MATLAB Script mit dem Namen Uebung3_5, das die Function „PrimeCheck“ einmal mit einem und danach mit zwei Ausgabeargumenten aufruft. Programmieren Sie, um die von Ihnen programmierte Function zu testen, in diesem Script zusätzlich folgende Anzeige im „Command Window“ (Beispiel für Anzahl an Primzahlen von 5):

```
Funktionsaufruf mit einem Ausgabeargument
Anzahl Primzahlen: 5
Maximale Zahl: 12
Funktionsaufruf mit zwei Ausgabeargumenten
```

```
Anzahl Primzahlen: 5
Maximale Zahl: 12
Primzahl 1: 2
Primzahl 2: 3
Primzahl 3: 5
Primzahl 4: 7
Primzahl 5: 11
```

d) Welches Ergebnis erwarten Sie für eine Anzahl an Primzahlen von 3?

e) Was ist das Ergebnis für eine Anzahl an Primzahlen von 1000?

Lösung: Von 0 bis zur Zahl 7919 gibt es 1000 Primzahlen.

3.6 Fehlerbehandlung mit Try-/Catch

Für die Übung sind zunächst folgende zwei Matrizen zu definieren.

$$A = \begin{bmatrix} 2 & 5 & 7 \\ 6 & 8 & 6 \\ 2 & 2 & 7 \end{bmatrix} \quad B = \begin{bmatrix} -2 & -6 \\ 4 & 4 \\ -5 & 8 \end{bmatrix}$$

Die folgenden beiden Ausdrücke sind nicht berechenbar und ergeben „normalerweise“ eine Fehlermeldung:

Ausdruck 1: E1 = A + B

Ausdruck 2: E2 = B A

a) Erstellen Sie ein Script mit dem Namen Uebung3_6, das „versucht“ („try“), jeden Ausdruck einzeln zu berechnen und Fehler „abfängt („catch“). Benutzen Sie als Variablennamen für die Informationen zum Fehler (MException object) Fehler1 für den Ausdruck 1 und Fehler2 für den Ausdruck 2. Überprüfen Sie im „Workspace Browser“ und „Variables Editor“, ob die beiden Variablen für Fehler richtige Ergebnisse liefern.

b) Erweitern Sie das Script um Fehleranzeigen im „Command Window“, bei denen die in den Variablen Fehler1 und Fehler 2 gespeicherten Informationen verwendet werden. Ob ein Fehler aufgetreten ist, kann dadurch festgestellt werden, ob die jeweilige Variable, d. h. Fehler1 bzw. Fehler2 existiert. Eine programmatische Abfrage ermöglicht die MATLAB Function „exist“. Die Anzeige im „Command Window“ soll folgende Informationen beinhalten:

```
1. Ausdruck in Zeile 8 ist fehlerhaft.
Fehlerursache: Matrix dimensions must agree.

2. Ausdruck in Zeile 13 ist fehlerhaft.
Fehlerursache: Inner matrix dimensions must agree.
```

3.7 Daten einlesen, verarbeiten und speichern

Erstellen Sie eine Excel® Datei, mit den in Tabelle Ü.2 gezeigten Messwerten. Speichern Sie die Datei unter dem Namen

Uebung_3_7_Excel.xlsx

im MATLAB Verzeichnis (Folder) in dem Sie diese Übung bearbeiten.

Tabelle Ü.2 Messdaten für Excel® Datei

Messpunkt	Gemessenes Signal
1	0
2	1,1
3	1
4	0,9
5	0,95
6	1,02
7	1,03
8	0,97
9	0,99
10	1

Der Signalverlauf soll nach der Messung gefiltert (geglättet) werden. Hierzu soll folgender Algorithmus verwendet werden:

$$x_F(k+1) = e^{-\frac{1}{c}} x_F(k) + \left(1 - e^{-\frac{1}{c}}\right) x_M(k);\ k = 1, 2, 3, \ldots 9, 10 \qquad (Ü.5)$$

x_F	Gefiltertes Signal
x_M	Gemessenes Signal
k	Messpunkt
c	Faktor

Dabei sind k die einzelnen Messpunkte, die nur ganzzahlige Werte annehmen.

Schreiben Sie nach und nach alle Anweisungen zur Lösung dieser Aufgabe in ein einziges MATLAB Script mit dem Namen Uebung3_7. Überprüfen Sie unmittelbar, nachdem Sie die Anweisung(en) für eine Aufgabe „eingebaut“ haben, im „Workspace Browser“ oder „Variables Editor“, ob diese korrekt arbeitet.

a) Lesen Sie die Messdaten aus der Excel® Datei ein und speichern diese in MATLAB in getrennten Variablen für die Nummer des Messpunktes (k) und den gemessenen Signalwert (xM).

b) Visualisieren Sie den gemessenen Signalverlauf (siehe Bild Ü.6). Benutzen Sie dabei die grafische MATLAB Function „stairs“.

c) Berechnen Sie den gefilterten Signalverlauf für folgende vier Werte von c = 0,5, 1, 2 und ∞. Nutzen Sie hierfür die MATLAB Function „exp“, die MATLAB Konstante „inf“ sowie die MATLAB Vektorverarbeitung. Speichern Sie die Ergebnisse für die gefilterten Signalverläufe in einer Variablen mit dem Namen xF.

```
Hinweis: Die berechneten gefilterten Signalverläufe müssen folgende
Werte ergeben:
xF =
         0         0         0         0
         0         0         0         0
    0.9511    0.6953    0.4328         0
    0.9934    0.8879    0.6560         0
    0.9126    0.8956    0.7520         0
    0.9449    0.9300    0.8299         0
    1.0098    0.9869    0.9047         0
    1.0273    1.0141    0.9540         0
    0.9778    0.9862    0.9603         0
    0.9883    0.9886    0.9720         0
```

d) Speichern Sie die Variablen k, xM und xF in einer MATLAB mat-Datei mit den Namen:

Uebung_3_7_Mat.mat

e) Löschen Sie die Variablen k, xM, und xF programmatisch im Workspace. Dies ist erforderlich, um anschließend das Einlesen der Daten überprüfen zu können.

f) Lesen Sie die Daten aus der mat-Datei ein und vervollständigen Sie die Grafik, so dass sich das in Bild Ü.6 gezeigte Aussehen ergibt.

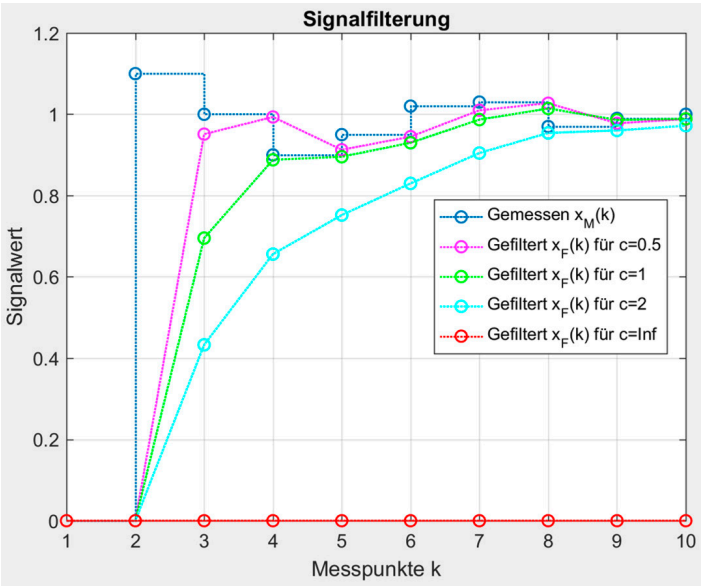


Bild Ü.6 Signalverläufe

3.8 Zweidimensionale Visualisierung und Kurvenanalyse

Der zeitliche Verlauf einer Größe y ist durch folgende Funktion beschrieben:

$$y(t) = 5 + 12t - t^2 \tag{Ü.6}$$

Diese Funktion soll grafisch visualisiert und ausgewertet werden. Ergänzen Sie nach und nach die Teilaufgaben in einem Script mit dem Namen Uebung3_8. Überprüfen Sie unmittelbar nachdem Sie die Anweisung(en) für eine Teilaufgabe erstellt haben, ob diese korrekt arbeitet.

a) Visualisieren Sie die Funktion in einem Grafikfenster für den Zeitbereich $t = 0$ s bis $t = 10$ s, und einen Zeitschritt von $h = 0{,}1$ s. Fügen Sie die in Bild Ü.7 gezeigten Achsbeschriftungen, den Titel und ein Gitternetz hinzu. Benutzen Sie die von MATLAB als Standard verwendete Linienfarbe und eine Linienstärke von 1. Zur Veränderung der Linienstärke benötigen Sie die Eigenschaft „linewidth“ der MATLAB Function „plot“.

b) Zu welchem Zeitpunkt erreicht die Kurve das Maximum? Wie groß ist das Maximum? Generieren Sie eine Anzeige im „Command Window“ (Erscheinungsbild siehe unten). Zeichnen Sie den Punkt als rotes Kreuz mit großem Marker in die Kurve ein (Bild Ü.7). Zur Veränderung der Größe des Markers benötigen Sie die Eigenschaft „markersize“ der MATLAB Function „plot“.

c) Zu welchem Zeitpunkt überschreitet y das erste Mal den Wert 30? Generieren Sie eine Anzeige im „Command Window“ (Erscheinungsbild siehe unten). Zeichnen Sie den Punkt als grünen Kreis mit großem Marker in die Kurve ein (Bild Ü.7).

d) Zu welchem Zeitpunkt überschreitet y das letzte Mal den Wert 35? Generieren Sie eine Anzeige im „Command Window“ (Erscheinungsbild siehe unten). Zeichnen Sie den Punkt als Stern in Magenta mit großem Marker in die Kurve ein (Bild Ü.7).

e) Fügen Sie eine Legende ein, welche rechts unten im Diagramm platziert sein soll (Bild Ü.7). Zur Veränderung der Platzierung der Legende benötigen Sie die Eigenschaft „location“ der MATLAB Function „legend“.

f) Fügen Sie, wie in Bild Ü.7 dargestellt, gestrichelte Linien von den Achsen zu den Markierungspunkten ein.

Die Anzeige im „Command Window“, nach Abschluss aller Teilaufgaben, soll folgendes Erscheinungsbild haben:

```
Der Maximalwert von y wird bei t=6s erreicht.
Der Maximalwert beträgt y=41.
Der y-Wert 30 wird das erste Mal bei t=2.7s überschritten.
Der y-Wert 35 wird das letzte Mal bei t=8.4s überschritten.
```

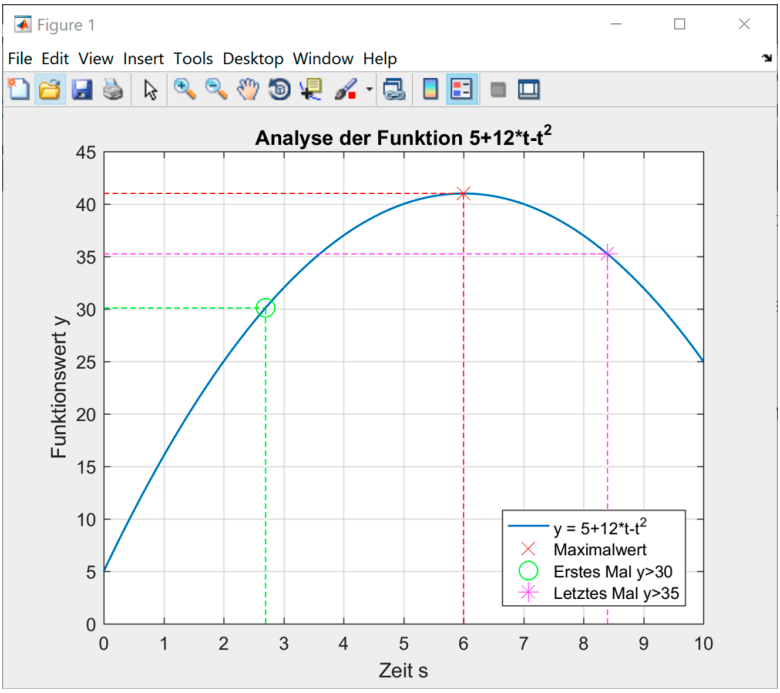


Bild Ü.7 Grafik nachdem alle Teilaufgaben abgeschlossen sind

3.9 Dreidimensionale Visualisierung

Die Oberfläche eines Formteils wird durch folgende Gleichungen analytisch beschrieben:

$$r(x,y) = \sqrt{x^2 + y^2} \tag{Ü.7a}$$

$$Z(r) = \begin{cases} \sin^2\left(\frac{\pi}{2} r\right), & 0 \le r \le 1 \\ 1, & 1 < r < \infty \end{cases} \tag{Ü.7b}$$

Ergänzen Sie nach und nach die Teilaufgaben in einem Script mit dem Namen Uebung3_9. Überprüfen Sie unmittelbar nachdem Sie die Anweisung(en) für eine Teilaufgabe erstellt haben, ob diese korrekt arbeitet.

a) Visualisieren Sie die Formfläche mit der MATLAB Function „surf“, in den Wertebereichen für die Koordinaten x und y zwischen -1 und 1, mit jeweils 50 Punkten für jede Koordinatenachse im vorgegebenen Wertebereich. Fügen Sie zusätzlich ein Gitternetz, Achsbeschriftungen und einen Titel, hinzu (siehe Bild Ü.8).

b) Ergänzen Sie, wie in Bild Ü.8 gezeigt, die Grafik um eine gestrichelt in rot eingezeichnete Randlinie für einen Radius $r = 1$.

c) Ergänzen Sie die Grafik um eine Legende (siehe Bild Ü.8).

d) Ergänzen Sie die Grafik um fünf strichpunktiert in schwarz eingezeichnete Achslinien (siehe Bild Ü.8).

e) Erweitern Sie das Script um eine Animation, unter Nutzung der MATLAB Function „pause“. Die Animation soll wie folgt ablaufen:

- Nachdem die in den vorangegangen Teilaufgaben erstellte Grafik für 2 s angezeigt wurde, soll um die Azimuthachse, in Schritten von 0,5° bis auf 0°, geschwenkt werden. Nutzen Sie hierfür die MATLAB Function „view“. Die Elevationsachse soll dabei einen konstanten Wert von 30° haben. Zwischen den einzelnen Schritten ist jeweils eine Pause von 0,1 s einzufügen. Die Winkel für die Azimuth- und Elevationsachse sind jeweils im Titel mit anzuzeigen. Ist die Grafikleistung des für die Übung benutzten Computers nicht hoch genug, kann die Schrittweite vergößert werden.
- Anschließend soll ein Querschnitt in der xz-Ebene für 4 s gezeigt werden (Azimuth- und Elevationswinkel jeweils 0°).
- Danach soll, in Winkelschritten von 1°, die Elevationsachse auf 90° gedreht werden. Zwischen den einzelnen Schritten ist jeweils eine Pause von 0,1 s einzufügen.
- Zum Abschluss soll, nach einer Pause von 2 s, die Darstellung für einen Azimuthwinkel von $-40°$, und einen Elevationswinkel von 30°, erfolgen.

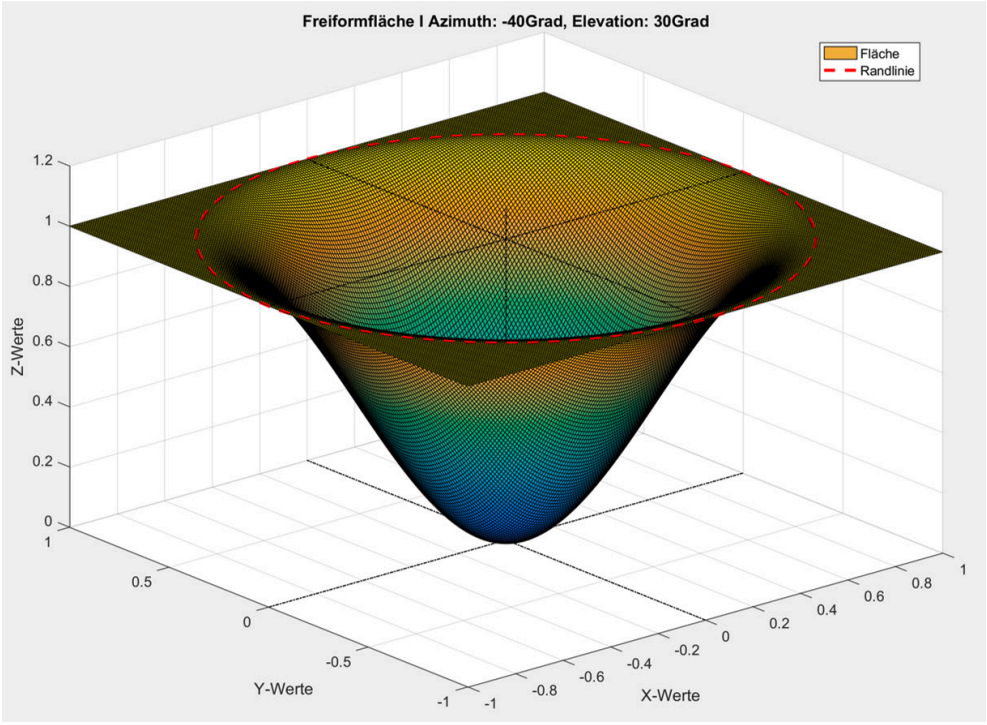


Bild Ü.8 Grafik nachdem alle Teilaufgaben abgeschlossen sind

3.10 Visualisierung von Gleichungssystemen

Folgendes lineares Gleichungssystem mit drei Unbekannten sei gegeben:

$$\begin{aligned} 4x_1 + 2x_2 + 3x_3 &= 5 \\ 3x_1 - 7x_2 + 5x_3 &= -8 \\ -6x_1 + 7x_2 + 9x_3 &= 7 \end{aligned} \tag{Ü.8}$$

Das Gleichungssystem kann auch als Matrix A und den Vektoren $\underline{b}$ und $\underline{x}$ dargestellt werden:

$$A\,\underline{x} = \underline{b}\ ;\ \underline{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \tag{Ü.9}$$

a) Erstellen Sie ein Script mit dem Namen Uebung3_10, in dem die Matrix A und der Vektor $\underline{b}$ als Variablen mit den entsprechenden Werten definiert sind.

b) Weisen Sie nach, dass das Gleichungssystem eine eindeutige Lösung besitzt. Nutzen Sie hierfür die MATLAB Function „det“.

c) Lösen Sie das Gleichungssystem mit Nutzung des MATLAB Operators „left matrix divide \“. Überprüfen Sie das Ergebnis durch eine eigene analytische Lösung des Gleichungssystems.

Lösung

In MATLAB *Analytische Lösung*

$$\underline{x} = \begin{pmatrix} 0.5220 \\ 1.3954 \\ 0.0404 \end{pmatrix} \quad \underline{x} = \begin{pmatrix} \frac{891}{1707} \\ \frac{2710716}{1942566} \\ \frac{23}{569} \end{pmatrix}$$

d) Die drei Gleichungen können jeweils auch als Ebenengleichungen, in einem dreidimensionalen rechtwinkligen Koordinatensystem, interpretiert werden. Die Achsen des Koordinatensystems (x, y, z) werden dazu wie folgt definiert:

$$\begin{aligned} x &= x_1 \\ y &= x_2 \\ z &= x_3 \end{aligned} \tag{Ü.10}$$

Die Ebenengleichungen können zeilenweise wie folgt dargestellt werden:

$$\underline{z} = C\begin{pmatrix} x \\ y \end{pmatrix} + \underline{d}\ ;\ \underline{z} = \begin{pmatrix} z_1 \\ z_2 \\ z_3 \end{pmatrix} \tag{Ü.11}$$

Berechnen Sie die Matrix C, und den Vektor $\underline{d}$, mit Nutzung der MATLAB Anweisungen zur Matrix- bzw. Vektorverarbeitung. Überprüfen Sie die Ergebnisse im „Variables Editor“ mit einer eigenen analytischen Lösung.

e) Visualisieren Sie die drei Ebenen in einer dreidimensionalen Grafik. Berechnen Sie hierzu in MATLAB die drei Matrizen für die Ebenen mit den Variablennamen Z1, Z2 und Z3. Nutzen Sie hierfür for-Schleifen und die MATLAB Function „eval“ zur Wertezuweisung zu den Variablen Z1, Z2 und Z3. Die Farbe der ersten Ebene soll orange (RGB-Farbe in MATLAB [1 0.5 0]), die der zweiten Ebene cyan und die der dritten Ebene magenta sein. Die Ebenen sollen keine Rahmenlinien besitzen. Die Einstellung erfolgt über die Eigenschaft „EdgeColor“ auf ‚none‘ in der MATLAB Function „surf“. Das Ergebnis soll wie in Bild Ü.9 gezeigt aussehen.

f) Zeichnen Sie zusätzlich den in MATLAB berechneten Schnittpunkt der Ebenen mit einem schwarzen Kreuz ein. Setzen Sie die Eigenschaft für die Größe des Markers auf 16. Das Ergebnis soll wie in Bild Ü.9 gezeigt aussehen.

g) Erweitern Sie die Grafik um eine Legende. Die Legende soll oben rechts eingefügt werden (siehe Bild Ü.9). Hierfür benötigen Sie die Eigenschaft „location“ der MATLAB Function „legend“.

h) Fügen Sie der Grafik die in Bild Ü.9 gezeigten strichpunktierten und gepunkteten Orientierungslinien, in einer Linienstärke von jeweils 0,75, hinzu.

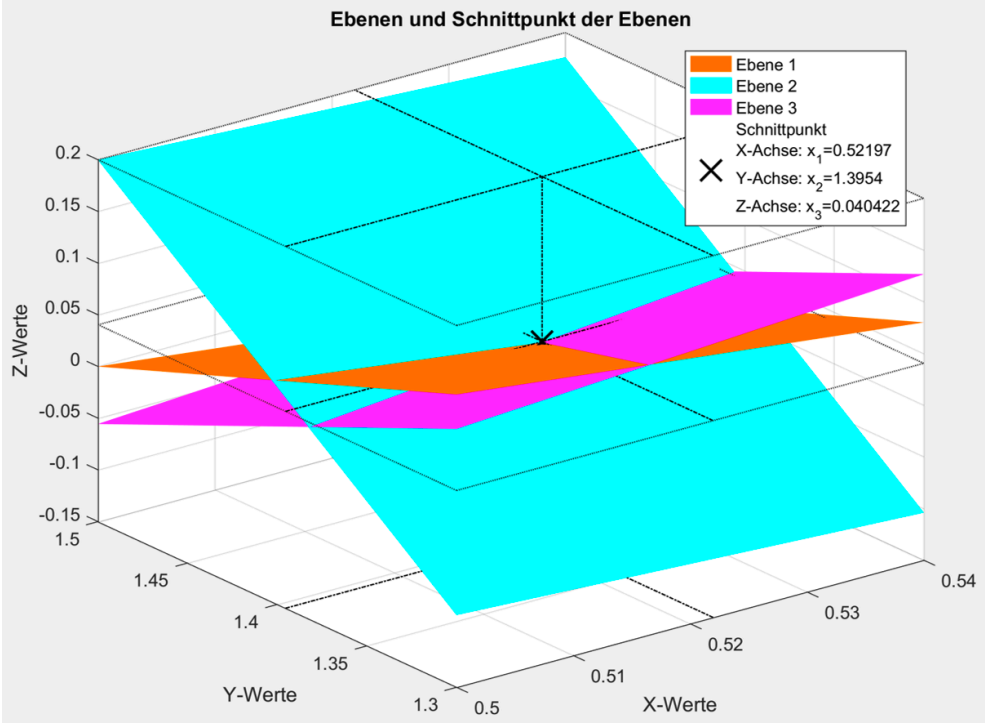


Bild Ü.9 Ebenen und Schnittpunkt der Ebenen

Übung 4 – Grafische Bedienoberflächen

Zur zeiteffizienten Durchführung der Übung sind einige Files vorgefertigt. Diese Files sind bei den Programmbeispielen zum Lehrbuch im folgenden Unterverzeichnis abgelegt: Uebungen/Uebung4/Vorlagen

Für das aus dem Kapitel „Grafische Bedienoberflächen“ bekannte schwingungsfähige System (Bild Ü.10) soll eine grafische Bedienoberfläche erstellt werden.

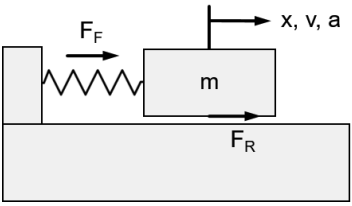


Bild Ü.10 Schwingungsfähiges System (Ein-Masse-Schwinger)

Die das System charakterisierenden mechanischen Kenngrößen sind:

- Masse (m)
- Federkonstante (c)
- Dämpfungskoeffizient (d)

Daraus können die beiden charakteristischen Kenngrößen einer gedämpften Schwingung berechnet werden:

$$\omega_0 = \sqrt{\frac{c}{m}} \tag{Ü.1a}$$

$$D = \frac{1}{2} d \sqrt{\frac{1}{c\,m}} \tag{Ü.1b}$$

ω_0	Kennkreisfrequenz	*Characteristic angular frequency*	rad/s
D	Dämpfungsgrad	*Damping grade*	

Die Kreisfrequenz der Schwingung wird als Eigenkreisfrequenz bezeichnet und berechnet sich zu:

$$\omega_N = \omega_0 \sqrt{1 - D^2} \tag{Ü.2}$$

ω_N	Eigenkreisfrequenz	*Natural angular frequency*	rad/s

Lenkt man die Masse aus der Ruheposition aus, so kann die Periodendauer bestimmt werden (Bild Ü.11). Die Eigenkreisfrequenz und die Eigenfrequenz kann daraus berechnet werden:

$$\omega_N = \frac{2}{T} \quad f_N = \frac{1}{T} \tag{Ü.3}$$

f_N	Eigenfrequenz	*Natural frequency*	Hz
T	Periodendauer	*Period time*	s

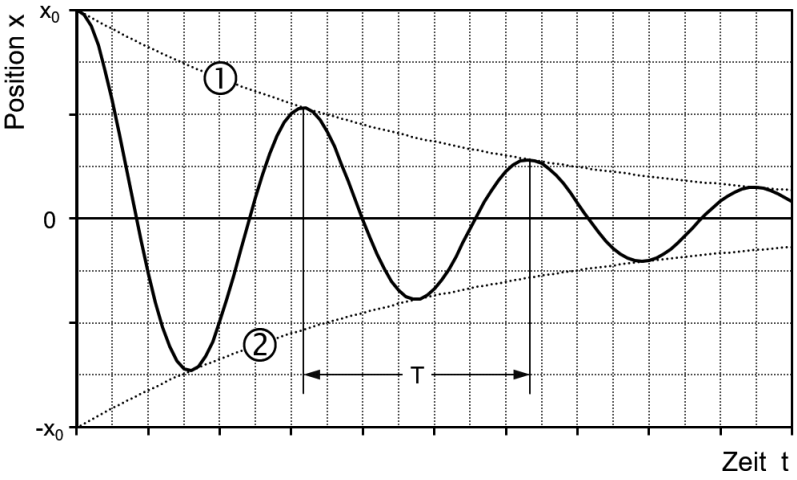


Bild Ü.11 Zeitliches Verhalten der Position

Die analytische Lösung für das zeitliche Verhalten der Position x lautet:

$$x(t) = x_0 \underbrace{\frac{e^{-D\omega_0 t}}{\sqrt{1-D^2}}}_{x_{Ein}} \sin\left\{\underbrace{\omega_0\sqrt{1-D^2}}_{N} t + \arccos(D)\right\};\ 0 < D < 1 \tag{Ü.4}$$

x	Position	*Position*	m
x_0	Auslenkung zum Zeitpunkt t = 0	*Expansion at time t = 0*	m
x_{Ein}	Hüllkurve	*Envelope*	m

Für das System Ein-Masse-Schwinger soll Schritt für Schritt die in Bild Ü.12 gezeigte grafische Bedienoberfläche erstellt werden. Die in der Übung zu erstellende Bedienoberfläche finden Sie auch als App (eigenständige Applikation) bei den Programmbeispielen zum Lehrbuch im folgenden Unterverzeichnis: Eigenständige Applikationen/Uebung4. Hinweise zur Installation einer MATLAB App finden sich im Abschnitt A.1.2 „Eigenständige Applikationen“. Der Filename zur web-basierten Installation der App ist:

AppInstaller_Uebung4_Web.exe

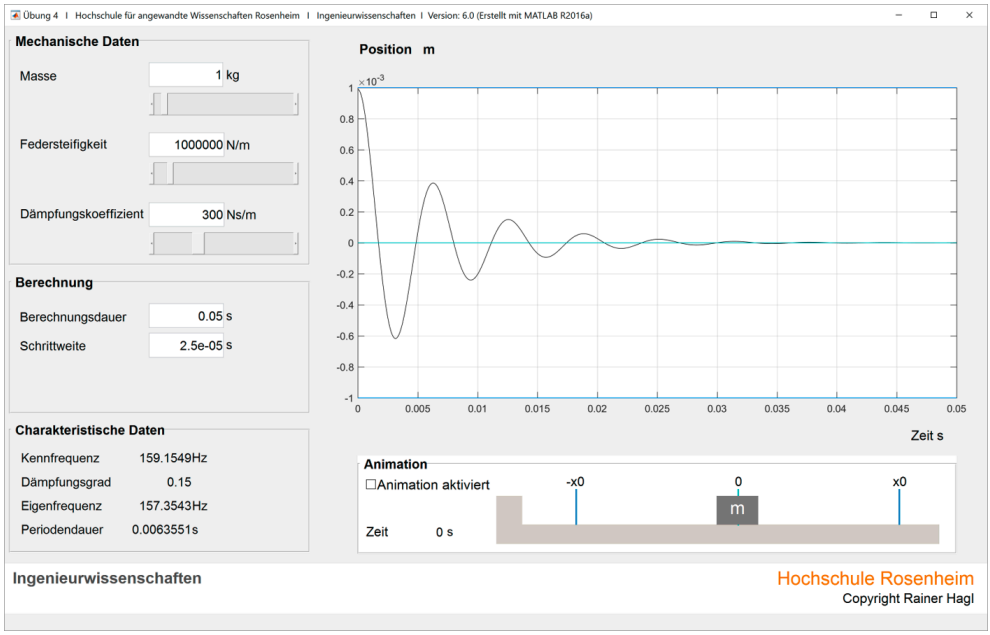


Bild Ü.12 Grafische Bedienoberfläche für Ein-Masse-Schwinger

Die in Bild Ü.13 gezeigte Vorlage (Dateiname: Uebung4Figure.fig) kann zur Erstellung benutzt werden und ist bei den Files zur Übung zum Download gespeichert.

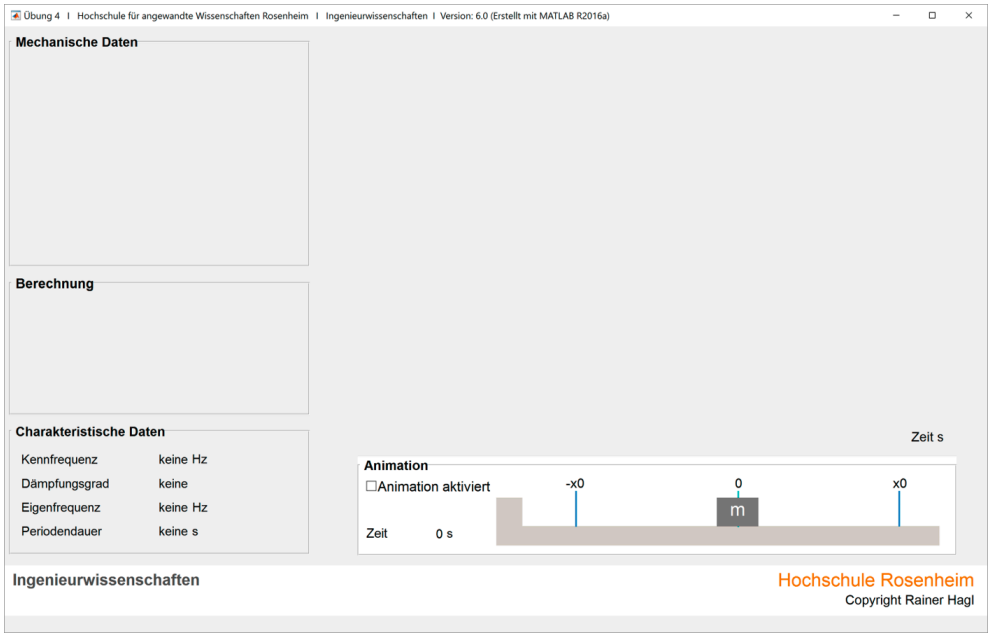


Bild Ü.13 Vorlage

Für die im Rahmen der Vorlesung und dieser Übung behandelten Teilaspekte von grafischen Bedienoberflächen sind die nachfolgend beschriebenen Einstellungen bereits vorgenommen (Bild Ü.14 und Bild Ü.15).

- Resize Behavior

  „Proportional“ ausgewählt (dadurch kann die Größe der programmierten grafischen vom Nutzer verändert werden)

- Command-line accessibility

  On (GUI may become Current Figure from Command Line)

- „Generate FIG file only“ ausgewählt

  Eigenschaft (property) HandleVisibility in der von Ihnen programmierten grafischen Bedienoberfläche muss auf on eingestellt sein (Bild Ü.15). Sie darf nicht auf callback oder off eingestellt sein.

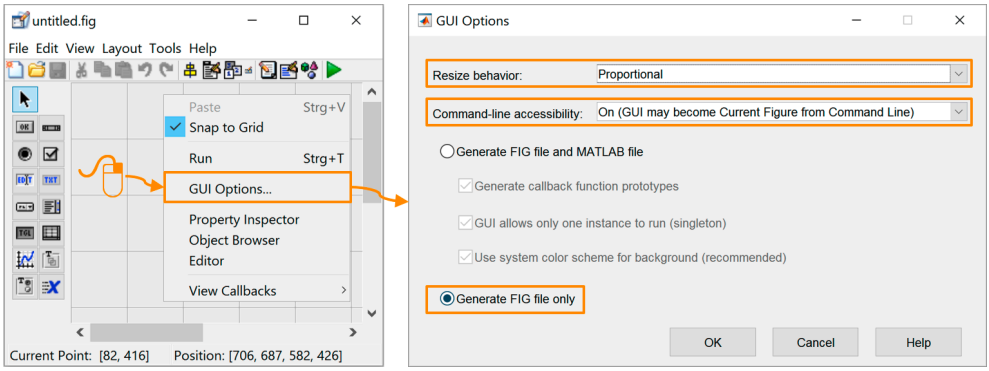


Bild Ü.14 Einstellung GUI Options

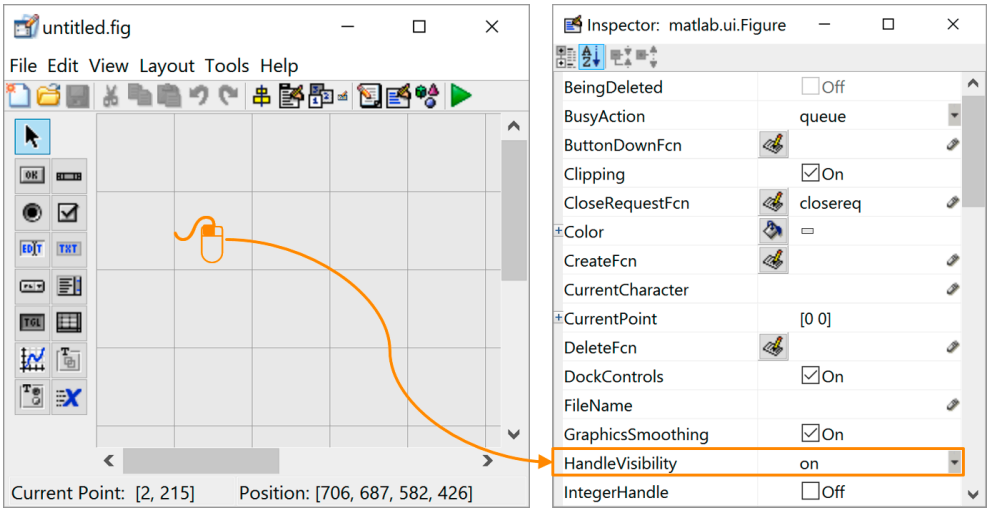


Bild Ü.15 Einstellung HandleVisibility

Wenn Sie nicht Generate FIG file only auswählen, können Sie sich, mit den in Abschnitt 3.2 behandelten Punkten, das Verhalten an einigen Stellen nicht erklären und es gibt Fehler, die Sie nicht lösen können. Bitte beachten Sie dies, wenn Sie selbst neue Files erstellen. Entsprechendes gilt für die Einstellung der „Command-line accessibility“.

Bei falscher Einstellung der HandleVisibility werden die Identifizierungskennzeichen nicht richtig zugeordnet!

Bitte beachten Sie dies, wenn Sie grafische Bedienoberflächen von Anfang an selbst erstellen.

Zur Lösung der Aufgabe sind die in Tabelle Ü.3 gezeigten Variablennamen zu verwenden. Um die Übersichtlichkeit zu erhöhen, sind zur Lösung der Aufgabe die in Tabelle Ü.4 aufgeführten Programmnamen zu verwenden. Die Aufgaben der einzelnen Programme sind kurz beschrieben. Der Status der Programme direkt nach dem Download ist ebenfalls angegeben. Die Bedienoberfläche wird mit „Uebung4“ gestartet (Script Ü.1). Den Kommentaren der Anweisungen können deren Aufgaben entnommen werden.

Tabelle Ü.3 Variablennamen

Parameter	Variablenname	Einheit
Berechnungsdauer	Berechnungsdauer	s
Dämpfungsgrad	Daempfungsgrad	
Eigenfrequenz	Eigenfrequenz	Hz
Federsteifigkeit	Federsteifigkeit	N/m
Kennfrequenz	Kennfrequenz	Hz
Masse	Masse	kg
Kennkreisfrequenz	Omega0	rad/s
Eigenkreisfrequenz	OmegaN	rad/s
Periodendauer	Periodendauer	s
Dämpfungskoeffizient	Daempfungskoeffizient	Ns/m
Zeitpunkt der Berechnung	t	s
Position	x	m
Position zum Zeitpunkt $t = 0$	x0	m

Tabelle Ü.4 MATLAB Programmnamen (ohne Callback Programme)

Programmnamen	Status ①	Beschreibung
AnalytischeLoesung.m	✓	Analytische Lösung für den zeitlichen Verlauf der Position
BerechnungCharakteristischeDaten.m	✓	Berechnung der charakteristischen Daten
BerechnungPositionsverlauf.m	✓	Berechnung des zeitlichen Positionsverlaufes
GUIAnimation.m	0	Animation in der Bedienoberfläche
GUIGrafik.m	0	Grafik in der Bedienoberfläche
Uebung4Figure.fig	Δ	Grafische Bedienoberfläche
InitialisierungVariablen.m	✓	Initialisierung der Variablen
Uebung4.m	✓	Start der Bedienoberfläche
SetGUIDaten.m	0	Aktualisierung der grafischen Bedienoberfläche

① Wie diese bei den Programmbeispielen abgelegt sind.

✓ Diese Programme müssen nicht verändert werden. Sie wurden vorab erstellt, damit das Übungsthema „grafische Bedienoberfläche" zeiteffizient bearbeitet werden kann.

0 Diese Programme sind nur Vorlagen ohne Befehle und müssen erstellt werden.

Δ Diese Progamme sind Vorlagen und müssen ergänzt werden.

Tabelle Ü.5 Callback Programmnamen

Programmnamen	Beschreibung
CallBackMasse.m	Numerische Eingabe der Masse
CallBackSliderMasse.m	Slider Masse
CallBackFedersteifigkeit.m	Numerische Eingabe der Federsteifigkeit
CallBackSliderFedersteifigkeit.m	Slider für die Federsteifigkeit
CallBackDaempfungskoeffizient.m	Numerische Eingabe des Dämpfungskoeffizienten
CallBackSliderDaempfungskoeffizient.m	Slider für den Dämpfungskoeffizienten
CallBackBerechnungsdauer.m	Numerische Eingabe Berechnungsdauer
CallBackSchrittweite.m	Numerische Eingabe der Schrittweite
CallBackAnimationSteuerung	Check Box zur Steuerung der Animation (Animation aktiv oder inaktiv)

Script Ü.1 Script zum Starten der Übung 4 (Uebung4.m)

```
% Initialisierung der Variablen
InitialisierungVariablen;
% Berechnung der charakteristischen Daten
BerechnungCharakteristischeDaten;
% Öffnen der grafischen Bedienoberfläche (Grafical User Interface, GUI)
open(,Uebung4Figure.fig');
% Variablenzuweisung der Handle (Identifizierungskennzeichen) der GUI
GUI.Handles = guihandles;
% Zuweisung aller variablen Eigenschafen von grafischen Bedienelementen
% der grafischen Bedienoberfläche
SetGUIDaten;
% Positionsverlauf berechnen (Analytische Lösung)
BerechnungPositionsverlauf;
% Grafik in GUIanzeigen
GUIGrafik;
```

Das Script „InitialisierungVariablen“ initialisiert alle erforderlichen Variablen. Die charakteristischen Daten der Schwingung, wie Eigenfrequenz und Dämpfungsgrad, werden im Script „BerechnungCharakteristischeDaten“ berechnet. Anschließend wird die grafische Bedienoberfläche mit dem Namen „Uebung4Figure.fig“ gestartet. Die Handle der grafischen Bedienoberfläche werden dem Structure Array „GUI.Handles“ zugewiesen. Die Zuweisung der Eigenschaften einzelner grafischer Bedienelemente erfolgt mit dem Script „SetGUIDaten“.

Eine grobe Orientierung zur schrittweisen Erstellung der grafischen Bedienoberfläche zeigt Tabelle Ü.6.

Damit die Übungsaufgabe zeiteffizient gelöst werden kann, sind die angegebenen Schritte bei der Programmierung einzuhalten.

Die Verarbeitung von Eingaben in der grafischen Bedienoberfläche erfolgt erst ab Arbeitsschritt ❻. In Arbeitsschritt ❶ bis ❺ ist noch keine Funktion zur Verarbeitung von Eingaben in der grafischen Bedienoberfläche durch den Nutzer vorgesehen.

Tabelle Ü.6 Schrittweises Vorgehen zur Erstellung der grafischen Bedienoberfläche

❶	Platzieren Sie alle Bedienelemente im Bereich (panel) für „Mechanische Daten“ und benennen Sie die einzelnen Bedienelemente wie in Bild Ü.14 dargestellt.
❷	Erstellen Sie im Skript mit dem Programmnamen „SetGUIDaten“ die erforderlichen Anweisungen, so dass die charakteristischen Daten der Schwingung im Bereich (panel) „Charakteristische Daten“ nach dem Start der grafischen Bedienoberfläche mit dem Script „Uebung4“ angezeigt werden.
❸	Platzieren Sie alle Bedienelemente im Bereich (panel) „Berechnung“ und benennen die einzelnen Bedienelemente wie in Bild Ü.12 dargestellt. *In diesem Arbeitsschritt sind nur die Bedienelemente zu platzieren und die Eigenschaften String, Tag und TooltipString einzustellen.*
❹	Erweitern Sie das Script mit dem Programmnamen „SetGUIDaten“ so dass alle Daten im Bereich (panel) „Mechanische Daten“ und „Berechnung“ nach dem Start der grafischen Bedienoberfläche den Werten im Workspace entsprechen.
❺	Erstellen Sie ein Grafikprogramm (GUIGrafik), um den Positions-Zeit-Verlauf darzustellen, das nach dem Start der grafischen Bedienoberfläche mit dem Script „Uebung4“ angezeigt wird.
❻	Erstellen Sie die Callback für die Edit, Slider und Check Box Bedienelemente. In der Property (Eigenschaft) des Bedienelementes ist der Callback der Einfachheit halber als Script zu realisieren. Für die Scripts sind die in Tabelle Ü.5 aufgeführten Callback Programmnamen zu verwenden. Gehen Sie dabei wie folgt vor: ▪ Beginnen Sie zunächst mit dem Edit Bedienelement für die Masse, für das der Callback Programmname „CallBackMasse“ (MATLAB Script) vorgesehen ist. ▪ Überprüfen Sie als erstes, ob Eingaben in der Variablen für die Masse (Variablenname: Masse) im Workspace korrekt gespeichert werden. ▪ Überprüfen Sie anschließend, ob fehlerhafte Eingaben, wie Werte ≤ 0 oder alfanumerische Werte ignoriert werden. Im Workspace muss nach einer fehlerhaften Eingabe weiterhin der vorherige Wert für die Masse gespeichert sein und im Edit Bedienelement die fehlerhafte Eingabe durch den Wert der Masse aus dem Workspace ersetzt sein. ▪ Ergänzen Sie den Callback so, dass die charakterischen Daten und die Grafik für den Positionsverlauf aktualisiert werden. ▪ Gehen Sie entsprechend für die restlichen Bedienelemente vor.
❼	Testen Sie systematisch alle Programmfunktionen.
❽	Erstellen Sie ein Programm (GUIAnimation) zur Animation der Bewegung, das nach Start der Einzelberechnung (Push button: Einzel) die Bewegung grafisch visualisiert.

Übung 5 – Zahlenformate

5.1 Wandlung Dezimalzahl in Binärzahl

a) Wandeln Sie die Dezimalzahl 54 in eine 8Bit-Binärzahl um. Die Binärzahl soll im Zweierkomplement dargestellt werden. Stellen Sie den Lösungsweg ohne Nutzung von MATLAB dar.

b) Wandeln Sie die Dezimalzahl -20 in eine 8Bit-Binärzahl um. Die Binärzahl soll im Zweierkomplement dargestellt werden. Stellen Sie den Lösungsweg ohne Nutzung von MATLAB dar.

5.2 Wandlung Binärzahl in Dezimalzahl

a) Wandeln Sie die im Zweierkomplement dargestellte 8Bit-Binärzahl 00100010 in eine Dezimalzahl um. Stellen Sie den Lösungsweg ohne Nutzung von MATLAB dar.

b) Wandeln Sie die im Zweierkomplement dargestellte 8Bit-Binärzahl 10000100 in eine Dezimalzahl um. Stellen Sie den Lösungsweg ohne Nutzung von MATLAB dar.

5.3 Wandlung Dezimalzahl in Hexadezimalzahl

a) Wandeln Sie die Dezimalzahl 3119 in eine 16Bit-Hexadezimalzahl um. Stellen Sie den Lösungsweg ohne Nutzung von MATLAB dar.

b) Wandeln Sie die Dezimalzahl -20 in eine 8Bit-Hexadezimalzahl um. Stellen Sie den Lösungsweg ohne Nutzung von MATLAB dar.

5.4 Wandlung von Hexadezimalzahlen

a) Wandeln Sie die 8Bit-Hexadezimalzahl $5A_{hex}$ in eine Dezimalzahl und in eine 8Bit-Binärzahl um. Stellen Sie den Lösungsweg ohne Nutzung von MATLAB dar.

b) Wandeln Sie die 8Bit-Hexadezimalzahl $F1_{hex}$ (Darstellung im Zweierkomplement) in eine Dezimalzahl um. Stellen Sie den Lösungsweg ohne Nutzung von MATLAB dar.

5.5 Wertebereich

a) Es stehen 12Bit zur Verfügung. Welcher ganzzahlige Wertebereich ergibt sich, sofern positive und negative Zahlen darzustellen sind?

b) Wie viele Bit sind mindestens erforderlich, um alle ganzen Zahlen zwischen –10 000 und 10 000 darstellen zu können?

5.6 Byteangaben

a) Wie viele Byte haben 64Bit? Stellen Sie den Lösungsweg dar.

b) Zum Speichern von Zahlen steht ein Speicherbereich von 1MByte zur Verfügung. Wie viele 64Bit-Zahlen können maximal gespeichert werden? Stellen Sie den Lösungsweg dar.

5.7 Gleitkommazahlen

a) Zur Darstellung einer Gleitkommazahl wird das 32Bit-Format nach IEEE 754 (single precision) verwendet. Welcher Gleitkommazahl entspricht folgendes Bitmuster?

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	1	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Stellen Sie den Lösungsweg dar.

b) Die Gleitkommazahl 18,25 soll im 32Bit-Format nach IEEE 754 (single precision) dargestellt werden. Geben Sie das Bitmuster an. Stellen Sie den Lösungsweg dar.

5.8 Komplexe Zahlen

Erstellen Sie ein MATLAB Script, das Berechnungen für beliebige komplexe Zahlen z durchführt. Benutzen Sie als Testfall folgenden Spaltenvektor mit komplexen Zahlen:

$$z = \begin{bmatrix} i \\ 1+i \\ -2+2i \end{bmatrix}$$

Schreiben Sie nach und nach alle Anweisungen zur Lösung dieser Aufgabe in ein einziges MATLAB Script mit dem Namen Uebung5_8. Benutzen Sie die angegebenen Variablennamen für die Ergebnisse. Überprüfen Sie unmittelbar nachdem Sie die Anweisung(en) für eine Aufgabe „eingebaut“ haben, im Workspace Browser oder Variables Editor, ob diese korrekt arbeitet. Für einige Lösungen sind die angegebenen MATLAB Functions erforderlich bzw. effizient. Informationen zu den MATLAB Functions, insbesondere zur Syntax, finden Sie in der MATLAB integrierten Dokumentation.

a) Elementweise Quadrieren

$e1 = z^2$

b) Real- und Imaginärteil extrahieren (MATLAB Functions „real“ und „imag“)

$e2R = Re(z)$

$e2I = Im(z)$

c) Betrag (MATLAB Function „abs“)

$e3 = [z]$

d) Phasenwinkel von z in Grad (MATLAB Function „angle“)

$e4 = ?$

e) Konjugiert komplexe Zahlen von z (MATLAB Function „conj“)

$e5 = ?$

f) Elementweise Berechnung

$$e6 = \left(\frac{|z|}{z} \right)^4$$

Übung 6 – Zeitgesteuerte Systeme (Simulink)

Zur zeiteffizienten Durchführung der Übung sind einige Files vorgefertigt. Diese Files sind bei den Programmbeispielen zum Lehrbuch im folgenden Unterverzeichnis abgelegt: Uebungen/Uebung6/Vorlagen

Für das bereits bekannte schwingungsfähige System (Bild Ü.16) soll ein Modell in Simulink erstellt werden.

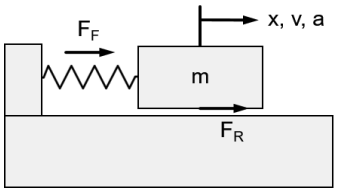


Bild Ü.16 Schwingungsfähiges System (Ein-Masse-Schwinger)

Die Bewegungsgleichung (Differentialgleichung) lautet:

$$\underbrace{m\ddot{x}}_{\substack{\text{Beschleunigungs-}\\ \text{kraft}}} + \underbrace{d\dot{x}}_{\substack{\text{Reib-}\\ \text{kraft}}} + \underbrace{cx}_{\substack{\text{Feder-}\\ \text{kraft}}} = 0 \qquad (Ü.5)$$

m	Masse	*Mass*	kg
c	Federsteifigkeit	*Spring stiffness*	N/m
d	Dämpfungskoeffizient	*Damping coefficient*	Ns/m
x	Position	*Position*	m
$\dot{x}$	Geschwindigkeit	*Velocity*	m/s
$\ddot{x}$	Beschleunigung	*Acceleration*	m/s^2

Als Anfangsbedingung (Wert zum Zeitpunkt t=0) für die Position und die Geschwindigkeit soll von folgenden Werten ausgegangen werden:

Position $\quad x(t_0) = x_0; \quad t_0 : t = 0 \qquad (Ü.6a)$

Geschwindigkeit: $\quad \dot{x}(t_0) = v(t_0) = 0 \qquad (Ü.6b)$

Durch Nutzung der Geschwindigkeit als weitere Größe ergeben sich zwei Differentialgleichungen erster Ordnung.

① Differentialgleichung für die Geschwindigkeit

$$\dot{x} = v \qquad (Ü.7a)$$

② Differentialgleichung für die Beschleunigung

$$\dot{v} = -\frac{d}{m}v - \frac{c}{m}x \qquad (Ü.7b)$$

Das vollständige Simulink Modell zur numerischen Berechnung, zur Visualisierung, und zum Speichern der Berechnungsergebnisse im Workspace, zeigt Bild Ü.17.

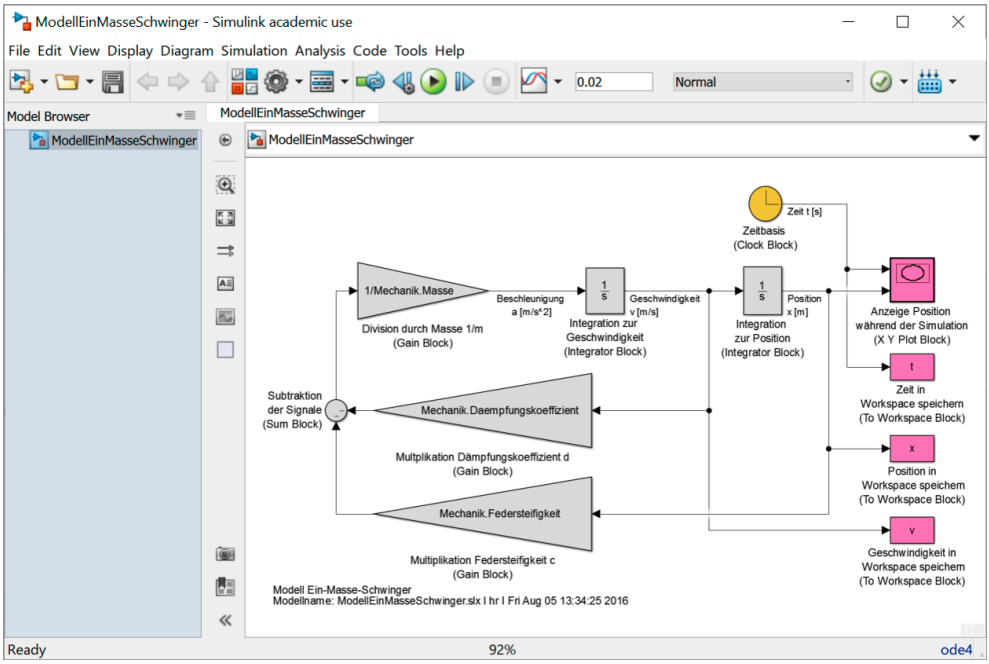


Bild Ü.17 Simulink Modell

Um die Übung zeiteffizient bearbeiten zu können, gibt es eine Vorlage die zum Gesamtsystem zu ergänzen ist (Bild Ü.18). Diese kann heruntergeladen werden. Der Name der Simulink Modellvorlage (Englisch: Template) ist:

ModellVorlage.slx

Diese Vorlage ist ohne die Ergänzungen nicht lauffähig. Beim Start ergeben sich Fehlermeldungen.

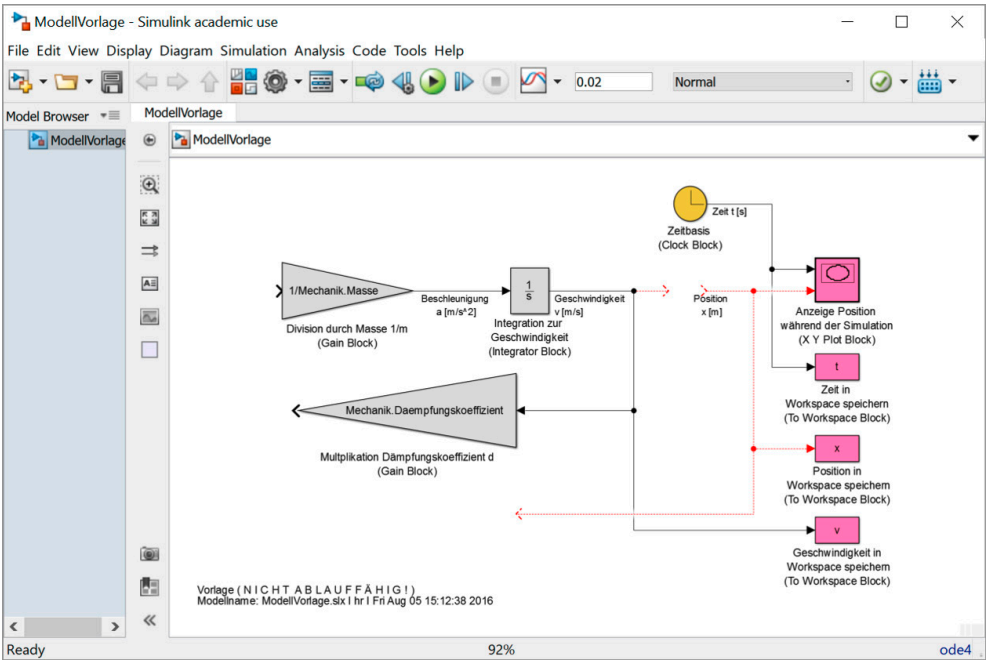


Bild Ü.18 Vorlage (ModellVorlage.slx)

Zur Lösung der Aufgabe werden die in Tabelle Ü.7 gezeigten Variablennamen verwendet. Nicht aufgeführte Variablennamen sind für das grundsätzliche Verständnis nicht von Bedeutung.

Tabelle Ü.7 Variablennamen

			Variablenname	Wert nach Initialisierung	Einheit
Mechanik	Masse	m	Mechanik.Masse	1	kg
	Federsteifigkeit	c	Mechanik.Federsteifigkeit	1e6	N/m
	Dämpfungskoeffizient	d	Mechanik.Daempfungskoeffizient	300	Ns/m
	Position zum Zeitpunkt t = 0	x_0	Mechanik.x0	0.001	m
Simulation	Berechnungsdauer	T_{Final}	Simulation.TFinal	0.02	s
Numerische Integration (solver)	Numerisches Integrationsverfahren		Solver.Type		s
	Schrittweite für numerische Integration	h	Solver.FixedStep	0.0001	s

Tabelle Ü.7 Variablennamen *(Fortsetzung)*

			Variablenname	Wert nach Initialisierung	Einheit
Workspace	Berechnungszeitpunkt	t	t		s
	Position	x	x		m
	Geschwindigkeit	v	v		m/s
	Schrittweite	h	h		s
	Berechnungsfehler	e	e		m

Zur Lösung der Übungsaufgabe werden die in Tabelle Ü.8 aufgeführten Programme verwendet. Die Aufgaben der einzelnen Programme sind kurz beschrieben. Der Status der Programme wie diese bei den Programmbeispielen abgelegt sind, ist ebenfalls angegeben.

Tabelle Ü.8 MATLAB Programmnamen

Programmname	Status①	Beschreibung
InitialisierungDaten.m	✓	Initialisierung der Daten
ModellVorlage.slx	Δ	Simulink Modellvorlage zur Ergänzung (Template)
Visualisierung.m	✓	Visualisierung der Berechnungsergebnisse
AnalytischeLoesung.m	✓	Analytische Lösung
EinMasseSchwinger.m	✓	Start der Berechnung mit konstanter Schrittweite und Visualisierung
VariableStep.m	✓	Start der Berechnung mit variabler Schrittweite und Visualisierung

① Wie diese bei den Programmbeispielen abgelegt sind.

✓ Diese Scripts müssen nicht verändert werden. Sie wurden erstellt, damit die Übung zeiteffizient bearbeitet werden kann.

Δ Dieses Script ist eine Vorlage und muss ergänzt werden.

Zur Initialisierung aller Variablen ist das MATLAB Script „InitialisierungDaten.m“ aufzurufen. Es initialisiert alle erforderlichen Variablen.

Die in Bild Ü.18 gezeigte Simulink Modellvorlage ist im File „ModellVorlage.slx“ gespeichert. Damit die vorgefertigten Programme funktionieren, benennen Sie die Datei „ModellVorlage.slx“ vor Ihren Ergänzungen in „ModellEinMasseSchwinger.slx“ um. Ein Aufruf des MATLAB Script „Visualisierung.m“ visualisiert die Ausgabe der im Workspace gespeicherten Simulationsergebnisse (Bild Ü.19). Mit dem MATLAB Script „EinMasseSchwinger.m“ erfolgt ein automatisierter Ablauf der Simulation und Ergebnisvisualisierung (Script Ü.2). Dabei wird eine konstante Schrittweite (fixed-step) benutzt.

Script Ü.2 MATLAB Script für konstante Schrittweite (EinMasseSchwinger.m)

```
% Start Simulink Berechnung
SimulationOutput = ...
 sim('ModellEinMasseSchwinger',...
     'Solver',Solver.FixedStepType,...
     'FixedStep',num2str(Solver.FixedStep),...
     'StopTime',num2str(Simulation.TFinal));
% Simulationsergebnisse im Workspace speichern
t = SimulationOutput.get('t');
x = SimulationOutput.get('x');
v = SimulationOutput.get('v');
% Schrittweiten Modus für die Visualisierung festlegen
Solver.Schrittweitenmodus = 'fixed step';
% Visualisierung der Berechnungsergebnisse
Visualisierung;
```

Für Berechnungen mit variabler Schrittweite (variable-step) ist das MATLAB Script „VariableStep.m“ zu benutzen (Script Ü.3).

Script Ü.3 MATLAB Script für variable Schrittweite (VariableStep.m)

```
% Start Simulink Berechnung
SimulationOutput = ...
 sim('ModellEinMasseSchwinger',...
     'Solver',Solver.VariableStepType,...
     'AbsTol',num2str(Solver.AbsTol),...
     'RelTol',num2str(Solver.RelTol),...
     'MaxStep',num2str(Simulation.TFinal/20),...
     'StopTime',num2str(Simulation.TFinal));
% Simulationsergebnisse im Workspace speichern
t = SimulationOutput.get('t');
x = SimulationOutput.get('x');
v = SimulationOutput.get('v');
% Schrittweiten Modus für die Visualisierung festlegen
Solver.Schrittweitenmodus = 'variable step';
% Visualisierung der Berechnungsergebnisse
Visualisierung;
```

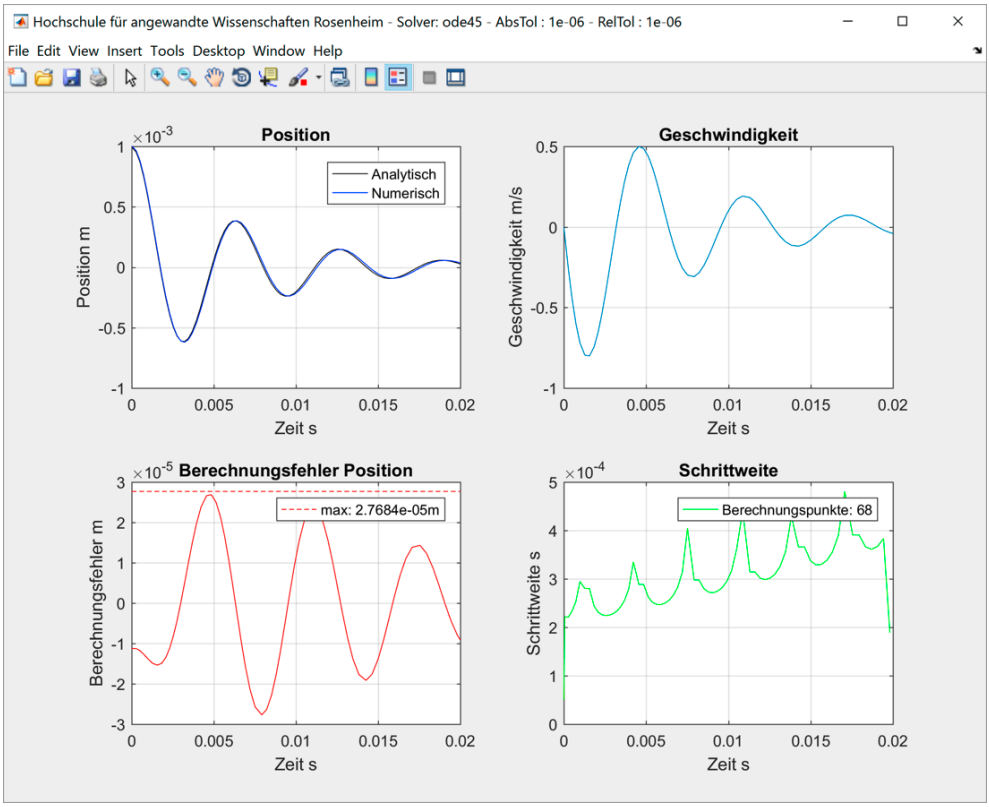


Bild Ü.19 Visualisierung der Berechnungsergebnisse (Variable Schrittweite)

Führen Sie die Übung in den in Tabelle Ü.9 gezeigten Schritten durch.

Tabelle Ü.9 Schrittweises Vorgehen

❶	Öffnen Sie den File „ModellVorlage.slx“ in Simulink und speichern die Datei unmittelbar danach unter dem Namen „ModellEinMasseSchwinger.slx“ ab. Diese Datei ist zu ergänzen. Der Dateiname wird in den vorgefertigten MATLAB Scripts verwendet!
❷	Ergänzen Sie das Simulink Modell um die fehlenden Blöcke und Verbindungen.
❸	Legen Sie die Eigenschaften (Properties) der von Ihnen ergänzten Blöcke fest.
❹	Setzen Sie in Configuration Parameters die Parameter auf die in Bild Ü.20 gezeigten Werte. Starten Sie anschließend das Simulink Modell. Überprüfen Sie die Richtigkeit der Berechnungsergebnisse.
❺	Benutzen Sie zur automatischen Berechnung und Visualisierung das MATLAB Script „EinMasseSchwinger.m“. Die Auswertungsergebnisse werden automatisch im Command Window angezeigt (Bild Ü.21).
❻	Untersuchen und vergleichen Sie die Eigenschaften numerischer Integrationsverfahren. Ergänzen Sie Tabelle Ü.10 mit Werten für die Anzahl an Berechnungspunkten und den maximalen Berechnungsfehler. Die Auswerteergebnisse werden automatisch im Command Window ausgegeben (Bild Ü.21). Benutzen Sie für Berechnungen mit konstanter Schrittweite das MATLAB Script „EinMasseSchwinger.m“. Für Berechnungen mit variabler Schrittweite ist das MATLAB Script „VariableStep.m“ vorgesehen.

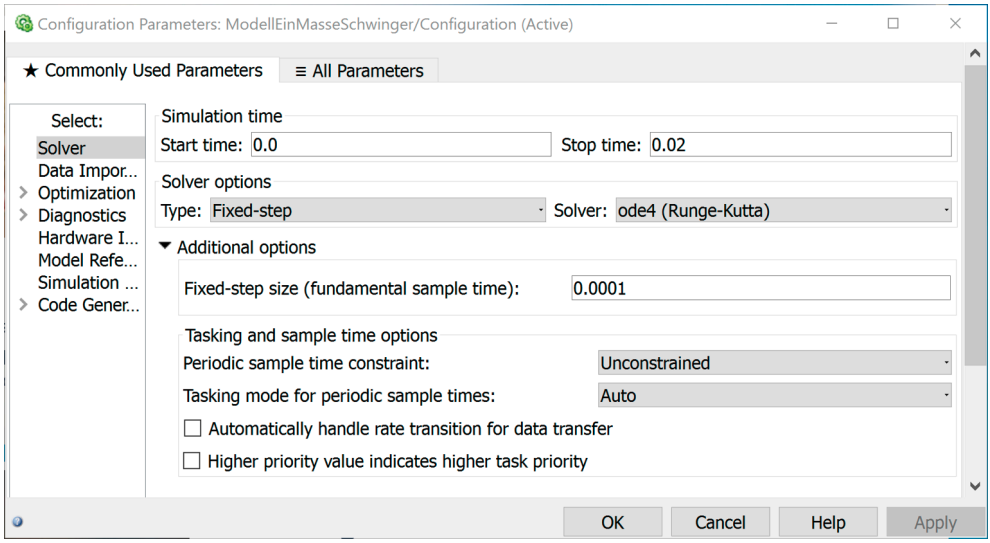


Bild Ü.20 Configuration Parameters

```
-------------------------------------------------------
A u s w e r t u n g s e r g e b n i s s e
Solver: ode4 - fixed step size
Schrittweite: 0.0001s
Berechnungsfehler: 2.7691e-05m
Berechnungspunkte: 201
-------------------------------------------------------
```

Bild Ü.21 Anzeige der Auswertungsergebnisse im Command Window

Tabelle Ü.10 Untersuchung numerischer Integrationsverfahren

	Runge-Kutta vierter Ordnung	Euler	Runge-Kutta vierter Ordnung	
MATLAB solver	ode4	ode1	ode45	
Schrittweite	0.0001 konstant	0.0001 konstant	- variabel	s
AbsTol	-	-	10^{-6}	
RelTol	-	-	10^{-6}	
Berechnungspunkte				
Maximaler Berechnungsfehler				m

A.3 Lösungen der Übung 5

5.1 Wandlung Dezimalzahl in Binärzahl

a) Lösungsweg

Faktorisierung der Zahl 54

	Wert		Summe	Rest
0	Vorzeichen			
0	·64	= 0	0	54
1	·32	= 32	32	22
1	·16	= 16	48	6
0	·8	= 0	48	6
1	·4	= 4	52	2
1	·2	= 2	54	0
0	·1	= 0	54	0

Lösung

00110110

b) Lösungsweg

Faktorisierung der Zahl

	Wert		Summe	Rest
0	Vorzeichen			
0	·64	= 0	0	20
0	·32	= 0	0	20
1	·16	= 16	16	4
0	·8	= 0	16	4
1	·4	= 4	20	0
0	·2	= 0	0	0
0	·1	= 0	0	0

Konvertierung in negative Zahl (Darstellung im Zweierkomplement)

	MSB							LSB
Zahl	0	0	0	1	0	1	0	0
Negieren	1	1	1	0	1	0	1	1
+1	0	0	0	0	0	0	0	1
Zahl − 20	1	1	1	0	1	1	0	0

Lösung

11101100

5.2 Wandlung Binärzahl in Dezimalzahl

a) Lösungsweg

	Wert		Summe
0	·64	= 0	0
1	·32	= 32	32
0	·16	= 0	32
0	·8	= 0	32
0	·4	= 0	32
1	·2	= 2	34
0	·1	= 0	34

Lösung

34

b) Lösungsweg

Zahl ist negativ! Konvertierung in eine positive Zahl.

	MSB						LSB
	1	0	0	0	0	1	0	0
Negieren	0	1	1	1	1	0	1	1
+1	0	0	0	0	0	0	0	1
Positive Zahl	0	1	1	1	1	1	0	0

	Wert		Summe
1	·64	= 64	64
1	·32	= 32	96
1	·16	= 16	112
1	·8	= 8	120
1	·4	= 4	124
0	·2	= 0	124
0	·1	= 0	124

Lösung −124

Nur zur Kontrolle!

	MSB						LSB
Positive Zahl	0	1	1	1	1	1	0	0
Negieren	1	0	0	0	0	0	1	1
$+1$	0	0	0	0	0	0	0	1
Negative Zahl	1	0	0	0	0	1	0	0

5.3 Wandlung Dezimalzahl in Hexadezimalzahl

a) Faktorisierung

		Wert		Summe	Rest
$16^3 = 4096$	0	$\cdot 4096$	$= 0$	0	3119
$16^2 = 256$	12	$\cdot 256$	$= 3072$	3072	47
$16^1 = 16$	2	$\cdot 16$	$= 32$	3104	15
$16^0 = 1$	15	$\cdot 1$	$= 15$	3119	0

Umwandlung in Hexadezimaldarstellung

	Hexadezimal
$0 \rightarrow$	0
$12 \rightarrow$	C
$2 \rightarrow$	2
$15 \rightarrow$	F

Lösung

$0C2F_{hex}$

b) Lösungsweg

Wandlung in Binärzahl (Zweierkomplement) siehe Aufgabe 5.1 b)

	MSB						LSB
Zweierkomplement Zahl -20	1	1	1	0	1	1	0	0
	$1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0$				$1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0$			
	$1 \cdot 8 + 1 \cdot 4 + 1 \cdot 2 + 0 \cdot 1 = 14$				$1 \cdot 8 + 1 \cdot 4 + 0 \cdot 2 + 0 \cdot 1 = 12$			
Hexadezimalzahl	E				C			

Lösung

EC_{hex}

5.4 Wandlung von Hexadezimalzahlen

a) Lösungsweg

Hexa-dezimalzahl	5				A					Dezimalzahl
	$5 \cdot 16^1$				$10 \cdot 16^0$				→	$5 \cdot 16 + 10$
	$0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$				$1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0$					90
Binärzahl	0	1	0	1	1	1	0	0		
	MSB							LSB		

Lösung

Binärzahl: 01011010 Dezimalzahl: 90

b) Lösungsweg

Hexadezimalzahl	F				1			
Binärzahl (Zweierkomplement)	1	1	1	1	0	0	0	1
	MSB							LSB

Vorzeichenbit (MSB) ist 1 → Zahl ist negativ! Konvertierung in eine positive Zahl.

	MSB							LSB
Binärzahl (Zweierkomplement)	1	1	1	1	0	0	0	1
Negieren	0	0	0	0	1	1	1	0
$+1$	0	0	0	0	0	0	0	1
Positive Zahl	0	0	0	0	1	1	1	1

	Wert		Summe
0	$\cdot 64$	$= 0$	0
0	$\cdot 32$	$= 0$	0
0	$\cdot 16$	$= 0$	0
1	$\cdot 8$	$= 8$	8
1	$\cdot 4$	$= 4$	12
1	$\cdot 2$	$= 2$	14
1	$\cdot 1$	$= 1$	15

Lösung

-15

5.5 Wertebereich

a) Lösungsweg

$-2^{12-1} \ldots (2^{12-1}-1)$

Lösung

$-2048 \ldots 2047$

b) Lösungsweg

$-2^{15-1} \ldots (2^{15-1}-1) = -16\,384 \ldots 16\,383$

Lösung

15 *Bit*

5.6 Byteangaben

a) Lösungsweg

64 *Bit*/(8*Bit*/*Byte*)

Lösung

8 Byte

b) Lösungsweg

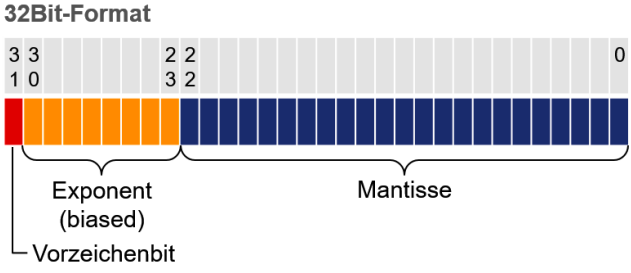
1 *MByte*/(8 *Byte*/*Zahl*) = (1 000 000 *Byte*)/(8 *Byte*/*Zahl*)

Lösung

125 000 64 Bit-Zahlen

5.7 Gleitkommazahlen

a) Lösungsweg



3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0									
0	1	0	0	0	0	0	1	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Vorzeichenbit	S	$=$	0
Vorzeichen		$=$	1
„Biased exponent“	E	$= 1 \cdot 128 + 1 \cdot 2$	130
Exponent	$e = E - B$	$= 130 - 127$	3
Mantisse	M	$= \frac{1}{2} + \frac{1}{4} + \frac{0}{8} + \frac{1}{16}$	
		$= 0{,}5 + 0{,}25 + 0 + 0{,}0625$	0,8125
	$m = 1.M$	$=$	1,8125

$$x = s \cdot m \cdot 2^e = 1{,}8125 \cdot 2^3 = 1{,}8125 \cdot 8 = 14{,}5$$

Lösung

$x = 14{,}5$

b) Lösungsweg

Normalisierung

$$\frac{18{,}25}{2^4} = \frac{18{,}25}{16} = 1{,}140625 = 1 + 0{,}140625$$

Vorzeichen	s			1
Vorzeichenbit	S	$=$		0
Exponent	e	$=$		4
„Biased exponent“	$E = B + e$	$=$	$127 + 4$	131
Mantisse	M	$=$		0,140625

Faktorisierung „Biased Exponent“ (E)

	Wert		Summe
1	$\cdot 128$	$= 128$	128
0	$\cdot 64$	$= 0$	128
0	$\cdot 32$	$= 0$	128
0	$\cdot 16$	$= 0$	128
0	$\cdot 8$	$= 0$	128
0	$\cdot 4$	$= 0$	128
1	$\cdot 2$	$= 2$	130
1	$\cdot 1$	$= 1$	131

Faktorisierung Mantisse (M)

	Wert		Wert		Summe	Rest
0	$\cdot\frac{1}{2}$	$= 0$	0,5	$= 0$	0	0,140625
0	$\cdot\frac{1}{4}$	$= 0$	0,25	$= 0$	0	0,140625
1	$\cdot\frac{1}{8}$	$= 1$	0,125	$= 0{,}125$	0,125	0,015625
0	$\cdot\frac{1}{16}$	$= 0$	0,0625	$= 0$	0,125	0,015625
0	$\cdot\frac{1}{32}$	$= 0$	0,03125	$= 0$	0,125	0,015625
1	$\cdot\frac{1}{64}$	$= 1$	0,015625	$= 0{,}015625$	0,140625	0

Lösung

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	1	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

5.8 Komplexe Zahlen

a) Elementweise Quadrieren

```
e1 =
      -1.0000 + 0.0000i
       0.0000 + 2.0000i
       0.0000 - 8.0000i
```

b) Real- und Imaginärteil extrahieren

```
e2R =         e2I =
       0             1
       1             1
      -2             2
```

c) Betrag

```
e3 =
       1.0000
       1.4142
       2.8284
```

d) Phasenwinkel von z in Grad

```
e4 =
        90
        45
       135
```

e) Konjugiert komplexe Zahl von z

```
e5 =
       0.0000 - 1.0000i
       1.0000 - 1.0000i
      -2.0000 - 2.0000i
```

f) Elementweise Berechnung

```
e6 =
       1.0000
      -1.0000
      -1.0000
```