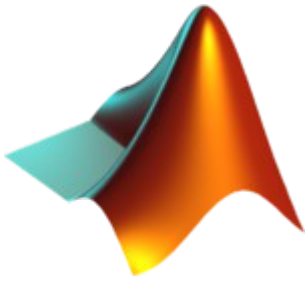
Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

informatik
die zukunft

Seminar „Modellierung und Simulation“



Thema	MATLAB und Simulink
Autor	Anna Fuchs
Datum	31. März 2013

Inhaltsverzeichnis

1. Vorwort

Im Rahmen eines Seminars soll eine Einführung in MATLAB und Simulink gegeben werden. Diese Ausarbeitung ist keine Anleitung zum Selbstlernen, sondern dient der Orientierung. Es soll das Verständnis über die Möglichkeiten und den Nutzen von MATLAB gefördert werden, ohne zuvor tief in die Materie einsteigen zu müssen. Insbesondere geht es um die Fähigkeiten, numerische Probleme zu lösen, wie z.B. gewöhnliche Differentialgleichungen. Auf tiefer gehende Informationen zu einigen Themen wird nach Möglichkeit verwiesen.

2. Eckdaten

MATLAB (Matrix Laboratory) ist eine kommerzielle Software zum Lösen mathematischer Probleme. Den gleichen Namen trägt auch die Programmiersprache, die in der Software verwendet wird. Die Entwicklung von MATLAB geht an das Ende der 70-er Jahre zurück. An der Universität New Mexico hat der damalige Professor der Informatik Cleve Moler[1] den Studenten einen besseren Zugang zu komplexerer Mathematik gewähren wollen, ohne dass die Studenten programmieren können mussten. Speziell ging es um Linpack und EINPACK - FORTRAN Bibliotheken für Lineare Algebra. Dazu hat er damals in FORTRAN den Prototypen vom heutigen MATLAB entwickelt und kostenlos zur Verfügung gestellt. Die Software stieß auf großes Interesse auch an anderen Hochschulen. John N. Little lernte die Software 1983 während eines Besuches in der Stanford University kennen und erkannte sofort das hohe Potential von MATLAB. Kurzerhand hat sich der Ingenieur mit Moler und einem weiteren Kollegen Steve Bangert zusammengeschlossen und das Unternehmen The MathWorks gegründet. Gleichzeitig wurde MATLAB in C umgeschrieben und war lange eher unter dem Namen JACPACK bekannt. Ab sofort wurde das Paket nur noch mit kommerzieller Lizenz verbreitet. MATLAB hat großen Einsatz in linearer Algebra und besonders bei numerischen Berechnungen erlangt. Und noch heute ist MATLAB eines der besten und am weitesten verbreiteten Werkzeuge zum Lösen mathematischer, vor allem numerischer Probleme. Über 1 Mio. Wissenschaftler weltweit benutzen MATLAB.
Die aktuelle Version 8.0 vom 11.09.2012 ist in C und Java geschrieben. Zu den unterstützen Betriebssystemen zählen Windows, Mac OS, Linux, Unix, aber seit 2010 nicht mehr Solaris. Das Interface gibt es in englischer (und nach unsicheren Quellen in japanischer) Sprache.

3. Einsatzgebiet

Die Stärke von MATLAB liegt ganz klar in den vielfältigen Möglichkeiten, numerische Probleme zu lösen. Es sind jedoch auch symbolische Berechnungen möglich (siehe Unterschied in Kapitel 4). Die Umgebung bietet außerdem eine optimale Spielwiese für Algorithmenentwicklung. Mit MATLAB kann man mühelos Daten aus Datenbanken, Tabellen anderer Software (z.B. Excel) einlesen. Auch größere, automatisiert erfasste Datenmengen (z.B. von einer Lichtschranke) lassen sich direkt erfassen und verarbeiten. Für die grafische Visualisierung der Ergebnisse bietet MATLAB ausgereifte Grafik- und spezielle Statistikfunktionen. Vielerlei technische und mathematische Probleme lassen sich modellieren und simulieren. Darüber hinaus ermöglicht MATLAB das Programmieren benutzerfreundlicher Bedienoberflächen auch für Nutzer mit weniger Computerkenntnissen. Zu den Themengebieten zählen

- Strömungsmechanik
- Signalverarbeitung
- Bild- und Videoverarbeitung
- Steuerungs- und Regelungssysteme
- Finanzmathematik
- Bioinformatik
- Testen und Validieren im Allgemeinen

1 http://de.wikipedia.org/wiki/Cleve_Moler

4. Einführung in MATLAB

MATLAB besteht aus zwei grundlegenden Komponenten, dem Kern und den Erweiterungspaketen.
Der Kern der Software ist nicht änderbar, sowie der Quellcode nicht einsehbar. Die Hauptbestandteile sind dabei die gleichnamige Programmiersprache, die Arbeitsumgebung, das Grafiksystem, die vordefinierte Funktionsbibliothek und die Programmierschnittstelle zu C, FORTRAN und weiteren Sprachen.
Die Erweiterungspakete werden Toolboxes genannt. Der Quellcode ist einsehbar, änderbar und erweiterbar, da eine Toolbox aus Dateien vom MATLAB Format (.m) bestehen und als gewöhnliche Programme mit freigegebenen Quellcode behandelt werden können. Beim Arbeiten werden die Pakete nur nach Bedarf geladen, damit das System nicht unnötig belastet wird. Es gibt über 50 definierte Pakete. Die meisten davon sind unter kommerzieller Lizenz, wie die beiden populärsten Pakete Simulink und Stateflow (siehe Kapitel 5) erhältlich.
In diesem Kapitel geht es um die Einführung in den MATLAB Kern.
Die Arbeitsweise mit MATLAB gestaltet sich entweder interaktiv oder als gewöhnliches Programmieren.

4.1 Arbeitsumgebung

Nach dem Start sieht man den Desktop von MATLAB, eine integrierte Arbeitsumgebung. Für gewöhnlich findet man Fenster wie in Abb. 4.1. Diese lassen sich aber flexibel einstellen, verschieben oder gar verdecken. Das aktuelle Layout lässt sich im Menü *Desktop/ Save Layout* speichern.

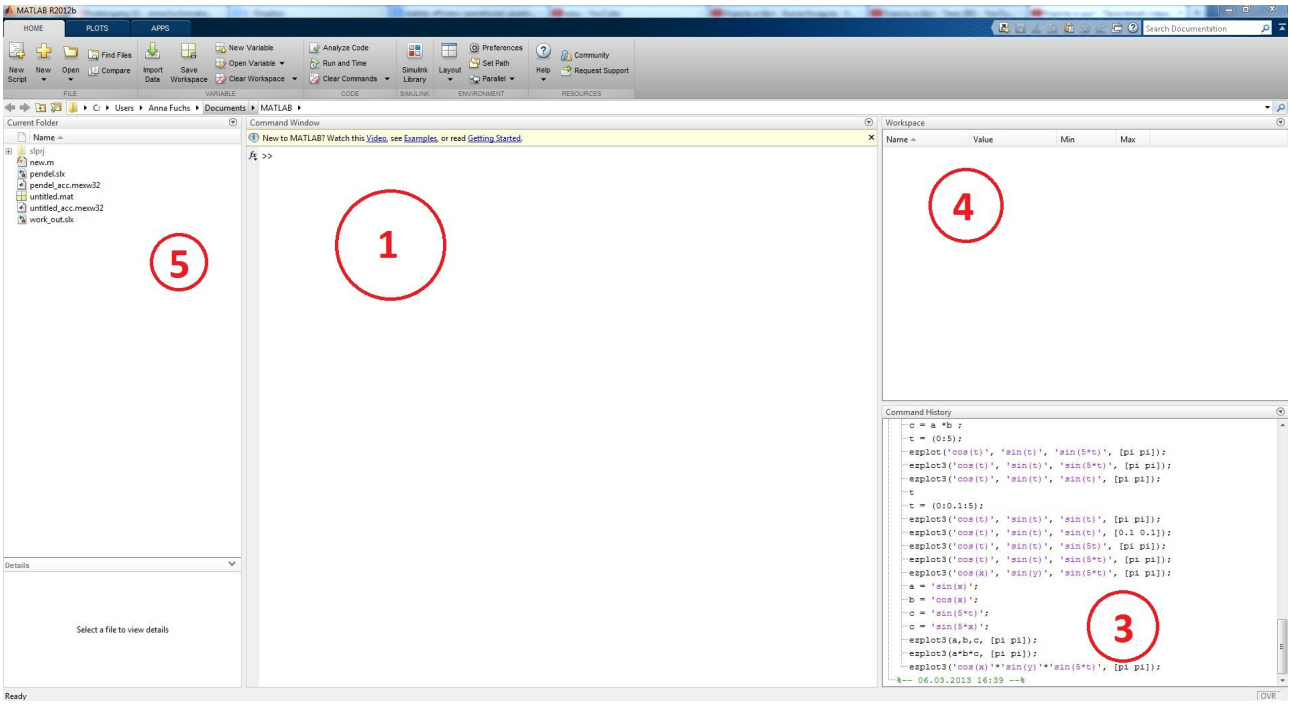


Abb. 4.1 Arbeitsumgebung von MATLAB

1. *Command Window*
   ist der Kern von MATLAB. Hier erfolgen die Eingaben in das *Workspace*. Das Ergebnis der Berechnungen wird ebenfalls in diesem Fenster angezeigt, sofern nicht durch ein Semikolon unterdrückt. Auch Programmaufrufe werden in diesem Fenster getätigt, wie z.B. das Laden der Erweiterungspakete.
   Der Prompt >> bedeutet Eingabebereitschaft.

```
>> 2 + 2
ans =

      4
```

2. *Editor (nicht im Bild)*
   In dem Editor können Programme, Skripte oder einzelne Funktionen geschrieben und editiert werden. Der Editor beinhaltet auch das Hervorheben spezifischer Operatoren und die automatische Syntax Prüfung.

3. *Command History*
   Die bisher eingegebenen Befehle werden in diesem Fenster gespeichert. Aus diesem Verlauf, der sich durchsuchen lässt, können wiederholt benötigte Befehle mit einem Doppelklick ausgeführt werden oder einzelne Befehle ausgeschnitten, herauskopiert oder gelöscht werden. Aus mehreren selektierten Befehlen lassen sich direkt Skripte erstellen. Letzte Befehle können auch durch die Pfeiltasten „oben“ bzw. „unten“ abgerufen werden (ähnlich dem Terminal).

4. *Workspace Browser*
   Alle angelegten Variablen und Objekte werden in diesem Fenster mit Namen und Wert angezeigt. Diese lassen sich löschen, wieder einlesen oder mit einem Variablen-Editor verändern. Im Menü lassen sich weitere Informationen zu den Variablen anzeigen.

5. *Current Directory Browser*
   hilft sich im Dateisystem zurechtzufinden. Das aktuelle Verzeichnis kann für Lade- und Speichervorgänge ausgewählt werden. Dieses Fenster sollte nicht mit dem *„path“* verwechselt werden, in dem angegeben wird, wo sich die Projektdateien von MATLAB befinden (lässt sich in den Einstellungen festlegen).

Weiterhin finden man etwas versteckt den *Profiler*, der die Rechenzeit einzelner Befehle analysiert und die *Shortcut-Leiste*, in der ein oder mehrere Befehle hinterlegt werden können, die jederzeit durch einen Doppelklick ausführbar sind.

Einer der wichtigsten Befehle *help* stellt umfangreiche Hilfe zur Verfügung

```
>> help [befehl]
```

4.2 Interaktiv

Berechnungen, Grafische Darstellungen etc. können im Direktmodus vorgenommen werden. Dazu werden Kommandos im *Command Window* eingegeben.

Nachfolgender Einstieg gibt eine Basis nicht nur zum interaktiven Arbeiten, sondern auch zum Programmieren mit MATLAB.

4.2.a Konstanten

Für ein mathematisches Werkzeug ist es essenziell zu wissen, wie die Zahlen dargestellt und verarbeitet werden.
Gemäß englischsprachiger Konventionen werden Dezimalzahlen mit Punkt notiert, während das Komma

aufeinanderfolgende Anweisungen in derselben Zeile trennt. Die eingegebene Zahl wird der Variablen `ans` zugewiesen, sofern keine manuelle Zuweisung vorgenommen wurde (siehe nächsten Abschnitt).
MATLAB ist eine matrixbasierte Software. Jede Zahl ist somit auch eine 1x1 Matrix. Mit dem Befehl `size()` lässt sich das nachvollziehen. Mit dem Semikolon kann man die Ausgabe im *Command Window* unterdrücken, im *Workspace* Fenster lässt sich die Darstellung aber gut einsehen, wenn `ans` eine 2 zugewiesen wird.
Selbstverständlich gibt es in MATLAB auch die Unterscheidung zwischen reellen und imaginären Zahlen und viele (vordefinierte) Funktionen, die man auf Konstanten anwenden kann (wie Sinus- oder Wurzelfunktion), auf die aber hier nicht weiter eingegangen wird.

4.2.b Variablen

Für komplexere Berechnung kommt man mit Zahlen alleine nicht gut aus. Die Variablen sind unverzichtbar. Die MATLAB eigene Variable `ans` haben wir schon kennen gelernt.
Eine Variable ist ein reservierter Speicherplatz im Hauptspeicher des Computers. Eine Variable kann durch einen eindeutigen Namen angesprochen werden und besitzt für gewöhnlich einen Datentypen.
In MATLAB braucht man von dieser Definition wenig Kenntnis haben. Im Gegensatz zu vielen anderen Programmiersprachen braucht man sich weder um die Speicherreservierung, noch um die Deklarierung der Variablen zu kümmern. Auch den Datentypen braucht man nicht angeben, er wird automatisch bestimmt (meist double). Einer Variablen kann man direkt ohne Vorbereitungen einen Wert zuweisen. Regel der Namensgebung unterscheiden sich nicht von modernen Programmiersprachen (können in der Dokumentation oder Hilfe nachgelesen werden).
Alle verwendeten Variablen erscheinen im *Workspace*, können mit *Save* auf die Festplatte gespeichert, mit *Load* geladen werden. Die Variablen können mit *clear* im *Command Window* oder einzeln im *Workspace* gelöscht werden.
Besonders wichtige Werten wie z.B. die Kreiszahl Pi sind vorbelegte Variablen, die ggf. auch zu Konstanten zählen. Dazu zählen auch die besonderen Zahlenwerte oder Typen wie *inf (infinite)* und *NaN (Not a Number)*. So gibt es eine Definition der Division reeller Zahlen durch 0 (`ans = inf`). Auch die Sonderfälle „0 durch 0“, oder Division zweier unendlicher Werte sind fest definiert. Die vordefinierte Variable `eps` stellt die relative Fließkomma-Genauigkeit dar. Für die Euler-Zahl gibt es keine vordefinierte Variable, sie kann aber mit dem Befehl `exp(1)` erzeugt werden.

4.2.c Vektoren und Matrizen

Wenn es um große Datenmengen geht, so werden sie meist in Matrizen oder Vektoren abgelegt.
MATLAB bietet vielfältige Funktion zum Erzeugen, Verarbeiten und Analysieren von Matrizen.

Die einfachste Art, eine Matrix zu erzeugen, ist direkte Eingabe innerhalb eckiger Klammern [].

```
>> zeile =  [1 2 3]

zeile =

     1     2     3
```

Ein Semikolon zwischen den Einträgen bewirkt die spaltenweise Belegung des Vektors.

```
>> spalte = [1; 2; 3;]

spalte =

     1
     2
     3
```

Mit dem Befehl `size` kann man sich die Größe einer Variablen anzeigen.

```
>> size(spalte)

ans =

     3     1
```

Größere Vektoren, deren Daten strukturiert sind, indem z.B. die Differenz der jeweils benachbarten Elemente konstant ist, lassen sich durch folgende Syntax definieren:

```
>> x = (2:1:10)

x =

     2     3     4     5     6     7     8     9    10
```

Der erste Eintrag in der runden Klammer ist der Startwert, der folgende mit einem Doppelpunkt abgetrennte Wert ist die konstante Differenz, der letzte Wert entsprechend die Grenze.

Matrizen lassen sich ähnlich einfach definieren.

```
>> M = [1 2 3; 4 5 6]

M =

      1     2     3
      3     4     5
```

Da Matrizen aber sehr oft verwendet werden, gibt es eine Reihe an Befehlen, die spezielle Matrizen generieren. Dazu zählt z.B. die Einheitsmatrix.

```
>> X = eye(3)

X =

      1     0     0
      0     1     0
      0     0     1
```

Diese Funktion nimmt dabei bis zu 2 ganzzahlige Werte entgegen, die die Dimensionen der Matrix festlegen. Bei einem Parameter wird eine quadratische Matrix generiert.
Nach gleichen Regeln kann man ganz einfach eine homogene Matrix, die nur aus Einsen besteht, erzeugen. Dazu benötigt man die Funktion `ones(`*`size_value`*`)`, oder entsprechend `zeros(`*`size_value`*`)`, wenn eine Nullmatrix generiert werden soll.
Eine besondere Matrix lässt sich mich dem Befehl `magic(`*`size_value`*`)` generieren.

```
>> M = magic(4)

M =

    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
```

Sollte man nicht verstehen, was eine Funktion tut, kann man sich immer die Dokumentation ansehen.

```
>> help magic
 magic  Magic square.
    magic(N) is an N-by-N matrix constructed from the integers
    1 through N^2 with equal row, column, and diagonal sums.
    Produces valid magic squares for all N > 0 except N = 2.

    Reference page in Help browser
       doc magic
```

MATLAB bietet auch eine Reihe an vordefinierten Funktionen, mit den Matrizen weiterverarbeitet werden können.
So gibt es mehrere Alternativen, um eine Matrix zu transponieren (`flipud(M)`; `fliplr(M)`; `rot90(M)`).
Mit dem Befehl

```
>> v = diag (M);
```

lässt sich die Hauptdiagonale der Matrix M in einem Vektor v erzeugen. Verschiedene Befehle können auch verschachtelt eingesetzt werden. Mit

```
>> s = sum (diag(M));

s =   34
```

berechnet man die Summe der Diagonalen der Matrix M.
Die Summe aller Zeilen kann mit `sum (M, 1)` in einen Spaltenvektor geschrieben werden, die aller Spalten mit `sum (M, 1)` in einen Zeilenvektor. Der zweite Parameter gibt die Nummer der Dimension an, die aufsummiert werden soll. Alternativ gilt auch `sum (M)` für die erste Dimension, oder `sum(M')` für die erste Dimension der transponierten Matrix M, also die zweite Dimension der ursprünglichen Matrix M.
Mit `tril(M)` oder `triu(M)` lässt die die untere, bzw. obere Dreiecksmatrix in eine Nullmatrix kopieren.

```
>> tril(M)

ans =

    16     0     0     0
     5    11     0     0
     9     7     6     0
     4    14    15     1

>> triu(M)

ans =

    16     2     3    13
     0    11    10     8
     0     0     6    12
     0     0     0     1
```

Einige Operationen auf Elemente eines Vektors oder einer Matrix müssen gekennzeichnet werden. Im interaktiven Modus wird man sofort auf syntaktische Fehler aufmerksam gemacht. So z.B. beim Versuch die Elemente des Vektors x zu quadrieren und dem Vektor y zuzuweisen:

```
>> x = (2:1:10)

x =

     2     3     4     5     6     7     8     9    10
```

```
>> y = x^2
Error using  ^
Inputs must be a scalar and a square matrix.
To compute elementwise POWER, use POWER (.^) instead.
```

Das Quadrieren eines Vektors ist als Operation nicht definiert. Was wir brauchen ist das elementweise Quadrieren. Hierfür ist die *dot*-Operation nötig:

```
>> y = x.^2

y =

     4     9    16    25    36    49    64    81   100
```

Andere Operationen wie das Addieren eines Skalars oder Multiplizieren mit einem Skalar funktionieren wie gewohnt.

Für Polynomfunktionen und ihre Analyse bietet MATLAB sehr bequeme Funktionen an. Viele dieser Funktionen nehmen einen Koeffizienten-Vektor entgegen. Der Vektor muss mit dem Koeffizienten der höchsten Ordnung beginnen. Die Funktion `roots(coeff_vec)` bestimmt die Nullstellen der Funktion.

```
>> p = [-2 -3 4];
>> roots(p)

ans =

   -2.3508
    0.8508
```

Sind dagegen nur Nullstellen bekannt, berechnet die Funktion `poly(roots_vec)` die Koeffizienten des Polynoms, der diese Nullstellen hat.

```
n =

     2     3

>> poly(n)

ans =

     1    -5     6
```

Man möchte oft eine Wertetabelle zu einer Polynomfunktion erstellen. Hierzu braucht man einen Vektor, in dem die Intervalle und der Schritt angegeben werden. An diesen Stellen möchte man den Funktionswert berechnen und ggf. in einen weiteren Vektor schreiben. Dafür ist eine Funktion `polyval(coeff_vec, intervall_vec)` definiert.

```
>> p = [1 -5 6]

p =

     1    -5     6

>> x = [0:0.5:2]

x =

         0    0.5000    1.0000    1.5000    2.0000
```

```
>> y = polyval (p,x)

y =

    6.0000    3.7500    2.0000    0.7500         0
```

MATLAB bietet eine ganze Reihe mehr Funktionen zum Definieren und Verarbeiten von Vektoren und Matrizen. An der Stelle wurde die grundlegende Art und Weise erklärt, wie diese Funktionen aufgebaut sind.

4.3 Symbolisch vs. numerisch

MATLAB ist in erster Linie für Numerische Berechnungen ausgelegt, kann aber auch symbolische. In diesem Abschnitt wird beispielhaft der Unterschied gezeigt.

Numerische Berechnungen sind immer nur näherungsweise Brechungen. Daher können diese auch nur auf konkreten Werten in konkreten Intervallen vorgenommen werden. Das Prinzip lässt sich sehr gut am Beispiel des Differenzierens verstehen.

Man definiere einen Vektor a

```
>> a = (0:1:5)

a =

     0     1     2     3     4     5
```

und einen Vektor b

```
>> b = a.^2

b =

     0     1     4     9    16    25
```

Mit dem Befehlt `diff(x)` kann man den Differentialquotienten berechnen.

```
>> d = diff(b)

d =

     1     3     5     7     9
```

Man erhält einen Vektor, der um ein Element kürzer ist, als Vektor b. Der neue Vektor enthält die Differentialquotienten der jeweils benachbarten Elemente des Vektors b (Nach der Regel [f(x1) – f(x2) / (x1 – x2)] → (9 – 4)/(4-3) = 5).

Wendet man die Funktion auf Symbole an, so erhält man die allgemein Ableitung der symbolischen Funktion, also den Grenzwert des Differentialquotienten. Hierzu muss man Gebrauch von der Symbolic Math Toolbox machen. Mit dem Befehl

```
>> syms x y
```

definiert man zwei symbolische Variablem x und y. Weiter definiert man y als eine Funktion auf x:

```
>> y = x^2

y =

x^2
```

Hier Braucht man keinen dot-Operator, da es sich um Symbole handelt. Die Funktion diff(v, u) [differenziere y nach x] liefert nun wie erwartet die Ableitung der Funktion y.

```
>> diff (y,x)

ans =

2*x
```

Für symbolische Berechnungen braucht man weder konkrete Wert für y und x, noch einen Intervall, in dem differenziert werden soll. Das Ergebnis einer symbolischen Berechnung ist auch ein symbolischer Ausdruck, das einer numerischen Berechnung dagegen ist immer ein konkreter Wert.

4.4 Programmieren in MATLAB

Es ist relativ unübersichtlich und schlecht handhabbar, alle Anweisungen im Command Window durchzuführen, da auch das Editieren nicht immer einfach ist. Es können .M-Files erstellt werden, in den Befehle gespeichert werden. In diesem Abschnitt wird nur sehr kurz auf das Programmieren in MATLAB eingegangen.
In MATLAB gibt es Funktionen und Skripte, die man zu ganzen Programmen zusammenfassen kann. Diese können in der Sprache MATLAB geschrieben werden, oder auch unter Benutzung von Sprachkopplern oder entsprechenden Toolboxes in anderen Sprechen wie C (siehe mehr zu Kapitel xy). Es gibt auch die Möglichkeit, Programme mit Hilfe eine grafischen Oberfläche zu schreiben. Dazu muss man eine GUI definieren[2]. Dieses Verfahren wird hier nicht ausführlich beschrieben.

In MATLAB gibt es wie in vielen anderen Programmiersprachen Kontrollstrukturen und Schleifen.

4.4.a Funktionen

MATLAB Funktionen unterschieden sich in der Semantik nicht von den Funktionen anderer Sprachen. Die Funktionen haben Ein- und Ausgabeparameter. Sie können in Dateien im Format .m abgelegt werden. Eine Funktion ist nach folgendem Muster aufgebaut:

function ergebnisparameter_liste = name_der_funktion (eingabeparameter_liste)

    Kommandos...

Mehr über Funktionen kann in der Dokumentation nachgelesen werden.

4.4.b Skripte

MATLAB Skripte sind Kommandofolgen und können aus Programmen oder interaktivem Modus über den Namen aufgerufen werden. Der Unterschied zu den Funktionen besteht darin, dass es keine Ein- und Ausgabeparameter gibt.

2 http://www.mathworks.de/videos/creating-a-gui-with-guide-68979.html

4.5. Differentialgleichungen

MATLAB ist nicht die beste Software für symbolische Berechnungen, aber eine der besten für numerische. Das Herz der numerischen Berechnungen sind Differentialgleichungen (DGL). Nicht alle DGL lassen sich überhaupt analytisch (symbolisch) lösen. In MATLAB ist kein symbolisches Lösen von DGL vorgesehen, auch nicht von den, die sich grundsätzlich analytisch lösen lassen.
Beispiel:

$$x' = x$$

Die Frage ist also, welche Funktion ergibt abgeleitet die Funktion selbst. Ohne tiefe mathematische Kenntnisse erkennt man, dass die Lösung eine einfache Exponentialfunktion ist. Dies wäre eine analytische Lösung, die MATLAB ohne Hinzuziehen von weiteren Toolboxes nicht unterstützt.
Umso besser sind in MATLAB aber numerische Lösungsverfahren möglich. Numerisches Lösen von DGL ist an feste Regeln gebunden.

1. Es darf keine Integrationskonstante vorkommen
   Das bekannte +C, das man beim symbolischen Integrieren nicht vergessen darf, darf hier nicht vorkommen. Das hat zur Folge, dass das Intervall angegeben werden muss. Beim Einsetzen der Grenzen würde die Integrationskonstante wegfallen.

2. Es darf keine Formvariable vorkommen.
   *3*x² + **a***x + 4* ist nicht möglich
3. MATLAB kann keine Differentialgleichungen höherer Ordnung als 1 lösen. Es heißt, es dürfen nur maximal die Funktion und ihre Ableitung im Ausdruck vorkommen. Alle Gleichungen anderer Art müssen zu Gleichungssystemen 1. Ordnung umgeschrieben werden.

MATLAB bietet viele Lösungsverfahren an. Der Standard ist unter dem Funktionsnamen ode45 bekannt und bedeutet das Verfahren nach Runge-Kutta mit Formelpaar der Konvergenzordnung 4 und 5 [3]. Je nach Ausgangslage wählt man ein anderes Verfahren. ode45 ist ein Befehl, der einen Verweis auf eine Funktion, das Intervall und die Anfangsbedingung entgegen nimmt. Hierzu muss man eine einfache Funktion definieren, die die rechte Seite der Gleichung darstellt.

new.m

```
function dy = blub(x,y)      %name
dy = y;                      %rechte Seite der gleichung
```

Andere Verfahren setzten ggf. eine andere Form der Funktion voraus.
Zum Lösen der DGL muss in MATLAB folgendes eingegeben werden:

```
>> [x,y] = ode45('new', [0,3],1);
```

Nun kann man sich ansehen, welche Werte berechnet wurden.

```
>> plot(x,y)
```

Es ist nicht schwer zu erkennen, dass der Graph dem der e-Funktion entspricht. Als Anfangsbedingung lässt sich auch ein Vektor übergeben.

```
>> [x,y] = ode45('new', [0,3],[1:5]);
>> plot(x,y)
```

3 http://de.wikipedia.org/wiki/Runge-Kutta-Verfahren

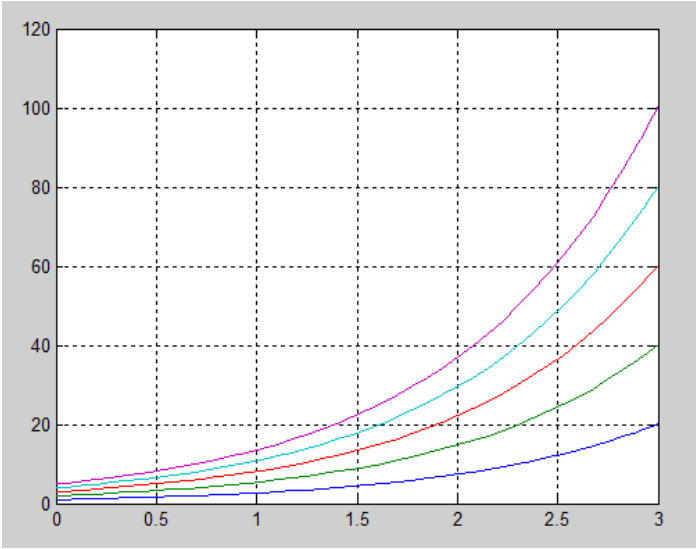


Abb. 4.5.1 Graphen der e-Funktion mit verschiedenen Anfangsbedingungen

Damit kann man sehr bequem untersuchen, wie sich die Anfangsbedingung auf das Ergebnis auswirkt. Ausführlichere Anleitung zu grafischen Funktionen folgt im nächsten Abschnitt.

4.6 Grafische Ausgabe

MATLAB bietet sehr viele Möglichkeiten sehr einfach vielfältige 2- und 3-dimensionale Graphen und Diagramme zu erstellen.
Mit dem Befehl

```
>> figure(1)
```

erstellt man zunächst ein grafisches Fenster. Die Eingabezahl ist dabei nur ein Index des Fensters. Ohne dieses Befehls würde man die Grafiken immer im selben Fenster generieren und die alten somit überschreiben.
plot(x_vec,y_vec) ist einer der einfachsten Befehle der Visualisierung. Im Menü von *figure* lassen sich viele Einstellungen vornehmen, die die Achsen formatieren oder den Graphen beschriften lassen. Anders gibt es die definierten Befehle, wie z.B.

```
>> grid on
```

um das Gitternetz im Grafikfenster zu aktivieren oder

```
>> title ('Beispiel-Titel')
```

um den Graphen zu beschriften. Mit

```
>> text(x_value, y_value,' \leftarrow Beispieltext')
```

lassen sich bestimmte Punkte beschriften.

```
>> x = [-2:0.1:2];
>> y = x.^2;
>> plot(x,y)
>> title('x^2');
>> text(0,0, '\leftarrow Nullstelle');
```

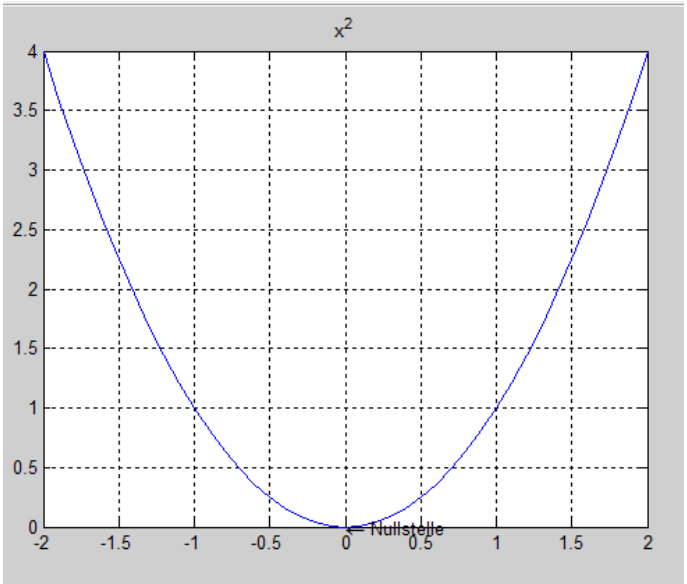


Abb. 4.6.1 Eine Normalparabel mit gekennzeichneter Nullstelle

Die Einstellungen gelten nur für das aktuelle Grafikfenster. Daher lohnt es sich, die Befehle in einer .m Datei abzuspeichern, die z.B. eine Figur als Parameter annimmt und bestimmte Grundeinstellungen in einem Funktionsaufruf vornimmt.

Einige Befehle kennt man nicht auswendig oder kann weiß nicht mehr genau welche Art Graphen sie erzeugen. Eine Alternative dazu bietet das interaktive Plotten. Man generiert Datenvektoren, die im *Workspace* angezeigt werden. Wählt man eine oder mehrere Variablen und wechselt in die PLOTS Ansicht, erscheint die Vorschau an möglichen Plot-Funktionen, die auf die ausgewählten Datenstrukturen angewendet werden können.

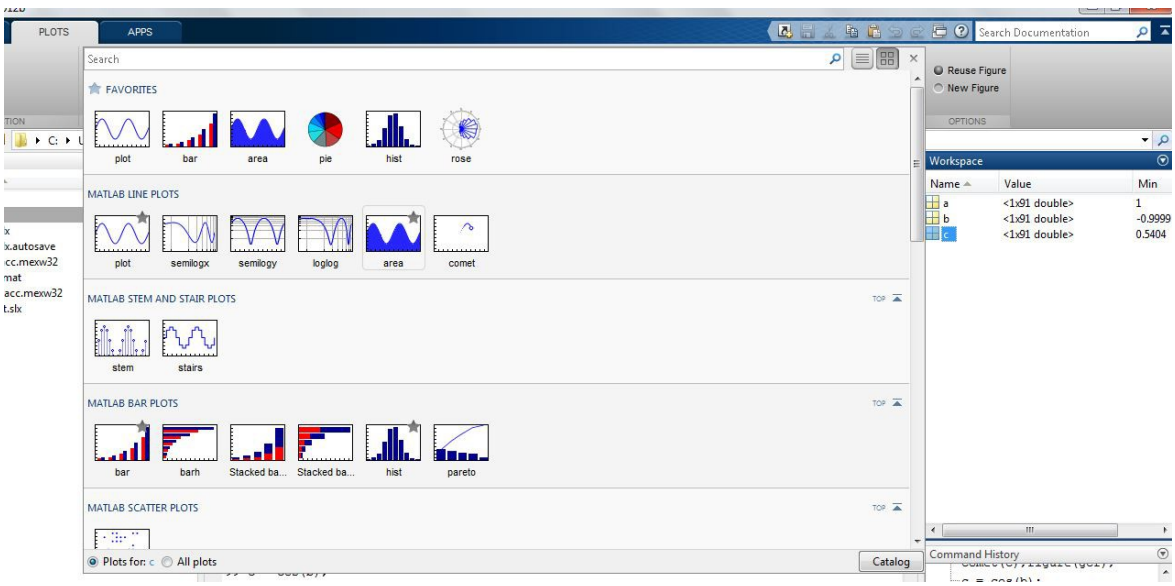


Abb. 4.6.2 Übersicht der grafischen Funktionen

In der Übersicht der vordefinierten Befehle gibt es auch animierte Plots, wie z.B. *comet.* Besonders wichtige Plot-Funktionen kann man mit einem Klick auf den Stern zu Favoriten hinzufügen.

Auch 3-dimensionale Graphen lassen sich problemlos generieren.

```
>> x = (1:10);
>> y = (1:10);
>> ezmesh('sin(x)*cos(y)')
```

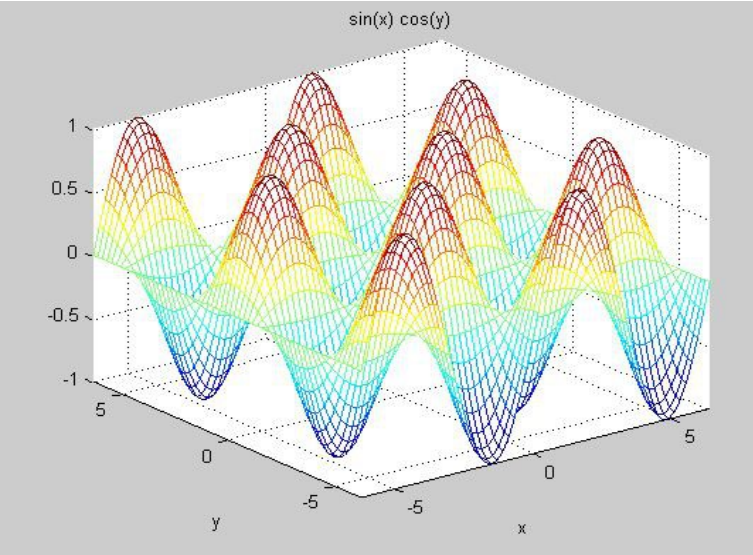


*Abb. 4.6.3 'sin(x)*cos(x)'*

Einige Funktionen nehmen Ausdrücke in ' ' entgegen, die elementweise ausgewertet werden. Ohne die Apostrophe gibt es sonst einen Fehler, da MATLAB versucht ein Skalarprodukt aus den entstehenden Vektor zu bilden.
Viele Funktionen nehmen schon eine Menge Analysearbeit ab, indem markante Bereiche farblich dargestellt und hervorgehoben werden.

5. Einführung in Simulink

Simulink ist eine Toolbox von MATLAB und stellt eine Blockdiagrammumgebung für Entwurf und Simulation dar. Mit Simulink lassen sich zeitgesteuerte, mit Stateflow, der zweitverbreitetsten Toolbox, ereignisgesteuerte Simulationen entwerfen. Der Einsatz von Simulink unterstützt kontinuierliches Testen und Verifizieren von eingebetteten Systemen, Algorithmen und Programmen. Simulink umfasst einen Grafikeditor und Bibliotheken mit vordefinierten Blöcken. Es lassen sich aber auch eigene Blöcke definieren, z.B. direkt in MATLAB, die als gewöhnliche .m Programme nach Simulink importiert werden können.
Simulink steht in ständiger Interaktion mit MATLAB, zudem Simulink sich nur in MATLAB mit
`>> simulink` ausführen lässt und keine eigenständige Software ist. In MATLAB lassen sich die Simulationsergebnisse besser analysieren. Die in MATLAB entwickelten Algorithmen lassen sich nach Simulink exportieren und können dort weiterverwendet werden.

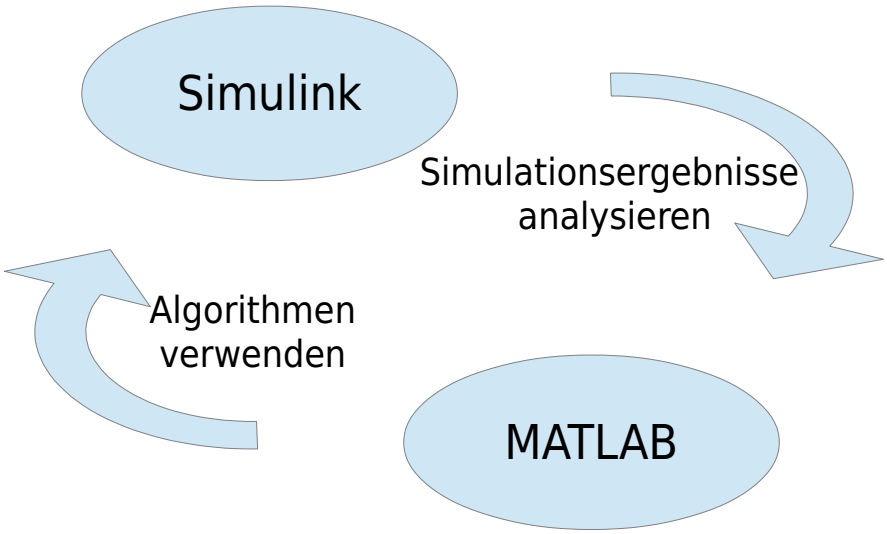


Abb. 5.1 Interaktion von MATLAB und Simulink

Nach dem Aufruf erscheint ein neuen Fenster mit der Auflistung der *Libraries*, einigen Blöcken und einer Menüleiste.

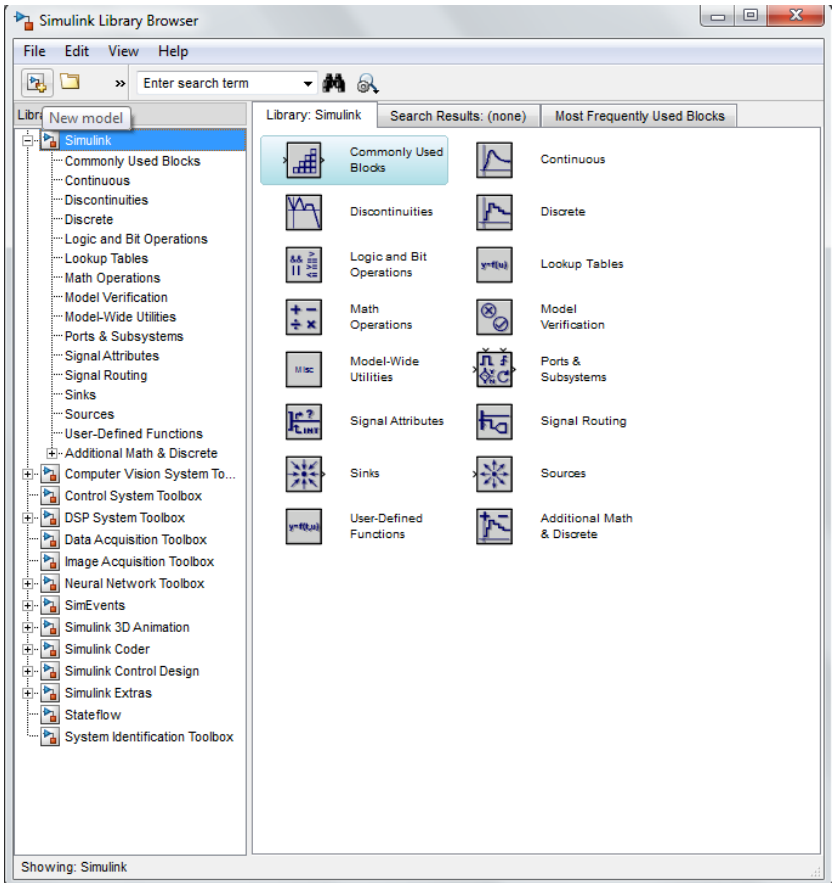


Abb. 5.2 Simulink Arbeitsfläche

In der linken oberen Ecke lässt sich mit dem Button *New model* ein neues Modell erzeugen. Dabei erscheint ein neues Fenster mit einer leeren Arbeitsfläche. In diesen Arbeitsbereich kann man ausgewählte Blöcke direkt herüberziehen. Einige der Blöcke werden hier beispielhaft erläutert.

In der Liste der Bibliotheken erscheinen auch andere Toolboxes, wie z.B. Stateflow, nur dann, wenn diese in dem Paket enthalten sind. Die Studentenlizenz der Universität Hamburg beinhaltet alle auf dem Screenshot aufgeführten Erweiterungspakete. An dieser Stelle ist nur Simulink interessant. In dem Unterverzeichnis sind verschiedene Blöcke nach Gruppen sortiert. Am Ende der Liste findet man die Gruppe *Sources*. In dieser Gruppe sind Blöcke zusammengefasst, die Quell- oder Eingangssignale darstellen. Ein Eingangssignal ist z.B. Sinus. Diesen Block ziehen wir an der Stelle in die Arbeitsfläche. Unter der Überschrift *Sinks* verbergen sich Ausgabeblöcke. Der einfachste Block ist der *Scope*, der einen Eingangsignal hat. Diesen Block ziehen wir auch in die Arbeitsfläche und verbinden ihn mit dem *Sinus-Wave* Block. Die erste Simulation ist damit schon fertig. In der Menüleiste oben sieht man einen grünen *Play-Button*, mit dem sich die Simulation starten lässt. Rechts davon sieht man einen Eingabebereich, in dem, eine 10.0 eingetragen ist, sofern nichts geändert wurde. Diese Zahl bedeutet den Zeitschritt. Drückt man auf den *Play-Button*, wird eine Simulation mit 10.0 Zeitschritten durchgeführt. Ein Doppelklick auf den *Scope-Block* öffnet ein Aufgabefenster mit dem Ergebnis. Zu sehen ist eine Sinuswelle. Die Anzeigeparameter lassen sich in dem Fenster anpassen. *Autoscale* passt die Koordinaten dem Ergebnis an.

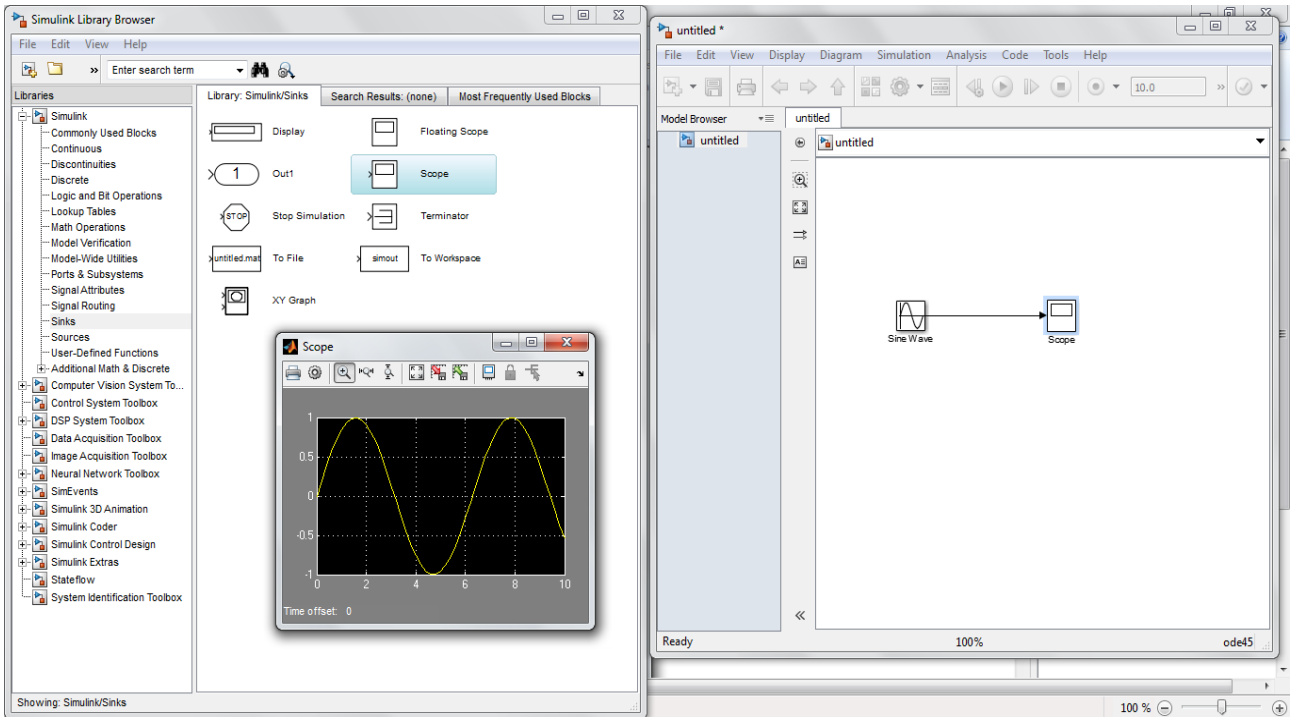


Abb. 5.3 Sinussignal in Simulink

Ein Doppelklick auf dem Sinus-Block öffnet ebenfalls ein neuen Fenster, in dem viele Einstellungen vorgenommen werden können, wie z.B. Amplitude, Frequenz, Phase und vieles mehr.

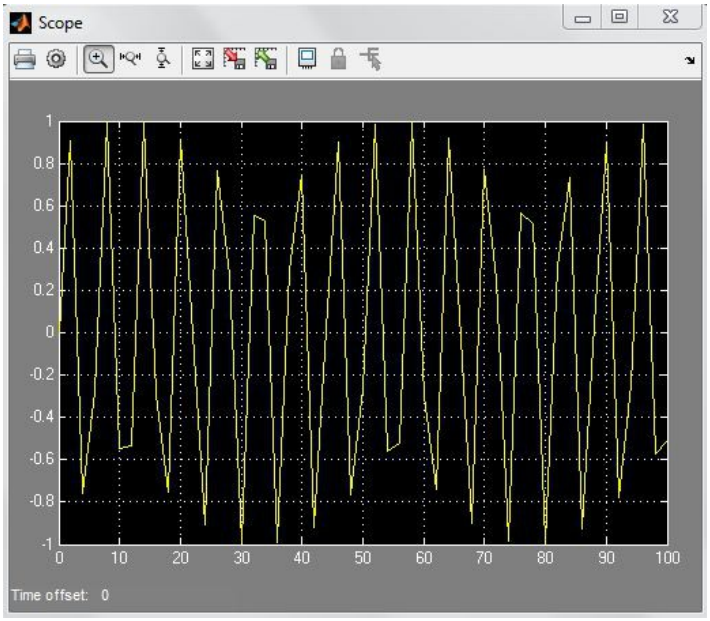


Abb 5.4 Refine Factor = 1

Ändert man das Zeitintervall von 10.0 auf 100.0 und lässt die Simulation laufen, kommt ein unerwartet ungenauer zackiger Graph, der eine Sinuskurve erahnen lässt.

Das liegt an der Diskretisierung. Für einen Graphen ist eine bestimmte Anzahl an Werten festgelegt, die gemessen werden, z.B. 100, die auf den ganzen Graphen verteilt werden. Bei längeren Simulationen reichen die Werte nicht mehr aus, um ein hinreichend übergangsfreien Verlauf zu simulieren. Neben dem Play-Button befindet sich das Zahnrad Symbol, in dem Simulationseinstellungen vorgenommen werden können. Unter der Überschrift Data Import/Export findet man eine Option *Refine factor*, die normalerweise den Wert 1 hat. Ein höherer Wert verfeinert die Ausgabe und erhöht die Diskretisierung.

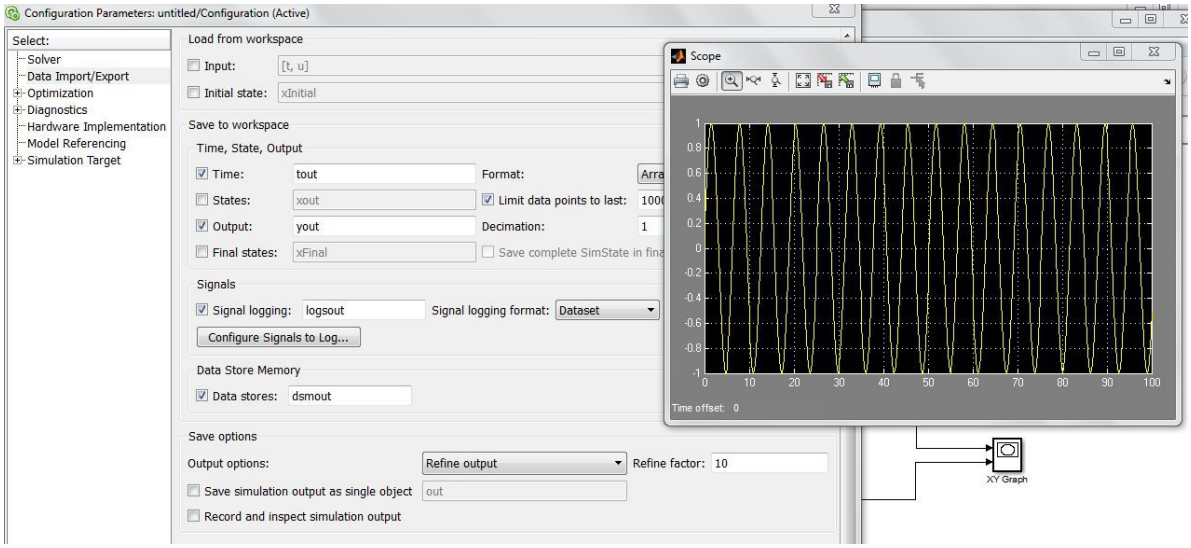


Abb. 5.5 Refince Facotr = 10

Es können selbstverständlich mehrere Signaleingänge miteinander verknüpft werden. Dabei gibt es die Möglichkeit, die Signale z.B. arithmetisch zu verknüpfen. Addiert man zwei Sinussignale mit einer leicht unterschiedlichen Frequenz, ergibt sich eine Schwebung[4]. Wenn man nicht mehr genau weiß, in welcher Kategorie sich der Summen-Block befindet, kann man nach den Schlüsselworten im Such-Fenster suchen (z.B. sum). Dieser Block lässt sich im Menü (→ Doppelklick) auf mehrere Summanden erweitern.

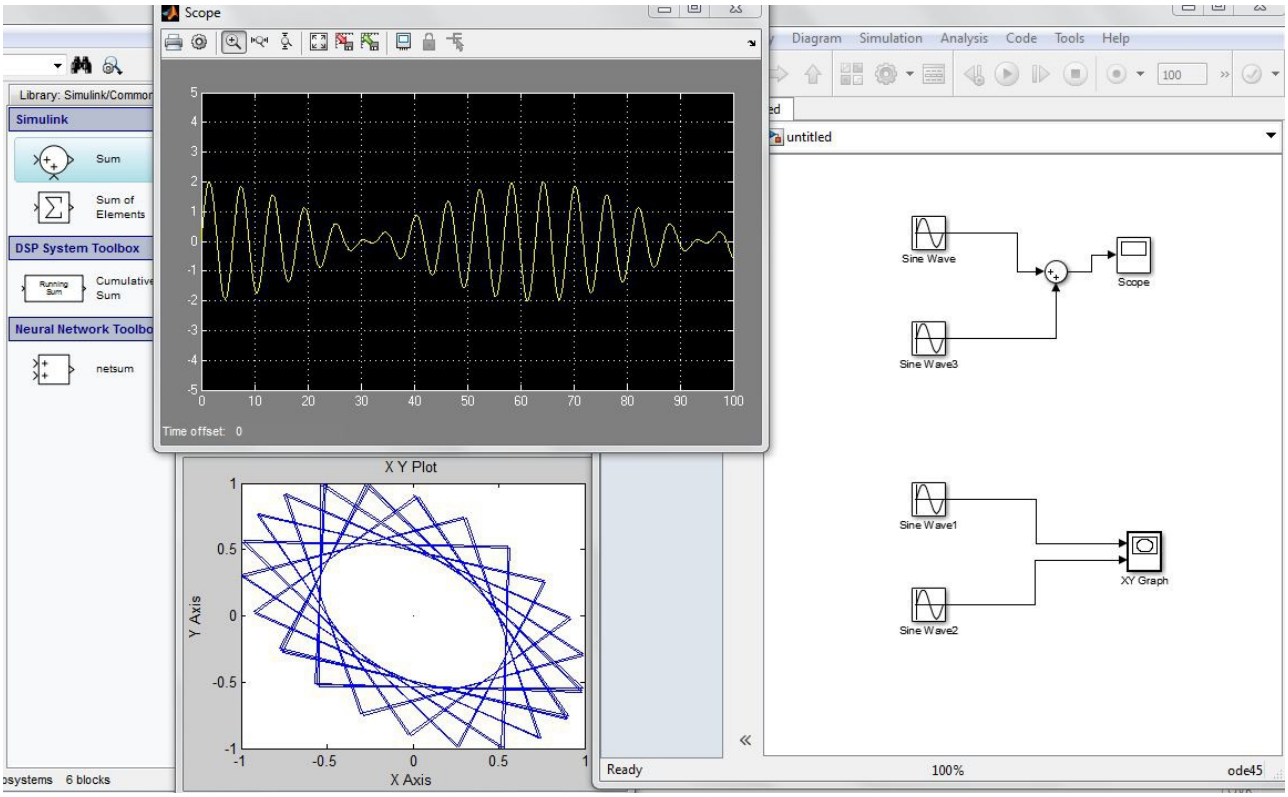


Abb 5.6. Verknüpfung von Sinussignalen

Es gibt einen *Scope* mit zwei Eingängen. Mit diesem Block lassen sich zwei verschiedene Signale verknüpfen, die jeweils die x- und y-Achse darstellen, wobei der Zeitschritt für beide die Variable darstellt.

Signale lassen sich durch weitere Blöcke auch modifizieren. Dazu müssen die Blöcke Ein- und Ausgänge haben, wie z.B. *Zero-Order-Hold*. All die unzähligen Optionen und Blöcke werden nicht weiter im Detail behandelt.
Sehr wichtig zu zeigen ist aber die Art Differentialgleichungen in Simulink umzusetzen. Im Gegensatz zu MATLAB können in Simulink DGLs beliebiger Ordnung gelöst werden. Ansonsten gelten dieselben Voraussetzungen, wie im Kapitel 4.4.a beschrieben. Dazu wird der *Integrator*-Block benötigt. Dieser hat einen Ein- und einen Ausgang. Der Trick hierbei ist, dass der Eingang des Blocks gleichzeitig sein Eingabesignal ist. Der Wert der Funktion wird zum Integrieren eingegeben, herauskommen soll dasselbe. An die Schleife lässt sich eine Verbindung anhängen, die zu einem *Scope* führt. In den Optionen des Integrators lässt sich die Anfangsbedingung festlegen. In diesem Feld kann ähnlich wie in MATLAB auch ein Vektor übergeben werden.

4 http://de.wikipedia.org/wiki/Schwebung

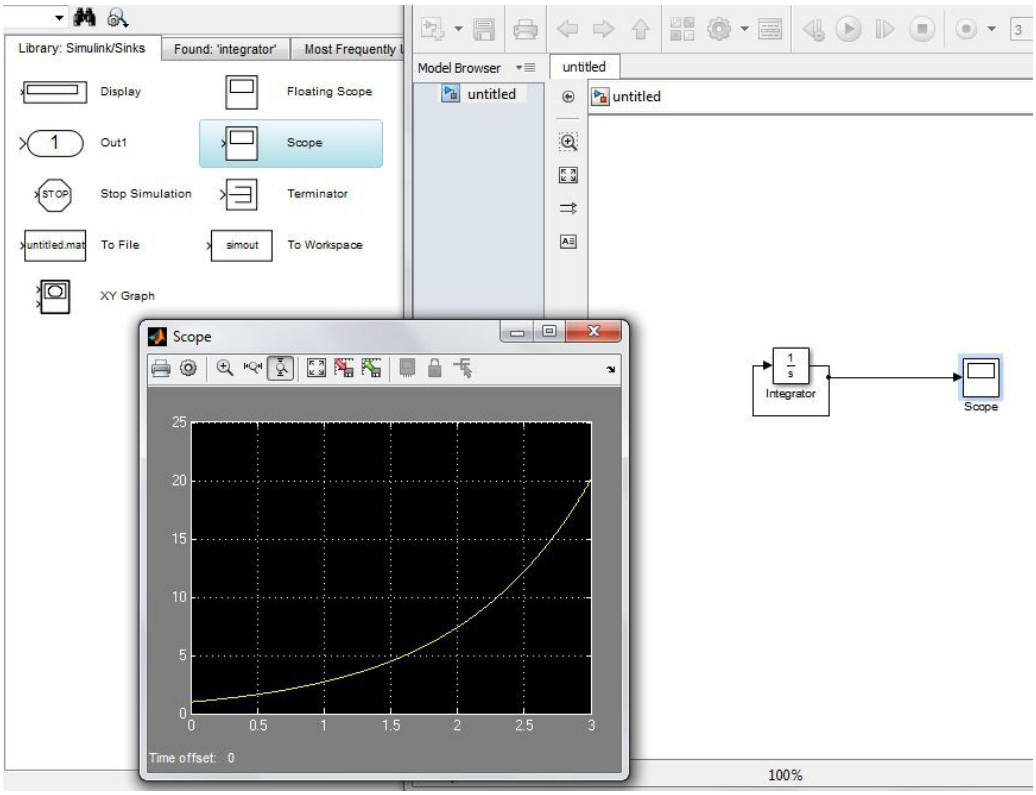


Abb. 5.7 Differentialgleichung in Simulink

Mir der Anfangsbedingung 1 und dem Intervall [0,3] erhält man wie erwartet den Graphen der e-Funktion. Mehrere Verschachtlungen der Integratoren bedeuteten einen jeweils höheren Grad der DGL.

Ein einfaches Beispiel aus der Schulphysik soll in MATLAB umgesetzt werden. Es geht um eine freie gedämpfte Schwingung eines Federpendels. Die allgemeine Gleichung lautet

$$\ddot{x} = \frac{-k}{m} \cdot x - \frac{b}{m} \cdot \dot{x} + g$$

b	Dämpferkonstante
k	Federkonstante
g	Gravitationskonstante
m	Masse
x	Örtliche Auslenkung

Für die Gravitationskonstante benötigt man den Block *Constant*, der in dem Kontext einen unveränderlichen Wert hält, sofern man sich auf der Erde befindet und tolerant gegenüber kleineren Ungenauigkeiten ist. k und b werden jeweils mit einem *Slider Gain* Block dargestellt. Dieser Block definiert einen Bereich, in dem der Wert liegen kann. Das Intervall ist sehr vom Vorteil, wenn man die Simulation nicht komplett abspielen möchte, sondern schrittweise ausführt. Die Werte in dem Block lassen sich nämlich zur Simulationszeit verändern. Daran kann man direkt erkennen, wie sich die beiden Konstanten auf die Auslenkung auswirken. Die Masse soll einfachheitshalber ein Kilogramm betragen und wird in dieser Simulation vernachlässigt. Die zweite Ableitung nach der Zeit signalisiert, dass wir zwei Integratoren brauchen, die jeweils in einer Schleife miteinander verbunden werden. Ein Summen-Block lässt sich nicht nur auf mehrerer Summanden erweitern, sondern lässt natürlich auch negative Vorzeichen zu.

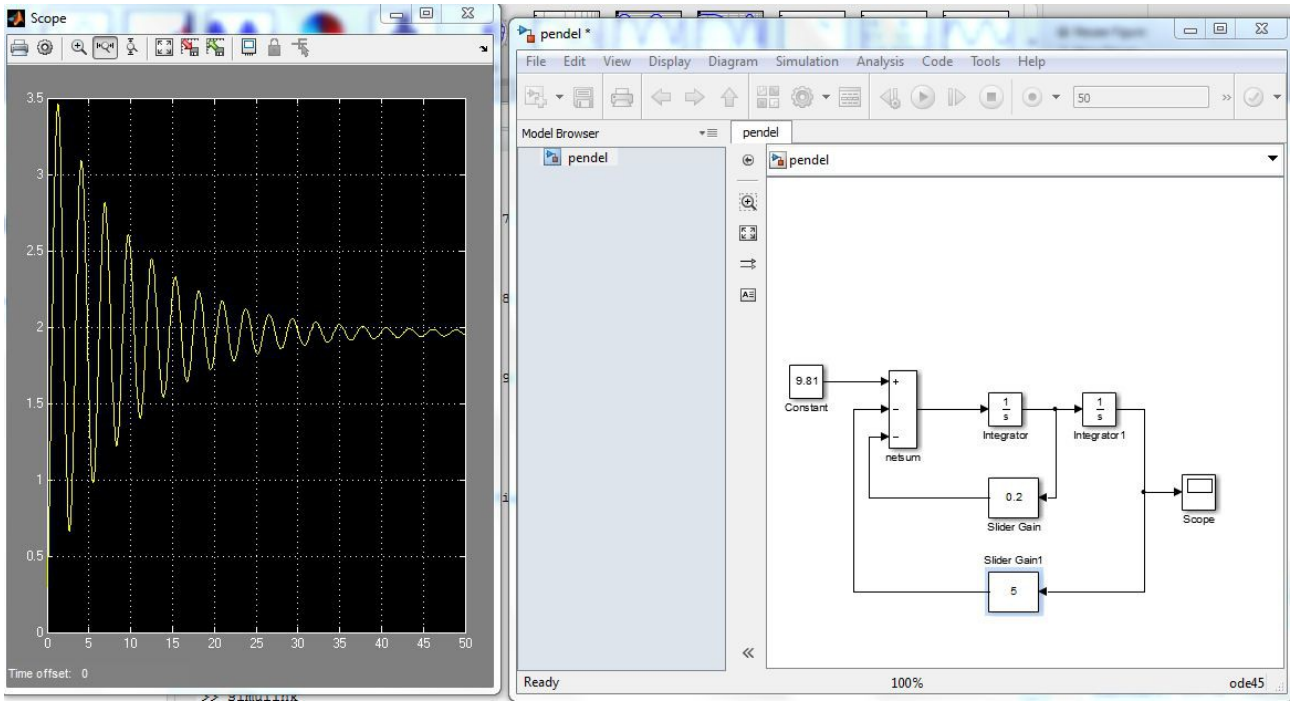


Abb. 5.8 Simulation eines Federpendels in Simulink

Es ist nicht sehr einfach, den Ablauf der Integratoren zu verstehen. Mit Erfahrung, nachdem man selbst einige simple Beispiele umgesetzt hat, entwickelt man ein Gefühl für die Funktion einzelner Blöcke und ihres Zusammenspiels.

Es lassen sich auch sehr viel komplexere Zusammenhänge simulieren. Dabei verliert man schnell der Überblick. Einzelne Blöcke können umbenannt werden, indem man den Schriftzug von z.B. *Constant* anklickt und den Text „Gravitationskonstante“ eintippt. Eine ganze Blockgruppe kann auch zu einem Modul zusammengefasst werden, damit es auf der Arbeitsfläche nicht so unübersichtlich wird. Es gibt auch `fcn`-Blöcke, die entweder weitere Blöcke oder aber eine aus MATLAB importierte Funktion enthalten können.

Die erzeugten Ergebnisse können nach MATLAB überführt werden, um diese besser zu analysieren. Dazu schaltet man einen *Workspace*-Block (mit dem *Workspace* ist das von MATLAB gemeint). Man sollte aufpassen, welchen Typs die Ergebnisse sind, die man nach Workspace umleitet. In den Einstellungen des Blocks unter der Option *Save format ist* normalerweise *Timeseries* eingestellt. Möchte man die Werte in Form einer gewöhnlichen Matrix erhalten, so stellt man die Option auf *Array* um. Nach demselben Prinzip lassen sich auch Daten aus dem Block „*From Workspace*“ aus MALTAB eingeben.

6. Vor- und Nachteile

MATLAB wurde für einfachere Benutzung und Umsetzung der mathematischen Bibliotheken entwickelt. Die Motivation für Simulink auf der Herstellerseite beginnt mit Worten „You are an engineer“[5]. Die Zielgruppe des Produktes sind Ingenieure, die in erster Linie bemüht sind, ihre Systeme möglichst korrekt und einfach umzusetzen. Es geht also um das Lösen der „Engineering“- Probleme, nicht um das „Coden oder Debuggen“. Dies bleibt dem Benutzer tatsächlich weitgehend erspart. Die Frage ist nur zu welchem Preis.

5 http://www.mathworks.de/videos/simulations-made-easy-with-simulink-68945.html?type=large

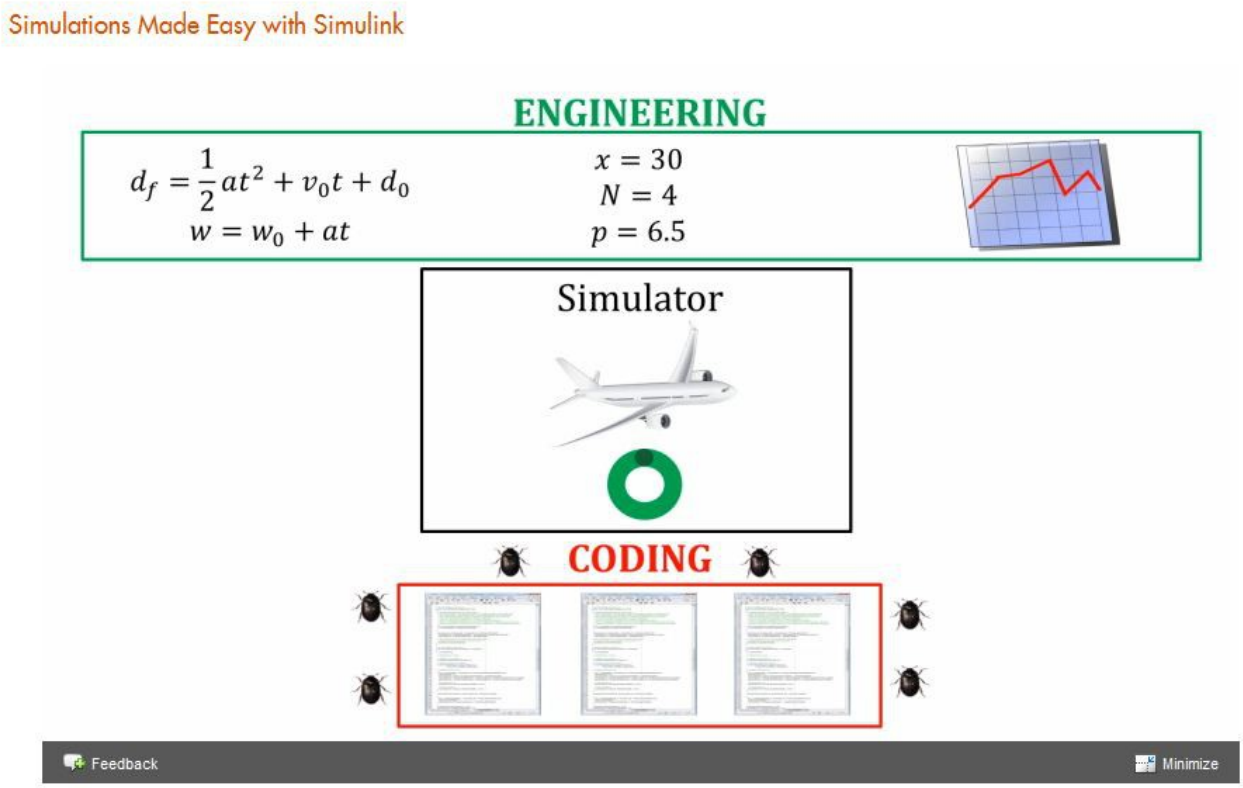


Abb. 6.1. Motivation für Simulink, Quelle: [1]

Die Vor- bzw. Nachteile werden hier relativ zu anderen Sprachen aufgeführt. Es ist allerdings schwierig eine Software mit integrierter Sprache mit anderen, klassischen Programmiersprachen zu vergleichen. Dennoch werden als Kriterium des Vergleichs oft der Aufwand ein Programm zu schreiben und die Effizienz des fertigen Programms genannt. In den folgenden Abschnitten werden u.A. diese Punkte aufgegriffen. Der Vergleich mit anderen mathematischen Paketen wie MAPLE, MuPAD etc. findet im Kapitel 7 statt.

6.1 Vorteile

Die Bedienung von MATLAB ist intuitiv und einfach. Man braucht sich nicht um MATLAB-spezifische Komponenten zu kümmern wie z.B. Deklarierung von Variablem oder Speicherverwaltung. Die Informatik-Komponente wird weitgehend klein gehalten, was für die Zielgruppe nur von Vorteil ist. Trotzdem hat MATLAB eine sehr hohe Kompatibilität. Das gilt auch für die Erweiterungspakete. Es kann Code aus vielen anderen Sprachen wie C, C++, FORTRAN, Java, ActiveX oder .NET eingebunden werden. Aus Simlunk-Simulationen lässt sich sogar Code generieren (mittels einer weiteren Toolbox). Auch für Mikrokontroller gibt es weite Unterstützung, so kann der Code hier direkt eingebunden oder erzeugt werden. Aber auch wenn es um Eingabedaten geht, erweist sich MATLAB als sehr flexibel. Es können ganze Excel-Tabellen, SQL-Datenbanken oder Daten direkt von einem Gerät wie z.B. einer Lichtschranke eingelesen werden. Auch die Interaktion mit den Konkurrenten oder Partnern (siehe Kapitel 7 „Alternativen“) ist möglich. Ohne jegliche Änderungen können MATLAB Programme auf diversen Plattformen ausgeführt werden. Die Kompatibilität und Flexibilität von MATLAB ist sehr ausgeprägt.
Für numerische Anwendungen bietet MATLAB ungeahnte Möglichkeiten. Das materialorientiertes Konzept unterstützt den Benutzer in seinem Vorhaben, man vermisst kaum eine Funktion.
Als Analysewerkzeug bietet MATLAB hervorragende grafische 2-, 3-D, animierte und statistische Funktionen. Auch die Auflistung mit einer Vorschau erleichtert die Arbeit sehr. Einige Funktionen beinhalten schon Analyseschritte und helfen somit die Ergebnisse auszuwerten.
Dank der guten Qualität hat das System einen hohen Marktanteil und wird gut und lange gepflegt. Daher gilt es als sehr zuverlässig.

6.2 Nachteile

In erster Linie ist das Paket nicht frei verfügbar. Dies muss nicht immer ein Nachteil sein, da gewisse Qualität ihren Preis hat, jedoch ist die freie Verfügbarkeit beschränkt.
Der größte Kritikpunkt an MATLAB ist aber die Effizienz. Fast die gesamte Funktionalität ist auf numerische Probleme ausgerichtet. Andere Problemtypen lassen sich nur schwierig bewältigen oder man muss in Vielerlei Hinsicht umdenken. Die Ausführung der Schleifen ist viel langsamer als Arbeit mit Vektoren. Folgendes Beispiel zeigt den Unterschied:

```
>> x = 1:10

x =
     1     2     3     4     5

>>y = sqrt(x);
```

```
for x = 1:5

y = sqrt(x);

end
```

Da MATLAB relativ „hardware-fern“ ist, gibt es kaum Möglichkeiten zur Optimierung im Gegensatz zu z.B. C (Compiler, Speicherverwaltung). Um das beste Ergebnis zu erreichen muss man, wie im oberen Beispiel, ganz genau verstehen, wie MATLAB intern funktioniert. Von einer Zielgruppe, die aber gerade dem Hintergrund der Informatik fern bleiben möchte, ist es schwierig. Die Optimierung der Mathematik in Form von unterschiedlichen Algorithmen muss ebenfalls an die matrixorientierte Arbeitsweise von MATLAB angepasst werden.

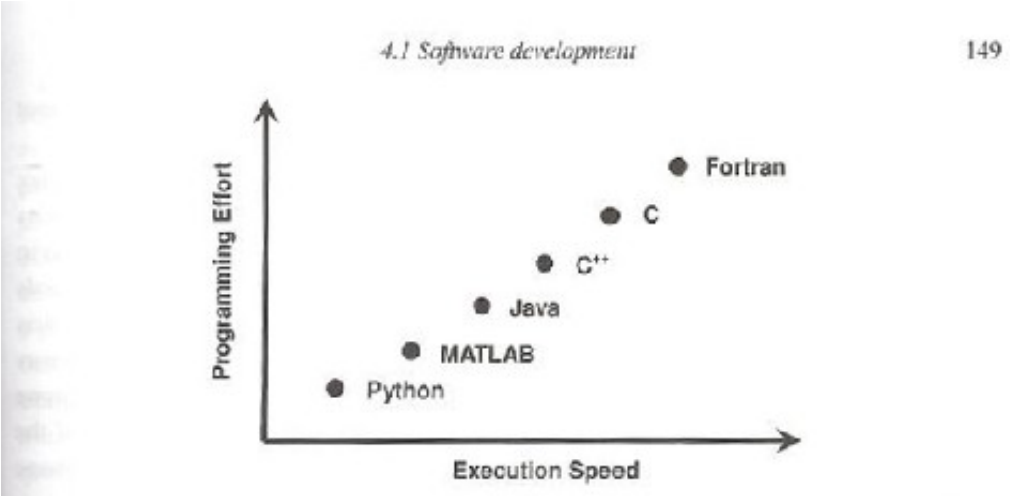


Abb. 6.2 Qualitatives Beispiel für Programmieraufwand gegen Programmeffizienz (Wilson, 2009), Quelle: [2]

Die Bequemlichkeit im Umgang mit MATLAB hat einen hohen Preis. Man überlässt einerseits die Verantwortung für den Speicher und die Datentypen dem System, wird andererseits aber der Kontrolle darüber entbunden. MATLAB ist eine Interpretersprache, die im Gegensatz Assembler oder Compiler die Befehle nicht direkt in eine auf dem System ausführbare Datei übersetzt, sondern erst einliest, analysiert und dann ausführt. Dies erfolgt nicht wie beim Compiler vor der Ausführung, sondern zur Laufzeit und ist mit entsprechend viel Aufwand verbunden. In der Abbildung 6.2 ist ein qualitatives Beispiel für den Vergleich von vielen Sprachen bezüglich des Verhältnisses von Aufwand und Effizienz angeführt. MATLAB findet sich relativ weit unten wieder.

7. Alternativen

MATLAB ist natürlich nicht der Alleinherrscher auf dem Markt der Mathematiksoftware.
Zu den kommerziellen Konkurrenten oder auch Partnern im Bereich der symbolischen Berechnungen zählen u.A. MAPLE, MATHEMATICA und MuPAD. Sie alle sind überragend auf ihrem Gebiet. MATLAB kann hier nicht mithalten, und beinhaltet derzeit die Mindestausstattung von MuPAD (früher war MAPLE Teil der symbolischen Bibliothek von MATLAB).
Auf dem Gebiet der numerischen Probleme bietet MATLAB hingegen mit das beste Paket. Lediglich in Ingenieurmathematik dürfte MATHCAD leichte Fortschritte haben, da mehr fachspezifische Funktionen vordefiniert sind. Dieser geringe Nachteil wird jedoch durch die Vielfalt der Erweiterungspakete ausgeglichen.
Mit GNU Octave[6], Scilab[7] und FreeMat[8] ist auch das Angebot an OpenSource Software gedeckt.

8. Fazit

MATLAB ist eine sehr interessante und vielfältige Software, die einen intuitiven und leichteren Zugang zu komplexerer Mathematik gewährt.
Die Möglichkeiten und Facetten von MATLAB und Simulink sind so umfangreich, dass schon viele Bücher verfasst und sogar ganze Vorlesungen darüber gehalten werden. In dieser Arbeit wurde lediglich ein Einstieg gewährt, der aber einen ausreichenden Überblick geben sollte, wie viel mehr mit der Software möglich ist. Darüber hinaus sollte es klarer geworden sein, welche Bereiche und Zielgruppen das Produkt in erster Linie anspricht und aus welcher Motivation heraus es entwickelt wurde. Es ist ein Paket für Anwender der numerischen Mathematik, für Ingenieure, aber nicht primär für Informatiker. Die grafischen Möglichkeiten sprechen jedoch eine sehr breite Gruppe an, da sich mit diesen beliebige Daten aus allen Bereichen analysieren lassen. Mit der Vielzahl an Erweiterungspaketen werden auch spezifischere Probleme abgedeckt. Die Flexibilität von MATLAB erstaunt. Für ganz spezielle Probleme bleibt noch immer die Möglichkeit eigene Pakete zu definieren oder andere Programme und Bibliotheken einzubinden.
Alles in allem in MATLAB eine sehr gelungene Software, die ihren Preis an Geld und Effizienz hat.

6 http://www.gnu.org/software/octave/
7 http://www.scilab.org/
8 http://freemat.sourceforge.net/

9. Quellen

Literaturverzeichnis

Angermann A., Beuschel M., Rau M., Wohlfarth U. Matlab-Simulink-Stateflow. *Grundlagen,Toolboxen, Beispiele* (Oldenbourg, 2005)(472s)

Benker H. *Ingenieurmathematik kompakt. Problemlösungen mit MATLAB* (Springer,2010)(273s)

Beucher O. *Wahrscheinlichkeitsrechnung und Statistik mit MATLAB* (Springer, 2007)(535s)

Buskens C. *MATLAB im Selbststudium. Eine Einführung* (Vorlesungsbegleitende Ausarbeitung,2004)(60s)

Gekeler E.W. *Mathematische Methoden zur Mechanik. Ein Handbuch mit MATLAB* (Springer,2006)(624s)

Gunther M., Jungel A. *Finanzderivate mit MATLAB* (Vieweg+Teubner, 2nd ed., 2010)(352s)

Hauser F., Luchko Y. *Mathematische Modellierung mit MATLAB* (Spektrum, 2011)(345s)

Hoffmann J. MATLAB und Simulink. *Beispiel orientierte Einführung in die Simulation dynamischer Systeme* (Addison Wesley, 1998)(505s)

Lowe A. *Chemische Reaktionstechnik mit MATLAB und SIMULINK* (Wiley-WCH,2001)(403s)

Oberkampf W.L., Roy C.J. *Verification and Validation in Scientific Computing* (Cambridge University Press, 2010)

Quarteroni A., Saleri F. *Wissenschaftliches Rechnen mit MATLAB* (Springer, 2006)(269s)

Werner M. *Digitale Signalverarbeitung mit MATLAB* (Vieweg Teubner, 4th ed., 2008)(294s)

Klöden, W. (2009), *Handbuch MATLAB & Simulink zum Gebrauch in den Lehrveranstaltungen des Vertiefungsfachs Verfahrensautomatisierung,* Dresden: Technische Universität Dresden, Fakultät Maschinenwesen, Institut für Verfahrenstechnik & Umwelttechnik

Web-Verzeichnis

The MathWorks, Inc. (2013): http://www.mathworks.de/products/matlab/

Abbildungsverzeichnis

Sofern nichts vermerkt, wurden die Abblidungen eigenhändig erstellt

[1] http://www.mathworks.de/videos/simulations-made-easy-with-simulink-68945.html?type=large

[2] Oberkampf W.L., Roy C.J. *Verification and Validation in Scientific Computing* (Cambridge University Press, 2010)