Ü 1 Erste Übungseinheit

Vier wichtige Regeln:
1. Es funktioniert nie beim ersten Mal.
2. Es wird auch beim zweiten Mal wahrscheinlich nicht funktionieren.
3. Es funktioniert besser, wenn alle Kabel angesteckt sind.
4. Wenn sonst nichts mehr hilft, lesen Sie die Anleitung.

Ü 1.1 MATLAB

Die Übungen beginnen mit einer Einführung in die Rechen- und Programmierumgebung MATLAB (steht abkürzend für „MATrix LABoratorium“). Dieses Programm ist im universitären Bereich und in der industriellen Praxis ein Standardwerkzeug für technisch-wissenschaftliche Berechnungen. Für viele numerischen Aufgaben bietet MATLAB Lösungsfunktionen und Methoden zur Visualisierung. Gleichzeitig ist es eine moderne Programmiersprache, in der Sie eigene Anwendungen entwickeln können.

Diese Übungseinheit soll Ihnen eine rasche Einführung geben.

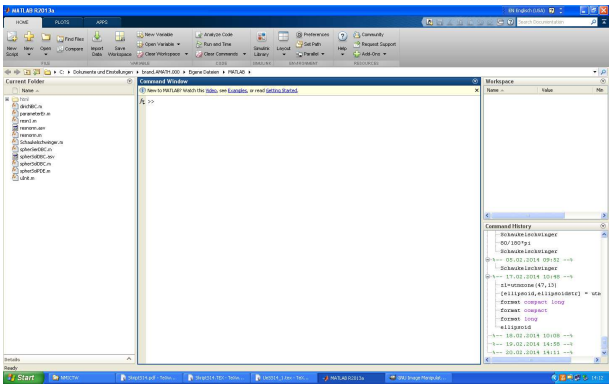
Inzwischen bietet auch die Programmiersprache Python mit ihren Zusatzpaketen (numpy, scipy) nahezu die gleiche Funktionalität wie MATLAB. Alle Übungsaufgaben lassen sich ebensogut in Python lösen. Wenn Sie mit Python vertraut sind, wird Ihnen numerisches Rechnen in MATLAB leicht fallen, und umgekehrt.

Einige weitere kostenlose Programmier- und Rechenumgebungen funktionieren ebenfalls mehr oder weniger ähnlich wie MATLAB: Scilab, Octave, FreeMat, Julia, R. Auch dafür gilt: Wenn Sie mit MATLAB gut vertraut sind, wird Ihnen der Einstieg zu diesen Softwarepaketen leicht fallen (und umgekehrt).

Ü 1.2 Der Anfang

Starten Sie MATLAB auf Ihren Rechnern wie übliche MS-Windows-Anwendungen durch Doppelklick auf das MATLAB-Zeichen. Sobald die MATLAB-Benutzeroberfläche (MATLAB *Desktop*) am Schirm erscheint, sind Sie bereit für die erste Lektion.

Sie sehen die MATLAB-Benutzeroberfläche mit der *Command Window* als großes Fenster in der Mitte.



Wenn die MATLAB-Benutzeroberfläche anders aussieht als hier nebenan abgebildet – es gibt sehr viele Konfigurationsmöglichkeiten – dann stellen Sie zuerst einmal die voreingestellte Standard-Oberfläche wieder her. Finden Sie das *Layout*-Symbol in der Mitte der oberen Leiste und klicken Sie auf *Layout–SELECT LAYOUT–Default*

Probieren Sie, wenn Sie wollen, einige der sonst noch angebotenen Möglichkeiten aus. Damit die Oberfläche wirklich so aussieht, wie hier gezeigt, müssen Sie *Layout–SHOW–Command History–Docked* einstellen.

Aber zum Schluss stellen Sie bitte mit *Default* das Standard-Layot wieder her. Sie sehen dann – von links nach rechts – vier Fenster: *Current Folder, Command Window, Workspace, Command History*. Wir arbeiten erst einmal im *Command Window*. Um die weiteren Fenster kümmern wir uns später.

Das *Command Window* liegt wie ein großes leeres Blatt vor Ihnen. Aber anders als in einem Textverarbeitungsprogramm können Sie nicht einfach irgendwohin schreiben, sondern immer nur in die aktuelle Befehlszeile. Die ist am Schirm durch die Eingabeaufforderung (den *prompt*) `>>` markiert.

Fangen Sie an und geben Sie die folgenden Befehle ein. Was Sie eintippen sollen, ist auch in den Unterlagen durch vorangestelltes `>>` markiert. Darunter folgt die Antwort von MATLAB (das, was Sie auf dem Schirm sehen sollten, nachdem Sie die Eingabetaste gedrückt haben).

```
>> 3-2

ans =

     1
```

Geben Sie 3-2 ein. Das Resultat wird unter dem Standard-Namen „ans“ (wie „eins“, oder „answer“, je nach Muttersprache) gespeichert.

```
>> ans

ans =

     1
```

Sie können das Resultat unter der Bezeichnung „ans“ jederzeit wieder abrufen...

```
>> zwa=ans+ans

zwa =

     2
```

...oder in weiteren Rechenausdrücken verwenden. Sie können den Wert eines Ausdrucks auch einer Variablen zuweisen.

```
>> y=2^2 + log(pi)*sin(zwa);
>> y

y =

    5.0409

>>
```

Sie berechnen $2^2 + \log(\pi) \cdot \sin(2)$. Ein **Strichpunkt** am Ende unterdrückt Ergebnis-Ausgabe und Zeilenvorschub. MATLAB erinnert sich trotzdem an y. Sie können den Wert von y jederzeit abrufen, indem sie einfach „y“ eintippen

```
>> format long g
>> y
y =
          5.04089993961332
>> y^10
ans =
          10594507.4098595
>>
>> format long e
>> y^10
ans =
     1.059450740985953e+07
>> format short g
>> y^10
ans =
   1.0595e+07

>>
```

MATLAB rechnet mit etwa sechzehnstelliger Genauigkeit. In der Standard-Einstellung zeigt es nur vier Nachkommastellen an. Sie können unterschiedliche Anzeigeformate einstellen. Das `e`-Format verwendet immer die Exponentialschreibweise, das `g`-Format schreibt, wenn möglich, das Ergebnis als Gleitkommazahl ohne Exponent.
`format compact` unterdrückt Leerzeilen bei der Ausgabe, damit passt mehr Ausgabe auf eine Bildschirmseite. `format loose` fügt Leerzeilen zur besseren Lesbarkeit ein.

```
>> theta=acos(-1)

theta =

    3.1416
```

MATLAB kennt trigonometrische Funktionen. Das ist der Arcus Cosinus von -1 (**Im Bogenmaß!**)

```
>> help elfun

  Elementary math functions.

  Trigonometric.
    sin         - Sine.
    .
    .
    .
```

MATLAB kennt eine große Menge sogenannter elementarer Funktionen. Sie können so eine Liste verfügbarer Funktionen aufrufen. Zu jeder einzelnen Funktion lässt sich weitere Hilfe anfordern.

Versuchen Sie, über die verschiedenen Möglichkeiten der MATLAB-Hilfe herauszufinden: Wie heißt die Funktion `sinh` mit vollem Namen?

Arbeitsauftrag: Verwenden Sie den *help browser* und zeichnen Sie einen Funktionsgraph Die ausgiebige MATLAB-Hilfe (der *help browser*) gibt mehr Information zu `sinh` und zeigt auch gleich, wie sich die Funktion plotten lässt.

Lassen Sie MATLAB nach der Anleitung in der Hilfe einen Funktionsgraph von `sinh` zeichnen.

Ü 1.2.1 Eingabe wiederholen, frühere Befehle kopieren

Wenn Sie sich irgendwo vertippen und die Eingabe wiederholen müssen, brauchen Sie nicht alles erneut eintippen. Sie können mit der Taste ↑ früher eingegebene Befehle hereinholen und gegebenenfalls korrigieren.

Wenn Sie den oder die ersten Buchstaben eines früheren Befehls eingeben und dann ↑ tippen, erinnert sich MATLAB und vervollständigt jene Befehle, die mit diesen Buchstaben begannen.

Sie können auch aus dem Fenster rechts unten, der *Command History*, Befehle mit Rechtsklick-Copy kopieren und mit Rechtsklick-Paste in das *Command Window* einfügen. Noch schneller geht Rechtsklick-*Evaluate Selection* oder Markieren-F9

Arbeitsauftrag: Zeichnen Sie Ihnen vertrautere Funktionen: Sinus, Cosinus, Exponentialfunktion, nach dem Muster des vorigen Abschnitts. Tippen Sie nicht alles wieder neu ein. Verwenden Sie die Pfeiltaste und/oder die *Command History*, um frühere Befehle herzuholen und nur den Funktionsnamen auszubessern.

Ü 1.2.2 Ausgabe unterdrücken, Schirm reinigen, Notbremse

Geben Sie folgenden Befehl ein:

```
>>  0:10000

ans =

  Columns 1 through 13

    0     1     2     3
  .
  .
  .
       9990        9991

  Columns 9997 through 10001

       9996        9997

>>
```

Dieser Befehl erzeugt eine lange Liste. Er dient hier nur als Beispiel für eine Anweisung, die den Schirm mit Unmengen von Output vollramscht. Solche Befehle sollten Sie einen Strichpunkt abschließen. Dann werden intern die Berechnungen durchgeführt, aber Bildschirmausgabe unterdrückt.

```
>>  0:10000;
>>
```

So ists besser.

Wenn aber doch einmal versehentlich eine Anweisung nicht und nicht enden will: Die Strg C -Taste beendet die laufende MATLAB-Berechnung. Sie können damit auch begonnene Befehlszeilen löschen.

Wenn MATLAB auch auf Strg C nicht reagiert, lässt es sich nur mehr über den *Windows Task Manager* stoppen. Dann ist aber die gesamte Sitzung mit allen bisher berechneten Werten verloren.

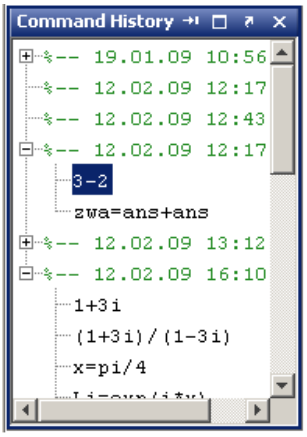
Wenn Sie alle Ein- und Ausgaben im *Command Window* löschen wollen, verwenden Sie den Befehl

```
>> clc
>>
```

Jetzt wird reiner Tisch gemacht

Ü 1.3 Speichern von Befehlen in einer Skript-Datei

Wenn Sie bis jetzt emsig gearbeitet haben, wollen Sie vielleicht auch Ihr Werk abspeichern. Das Fenster links unten, die *Command History*, hat alle Befehle soweit getreulich bewahrt. (Wenn Sie dieses Fenster nicht sehen, müssen Sie *Layout–SHOW–Command History–Docked* einstellen!)



Scrollen Sie durch die Befehle der *Command History*. Sie sehen alle von Ihnen eingegebenen Befehle. Wenn Sie weiter in die Vergangenheit zurückscrollen, finden Sie – durch Datum- und Zeitangabe gekennzeichnet – Befehle früherer Sitzungen. Sie lassen sich durch Anklicken öffnen oder schließen.

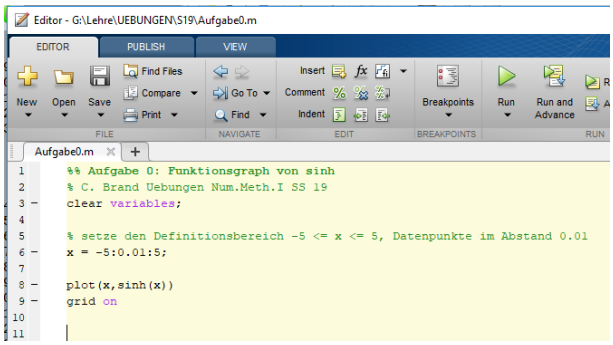
Angenommen, Sie wollen die Befehle der ersten Übungsaufgaben (Funktionsgraph von Sinus hyperbolicus zeichnen) speichern. Gehen Sie so vor:

Markieren Sie (Klick!) den ersten Befehl; halten Sie die Umschalttaste ⇑ gedrückt und scrollen sie in der Befehlsliste nach unten; markieren Sie (Klick!) den letzten Befehl. Die markierten

Befehle sind nun dunkelblau hinterlegt. Recktsklick→*Create Script*. Es öffnet sich ein Fenster des Editors.

Im Editor können Sie – anders als im *Command Window* – Befehle bearbeiten, wie in einem Texteditor. Sie können Zeichen an beliebiger Stelle einfügen und löschen

Empfehlenswert ist, *Kommentare* (beginnen mit %) einzufügen. Ihre Datei könnte dann so aussehen:



Speichern Sie Ihre Datei in der Windows-üblichen Weise (Menü *Save* – *Save as...*) – am besten gleich direkt auf einen mitgebrachten USB-Stick. MATLAB Skript-Dateien bekommen dabei automatisch den Typ `-.m`.

Mit dem Start-Symbol oben in der Menüleiste können Sie ihre Befehle als *Skript-M-Datei* starten. Das wirkt so als würden Sie die Befehle direkt ins *Command Window* eingeben. Die entsprechende Ausgabe erscheint im *Command Window*.

Die Skript-Datei soll aber tatsächlich alle Befehle enthalten, die zur Lösung einer Aufgabe notwendig sind. Typischer Anfängerfehler: das Skript verwendet Variablen, die im Skript nicht definiert sind. Es funktioniert, weil Sie die Variablen vorher definiert haben und daher die Variablen bereits im im Workspace vorhanden sind. Aber wenn Sie oder jemand anderer in einer neuen Matlab-Sitzung das Skript startet, kommt eine Fehlermeldung.

> Stellen Sie sicher, dass alle benötigten Variablen im Skript definiert werden!
> Tipp: Geben Sie ganz zu Beginn des Skripts den Befehl `clear variables`. Damit löschen Sie alle Variablen im Workspace und merken sofort, ob sie Definitionen im Skript vergessen haben

Aufgabe 1: Skript-Datei für Funktionsplot

Suchen Sie sich in MATLAB aus den Bereichen *Trigonometry, Exponents and Logarithms* oder, wenn Sie besonders neugierig sind, *Special Functions* eine Funktion aus, die Sie noch nicht kennen. Erstellen Sie ein Skript, das den Graphen dieser Funktion zeichnet. (Orientieren Sie sich am Screenshot auf Seite Ü-5)

> Tipp: Legen Sie sich für die Matlab-Dateien, die Sie in diesen Übungen erstellen, ein eigenes Unterverzeichnis an.
> Speichern Sie – falls Sie auf Uni-Rechnern arbeiten – am besten auf einem mitgebrachten USB-Datenträger!

Im Editor-Fenster sehen Sie, wenn die Skript-Datei fehlerfrei gespeichert ist, oben Mitte ein grünes „Run“-Pfeil-Symbol. Klick auf „Run“ führt alle Befehle im Skript erneut aus.

(Wenn eine Dialog-Box erscheint mit einem Text der Art *File D:/work/Aufgabe1.m is not found in the current folder blabla blabla*, dann klicken Sie einfach auf „Change Folder“)

Ü 1.4 Funktionen vom Typ $y = f(x)$ zeichnen

In diesem Abschnitt sollen Sie lernen, mit der Anweisung `plot` einfache Funktionsgraphen in der xy-Ebene (Die MATLAB-Hilfe spricht von „linear 2D-plots“) zu erstellen.

Arbeiten Sie im *Command Window* das folgende Beispiel durch. Es sollen die Graphen zweier Funktionen,

$$y = 3\cos x \quad \text{und } z = \log x$$

für den Definitionsbereich $0 < x \leq 25$ gezeichnet werden.

```
>> x=linspace(0.1,25,50)
```

Damit erzeugen Sie einen Zeilenvektor x, der 50 äquidistante Werte im Intervall $[0,1; 25]$ enthält. Entsprechend lang ist der Output am Schirm.

```
x =

  Columns 1 through 7

    0.1000    0.6082
1.1163    1.6245    2.1327
2.6408    3.1490
  .......

  Columns 43 through 49

   21.4429   21.9510   22.4592
22.9673   23.4755   23.9837
24.4918

  Column 50

   25.0000
```

Sie sehen, es werden tatsächlich 50 Werte erzeugt. In MATLABs Sichtweise ist x eine Matrix mit einer Zeile und 50 Spalten (deswegen „Columns“ im Output).

```
>> x=linspace(0.1,25,100);
```

Wiederholen Sie die Eingabe (mithilfe der Pfeiltaste), aber lassen Sie 100 Werte berechnen und fügen Sie einen Strichpunkt zum Abschluss an: Dadurch wird die Ausgabe unterdrückt und der Bildschirm bleibt übersichtlicher.
Übrigens werden, auch wenn Sie die Anzahl der Werte nicht angeben, also den Befehl in der Form `x=linspace(0.1,25);` verwenden, standardmäßig 100 Werte erzeugt.

```
>> y = 3*cos(x);
```

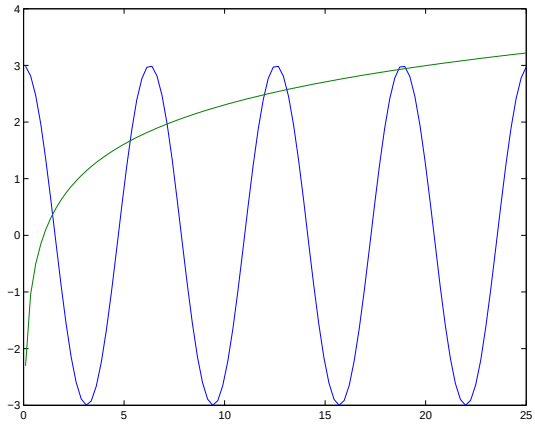
Dieser Befehl berechnet für jede Komponente im Vektor x den entsprechenden y-Wert.

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

Und damit erhalten Sie eine Zeichnung des Funktionsgraphen.

```
>> z=log(x);
>> plot(x,y,x,z)
```

Erzeugen Sie gleich noch einen zweiten Vektor, der den Funktionswerten von $\log x$ entspricht. Sie können beide Funktionsgraphen in ein Schaubild zeichnen. Achten Sie darauf, dass für die zweite Funktion nochmal die Angabe der x-Werte notwendig ist, obwohl beiden Funktionsgraphen die gleichen x-Werte zugrunde liegen.

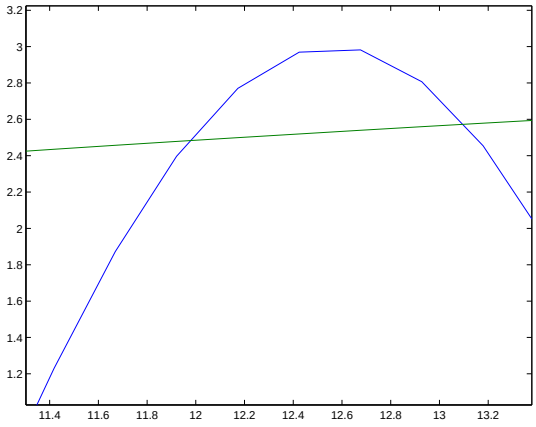


Der Graph sollte der obigen Abbildung entsprechen. Die Schnittpunkte der beiden Kurven entsprechen Lösungen der Gleichung

$$3\cos x = \log x$$

Vergleichen Sie dazu Abbildung 1 und Kapitel 1.4 im Vorlesungsskriptum, wo dieses Beispiel ausführlich diskutiert wird.

Wenn Sie die Zoom-in-Schaltfläche (mit dem Lupen-Symbol) anklicken, können Sie danach mit der Maus einen Bereich der Graphik vergrößert darstellen. Sie können also die Lösungen der Gleichung $3\cos x = \log x$ nähherungsweise aus der graphischen Darstellung ablesen. Das untenstehende Beispiel zeigt zwei Lösungen in der Nähe von $x = 11,9$ und $x = 13,1$.



Die Vergrößerung zeigt aber auch, dass nicht die „wirklichen“ Funktionen dargestellt werden, sondern die einzelnen Datenpunkten der Vektoren x, y und z stückweise durch Geraden verbunden sind. Deswegen kann man aus der Graphik die Lösungen der Gleichung $3\cos x = \log x$ natürlich nur im Rahmen der Zeichengenauigkeit darstellen.

Ü 1.5 Weitere Übungsbeispiele

Sie können, so weit Sie kommen, die folgenden Beispiele während dieser Übungseinheit ausarbeiten oder sonst auf Ihrem eigenen Rechner fertigstellen.

Speichern Sie die Aufgaben als Skript-Dateien auf einem USB-Datenträger, sodass Sie bei Befragungen Ihre Beispiele vorweisen können.

Aufgabe 2: Lösungen von $3\cos x = \log x$

Fertigen Sie als Skript-Datei eine genauere Version der obigen Funktionsgraphen mit 1000 Datenpunkten an. Geben Sie Ihrem Bild auch einen Titel und beschriften Sie die Achsen. (Informieren Sie sich in der MATLAB-Hilfe, beim Übungsleiter oder einer hilfreichen Nachbarin, wie das geht!)

Bestimmen Sie aus der Graphik alle Lösungen der Gleichung.

Hinweis: Beginnen Sie mit `x=linspace(0.1,25,1000);`

> Wenn Sie die Unterlagen in Papierform vor sich haben oder in Ihre digitale Kopie hineinschreiben können, tragen Sie hier Antworten ein. Schreiben Sie geforderte Antworten jedenfalls auch als Kommentarzeile in Ihr Skript.

Die Lösungen sind (auf drei Nachkommastellen genau):

_____

_____

Aufgabe 3: Quadratische Gleichung

Gesucht ist die betragskleinere Wurzel von

$$x^2 - 12345678x + 9 = 0$$

Verwenden Sie Rechenbefehle im *command window* und speichern Sie zum Schluss in übersichtlicher und kommentierter Form als Skript-Datei.

(Sinnvoller Weise setzen Sie dazu das Anzeigeformat `format long e`)

- Die gängige Lösungsformel $x_{1,2} = -p/2 \pm \sqrt{p^2/4 - q}$ liefert

  _____

- Die numerisch korrekte Berechnung der betragsmäßig *kleineren* Lösung einer quadratischen Gleichung berechnet zuerst die *betragsgrößere* Lösung x_{gr} mit der Standard-Formel. Die betragskleinere Lösung x_{kl} ergibt sich dann aus

  $$x_{\mathrm{kl}} = \frac{q}{x_{\mathrm{gr}}}$$

  Ergebnis: _____

- Auflösen nach dem linearen Term,

  $$x = \frac{x^2 + 9}{12345678}$$

  Fixpunkt-Iteration in der Form

```
>> x=0;
>> x = (x^2+9)/12345678
x =
    7.290000597780049e-007
```

Wiederhole, bis sich Wert nicht mehr ändert

Ergebnis: _____

Aufgabe 4: Fixpunkt-Iteration:

Finden Sie wie in der vorigen Aufgabe durch wiederholtes Auswerten einen Fixpunkt der Funktion $\phi(x) = 1/\exp x$. Startwert 5, Genauigkeit `format long e`.

Ergebnis: _____

Untersuchen Sie auch, wie sich von einem Schritt zum nächsten die Genauigkeit erhöht. Können Sie eine Regel der Art „pro Dezimalstelle Genauigkeit sind durchschnittlich x Iterationen notwendig“ finden?

Aufgabe 5: Wurzelbehandlung:

Schon die alten Babylonier berechneten Quadratwurzeln $\sqrt{a}$ mit der (oft als Heron-Verfahren bezeichneten) Iteration

$$x^{(0)} = a; \quad x^{(k+1)} = \frac{1}{2}\left(x^{(k)} + \frac{a}{x^{(k)}}\right) \text{ für } \quad k = 0, 1, 2, \ldots$$

Berechnen Sie $\sqrt{2}$ durch wiederholtes Auswerten der Iterationsvorschrift (Genauigkeit `format long e`). Testen Sie auch andere Wurzelberechnungen, z.B. $\sqrt{2005}$, $\sqrt{4711}$, $\sqrt{0{,}815}$ ... Untersuchen Sie bei allen Beispielen, wie sich von einem Schritt zum nächsten die Anzahl richtiger Stellen erhöht. Welche der folgenden Regeln beschreibt das Konvergenzverhalten am besten?

1. Der Fehler halbiert sich mit jeder Iteration
2. Pro Iteration gewinnt man zwei korrekte Dezimalstellen
3. Pro Iteration gewinnt man drei korrekte Dezimalstellen
4. Pro Iteration verdoppelt sich annähernd die Anzahl korrekter Stellen

Aufgabe 6: Newton-Verfahren:

Finden Sie mit dem Newton-Verfahren für $f(x) = x \tan x - 1$ die Nullstelle in der Nähe des Startwertes $x^{(0)} = 1$. Anleitung:

```
>> x=1;
>> f = x*tan(x)-1
f =
    5.574077246549023e-001
>> fstr = ...
fstr =
    4.982926545469661e+000
>> x = x - f/fstr
x =
    8.881364757099055e-001
```

Hinweis: Die Ableitung von $\tan x$ ist $1/\cos^2 x$.

Wiederholen Sie, bis das Ergebnis auf die volle Stellenanzahl genau ist. Prüfen Sie anhand des Endresultates die Anzahl korrekter Dezimalstellen bei den einzelnen Iterationsschritten.

Untersuchen Sie das Konvergenzverhalten (die zunehmende Anzahl korrekter Stellen), indem Sie eine Tabelle erstellen:

Schritt	korrekte Dezimalstellen
0	0
1	
2	
⋮	

Wie lässt sich das Konvergenzverhalten beschreiben?

Die wiederholte Ausführung eines Befehls der Art `x = x - f/fstr` in einem Programm schreit nach einer Schleife. MATLAB kennt natürlich solche Kontrollstrukturen, und wenn Sie damit vertraut sind, dann schreiben Sie in Ihrem Skript eine `for`-Schleife. Sonst reicht es auch, die Anweisung mit copy/paste mehrfach zu wiederholen.

So könnte in einer Skript-Datei der Code aussehen, wenn Sie eine Schleife verwenden:

```
format long e
x=1
for i=1:7
    f = x*tan(x)-1;
    fstr =  ...
    x = x-f/fstr
end
```

Aufgabe 7: Kepler-Gleichung

Die Vorlesungsfolien zur 1. Vorlesung zeigen als Beispiel die Kepler-Gleichung

$$x - \epsilon \sin x = m$$

(sie setzt verschiedene Parameter einer elliptischen Umlaufbahn in Beziehung – aber das müssen Sie nicht wissen!) Angenommen, $\epsilon = 1/10$ und $m = 2$ sind gegeben; x ist gesucht. Verschiedene Lösungswege sind möglich:

- Ablesen aus geeigneter graphischer Darstellung.
- Durch Fixpunkt-Iteration
- Newtonsches Verfahren
- Sekanten-Methode
- Intervallhalbierung

Suchen Sie sich zwei davon aus und schreiben Sie dazu ein Skript. Ihr Skript soll eine Folge von Näherungswerten ausgeben. Auf den Vorlesungsfolien sind auch Zahlenwerte angegeben, damit können Sie vergleichen.

Analysieren Sie das Konvergenzverhalten ihres Verfahrens und geben Sie an, wie sich der Fehler von einem Schritt zum nächsten reduziert.

Ü 1.6 Kurven vom Typ $\mathrm{x} = \mathrm{f}(\theta); \mathrm{y} = \mathrm{g}(\theta)$ zeichnen

Eine Funktion $f : x \mapsto f(x)$ weist jedem x-Wert genau einen y-Wert zu. Ein Kreis lässt sich in dieser Form nicht beschreiben, weil hier zum gleichen x-Wert zwei y-Werte gehören können.

In solchen Fällen kann man Kurven in Parameterform darstellen. Dabei sind sowohl die x- als auch die y-Werte Funktion eines Parameters (der hier θ heißt).

Wir wollen einen Kreis mit Radius 1 zeichnen. Damit der Kreis auch wirklich jenes Aussehen hat, das wir erwarten, erzeugen wir 100 Punkte, die auf dem Kreis liegen. Dazu verwenden wir die Beziehungen:

$$x = \cos\theta, \qquad y = \sin\theta, \qquad 0 \le \theta \le 2\pi$$

Die Folge der MATLAB-Anweisungen, die unsere Aufgabe lösen, könnte wie folgt aussehen:

```
>> theta = linspace(0, 2*pi, 100);
```
Erzeugt 100 äquidistante Werte zwischen 0 und 2π.

```
>> x=cos(theta);
>> y=sin(theta);
```
Erzeugt die x- und y- Koordinaten der 100 Punkte.

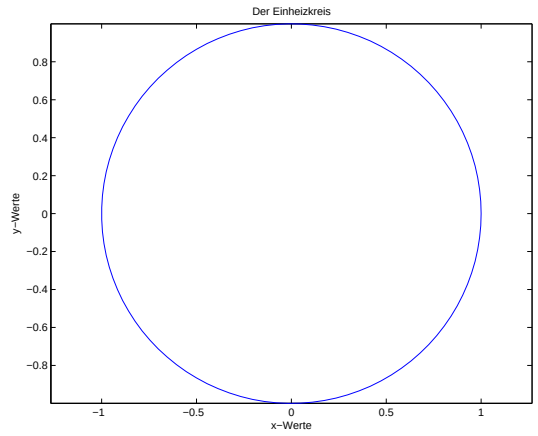
```
>> plot(x,y);
```
Zeichnet die Verbindunglinien von Punkt 1 zu Punkt 2 zu ... Punkt 100

Weitere Befehle, mit denen Sie Ihre Zeichnung formatieren können, sind

```
>> axis('equal');
```
Setzt die Skalierung der beiden Achsen gleich.

```
>> xlabel('x-Werte')
>> ylabel('y-Werte')
>> title('Der␣Einheizkreis')
```
Die x- und y-Achse werden beschriftet und Ihr Werk betitelt. (Strichpunkt oder nicht als Abschluss macht bei diesen Befehlen keinen Unterschied.)

Ihr Werk sollte nun so aussehen:



Sie können die Graphik auch interaktiv formatieren, wie Sie es von anderen Windows-Anwendungen gewohnt sind. Oben, neben dem Drucker-Symbol finden Sie die Schaltfläche „Edit plot“. Alternativ können Sie auch aus dem Menue „Tools–Edit Plot“ wählen. Danach können Sie durch Doppelklick Achsen oder andere Elemente der Zeichnung weiter editieren. Andere Schaltflächen erlauben Ihnen, Text, Pfeile oder Linien einzufügen.

Wichtig sind noch die Möglichkeiten, die Graphik zu speichern. Unter dem Menüpunkt „File–Export“ lässt sich die Graphik in verschiedenen Formaten speichern.

Ü 1.7 Elementweise Rechenoperationen

Es ist Ihnen vielleicht noch gar nicht bewusst gewesen: Sie haben beim Auswerten von Funktionstermen Rechenoperationen auf Datenvektoren angewandt. Der Befehl `x = cos(theta)` funktioniert sowohl für skalares θ, als auch für einen Datenvektor mit n Elementen. Auch Addition und Subtraktion lassen sich gleichermaßen auf skalare wie vektorielle Operanden anwenden.

Für Multiplikation, Division und die Potenzfunktion müssen Sie beim Verknüpfen von Datenvektoren aber unbedingt die elementweisen Operatoren `.*` `./` `.^` verwenden. Grund: Die „gewöhnlichen“ Operator-Symbole `*` `/` `^` verknüpfen in MATLAB Matrizen und Vektoren nach den Regeln der Matrix-Algebra. Diese Regeln wollen Sie *nicht* anwenden, wenn Sie Datenvektoren in Funktionsterme einsetzen.

Anfangs ist es oft schwer zu durchblicken, wann MATLAB Rechenoperationen von sich aus elementweise interpretiert, und wann man explizit den Punkt setzen muss.

> **Faustregel:** in Funktionstermen immer die elementweisen Operatoren verwenden, wenn Datenvektoren verknüpft werden.

Die Aufgaben 8 und 10 erfordern diese Unterscheidung. Dazu noch Hinweise:

```
>> t = linspace(0, 2*pi, 100);
>> x = sin(t)*2 - sin(t*2);
```

Erster Funktionsterm in Aufgabe 8: Hier reicht der „gewöhnliche“ `*` Operator, weil er nur in Operationen vom Typ „Skalar mal Vektor“ auftritt.

```
>> x = sin(t).*2 - sin(t.*2);
```

Es wäre aber auch nicht falsch, die Punkt-Form zu verwenden. Das Ergebnis bleibt gleich.

```
>> r = 2 - 2*sin(t) + sin(t) .* abs(cos(t)).^0.5 ./ (sin(t)+7/5)
```

Ein anderer Funktionsterm in Aufgabe 8: Aber hier muss überall, wo Vektoren verknüpft werden, ein Punkt-Operator stehen!

Ü 1.8 Weitere Übungsbeispiele

Dokumentieren Sie Ihre Übungsbeispiele als Skript-M-Dateien. Speichern Sie Zeichnungen als `JPEG`-Dateien ab, dann sind diese Dateien außerhalb MATLABs mit jedem Bildbetrachtungsprogramm anzusehen. Wenn Sie Zeichnungen als `.fig`-Dateien abspeichern – das ist MATLABs Voreinstellung – dann können Sie diese Zeichnungen nur innerhalb Matlabs öffnen.

Aufgabe 8:

Zeichnen Sie die Kardioide oder Herzkurve. Das ist eine Kurve, die durch folgende Parameterdarstellung für $0 \leq t < 2\pi$ gegeben ist.

$$x = 2\sin(t) - \sin(2t)$$
$$y = 2\cos(t) - \cos(2t)$$

Der Name Kardioide geht auf Giovanni di Castiglione, einen italienischen Mathematiker im 18. Jahrhundert, zurück. Für mich sieht sie mehr wie ein Apfel ohne Stängel aus. Eine schönere Herzkurven-Gleichung ist hier gegeben[1]:

$$r = 2 - 2\sin t + \sin t \frac{\sqrt{|\cos t|}}{\sin(t) + 7/5}$$
$$x = r\cos(t)$$
$$y = r\sin(t)$$

Hinweis: Verwenden Sie Sie für $0 \leq t < 2\pi$ mindestens 500 Teilpunkte und achten Sie darauf, an den richtigen Stellen `.*` und `./` zu verwenden!

Aufgabe 9: 3D-Plots

Zeichnen Sie mittels `plot3(x,y,z)` eine Spirale:

$$x(t) = \sin(t), \quad y(t) = \cos(t), \quad z(t) = t, \quad 0 \leq t \leq 20.$$

Verwenden Sie die Schaltfläche „Rotate 3D“, um Zeichnungen der Kurve aus zwei verschiedenen Blickwinkeln anzufertigen.

Aufgabe 10: Für die Fisch

Diskutieren Sie die Funktion

$$y = 3 - \frac{1}{2}\arctan(\frac{2\,x - 5}{x - 2}) + \frac{1}{10}\sin(7\,x) \quad 0 \leq x \leq 4$$

anhand eines Graphen. Wo liegen Supremum und Infimum? In welchen Bereichen ist die Funktion stetig? differenzierbar? An welcher Stelle ragt eine Haifischflosse aus den Wellen?

Wenn die letzte Frage für Sie sinnlos erscheint, haben Sie die Funktion vermutlich falsch gezeichnet. Beachten Sie bei der Berechnung der y-Werte, dass alle Rechenoperationen elementweise (für jeden einzelnen x-Wert) berechnet werden müssen. Wenn einer der Partner in einer Rechenoperation ein Skalar, der andere ein Vektor ist, funktioniert das automatisch, wie z.B. in der Anweisung `2*x-5`. Wenn beide Partner Vektoren sind, müssen Sie unbedingt die elementweisen Operatoren `.*` und `./` verwenden. Beispiel: der Ausdruck $(2x-5)/(x-2)$ muss geschrieben werden `(2*x-5)./(x-2)`.

[1] Quelle: `http://mathworld.wolfram.com/HeartCurve.html`