Einführung in Mirana

Die Programmiersprache Mirana ist eine interaktive Skriptsprache, die unter anderem Elemente aus BASIC, C und Matlab (bzw. Matlab-Ähnliche wie Scilab, Octave, Freemat...) in sich vereinigt und auch objektorientierte Programmierung erlaubt. Ihre Syntax ist bewusst einfach und leicht verständlich gehalten, gleichzeitig kann sie auch sehr mächtig sein. So gelingt etwa das Lösen eines linearen Gleichungssystems aus komplexen Zahlen mit z. B. 500 Gleichungen und 500 komplexen Unbekannten in einer einzigen Zeile mit nur 5 Zeichen, die Berechnung kann dann aber einige Sekunden dauern:

```
X=A\Y
```

Oder um über tausende gemessener Punkte x[i], y[i] eine Näherungsparabel P zu fitten genügt:

```
P = y/[x^2; x; 1+0*x]
```

Auch eine Grafik ist schnell erstellt:

```
x=[-20:0.1:20]:Plot(x,sin(x)/x)
```

oder ein Bild:

```
Plot Fractal(Matrix(400,400,0.007,0x32)-0.7) fmod 5,1
```

Ein einfaches Hallo-Welt-Programm kommt mit einer Zeile aus:

```
MsgBox "Hello, world!"
```

Durch einen integrierten Assistenten (Wizard) kann man mit wenigen Mausklicks ein lauffähiges Programm mit dialogbasierter Benutzeroberfläche erstellen.
Auch Code anderer Programmiersprachen wie C kann per DLL-Aufruf, COM/OLE- oder DDE-Schnittstelle eingebunden werden, außerdem ist die volle Windows-API bei Bedarf auch direkt zugänglich.

Erste Schritte

Starten Sie das Programm Mirana.exe.
Dann öffnen Sie in der Menüzeile das Menü Windows, und wählen Console oder klicken auf  (alternativ kann man auf das Brillensymbol klicken und dort auf die Schaltfläche Console).

In das sich nun geöffnete Fenster mit dem Titel CommandWindow.txt gibt man unten ein:

```
?2+3
```

Wenn sich nun ein Fenster mit dem Titel Data öffnet, kann man es verkleinern und auf die Seite schieben, es wird im Moment nicht benötigt.
Das Ergebnis der Rechnung wird direkt unter der eingegebenen Zeile angezeigt:

```
?2+3
= 5
```

Außerdem erscheint das Ergebnis noch in dem Fenster Data, doch die Ausgabe dort ist erst für die Anzeige größerer Zahlenfelder oder Textfelder wichtig, welche in der Konsole aus Platzgründen nicht angezeigt werden.

Geben Sie nun ein:

```
?(2+4)*6-2^4*2+0.75
```

Es gilt die übliche Regel Punkt vor Strich, das Potenzzeichen ^ hat aber eine noch höhere Priorität. Durch Klammern kann man die Prioritäten (also was zuerst berechnet wird) ändern. Eine Aufstellung der Prioritäten findet man in der Hilfe (F1) unter Operatoren.
Bei Gleitkommazahlen muss man einen Dezimalpunkt statt eines Kommas benutzen, 0,74 wäre nicht erlaubt.

Weiterhin können viele vordefinierte Funktionen benutzt werden wie die Quadratwurzel sqrt (engl. square root):

```
?sqrt(36)
```

```
=6
```
In dem Fall könnte die Klammer auch weggelassen werden, also sqrt 36, aber in anderen Fällen ist die Klammer zwingend erforderlich, etwa bei selbst definierten Funktion oder verschachtelten Funktionen mit mehr als einem Parameter.

Damit könnte man Mirana schon als Taschenrechnerersatz benutzen. So funktionieren auch die anderen bei einem besseren Taschenrechner üblichen Funktionen wie sin, cos, tan, log,...
Dabei ist zu beachten, dass die trigonometrischen Funktionen wie sin oder cos den Winkel nicht in Grad übergeben bekommen, sondern im Bogenmaß, wobei 360° = 2*!pi entspricht (geben Sie mal ein ?!pi). Die Umkehrfunktionen sind dann asin, acos, atan, ..., diese liefern den Winkel ebenfalls im Bogenmaß. Beim Logarithmus muss man beachten, dass log der natürliche Logarithmus ist, der dekadische ist log10, beliebige Basen erhält man durch log(x)/log(basis) oder auch log(x,basis).

Geben Sie mal folgendes ein:
?!pi
?!Time
?!RelTime
?!SystemTime
?!AppPath
?!Fac[4]
Mit ! beginnen verschiedene vordefinierte Variablen, wie die Kreiszahl Pi (π).
!Time gibt Datum und Zeit aus, in der englischen Form wäre es !TimeE, außerdem wird danach in !StrResult noch der Wochentag abgelegt (deutsch oder englisch).
!RelTime ist die mikrosekundengenaue relative Zeit in Sekunden, normalerweise die Anzahl Sekunden seit Einschalten des Rechners (!WinTime wären die Sekunden seit Start von Windows, diese Zeit ist aber nur zehntelsekundengenau).
!SystemTime ist die Anzahl Sekunden seit Mitternacht (00:00:00) am 1. Januar 1970 Greenwich Mean Time (GMT).
!AppPath ist der Pfad, in dem Mirana.exe gestartet wurde.
!Fac[4] ist die Fakultät von 4 oder 4! = 1 * 2 * 3 * 4

Man kann vorhandene Einträge auch mehrfach ausgeben. Gehen Sie mit dem Kursor wieder hoch zu ?!Time und drücken nochmals die Eingabetaste. Leerzeilen kann man in der Console per Eingabetaste nur erzeugen, wenn gleichzeitig Strg gedrückt ist. Man kann auch Zeilen markieren, kopieren und einfügen (entweder per rechte Maustaste das Kontextmenü öffnen und dann copy und paste, oder per Strg+C und Strg+V, oder über die Symbolleiste oben oder das Menü Edit).

Über die Console kann man auch Befehle eingeben. Gehen Sie in der Console wieder ganz nach unten und geben Sie ein:
```
MsgBox "Hello, world!"
```

Als nächstes benutzen wir Zuweisungsbefehle, um Variablen zu definieren. Diese müssen mit einem Buchstaben oder Unterstrich _ beginnen und können sonst Buchstaben und Ziffern enthalten. Zwischen Groß- und Kleinschreibung wird dabei nicht unterschieden.

Geben Sie nun ein:
x=3
y=x*2+6
?x
?y

Rufen Sie nun das „Watch Window“ auf, entweder über das Debug-Menü, das Brillensymbol  oder durch Strg+Q und klicken dort auf „Var's”. Die bisher definierten Variablen werden nun angezeigt.
Markieren Sie nun mit der Maus in der vorher eingegebenen Zeile y=x*2+6 die Zeichenfolge x*2 und drücken Taste F9 (oder rechte Maustaste und „Show Value“): im Watch Window sollte in der oberen Zeile nun x*2 und unten 6 stehen. Die obere Zeile kann man auch verändern und dann auch „Show Value“ anklicken (oder Alt+S). Einzelne Variablen muss man nicht markieren, es reicht, wenn der Kursor vor oder bei mehreren Zeichen mitten in der Variable steht, wenn man F9 drückt.

Neben Zahlen kann eine Variable auch Zeichenketten (hier String genannt) aufnehmen.
Geben Sie ein:

```
s="Datum und Uhrzeit: "
t=!Time
?s
?t
```

Man kann Strings auch zusammenfügen und zerlegen:

```
?s+t
?"Datum: "+t[:9]+", Uhrzeit: "+t[11:15]
```

Mit den eckigen Klammern kann man auch auf einzelne Buchstaben zugreifen und ändern.
Mit `t[2]='/', t[5]='/'` ersetzt man z. B. im Datum die Punkte durch Schrägstriche, wie im Englischen üblich. Mit ?s[0] erhält man aber nicht "D", sondern 68, den ASCII-Code von D. Den erhält man auch, wenn man ?'D' eingibt. Mit $ kann ein ASCII-Code in einen String der Länge eins umgewandelt werden, d. h. ?$s[0] liefert "D". Ein Bereich ergibt aber immer ein String, auch wenn der Bereich nur ein Zeichen umfasst, d. h. ?s[0:0] gibt "D" zurück. Bereiche können auch durch andere Strings ersetzt werden; ist der Bereich ein leer, wird der String eingefügt, z. B.

```
s[10:9]="auch "
```

?s -> "Datum und auch Uhrzeit".

Den letzten Buchstaben erhält man übrigens mit s[!max] oder kurz s[!m].
Um Zahlen in Strings einzufügen, kann man # oder #. davorsetzen. Bei # werden nur ganze Zahlen ausgegeben, falls nötig, wird gerundet. Auch #, ist möglich. Dann wird statt Dezimalpunkt ein Komma verwendet:

```
?"2+3="+#(2+3)
s="Pi="+#.!pi+" oder genauer "+#..!pi
?s
?#,,!pi
```

Mehr Informationen über die Raute erhält man, indem man # markiert oder den Kursor davorsetzt und F1 drückt. Im Hilfetext dann bis zu # herunterscrollen.

Eine Variable kann aber auch mehr als eine einzige Zahl enthalten:

```
x=[3:7]
?x
```

So wird eine Matrix mit 5 Spalten und einer Zeile erzeugt. Da eine Matrix auch sehr groß sein kann, wird der Inhalt nicht mehr in der Konsole angezeigt, sondern nur noch im Datenfenster. Markieren Sie mal x und drücken F9. Jetzt wird der Inhalt auch im Watch Window angezeigt. Man kann hier auch auf Var's klicken, dann auf x = Matrix 5 * 1 und danach auf „Show Value“.

Man kann auch beliebige Werte eingeben:

```
x=[ 7 3 1.25 10:15 30:25, 2+1/7]
?x
```

Die einzelnen Werte oder Bereiche sind dabei durch Leerzeichen oder Komma getrennt.
Bei Bereichen kann man auch das Inkrement angeben:

```
x=[-10:0.1:10]
?x
```

Jetzt geht es in 0.1-Schritten von -10 bis +10. Wenn im Datenfenster nicht alles angezeigt wird, einfach Taste Ende drücken. Übersichtlicher wird es, wenn man im Watch-Window x auswählt und auf Plot klickt. Jetzt sieht man ein Diagramm, dessen x-Achse von 0 bis 200 geht und die y-Achse von -10 bis 10. Dies kann man auch per Befehl erreichen:

```
Plot x
```

Mehr Informationen zu Matrizen lassen sich übrigens in der Hilfe finden unter Matrix.

Geben Sie nun ein:

```
Plot sin(x)
```

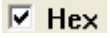
Man erhält eine Sinuskurve. Allerdings ist die x-Achse noch nicht richtig beschriftet. Geben Sie daher ein:

```
Plot x,sin(x)
```

Jetzt geht die x-Achse von -10 bis 10, und man sieht, dass die Kurve Nulldurchgänge bei 0, pi, 2*pi usw. hat. Um den Wert besser ablesen zu können, zieht man mit der linken Maustaste ein kleines Rechteck um einen Nulldurchgang. Dadurch wird die Kurve gezoomt. Ein Klick in das Diagramm stellt danach die alte Ansicht wieder her.
Fügen wir nun den Kosinus als zweite Kurve hinzu:
Plot cos(x),"Add"

Man kann auch weitere Fenster öffnen und die Farbe und Dicke festlegen:

```
y=sin(x)/x
Plot x,y,0xff,"Window=2, Title=Sinc-Funktion"
Plot -y,"Window=2, Add",0x0300ff00
```

Die mit 0x beginnende Zahl, die vor oder hinter dem String stehen kann, bestimmt Farbe und Dicke. Es handelt sich dabei um eine Hexadezimalzahl, also eine Zahl im 16er-System, welche zusätzlich zu den Ziffern 0 bis 9 noch die Buchstaben a bis f verwendet. Dies ist zur Angabe der Farbe hilfreich, weil hier ein aus 3 Byte bestehender RGB-Wert angegeben wird. Dabei das unterste Byte der Rotwert, das mittlere ist der Grünwert, der obere der Blauwert.
Jeder dieser Werte kann von 0 bis 255 oder hexadezimal von 0x00 bis 0xff gehen. Da man bei einem 32-Bit-System typischerweise mit 32-Bit-Werten rechnet, hat man vorne ein Byte frei, welches für die Dicke benutzt wird. Die Hexadezimalzahl setzt sich folgendermaßen zusammen: 0xDDBBGGRR, d. h 0x0300ff00 ist grün mit Dicke 3, 0xff ist rot ohne Dickenwert, was die kleinste Dicke 1 bedeutet. Dickenwerte von 0xf1 bis 0xf4 führen zu verschieden gestrichelten Linien der Dicke 1.
Übrigens: Ob ganzzahlige Werte hexadezimal oder dezimal angezeigt werden, kann man im Watch-Window mit Checkbox Hex  einstellen.

Da beim Befehl Plot x,y einfach alle Punkte (x,y) mit Linien verbunden werden, kann man auch folgendes eingeben:

```
Plot sin(x),cos(x)
```

Falls eine Ellipse gezeichnet wird, veränderen Sie bitte die Größe des Fensters so, dass ein Kreis zu sehen ist. Der Kreis ist eine der sog. Lissajous-Figuren (siehe Wikipedia). Weitere kann man so erzeugen:

```
Plot sin(x*2),cos(x)
Plot cos(x),sin(x*2)
```

```
Plot cos(x*3),sin(x*2)
Plot cos(x),sin(x*11/12)
```

Beim letzen reicht der Wertebereich von x nicht aus. So sieht es besser aus:

```
x=[0:0.1:100] : Plot cos(x),sin(x*11/12)
```

Mit dem Doppelpunkt kann man zwei oder mehrere Befehle in einer Zeile voneinander trennen. Sie werden dann sofort hintereinander ausgeführt.

Interessanter werden die Lissajous-Figuren, wenn man das ganze in Bewegung sieht. Dazu reicht die Konsole nicht mehr aus, daher schreiben wir nun ein kleines Programm.

Das erste Programm

Wir klicken nun auf das Weiße-Blatt-Symbol oben links oder im Menü File auf New.
In das nun geöffnete Fenster geben wir ein:

```
; Lissajous-Figuren
x=[0:0.1:100]
Loop
  Plot cos(x),sin(x+t)
  t+=0.01
  Pause 10
EndLoop
```

Bevor das Programm gestartet wird, sollten wir es speichern. Dazu geben klicken wir auf das Diskettensymbol oder im Menü File auf „Save As...“, wählen als Verzeichnis z .B. „C:\Mirana\Makros“ und geben als Dateinamen ein „Lissajous“. Danach starten wir das Programm mit Taste F5 oder Klick auf das rote R. Wenn eine Fehlermeldung erscheint, sollte man sich die Zeile, an der der Kursor steht, noch mal genau anschauen. Meist liegt dort der Fehler.

Nun können wir die Plot-Zeile verändern. Probieren Sie mal folgende Varianten aus:

```
  Plot cos(x),sin(x*2+t)
  Plot cos(x),sin(x*3/2+t)
  Plot cos(x),sin(x*5/6+t)
  Plot cos(x),sin(x*1.01+t)
```

Nach jeder Änderung muss das Programm mit F5 o. ä. neu gestartet werden. Zum Abbrechen kann man die Esc-Taste drücken oder auf das Stopp-Symbol Stop klicken. Die Start- und Stoppbefehle stehen auch in dem Menü Debug. Hier gibt es auch den Menüpunkt „Single Step“, den man auch mit der Taste F8 oder {} ausführen kann. Drücken Sie nun mehrfach F8 und beobachten Sie, wie der Kursor durch das Programm wandert. Sie können auch F8 gedrückt halten oder abwechselnd F8 und F5 drücken.
Man sieht, dass das Programm immer wieder den Bereich zwischen Loop und EndLoop abarbeitet. Es handelt sich dabei um eine Endlosschleife. Würde man hinter Loop eine Zahl setzen, etwa Loop 10, so würde die Schleife nur 10 mal durchlaufen. Mit dem Befehl BreakLoop kann man eine Schleife vorzeitig abbrechen. Dagegen ist es verboten, mit dem Befehl Goto aus einer Schleife herauszuspringen oder in eine Schleife hineinzuspringen.
Für weitere Informationen einfach Loop anklicken und F1 drücken.

Markieren Sie nun das t und drücken F9. Drücken Sie ein paar mal F8 und klicken auf „Show Value“. Sie sehen, wie der Wert sich ändert. Drücken Sie F5 und mehrmals „Show Value“. Noch besser sieht man den Verlauf der Variable t, wenn man Alt+S gedrückt hält (das Watch-Window muss dabei den Fokus haben). Es funktioniert auch, wenn man t im Kodefenster markiert und F9 gedrückt hält.

Der Befehl += erhöht eine Variable um einen bestimmten Wert. Dabei ist variable+=1 das gleiche wie variable=variable+1, nur kürzer, vor allem, wenn man lange Variablennamen benutzt.

Der Befehl Pause 10 hält das Programm für 10 ms an. Ohne diese Pause würde das Programm viel zu schnell ablaufen. Vor allem bei Endlosschleifen ist der Pause-Befehl wichtig, da der Prozessor sonst mit Höchstleistung immer die selben Befehle abarbeiten würde, dabei viel Strom verbraucht, warm wird und der Lüfter zu lärmen beginnt. Solange der Pausenwert ein Vielfaches von 10 Millisekunden ist, wird der Prozessor nur schlafen gelegt oder andere Programme erhalten Rechenzeit. Ist es kein Vielfaches von 10 ms, wird versucht, die Zeit möglichst genau einzuhalten, was die Prozessorbelastung aber erhöhen kann (je nach Betriebssystemversion kann man den Prozessor ohne größeren Aufwand nur für jeweils 10 oder 16 ms schlafen legen, der Rest muss in Programmschleifen verheizt werden).

Zeilen, die mit einem Semikolon (**;**) beginnen, sind Kommentarzeilen, sie werden nicht ausgeführt. Auch wenn nach einem Befehl ein Semikolon steht, wird alles, was danach in der Zeile steht, ignoriert.

Bis jetzt wäre das Programm aber noch nicht selbständig ausführbar (also ohne Mirana.exe) - schließlich fehlen die Bedienelemente. Man könnte das Programm nicht einmal beenden und müsste es per Taskmanager abschießen. Daher bauen wird noch eine kleine Bedienoberfläche ein. Wir sollten damit die Animation der Grafik starten und anhalten können, die Geschwindigkeit verändern und die Art der Figur (z. B. per sin(x*Zähler/Nenner+t)) verändern können.

Eine Möglichkeit wäre, mit F1 die Hilfe zu öffnen, die Liste mit dem Abwärtspfeil oben rechts öffenen und nach BeginDialog zu suchen (einfach „beg“ eintippen). Das dort gezeigte Beispiel könnte man aus Ausgangspunkt benutzen und passend abändern.

Wir wollen hier aber den Assisenten benutzen und rufen im Menü „File“ den Menüpunkt „New (Wizard)…“ auf.
Als erstes klicken wir auf den Button „Zurücksetzen“. Bei der Bestätigung durch OK halten wir aber die Umschalttaste gedrückt, damit wir zunächst einen leeren Dialog erhalten (kann mit Vorschau überprüft werden).
Unter „Name der neuen Anwendung“ geben wir nun ein: „LissajousDemo“ (nicht „Lissajous“, wenn wir unser bisherigens Programm noch behalten wollen).
Die Fensterbreite setzen wir auf 140. Hinter „Schieberegler“ schreiben wir 3 und Größe 50, „Min“ auf 1 und „Max“ auf 50. Bei „zus. Pushbuttons“ geben wir 2 ein. Das Ergebnis kann man jederzeit mit Vorschau anschauen, auch die Eingabetaste erzeugt eine Vorschau. Mit Klick auf Beenden oder Taste ESC schließt man das Vorschaufenster wieder.
Zum Abschluss klicken wir auf „Erstellen“. Nach einer ev. Nachfrage sollte sich ein neues Fenster mit dem Titel „LissajousDemo.mac“ öffnen.

Hier fügen wir **vor** der Zeile, die mit „Permanent“ beginnt, folgende Zeile ein:

```
Inkrement=0.01, Zähler=1, Nenner=1
```

Dann suchen wir die Zeile „Loop“ und ersetzen sie mit folgenden Zeilen:

```
x=[0:0.1:100]
Loop
  if not gestoppt
    Plot cos(x),sin(x*Zähler/Nenner+t)
    t+=Inkrement
  endif
```

Weiter unten ändern wir den Wert hinter Pause von 50 auf 10.

Wenn wir nun F5 drücken, sollte sich schon etwas tun. Allerdings haben die Steuerelemente noch die falsche Beschriftung und tun auch noch nichts Sinnvolles. Das ändert sich, wenn man die Variablen cValue1, cValue2 und cValue3 durch Inkrement, Zähler und Nenner ersetzt (jeweils zwei Mal). Das geht am besten mit Menüpunkt Edit / Replace… (Strg+H). Wenn man hier „Alle ersetzen“ benutzt, ist man sicher, nichts vergessen zu haben. (Andere Möglichkeit: Doppelklick auf Inkrement, Strg+C, Doppelklick auf cValue1, F3, Strg+V, F3, Strg+V, F3, …)
Jetzt noch die Texte "Wert 1:", "Wert 2:", "Wert 3:", "Taste 1" und "Taste 2" abändern in "Inkrement:", "Zähler:", "Nenner:", "Start" und "Stop". Danach Speichern nicht vergessen. (Übrigens, falls das Speichern mal vergessen und Mirana beendet wurde oder der Rechner stürzt vorher ab: Bei jedem Makrostart wird der aktuelle Stand im Verzeichnis von Mirana.exe unter einem Namen wie „LissajousDemo.mac.tmp“ gesichert. Also dann erst diese Datei in Sicherheit bringen, **bevor** die veralte Version des Makros nochmal gestartet wird, denn sonst wird diese Sicherheitskopie wieder überschrieben.)

Der Test mit F5 zeigt, dass die Paramenter Zähler und Nenner jetzt wie gewünscht funktionieren. Nur das Inkrement kann man mit dem Schieber noch nicht vernünftig einstellen. Dazu suchen wird die Zeile, die mit „EditNumber Inkrement,“ beginnt und ändern in der folgenden Zeile die letzten drei Paramenter des Scrollbar-Befehls ab von „1,50,1“ in „0.001, 0.2, 0.001“ Diese Parameter haben die Bedeutung Minimum, Maximum, Schrittweite. Damit der Start- und der Stop-Button richtig funktioniert, ersetzen wir nach der Zeile „if push=1“ des MsgBox-Befehl durch

```
  gestoppt = 0
```

und nach „elseif push=2“ entsprechend

```
  gestoppt = 1
```

Jetzt sollte alles wie gewünscht funktionieren. Falls nicht, versuchen Sie mal, das Programm zu debuggen oder vergleichen Sie es mit folgender Lösung:

```
; LissajousDemo.mac
;*****************************************************************************

Inkrement=0.01, Zähler=1, Nenner=1
Permanent Inkrement, cValue2, Nenner

Decl IsWindow "l=l,user32"
xr=8,yr=6,Dx1=40,SxEd=30,SxScr=50,SxPush=50

; Erzeugen des Dialogs
BeginDialog 10,10,140,94,"LissajousDemo"
  xc=xr,yc=yr
  Text xc,yc+2,"Inkrement:"
  EditNumber Inkrement,xc+Dx1,yc,SxEd,12,1
  ScrollBar xc+Dx1+SxEd+4,yc,SxScr,8,0.001,0.2,0.001
  yc+=16
  Text xc,yc+2,"Zähler:"
  EditNumber Zähler,xc+Dx1,yc,SxEd,12,1
  ScrollBar xc+Dx1+SxEd+4,yc,SxScr,8,1,50,1
  yc+=16
  Text xc,yc+2,"Nenner:"
  EditNumber Nenner,xc+Dx1,yc,SxEd,12,1
  ScrollBar xc+Dx1+SxEd+4,yc,SxScr,8,1,50,1
  yc+=16
  PushButton push,xc,yc,SxPush,12,"Start",1
  xc=82
  PushButton push,xc,yc,SxPush,12,"Stop",2
  yc+=20
  xc=45
  PushButton push,xc,yc,SxPush,14,"&Beenden",99
EndDialog
hDlg=!DialogHandle
```

```
; Dialogschleife
x=[0:0.1:100]
Loop
  if not gestoppt
    Plot cos(x),sin(x*Zähler/Nenner+t)
    t+=Inkrement
  endif
  BreakLoop not IsWindow(hDlg)
  if push=1
    gestoppt = 0
    push=0
  elseif push=2
    gestoppt = 1
    push=0
  elseif push=99
    BreakLoop
  endif
  Pause 10
EndLoop
CloseDialog hDlg
End
```

Zur Erläuterung: Der Befehl Permanent sorgt dafür, dass die Variablen dahinter ihren Wert behalten, auch wenn das Programm beendet wird. Die Variable yc in der Dialogdefinition ist hilfreich bei der Positionierung der Steuerelemente (Controls). Wenn man ein neues Element dazwischen schieben will, verschieben sich die anderen automatisch nach unten und man muss nur noch den Dialog als ganzes vergrößern.
Die Zeile `if not gestoppt` enthält die Not-Funktion. Alles was zwischen if und endif liegt, wird nur ausgeführt, wenn gestoppt 0 ist. Man hätte auch schreiben können:
`if gestoppt=0` oder `if gestoppt<>1`
Beachte: Die Variable Zähler enthält einen Umlaut. Dies ist in Mirana erlaubt, in den meisten anderen Programmiersprachen aber verboten.
Zur Bedienung: Die Scrollbars kann man auch mit der rechten Maustaste verstellen, was wesentlich feinfühliger geht, da der Mauszeiger dann abgeschaltet wird. Das geht aber nur in Mirana. In die Zahlenfelder kann man auch direkt Werte eintippen, muss dann aber die Eingabetaste drücken, damit sie vom Programm übernommen werden.

Nun sollten wir das Programm kompilieren, damit man es direkt per Explorer starten kann.
Im Menü Build wählen wir Compile... aus. Bei „Icon file“ geben wir z. B. ein:
C:\Mirana\Icons\Lissa1.ico
und klicken auf Build. Dadurch wurde eine Datei Lissajous.exe erzeugt. Diese kann man auch auf einen anderen Rechner kopieren, muss aber die Datei ImgViewM.dll (oder auch ImgView.dll) mitkopieren, da sonst der Plot-Befehl nicht funktioniert.

Das Feld für die Icon-Datei kann man auch leer lassen, wenn man kein passendes Icon hat. Dann wird das Icon von Miranex.exe verwendet. Icons kann man mit einem Iconeditor wie dem kostenlosen Junior Icon Edit (unter Tools) selbst erzeugen oder mit ResHacker aus einem anderen Programm extrahieren (auch unter Tools oder http://www.dvddemystifiziert.de/sonstiges/reshack.zip). Um das Icon Lissa1.ico zu erzeugen, empfiehlt es sich, hinter den Plot-Befehl noch ein ,0x067f0000 gesetzt, damit die Kurve fetter wird, dann wird das Fenster kleiner gemacht und mit Strg+Druck in die Zwischenablage kopiert, in Junior Icon Edit mit Strg+V eingefügt und mit der Maus ein gestricheltes Auswahlrechteck über die blaue Figur gezogen. Dann auf OK gedrückt und File / Save as... Lissa1.