■ LEAST SQUARES FIT
Methode der kleinsten Quadrate
mit beliebigen Funktionen

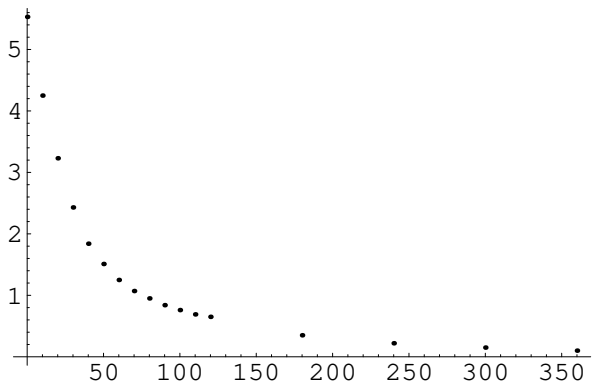
Wie kann man eine "beliebige", in nichtlinearer Weise von Parametern abhängige Funktion an Daten anpassen?

Nehmen wir dazu in einem Beispiel einen Datensatz:

```
In[1]:= data = ReadList["data2.dat", {Number, Number}]

Out[1]= {{0, 5.53}, {10, 4.25}, {20, 3.23}, {30, 2.43}, {40, 1.84}, {50, 1.51},
         {60, 1.25}, {70, 1.07}, {80, 0.95}, {90, 0.84}, {100, 0.76}, {110, 0.69},
         {120, 0.65}, {180, 0.35}, {240, 0.22}, {300, 0.15}, {360, 0.1}}
```

Wir können uns den Datensatz mit dem folgenden Befehl anzeigen lassen:

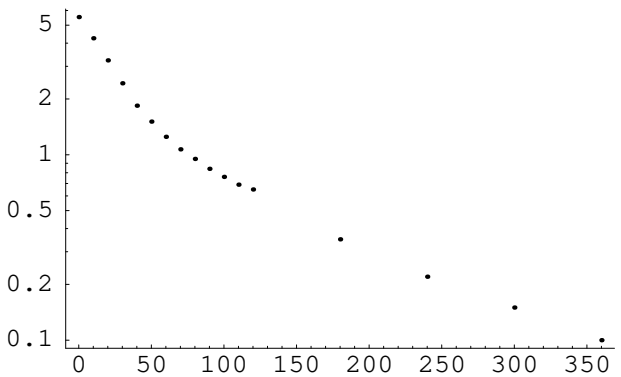
```
In[2]:= dataplot = ListPlot[data]
```




```
Out[2]= - Graphics -
```

Eine Ahnung, welche Funktionen hinter den Daten steckt, kann man bei einer halblogarithmischen Darstellung erkennen:
(Im Normalfall kennt man bereits die funktionale Abhängigkeit)

```
In[3]:= << Graphics`Graphics`
        datalogplot = LogListPlot[data]
```




```
Out[4]= - Graphics -
```

Um die LogPlot−Befehlebenutzen zu können, muss erst das Graphics−Modulgeladen werden.

In dieser halblogarithmischen Darstellung zeigen sich zwei "lineare" Bereiche.
Solche Daten lassen sich sehr gut mit einem Modell anpassen, mit
f(x) = a Exp[−bx] + c Exp[−dx],
wobei die Zahlwerte als y = f(x) interpretiert werden.
Das Ziel ist jetzt, für die Parameter a,b,c und d Werte zu finden,
sodass f(x) die Daten möglichst gut beschreibt.
Nehmen wir weiterhin an, dass wir bereits eine Annahme für den
ungefähren Wert dieser Parameter haben.
In den meisten Situationen ist dies auch der Fall:

```
In[5]:= soln = {a->1.61, b->0.0076, c->3, d->0.05} ;
```

Jetzt definieren wir eine Funktion y, die für gegebene Werte der vier
Parameter und einem Wert x den entsprechenden Wert von f(x) liefert:

```
In[6]:= Attributes[y] = {Listable} ;
        y[x_, a_, b_, c_, d_] := a Exp[-b x] + c Exp[-d x]

In[8]:= Attributes[yy] = {Listable} ;
        yy[x_] = y[x,a,b,c,d] /. soln
```

```
Out[9]= 3 e^(-0.05 x) + 1.61 e^(-0.0076 x)
```

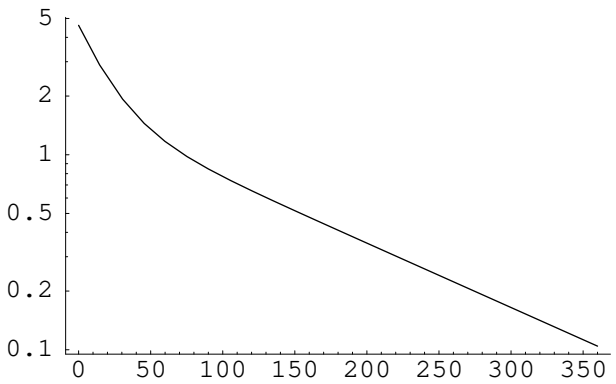
Als nächstes bereiten wir die Daten für diese neue Funktion vor:

```
In[10]:= xlist = Transpose[data][[1]] ;
         ylist = Transpose[data][[2]] ;
```

```
General::spell1 :
 Possible spelling error: new symbol name "ylist" is similar to existing symbol "xlist".
```

Wir können uns anzeigen lassen, wie gut unsere erste Annahme der
Parameter zu den Daten passt. Zunächst zeigen wir dazu den Verlauf
der Funktion yy(x) für den Bereich der durch die x−Werteabgedeckt wird:

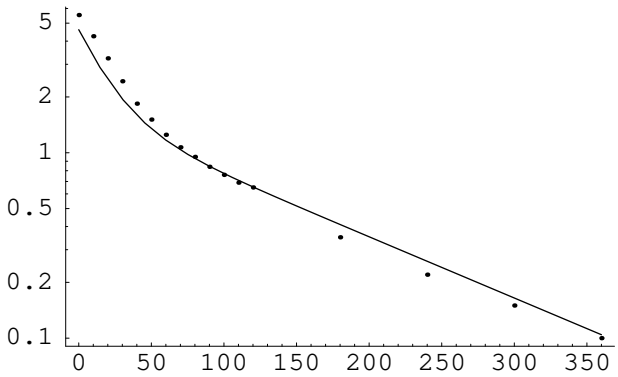
```
In[12]:= yyplot = LogPlot[yy[x], {x, Min[xlist], Max[xlist]}]
```




```
Out[12]= - Graphics -
```

Um jetzt zu sehen, wie gut diese Kurve zu den Daten passt,
stellen wir die Daten zusammen mit der Kurve dar:

```
In[13]:= Show[datalogplot, yyplot]
```




```
Out[13]= - Graphics -
```

Machen wir weiter mit der Analyse. Üblicherweise wird als Maß, wie gut ein Modell zu den Daten passt, die mittlere quadratische Abweichung der Messwerte zum Modell benutzt.
Als mittlere quatratische Abweichung können wir definieren:

```
In[14]:= rmserror[a_, b_, c_, d_] := Sqrt[ Apply[ Plus,
                    (ylist - y[xlist, a, b, c, d])^2]]

In[15]:= rmserror[a,b,c,d] /. soln

Out[15]= 1.61543
```

Das echte Ziel ist es jetzt, für die Parameter a,b,c,d Werte zu finden, für die diese mittlere quadratische Abweichung minimal wird:

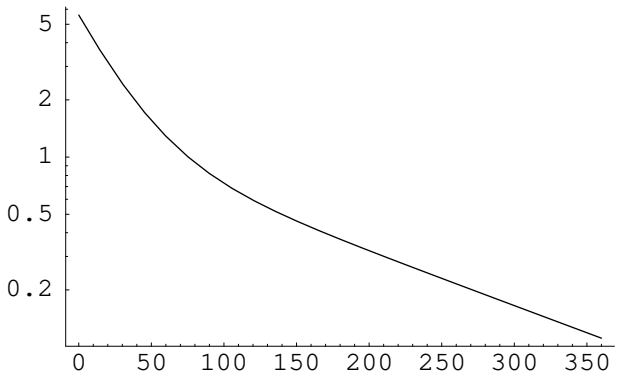
```
In[16]:= result = FindMinimum [ rmserror [ a, b, c, d],
           {a, {1,2}}, {b, {0,.02}}, {c, {3,4}}, {d, {0,.1}}]

Out[16]= {0.19049, {a → 1.18693, b → 0.00657126, c → 4.40442, d → 0.0366669}}
```

OK. Mathematica hat diese Werte gefunden. Wir müssen sie nur noch anwenden:

```
In[17]:= msoln = result[[2]]

         General::spell1 :
          Possible spelling error: new symbol name "msoln" is similar to existing symbol "soln".

Out[17]= {a → 1.18693, b → 0.00657126, c → 4.40442, d → 0.0366669}

In[18]:= finalcurve = y[x,a,b,c,d] /. msoln

Out[18]= 4.40442 e^(-0.0366669 x) + 1.18693 e^(-0.00657126 x)
```

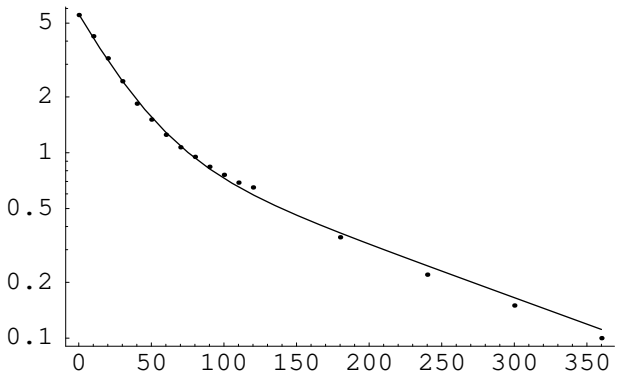
Wir können jetzt zeigen, dass diese neuen Werte die Daten deutlich besser beschreiben:

In[19]:=
```
newplot = LogPlot[finalcurve, {x, Min[xlist], Max[xlist]}]
```



Out[19]= - Graphics -

In[20]:=
```
Show[datalogplot, newplot]
```



Out[20]= - Graphics -