最小二乗法とロバスト推定（M 推定）

Maplesoft / サイバネットシステム（株）

はじめに

最小二乗法は、データフィッティングをはじめとしてデータ解析ではもっともよく用いられる手法のひとつです。Maple では、`CurveFitting` パッケージの `LeastSquares`コマンドや `Statistics` パッケージの `Fit` コマンド・`NonlinearFit` コマンドなどを用いてデータに適合する数式モデルを求めることが可能です。

一方で、最小二乗法は、データ中にノイズをはじめとした例外値が含まれている場合に、数式モデルが大きくズレてしまうことも多々あります。
この資料では、データ中に含まれる誤差の影響を可能な限り少なくするための手法としてよく知られている、ロバスト推定（M 推定）法とその Maple 上での簡易的な利用法について解説します。

まず最初に、例外値が入ったデータでは最小二乗法によるフィッティングがどの程度ズレてくるのかを実際に確認してみましょう。

例外値の影響

ここでは簡便に、線形式 $model = a \cdot x + b$ を考えます。また、係数 a, b は次の通りとします：

```
> model := x -> a*x + b
```

$$model := x \to a\,x + b \tag{2.1}$$

```
> a := 0.975 :
  b := 0.378 :
```

データのリストを変数 *data* に用意します。まず、元となる X 座標（*Xdata* 変数で定義）、Y 座標（*Ydata* 変数で定義）のリストを用意します。

```
> Xdata := [seq(x, x = 1.0 ..10.0, 0.5)]
```

$$Xdata := [1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0, 5.5, 6.0, 6.5, 7.0, 7.5, 8.0, 8.5, 9.0, 9.5, 10.0] \tag{2.2}$$

```
> Ydata := [seq(model(x), x = Xdata)]
```

$$Ydata := [1.3530, 1.8405, 2.3280, 2.8155, 3.3030, 3.7905, 4.2780, 4.7655, 5.2530, 5.7405, 6.2280, 6.7155, 7.2030, 7.6905, 8.1780, 8.6655, 9.1530, 9.6405, 10.1280] \tag{2.3}$$

さらに、Y 座標値には平均 0.01, 分散 0.2 の正規分布によるノイズを乗せます。これらのホワイトノイズは `Statistics` パッケージのコマンドを用いて生成します。

```
> with(Statistics) :
> noise := convert(Sample(RandomVariable(Normal(0.01, 0.2)), nops(Ydata)), list)
```

$$noise := [0.0527880651001289, 0.141393388886779, 0.370033864469223, 0.221579894712679, -0.179777081105624, 0.00503621480669719, 0.116070689647748, 0.312143492380907, -0.0768314912562584, \tag{2.4}$$

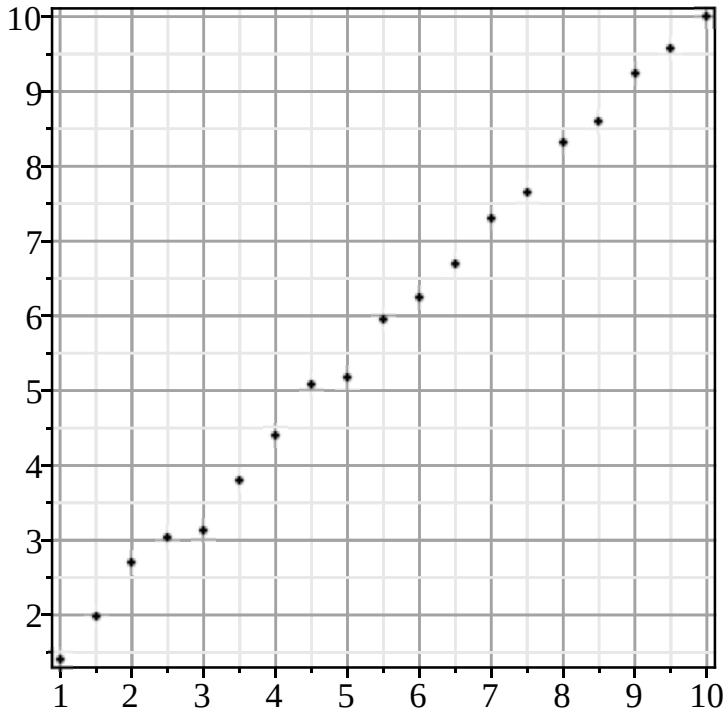
0.211982557525160, 0.0210213058896660, −0.0234744083210681,
0.100618098101482, −0.0491707419073126, 0.143892909779326,
−0.0671714263256098, 0.0896808641178227, −0.0648626832562612,
−0.123395574832661]

```
> Ydata := Ydata + noise
```

$$Ydata := [1.40578806510013, 1.98189338888678, 2.69803386446922, \quad \textbf{(2.5)}$$
3.03707989471268, 3.12322291889438, 3.79553621480670, 4.39407068964775,
5.07764349238091, 5.17616850874374, 5.95248255752516, 6.24902130588967,
6.69202559167893, 7.30361809810148, 7.64132925809269, 8.32189290977933,
8.59832857367439, 9.24268086411782, 9.57563731674374, 10.0046044251673]

作ったデータをグラフ描画して確認してみましょう。

```
> with(plots):
> gData := pointplot(Xdata, Ydata):
> display(gData)
```



このデータを元にした線形モデルへのフィッティングは `Statistics` パッケージの `LinearFit` コマンドで求まります。

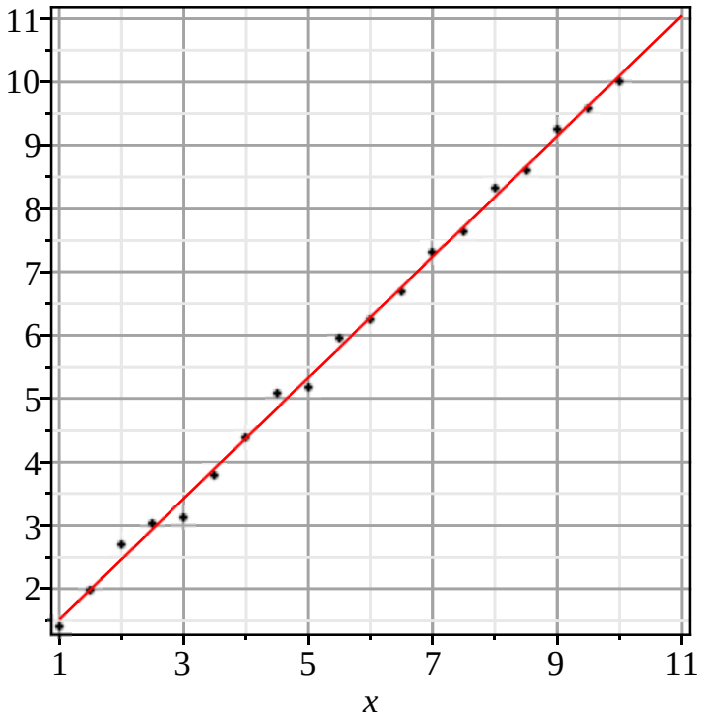
```
> fit1 := LinearFit([1, x], Xdata, Ydata, x)
```

$$fit1 := 0.559902853043000 + 0.953424916082256\, x \quad \textbf{(2.6)}$$

データ点のグラフと共に描画してみましょう。

```
> gFit1 := plot(fit1, x = 1..11, color = red):
```

```
> display(gData, gFit1)
```



ここで、もしもデータ点の Y 座標値にいくつかの大きな例外値が含まれた場合を考えてみましょう。
以下では、Y 座標値の5番目と18番目にズレた値を設定してみます。

```
> Ydata[5] := 7.384
```

$$Ydata_5 := 7.384 \quad \textbf{(2.7)}$$

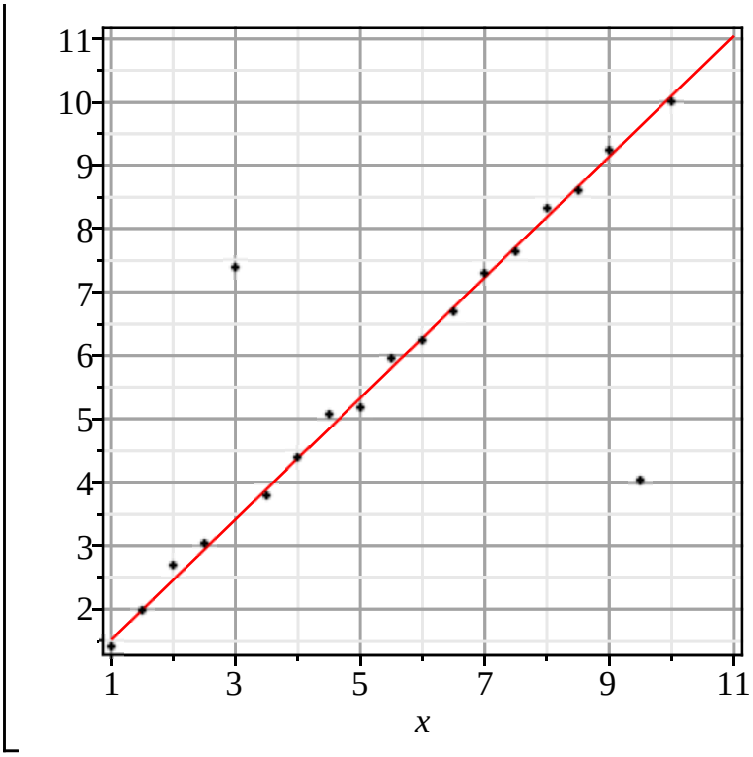
```
> Ydata[18] := 4.038
```

$$Ydata_{18} := 4.038 \quad \textbf{(2.8)}$$

```
>
```

もう一度グラフで確認してみます。

```
> gNewData := pointplot(Xdata, Ydata):
> display(gFit1, gNewData)
```



ズレたデータに線形フィッティングを適用して数式モデルを確認してみます。*fit1* の結果と確認してみると傾きと切片の値も大きく変わったことがわかります。

```
> fit2 := LinearFit( [1, x], Xdata, Ydata, x)
```

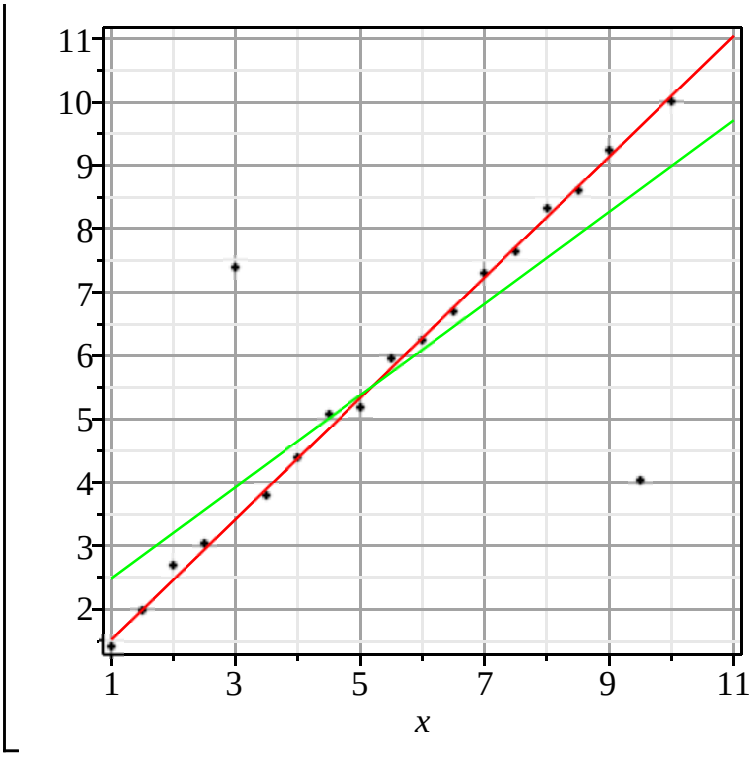
$$fit2 := 1.75876077631513 + 0.723231989978825\,x \quad \textbf{(2.9)}$$

```
> fit1
```

$$0.559902853043000 + 0.953424916082256\,x \quad \textbf{(2.10)}$$

グラフでも確認してみましょう。（元のフィッティングしたモデル式が赤色、新しいデータに対するモデル式が緑色で描画されています）

```
> gFit2 := plot( fit2, x = 1 ..11, color = green) :
> display(gNewData, gFit1, gFit2)
```



次の章では、このような例外値に対応するためのロバスト推定法を紹介します。

Tukey の Biweight 推定法

前章で見たように、最小二乗法はデータ点と数式モデルとの距離を最小化する手法です。従って、例外値のように大きくズレた値が含まれていた場合には、その影響を直接的に受けてしまうことを意味しています。

ここで紹介する Tukey による Biweight 推定法は、誤差の大きさに応じてデータ点に重みを付ける方法です。具体的には、誤差が大きい場合には重みを小さくすることで、例外値の影響を小さくする手法です。

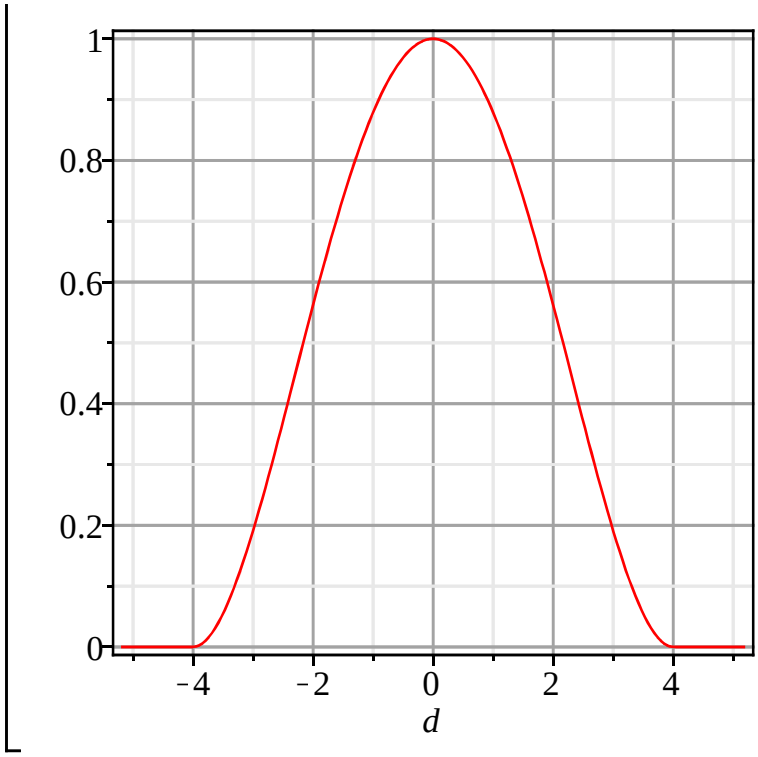
そこで、以下のような関数 biweight を定義します。

```
> biweight := (w, d) → piecewise(-w ≤ d and d ≤ w, (1 - (d/w)^2)^2, 0)
```

$$biweight := (w, d) \to piecewise\left(-w \le d \text{ and } d \le w, \left(1 - \left(\frac{d}{w}\right)^2\right)^2, 0\right) \qquad \textbf{(3.1)}$$

ここで、d : 誤差、w : 重み（誤差の許容範囲値）です。
例えば重み $w = 4.0$ とすると、この関数は次のようなグラフとなります :

```
> plot(biweight(4.0, d), d = -5.2..5.2)
```



ここで定義した biweight 関数を元のデータとフィットしたモデルとの誤差値に適用して行きます。
まず、*fit2* 式による誤差値のデータを作りましょう。

```
> errData := Ydata - [seq(eval( fit2, x = xv), xv = Xdata) ]
```

$$errData := [-1.07620470119383, -0.861715372396589, -0.507190891803558, -0.529760856549514, 3.45554325374839, -0.494536526434320, -0.257618046582682, 0.0643387611610651, -0.198752217465512, 0.215945836326493, 0.150868589701586, 0.232256880501440, 0.482233391934579, 0.458328556936372, 0.777276213633598, 0.692095882539250, 0.974832177993271, -4.59146468111397, 1.01352374906396] \quad \textbf{(3.2)}$$

この誤差値に biweight 関数を適用して、重み値のベクトルを作ります。なお、ここでは重みの許容値を $w = 3.0$ とします。

```
> weightVector := Vector( [seq(biweight(3.0, di), di = errData) ])
```

$$weightVector := \begin{bmatrix} 1 .. 19\ Vector_{column} \\ Data\ Type: anything \\ Storage: rectangular \\ Order: Fortran\_order \end{bmatrix} \quad \textbf{(3.3)}$$

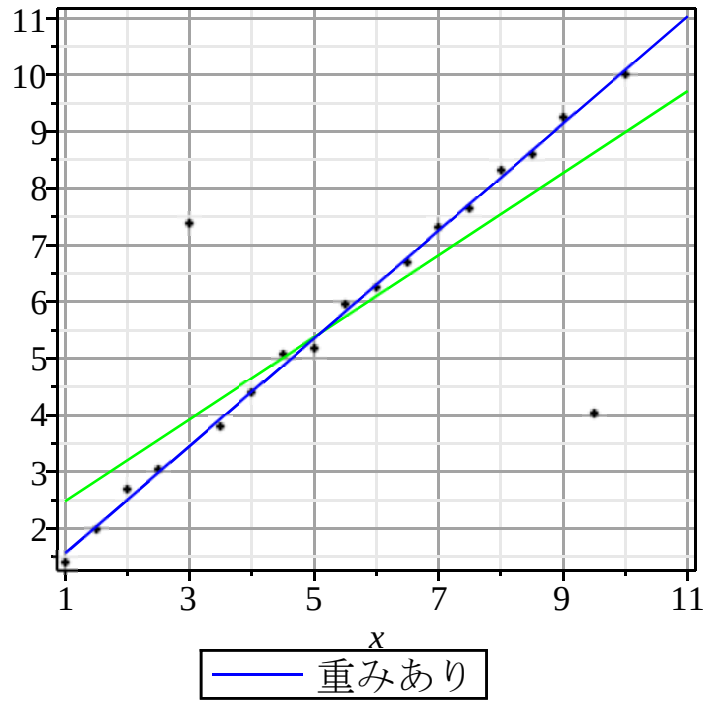
計算された重み値のベクトルデータを用いて、再び `LinearFit` コマンドに適用します。重み値のベクトルは `weights` オプションに指定します。

```
> fit3 := LinearFit([1, x], Xdata, Ydata, x, 'weights'= weightVector)
```

$$fit3 := 0.613964099258749 + 0.947435722116262\,x \quad \textbf{(3.4)}$$

新しいモデル式 *fit3*が得られました。グラフでその違いを確認してみましょう。

```
> gFit3 := plot(fit3, x = 1 ..11, color = blue, legend
      = "重みあり") :
> display(gNewData, gFit2, gFit3)
```



上記を見ると、例外値の影響を少なくしたフィッティング結果が得られていることがわかります。
また、フィッティングした数式モデルの結果を改めて確認すると次のようになっています。

```
> 'fit1'= fit1;
  'fit2'= fit2;
  'fit3'= fit3;
```

$$fit1 = 0.559902853043000 + 0.953424916082256\,x$$
$$fit2 = 1.75876077631513 + 0.723231989978825\,x$$
$$fit3 = 0.613964099258749 + 0.947435722116262\,x \quad \textbf{(3.5)}$$

```
> a, b
```

$$0.975,\ 0.378 \quad \textbf{(3.6)}$$

次の章では、Biweight 法を非線形のモデルフィッティングでも適用する方法を解説します。

非線形モデルでの Biweight 法適用例

線形の場合と同様の手順で、非線形モデルフィッティングにも Biweight 法による M 推定を適用してみます。
まず、データの元となるモデル式を次のように定義します：

```
> restart
> model := t -> p0*exp(p1*t^2)*(p2*sin(p3*t-p4)+p5*cos(p6*t-p7))+p8
```

$$model := t \mapsto p0\, e^{p1\, t^2} \left(p2 \sin(p3\, t - p4) + p5 \cos(p6\, t - p7)\right) + p8 \qquad \textbf{(4.1)}$$

各パラメータ値は以下とします。また、このパラメータ値を代入したモデル式を f で定義します。

```
> params := evalf([p0=1.09, p1=-0.89, p2=1.23, p3=5.08*Pi, p4=-4.21, p5=1.0,
      p6=1.87*Pi, p7=-3.38, p8=3.23]);
```

$$params := [p0 = 1.09,\ p1 = -0.89,\ p2 = 1.23,\ p3 = 15.95929068,\ p4 = -4.21,\ p5 = 1.0,\ p6 = 5.874778263,\ p7 = -3.38,\ p8 = 3.23] \qquad \textbf{(4.2)}$$

```
> f := unapply(eval(model(t), params), t);
```

$$f := t \to 1.09\, e^{-0.89\, t^2} \left(1.23 \sin(15.95929068\, t + 4.21) + 1.0 \cos(5.874778263\, t + 3.38)\right) + 3.23 \qquad \textbf{(4.3)}$$

データ点の個数 N, 及びデータ点のリスト $tvals$, $fvals$ を次のように用意します。

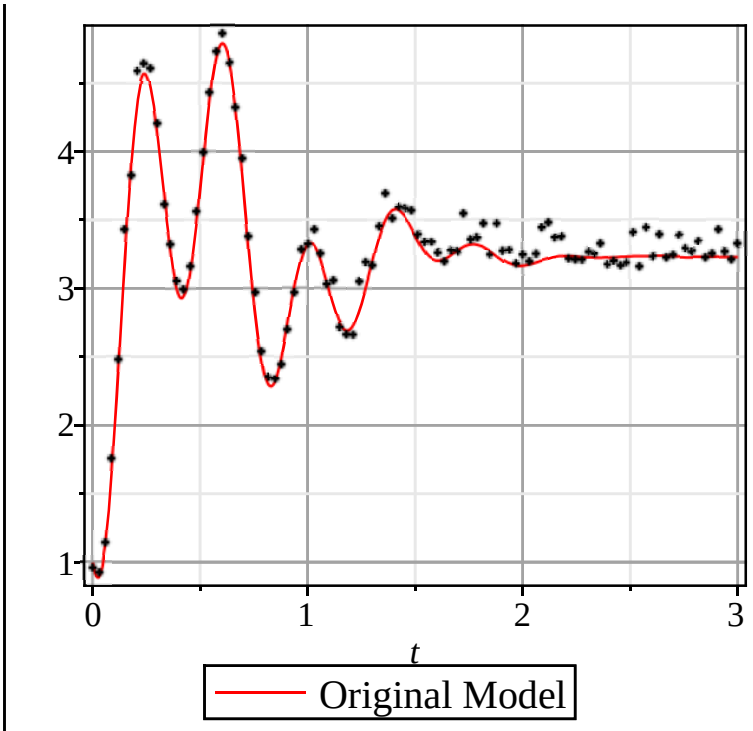
```
> N := 100:
> tvals := [seq(t, t=0..3.0, 3.0/(N-1))]:
> fvals := [seq(f(tv), tv=tvals)]:
>
```

最後に、正規分布のノイズ値を用意し、$fvals$ の値に加えます。

```
> with(Statistics):
> noise := convert(Sample(RandomVariable(Normal(0.06, 0.09)), N), list):
> fvals := fvals + noise:
```

グラフで確認してみます。

```
> with(plots): setoptions(gridlines=true, axes=boxed):
> gModel := plot(f(t), t=0..3.0, color=red, legend
      ="Original Model"):
> gPoint := pointplot(tvals, fvals):
> display(gModel, gPoint)
```



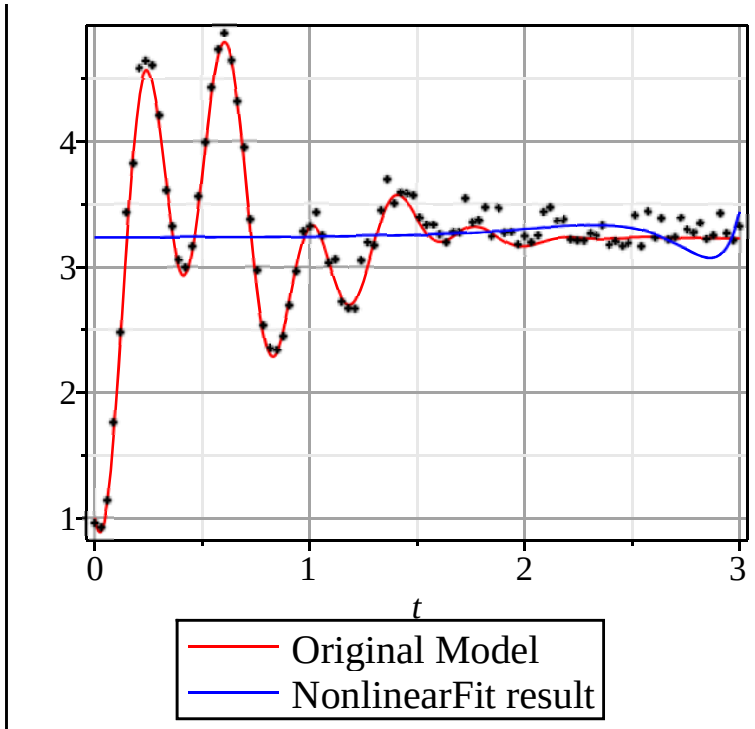
さて、まずは `Statistics` パッケージの `NonlinearFit` コマンドでデータに対するモデルフィッティングを適用してみます。

```
> fit1 := unapply(NonlinearFit(model(t), tvals, fvals, t), t)
```

$$fit1 := t \rightarrow -0.001967249988057751\, e^{1.0228555551187724 t^2} \left(-3.3245830837886667 \sin(1.8444848206448388\, t - 6.144314694383094) + 3.4153354461645185 \cos(1.6690059591912618\, t - 7.173823121307271) \right) + 3.239460872176842 \tag{4.4}$$

このフィッティングがどのような結果であるかをグラフで確認します。

```
> gFit1 := plot(fit1(t), t = 0..3.0, color = blue, legend
      = "NonlinearFit result") :
> display(gModel, gPoint, gFit1)
```



オリジナルのモデル式の振る舞いと比べると、振幅や周波数で大きく異なっているのがわかります。
そこで、再び biweight 関数を定義し、誤差値から重みデータを計算します。

誤差値を計算：

```
> err := fvals - [seq( fit1(t), t = tvals) ]:
```

biweight 関数を定義：

```
> biweight := (w, d)->piecewise( -w <= d and d <= w, (1 - (d/w)^2)^2, 0)
```

$$biweight := (w, d) \rightarrow piecewise\left(-w \le d \text{ and } d \le w, \left(1 - \left(\frac{d}{w}\right)^2\right)^2, 0\right) \qquad \textbf{(4.5)}$$

重みベクトルを計算：

```
> weightVector := Vector( [seq(biweight(9.3, d), d = err) ])
```

$$weightVector := \begin{bmatrix} 1 \text{ .. } 100\ Vector_{column} \\ Data\ Type:\ anything \\ Storage:\ rectangular \\ Order:\ Fortran\_order \end{bmatrix} \qquad \textbf{(4.6)}$$

上で計算した重み値を与えて、再度 `NonlinearFit` コマンドを適用してモデル式を求めてみます：

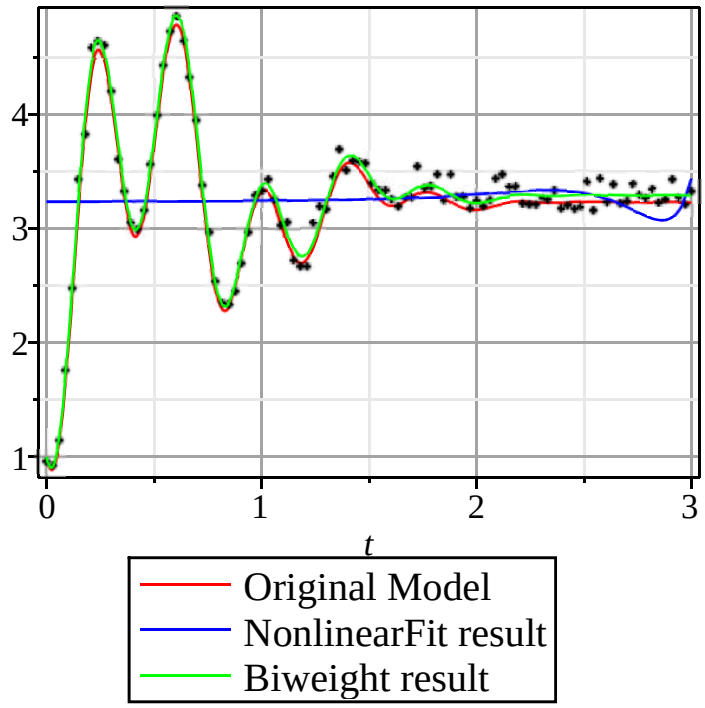
```
> fit2 := unapply(NonlinearFit(model(t), tvals, fvals, t, 'weights'= weightVector), t)
```

$$fit2 := t \rightarrow \qquad (4.7)$$

$$-0.0524795869584134\, e^{-0.8882094510094865\, t^2} (21.14632104159592 \sin(5.869437342638379\, t + 33.23187612702296) + 25.960192064206527 \cos(15.87898789528643\, t - 13.036600799590559)) + 3.2895176742385526$$

グラフで結果を確認してみます。（ここでフィッティングしたモデル式は緑色の曲線で示します）

```
> gFit2 := plot(fit2(t), t = 0..3.0, color = green, legend
      = "Biweight result"):
> display(gModel, gPoint, gFit1, gFit2)
```



オリジナルのモデル式に極めて近い結果が得られました。

上記の例では、ある特定の重み値を使いました。この値が適切かどうか、そしてより一般には重みの最適値はデータやモデル式によってまったく異なるため、複数回の試行が必要になることに注意が必要です。

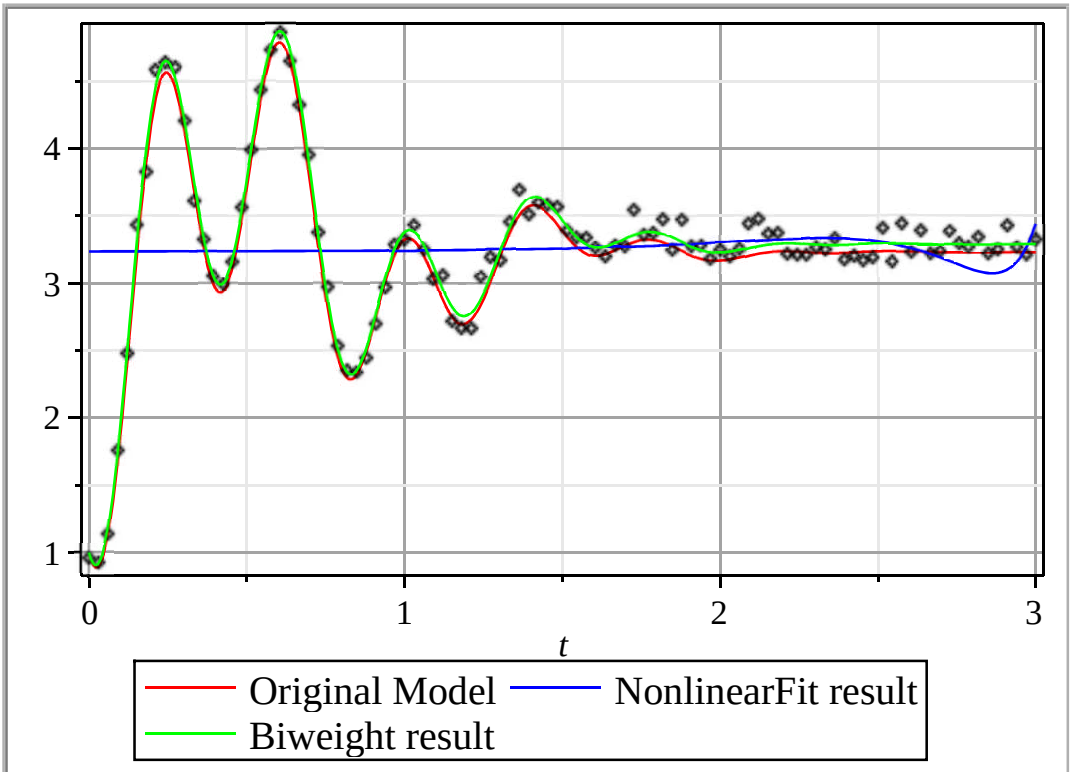
次の例では、この *w* 値を探索するための簡易的なアプリケーションを用意しています。ロバスト推定を手軽に行うための使用例として参考にしてください。

重み値の対話的な変更によるフィッティング変化

このアプリケーションを実行するには、まず下記のコード領域ボタンを押してコードを実行してください。その後、スライダーを動かすことで重み値を動的に変化させたフィッティング結果を確認することができます。

```
weightVecFunc := proc(w)
```

weight: 0 200 400 600 800 1000 (× 0.1) 9. 3



$$-0.0524795869584134\, e^{-0.8882094510094865\, t^2} (21.14632104159592 \sin(5.869437342638379\, t + 33.23187612702296) + 25.960192064206527 \cos(15.87898789528643\, t - 13.036600799590559)) + 3.2895176742385526$$