特異値分解を用いた曲面と画像データのスプライン補間

Maple には CurveFitting パッケージや Statistics パッケージ（統計解析パッケージ）のコマンド群を用いた多次元の曲面フィッティングを行う方法が用意されています。この資料では、標準パッケージのコマンドでは実現できない、2次元データに対するスプライン補間式の生成を、特異値分解を用いて実現する方法を解説しています。

行列の基本操作

Maple では、行列用のデータ型として Matrix 型が用意されています。Matrix コマンドまたはパレットの [行列] から必要なサイズを選択して行列データを入力することが可能です。

図：行列入力パレット

次は、Matrix 型を用いた行列の定義です。

```
> restart
> S := Matrix([[1, 2, 3], [2, 3, 4], [3, 4, 5]])
```

$$S := \begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 4 & 5 \end{bmatrix} \quad \textbf{(1.1)}$$

行列 S のある行・列の値を変更するときは、[] の記述で要素番号を指定し、値の割当てを行います。

```
> S[2, 2] := 3.2 ;
  S[3, 1] := 1.5;
```

$$S_{2,2} := 3.2$$
$$S_{3,1} := 1.5 \quad \textbf{(1.2)}$$

```
> S
```

$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & 3.2 & 4 \\ 1.5 & 4 & 5 \end{bmatrix} \quad \textbf{(1.3)}$$

複数の行列（またはベクトル）を連結するには、Matrix ・Vector のコンストラクタを使って定義することで可能です。例えば、次の 3 行 2 列の行列 K を考えます。

```
> K := Matrix([[x, y], [u, w], [g, h]])
```

$$K := \begin{bmatrix} x & y \\ u & w \\ g & h \end{bmatrix} \tag{1.4}$$

先に定義した Sと K を横に連結するには次のように記述します。

> $Matrix([S, K])$

$$\begin{bmatrix} 1 & 2 & 3 & x & y \\ 2 & 3.2 & 4 & u & w \\ 1.5 & 4 & 5 & g & h \end{bmatrix} \tag{1.5}$$

ベクトルの場合も同様に連結することが可能です。

> $v := Vector([0.1, 0.2, 0.3])$

$$v := \begin{bmatrix} 0.1 \\ 0.2 \\ 0.3 \end{bmatrix} \tag{1.6}$$

> $Matrix([S, v])$

$$\begin{bmatrix} 1 & 2 & 3 & 0.1 \\ 2 & 3.2 & 4 & 0.2 \\ 1.5 & 4 & 5 & 0.3 \end{bmatrix} \tag{1.7}$$

行列の転置や逆行列は、LinearAlgebra パッケージを用いなくても次の記述方法でも計算可能です。

行列の転置：

> $S^{\%T}$

$$\begin{bmatrix} 1 & 2 & 1.5 \\ 2 & 3.2 & 4 \\ 3 & 4 & 5 \end{bmatrix} \tag{1.8}$$

逆行列：

> S^{-1}

$$\begin{bmatrix} -0. & 1.25000000000000000 & -1. \\ -2.49999999999999956 & 0.312499999999999945 & 1.24999999999999978 \\ 1.99999999999999956 & -0.625000000000000000 & -0.499999999999999778 \end{bmatrix} \tag{1.9}$$

特異値分解

この資料では、特異値分解を用いて曲面のスプライン補間を実現します。Maple では任意の行列に対する特異値を LinearAlgebra パッケージの SingularValues 関数で計算することができます。特異値とは、ある種の固有値に相当するものです。（特異値の定義・計算方法については別途文献等を参照ください）

例）以下の行列 M に対して、特異値分解を計算します。

まず、線形代数の計算を行うための LinearAlgebra パッケージを読み込みます。

```
> restart
> with(LinearAlgebra) :
```

行列を定義します。

```
> M := Matrix([[1.5, 2.5, 1.0, 2.5], [3.5, 1.5, 2.0, 4.0], [1.5, 3.5, 1.5, 1.0]])
```

$$M := \begin{bmatrix} 1.5 & 2.5 & 1.0 & 2.5 \\ 3.5 & 1.5 & 2.0 & 4.0 \\ 1.5 & 3.5 & 1.5 & 1.0 \end{bmatrix} \tag{2.1}$$

この行列の特異値を SingulaValues コマンドで計算します。output オプションを指定して戻り値の形式を指定しています。

```
> U, S, Vt := SingularValues(M, output = ['U','S','Vt'])
```

$$U, S, Vt := \left[\begin{bmatrix} -0.500046645092658171 & 0.172987907043059969 & 0.848544952697521504 \\ -0.725171706739187072 & -0.619244801837874002 & -0.301101097875360102 \\ -0.473370202353878778 & 0.765905385418874208 & -0.435097221445752302 \end{bmatrix}, \begin{bmatrix} 7.77201080541906464 \\ 2.64365747191335121 \\ 0.779052765636532250 \end{bmatrix}, \begin{bmatrix} -0.514439100106394042 & -0.513981514062447964 & -0.342311073505821961 & -0.594977768020863330 \\ -0.287108627272472550 & 0.826230870308636956 & 0.0315307041026361910 & -0.483649666384192989 \\ -0.556678911641834028 & 0.188524405982474996 & -0.521534731848173672 & 0.618521350608264852 \\ -0.585687320812652402 & -0.132755792717534127 & 0.780916427750202868 & 0.171801614105044742 \end{bmatrix}\right] \tag{2.2}$$

ここで、U はユニタリ行列、Vt はユニタリ行列の随伴行列、S は特異値を表します。

特異値分解で得られた U, S, Vt は次の計算によって元の行列 M の値と一致します。（ただし、計算誤差を含むことに注意してください）

```
> U.DiagonalMatrix(S, 3, 4).Vt
```

$$\begin{bmatrix} 1.49999999999999844 & 2.49999999999999912 & 0.99999999999999922 & 2.49999999999999866 \\ 3.49999999999999734 & 1.49999999999999644 & 1.99999999999999934 & 3.99999999999999822 \\ 1.49999999999999800 & 3.49999999999999912 & 1.49999999999999934 & 0.99999999999999844 \end{bmatrix} \tag{2.3}$$

元の行列 M と再計算した **(2.3)** 式の差を計算し、fnormal コマンドで機械精度以下の誤差を 0 に丸めます。

> $map(fnormal, M - \mathbf{(2.3)})$

$$\begin{bmatrix} 0. & 0. & 0. & 0. \\ 0. & 0. & 0. & 0. \\ 0. & 0. & 0. & 0. \end{bmatrix} \quad \mathbf{(2.4)}$$

>

特異値分解を用いた 2 次元スプライン補間アルゴリズム

前節で説明された行列の特異値分解を、U, S, Vt の各行列の要素で表現してみます；

$$M := \sum_{k=1}^{r} \sigma_k \cdot u_k \cdot v_k^T$$

ここで、σ_kは行列 S の k 番目の特異値、u_k, v_k^T は行列 U, Vt の行・列ベクトルです。また、r は行列の階数です。
この分解を利用して、各行ベクトル(U)・列ベクトル(V)に対する１次元の補間式を構成し特異値を重みとして掛け合わせたものは、上述の定義から元の行列 M の各要素において一致することは自明です。すなわち、以下の式が言えます；

$$z(x, y) = \sum_{k=1}^{r} \sigma_k \cdot u_k(x) \cdot v_k(x)$$

$$M_{i,j} = z(x_i, y_j)$$

なお、１次元ベクトル式に対する補間・近似はいくつかの方法が考えられます。（例：多項式補間、ラグランジュ補間、スプライン補間、等）
本稿では各点を必ず通るという目的において、スプライン補間を利用します。

コードの定義

前節のアルゴリズムを Maple コードエディタ領域内に記述し、定義します。ボタンを押すか、またはコードエディタ領域上で右クリックし、[コードの実行]メニューを選択すると SVDInterpolate コマンドを定義します。

```
SVDInterpolate := proc(M::Matrix, vx::name, Xc::list, vy::name,
Yc::list, opt)
```

ここで、SVDInterpolate の引数は、M: 曲面データの行列、vx: X軸方向の変数名、Xc: X軸方向の座標データのリスト、vy: Y軸方向の変数名、Yc: Y軸方向の座標データのリスト、opt: オプション（スプライン補間の次数等）です。

数値実験

定義した SVDInterpolate コマンドを使っていくつかのデータに対して曲面のスプライン補間を計算してみます。

実験１：曲面の補間

次のデータ（行列）を考えます。

```
> R := Matrix([[0.2, 1.2, 1.2, 2.2, 3.5, 4.2],
               [0.1, 4.6, 6.4, 8.8, 5.4, 8.2],
               [0.2, 6.4, 7.3, 10.8, 8.2, 8.8],
               [0.0, 2.5, 3.1, 6.7, 3.4, 1.2]]):
```

データ行列 R の X座標, Y 座標は以下とします；

```
> Xc := [1.1, 2.6, 4.1, 5.0];
  Yc := [1.2, 2.0, 2.6, 3.5, 6.0, 6.8];
```

$$Xc := [1.1, 2.6, 4.1, 5.0]$$

$$Yc := [1.2, 2.0, 2.6, 3.5, 6.0, 6.8] \quad \textbf{(5.1.1)}$$

座標データの長さ（個数）をそれぞれ lenX, lenY 変数に割り当てておきます。

```
> lenX := nops(Xc)
```

$$lenX := 4 \quad \textbf{(5.1.2)}$$

```
> lenY := nops(Yc)
```

$$lenY := 6 \quad \textbf{(5.1.3)}$$

用意したデータ R 及び範囲 Xc, Yc から、特異値分解によるスプライン曲面補間を計算してみます。なお、スプラインの次数は 1 としておきます。x, y は、戻り値に用いる変数です。

```
> ret := SVDInterpolate(R, x, Xc, y, Yc, degree = 1)
ret
```

$$:= 26.1432320744771971 \begin{cases} 0.05786749340 - 0.2472409615\,x & x < 2.6 \\ -0.3493677480 - 0.09061202247\,x & x < 4.1 \\ -2.620691893 + 0.4633694767\,x & otherwise \end{cases} \begin{cases} 0.4539493119 \\ -0.0780652771 \\ 0.1881211744 \\ -0.8362541200 \\ 0.03007586875 \end{cases}$$

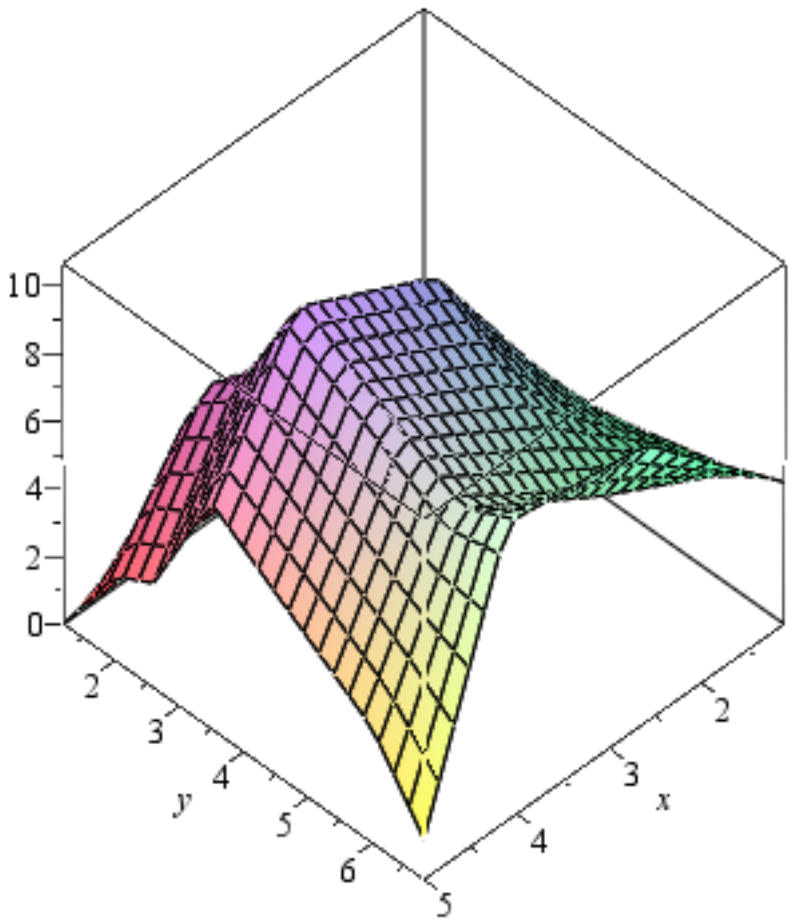
$$+ 3.93078991558797908 \begin{cases} -0.8076673773 + 0.2361097468\,x & x < 2.6 \\ -0.4900215935 + 0.1139382914\,x & x < 4.1 \\ -3.832738506 + 0.9292350991\,x & otherwise \end{cases} \begin{cases} -0.2140151704 + \\ -0.01457627583 + \\ -1.232580641 + \\ 1.537068507 - \\ 5.109575158 - \end{cases}$$

$$+1.83357493877547695\begin{cases}1.644708925-0.8785472993\,x & x<2.6\\ -2.071951849+0.5509376137\,x & x<4.1\\ -0.3731432150+0.1365940442\,x & otherwise\end{cases}\begin{cases}0.2772712869-0\\ 1.363257890-0.\\ -1.958809043+0\\ -1.204252095+0\\ 8.533298000-1\end{cases}$$

$$+0.835649617622550190\begin{cases}-0.4260041495-0.01294704413\,x & x<2.6\\ -2.412548881+0.7511086220\,x & x<4.1\\ 5.469930312-1.171447279\,x & otherwise\end{cases}\begin{cases}-1.182515228+\\ 2.619965208-\\ 2.322761877-\\ -1.380059925+\\ 3.233081178-\end{cases}$$

計算結果を関数化し、プロットしてみます。

```
> Rfit := unapply(ret, [x, y]) :
> g1 := plot3d(Rfit(x, y), x = Xc[1]..Xc[-1], y = Yc[1]..Yc[-1], axes = boxed) :
  g1
```

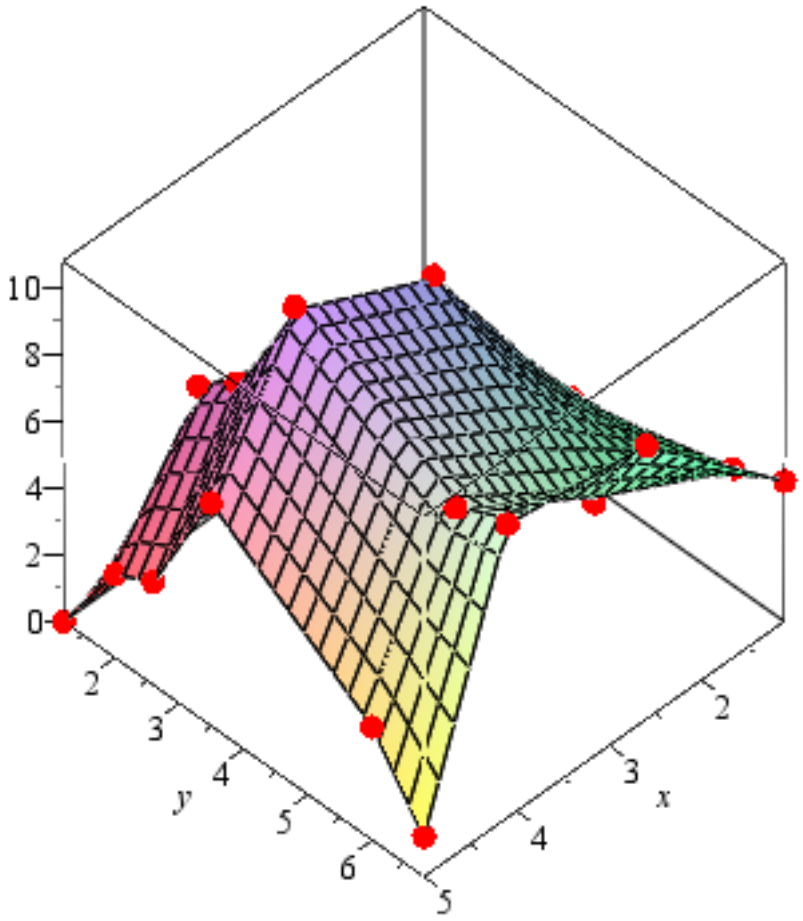


元のデータ点群と重ね合わせて描画してみます。点群データを座標データと共にリスト化するため、seq コマンドを使って用意します。

> $pts := [seq(seq([Xc[i], Yc[j], R[i,j]], i = 1..lenX), j = 1..lenY)]$

$pts := [[1.1, 1.2, 0.2], [2.6, 1.2, 0.1], [4.1, 1.2, 0.2], [5.0, 1.2, 0.], [1.1, 2.0, 1.2], [2.6, 2.0, 4.6], [4.1, 2.0, 6.4], [5.0, 2.0, 2.5], [1.1, 2.6, 1.2], [2.6, 2.6, 6.4], [4.1, 2.6, 7.3], [5.0, 2.6, 3.1], [1.1, 3.5, 2.2], [2.6, 3.5, 8.8], [4.1, 3.5, 10.8], [5.0, 3.5, 6.7], [1.1, 6.0, 3.5], [2.6, 6.0, 5.4], [4.1, 6.0, 8.2], [5.0, 6.0, 3.4], [1.1, 6.8, 4.2], [2.6, 6.8, 8.2], [4.1, 6.8, 8.8], [5.0, 6.8, 1.2]]$ **(5.1.5)**

> $g2 := plots[pointplot3d](pts, symbolsize = 24, symbol = solidsphere, color = red):$
> $plots[display](g1, g2)$



この曲面補間は、各軸方向の１次元スプラインから構築されているので、データ点（ノード点）においては曲面の補間式とデータ点はほぼ一致しています。実際にいくつかのデータ点でも確認してみます。例えば、座標 $\{Xc[1], Yc[3]\}$ の点における $R[1,3]$ の値で確認してみます。

> $Xc[1], Yc[3], R[1,3]$

$1.1, 2.6, 1.2$ **(5.1.6)**

> $Rfit(Xc[1], Yc[3])$

1.200000001 **(5.1.7)**

実験２：画像データの数式補間

本資料で定義した、曲面のスプライン補間を計算する SVDInterpolate コマンドを使って、画像データの補間を計算してみましょう。
まず、画像データを指定して読み込んでみます。画像データは Maple のサンプル用

データフォルダに置いてあります。

```
> fname := cat(kernelopts(datadir), "\\images\\FingerPrint.jpg")
```

$$fname := \text{"C:\Program Files\Maple 13\data\images\FingerPrint.jpg"} \quad \textbf{(5.2.1)}$$

ImageTools パッケージに用意されている Read コマンドを用いて画像ファイルを読み込みます。

```
> imgData := ImageTools[Read](fname)
```

$$imgData := \begin{bmatrix} 1..240 \times 1..256\ Array \\ Data\ Type: float_8 \\ Storage: rectangular \\ Order: C\_order \end{bmatrix} \quad \textbf{(5.2.2)}$$

読み込んだ画像データを表示してみます。（別ダイアログで表示されます）

```
> ImageTools[View](imgData)
```

ImageTools[View] コマンドで表示された画像

この例では、読み込んだ指紋の画像を SVDInterpolate コマンドで数式化します。画像データが格納されている imgData は Array 型となっているので、これを行列(Matrix)型に変換します。

```
> imgMat := convert(imgData, Matrix)
```

$$imgMat := \begin{bmatrix} 240 \times 256\ Matrix \\ Data\ Type: float_8 \\ Storage: rectangular \\ Order: C\_order \end{bmatrix} \quad \textbf{(5.2.3)}$$

画像の縦横を反転させるため、行列データの転置を取ります。

> $imgMat := imgMat^{\%T}$

$$imgMat := \begin{bmatrix} 256\ x\ 240\ Matrix \\ Data\ Type: float_8 \\ Storage: rectangular \\ Order: C\_order \end{bmatrix} \quad \textbf{(5.2.4)}$$

次に、行列のサイズ情報から画像補間のために使用する座標値を作ります。行列のサイズは LinearAlgebra パッケージの Dimensions コマンドで取得します。

> $LinearAlgebra[Dimension](\textbf{(5.2.4)})$

$$256, 240 \quad \textbf{(5.2.5)}$$

取得したサイズから、縦・横の座標値を用意します。

> $axisW := [seq(k, k = 1..\textbf{(5.2.5)}[1])]$

$axisW := [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23,$ **(5.2.6)**
24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45,
46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67,
68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89,
90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108,
109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124,
125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140,
141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156,
157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172,
173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188,
189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204,
205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220,
221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236,
237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252,
253, 254, 255, 256]

> $axisH := [seq(k, k = 1..\textbf{(5.2.5)}[2])]$

$axisH := [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23,$ **(5.2.7)**
24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45,
46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67,
68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89,
90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108,
109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124,
125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140,
141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156,
157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172,
173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188,
189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204,
205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220,
221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236,
237, 238, 239, 240]

画像データに対して曲面補間スプラインを適用してみます。また、time コマンドで補間スプライン式の計算に要する時間を計測しておきます。

> $st := time(\ ):$

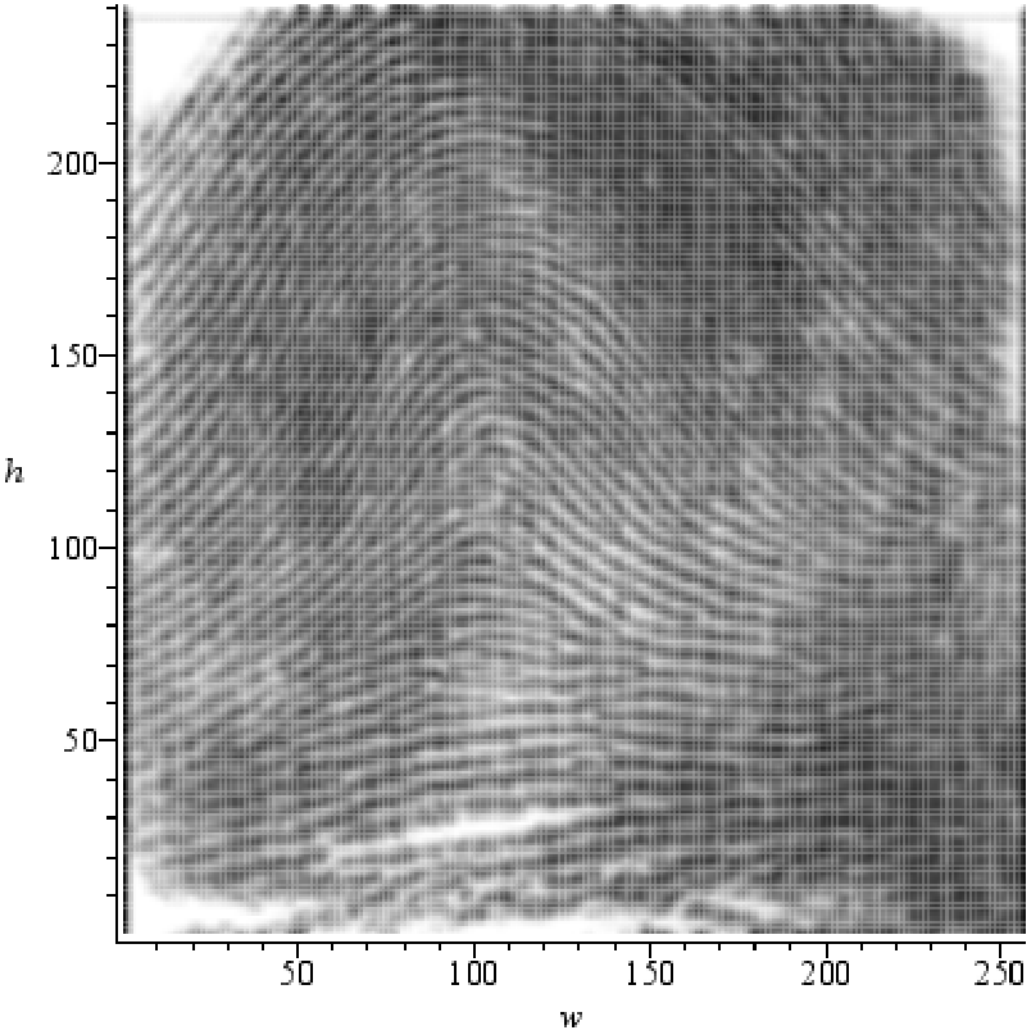
```
fitImg := unapply(SVDInterpolate(imgMat, w, axisW, h, axisH, degree = 2), [w,
    h]) :
time( ) - st
```

17.469 **(5.2.8)**

短い時間でスプライン式を構築できています。（使用環境：Intel® Core 2 Duo 3.0GHz, 2GB RAM, WindowsXP SP2）

作ったスプライン式をプロットしてみます。

```
> plots[densityplot](fitImg(w, axisH[-1]-h), w = axisW[1]..axisW[-1], h
    = axisH[1]..axisH[-1], style = patchnogrid, grid = [150, 150])
```



densityplot コマンド中の grid オプションで縦・横のプロット点数を指定しています。この値を増やすと画像はより精細になりますが、計算時間も相応に増えます。

ここで作成した imgFit の中身は、すべて明示的な区分関数として定義されています。従って、区分関数式を元にして C 言語のソースに変換したり、またネイティブコード化してコードの実行速度を高速化させることも可能です。