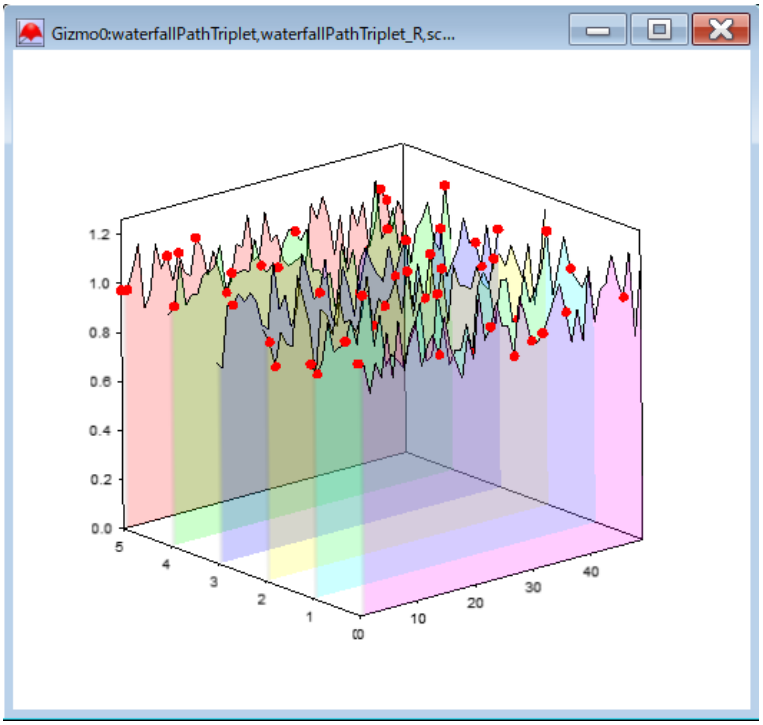
CONTENTS

サンプルの Experiment – Gizmo Waterfall Demo

クイックノート

メニュー File → Example Experiments → Visualization → Waterfall

この Experiment は、Gizmo でウォーターフォールプロットを作る方法を説明します。
これには、Igor Pro v9.0 以降が必要です。



出発点は、2D 行列（matrixData）に格納されたスペクトルデータです。
各列は、温度や圧力などの特性に対応し、各行は特定の波長に対応します。
matrixData のウェーブスケーリングには、波長と列プロパティの両方が設定されているものと想定します。
データが矩形グリッドでサンプリングされていない場合は、以下に説明する方法を使う必要があります。

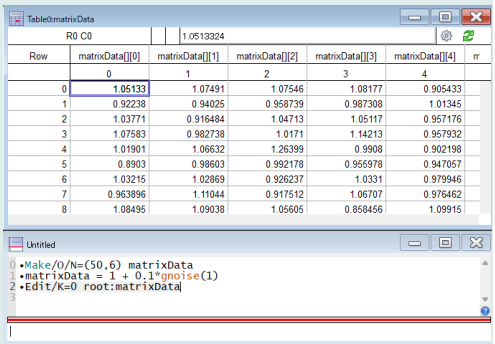
はじめに

新規の Experiment からの手順を説明しますが、データはサンプルの Experiment からコピーして使っても構いあません。

手順

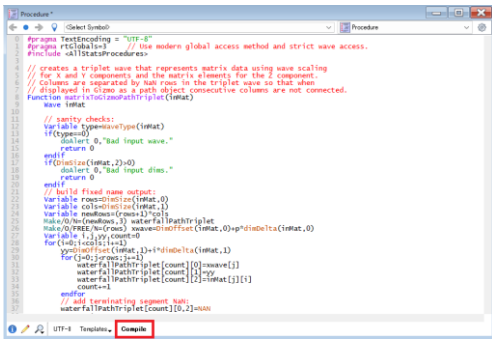
1. コマンドラインで次を実行してサンプルデータ（行列データ）を作ります。

```
Make/O/N=(50,6) matrixData
matrixData = 1 + 0.1*gnoise(1)
```



2. メニュー Windows → Procedure Windows → Procedure Window を開き、サンプルの Experiment からプロシージャをすべてコピーします。

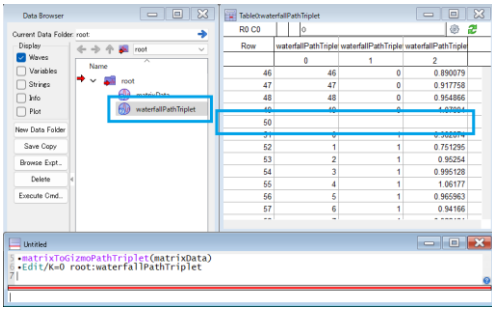
コピーしたら、ウィンドウ下部の Compile ボタンをクリックします。



3. 行列データをトリプレットウェーブに変換します。
Gizmo のパス（Path）オブジェクトには、トリプレットウェーブ（3 列ウェーブで、各行がパスの頂点の XYZ 座標を含む）が必要です。
各列は、異なるパスオブジェクトとして、または単一のパスオブジェクトの異なるセグメントとして表示されるべきです。
パスのトリプレットに NaN 値の行を挿入することで、セグメントを分割することができます。

4. [Step 1] 入力行列からトリプレット waterfallPathTriplet を作成するには、メインプロシージャファイルの関数 matrixToGizmoPathTriplet(inMat) を使います。
コマンドウィンドウで次を実行します：

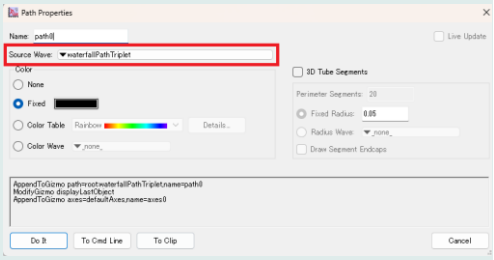
```
matrixToGizmoPathTriplet(matrixData)
```



waterfallPathTriplet というウェーブ（パスオブジェクト）が作成され、内容を見ると 50 行ごとに NaN の行が挿入されています。

5. [Step 2] パスオブジェクトから Gizmo プロットを作成します。

メニュー Windows → New → 3D Plot を選択し、メニュー Gizmo → Append Path を選択します。
Source Wave が waterfallPathTriplet になっていることを確認してください。



GUI を使わない場合は、以下のコマンドを実行します。

```
NewGizmo
AppendToGizmo defaultPath=root:waterfallPathTriplet
```

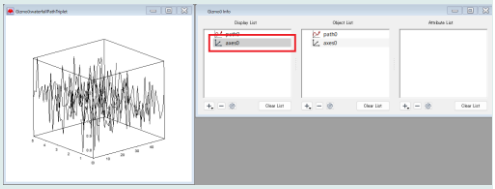
6. **[Step 3]** 軸を整理します（前面にある軸を消す）。

GUI を使わない場合は、以下のコマンドを実行すると一度に処理できます（GUI を使うときはこの手順は不要です）。

```
ModifyGizmo ModifyObject=axes0,objectType=Axes,property={3,ticks,3}
ModifyGizmo ModifyObject=axes0,objectType=Axes,property={2,visible,0}
ModifyGizmo ModifyObject=axes0,objectType=Axes,property={6,visible,0}
ModifyGizmo ModifyObject=axes0,objectType=Axes,property={11,visible,0}
```

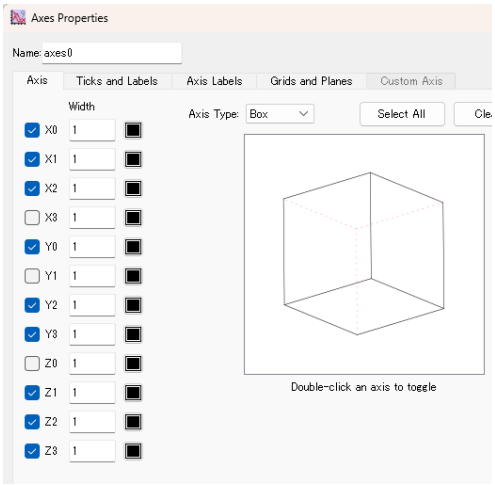
7. GUI での操作の場合は次のようになります。

Gizmo0 コントロールパネルで axes0 をダブルクリックします。

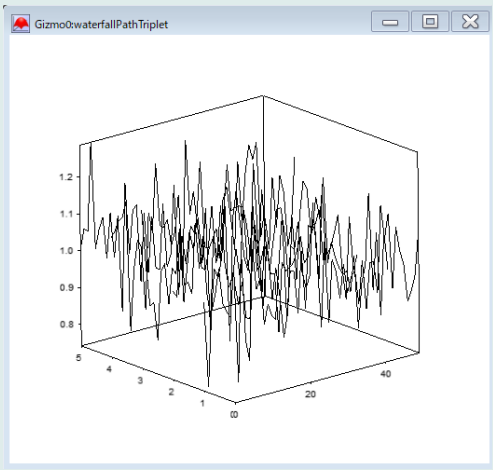


8. **「Axis」タブで、X3、Y1、Z0 のチェックを外します。**

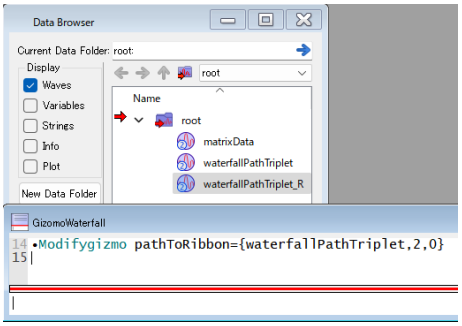
「Ticks and Labels」タブを選択し、Axis ポップアップメニューから Z1 を選択します。
Show Tick Marks チェックボックスと Show Numerical Labels チェックボックスをオンにし、Do It をクリックします。



9. グラフは右図のようになります。



10. **[Step 4]** リボンオブジェクトを追加して、スペクトルに垂直方向の塗りつぶしを適用します。
そのためには、パスをリボンに変換する必要があります。



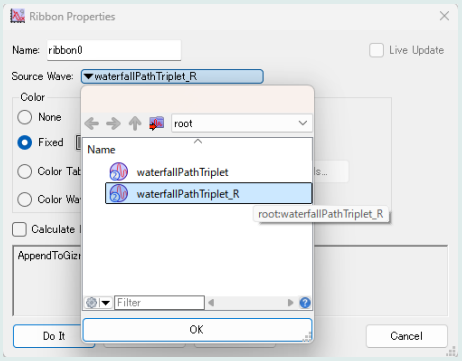
コマンドラインを使う場合は次のように入力します。

```
Modifygizmo pathToRibbon={waterfallPathTriplet,2,0}
```

これで、waterfallPathTriplet_R というウェーブが作成されます。

11. **Gizmo0 Info パネルの Object List 下の「+」マークで Ribbon を選択します。**
Source Wave で waterfallPathTriplet_R を選択します。

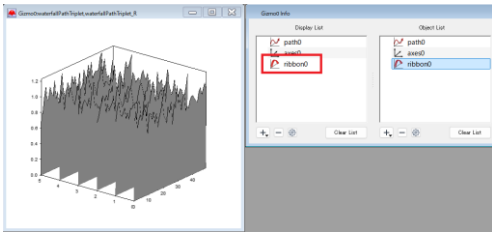
Do It をクリックします。



12. **ribbon0 を Object List から Display List にドラッグして、Display List に追加します。**

コマンドラインでは次のように入力します。

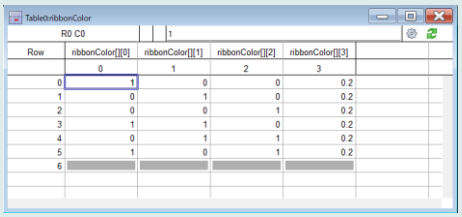
```
AppendToGizmo
    ribbon=root:waterfallPathTriplet_R,name=ribbon0
```



13. **[Step 4]** リボンに色を適用します。

リボン全体に単一の一定した色を適用することもできますし、スペクトルの各平面に異なる色を与えるカラーウェーブを適用して、よりクリエイティブな表現をすることもできます。

ここでは、スペクトル平面のそれぞれに RGBA のエントリーを 1 つずつ含むウェーブを作ることから始めます。



コマンドウィンドウで次を実行します。

```
Make/O/N=(6,4) ribbonColor
Edit ribbonColor
```

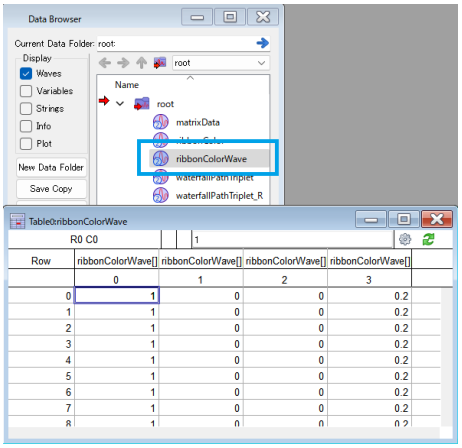
このテーブルにサンプルの `Experiment` からデータをコピーします。

14. リボンオブジェクトのカラーウェーブを作成するには、関数 makeColorWaveForRibbon() を使って、各スペクトル平面でトリプレットの値をベースとした色を生成します。

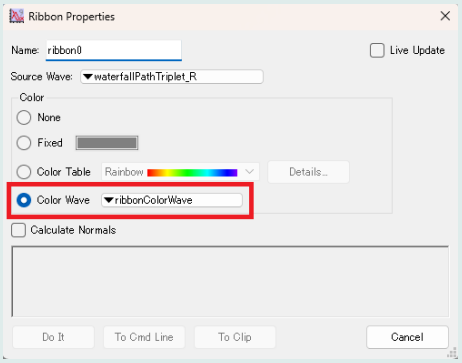
コマンドウィンドウで次を実行します。

```
makeColorWaveForRibbon(waterfallPathTriplet_R,ribbonColor)
```

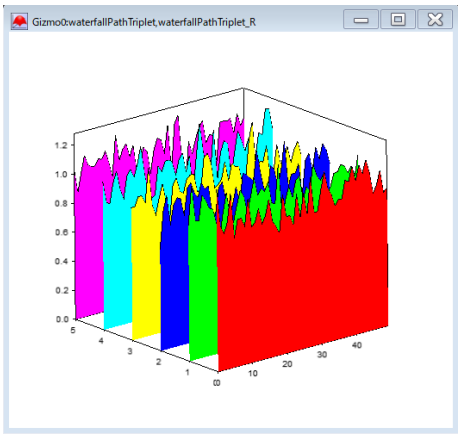
ribbonColorWave というウェーブが生成されます。



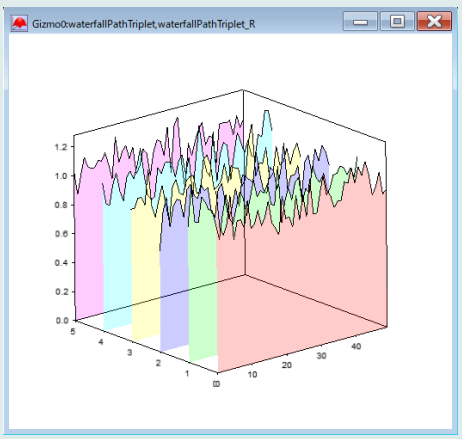
15. **Gizmo0 Info パネルで ribbon0 をダブルクリックして、Ribbon Properties の Color セクションで Color Wave ラジオボタンクリックし、ribbonColorWave を選択します。**
Do It をクリックします。



16. **[Step 6]** さらに次に進みます。
リボンが不透明な塗りつぶしを使っているため、スペクトルの表示は理想的ではありません。
リボンを半透明にして表示するには、Gizmo メニューから「Enable Transparency Blend」を選択します。



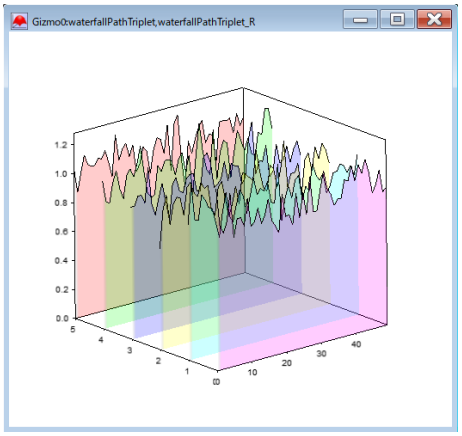
17. かなり良くなりましたが、表示にまだ問題があることは明らかであり、これはスペクトル平面が描画される順序と関係があります。
Z 軸を中心に表示を回転させるとそのことがわかります。
Z 軸が左側に来るように表示を回転させると、リボンウェーブのデータを簡単に反転させることができます。
ここでは、コマンドで反転させてみます。



18. **コマンドウィンドウで次を実行します。**

```
MatrixOP/O
    waterfallPathTriplet_R=reverseCols(waterfallPathTriplet_R)
```

一般的には、リボンカラーウェーブも反転させるべきですが、このケースではすべてのセグメントが同じサイズなので、カラーウェーブの順序を反転させる必要はありません。

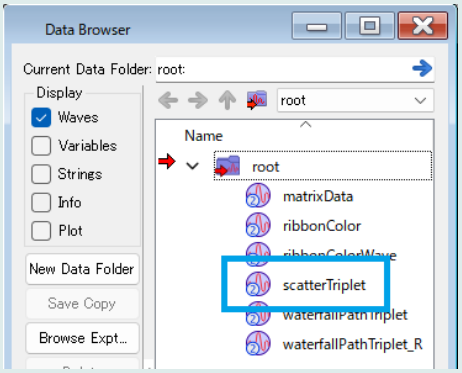


19. この表示に追加したいデータセット（例えば、間引きした実験測定値）がある場合は、トリプレットウェーブ表示を使って追加することができます。

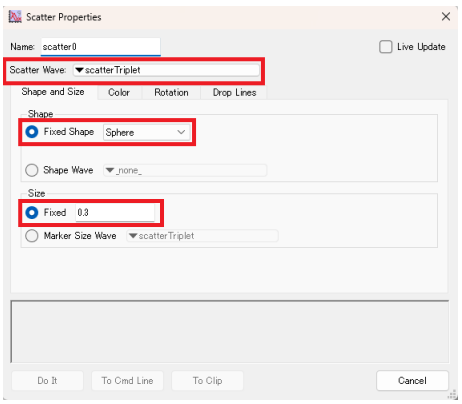
例えば、コマンドウィンドウで次を実行します。

```
makeSampleScatterFromMatrix(matrixData,10)
```

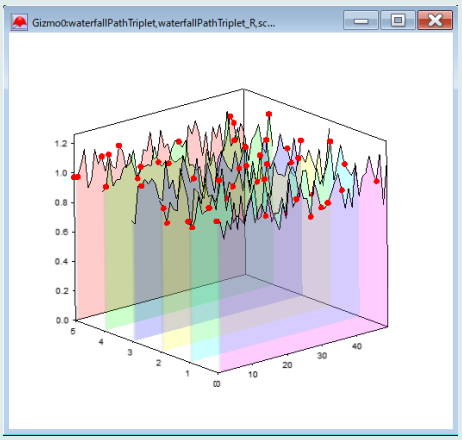
scatterTriplet というウェーブが生成されます。
10 は間引きの度合いです（表示するポイントの数）。



20. **Gizmo0 Info パネルの Object List の下の「+」マークで、Scatter を選択します。**
Scatter Wave で、scatterTriplet を選択し、Shape セクションの Fixed Share を Sphere、Size セクションの Fixed を 0.3 にして Do It をクリックします。

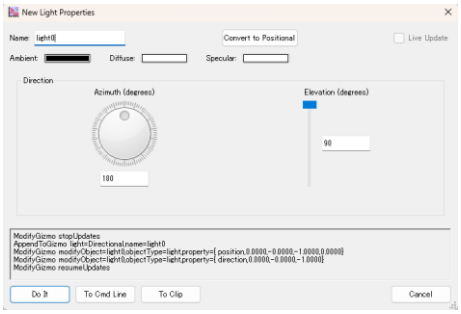


21. **scatter0 を Object List から Display List にドラッグします。**



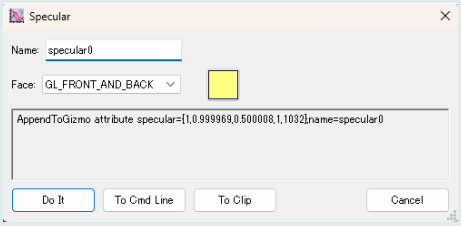
22. 散布図のポイントはソリッドな Sphere なので、サンプルのように光を当てます。

Gizmo0 Info パネルの Object List の下の「+」マークで、Light を選択します。
ライトの位置は適当に決めます（例では Azimuth は 70、Elevation は -45）。



23. 次に、ライトの属性（Attribute）を決めます。

Gizmo0 Info パネルの Attribute List の下の「+」マークで、Specular を選択します。
ポップアップメニューから GL_FRONT_AND_BACK を選択し、色を黄色にします。

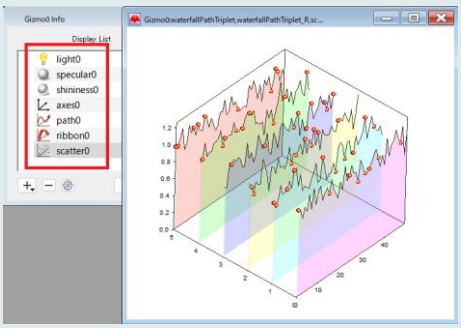


24. **次に、同様に Shininess を選択します。**
Front Shininess を 5 に、Back Shininess を 20 に設定します。

25. **Object List の light0、Attribute List の specular0、shininess0、blendFunc0 を Display List に移動します。**

適用される順番を示すため、Display List 内の順番は重要です。
上から
light0
specular0
shininess0
axes0…
のように並べてください。

結果は右図のようなグラフになります。



プロシージャの内容

メニュー Windows → Procedure Windows → Procedure Window 内のコードを掲載します。

```
// X と Y のコンポーネントにウェーブスケーリングを、Z のコンポーネントにマトリックス要素を使用して、
// マトリックスデータを表すトリプレットウェーブを作成。
// トリプレットウェーブでは、列は NaN 行によって分離されているため、Gizmo にパスオブジェクト
// として表示された場合、連続するカラムは連結されない。
Function matrixToGizmoPathTriplet(inMat)
        Wave inMat
```

```
        // サニティチェック（明らかな不整合がないかチェック）：
        Variable type=WaveType(inMat)
        if(type==0)
                doAlert 0,"Bad input wave."
                return 0
        endif
        if(DimSize(inMat,2)>0)
                doAlert 0,"Bad input dims."
                return 0
        endif

        // 固定名の出力を構築：
        Variable rows=DimSize(inMat,0)
        Variable cols=DimSize(inMat,1)
        Variable newRows=(rows+1)*cols
        Make/O/N=(newRows,3) waterfallPathTriplet
        Make/O/FREE/N=(rows) xwave=DimOffset(inMat,0)+p*dimDelta(inMat,0)
        Variable i,j,yy,count=0
        for(i=0;i<cols;i+=1)
                yy=DimOffset(inMat,1)+i*dimDelta(inMat,1)
                for(j=0;j<rows;j+=1)
                        waterfallPathTriplet[count][0]=xwave[j]
                        waterfallPathTriplet[count][1]=yy
                        waterfallPathTriplet[count][2]=inMat[j][i]
                        count+=1
                endfor

                // セグメントを切るために NaN を追加：
                waterfallPathTriplet[count][0,2]=NAN
                count+=1
        endfor
End


// リボンを表すトリプレットウェーブを取り、colData で渡された個々の色を使って、
// 各リボンセグメントに一定の色を提供する。
// セグメントの数が指定された色数を超える場合、残りのセグメントには最後の色が使用される。
Function makeColorWaveForRibbon(inRibbon,colData)
        Wave inRibbon,colData

        // サニティチェック（明らかな不整合がないかチェック）：
        Variable rows=DimSize(inRibbon,0)
        if(rows<=0 || dimSize(inRibbon,1)!=3)
                doAlert 0,"Bad dimensionality for ribbon wave."
                return 0
        endif
        Variable colorRows=DimSize(colData,0)
        if(colorRows<=0 || dimSize(colData,1)!=4)
                doAlert 0,"Bad dimensionality for colData wave."
                return 0
        endif

        // 出力を作成：
        Make/O/N=(rows,4) ribbonColorWave=0  // NaN の行には alpha をゼロに設定。
        Variable i,segmentCounter=0
        for(i=0;i<rows;i+=1)
                // 各セグメントごとに色を変える：
                if(segmentCounter<colorRows && numType(inRibbon[i][0])==2)
```

```
                        do
                                i+=1
                                if(i>=rows)
                                        break
                                endif
                        while(numType(inRibbon[i][0])==2)
                        segmentCounter+=1
                endif
                if(i<rows)
                        ribbonColorWave[i][0,3]=colData[segmentCounter][q]
                endif
        endfor
End


// 行列データからデータをサンプリングしてウェーブを作成。
Function makeSampleScatterFromMatrix(inMat,numScatterPtsPerCol)
        Wave inMat
        Variable numScatterPtsPerCol

        Variable rows=DimSize(inMat,0)
        Variable cols=DimSize(inMat,1)
        if(cols<2 || rows<2)
                doAlert 0,"Bad dimensionality for inMat."
                return 0
        endif
        Variable outRows=cols*numScatterPtsPerCol,i,j,index,counter=0
        Make/O/N=(outRows,3) scatterTriplet
        for(i=0;i<cols;i+=1)
                for(j=0;j<numScatterPtsPerCol;j+=1)
                        index=trunc(abs(enoise(rows)))
                        scatterTriplet[counter][0]=DimOffset(inMat,0)+index*dimDelta(inMat,0)
                        scatterTriplet[counter][1]=DimOffset(inMat,1)+i*dimDelta(inMat,1)
                        scatterTriplet[counter][2]=inMat[index][i]
                        counter+=1
                endfor
        endfor
End
```