

Process Modeling Using Inference for .NET and Dew Research MtxVec Numerical Libraries

What is process modeling?

Process modeling entails using computational methods to describe, represent and simulate the functioning of real-world scientific and engineering processes. Computer modeling represents a complementary method to laboratory. The purposes of modeling include:

- analysis and understanding of observed phenomena;
- testing hypothesis and theories;
- predicting process outputs under conditions which are too difficult or too expensive to measure directly;
- calibration of measurement systems; and
- process optimization.

Process modeling describes the variation in one observed quantity (response) as a function of a deterministic component, typically a mathematical function of one or more quantities (factors), and a random component (noise), typically described by a probability distribution. For example, the variation in the measured expansion of a metal like copper can be described by partitioning the variability into a deterministic part, which is a function of temperature, and some left-over random error.

How does one build process models?

Process models are built by fitting experimental data to mathematical functions. Linear least squares regression is the most widely used modeling method. When people talk of using "regression", "linear regression" or "least squares," they are fitting their data to empirical models.

Nonlinear least squares regression extends linear least squares regression for use with a much larger and more general class of empirical models. Almost any closed form mathematical function can be incorporated in a nonlinear regression model. And, unlike linear regression, there are very few limitations on the way parameters can be used in the functional part of a nonlinear regression model.

Polynomial models are among the most frequently used empirical models for fitting functions. A rational function model is a generalization of the polynomial model. Rational function models contain polynomial models as a subset. For example, the function below is a quadratic/quadratic rational function model that can be used to describe the coefficient of thermal expansion of copper metal as a function of temperature,

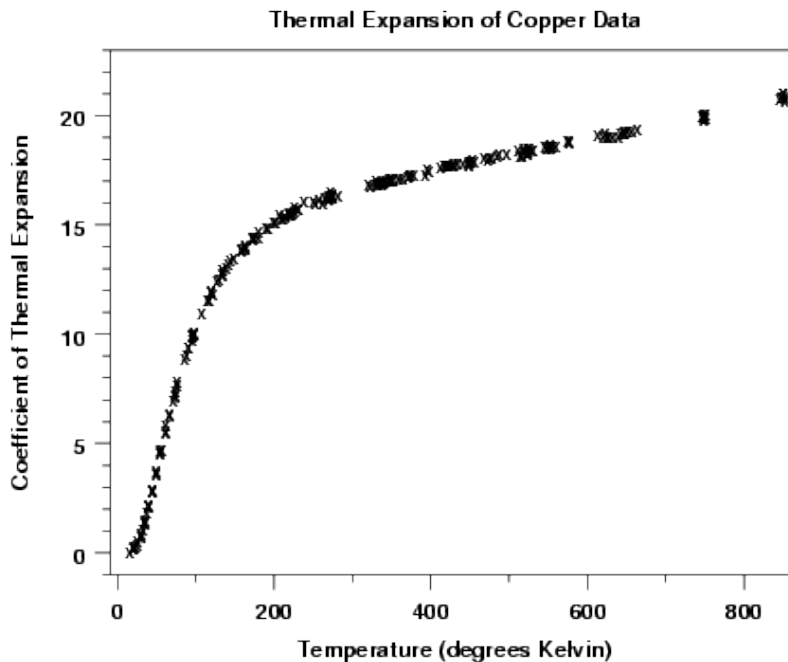
$$Y = \frac{\beta_0 + \beta_1 X + \beta_2 X^2}{1 + \beta_3 X + \beta_4 X^2}$$

where Y corresponds to the coefficient of thermal expansion, X corresponds to the temperature in degrees Kelvin, and β_i corresponds to estimated parameters obtained by nonlinear least squares. Nonlinear least squares parameter estimation is in iterative method which employs estimated starting values.

Case Study: Coefficient of Thermal Expansion of Copper Metal

The data

The data result from a National Institute of Standards and Technology (NIST) experimental study involving the thermal expansion of copper. The response variable is the coefficient of thermal expansion, and the predictor variable is temperature in degrees. Let us look at the data in a simple plot.



The plot exhibits a steep initial slope that levels off to a more gradual slope. This type of response curve can often be modeled with a rational function model, like the quadratic/quadratic rational function model described earlier. The plot further indicates that there appear to be no gross outliers in the data.

Fitting the data using Dew Research MtxVec Numerical Libraries

Fitting the thermal expansion data to the quadratic/quadratic function model requires non-linear fitting. This involves a three-step process: (1) define the fit function, (2) obtain initial estimates of the beta coefficients, and (3) iterate the beta coefficients using the optimization method used in nonlinear

regression. With a great deal of time and effort, one could write code that implemented and executed the algorithms of this three-step process. But there is a much easier way. Using specialized numerical libraries like those supplied by Dew Research in the Inference for .NET environment allows for rapid development and deployment of the solution using some simple IronPython glue code. This is illustrated below.

Define the quadratic/quadratic rational function

We begin with defining the quadratic-quadratic rational function to which the data will be fit:

```
def FracQuadQuad(B, X):
    return
    (ValueAccessor.GetTVecValue(B,0)+X*ValueAccessor.GetTVecValue(B,1)+X*X*ValueAccessor.
    GetTVecValue(B,2))/(1.0 + ValueAccessor.GetTVecValue(B,3)*X +
    ValueAccessor.GetTVecValue(B,4)*X*X)
```

Obtain initial estimates of regression coefficients

Next, we need initial estimates for the regression coefficients i.e. the values for B[0]..B[4]. As suggested at NIST [pages](#), we'll use linear fit on selected points to get initial estimates for regression coefficients. For this we can use the MullinRegress routine available in the MtxVec library. Here is the implementation:

```
from Dew.Stats import *
from Dew.Stats.Units import *
from Dew.Math import *
from InferenceDataAccess import ValueAccessor
from ZedGraph import SortType
```

Define the function to perform an initial estimate on fractional model by performing a linear fit:

```
def FracInitEstimates(x, y, b, nom, denom):
    m1 = Matrix()
    m2 = Matrix()
    m3 = Matrix()

    m1.VanderMonde(nom+1,x)

    m1.FlipHor()

    m2.VanderMonde(denom+1,x)

    m2.Resize(m2.Rows,denom)
    m2.FlipHor()

    for i in range(m2.Rows):
        m2.Mul(ValueAccessor.GetVectorValue(y,i),i*m2.Cols,m2.Cols)

    m3.ConcatHorz(m1,m2)
    Regress.MullinRegress(y,m3,b,False,None,None)
```

Perform non-linear least squares fit

Now all we must do is put all this together and perform the non-linear fit. Towards this end, we will use Dew Stats `TMtxNonLinReg` control. The call to `FitModel()` routine first loads data, then performs initial estimation of regression coefficients and then uses initial estimates in actual non-linear regression. Here is the implementation:

```
nlr = TMtxNonLinReg()
residuals = Vector()
initx = Vector()
inity = Vector()

try:
    nlr.X.LoadFromFile("C:\\Temp\\nist_001_temp.vec")
    nlr.Y.LoadFromFile("C:\\Temp\\nist_001_coeff.vec")

    xPointsOrig = ValueAccessor.GetTVecArray(nlr.X)
    yPointsOrig = ValueAccessor.GetTVecArray(nlr.Y)

    # connect to regression function
    nlr.RegressFunction += FracQuadQuad

    # Create the arrays.
    x = [10,50,120,200,800]
    x = Array[float](x)
    y = [0,5,12,15,20]
    y = Array[float](y)

    # initial values for coefficients
    initx.SetIt(x)
    inity.SetIt(y)

    FracInitEstimates(initx,inity,nlr.B,2,2)
    # nlr.B returns 0.301e+1,0.369,-0.683e-2,-0.112e-1,-0.306e-3]

    # Perform non-linear regression
    nlr.Recalc()
    # Calculate residuals
    residuals.Sub(nlr.YCalc,nlr.Y)

    # Record the points for graphing.
    xPoints = ValueAccessor.GetTVecArray(nlr.X)
    yPoints = ValueAccessor.GetTVecArray(nlr.YCalc)
    yPointsOrig = ValueAccessor.GetTVecArray(nlr.Y)

finally:
    nlr.Free()
```

Plot the fit on top of the data

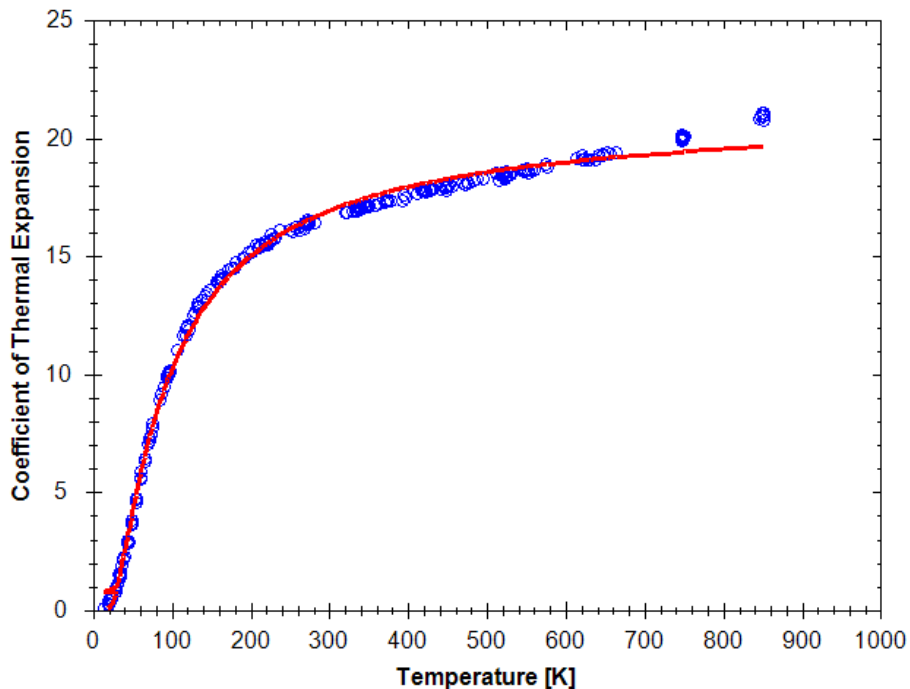
Now let take a look at the results. We create a plot that compares the experimental thermal coefficient values (symbols) with the corresponding calculated values (line) based on the non-linear fit. Here is the implementation:

```
# Create the graph pane.
graphPane = InferenceGraph.CreateGraphPane("", "Temperature [K]", "Coefficient of
Thermal Expansion")

# Plot the calculated data based on the non-linear fit.
#
# Ready the data.
ppl = InferenceGraph.CreatePointPairList(xPoints, yPoints)
ppl.Sort(SortType.YValues)
# Plot.
line = graphPane.AddCurve("", ppl, Color.Red)
line.Symbol.Type = SymbolType.None
line.Line.Width = 3

# Plot the original, experimental data.
#
# Ready the data.
ppl = InferenceGraph.CreatePointPairList(xPoints, yPointsOrig)
ppl.Sort(SortType.YValues)
# Plot.
line = graphPane.AddCurve("", ppl, Color.Blue)
line.Symbol.Type = SymbolType.Circle
line.Line.IsVisible = False

InferenceGraph.CreateInferenceFigureAsPNG(graphPane)
```



Conclusion

We conclude that the quadratic/quadratic rational function model does in fact provide a reasonable empirical model for this data set. Further evaluation would reveal that a cubic/cubic rational function model does even a bit better.