"VECTORS" AS COLUMNS OF DATA

Often Mathcad "vectors" are used to simply hold columns of data values. Obviously, those columns do not normally have anything to do with a vector in the normal sense. For example, dot products and cross products of such data columns are usually meaningless, as are multiplication by matrices, rotation, eigenvalues, etcetera.

To illustrate this, assume that we have placed a hot bar with its ends in contact with ice water. The temperature of the bar will then decay over time to 0 degrees Centigrade. We have measured the temperature in the final decay stages at 6 times spaced half a minute apart. Taking the first of these times as time zero, the measured (subscript m) times are:

$ORIGIN := 1 \qquad i := 1..6 \qquad t_{m_i} := (i-1)\cdot 0.5\,min$

$i =$

1
2
3
4
5
6

$$t_m = \begin{pmatrix} 0 \\ 0.5 \\ 1 \\ 1.5 \\ 2 \\ 2.5 \end{pmatrix} \cdot min$$

Subscript m means Measured.

Note that i is a range variable but t_m a vector. That allows the times to be unequally spaced, if needed. And you can change a time, if it is wrong.

Unknown to us, the exact temperature in this range of times is given by (open the measuring cup to get the degrees Centigrade):

$$T_x(t) := 0\,°C + \exp\left(-\frac{t}{3min}\right)\cdot(28.7\,°C - 0\,°C)$$ Subscript x means eXact

Our thermometer is very well calibrated, and measures the 6 temperatures with negligible error. But its digital display only shows the temperatures rounded to whole degrees. That means that the temperatures that we measure will be:

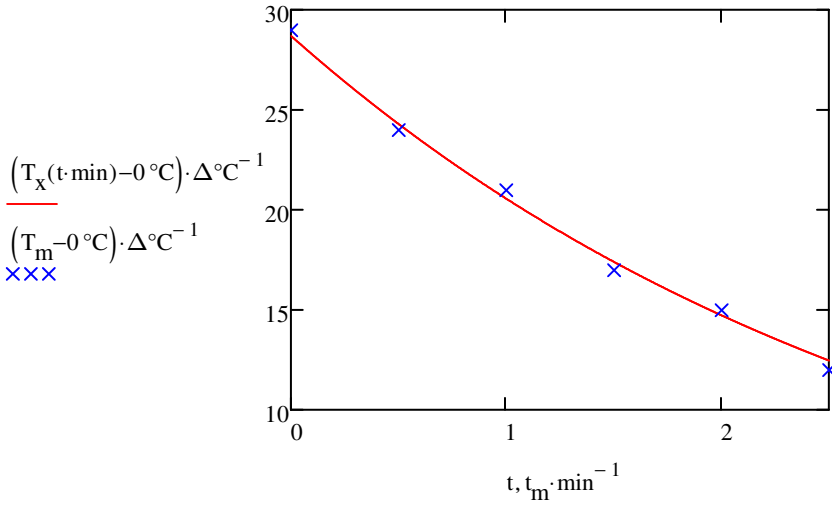
$$T_m := round\left[\left(T_x\left(t_m\right) - 0\,°C\right)\cdot \Delta°C^{-1}\right]\Delta°C + 0\,°C$$

So the only thing *we* know is the following measured times and temperatures:

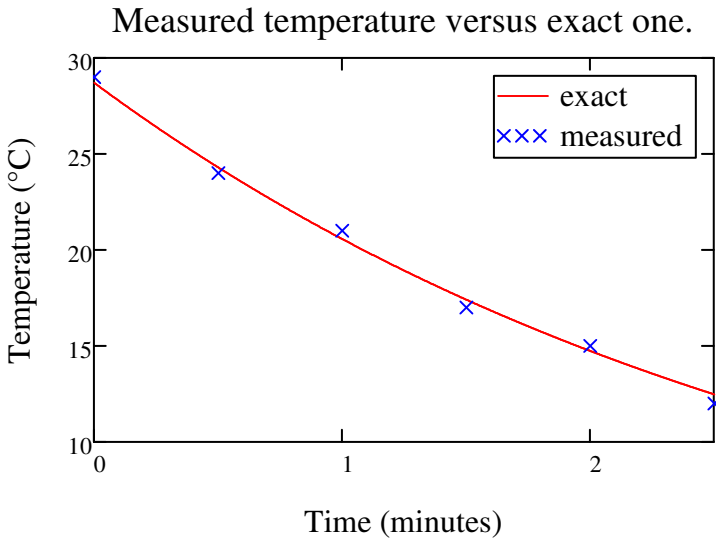
$$t_m = \begin{pmatrix} 0 \\ 0.5 \\ 1 \\ 1.5 \\ 2 \\ 2.5 \end{pmatrix} \cdot min \qquad T_m = \begin{pmatrix} 29 \\ 24 \\ 21 \\ 17 \\ 15 \\ 12 \end{pmatrix} .°C$$

Note that T_m could not even be a range variable, because the values are not equally spaced. Also, you *want* it to be a "vector", so that you can enter and modify the individual temperatures.

Let's plot the measured data against the exact temperature:



The presentation quality version would be:



Note the errors caused by rounding the temperature to whole degrees. Let's quantify these errors. The individual errors are:

$$err_m := T_m - T_x(t_m)$$

$$err_m = \begin{pmatrix} 0.3 \\ -0.294 \\ 0.436 \\ -0.407 \\ 0.265 \\ -0.473 \end{pmatrix} K$$

In quantifying these errors, we are interested in the *absolute values* of the individual errors. But to get those, it is not enough to enclose err_m between absolute signs, by typing |err.m=:

$|err_m| = 0.909\,K$ Mathcad gives us the *length* of the 6-dimensional vector err_m!

To get the individual absolute errors, as a "vector", we must *Vectorize* the absolute sign operation using Ctrl+-. Try typing |err.mSPACECtrl+-=:

$$\overrightarrow{|err_m|} = \begin{pmatrix} 0.3 \\ 0.294 \\ 0.436 \\ 0.407 \\ 0.265 \\ 0.473 \end{pmatrix} K$$

Use the "Vectorize" operator to apply operations for each element separately, (producing a vector result), instead of in the normal vector way.

Now we can let Mathcad find the largest of these errors using the "max" function.

$\max\left(\overrightarrow{|err_m|}\right) = 0.473\,K$ cannot exceed 0.5, of course

Or it can find the average, by summing them (sum in the vector and matrix toolbar) and dividing that sum by the number of terms. The number of terms is available as the "last" function, which really gives the index of the last term of the vector (6 in this case):

$$\frac{\sum \overrightarrow{|err_m|}}{last(err_m)} = 0.362\,K$$ smaller than the maximum error, of course

The very useful "last" function gives you the index value of the last element of a vector.

For various reasons, it is often more convenient to find the "root-mean-square" (rms) error instead of the plain average error above. To find the rms error, you first find the average *square* error, then take a square root of that. The plain dot product of err_m with itself already gives you the sum of the squares of the errors. So you only need to divide that by the number of terms and take a square root:

$$\sqrt{\frac{err_m \cdot err_m}{last(err_m)}} = 0.371\,K$$ always in between the average and maximum error

Linear interpolation

Suppose, say, that you want the temperature after 85 seconds. You did not measure the temperature at that time. And you do not know the exact solution. To get a value for the temperature at 85 s, you must use some form of "interpolation".Linear interpolation is the simplest form of interpolation, and the most robust one. Mathcad uses linerp to do it. Let's first define a function to evaluate the linear interpolation and the desired time:

$T_{li}(t) := \text{linterp}(t_m, T_m, t)$ li means linear interpolation

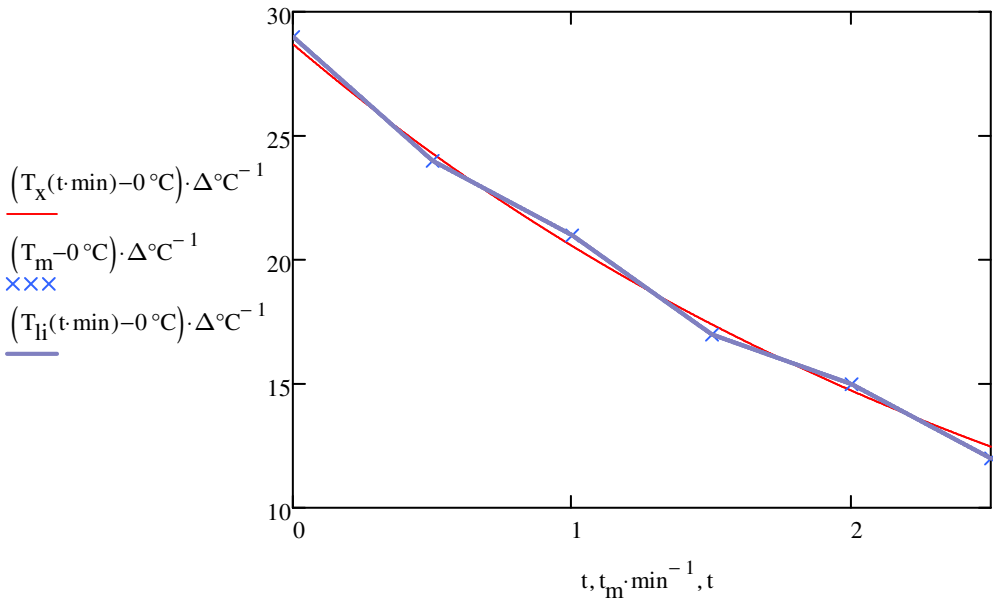
Now print out the linearly interpolated temperature at the desired time:

$T_{li}(85s) = 17.667 \cdot °C$

The error in this approximation is:

$T_{li}(85s) - T_x(85s) = -0.231 \cdot \Delta°C$ Use the measuring cup for $\Delta°C$.

Let's plot the linear interpolation versus the measurements and exact temperature (copy over the previous plot and modify):



Let's also check the errors in general. To do so, we will use 101 times spread out over the entire measured range;

$i := 1 .. 101$ $t_{c_i} := (i - 1) \cdot 0.025min$ the c means check $err_{li} := T_{li}(t_c) - T_x(t_c)$

$\max(\overrightarrow{|err_{li}|}) = 0.473\,K$ $\dfrac{\sum \overrightarrow{|err_{li}|}}{\text{last}(err_{li})} = 0.192\,K$ $\sqrt{\dfrac{err_{li} \cdot err_{li}}{\text{last}(err_{li})}} = 0.225\,K$

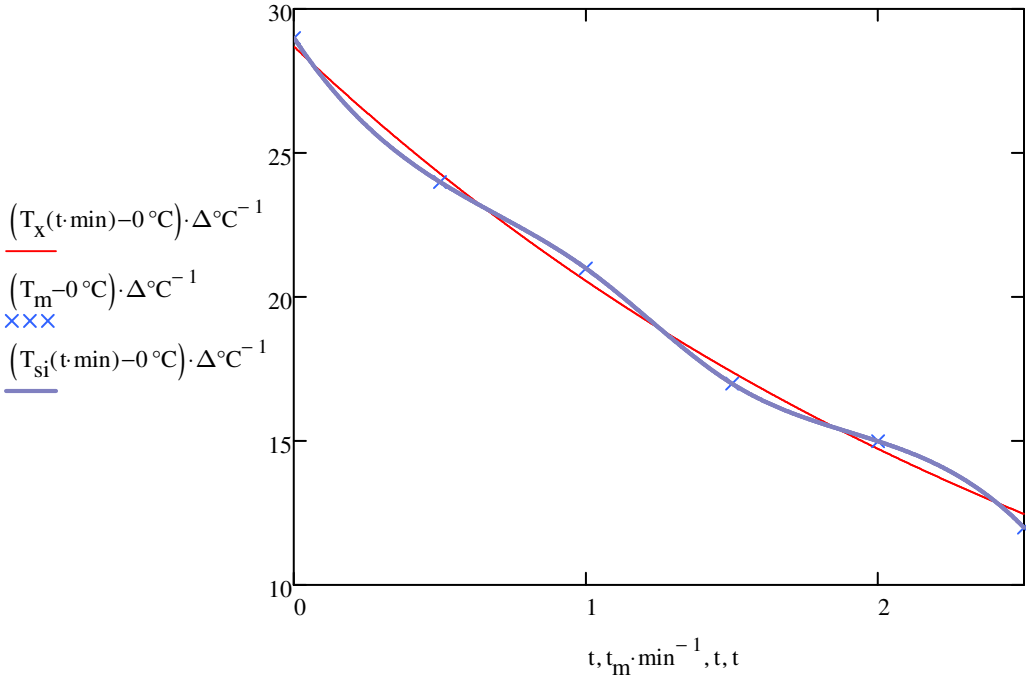
Spline interpolation

Since the temperature curve is nonlinear, you would *normally* get better results using a cubic spline. This uses curved cubics instead of straight lines between measured points:

$spline := cspline(t_m, T_m)$ You must create the spline first.

$T_{si}(t) := interp(spline, t_m, T_m, t)$ Create a function to simplify evaluation; si is for spline interpolation.

Unfortunately, the "random" round-off errors in the temperatures make the error in the spline value *greater* than that in linear interpolation (copy and paste from above and change li into si):



The smooth spline, in trying to weave through all these inaccurate data points, swings further away from the correct solution than the linear interpolation.

$$err_{si} := T_{si}(t_c) - T_x(t_c)$$

$\max(\overrightarrow{|err_{si}|}) = 0.489\,K$

$\frac{\sum \overrightarrow{|err_{si}|}}{last(err_{si})} = 0.278\,K$

$\sqrt{\frac{err_{si}\cdot err_{si}}{last(err_{si})}} = 0.311\,K$

Normally spline interpolation will be very much more accurate that linear interpolation for a smooth, nonlinear function.
But if the data has significant random errors, that is not necessarily true.

Extrapolation

How about finding the temperature at later times than 2.5 minutes? It is supposed to go to zero for large enough times? Let's see what we get.

$T_{li}(5\text{min}) = -3\cdot°\text{C}$ Bad, the temperature should not become negative.

$T_{si}(5\text{min}) = -193\cdot°\text{C}$ Much, much worse still.

Be very, very, careful when evaluating approximating functions outside the range of the measured data!

Linear least square approximation

If we want a more accurate approximation of the true temperature, (while still not knowing the exact solution), we must stop demanding that the temperature curve goes *exactly* through those *nonexact* measured points.

So what we will do is:
1) Prescribe some non-swinging, smooth curve with some adjustable coefficients in it.
2) Find these coefficients from the demand that the root-mean-square error from the measured values is as small as possible.

The above procedure called the "method of least squares". We will indicate it as "ls". You may also find the method referred to as "linear regression," but that is really a somewhat more general term.

As a first try, let's take the non-swinging curve to be a straight line:

$$T_{lls} = C_0 + C_1\, t$$

Constant C_0 is called the "intercept" of the line, because that is the y-value where the line intercepts the T-axis. C_1 is called the slope for reasons given in Calculus 1. Mathcad will give you these two coefficients using functions of those names. But it is shorter to use the "line" function that gives you the two coefficients as a vector:

ORIGIN := 0 It is now more concenient to number the coefficients from 0

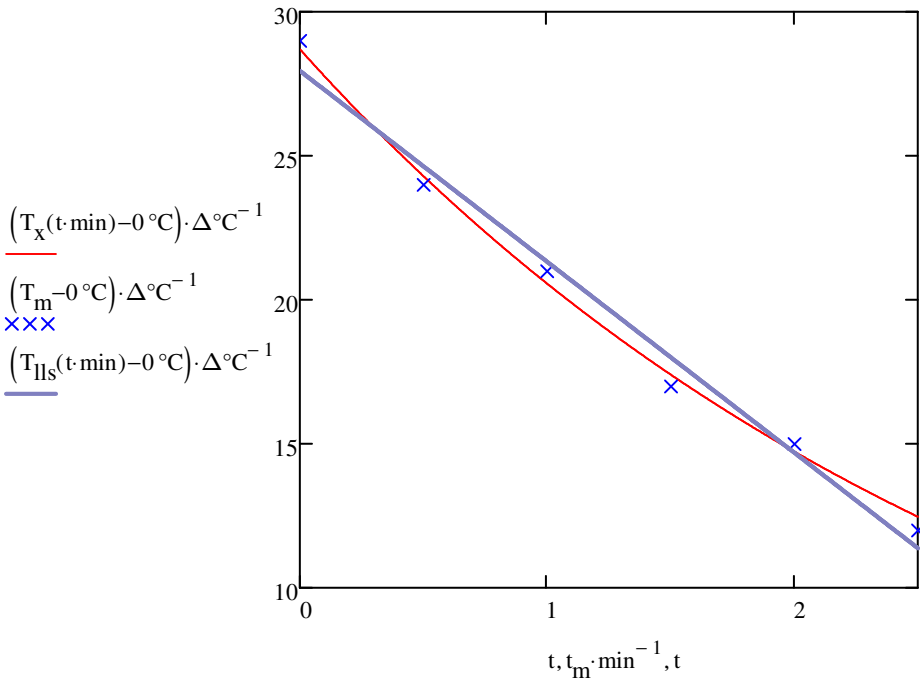
$$C_{lls} := \text{line}\left(t_m \cdot \text{min}^{-1}, T_m \cdot \text{K}^{-1}\right) = \begin{pmatrix} 301.102 \\ -6.629 \end{pmatrix}$$ *Note: do not use units!*

$$T_{lls}(t) := \left(C_{lls_0} + C_{lls_1} \cdot t \cdot \text{min}^{-1}\right) \cdot \text{K}$$

The "stderr" function gives you some average error in the approximation of the data points:

$$\text{stderr}\left(t_m \cdot \text{min}^{-1}, T_m\right) = 0.881\,\text{K}$$

That is bad, because the actual error in the measured data is less than half a degree. The temperature is not well approximated by a straight line in the measured region. To verify, let's compute errors and plot as before:



$$err_{lls} := T_{lls}(t_c) - T_x(t_c)$$

$$\max\left(\overrightarrow{|err_{lls}|}\right) = 1.092\,K$$

$$\frac{\sum\overrightarrow{|err_{lls}|}}{last(err_{lls}) + 1} = 0.498\,K$$

$$\sqrt{\frac{err_{lls}\cdot err_{lls}}{last(err_{lls}) + 1}} = 0.56\,K$$

Higher order least square approximation

We can get a better approximation if we approximate the temperature by a quadratic (parabola) instead of a straight line:

$T_{qls} = C_0 + C_1\, t + C_2\, t^2$

Such a function is called a *linear combination of the 1, t, and t^2 functions.*

You might say that instead of taking one term in the Taylor series, we take two. But actually, *any* function can be accurately approximated by a polynomial (power series) over a finite interval. It does not have to have a converging Taylor series on that interval.

First we must tell Mathcad that we want to approximate with a linear combination of 1, t, and t^2. That is done by defining a vector function of them:

$$\text{funcs}_{qls}(t) := \begin{pmatrix} 1 \\ t \\ t^2 \end{pmatrix}$$

After all, if your variable would be periodic of a known period, you would probably want to approximate with sines and/or cosines of that period, instead of with powers.

Finding the quadratic regression now proceeds as follows:

$$C_{qls} := \text{linfit}\left(t_m \cdot \text{min}^{-1}, T_m \cdot K^{-1}, \text{funcs}_{qls}\right) = \begin{pmatrix} 301.936 \\ -9.129 \\ 1 \end{pmatrix}$$

$$T_{qls}(t) := \left[C_{qls_0} + C_{qls_1} \cdot t \cdot \text{min}^{-1} + C_{qls_2} \cdot \left(t \cdot \text{min}^{-1}\right)^2\right] \cdot K$$

$$\text{err}_{qls} := T_{qls}(t_c) - T_x(t_c)$$

Note: if you evaluate using a dot product between C_{qls} and funcs_{qls}(t min), you need to evaluate err_{qls} using a range subscript i on it and t_c.

$$T_{qls}(t) := C_{qls} \cdot \text{funcs}_{qls}\left(t \cdot \text{min}^{-1}\right) \cdot K$$

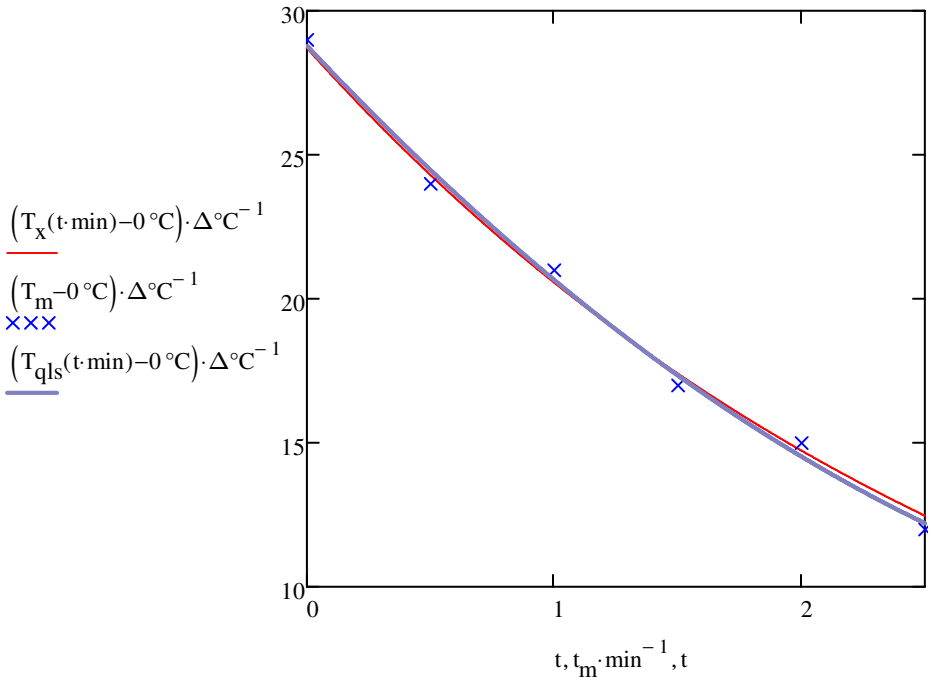
$$i := 0 .. \text{last}(t_c) \qquad \text{err}_{qls_i} := T_{qls}\left(t_{c_i}\right) - T_x\left(t_{c_i}\right)$$

$$\max\left(\overrightarrow{\left|\text{err}_{qls}\right|}\right) = 0.259\,K$$

$$\frac{\sum \overrightarrow{\left|\text{err}_{qls}\right|}}{\text{last}\left(\text{err}_{qls}\right) + 1} = 0.144\,K$$

$$\sqrt{\frac{\text{err}_{qls} \cdot \text{err}_{qls}}{\text{last}\left(\text{err}_{qls}\right) + 1}} = 0.16\,K$$

That is a lot better. A quarter of a degree error is pretty good.

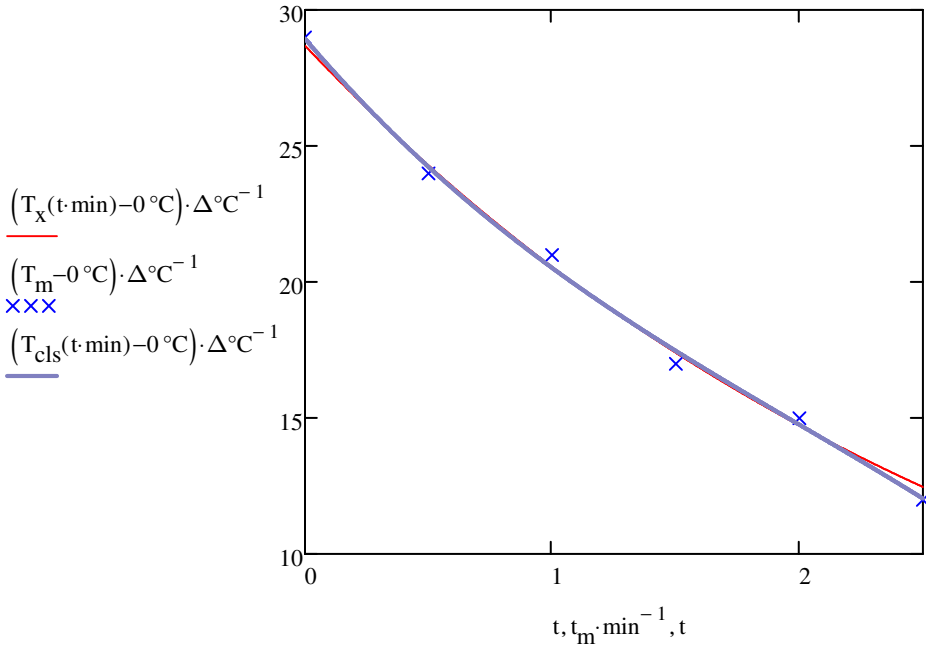

$\left(T_x(t\cdot\min)-0\,°C\right)\cdot\Delta°C^{-1}$
$\left(T_m-0\,°C\right)\cdot\Delta°C^{-1}$
$\left(T_{qls}(t\cdot\min)-0\,°C\right)\cdot\Delta°C^{-1}$
30
25
20
15
10
0
1
2
$t, t_m\cdot\min^{-1}, t$

Try cubic:

$$\text{funcs}_{cls}(t) := \begin{pmatrix} 1 \\ t \\ t^2 \\ t^3 \end{pmatrix}$$

$$C_{cls} := \text{linfit}\left(t_m \cdot \text{min}^{-1}, T_m \cdot K^{-1}, \text{funcs}_{cls}\right) = \begin{pmatrix} 302.102 \\ -10.651 \\ 2.667 \\ -0.444 \end{pmatrix}$$

$$T_{cls}(t) := \left[C_{cls_0} + C_{cls_1} \cdot t \cdot \text{min}^{-1} + C_{cls_2} \cdot \left(t \cdot \text{min}^{-1}\right)^2 + C_{cls_3} \cdot \left(t \cdot \text{min}^{-1}\right)^3\right] \cdot K$$



$$\text{err}_{cls} := T_{cls}(t_c) - T_x(t_c)$$

$$\max\left(\overrightarrow{|\text{err}_{cls}|}\right) = 0.425\,K \qquad \frac{\sum \overrightarrow{|\text{err}_{cls}|}}{\text{last}(\text{err}_{cls}) + 1} = 0.084\,K \qquad \sqrt{\frac{\text{err}_{cls} \cdot \text{err}_{cls}}{\text{last}(\text{err}_{cls}) + 1}} = 0.118\,K$$

Oops, the maximum error went up again! The rule of thumb is:

The number of coefficients in your least square expression (here 4) should not be larger than roughly the square root of your number of measured points (here 6).

It's not too bad yet, but do not try higher than cubic!

Non-polynomial least square approximation

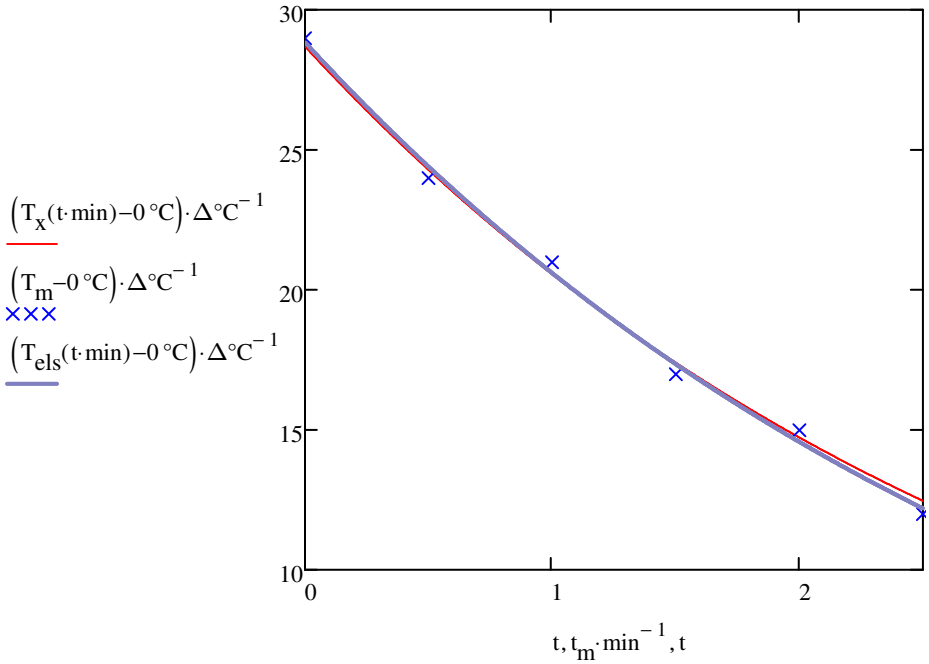
You can often get better results if you use some physical knowledge about your problem. For heat conduction, the final stages will be an exponential decay in time. So it is probably a much better idea to approximate the temperature curve as:

$T = C_0 \exp(C_1 t) + C_2$

That can be done with the "expfit" function:

$$C_{els} := \text{expfit}\left(t_m \cdot \text{min}^{-1}, T_m \cdot \text{K}^{-1}\right) = \begin{pmatrix} 30.667 \\ -0.313 \\ 271.334 \end{pmatrix}$$

$$T_{els}(t) := \left(C_{els_0} \cdot \exp\left(C_{els_1} \cdot t \cdot \text{min}^{-1}\right) + C_{els_2}\right) \cdot \text{K}$$



$$\text{err}_{els} := T_{els}(t_c) - T_x(t_c)$$

$$\max\left(\overrightarrow{\left|\text{err}_{els}\right|}\right) = 0.271\,\text{K}$$

$$\frac{\sum \overrightarrow{\left|\text{err}_{els}\right|}}{\text{last}\left(\text{err}_{els}\right) + 1} = 0.112\,\text{K}$$

$$\sqrt{\frac{\text{err}_{els} \cdot \text{err}_{els}}{\text{last}\left(\text{err}_{els}\right) + 1}} = 0.131\,\text{K}$$

The accuracy is about the same as we get from the quadratic approximation. So that worked pretty well. But note that the third coefficient is not quite right. C_3 should be 273.15, not 271.

If we put that in explicitly, we may get a better solution:

$T = C_0 \exp(C_1 t) + 273.15$

We need to define a function the following way:

$$\text{funcs}_{gls}(t, C_0, C_1) := \begin{pmatrix} C_0 \cdot \exp(C_1 \cdot t) + 273.15 \\ \exp(C_1 \cdot t) \\ C_0 \cdot t \cdot \exp(C_1 \cdot t) \end{pmatrix}$$

The approximation

Its C_0 partial derivative

Its C_1 partial derivative

Watch those partial derivatives! It is easy to start differentiating with respect to t instead of the coefficients.

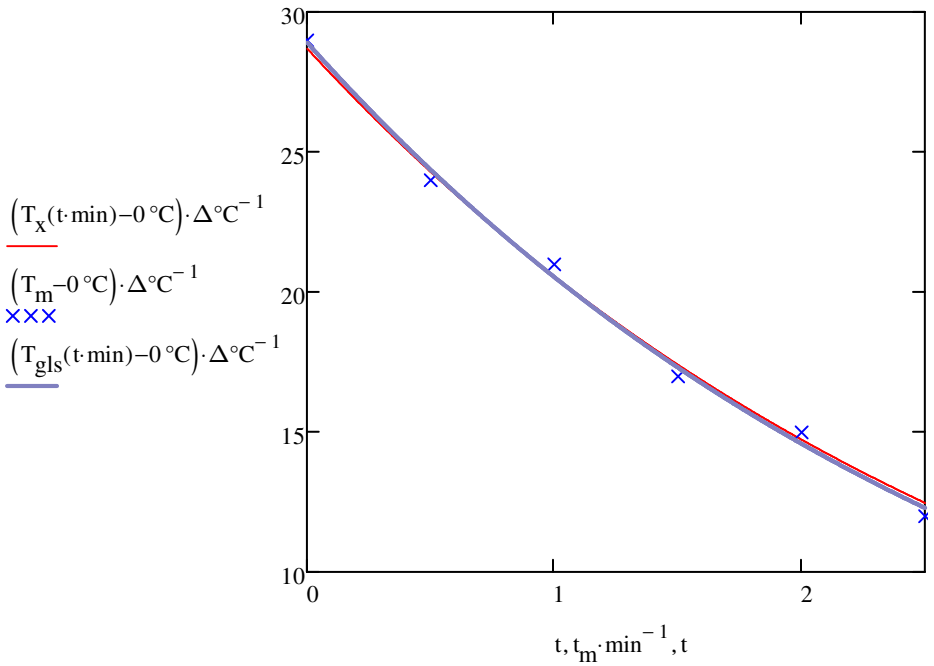
Since expfit does not work any more, we must use the more general "genfit" function. Then you must provide initial guesses for the coefficients. We will use the expfit ones for that.

$$C_{gls_0} := C_{els_0} \qquad C_{gls_1} := C_{els_1}$$

Now we are ready to use genfit:

$$C_{gls} := \text{genfit}\left(t_m \cdot \text{min}^{-1}, T_m \cdot \text{K}^{-1}, C_{gls}, \text{funcs}_{gls}\right) = \begin{pmatrix} 28.916 \\ -0.342 \end{pmatrix}$$

$$T_{gls}(t) := \left(C_{gls_0} \cdot \exp\left(C_{gls_1} \cdot t \cdot \text{min}^{-1}\right) + 273.15\right) \cdot \text{K}$$



$$err_{gls} := T_{gls}(t_c) - T_x(t_c)$$

$$\max\left(\overrightarrow{|err_{gls}|}\right) = 0.216\,K \qquad \frac{\sum \overrightarrow{|err_{gls}|}}{last(err_{gls}) + 1} = 0.106\,K \qquad \sqrt{\frac{err_{gls} \cdot err_{gls}}{last(err_{gls}) + 1}} = 0.12\,K$$

If you compare, you will agree that this is clearly the best solution of all.

Note that the maximum error in the measured data is 0.5 degrees, either way, so the average error would be something like 0.25 degrees. The *maximum* error in the exponential fit is *smaller* than the *average* error in the measured data. Least squares will do that for you. Assuming the errors are random, they will average away in the least square approximation.

Increasing the number of measured points is good for least square approximation.

However, increasing the number of points beyond a certain level will not improve linear interpolation, and will make cubic spline interpolation worse.