Forecasting trigger functions

October 9, 2015

First things first. Forecasting is *not* magic, it's mostly statistics. You've probably heard that "There are three kinds of lies: lies, damned lies, and statistics." Of course, conclusions based on statistics cannot be always exactly correct, but if used *intelligently* statistics can be a very powerful tool. The following Guide will hopefully assist you in using Zabbix predictive capabilities wisely and efficiently. You can also go into Details of the implementation if you feel the need. And if you are really keen on knowing all the underlying mathematics, welcome to the Reference!

1 Guide

1.1 Visualising the predictions

At the present moment we cannot display the graph of expression Zabbix server fits to your data to make predictions (if you really need to see it go straight to the Section 4) but you can see and plot the end result of forecast if you create a calculated item like this:

```
forecast(host:item,1h,,1h)
```

Or like this:

```
timeleft(host:item,1h,,0)
```

Problem is, `forecast()` shows *now* what is expected value of item *after some time*, so it's tricky to compare forecast and original item and conclude whether forecasting works. Create one more calculated item:

```
last(host:item_forecast,#1,1h)
```

Now you can create a graph with both your original item and shifted item forecast. (You will need to wait an hour until `last()` becomes supported.) Figure 1 shows example of what you will see.

Go to WolframAlpha or use your favourite mathematical package (Mathematica, Matlab, GNU Octave, R, Calc or even Excel...) to play with different fits and learn more about different mathematical functions supported by Zabbix. Fitting algorithms are basically the same everywhere, so you will be able to transfer obtained knowledge from math package to Zabbix.

1.2 Choosing appropriate interval and forecast horizon

You should not make a forecast for say a year based on the data of last hour. Even minuscule jiggling in last hour data will result in huge oscillations in forecast.

You should not base a forecast for say a minute on the data of last month. If the direction your system is going in changed in last five minutes such forecast will still be going in the old direction.

Sweet spot is somewhere in between. Where exactly? This may depend on many factors. Start your search with ratio 1:1 between your interval and forecast horizon. Try different ratios and choose which one works better. Figure 3 gives some idea of what forecasting with different interval:horizon ratios might look like.

Note: Exponential and higher degree polynomial are fast growing functions. Using long forecast horizon may result in *huge* return values.

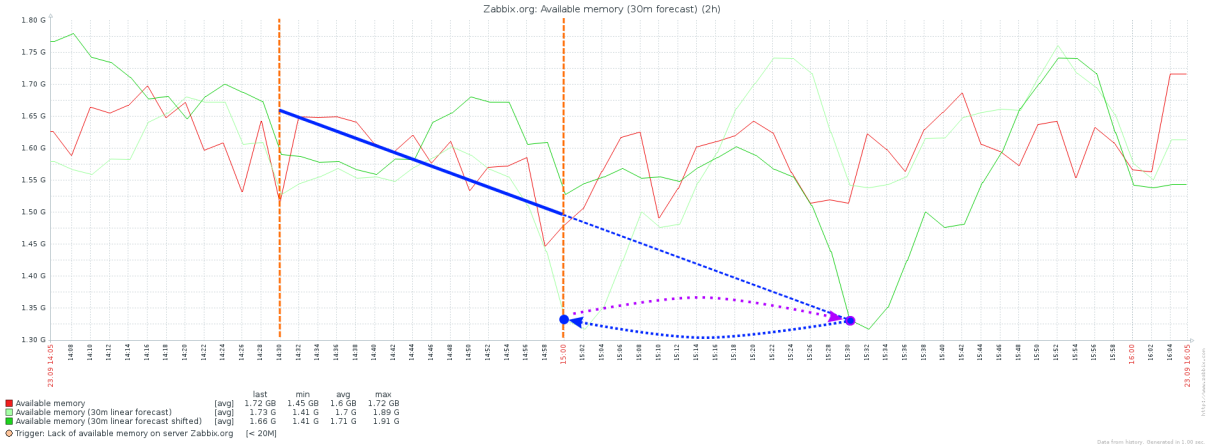

Figure 1: Red line on graph shows original item values, light green line shows `forecast(item,30m,,30m)` and dark green line shows shifted forecast obtained using `last(item.forecast,#1,30m)`. Dashed orange lines show interval of 30 minutes on which we base forecast. Solid blue line shows best fit and dashed blue line extrapolates it 30 minutes ahead. Blue arrow shows that value "from the future" appears on graph "now", but purple arrow shows shifting it back with `last()`. From the picture we see here we can conclude that available memory on the system unfortunately is unpredictable.

1.3 Reliability of the predictions (and how it's related to interval)

Make sure there are *enough* values in the interval. This will make your forecast more stable in case of outliers or substantial noise and randomness in the data. How many values is *enough*? Again, there is no answer for everybody. More values is better but do not overdo it because more values mean more calculation and even worse than that are "memory effects" (See Figure 3). And unfortunately forecast error for M points is proportional to $\frac{1}{\sqrt{M}}$, so increasing interval won't pay off as well as you probably expect in terms of increasing precision.

1.3.1 Full interval of "good fit" is needed for the prediction to be reliable

Long intervals are good for accuracy, but they cause a lag when something goes "not as planned". Let's consider an example.

```
{host:vfs.fs.size[/,free].timeleft(1h,,0)} < 1h
```

This trigger expression will warn you if you are going to run out of disk space less than in one hour time. Imagine that you are already low on disk space but you can live with that for days probably because you do not write huge amounts of data on the disk. At least you weren't during the past hour. But in the last minute someone very nasty came and started downloading big files from the Internet. The above trigger will not fire because it still "remembers" that you had not been writing to the disk something substantial for a long time. Last minute won't contribute much if your forecast is based on one hour interval. It will take approximately one hour for `timeleft()` to get back on track of returning accurate prognosis.

Another aspect is that typical real-life monitoring data has a lot of "breaks", things change quickly in the modern world. Too quickly for 18$^{\text{th}}$ century forecasting methodology sometimes. Consider free disk space. It goes down at a steady pace and the `timeleft()` above is telling you that you will need a new disk after a month. Then you (or anybody else) delete a substantially large file, free disk space reading jumps up and `timeleft()` thinks you will never run out of disk space. And it will be telling you this very fairy tale *for about one hour*! This is too long, anything can happen in this hour. We need to get back on rails as soon as possible. And we won't believe that we will never run out of disk space, of course we will.

```
(timeleft("vfs.fs.size[/,free]",1h,,0) < 100d)
*
timeleft("vfs.fs.size[/,free]",1h,,0)
+
(timeleft("vfs.fs.size[/,free]",1h,,0) >= 100d)
*
(
  (timeleft("vfs.fs.size[/,free]",30m,,0) < 100d)
```

```
  *
  timeleft("vfs.fs.size[/,free]",30m,,0)
  +
  (timeleft("vfs.fs.size[/,free]",30m,,0) >= 100d)
  *
  (
    (timeleft("vfs.fs.size[/,free]",15m,,0) < 100d)
    *
    timeleft("vfs.fs.size[/,free]",15m,,0)
    +
    (timeleft("vfs.fs.size[/,free]",15m,,0) >= 100d)
    *
    (
      (timeleft("vfs.fs.size[/,free]",7m,,0) < 100d)
      *
      timeleft("vfs.fs.size[/,free]",7m,,0)
      +
      (timeleft("vfs.fs.size[/,free]",7m,,0) >= 100d)
      *
      (
        100d
      )
    )
  )
)
```

This is a "hackish" expression for calculated item, because it relies on "true" being equal to 1 and "false" being equal to 0. But it can be rewritten as a completely legal trigger expression.

```
{host:vfs.fs.size[/,free].timeleft(1h,,0)} < 100d
and
{host:vfs.fs.size[/,free].timeleft(1h,,0)} < 1h
or
{host:vfs.fs.size[/,free].timeleft(1h,,0)} >= 100d
and
(
  {host:vfs.fs.size[/,free].timeleft(30m,,0)} < 100d
  and
  {host:vfs.fs.size[/,free].timeleft(30m,,0)} < 1h
  or
  {host:vfs.fs.size[/,free].timeleft(30m,,0)} >= 100d
  and
  (
    {host:vfs.fs.size[/,free].timeleft(15m,,0)} < 100d
    and
    {host:vfs.fs.size[/,free].timeleft(15m,,0)} < 1h
    or
    {host:vfs.fs.size[/,free].timeleft(15m,,0)} >= 100d
    and
    (
      {host:vfs.fs.size[/,free].timeleft(7m,,0)} < 100d
      and
      {host:vfs.fs.size[/,free].timeleft(7m,,0)} < 1h
    )
  )
)
```

This trigger will warn you if you are going to run out of free disk space in less than one hour based on the most accurate (based on the longest interval) prognosis which is not pointing into infinity. Figure 2 depicts how "out-of-the-box" `timeleft("vfs.fs.size[/,free]",1h,,0)` and more sophisticated expression respond to the described situation. Ordinary `timeleft()` stays "deaf" for almost one hour, whereas "complicated timeleft" never exceeds a reasonable mark of 100 days (does not spoil your graphs)

Figure 2: Comparison of response to the same situation of ordinary `timeleft()` and more complicated combination of `timeleft()`'s (see text for more details). Note that ~ 30 thousand years (in the upper graph) is $\sim 10^{12}$ seconds—our synonym of "never".

and just after 6–7 minutes we get more or less reasonable estimate which gets more and more precise as time goes.

Typical behaviour of linear fit "crawling" over sudden drops or "steps" in otherwise very steady and predictable data can be seen in Figure 3. Since the "drop" is literally sudden and there had been no evidence it would happen right until the moment when it happened all forecasts "lag" forecasting horizon (20 minutes in this case) behind the "drop". Then they all "react" and the shorter is the interval the more vigorous is reaction. After some time all forecasts are back on track the last being the forecast with the longest interval.

1.4 When and why to use time shift?

Of course, to make contemporary prognosis it is highly recommended to use the most recent data available. But sometimes you want to use particular time interval for your prediction and time shift allows you to do that.

For example, you can make prediction as if there were no recent data and then compare this "forecast from the past" with latest values you actually gathered. If the difference is not too big then your forecast without time shift is likely to be correct.

```
({host:item.forecast(1h,1h,1h,,avg)} - {host:item.avg(1h)}) *
({host:item.forecast(1h,1h,1h,,avg)} - {host:item.avg(1h)}) /
{host:item.avg(1h)} / {host:item.avg(1h)} < 0.01 and
{host:item.forecast(1h,,1h)} > limit
```

This trigger will fire if you have *very* good chances to go over the limit in the upcoming hour. (Actually, for this example to work, "time" should be counted from "now"-"time shift" and "avg" applied on interval from "now"-"time shift" to "now"-"time shift"+"time". In the current implementation "time" is always counted from "now" and interval from "now" to "now"+"time" is analysed. Take a look at ZBXNEXT-2972 if you'd like to have forecast validation capabilities.)

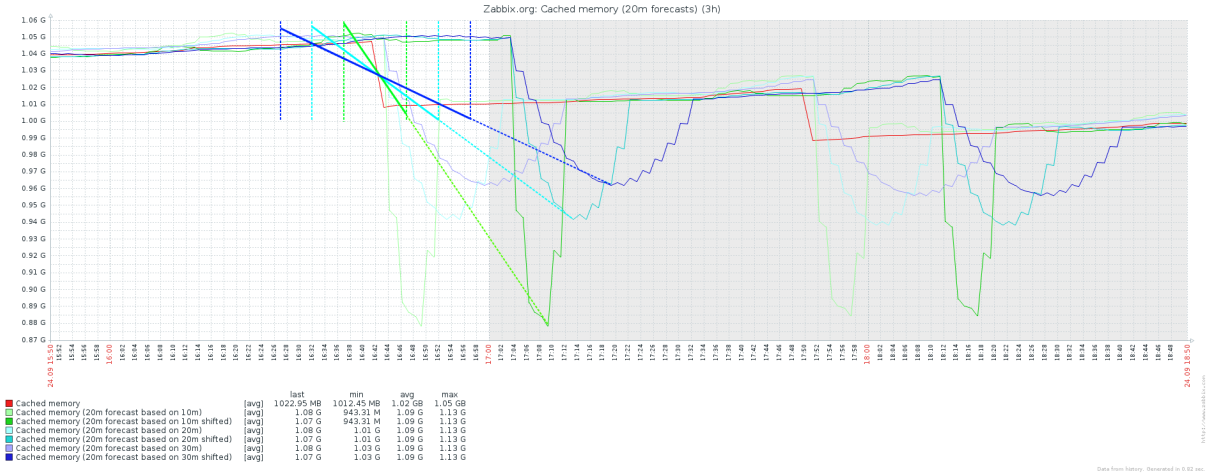

Figure 3: Original item values are plotted in red, light lines are results of `forecast()` and darker ones are results of `forecast()` shifted with `last()`. Fitting intervals, fitted straight lines and their extrapolation are shown in the middle of the "drop".

As mentioned above you should use longer intervals with more data points to obtain more accurate long-term forecasts. But forecasts based on longer intervals can be very slow to respond to the rapid change in trend. So you would normally use longer interval but switch to shorter interval automatically in the time of change. (Replace exact comparisons with approximate like in the example above.)

```
{host:item.forecast(1h,,1h)} > limit and
{host:item.forecast(30m,30m,1h)} = {host:item.forecast(30m,,1h)} or
{host:item.forecast(30m,,1h)} > limit and
{host:item.forecast(30m,30m,1h)} <> {host:item.forecast(30m,,1h)}
```

This trigger expression will look at precise but inert forecast, but skip it in favour of less precise but more agile forecast if trend change occurred in the last hour.

1.5 Determining which fit to use

In general case, these words of statistician George Box may be very useful:

> Now it would be very remarkable if any system existing in the real world could be exactly represented by any simple model. However, cunningly chosen parsimonious models often do provide remarkably useful approximations. For example, the law $PV = RT$ relating pressure P, volume V and temperature T of an "ideal" gas via a constant R is not exactly true for any real gas, but it frequently provides a useful approximation and furthermore its structure is informative since it springs from a physical view of the behavior of gas molecules.
>
> For such a model there is no need to ask the question "Is the model true?". If "truth" is to be the "whole truth" the answer must be "No". The only question of interest is "Is the model illuminating and useful?".

If you have insights on how your monitored system behaves and which mathematical patterns it follows and Zabbix forecasting trigger functions support such fits—use them!

Otherwise start with linear fit (which is default) and move on from there if you need. If you clearly see that your data is not straight but is curved you may want to try polynomial fit. Start with polynomial2 (quadratic parabola), switch to polynomial3 (cubic parabola) if polynomial2 does not satisfy you, and so on... But, please, keep in mind that higher order polynomials can give rise to many unpleasant numeric complications.

Power fit is very adaptive. It can be as straight as $x = kt$ or as fast growing as $x = kt^n$ or as downhill as $x = \frac{k}{t^n}$ depending on input data. May be useful when your data has "ups" and "downs".

Exponential fit grows very, *very* quickly. It may be used for peak detection.

It is more complicated to come up with a good example where to use logarithmic fit, but if your system behaves like this—use logarithmic fit and let us know about your experience!

1.6 When and why to use other mode than "value"

Modes other than "value" are there to emulate existing `min()`, `max()`, `avg()` and `delta()` trigger functions for time intervals in the future. Perhaps you already have a trigger like that which works perfectly fine:

```
{host:item.max(5m)} > limit
```

Based on this trigger you can make predictive one like:

```
{host:item.forecast(10m,,5m,,max)} > limit
```

Note: Modes "min", "max", "avg", "delta" do not forecast how your data will fluctuate in the future! This are simply minimum, maximum, average and delta of the *fitted expression* in the interval from "now" to "now" + "time".

These additional modes do not provide something particularly interesting for monotonous and stable fits (namely linear, logarithmic, exponential, power) but for polynomials it is safer to use "min" and "max" (for lower and upper limits correspondingly) since polynomials may have multiple "humps" which will be out of your field of view if you just look at the "value". And thanks to unstable nature of polynomial fits these "humps" can grow "out of nowhere", again, outside of your field of view as new values come into your interval.

1.7 Dealing with periodic behaviour

Fit options currently supported by Zabbix do not allow any periodicity in forecasts. How to act if you have (more or less) periodic pattern in data? Say server load usually increases in working hours Monday to Friday but stays low during nights and weekends.

One option is to forget about periodicity and predict average values. Provide a sufficiently long interval containing 5–10 periods and you can forecast how will average value of the item evolve in long-term scenario.

Another option is to restrict yourself with short-term forecasts. This time you need your interval and forecasting horizon to be 5–10 times shorter than one period.

But you can also think outside of the box and combine usual trigger functions with forecasting ones to write something like that:

```
{host:item.forecast(7d,,1d)} +
({host:item.last(,6d)} - {host:item.forecast(7d,,-6d)}) > limit
```

Note: Support for negative parameters needed to write such expressions. Consider voting for ZBXNEXT-2969.

2 Details of the implementation

2.1 Trends data is not used

No trend data at the moment, only history. Therefore, adjust your value cache size accordingly if you are planning to use forecasting capabilities. Do not check items which need longer intervals too frequently. Or create additional calculated item which performs some averaging before data is passed to the forecasting function.

2.2 Min, max values of the returned floats

Return value of `forecast()` is in range from $-\left(10^{12}-10^{-4}\right)$ to $10^{12}-10^{-4}$. If the actual prediction is outside this range it will be cropped to $\pm\left(10^{12}-10^{-4}\right)$. In case of error return value is -1, trigger function will not become not supported.

Return value of `timeleft()` is normally in range from 0 to $10^{12}-10^{-4}$. If the actual time to reach the threshold is larger or the threshold cannot be reached at all return value is $10^{12}-10^{-4}$. In case of error return value is -1, trigger function will not become not supported.

Table 1: Fitted expressions

"fit"	$x = f(t)$
linear	$x = a + bt$
polynomial$N$	$x = \sum_{n=0}^{N} a_n t^n$
exponential	$x = a \exp(bt)$
logarithmic	$x = a + b \log(t)$
power	$x = at^b$

2.3 Types of mathematical errors to expect and suggestions on next steps to recover from them

We did our best to make forecasting algorithm as robust as possible and these errors are uncommon, but sometimes input data is really tough to work with.

1. Matrix inversion error. Matrix inversion is essential step of our least squares fitting procedure. This usually means that you have too long interval in combination with higher degree polynomial or huge item values. Adjust your parameters or scale down item values.
2. Polynomial root finding error. We need to find polynomial roots when we are interested in polynomial minima, maxima and intersections with threshold. There are exact formulas for quadratic equations, but for higher degree polynomials it is an iterative numerical procedure. We assume that t is a good *enough* root of $f(t)$ if $|f(t)| < \epsilon$, $\epsilon = 10^{-6}$. We stop iterations when all roots are *good enough* (it takes 10–15 iterations in more or less simple situations) or we hit iteration count limit—200 iterations. You may want to tune ϵ and iteration count limit if you are constantly getting this error.
3. Zero or negative value under logarithm. Item values must be positive if you want to use exponential or power fit.
4. Numeric overflow error. Probably a mixture of *very* small and *very* large values in a combination with higher degree polynomial. Scale item values or use different fit.

3 Reference

3.1 Model

The main idea behind forecasting is quite simple. We have data points (t_i, x_i), where x_i are item values and t_i are corresponding timestamps. We assume that there is some law underlying our data:

$$x_i = f(t_i) + \Delta x_i, \tag{1}$$

where Δx_i denotes error. Error can be due to our data were measured with imperfect instruments or due to the law being far from reality. Also, errors may be purely random. Note that there is error only for x_i, because we believe t_i is known *exactly*.

Our task now is to find $f(t)$ and then we will be able to feed it with t values from the future and deduce something useful about upcoming events.

To narrow our field of search we restrict $f(t)$ to have certain looks and limited set of parameters (See Table 1).

3.2 Least squares

The trickiest part is how we define "best fit". We assume that Δx_i are *independent identically distributed* random variables all have normal distribution:

$$p(\Delta x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{\Delta x^2}{2\sigma^2}\right). \tag{2}$$

Distribution parameter σ is unknown to us, but it makes no problem. Let's estimate *probability, that a specific* $f^\star(t)$ *produced our set of data* (t_i, x_i). Skipping constant coefficient in front of $\exp(\ldots)$ for simplicity.

$$P(f^\star) \sim \prod_i \exp\left(-\frac{\Delta x_i^2}{2\sigma^2}\right) = \exp\left(-\frac{1}{2\sigma^2}\sum_i \Delta x_i^2\right). \tag{3}$$

Table 2: Substitutions

"fit"	ξ_i	$\boldsymbol{\tau_i}$	$\boldsymbol{a}^t$
linear	x_i	$\begin{pmatrix}1 & t_i\end{pmatrix}$	$\begin{pmatrix}a & b\end{pmatrix}$
polynomial$N$	x_i	$\begin{pmatrix}1 & t & t^2 & \dots & t^N\end{pmatrix}$	$\begin{pmatrix}a_0 & a_1 & a_2 & \dots & a_N\end{pmatrix}$
exponential	$\log(x_i)$	$\begin{pmatrix}1 & t_i\end{pmatrix}$	$\begin{pmatrix}\log(a) & b\end{pmatrix}$
logarithmic	x_i	$\begin{pmatrix}1 & \log(t_i)\end{pmatrix}$	$\begin{pmatrix}a & b\end{pmatrix}$
power	$\log(x_i)$	$\begin{pmatrix}1 & \log(t_i)\end{pmatrix}$	$\begin{pmatrix}\log(a) & b\end{pmatrix}$

Table 3: Return values of `forecast()` function

"mode"	`forecast()`
value	$f^\star\left(t_{\text{"now"}} + t_{\text{"time"}}\right)$
max	$\max_{t_{\text{"now"}} \le t \le t_{\text{"now"}} + t_{\text{"time"}}} f^\star(t)$
min	$\min_{t_{\text{"now"}} \le t \le t_{\text{"now"}} + t_{\text{"time"}}} f^\star(t)$
delta	$\max_{\dots} f^\star(t) - \min_{\dots} f^\star(t)$
avg	$\frac{1}{t_{\text{"time"}}} \int_{t_{\text{"now"}}}^{t_{\text{"now"}} + t_{\text{"time"}}} f^\star(t)\,dt$

The best $f^\star(t)$ is obviously the one which has the largest $P(f^\star)$ and largest $P(f^\star)$ is when $\sum_i \Delta x_i^2$ is the smallest. So, to get best fit we need to minimize $\sum_i \left(x_i - f^\star(t_i)\right)^2$. This is called least squares method.

3.3 Linearisation

Using matrix multiplication we can rewrite all expressions from Table 1 in a simple form:

$$\boldsymbol{\xi} = \boldsymbol{T}\boldsymbol{a} + \boldsymbol{\Delta\xi}, \quad \boldsymbol{\xi} = \begin{pmatrix}\dots\\ \xi_i \\ \dots\end{pmatrix}, \quad \boldsymbol{T} = \begin{pmatrix}\dots\\ \boldsymbol{\tau_i} \\ \dots\end{pmatrix}, \quad \boldsymbol{\Delta\xi} = \begin{pmatrix}\dots\\ \Delta\xi_i \\ \dots\end{pmatrix}. \tag{4}$$

The meaning of $\boldsymbol{\xi_i}$, $\boldsymbol{\tau_i}$ and $\boldsymbol{a}$ is explained in Table 2. By applying $\log(\dots)$ we distort the normal distribution of Δx_i but usually we still get acceptable results this way. There is somewhat "hackish" way to deal with that but we don't use it because it's "hackish". Feel free to prove us wrong with a link to proper mathematical proof of this method.

3.4 Minimization

The minimum of $\sum_i \Delta\xi_i^2 = \boldsymbol{\Delta\xi}^t\boldsymbol{\Delta\xi}$ is when

$$\frac{\partial\left(\boldsymbol{\Delta\xi}^t\boldsymbol{\Delta\xi}\right)}{\partial \boldsymbol{a}^t} = 0. \tag{5}$$

Since $\boldsymbol{\Delta\xi} = \boldsymbol{\xi} - \boldsymbol{T}\boldsymbol{a}$ this yields:

$$\frac{\partial\boldsymbol{\Delta\xi}^t\boldsymbol{\Delta\xi}}{\partial\boldsymbol{a}^t} = \frac{\partial\left(\boldsymbol{\xi} - \boldsymbol{T}\boldsymbol{a}\right)^t\left(\boldsymbol{\xi} - \boldsymbol{T}\boldsymbol{a}\right)}{\partial\boldsymbol{a}^t} = \\ = \frac{\partial}{\partial\boldsymbol{a}^t}\left(\boldsymbol{\xi}^t\boldsymbol{\xi} - \boldsymbol{\xi}^t\boldsymbol{T}\boldsymbol{a} - \boldsymbol{a}^t\boldsymbol{T}^t\boldsymbol{\xi} + \boldsymbol{a}^t\boldsymbol{T}^t\boldsymbol{T}\boldsymbol{a}\right) = -2\boldsymbol{T}^t\boldsymbol{\xi} + 2\boldsymbol{T}^t\boldsymbol{T}\boldsymbol{a} = 0. \tag{6}$$

From there:

$$\boldsymbol{a} = \left(\boldsymbol{T}^t\boldsymbol{T}\right)^{-1}\boldsymbol{T}^t\boldsymbol{\xi}. \tag{7}$$

See proof for details.

When we have $\boldsymbol{a}$ we can make inverse substitution and get the desired $f^\star(t)$.

3.5 Return values in different modes

What `forecast()` function should return is specified in Table 3. How it calculates these things is specified in Table 4.

Function `timeleft()` obtains $f^\star(t)$, then solves equation $f^\star(t) = x_{\text{"threshold"}}$, finds the closest root $t_0 > t_{\text{"now"}}$ and returns $t_0 - t_{\text{"now"}}$ (or -1 if there are no such roots). Some details on how it does this are in Table 4.

Table 4: Exact formulas used in calculations

		linear	polynomial$N$	exponential	logarithmic	power
`forecast()`	value	$f^\star(t_\mathrm{r})$	$f^\star(t_\mathrm{r})$	$f^\star(t_\mathrm{r})$	$f^\star(t_\mathrm{r})$	$f^\star(t_\mathrm{r})$
	max	$\max\{f^\star(t_\mathrm{l}), f^\star(t_\mathrm{r})\}$	*	$\max\{f^\star(t_\mathrm{l}), f^\star(t_\mathrm{r})\}$	$\max\{f^\star(t_\mathrm{l}), f^\star(t_\mathrm{r})\}$	$\max\{f^\star(t_\mathrm{l}), f^\star(t_\mathrm{r})\}$
	min	$\min\{f^\star(t_\mathrm{l}), f^\star(t_\mathrm{r})\}$	*	$\min\{f^\star(t_\mathrm{l}), f^\star(t_\mathrm{r})\}$	$\min\{f^\star(t_\mathrm{l}), f^\star(t_\mathrm{r})\}$	$\min\{f^\star(t_\mathrm{l}), f^\star(t_\mathrm{r})\}$
	delta	$\lvert f^\star(t_\mathrm{l}) - f^\star(t_\mathrm{r})\rvert$	*	$\lvert f^\star(t_\mathrm{l}) - f^\star(t_\mathrm{r})\rvert$	$\lvert f^\star(t_\mathrm{l}) - f^\star(t_\mathrm{r})\rvert$	$\lvert f^\star(t_\mathrm{l}) - f^\star(t_\mathrm{r})\rvert$
	avg	$\frac{f^\star(t_\mathrm{l})+f^\star(t_\mathrm{r})}{2}$	$\frac{F(t_\mathrm{r})-F(t_\mathrm{l})}{t_\mathrm{r}-t_\mathrm{l}}$	$\frac{f^\star(t_\mathrm{r})-f^\star(t_\mathrm{l})}{(t_\mathrm{r}-t_\mathrm{l})a_1}$	$f^\star(t_\mathrm{r}) + a_1\left(\log\left(1+\frac{t_\mathrm{r}-t_\mathrm{l}}{t_\mathrm{l}}\right)\frac{t_\mathrm{l}}{t_\mathrm{r}-t_\mathrm{l}} - 1\right)$	$\begin{cases} \frac{(f^\star(t_\mathrm{r})t_\mathrm{r}-f^\star(t_\mathrm{l})t_\mathrm{l})}{(t_\mathrm{r}-t_\mathrm{l})(a_1+1)}, & \text{if} \quad a_1 \neq -1 \\ \frac{\exp(a_0)\log\left(1+\frac{t_\mathrm{r}-t_\mathrm{l}}{t_\mathrm{l}}\right)}{t_\mathrm{r}-t_\mathrm{l}}, & \text{if} \quad a_1 = -1 \end{cases}$
`timeleft()`		$\frac{x_\mathrm{th}-a_0}{a_1} - t_\mathrm{l}$	**	$\frac{\log(x_\mathrm{th})-a_0}{a_1} - t_\mathrm{l}$	$\exp\left(\frac{x_\mathrm{th}-a_0}{a_1}\right) - t_\mathrm{l}$	$\exp\left(\frac{\log(x_\mathrm{th})-a_0}{a_1}\right) - t_\mathrm{l}$

Where $f^\star$ is expression from Table 1 with "best fit" coefficients, $t_\mathrm{l} = t_{\text{“now”}}$, $t_\mathrm{r} = t_{\text{“now”}} + t_{\text{“time”}}$, $F(t) = \sum_{n=1}^{N+1} \frac{a_{n-1}t^n}{n} + C$ is polynomial antiderivative, x_th is "threshold", a_i means i-th element of $\boldsymbol{a}$ from Table 2.

* We solve $\frac{df^\star(t)}{dt} = 0$ using ** (with $N' = N-1$) and search for maximum and minimum among t_l, t_r and roots lying in between; $\frac{df^\star(t)}{dt} = \sum_{n=0}^{N-1}(n+1)\,a_{n+1}t^n$.

** Exact formulas for $N = 1$ (see linear case) and $N = 2$, (Weierstrass—)Durand—Kerner method for $3 \leq N \leq 6$.

3.6 Performance

Forecasting process is done in four stages:

1. getting data from value cache;
2. preparing data (according to Table 2);
3. least squares fitting;
4. calculating necessary return value (according to Table 3).

First step is no different from other trigger functions dealing with history (`avg()`, `count()`, `delta()`, `max()`, `min()`, `sum()`) and takes time linearly dependent on number M of item values in a given interval.

Second step also takes linear time but different for different fits. Linear is the fastest and involves no additional calculations. Polynomial takes more time since we need to raise every timestamp to power 1, 2, ..., N. Exponential and logarithmic are even slower because we call expensive $\log()$ function for either every timestamp or every item value respectively. Power fit is the slowest on this stage because we need to $\log()$ both timestamps *and* item values.

Fitting is linear on M as well and identical to all fits except polynomial$N$ for which it is $\frac{N+1}{2}$ times slower.

Last step does not depend on M, we've already got $f^\star()$ expression and simply analyse it. For linear, exponential, logarithmic and power fits for `forecast()` in any mode and for `timeleft()` it is a "single shot" formula. Of course, simpler one for linear, more complicated one consisting of $\exp()$'s and $\log()$'s for others. For polynomial's `forecast()` in modes "value" or "avg" it is straightforward formula as well (however, a longer one for larger N, obviously).

Things get more complicated for polynomial's `timeleft()` and `forecast()` in modes "max", "min" and "delta". But even here we've got formulas for smaller N (1–2 for `timeleft()`, 1–3 for `forecast()`). For higher degrees we have no choice apart from solving equation numerically. It can take up to 200 iterations (each taking time proportional to N) with no guarantee of success in most difficult cases (but they are very uncommon).

All in all, `forecast()` and `timeleft()` with default linear fit and polynomial2–3 are reasonably cheap performance-wise. More exotic fits have greater performance impact but they won't be used as widely.

4 Cross validating results from Zabbix (getting out the data and coefficients)

To get the data out from Zabbix one can use direct queries to Zabbix database, Zabbix API or simply copy data manually from Zabbix frontend➞Monitoring➞Latest data➞item➞Values.

Fitted expression is printed in server log file in human-readable form if `DebugLevel` is set to 4 or 5.