**A Lightweight, Self-adjusting, Baseline-less Data Monitor**

**By** [**Alex Chamchourine**](http://www.sqlservercentral.com/Authors/Articles/Alex_Chamchourine/366425/)**, 2016/02/23**

Introduction

As database professionals we all probably at some point had to deal with some kind of data monitoring. It could be system data from the very servers we manage or business operational data. And the main purpose of such monitoring usually is to answer one seemingly simple question – if the data shows the signs of being abnormal. The standard approach to tackle this is to have a baseline for the metric you are concerned with and compare the current value with the baseline.

That could be a really challenging task. Just imagine that your data has a high seasonality (low months vs. holiday season) or changes greatly between night and day hours. You might end up with collecting your baseline data hourly for an entire year. To make it worse - once the baseline is established you need to decide how much deviation is too much. In other words, passing what threshold is enough to warrant an alarm? You don’t want to be woken up in the middle of the night just to learn that nothing special is going on with your system; in that case you just set your threshold too low. Nor do you want to learn someday that your system has not worked as expected for a while but you were not aware, as you set your threshold too high.

So is there any other way? Can data itself give us a hint that something is going on, without baselines and thresholds? The answer is yes – if we use a statistical approach and let the data speak. In order to do so we will need...

A Bit of a Theory

The best approach would be to judge the observed value based on its probability, as it’s much easier to deal with probabilities rather than the absolute values themselves. With probability one always can tell that the probability of 0.1% is indeed low and the probability of 99.9% is high. No questions here. Compare this to the case when you only know that your observed value differs from your average by 25%. This could mean anything: no issue at all if your data is highly dispersed and possibly critical situation if your data is very homogeneous.

To figure out what the probability is of seeing the observed value, we need to know the probability density function for your data. Simple. Well, not exactly. The function is known only for a few distributions, of which the Normal is the most widely used. If your data is distributed normally (or close enough), you can calculate probability for your observed value either by using the analytical probability density function or by looking through pre-calculated values in special tables.

For instance, to find out the value with the probability of 0.15% or less you just need to calculate the average value and the standard deviation of your data set (the functions are usually built-in in many program languages, including T-SQL). Once you know the average and the deviation it’s easy to find the value: AVG-3\*STDEV (a well-known [68-95-99.7](https://en.wikipedia.org/wiki/68%E2%80%9395%E2%80%9399.7_rule) rule). The rule refers to the both ends of the distribution and you are interested in only one, (either low or high values), thus you take only a half of the value. The problem though that there is a good chance that your data is not distributed normally.

Though you still can work from an assumption of normality I would like to introduce another approach. We can build our probability density function (table valued of course, not analytical) out of data at hand and use it in the exactly same way as one for the Normal Distribution to calculate probabilities. So what is a probability density function?

As complicated as it sounds it’s actually a data histogram on steroids. Imaging that you measure some characteristic of your system periodically. The value of each individual measure becomes a data point in your dataset (called time series). And a histogram simply displays how many of your data points fall to specific range of values you measured. That is shows a dependency of the quantities of data points (Axis Y) from measured values (Axis X).

To build a histogram we need to estimate a bin size (we will use [Freedman-Diaconis rule](https://en.wikipedia.org/wiki/Freedman%E2%80%93Diaconis_rule) as a guidance) and “pour” the data into the bins. More details can be found in the attached code. Once the histogram is completed we need to normalize the quantities inside the every bin by total number of data points in your data set. And the last step in converting your histogram to a probability density function is to divide normalized quantities by their respective value ranges (like “spreading” them over Axis X).

One can say – OK, I have the function but what about the probability? The probability is the area under the curve. That is to find the probability to see a value equal or greater than *V*, we need to find the area under the curve from *V* (on Axis X) to the very end (the right end) of the curve.

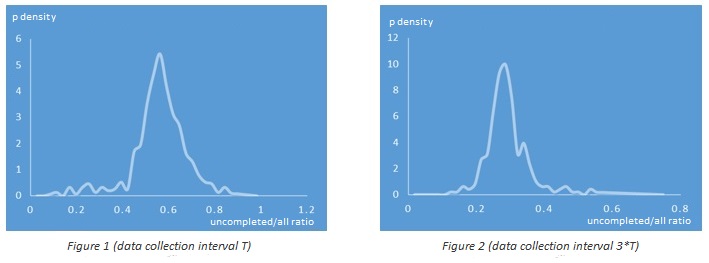
Now that we know how to go about it, we can get started.

The Problem

Imagine that you have a server (or more likely a farm of servers) where users upload their files. We have a table where each record represents one upload with a start time, an end time, a file size and maybe a status with success or failure. We would like, from the data alone, to be able to tell if our system is healthy or if there is trouble brewing.

One really important thing here is to choose a proper metric. Though a few can be employed, we will use a ratio of uncompleted uploads in a period of time to the total number of uploads in the same period. We just need to make sure that the chosen interval is such that the substantial number of uploads normally get finished within it so the ratio is not too close to 1. Usually about a half should suffice.

Due to the natural limitations of the selected metric (it’s always between 0 and 1), the values will not be distributed normally, but nevertheless the distribution likely will resemble the normal one (well up to some degree of course). The figures below represent an example of a real probability density function where the time interval was chosen to have about half of uploads completed for Figure 1 and ~70% for Figure 2 (it should be mentioned that the data was collected at different times).



http://www.sqlservercentral.com/Resources/Images/zoom.gif[Zoom in](javascript:;)  |  [Open in new window](javascript:;)

You can see that the probability density function on Figure 1 is quite similar to the one of the Normal Distribution and the function on Figure 2 is noticeably different (due to the longer right tail). And indeed if you calculate the threshold value for the probability 0.15% using the Normal Distribution and this article approach for Figure 1 dataset you won’t find any significant difference while for Figure 2 dataset the difference will be quite noticeable.

Bits and pieces

OK, let’s start coding! First of all as we will put all our objects to a schema called *monitor* we need to create it.

create schema monitor

go

Next we will create a couple of new types that we will be using to represent our dataset:

create type monitor.timeseries as table(value float)

go

and our probability density table function:

create type monitor.pfunction as table(right\_boundary float, cumulative\_probability float, id int)

go

Now the core procedure that generates the probability density function:

create procedure [monitor].[get\_pfunction]

@timeseries monitor.timeseries readonly

as

/\*

Generates cumulative probablity table function

Returns table of type monitor.pfunction

Note that cumulative probability function is descending from 1 to 0.

\*/

set nocount on

declare @t table(value float, id int identity(1,1))

insert @t(value)

select value from @timeseries order by 1

declare @bw float, @min float, @max float, @n float

select @n = count(\*), @max = max(value), @min = min(value) from @t

-- finding initial bin size

declare @iqr float, @n1 float, @n3 float, @q1 float, @q3 float, @f1 float, @f2 float, @nb int

set @n1 = (@n +1)/4.00

set @n3 = 3\*(@n +1)/4.00

if( @n1 != ceiling(@n1) )

begin

select @f1 = value from @t where id = floor(@n1)

select @f2 = value from @t where id = ceiling(@n1)

set @q1 = (@f1+@f2)/2

end

else

select @q1 = value from @t where id = @n1

if( @n3 != ceiling(@n3) )

begin

select @f1 = value from @t where id = floor(@n3)

select @f2 = value from @t where id = ceiling(@n3)

set @q3 = (@f1+@f2)/2

end

else

select @q3 = value from @t where id = @n3

-- this is interquartile value

set @iqr = @q3-@q1

-- using Freedman-Diaconis rule as a "guidance" to calculate initial bin size:

set @bw = 2\*@iqr\*power(cast(@n as float), -1/3.00)

-- if bin width is zero (which is a rare case) will use some rather arbitrary number

if( @bw = 0 ) set @bw = 2\*sqrt(@n)

declare @x table(f float, bw float, id int identity(1,1))

declare @h1 table(bw float, id int identity(1,1) primary key)

declare @h2 table(rb float, id int identity(1,1) primary key)

declare @hst table(rb float, frequency int, id int identity(1,1) primary key)

declare @hist table(rb float, frequency int, density float, old\_density float, id int identity(1,1) primary key)

declare @tp table(rb float, density float, cumulativeProb float, rba float, id int identity(1,1) primary key)

set @nb = ceiling((@max-@min)/@bw)

-- we are using variable bin size (proportional to value) if otherwise we have nonsensical number of bins

if( @nb > @n/3 /\*to keep # of bins reasonably low - rather arbitrary\*/ )

begin

insert @x(f, bw)

select case when value/@bw > 1 then ceiling(value/@bw) else 1 end, case when value/@bw > 1 then ceiling(value/@bw) \*@bw else @bw end

from @t

order by value

insert @h1(bw)

select distinct bw from @x order by bw

end

else

begin

insert @h1(bw) select top(cast(ceiling(@max/@bw) as int)) @bw from @t

end

declare @id int, @idmax int, @rb float = 0

select @id = min(id), @idmax = max(id) from @h1

while @id <= @idmax

begin

select @rb = @rb +bw from @h1 where id = @id

insert @h2(rb) values(@rb)

set @id = @id +1

if( @rb>@max) break

end

-- histogramm table

insert @hst(rb, frequency)

select h.rb, sum(case when t.value is not null then 1 else 0 end)

from @h2 h

left join @h2 hp on hp.id = h.id-1

left join @t t on h.rb >= t.value and (hp.id is null or hp.rb < t.value)

group by h.rb

order by 1

-- p density table

insert @hist(rb, frequency, density)

select h1.rb, h1.frequency, h1.frequency/@n/(h1.rb-isnull(h2.rb, 0))

from @hst h1

left join @hst h2 on h2.id = h1.id-1

order by h1.id

-- approximate "tail" to the right

declare @pd float, @x0 float

select @pd = density from @hist where id = (select max(id) from @hist)

select top 1 @id = id from @hist where density > @pd order by id desc

-- linear regression (y = ax +b)

declare @xa float, @ya float, @a float, @b float

select @xa = avg(rb), @ya = avg(density) from @hist where id >= @id

select @a = sum((rb-@xa)\*(density-@ya))/sum(power((rb-@xa),2)) from @hist where id >= @id

set @b = @ya -@a\*@xa

-- this will be the last point where p density = 0

set @x0 = -@b/@a

select @idmax = max(id) from @hist

-- adjust previous points p density according to regression (so that total probablity = 1)

while @id <= @idmax

begin

update @hist set density = @a\*rb +@b, old\_density = density where id = @id

set @id = @id +1

end

-- now we need to adjust density due to increased length

declare @mrb float

select @mrb = rb from @hist where id = @idmax

if( @x0 > @mrb )

begin

declare @dx float

set @dx = @mrb -(select rb from @hist where id = @idmax-1)

update @hist set density = density\*@dx/(@x0 -@mrb) where id = @idmax

end

delete from @hist where rb > @x0

insert @hist(rb, density) values(@x0, 0)

insert @tp(rb, density)

select rb, density from @hist order by id

-- calculate cumulative probability

declare @cp float, @idmin int

select @idmin = min(id), @idmax = max(id) from @tp

set @id = @idmax

while @id > @idmin

begin

select @cp = sum((h1.rb-isnull(h2.rb, 0))\*h2.density)

from @tp h1

join @tp h2 on h2.id = h1.id-1

where h1.id >= @id

update @tp set cumulativeProb = @cp where id = @id-1

set @id = @id -1

end

update @tp set cumulativeProb = 0 where id = (select max(id) from @tp)

set @id = @idmin

while @id <= @idmax

begin

-- the first record with none zero density should have probability 1 as well

update @tp set cumulativeProb = 1 where id = @id

select @pd = density from @tp where id = @id

if( @pd > 0 ) break

set @id = @id +1

end

-- finally we have a cumulative probability table function

select right\_boundary = rb, cumulative\_probability = cumulativeProb, id from @tp order by id

go

The procedure accepts the time series dataset as a table of *monitor.timeseries* type and returns a dataset of *monitor.pfunction* type. It’s important to note that this function expects that you are interested in the right tail of the density curve (high end values) rather than the close to 0 part and will extrapolate to find x-intercept value on the right only.

You can find the full code of this procedure (that contains some useful comments) in the attached script “*functionality.sql*”.

Finally we need two twin functions: one to calculate the probability of a value, and another to calculate a value by its probability.

create procedure monitor.get\_probability\_by\_value

@pfunction monitor.pfunction readonly,

@value float,

@probability float output

as

/\*

Returns probability of provided observed value (based on @pfunction)

\*/

set nocount on

declare @id2 int, @v2 float, @p2 float, @v1 float, @p1 float

select top 1 @id2 = id, @v2 = right\_boundary, @p2 = cumulative\_probability

from @pfunction

where right\_boundary >= @value

order by right\_boundary

set @probability = @p2 -- in case if @value == rb

if( (select right\_boundary from @pfunction where id = @id2) > @value )

begin

-- approximate lineary

select @v1 = right\_boundary, @p1 = cumulative\_probability

from @pfunction

where id = @id2 -1

if( @p1 != @p2 and @v1 != @v2 )

set @probability = (@value-@v2)\*(@p2-@p1)/(@v2-@v1) +@p2

end

-- if provided value is beyond x-intercept

if( @probability is null ) set @probability = 0

go

create procedure monitor.get\_value\_by\_probability

@pfunction monitor.pfunction readonly,

@probability float,

@value float output

as

/\*

Returns value estimate for provided probability (based on @pfunction)

This sp requires the following temp table to be created in the same session before sp call:

\*/

set nocount on

declare @id1 int, @v2 float, @p2 float, @v1 float, @p1 float

select top 1 @id1 = id, @p1 = cumulative\_probability, @v1 = right\_boundary

from @pfunction

where cumulative\_probability >= @probability

order by cumulative\_probability

set @value = @v1 -- in case if cprob == @probability

if( (select cumulative\_probability from @pfunction where id = @id1) > @probability )

begin

-- approximate lineary

select @v2 = right\_boundary, @p2 = cumulative\_probability

from @pfunction

where id = @id1 +1

if( @p1 != @p2 and @v1 != @v2 )

set @value = (@probability -@p2)\*(@v2 -@v1)/(@p2 -@p1) +@v2

end

go

Again the full code of these functions can be found in the attached script “*functionality.sql*”.

So far we created objects necessary for probability calculation. Next step is to create objects for the monitor itself. The table to store configuration values:

create table monitor.time\_series\_config(

series\_id int not null identity(1,1),

series\_name varchar(256) null,

window\_width\_seconds int not null,

max\_number\_of\_points int not null,

threshold\_probability float not null,

notify\_if\_found\_more\_than int not null,

db\_mail\_profile varchar(128) null,

email\_recipients varchar(1000) null,

email\_subject varchar(512) null,

constraint PK\_time\_series\_config primary key(series\_id)

)

end

go

/\*

window\_width\_seconds: should be selected so that a substantial number of records (a half for instance) within it

have their complete\_date set (that is completed)

max\_number\_of\_points: arbitrary number, large enough to provide sufficient number of point for our math (400-600)

threshold\_probability: the target probability we consider low enough to warrant an alert

notify\_if\_found\_more\_than: normally should be 0 but sometimes we want to make sure that it was not just a freak event

but something that persists, especially useful if you run your scan quite frequently

\*/

The time series table:

create table monitor.time\_series(

id bigint not null identity(1,1),

series\_id int not null,

create\_time datetime not null,

value float not null,

probability float null,

constraint PK\_time\_series primary key(id)

)

go

The procedure that generates a new data point, evaluates its probability, adds a new record to the time series table (see above) and trims the time series table if necessary:

create procedure [monitor].[new\_point]

@series\_id int

as

/\*

Metric: a ratio of uncompleted uploads in a period to the total number of uploads in the same period

Adds new observed value (point) to the time series and calculates its probability.

Also implements "sliding window" by removing the very first point (if applicable)

\*/

set nocount on

declare @timeseries monitor.timeseries

declare @pfunction monitor.pfunction

declare @now datetime = getdate()

declare @window\_width\_seconds int, @max\_number\_of\_points int, @threshold\_probability float

declare @error\_status int = -1

select @window\_width\_seconds = window\_width\_seconds,

@max\_number\_of\_points = max\_number\_of\_points,

@threshold\_probability = threshold\_probability

from monitor.time\_series\_config

where series\_id = @series\_id

declare @d1 datetime, @d2 datetime

set @d2 = @now

set @d1 = dateadd(ss, -@window\_width\_seconds, @now)

declare @current\_value float

-->> USE YOUR TABLE HERE INSTEAD OF monitor.uploads

select @current\_value = 1 -sum(case when complete\_date is not null and status\_id != @error\_status then 1 else 0 end)

/cast(case when count(\*) > 0 then count(\*) else 1 end as float)

from monitor.uploads

where create\_date >= @d1

and create\_date < @d2

and isnull(file\_size,0) > 0

--<< USE YOUR TABLE HERE INSTEAD OF monitor.uploads

declare @current\_probability float

-- finding current probability

-- load time series

insert @timeseries(value)

select value

from monitor.time\_series

where series\_id = @series\_id

-- re-calc cumulative probabilities (only if we have sufficient # of points, the number is rather arbitrary - you can use a different one)

if( (select count(\*) from @timeseries) >= 48 )

begin

insert @pfunction(right\_boundary, cumulative\_probability, id)

exec monitor.get\_pfunction @timeseries = @timeseries

exec monitor.get\_probability\_by\_value @pfunction = @pfunction, @value = @current\_value, @probability = @current\_probability output

end

-- add current point to the time series

insert monitor.time\_series(series\_id, create\_time, value, probability)

values(@series\_id, @now, @current\_value, @current\_probability)

-- if # of point > max limit remove the very first one

if( (select count(\*) from monitor.time\_series where series\_id = @series\_id) > @max\_number\_of\_points )

begin

delete from monitor.time\_series where id = (select min(id) from monitor.time\_series where series\_id = @series\_id)

end

go

The procedure that checks the probability of the last observed value and sends a notification if applicable:

create procedure [monitor].[scan]

@series\_id int

as

/\*

Scans time series to find if alarm/notification is warranted

\*/

set nocount on

declare @t table(probability float, value float)

declare @notify\_if\_found\_more\_than int, @threshold\_probability float

declare @series\_name varchar(256), @db\_mail\_profile varchar(128), @recipients varchar(1000), @subject varchar(1000)

select @notify\_if\_found\_more\_than = isnull(notify\_if\_found\_more\_than, 0),

@threshold\_probability = threshold\_probability,

@series\_name = isnull(series\_name, cast(series\_id as varchar(50))),

@db\_mail\_profile = db\_mail\_profile,

@recipients = email\_recipients,

@subject = email\_subject

from monitor.time\_series\_config

where series\_id = @series\_id

insert @t(probability, value)

select top(@notify\_if\_found\_more\_than +1) probability, value from monitor.time\_series where series\_id = @series\_id order by id desc

if( (select count(\*) from @t where probability < @threshold\_probability) <= @notify\_if\_found\_more\_than )

return

if( @db\_mail\_profile is not null and @recipients is not null )

begin

declare @value float, @probability float

select top(1) @probability = probability, @value = value from @t where probability < @threshold\_probability

set @subject = replace(replace(replace(@subject, '', @series\_name), '', @value), '', @probability)

exec msdb.dbo.sp\_send\_dbmail

@profile\_name = @db\_mail\_profile,

@recipients = @recipients,

@subject = @subject

end

else

print 'Value with less than threshold probability encountered!'

go

The full code for all these objects also can be found in the attached script “*functionality.sql*”.

The last component would be a job that will be obtaining a new time series data point and checking if sending a notification is warranted. You could schedule this job as you like. The only thing the job should do is to run the following:

declare @series\_id int

set @series\_id = 1 /\*or any other id\*/

exec [monitor].[new\_point] @series\_id = @series\_id

exec [monitor].[scan] @series\_id = @series\_id

go

Also in the attached script file, *"distributions used in the article.sql",* you can find scipt that generates a normally distributed dataset (to use it as a reference or play around with), and the one that generates distributions used in Figure 1 and Figure 2 . For your convenience I also included “*test data.sql*” script that creates the *uploads* table and populates it with some test data in case you would like to try the functionality but do not have anything immediately suitable in your system.

In conclusion I would like to discuss applicability and limitations.

Applicability and Limitations

Though it’s designed to be “agnostic” to underlying process and based solely on the data, it’s still important that you know your data (and the system) well. The ratio metric likely will not work as expected if your data collecting interval is too long comparing to the average processing time and you have too few records within. It likely will be hyper-sensitive as any single “out of normal” record will set off the alarm. In this case you might try to choose another metric like a process speed instead.

Note the fact that *get\_pfunction* is designed to work with the right tail (high end values) of the density curve. There are two reasons for that. The first one is that the most metrics we can come up with will be naturally limited from the left by 0 (like speed and count) and thus the curve might be steep there and not be able to produce a valuable insight (remember that we have to extrapolate). The second reason is that in case you aggregate your data and look for low values you might lose the signal among the high values “noise”.

Understanding the sensitivity is another important matter here. You don’t want it be either extremely sensitive or numb. It can be easily shown that the sensitivity of the density function from Figure 2 is significantly higher than the one from Figure 1. Imaging that you divided your records into a smaller subsets arbitrary or based on some characteristic of your system (for instance records coming from different servers) and calculated the ratio for each one. Then, given a target probability threshold of 0.15% and defining a failure as ratio = 1.0 , the Figure 2 function will “sense” the failure when about 50% of such subsets failed whereas the one from Figure 1 will need almost 80% of such. And it can be generalized that the further to left the curve peak is, the more sensitive it will be.

This approach (generating a probability function out of data samples, and using it to estimate the probability of the observed value) is universal and not limited to any particular process. The metric (the ratio of uncompleted uploads to all) needs to be adjusted to your concrete implementation however. For instance it could be a ratio of failed to succeeded events, or even a ratio of suspended to running processes for SQLServer sysprocesses system view.

**Resources:**

[distributions used in the article.sql](http://www.sqlservercentral.com/Files/distributions%20used%20in%20the%20article.sql/28219.sql) | [test data.sql](http://www.sqlservercentral.com/Files/test%20data.sql/28220.sql) | [functionality.sql](http://www.sqlservercentral.com/Files/functionality.sql/28257.sql)

THE DISTRIBUTIONS MENTIONED ABOVE ARE COPIED BELOW

THIS IS [distributions used in the article.sql](http://www.sqlservercentral.com/Files/distributions%20used%20in%20the%20article.sql/28219.sql)

set nocount on

if( object\_id('tempdb..##f1') is not null ) drop table ##f1

if( object\_id('tempdb..##f2') is not null ) drop table ##f2

create table ##f1(value float)

create table ##f2(value float)

insert ##f1(value) values(0.404664)

insert ##f1(value) values(0.411711)

insert ##f1(value) values(0.406769)

insert ##f1(value) values(0.454175)

insert ##f1(value) values(0.49052)

insert ##f1(value) values(0.471205)

insert ##f1(value) values(0.410399)

insert ##f1(value) values(0.390932)

insert ##f1(value) values(0.438453)

insert ##f1(value) values(0.466301)

insert ##f1(value) values(0.461548)

insert ##f1(value) values(0.437986)

insert ##f1(value) values(0.496163)

insert ##f1(value) values(0.549583)

insert ##f1(value) values(0.48398)

insert ##f1(value) values(0.495792)

insert ##f1(value) values(0.382479)

insert ##f1(value) values(0.36211)

insert ##f1(value) values(0.385938)

insert ##f1(value) values(0.431022)

insert ##f1(value) values(0.433307)

insert ##f1(value) values(0.486788)

insert ##f1(value) values(0.487505)

insert ##f1(value) values(0.538729)

insert ##f1(value) values(0.524079)

insert ##f1(value) values(0.436053)

insert ##f1(value) values(0.488165)

insert ##f1(value) values(0.451935)

insert ##f1(value) values(0.421691)

insert ##f1(value) values(0.460014)

insert ##f1(value) values(0.455187)

insert ##f1(value) values(0.360213)

insert ##f1(value) values(0.463027)

insert ##f1(value) values(0.473961)

insert ##f1(value) values(0.419027)

insert ##f1(value) values(0.368725)

insert ##f1(value) values(0.362285)

insert ##f1(value) values(0.450096)

insert ##f1(value) values(0.453045)

insert ##f1(value) values(0.477791)

insert ##f1(value) values(0.500272)

insert ##f1(value) values(0.534384)

insert ##f1(value) values(0.430244)

insert ##f1(value) values(0.448488)

insert ##f1(value) values(0.45785)

insert ##f1(value) values(0.475429)

insert ##f1(value) values(0.441328)

insert ##f1(value) values(0.478685)

insert ##f1(value) values(0.357492)

insert ##f1(value) values(0.706812)

insert ##f1(value) values(0.431908)

insert ##f1(value) values(0.555934)

insert ##f1(value) values(0.534435)

insert ##f1(value) values(0.327881)

insert ##f1(value) values(0.25119)

insert ##f1(value) values(0.856135)

insert ##f1(value) values(0.786861)

insert ##f1(value) values(0.295044)

insert ##f1(value) values(0.632411)

insert ##f1(value) values(0.779621)

insert ##f1(value) values(0.430911)

insert ##f1(value) values(0.755927)

insert ##f1(value) values(0.290739)

insert ##f1(value) values(0.62358)

insert ##f1(value) values(0.673768)

insert ##f1(value) values(0.83376)

insert ##f1(value) values(0.226258)

insert ##f1(value) values(0.706458)

insert ##f1(value) values(0.762085)

insert ##f1(value) values(0.376293)

insert ##f1(value) values(0.390341)

insert ##f1(value) values(0.50219)

insert ##f1(value) values(0.574631)

insert ##f1(value) values(0.371231)

insert ##f1(value) values(0.342555)

insert ##f1(value) values(0.457256)

insert ##f1(value) values(0.40808)

insert ##f1(value) values(0.495835)

insert ##f1(value) values(0.550773)

insert ##f1(value) values(0.285912)

insert ##f1(value) values(0.346181)

insert ##f1(value) values(0.41889)

insert ##f1(value) values(0.718366)

insert ##f1(value) values(0.445509)

insert ##f1(value) values(0.479566)

insert ##f1(value) values(0.462465)

insert ##f1(value) values(0.41903)

insert ##f1(value) values(0.544645)

insert ##f1(value) values(0.573163)

insert ##f1(value) values(0.556824)

insert ##f1(value) values(0.546054)

insert ##f1(value) values(0.623435)

insert ##f1(value) values(0.473388)

insert ##f1(value) values(0.505243)

insert ##f1(value) values(0.450986)

insert ##f1(value) values(0.515105)

insert ##f1(value) values(0.555124)

insert ##f1(value) values(0.561244)

insert ##f1(value) values(0.450705)

insert ##f1(value) values(0.394115)

insert ##f1(value) values(0.473766)

insert ##f1(value) values(0.488428)

insert ##f1(value) values(0.444488)

insert ##f1(value) values(0.519138)

insert ##f1(value) values(0.513588)

insert ##f1(value) values(0.449676)

insert ##f1(value) values(0.576143)

insert ##f1(value) values(0.573328)

insert ##f1(value) values(0.47335)

insert ##f1(value) values(0.474556)

insert ##f1(value) values(0.523169)

insert ##f1(value) values(0.485162)

insert ##f1(value) values(0.380489)

insert ##f1(value) values(0.512442)

insert ##f1(value) values(0.51407)

insert ##f1(value) values(0.473181)

insert ##f1(value) values(0.470304)

insert ##f1(value) values(0.401941)

insert ##f1(value) values(0.3908)

insert ##f1(value) values(0.425016)

insert ##f1(value) values(0.523871)

insert ##f1(value) values(0.308815)

insert ##f1(value) values(0.321906)

insert ##f1(value) values(0.409897)

insert ##f1(value) values(0.446314)

insert ##f1(value) values(0.560931)

insert ##f1(value) values(0.464194)

insert ##f1(value) values(0.500075)

insert ##f1(value) values(0.47264)

insert ##f1(value) values(0.445598)

insert ##f1(value) values(0.426631)

insert ##f1(value) values(0.397661)

insert ##f1(value) values(0.485162)

insert ##f1(value) values(0.557291)

insert ##f1(value) values(0.481884)

insert ##f1(value) values(0.498747)

insert ##f1(value) values(0.522087)

insert ##f1(value) values(0.506829)

insert ##f1(value) values(0.396053)

insert ##f1(value) values(0.420888)

insert ##f1(value) values(0.486655)

insert ##f1(value) values(0.535563)

insert ##f1(value) values(0.505587)

insert ##f1(value) values(0.393512)

insert ##f1(value) values(0.462547)

insert ##f1(value) values(0.376732)

insert ##f1(value) values(0.378048)

insert ##f1(value) values(0.44528)

insert ##f1(value) values(0.39736)

insert ##f1(value) values(0.42016)

insert ##f1(value) values(0.490307)

insert ##f1(value) values(0.319226)

insert ##f1(value) values(0.342035)

insert ##f1(value) values(0.412616)

insert ##f1(value) values(0.458328)

insert ##f1(value) values(0.51431)

insert ##f1(value) values(0.429764)

insert ##f1(value) values(0.413803)

insert ##f1(value) values(0.390516)

insert ##f1(value) values(0.544214)

insert ##f1(value) values(0.477971)

insert ##f1(value) values(0.447717)

insert ##f1(value) values(0.510624)

insert ##f1(value) values(0.362848)

insert ##f1(value) values(0.363865)

insert ##f1(value) values(0.414423)

insert ##f1(value) values(0.464305)

insert ##f1(value) values(0.430226)

insert ##f1(value) values(0.352032)

insert ##f1(value) values(0.464909)

insert ##f1(value) values(0.541434)

insert ##f1(value) values(0.505119)

insert ##f1(value) values(0.454362)

insert ##f1(value) values(0.501041)

insert ##f1(value) values(0.609658)

insert ##f1(value) values(0.480257)

insert ##f1(value) values(0.662618)

insert ##f1(value) values(0.321815)

insert ##f1(value) values(0.45555)

insert ##f1(value) values(0.537332)

insert ##f1(value) values(0.402068)

insert ##f1(value) values(0.274058)

insert ##f1(value) values(0.289361)

insert ##f1(value) values(0.315414)

insert ##f1(value) values(0.421346)

insert ##f1(value) values(0.310936)

insert ##f1(value) values(0.185778)

insert ##f1(value) values(0.252713)

insert ##f1(value) values(0.516693)

insert ##f1(value) values(0.32254)

insert ##f1(value) values(0.277372)

insert ##f1(value) values(0.470145)

insert ##f1(value) values(0.381481)

insert ##f1(value) values(0.353642)

insert ##f1(value) values(0.464889)

insert ##f1(value) values(0.324558)

insert ##f1(value) values(0.272894)

insert ##f1(value) values(0.476963)

insert ##f1(value) values(0.565428)

insert ##f1(value) values(0.36504)

insert ##f1(value) values(0.241947)

insert ##f1(value) values(0.341859)

insert ##f1(value) values(0.220126)

insert ##f1(value) values(0.316576)

insert ##f1(value) values(0.602606)

insert ##f1(value) values(0.9)

insert ##f1(value) values(0.909707)

insert ##f1(value) values(0.572238)

insert ##f1(value) values(0.443478)

insert ##f1(value) values(0.47651)

insert ##f1(value) values(0.447368)

insert ##f1(value) values(0.254335)

insert ##f1(value) values(0.841912)

insert ##f1(value) values(0.854962)

insert ##f1(value) values(0.305687)

insert ##f1(value) values(0.801843)

insert ##f1(value) values(0.784)

insert ##f1(value) values(0.184898)

insert ##f1(value) values(0.273408)

insert ##f1(value) values(0.155149)

insert ##f1(value) values(0.73913)

insert ##f1(value) values(0.505254)

insert ##f1(value) values(0.433255)

insert ##f1(value) values(0.299497)

insert ##f1(value) values(0.372142)

insert ##f1(value) values(0.521454)

insert ##f1(value) values(0.34591)

insert ##f1(value) values(0.555242)

insert ##f1(value) values(0.5642)

insert ##f1(value) values(0.480892)

insert ##f1(value) values(0.542828)

insert ##f1(value) values(0.538613)

insert ##f1(value) values(0.472899)

insert ##f1(value) values(0.426247)

insert ##f1(value) values(0.375934)

insert ##f1(value) values(0.516466)

insert ##f1(value) values(0.28278)

insert ##f1(value) values(0.304205)

insert ##f1(value) values(0.441898)

insert ##f1(value) values(0.665689)

insert ##f1(value) values(0.518634)

insert ##f1(value) values(0.464765)

insert ##f1(value) values(0.413434)

insert ##f1(value) values(0.408542)

insert ##f1(value) values(0.513884)

insert ##f1(value) values(0.418362)

insert ##f1(value) values(0.408136)

insert ##f1(value) values(0.441474)

insert ##f1(value) values(0.480282)

insert ##f1(value) values(0.438054)

insert ##f1(value) values(0.461641)

insert ##f1(value) values(0.380615)

insert ##f1(value) values(0.489636)

insert ##f1(value) values(0.464874)

insert ##f1(value) values(0.431437)

insert ##f1(value) values(0.418729)

insert ##f1(value) values(0.427268)

insert ##f1(value) values(0.344223)

insert ##f1(value) values(0.346469)

insert ##f1(value) values(0.402411)

insert ##f1(value) values(0.403335)

insert ##f1(value) values(0.414943)

insert ##f1(value) values(0.436163)

insert ##f1(value) values(0.48092)

insert ##f1(value) values(0.463134)

insert ##f1(value) values(0.496777)

insert ##f1(value) values(0.4305)

insert ##f1(value) values(0.442264)

insert ##f1(value) values(0.46103)

insert ##f1(value) values(0.473791)

insert ##f1(value) values(0.434809)

insert ##f1(value) values(0.408282)

insert ##f1(value) values(0.379723)

insert ##f1(value) values(0.402675)

insert ##f1(value) values(0.436829)

insert ##f1(value) values(0.425687)

insert ##f1(value) values(0.382764)

insert ##f1(value) values(0.371007)

insert ##f1(value) values(0.473436)

insert ##f1(value) values(0.39974)

insert ##f1(value) values(0.377177)

insert ##f1(value) values(0.48445)

insert ##f1(value) values(0.43175)

insert ##f1(value) values(0.451193)

insert ##f1(value) values(0.51186)

insert ##f1(value) values(0.40056)

insert ##f1(value) values(0.432863)

insert ##f1(value) values(0.377849)

insert ##f1(value) values(0.494279)

insert ##f1(value) values(0.366531)

insert ##f1(value) values(0.397584)

insert ##f1(value) values(0.425829)

insert ##f1(value) values(0.530736)

insert ##f1(value) values(0.434163)

insert ##f1(value) values(0.48721)

insert ##f1(value) values(0.459213)

insert ##f1(value) values(0.44863)

insert ##f1(value) values(0.354018)

insert ##f1(value) values(0.454121)

insert ##f1(value) values(0.453432)

insert ##f1(value) values(0.447708)

insert ##f1(value) values(0.408662)

insert ##f1(value) values(0.425761)

insert ##f1(value) values(0.515293)

insert ##f1(value) values(0.385795)

insert ##f1(value) values(0.73029)

insert ##f1(value) values(0.255462)

insert ##f1(value) values(0.791667)

insert ##f1(value) values(0.772727)

insert ##f1(value) values(0.246927)

insert ##f1(value) values(0.570048)

insert ##f1(value) values(0.49197)

insert ##f1(value) values(0.75085)

insert ##f1(value) values(0.641822)

insert ##f1(value) values(0.226788)

insert ##f1(value) values(0.754545)

insert ##f1(value) values(0.395954)

insert ##f1(value) values(0.517474)

insert ##f1(value) values(0.752485)

insert ##f1(value) values(0.5584)

insert ##f1(value) values(0.544534)

insert ##f1(value) values(0.339398)

insert ##f1(value) values(0.54241)

insert ##f1(value) values(0.715984)

insert ##f1(value) values(0.349722)

insert ##f1(value) values(0.432571)

insert ##f1(value) values(0.537135)

insert ##f1(value) values(0.456362)

insert ##f1(value) values(0.56326)

insert ##f1(value) values(0.463409)

insert ##f1(value) values(0.465319)

insert ##f1(value) values(0.541064)

insert ##f1(value) values(0.623154)

insert ##f1(value) values(0.472617)

insert ##f1(value) values(0.491691)

insert ##f1(value) values(0.543768)

insert ##f1(value) values(0.476033)

insert ##f1(value) values(0.569506)

insert ##f1(value) values(0.434648)

insert ##f1(value) values(0.363919)

insert ##f1(value) values(0.452458)

insert ##f1(value) values(0.505337)

insert ##f1(value) values(0.485471)

insert ##f1(value) values(0.378437)

insert ##f1(value) values(0.35843)

insert ##f1(value) values(0.487657)

insert ##f1(value) values(0.504413)

insert ##f1(value) values(0.526817)

insert ##f1(value) values(0.537952)

insert ##f1(value) values(0.475703)

insert ##f1(value) values(0.408005)

insert ##f1(value) values(0.472314)

insert ##f1(value) values(0.464841)

insert ##f1(value) values(0.513988)

insert ##f1(value) values(0.469009)

insert ##f1(value) values(0.487454)

insert ##f1(value) values(0.507377)

insert ##f1(value) values(0.454697)

insert ##f1(value) values(0.515078)

insert ##f1(value) values(0.525763)

insert ##f1(value) values(0.536146)

insert ##f1(value) values(0.528017)

insert ##f1(value) values(0.521212)

insert ##f1(value) values(0.366886)

insert ##f1(value) values(0.351605)

insert ##f1(value) values(0.366381)

insert ##f1(value) values(0.570356)

insert ##f1(value) values(0.478458)

insert ##f1(value) values(0.360265)

insert ##f1(value) values(0.474574)

insert ##f1(value) values(0.405226)

insert ##f1(value) values(0.340684)

insert ##f1(value) values(0.403795)

insert ##f1(value) values(0.425441)

insert ##f1(value) values(0.441568)

insert ##f1(value) values(0.457059)

insert ##f1(value) values(0.391155)

insert ##f1(value) values(0.519044)

insert ##f1(value) values(0.432677)

insert ##f1(value) values(0.442904)

insert ##f1(value) values(0.515018)

insert ##f1(value) values(0.353253)

insert ##f1(value) values(0.443561)

insert ##f1(value) values(0.349958)

insert ##f1(value) values(0.524794)

insert ##f1(value) values(0.58451)

insert ##f1(value) values(0.424923)

insert ##f1(value) values(0.452452)

insert ##f1(value) values(0.481843)

insert ##f1(value) values(0.480046)

insert ##f1(value) values(0.589932)

insert ##f1(value) values(0.505495)

insert ##f1(value) values(0.579648)

insert ##f1(value) values(0.254289)

insert ##f1(value) values(0.425808)

insert ##f1(value) values(0.300908)

insert ##f1(value) values(0.45133)

insert ##f1(value) values(0.398884)

insert ##f1(value) values(0.320896)

insert ##f1(value) values(0.226217)

insert ##f1(value) values(0.222106)

insert ##f1(value) values(0.612)

insert ##f1(value) values(0.637975)

insert ##f1(value) values(0.20098)

insert ##f1(value) values(0.223806)

insert ##f1(value) values(0.183648)

insert ##f1(value) values(0.934732)

insert ##f1(value) values(0.551601)

insert ##f1(value) values(0.841667)

insert ##f1(value) values(0.328877)

insert ##f1(value) values(0.330729)

insert ##f1(value) values(0.212935)

insert ##f1(value) values(0.311648)

insert ##f1(value) values(0.389386)

insert ##f1(value) values(0.46281)

insert ##f1(value) values(0.337105)

insert ##f1(value) values(0.555827)

insert ##f1(value) values(0.428571)

insert ##f1(value) values(0.564223)

insert ##f1(value) values(0.492174)

insert ##f1(value) values(0.533673)

insert ##f1(value) values(0.50176)

insert ##f1(value) values(0.448972)

insert ##f1(value) values(0.55543)

insert ##f1(value) values(0.495924)

insert ##f1(value) values(0.480816)

insert ##f1(value) values(0.443031)

insert ##f1(value) values(0.578329)

insert ##f1(value) values(0.439461)

insert ##f1(value) values(0.397498)

insert ##f1(value) values(0.467249)

insert ##f1(value) values(0.478471)

insert ##f1(value) values(0.458229)

insert ##f1(value) values(0.465887)

insert ##f1(value) values(0.400789)

insert ##f1(value) values(0.424919)

insert ##f1(value) values(0.365607)

insert ##f1(value) values(0.38281)

insert ##f1(value) values(0.421475)

insert ##f1(value) values(0.417339)

insert ##f1(value) values(0.433015)

insert ##f1(value) values(0.337374)

insert ##f1(value) values(0.42081)

insert ##f1(value) values(0.368966)

insert ##f1(value) values(0.447459)

insert ##f1(value) values(0.455279)

insert ##f1(value) values(0.502615)

insert ##f1(value) values(0.473132)

insert ##f1(value) values(0.462027)

insert ##f1(value) values(0.389092)

insert ##f1(value) values(0.767628)

insert ##f1(value) values(0.51182)

insert ##f1(value) values(0.53835)

insert ##f1(value) values(0.395857)

insert ##f1(value) values(0.375639)

insert ##f1(value) values(0.461386)

insert ##f1(value) values(0.48097)

insert ##f1(value) values(0.4597)

insert ##f1(value) values(0.491452)

insert ##f1(value) values(0.481355)

insert ##f1(value) values(0.346663)

insert ##f1(value) values(0.438984)

insert ##f1(value) values(0.511345)

insert ##f1(value) values(0.521849)

insert ##f1(value) values(0.357464)

insert ##f1(value) values(0.489408)

insert ##f1(value) values(0.313941)

insert ##f1(value) values(0.487298)

insert ##f1(value) values(0.316535)

insert ##f1(value) values(0.523608)

insert ##f1(value) values(0.360242)

insert ##f1(value) values(0.458637)

insert ##f1(value) values(0.432343)

insert ##f1(value) values(0.448688)

insert ##f1(value) values(0.430758)

insert ##f1(value) values(0.444057)

insert ##f1(value) values(0.37925)

insert ##f1(value) values(0.361967)

insert ##f1(value) values(0.391345)

insert ##f1(value) values(0.526488)

insert ##f1(value) values(0.408032)

insert ##f1(value) values(0.373768)

insert ##f1(value) values(0.388619)

insert ##f1(value) values(0.459162)

insert ##f1(value) values(0.47653)

insert ##f1(value) values(0.440417)

insert ##f1(value) values(0.417936)

insert ##f1(value) values(0.431014)

insert ##f1(value) values(0.412809)

insert ##f1(value) values(0.516342)

insert ##f1(value) values(0.332131)

insert ##f1(value) values(0.346751)

insert ##f1(value) values(0.392949)

insert ##f1(value) values(0.453612)

insert ##f1(value) values(0.512471)

insert ##f1(value) values(0.625797)

insert ##f1(value) values(0.827377)

insert ##f1(value) values(0.452793)

insert ##f1(value) values(0.45082)

insert ##f1(value) values(0.643159)

insert ##f1(value) values(0.172481)

insert ##f1(value) values(0.694408)

insert ##f1(value) values(0.670404)

insert ##f1(value) values(0.32312)

insert ##f1(value) values(0.602649)

insert ##f1(value) values(0.282193)

insert ##f1(value) values(0.476631)

insert ##f1(value) values(0.52007)

insert ##f1(value) values(0.499757)

insert ##f1(value) values(0.564801)

insert ##f1(value) values(0.449872)

insert ##f1(value) values(0.441276)

insert ##f1(value) values(0.510569)

insert ##f1(value) values(0.329134)

insert ##f1(value) values(0.462277)

insert ##f1(value) values(0.332961)

insert ##f1(value) values(0.401981)

insert ##f1(value) values(0.51209)

insert ##f1(value) values(0.278327)

insert ##f1(value) values(0.312277)

insert ##f1(value) values(0.260093)

insert ##f1(value) values(0.2488)

insert ##f1(value) values(0.431558)

insert ##f1(value) values(0.399319)

insert ##f1(value) values(0.551872)

insert ##f1(value) values(0.18018)

insert ##f1(value) values(0.309859)

insert ##f1(value) values(0.12854)

insert ##f1(value) values(0.626566)

insert ##f1(value) values(0.474308)

insert ##f1(value) values(0.426829)

insert ##f1(value) values(0.0762195)

insert ##f1(value) values(0.378619)

insert ##f1(value) values(0.14527)

insert ##f1(value) values(0.41078)

insert ##f1(value) values(0.50015)

insert ##f1(value) values(0.432653)

insert ##f1(value) values(0.443916)

insert ##f1(value) values(0.391706)

insert ##f1(value) values(0.478239)

insert ##f1(value) values(0.561216)

insert ##f1(value) values(0.500221)

insert ##f1(value) values(0.437736)

insert ##f1(value) values(0.451151)

insert ##f1(value) values(0.477346)

insert ##f1(value) values(0.503649)

insert ##f1(value) values(0.365545)

insert ##f2(value) values(0.712045)

insert ##f2(value) values(0.686471)

insert ##f2(value) values(0.696259)

insert ##f2(value) values(0.694314)

insert ##f2(value) values(0.674786)

insert ##f2(value) values(0.710791)

insert ##f2(value) values(0.708557)

insert ##f2(value) values(0.716149)

insert ##f2(value) values(0.72814)

insert ##f2(value) values(0.728058)

insert ##f2(value) values(0.737251)

insert ##f2(value) values(0.72705)

insert ##f2(value) values(0.734243)

insert ##f2(value) values(0.751924)

insert ##f2(value) values(0.765429)

insert ##f2(value) values(0.765391)

insert ##f2(value) values(0.738318)

insert ##f2(value) values(0.732034)

insert ##f2(value) values(0.653806)

insert ##f2(value) values(0.73816)

insert ##f2(value) values(0.792958)

insert ##f2(value) values(0.784962)

insert ##f2(value) values(0.794029)

insert ##f2(value) values(0.737417)

insert ##f2(value) values(0.707994)

insert ##f2(value) values(0.708731)

insert ##f2(value) values(0.706109)

insert ##f2(value) values(0.715075)

insert ##f2(value) values(0.845076)

insert ##f2(value) values(0.878218)

insert ##f2(value) values(0.80366)

insert ##f2(value) values(0.745871)

insert ##f2(value) values(0.715618)

insert ##f2(value) values(0.546732)

insert ##f2(value) values(0.516139)

insert ##f2(value) values(0.593609)

insert ##f2(value) values(0.475381)

insert ##f2(value) values(0.548692)

insert ##f2(value) values(0.653212)

insert ##f2(value) values(0.735426)

insert ##f2(value) values(0.750538)

insert ##f2(value) values(0.72576)

insert ##f2(value) values(0.764626)

insert ##f2(value) values(0.738241)

insert ##f2(value) values(0.716905)

insert ##f2(value) values(0.713565)

insert ##f2(value) values(0.741147)

insert ##f2(value) values(0.761041)

insert ##f2(value) values(0.782466)

insert ##f2(value) values(0.787459)

insert ##f2(value) values(0.768004)

insert ##f2(value) values(0.75113)

insert ##f2(value) values(0.729394)

insert ##f2(value) values(0.705532)

insert ##f2(value) values(0.715508)

insert ##f2(value) values(0.717028)

insert ##f2(value) values(0.770106)

insert ##f2(value) values(0.753656)

insert ##f2(value) values(0.756841)

insert ##f2(value) values(0.756787)

insert ##f2(value) values(0.760646)

insert ##f2(value) values(0.729045)

insert ##f2(value) values(0.67343)

insert ##f2(value) values(0.693413)

insert ##f2(value) values(0.651503)

insert ##f2(value) values(0.652941)

insert ##f2(value) values(0.669439)

insert ##f2(value) values(0.690184)

insert ##f2(value) values(0.675647)

insert ##f2(value) values(0.660238)

insert ##f2(value) values(0.66426)

insert ##f2(value) values(0.680017)

insert ##f2(value) values(0.701655)

insert ##f2(value) values(0.750952)

insert ##f2(value) values(0.750602)

insert ##f2(value) values(0.765994)

insert ##f2(value) values(0.752814)

insert ##f2(value) values(0.721208)

insert ##f2(value) values(0.782606)

insert ##f2(value) values(0.825844)

insert ##f2(value) values(0.801096)

insert ##f2(value) values(0.769196)

insert ##f2(value) values(0.772775)

insert ##f2(value) values(0.332798)

insert ##f2(value) values(0.548096)

insert ##f2(value) values(0.581175)

insert ##f2(value) values(0.665852)

insert ##f2(value) values(0.732496)

insert ##f2(value) values(0.615963)

insert ##f2(value) values(0.626499)

insert ##f2(value) values(0.592736)

insert ##f2(value) values(0.674536)

insert ##f2(value) values(0.691304)

insert ##f2(value) values(0.74629)

insert ##f2(value) values(0.670424)

insert ##f2(value) values(0.711476)

insert ##f2(value) values(0.712764)

insert ##f2(value) values(0.741669)

insert ##f2(value) values(0.731062)

insert ##f2(value) values(0.73922)

insert ##f2(value) values(0.724975)

insert ##f2(value) values(0.730025)

insert ##f2(value) values(0.744013)

insert ##f2(value) values(0.726332)

insert ##f2(value) values(0.765834)

insert ##f2(value) values(0.749783)

insert ##f2(value) values(0.739145)

insert ##f2(value) values(0.705583)

insert ##f2(value) values(0.716418)

insert ##f2(value) values(0.723371)

insert ##f2(value) values(0.722517)

insert ##f2(value) values(0.730206)

insert ##f2(value) values(0.757517)

insert ##f2(value) values(0.729885)

insert ##f2(value) values(0.744935)

insert ##f2(value) values(0.729851)

insert ##f2(value) values(0.748853)

insert ##f2(value) values(0.746455)

insert ##f2(value) values(0.78091)

insert ##f2(value) values(0.765647)

insert ##f2(value) values(0.745952)

insert ##f2(value) values(0.76755)

insert ##f2(value) values(0.80655)

insert ##f2(value) values(0.704987)

insert ##f2(value) values(0.717612)

insert ##f2(value) values(0.687952)

insert ##f2(value) values(0.80073)

insert ##f2(value) values(0.710103)

insert ##f2(value) values(0.679688)

insert ##f2(value) values(0.563342)

insert ##f2(value) values(0.676062)

insert ##f2(value) values(0.615547)

insert ##f2(value) values(0.787473)

insert ##f2(value) values(0.671859)

insert ##f2(value) values(0.706357)

insert ##f2(value) values(0.746471)

insert ##f2(value) values(0.79257)

insert ##f2(value) values(0.749244)

insert ##f2(value) values(0.667802)

insert ##f2(value) values(0.695665)

insert ##f2(value) values(0.631906)

insert ##f2(value) values(0.686319)

insert ##f2(value) values(0.736994)

insert ##f2(value) values(0.713812)

insert ##f2(value) values(0.704104)

insert ##f2(value) values(0.720156)

insert ##f2(value) values(0.675512)

insert ##f2(value) values(0.679586)

insert ##f2(value) values(0.718894)

insert ##f2(value) values(0.733015)

insert ##f2(value) values(0.70994)

insert ##f2(value) values(0.656536)

insert ##f2(value) values(0.628399)

insert ##f2(value) values(0.668872)

insert ##f2(value) values(0.769991)

insert ##f2(value) values(0.720361)

insert ##f2(value) values(0.861426)

insert ##f2(value) values(0.732218)

insert ##f2(value) values(0.562411)

insert ##f2(value) values(0.715927)

insert ##f2(value) values(0.752518)

insert ##f2(value) values(0.724576)

insert ##f2(value) values(0.749364)

insert ##f2(value) values(0.732903)

insert ##f2(value) values(0.734356)

insert ##f2(value) values(0.721502)

insert ##f2(value) values(0.773499)

insert ##f2(value) values(0.79539)

insert ##f2(value) values(0.769932)

insert ##f2(value) values(0.72565)

insert ##f2(value) values(0.74236)

insert ##f2(value) values(0.733289)

insert ##f2(value) values(0.745974)

insert ##f2(value) values(0.696851)

insert ##f2(value) values(0.696756)

insert ##f2(value) values(0.662208)

insert ##f2(value) values(0.657964)

insert ##f2(value) values(0.659458)

insert ##f2(value) values(0.749272)

insert ##f2(value) values(0.783111)

insert ##f2(value) values(0.60099)

insert ##f2(value) values(0.64439)

insert ##f2(value) values(0.705399)

insert ##f2(value) values(0.401005)

insert ##f2(value) values(0.386821)

insert ##f2(value) values(0.463299)

insert ##f2(value) values(0.520821)

insert ##f2(value) values(0.669315)

insert ##f2(value) values(0.656155)

insert ##f2(value) values(0.721321)

insert ##f2(value) values(0.793494)

insert ##f2(value) values(0.720092)

insert ##f2(value) values(0.720699)

insert ##f2(value) values(0.715783)

insert ##f2(value) values(0.756176)

insert ##f2(value) values(0.749093)

insert ##f2(value) values(0.743334)

insert ##f2(value) values(0.723767)

insert ##f2(value) values(0.744308)

insert ##f2(value) values(0.733745)

insert ##f2(value) values(0.692344)

insert ##f2(value) values(0.721451)

insert ##f2(value) values(0.693356)

insert ##f2(value) values(0.705706)

insert ##f2(value) values(0.744402)

insert ##f2(value) values(0.780386)

insert ##f2(value) values(0.723651)

insert ##f2(value) values(0.835497)

insert ##f2(value) values(0.846222)

insert ##f2(value) values(0.803256)

insert ##f2(value) values(0.723309)

insert ##f2(value) values(0.752592)

insert ##f2(value) values(0.790223)

insert ##f2(value) values(0.705731)

insert ##f2(value) values(0.712786)

insert ##f2(value) values(0.660873)

insert ##f2(value) values(0.708481)

insert ##f2(value) values(0.706755)

insert ##f2(value) values(0.671571)

insert ##f2(value) values(0.673895)

insert ##f2(value) values(0.702441)

insert ##f2(value) values(0.733286)

insert ##f2(value) values(0.811882)

insert ##f2(value) values(0.817897)

insert ##f2(value) values(0.84345)

insert ##f2(value) values(0.756004)

insert ##f2(value) values(0.792352)

insert ##f2(value) values(0.619339)

insert ##f2(value) values(0.77619)

insert ##f2(value) values(0.756771)

insert ##f2(value) values(0.76434)

insert ##f2(value) values(0.740382)

insert ##f2(value) values(0.740205)

insert ##f2(value) values(0.729681)

insert ##f2(value) values(0.712324)

insert ##f2(value) values(0.749004)

insert ##f2(value) values(0.765511)

insert ##f2(value) values(0.758195)

insert ##f2(value) values(0.638722)

insert ##f2(value) values(0.737346)

insert ##f2(value) values(0.63337)

insert ##f2(value) values(0.810979)

insert ##f2(value) values(0.707805)

insert ##f2(value) values(0.477233)

insert ##f2(value) values(0.453035)

insert ##f2(value) values(0.669777)

insert ##f2(value) values(0.737134)

insert ##f2(value) values(0.707485)

insert ##f2(value) values(0.751923)

insert ##f2(value) values(0.718198)

insert ##f2(value) values(0.751343)

insert ##f2(value) values(0.705627)

insert ##f2(value) values(0.702673)

insert ##f2(value) values(0.723058)

insert ##f2(value) values(0.690719)

insert ##f2(value) values(0.728683)

insert ##f2(value) values(0.710498)

insert ##f2(value) values(0.720908)

insert ##f2(value) values(0.718961)

insert ##f2(value) values(0.743855)

insert ##f2(value) values(0.771374)

insert ##f2(value) values(0.771476)

insert ##f2(value) values(0.739396)

insert ##f2(value) values(0.743733)

insert ##f2(value) values(0.765722)

insert ##f2(value) values(0.71083)

insert ##f2(value) values(0.716147)

insert ##f2(value) values(0.671351)

insert ##f2(value) values(0.697207)

insert ##f2(value) values(0.704697)

select count(\*) from ##f1

select count(\*) from ##f2

go

if( object\_id('tempdb..#v2') is not null ) drop table #v2

create table #v2(rb float, value float)

insert #v2 values(2, 6.69151E-05)

insert #v2 values(2.16, 9.18562E-05)

insert #v2 values(2.32, 0.000125289)

insert #v2 values(2.48, 0.000169801)

insert #v2 values(2.64, 0.000228657)

insert #v2 values(2.8, 0.000305951)

insert #v2 values(2.96, 0.000406761)

insert #v2 values(3.12, 0.000537337)

insert #v2 values(3.28, 0.000705301)

insert #v2 values(3.44, 0.000919863)

insert #v2 values(3.6, 0.001192044)

insert #v2 values(3.76, 0.001534907)

insert #v2 values(3.92, 0.001963777)

insert #v2 values(4.08, 0.00249645)

insert #v2 values(4.24, 0.003153363)

insert #v2 values(4.4, 0.003957726)

insert #v2 values(4.56, 0.004935577)

insert #v2 values(4.72, 0.006115763)

insert #v2 values(4.88, 0.007529808)

insert #v2 values(5.04, 0.009211655)

insert #v2 values(5.2, 0.011197265)

insert #v2 values(5.36, 0.01352405)

insert #v2 values(5.52, 0.016230133)

insert #v2 values(5.68, 0.019353428)

insert #v2 values(5.84, 0.022930538)

insert #v2 values(6, 0.026995483)

insert #v2 values(6.16, 0.031578281)

insert #v2 values(6.32, 0.036703406)

insert #v2 values(6.48, 0.042388181)

insert #v2 values(6.64, 0.048641135)

insert #v2 values(6.8, 0.055460417)

insert #v2 values(6.96, 0.062832318)

insert #v2 values(7.12, 0.070729983)

insert #v2 values(7.28, 0.079112395)

insert #v2 values(7.44, 0.087923715)

insert #v2 values(7.6, 0.097093027)

insert #v2 values(7.76, 0.106534573)

insert #v2 values(7.92, 0.116148502)

insert #v2 values(8.08, 0.125822171)

insert #v2 values(8.24, 0.135431986)

insert #v2 values(8.4, 0.144845776)

insert #v2 values(8.56, 0.15392563)

insert #v2 values(8.72, 0.162531132)

insert #v2 values(8.88, 0.170522894)

insert #v2 values(9.04, 0.177766264)

insert #v2 values(9.2, 0.18413507)

insert #v2 values(9.36, 0.189515263)

insert #v2 values(9.52, 0.193808308)

insert #v2 values(9.68, 0.196934181)

insert #v2 values(9.84, 0.198833853)

insert #v2 values(10, 0.19947114)

insert #v2 values(10.16, 0.198833853)

insert #v2 values(10.32, 0.196934181)

insert #v2 values(10.48, 0.193808308)

insert #v2 values(10.64, 0.189515263)

insert #v2 values(10.8, 0.18413507)

insert #v2 values(10.96, 0.177766264)

insert #v2 values(11.12, 0.170522894)

insert #v2 values(11.28, 0.162531132)

insert #v2 values(11.44, 0.15392563)

insert #v2 values(11.6, 0.144845776)

insert #v2 values(11.76, 0.135431986)

insert #v2 values(11.92, 0.125822171)

insert #v2 values(12.08, 0.116148502)

insert #v2 values(12.24, 0.106534573)

insert #v2 values(12.4, 0.097093027)

insert #v2 values(12.56, 0.087923715)

insert #v2 values(12.72, 0.079112395)

insert #v2 values(12.88, 0.070729983)

insert #v2 values(13.04, 0.062832318)

insert #v2 values(13.2, 0.055460417)

insert #v2 values(13.36, 0.048641135)

insert #v2 values(13.52, 0.042388181)

insert #v2 values(13.68, 0.036703406)

insert #v2 values(13.84, 0.031578281)

insert #v2 values(14, 0.026995483)

insert #v2 values(14.16, 0.022930538)

insert #v2 values(14.32, 0.019353428)

insert #v2 values(14.48, 0.016230133)

insert #v2 values(14.64, 0.01352405)

insert #v2 values(14.8, 0.011197265)

insert #v2 values(14.96, 0.009211655)

insert #v2 values(15.12, 0.007529808)

insert #v2 values(15.28, 0.006115763)

insert #v2 values(15.44, 0.004935577)

insert #v2 values(15.6, 0.003957726)

insert #v2 values(15.76, 0.003153363)

insert #v2 values(15.92, 0.00249645)

insert #v2 values(16.08, 0.001963777)

insert #v2 values(16.24, 0.001534907)

insert #v2 values(16.4, 0.001192044)

insert #v2 values(16.56, 0.000919863)

insert #v2 values(16.72, 0.000705301)

insert #v2 values(16.88, 0.000537337)

insert #v2 values(17.04, 0.000406761)

insert #v2 values(17.2, 0.000305951)

insert #v2 values(17.36, 0.000228657)

insert #v2 values(17.52, 0.000169801)

insert #v2 values(17.68, 0.000125289)

insert #v2 values(17.84, 9.18562E-05)

insert #v2 values(18, 6.69151E-05)

select min(value\*100000), max(value\*100000) from #v2

go

create function monitor.multiply(@rb float, @times int)

returns @m table(value float)

as

begin

declare @i int = 1

while @i < @times

begin

insert @m(value) values(@rb)

set @i = @i +1

end

return

end

go

if( object\_id('tempdb..##normdist') is not null ) drop table ##normdist

create table ##normdist(value float)

go

insert ##normdist(value)

select t.value

from #v2 x

Cross Apply monitor.multiply(x.rb, x.value\*10000) t

order by 1

go

drop function monitor.multiply

go

select min(value), max(value), avg(value), STDEV(value),count(\*) from ##normdist

go

**THIS IS TEST\_DATA.SQL**

if( not exists(select \* from sys.tables where object\_id = object\_id('monitor.uploads')) )

begin

create table monitor.uploads(series\_id int, file\_size int, create\_date datetime, complete\_date datetime, status\_id int)

end

go

set nocount on

if( object\_id('tempdb..#ts') is not null ) drop table #ts

create table #ts(value float)

truncate table monitor.time\_series

truncate table monitor.uploads

declare @i int, @j int, @d datetime, @dc datetime

declare @t table(file\_size int, load\_time int) -- regular

declare @t2 table(file\_size int, load\_time int) -- faster than regular

declare @t3 table(file\_size int, load\_time int) -- slower than regular

declare @t4 table(file\_size int, load\_time int) -- oddball slow

insert @t values(2000000, 100)

insert @t values(2000000, 100)

insert @t values(2000000, 100)

insert @t values(2000000, 100)

insert @t values(2000000, 100)

insert @t values(2000000, 100)

insert @t values(2000000, 100)

insert @t values(2000000, 100)

insert @t values(2000000, 100)

insert @t values(2000000, 100)

insert @t2 values(2000000, 50)

insert @t2 values(2000000, 50)

insert @t2 values(2000000, 50)

insert @t2 values(2000000, 50)

insert @t2 values(2000000, 50)

insert @t2 values(2000000, 50)

insert @t2 values(2000000, 50)

insert @t2 values(2000000, 50)

insert @t2 values(2000000, 50)

insert @t2 values(2000000, 50)

insert @t3 values(2000000, 200)

insert @t3 values(2000000, 200)

insert @t3 values(2000000, 200)

insert @t3 values(2000000, 200)

insert @t3 values(2000000, 200)

insert @t3 values(2000000, 200)

insert @t3 values(2000000, 200)

insert @t3 values(2000000, 200)

insert @t3 values(2000000, 200)

insert @t3 values(2000000, 200)

insert @t4 values(2000000, 100)

insert @t4 values(2000000, 150)

insert @t4 values(2000000, 150)

insert @t4 values(2000000, 150)

insert @t4 values(2000000, 150)

insert @t4 values(2000000, 150)

insert @t4 values(2000000, 150)

insert @t4 values(2000000, 150)

insert @t4 values(2000000, 150)

insert @t4 values(2000000, 150)

declare @f float

set @j = 1

while @j <= 250

begin

set @i = 1

set @dc = getdate()

set @d = dateadd(ss, -300, @dc)

while @i <= 100

begin

set @f = rand()

if( @j in (250))

begin

insert monitor.uploads(series\_id, file\_size, create\_date, complete\_date, status\_id)

select 1, x.file\_size, @d, case when dateadd(ss, x.load\_time, @d) > @dc then null else dateadd(ss, x.load\_time, @d) end, 0 from @t4 x

end

else if( @i%2 = 0 )

begin

if( rand() <= 0.5 )

begin

if( @f < 0.5 ) set @f = 1

insert monitor.uploads(series\_id, file\_size, create\_date, complete\_date, status\_id)

select 1, x.file\_size, @d, case when dateadd(ss, x.load\_time\*@f, @d) > @dc then null else dateadd(ss, x.load\_time\*@f, @d) end, 0 from @t2 x

end

else

begin

if( @f > 0.5 ) set @f = 0

set @f = @f+1

insert monitor.uploads(series\_id, file\_size, create\_date, complete\_date, status\_id)

select 1, x.file\_size, @d, case when dateadd(ss, x.load\_time\*@f, @d) > @dc then null else dateadd(ss, x.load\_time\*@f, @d) end, 0 from @t3 x

end

end

else

begin

insert monitor.uploads(series\_id, file\_size, create\_date, complete\_date, status\_id)

select 1, x.file\_size, @d, case when dateadd(ss, x.load\_time, @d) > @dc then null else dateadd(ss, x.load\_time, @d) end, 0 from @t x

end

set @d = dateadd(ss, 3, @d)

set @i = @i +1

end

set @j = @j +1

exec [monitor].[new\_point] @series\_id = 1

truncate table monitor.uploads

end

--select \* from monitor.time\_series

exec monitor.scan @series\_id = 1

**THIS IS FUNCTIONALITY.SQL**

**if( not exists(select \* from sys.schemas where name = 'monitor') ) exec('create schema monitor')**

**go**

**if( not exists(select \* from sys.types where schema\_id = schema\_id('monitor') and name = 'timeseries') )**

**begin**

**create type monitor.timeseries as table(value float)**

**end**

**go**

**if( not exists(select \* from sys.types where schema\_id = schema\_id('monitor') and name = 'pfunction') )**

**begin**

**create type monitor.pfunction as table(right\_boundary float, cumulative\_probability float, id int)**

**end**

**go**

**if( not exists(select \* from sys.procedures where object\_id = object\_id('[monitor].[get\_pfunction]')) ) exec('create procedure [monitor].[get\_pfunction] as select 1')**

**go**

**alter procedure [monitor].[get\_pfunction]**

**@timeseries monitor.timeseries readonly**

**as**

**/\***

**Generates cumulative probablity table function**

**Returns table of type monitor.pfunction**

**Note that cumulative probability function is descending from 1 to 0.**

**\*/**

**set nocount on**

**declare @t table(value float, id int identity(1,1))**

**insert @t(value)**

**select value from @timeseries order by 1**

**declare @bw float, @min float, @max float, @n float**

**select @n = count(\*), @max = max(value), @min = min(value) from @t**

**-- finding initial bin size**

**declare @iqr float, @n1 float, @n3 float, @q1 float, @q3 float, @f1 float, @f2 float, @nb int**

**set @n1 = (@n +1)/4.00**

**set @n3 = 3\*(@n +1)/4.00**

**if( @n1 != ceiling(@n1) )**

**begin**

**select @f1 = value from @t where id = floor(@n1)**

**select @f2 = value from @t where id = ceiling(@n1)**

**set @q1 = (@f1+@f2)/2**

**end**

**else**

**select @q1 = value from @t where id = @n1**

**if( @n3 != ceiling(@n3) )**

**begin**

**select @f1 = value from @t where id = floor(@n3)**

**select @f2 = value from @t where id = ceiling(@n3)**

**set @q3 = (@f1+@f2)/2**

**end**

**else**

**select @q3 = value from @t where id = @n3**

**-- this is interquartile value**

**set @iqr = @q3-@q1**

**-- using Freedman-Diaconis rule as a "guidance" to calculate initial bin size:**

**set @bw = 2\*@iqr\*power(cast(@n as float), -1/3.00)**

**-- if bin width is zero (which is a rare case) will use some rather arbitrary number**

**if( @bw = 0 ) set @bw = 2\*sqrt(@n)**

**declare @x table(f float, bw float, id int identity(1,1))**

**declare @h1 table(bw float, id int identity(1,1) primary key)**

**declare @h2 table(rb float, id int identity(1,1) primary key)**

**declare @hst table(rb float, frequency int, id int identity(1,1) primary key)**

**declare @hist table(rb float, frequency int, density float, old\_density float, id int identity(1,1) primary key)**

**declare @tp table(rb float, density float, cumulativeProb float, rba float, id int identity(1,1) primary key)**

**set @nb = ceiling((@max-@min)/@bw)**

**-- we are using variable bin size (proportional to value) if otherwise we have nonsensical number of bins**

**if( @nb > @n/3 /\*to keep # of bins reasonably low - rather arbitrary\*/ )**

**begin**

**insert @x(f, bw)**

**select case when value/@bw > 1 then ceiling(value/@bw) else 1 end, case when value/@bw > 1 then ceiling(value/@bw) \*@bw else @bw end**

**from @t**

**order by value**

**insert @h1(bw)**

**select distinct bw from @x order by bw**

**end**

**else**

**begin**

**insert @h1(bw) select top(cast(ceiling(@max/@bw) as int)) @bw from @t**

**end**

**declare @id int, @idmax int, @rb float = 0**

**select @id = min(id), @idmax = max(id) from @h1**

**while @id <= @idmax**

**begin**

**select @rb = @rb +bw from @h1 where id = @id**

**insert @h2(rb) values(@rb)**

**set @id = @id +1**

**if( @rb>@max) break**

**end**

**-- histogramm table**

**insert @hst(rb, frequency)**

**select h.rb, sum(case when t.value is not null then 1 else 0 end)**

**from @h2 h**

**left join @h2 hp on hp.id = h.id-1**

**left join @t t on h.rb >= t.value and (hp.id is null or hp.rb < t.value)**

**group by h.rb**

**order by 1**

**-- p density table**

**insert @hist(rb, frequency, density)**

**select h1.rb, h1.frequency, h1.frequency/@n/(h1.rb-isnull(h2.rb, 0))**

**from @hst h1**

**left join @hst h2 on h2.id = h1.id-1**

**order by h1.id**

**-- approximate "tail" to the right**

**declare @pd float, @x0 float**

**select @pd = density from @hist where id = (select max(id) from @hist)**

**select top 1 @id = id from @hist where density > @pd order by id desc**

**-- linear regression (y = ax +b)**

**declare @xa float, @ya float, @a float, @b float**

**select @xa = avg(rb), @ya = avg(density) from @hist where id >= @id**

**select @a = sum((rb-@xa)\*(density-@ya))/sum(power((rb-@xa),2)) from @hist where id >= @id**

**set @b = @ya -@a\*@xa**

**-- this will be the last point where p density = 0**

**set @x0 = -@b/@a**

**select @idmax = max(id) from @hist**

**-- adjust previous points p density according to regression (so that total probablity = 1)**

**while @id <= @idmax**

**begin**

**update @hist set density = @a\*rb +@b, old\_density = density where id = @id**

**set @id = @id +1**

**end**

**-- now we need to adjust density due to increased length**

**declare @mrb float**

**select @mrb = rb from @hist where id = @idmax**

**if( @x0 > @mrb )**

**begin**

**declare @dx float**

**set @dx = @mrb -(select rb from @hist where id = @idmax-1)**

**update @hist set density = density\*@dx/(@x0 -@mrb) where id = @idmax**

**end**

**delete from @hist where rb > @x0**

**insert @hist(rb, density) values(@x0, 0)**

**insert @tp(rb, density)**

**select rb, density from @hist order by id**

**-- calculate cumulative probability**

**declare @cp float, @idmin int**

**select @idmin = min(id), @idmax = max(id) from @tp**

**set @id = @idmax**

**while @id > @idmin**

**begin**

**select @cp = sum((h1.rb-isnull(h2.rb, 0))\*h2.density)**

**from @tp h1**

**join @tp h2 on h2.id = h1.id-1**

**where h1.id >= @id**

**update @tp set cumulativeProb = @cp where id = @id-1**

**set @id = @id -1**

**end**

**update @tp set cumulativeProb = 0 where id = (select max(id) from @tp)**

**set @id = @idmin**

**while @id <= @idmax**

**begin**

**-- the first record with none zero density should have probability 1 as well**

**update @tp set cumulativeProb = 1 where id = @id**

**select @pd = density from @tp where id = @id**

**if( @pd > 0 ) break**

**set @id = @id +1**

**end**

**-- finally we have a cumulative probability table function**

**select right\_boundary = rb, cumulative\_probability = cumulativeProb, id from @tp order by id**

**go**

**if( not exists(select \* from sys.procedures where object\_id = object\_id('monitor.get\_probability\_by\_value')) ) exec('create procedure monitor.get\_probability\_by\_value as select 1')**

**go**

**alter procedure monitor.get\_probability\_by\_value**

**@pfunction monitor.pfunction readonly,**

**@value float,**

**@probability float output**

**as**

**/\***

**Returns probability of provided observed value (based on @pfunction)**

**\*/**

**set nocount on**

**declare @id2 int, @v2 float, @p2 float, @v1 float, @p1 float**

**select top 1 @id2 = id, @v2 = right\_boundary, @p2 = cumulative\_probability**

**from @pfunction**

**where right\_boundary >= @value**

**order by right\_boundary**

**set @probability = @p2 -- in case if @value == rb**

**if( (select right\_boundary from @pfunction where id = @id2) > @value )**

**begin**

**-- approximate lineary**

**select @v1 = right\_boundary, @p1 = cumulative\_probability**

**from @pfunction**

**where id = @id2 -1**

**if( @p1 != @p2 and @v1 != @v2 )**

**set @probability = (@value-@v2)\*(@p2-@p1)/(@v2-@v1) +@p2**

**end**

**-- if provided value is beyond x-intercept**

**if( @probability is null ) set @probability = 0**

**go**

**if( not exists(select \* from sys.procedures where object\_id = object\_id('monitor.get\_value\_by\_probability')) ) exec('create procedure monitor.get\_value\_by\_probability as select 1')**

**go**

**alter procedure monitor.get\_value\_by\_probability**

**@pfunction monitor.pfunction readonly,**

**@probability float,**

**@value float output**

**as**

**/\***

**Returns value estimate for provided probability (based on @pfunction)**

**This sp requires the following temp table to be created in the same session before sp call:**

**\*/**

**set nocount on**

**declare @id1 int, @v2 float, @p2 float, @v1 float, @p1 float**

**select top 1 @id1 = id, @p1 = cumulative\_probability, @v1 = right\_boundary**

**from @pfunction**

**where cumulative\_probability >= @probability**

**order by cumulative\_probability**

**set @value = @v1 -- in case if cprob == @probability**

**if( (select cumulative\_probability from @pfunction where id = @id1) > @probability )**

**begin**

**-- approximate lineary**

**select @v2 = right\_boundary, @p2 = cumulative\_probability**

**from @pfunction**

**where id = @id1 +1**

**if( @p1 != @p2 and @v1 != @v2 )**

**set @value = (@probability -@p2)\*(@v2 -@v1)/(@p2 -@p1) +@v2**

**end**

**go**

**if( not exists(select \* from sys.tables where object\_id = object\_id('monitor.time\_series\_config') ) )**

**begin**

**create table monitor.time\_series\_config(**

**series\_id int not null identity(1,1),**

**series\_name varchar(256) null,**

**window\_width\_seconds int not null,**

**max\_number\_of\_points int not null,**

**threshold\_probability float not null,**

**notify\_if\_found\_more\_than int not null,**

**db\_mail\_profile varchar(128) null,**

**email\_recipients varchar(1000) null,**

**email\_subject varchar(512) null,**

**constraint PK\_time\_series\_config primary key(series\_id)**

**)**

**end**

**go**

**/\***

**window\_width\_seconds: should be selected so that a substantial number of records (a half for instance) within it**

**have their complete\_date set (that is completed)**

**max\_number\_of\_points: arbitrary number, large enough to provide sufficient number of point for our math (400-600)**

**threshold\_probability: the target probability we consider low enough to warrant an alert**

**notify\_if\_found\_more\_than: normally should be 0 but sometimes we want to make sure that it was not just a freak event**

**but something that persists, especially useful if you run your scan quite frequently**

**\*/**

**if( not exists(select \* from monitor.time\_series\_config) )**

**begin**

**-- TEST DATA - USE YOUR OWN INSTEAD**

**insert monitor.time\_series\_config(window\_width\_seconds, max\_number\_of\_points, threshold\_probability, notify\_if\_found\_more\_than, series\_name, db\_mail\_profile, email\_recipients, email\_subject)**

**values(300, 250, 0.0015, 0, 'Uploads', 'Profile1', null, 'Monitor-<series\_name>: Value <value>, Probability <probability>')**

**end**

**go**

**if( not exists(select \* from sys.tables where object\_id = object\_id('monitor.upload\_time\_series')) )**

**begin**

**create table monitor.time\_series(**

**id bigint not null identity(1,1),**

**series\_id int not null,**

**create\_time datetime not null,**

**value float not null,**

**probability float null,**

**constraint PK\_time\_series primary key(id)**

**)**

**end**

**go**

**if( not exists(select \* from sys.procedures where object\_id = object\_id('monitor.new\_point')) ) exec('create procedure monitor.new\_point as select 1')**

**go**

**alter procedure [monitor].[new\_point]**

**@series\_id int**

**as**

**/\***

**Metric: a ratio of uncompleted uploads in a period to the total number of uploads in the same period**

**Adds new observed value (point) to the time series and calculates its probability.**

**Also implements "sliding window" by removing the very first point (if applicable)**

**\*/**

**set nocount on**

**declare @timeseries monitor.timeseries**

**declare @pfunction monitor.pfunction**

**declare @now datetime = getdate()**

**declare @window\_width\_seconds int, @max\_number\_of\_points int, @threshold\_probability float**

**declare @error\_status int = -1**

**select @window\_width\_seconds = window\_width\_seconds,**

**@max\_number\_of\_points = max\_number\_of\_points,**

**@threshold\_probability = threshold\_probability**

**from monitor.time\_series\_config**

**where series\_id = @series\_id**

**declare @d1 datetime, @d2 datetime**

**set @d2 = @now**

**set @d1 = dateadd(ss, -@window\_width\_seconds, @now)**

**declare @current\_value float**

**-->> USE YOUR TABLE HERE INSTEAD OF monitor.uploads**

**select @current\_value = 1 -sum(case when complete\_date is not null and status\_id != @error\_status then 1 else 0 end)**

**/cast(case when count(\*) > 0 then count(\*) else 1 end as float)**

**from monitor.uploads**

**where create\_date >= @d1**

**and create\_date < @d2**

**and isnull(file\_size,0) > 0**

**--<< USE YOUR TABLE HERE INSTEAD OF monitor.uploads**

**declare @current\_probability float**

**-- finding current probability**

**-- load time series**

**insert @timeseries(value)**

**select value**

**from monitor.time\_series**

**where series\_id = @series\_id**

**-- re-calc cumulative probabilities (only if we have sufficient # of points, the number is rather arbitrary - you can use a different one)**

**if( (select count(\*) from @timeseries) >= 48 )**

**begin**

**insert @pfunction(right\_boundary, cumulative\_probability, id)**

**exec monitor.get\_pfunction @timeseries = @timeseries**

**exec monitor.get\_probability\_by\_value @pfunction = @pfunction, @value = @current\_value, @probability = @current\_probability output**

**end**

**-- add current point to the time series**

**insert monitor.time\_series(series\_id, create\_time, value, probability)**

**values(@series\_id, @now, @current\_value, @current\_probability)**

**-- if # of point > max limit remove the very first one**

**if( (select count(\*) from monitor.time\_series where series\_id = @series\_id) > @max\_number\_of\_points )**

**begin**

**delete from monitor.time\_series where id = (select min(id) from monitor.time\_series where series\_id = @series\_id)**

**end**

**go**

**if( not exists(select \* from sys.procedures where object\_id = object\_id('monitor.scan')) ) exec('create procedure monitor.scan as select 1')**

**go**

**alter procedure [monitor].[scan]**

**@series\_id int**

**as**

**/\***

**Scans time series to find if alarm/notification is warranted**

**\*/**

**set nocount on**

**declare @t table(probability float, value float)**

**declare @notify\_if\_found\_more\_than int, @threshold\_probability float**

**declare @series\_name varchar(256), @db\_mail\_profile varchar(128), @recipients varchar(1000), @subject varchar(1000)**

**select @notify\_if\_found\_more\_than = isnull(notify\_if\_found\_more\_than, 0),**

**@threshold\_probability = threshold\_probability,**

**@series\_name = isnull(series\_name, cast(series\_id as varchar(50))),**

**@db\_mail\_profile = db\_mail\_profile,**

**@recipients = email\_recipients,**

**@subject = email\_subject**

**from monitor.time\_series\_config**

**where series\_id = @series\_id**

**insert @t(probability, value)**

**select top(@notify\_if\_found\_more\_than +1) probability, value from monitor.time\_series where series\_id = @series\_id order by id desc**

**if( (select count(\*) from @t where probability < @threshold\_probability) <= @notify\_if\_found\_more\_than )**

**return**

**if( @db\_mail\_profile is not null and @recipients is not null )**

**begin**

**declare @value float, @probability float**

**select top(1) @probability = probability, @value = value from @t where probability < @threshold\_probability**

**set @subject = replace(replace(replace(@subject, '<series\_name>', @series\_name), '<value>', @value), '<probability>', @probability)**

**exec msdb.dbo.sp\_send\_dbmail**

**@profile\_name = @db\_mail\_profile,**

**@recipients = @recipients,**

**@subject = @subject**

**end**

**else**

**print 'Value with less than threshold probability encountered!'**

**go**