Fast and Accurate Calculation of Descriptive Statistics of Very Large Sets of Data

Area: Quantiles, Mean and Standard Deviation, Computational and Descriptive Statistics
Author: Anders Sköllermo Ph.D. **2023-09-12**

Abstract: We present a method for computing descriptive statistics which works well, and is very fast, also when the set of data is very large. We calculate the mean, the standard deviation and many quantiles. The method is usually a one-pass method.

The computed values are accurate, but not exact. We can decide a wanted accuracy before-hand. The requirements for implementing our method, called the Slot method, are a computer language which supports large and fast arrays and fast internal read/write in memory.

The classical (OS) method computes quantiles by sorting the data, and calculates the quantiles exactly. These values are often rounded to 3 or 4 correct digits when used.

The basic idea of our method is to do the reverse - approximate at the start and avoid sorting. We use this "lack of precision" to make the calculations faster. The method works well also for distributions that are highly skewed. It is intended for calculating descriptive statistics for static data, with positive and negative values. The method is quite general and not based on any specific computer language.

It has been tested thoroughly without any problems or malfunction, for data volumes up to 500 M data values and relative accuracy up to at least 6 correct digits. In most cases the Slot method is significantly faster than the OS method. In many cases no method can be much faster.

Basic facts and ideas:

1 - It is fast and easy to perform calculations on data in an array, also for arrays of several thousand elements.

2 - A large set of decimal values can be mapped onto a set of integer values, using a simple linear transformation. Each integer value corresponds to a narrow range of decimal values.

3 - Idea: Let the integer values be indices in a large array. Let each array element contain the number of original decimal values that are mapped to the corresponding index value.

4 - Method: We read data values from a large set of random data and use a linear transformation to calculate an integer number, which is used as index in an array. We then update a counter in this array element, but do not save the original data value. When all data values have been processed the original set of data has been converted to an ordered set of approximate data, where each index corresponds to a very narrow interval or slot.

For quantiles we will use the notation q25 for the smallest value, for which 25 percent of all the sorted data values are less than or equal to this value. q25 is often called the 25% quantile, or simply the 0.25 quantile. The q25 value does not need to be a value present in the data.

We have used the Slot method to calculate descriptive statistics (mean, standard deviation and 15 different quantiles) for a basic example of 5M data values.

The cpu-time needed to read the data and calculate all the statistical values, with an accuracy of 4 correct digits, using the Slot method, is only slightly more than the cpu-time needed to read the data and calculate the mean and the standard deviation exactly, but no quantiles at all.

We have scaled the basic example to 2M, 20M and 200M when testing this. (See Table 4 for details). The cpu-time mentioned above, is twice the cpu-time required to only read the data, but not perform any calculation at all. The Slot method often uses less than 10% of the cpu-time for the OS-method, independent of the size of the set of data, when 4 correct digits are required.

Background: An overview of the quantile concept is given in (1). Quantiles are calculated in descriptive statistics, using standard statistical software (2), (3), usually together with the mean, the standard deviation (called "stddev" below) and perhaps other statistical quantities.

The computation of the quantiles of large sets of data is a classical problem. In the OS-method we first sort the data, and then read the sorted data to compute the descriptive statistics. This is the traditional solution and has been implemented in several data handling packages e.g. (2) and (3). The method is very safe and reliable and based on system sort and user written code to calculate the descriptive statistics. There is an upper limit to the size of the set of data, because of the memory requirements.

The method we suggest in this paper is like using a very large number of bins in a histogram (4). But we will call them slots, because they are very, very narrow. The discussion below will use a general *"basic example*", a set of data obtained from a known theoretical distribution, with positive and negative data values.

A simple example - The Slot method: We start with a set of data values, distributed with positive and negative values, along the x-axis. We denote the values by *Xi, i = 1, 2, 3, ,* ***Totpop*** (total number of data values). The values are not sorted.

We have chosen to divide the x-axis into three different regions: The Middle region where we want all *Xi*-values of interest to fall, and the regions below ("Low") and above ("High").

A "Range of interest" (the Slot Range, with the width *SRange*) must be identified on the x-axis. This is the Middle region with lower boundary ***LLIM*** and upper boundary ***ULIM***. The Slot Range is the part of the x-axis, that is most important to include in our investigations. It shall be chosen so big, that all quantiles of interest fall in it.

Next, we divide the Slot Range into ***Nslot*** (total number of slots) equidistant slots, or slot intervals, where Nslot is a large integer number (typically 7500 or 15000). Each slot, from first to last, is numbered (1, 2, , Nslot) and corresponds to a precise small interval on the x-axis.

We define an Array, the *Slot Array* (SA for short*)*, with Nslot elements, before the processing starts. For each slot interval we define an array element, a counter, with initial value zero.

We read an *Xi*-value from our set of data and determine into which slot the *Xi*-value falls, by computing the slot interval number. The counter for this slot is increased by one. An *Xi*-value that falls on the lower slot boundary of a slot, is included in the counter for that slot. A value that falls on the upper slot boundary of a slot, is included in the counter for the next higher slot.

Then we read the next data value and repeat the process, until all data values have been processed. Note that we may end up with zero values in some counters.

The *Xi*-values that fall outside the Middle region, are said to fall in the "Low" or "High" regions and are counted separately in two variables *SALow* and *SAHigh*. No *Xi*-values are ignored in the Slot method, and no *Xi*-values are saved. Only the number of *Xi*-values in each slot and in the "Low" and "High" regions are saved.

When all the data values have been read, we have reduced the very large set of original data values to be represented by Nslot counters in the Slot Array. This array is an ordered set of data. For each slot we have: 1) The counter (array element) which states how many values have fallen in the slot. 2) The lower and upper boundary values. 3) The width of the slot.

We can now compute a set of statistical values using this new set of data. Each quantile value can be localized to a specific slot, which means that the accuracy of the computation depends entirely on the slotwidth. We have only used slots of equal size. In Table 1A we illustrate the method using 10 slots and Slot Range is -1 to +9. (Nslot=10, SRange= 10, LLIM= -1, ULIM= +9).

Table 1A: Definition of the Low, Middle and High regions.

	"Low" below Middle	**M i d d l e**										**"High" above Middle**
Slot No.		1	2	3	4	5	6	7	8	9	10	
Lower boundary	Minus infinity	-1	0	1	2	3	4	5	6	7	8	9
Upper boundary	-1	0	1	2	3	4	5	6	7	8	9	Plus infinity
Middle value		-0.5	0.5	1.5	2.5	3.5	4.5	5.5	6.5	7.5	8.5	

We assume that we have some information about suitable values for the Slot Range. If not, we can use very large values for SRange and Nslot. The total cpu-time increases slowly with increasing value of Nslot. If we change SRange 10 → 1000 and Nslot 5k → 500k (k stands for thousand), the total cpu-time only increases a little, but we still use the same slotwidth (0.002). (See Fig. 3B below for details).

<u>The calculations – Four steps</u>:

First – create the base for the search of slot number. We define the Middle region, with the Slot Range (with total width SRange, lower range boundary LLIM and the upper range boundary ULIM) and the number of intervals (Nslot). Each slot has a specific number j (1 to Nslot). Each slot represents a small interval on the x-axis with `Slotwidth = (ULIM - LLIM)/NSLOT`. This is a declaration and requires no cpu-time.

Second - Define an array, the "Slot Array" SA, with Nslot elements. Each array element SA(j) is initialized to zero. It will be used as a counter, of the number of data values that fall in slot j, between the inclusive lower boundary and the non-inclusive upper boundary (`LLIM + (j-1)*Slotwidth, LLIM + j*Slotwidth`).

Define two variables SALow and SAHigh and initialize them to zero. SALow is used as a counter of the number of *Xi*-values that fall below the lower boundary of the Middle region (i.e `Xi < LLIM`). SAHigh is used as a counter of the number of *Xi*-values that fall on or above the upper boundary of the Middle region (i.e `Xi >= ULIM`).

Define two variables SumXi and SumXi2 and initialize them to zero. For *Xi*-values outside the Middle Region, SumXi will be used to sum the *Xi*-values and SumXi2 will be used to sum the square of the *Xi*-values. These variables will enable us to compute the mean and the stddev with high accuracy. These declarations require no cpu-time.

Third – Fill the "Slot Array" - Read/fetch an *Xi*-value from the original big set of data. Use linear transformation to find the slot number j for this *Xi*-value. When j is between 1 and Nslot we increase the array element SA(j), for slot j, by one.

For *Xi*-values that fall outside the Middle region, the slot number is outside the limits 1 to Nslot. If the number is less than 1, we increase SALow by one and update SumXi and SumXi2. If the number is greater than Nslot, we increase SAHigh by one and update SumXi and SumXi2.

The linear transformation "maps" the interval LLIM to ULIM `(LLIM <= Xi < ULIM)` with *Xi*-values, into the interval 1 to Nslot `(1<= j <= NSLOT)` with slot numbers. Every *Xi*-value that is bigger than LLIM and smaller than ULIM, will be transformed into an integer number j, which lies between 1 and Nslot. Slot number 1 is the interval `(LLIM to LLIM+Slotwidth)`. Slot number Nslot is the interval `(ULIM–Slotwidth to ULIM)`.

Repeat the process until all *Xi*-values have been read/fetched. For each slot only the number of *Xi*-values that correspond to that interval is registered, not the actual *Xi*-values. The loop through the set of *Xi*-values is the main cpu-cost.

Each array element in the "Slot Array" SA, now contains the total number of *Xi*-values that fall within the boundaries of that slot. SA consists of only one single row with Nslot values.

Fourth – Use the "Slot Array": We can now compute all the wanted statistical quantities, like mean, stddev and all wanted quantiles, using the values in the Slot Array SA, the slot boundary values, SALow, SAHigh, SumXi and SumXi2.

Compute the mean and the stddev: The calculations of the mean and the stddev are special. All *Xi*-values that fall outside the Middle Region contribute a lot to the mean and the stddev. In the Third step, we have updated SumXi and SumXi2. This means that the contributions to the mean and the stddev from these *Xi*-values are calculated exactly.

When all the data values have been read, we loop through the Slot Array and compute the contribution to the mean and stddev from each slot, using the x-value of the middle of the slot.

Since the mid slot value represents each *Xi*-value with an error of at most half the slotwidth, the total impact on the calculated mean value is at most half the slotwidth. Similarly, the error in the calculated stddev is at most half the slotwidth. We can show theoretically that these errors are much smaller than one half slotwidth. The full code is in the Appendix.

If we use a small Middle Region the mean and the stddev will still be computed with high accuracy, since these calculations are exact on all values that fall outside the Middle Region. The cpu-time will increase since these calculations are performed on a large set of data values. We can use Richardson extrapolation to increase the accuracy, but we have not found it necessary.

Compute quantiles: For each desired quantile we can determine exactly in which specific slot the quantile value lies. The mid-point value of the slot interval can be used to estimate the quantile. We call this the mid-point rule (MP-rule). The values are approximate, with a conservative error estimate (maximum error) equal to half the slotwidth. The actual errors are often smaller.

We have also calculated the quantiles using a linear interpolation rule. This often gives a strongly reduced error. Theoretically this has a maximum error of one slotwidth.

By choosing a suitable value for the number of slots (Nslot), we can beforehand make sure that the maximum absolute error of all parts of the descriptive statistics meets the specified accuracy requirements.

The calculations of the mean, the stddev and all wanted quantiles only depend on the number of slots and not on the size of the original set of data. When Nslot is considerably smaller than Totpop, this step only requires a little cpu-time. (See Table 4 below for details).

Apply the Slot method: Let us now apply the Slot method on our simple example, defined above. Below in Table 1B we show the use of 10 slots (Nslot= 10). Slot Range is -1 to +9, (SRange= 10). Any quantile which falls within Slot1 – Slot10 has a maximum error of 0.5. The lower boundary is included in a slot, but not the upper boundary. The data values used below are the same as used in (5). Note that one data value, the value zero, belongs to Slot 2 ("0-1") and thus SA (2) equals 1.

In this example the SA-values and the MP-rule give: q10= 0.5, q25=1.5, q50= 2.5, q75= 4.5, q90= 5.5. Below SumSA is the sum of the SA-values, up to and including the current value. SumSAN, (the CDF, Cumulative Distribution Function), is the summarized and normalized value of the SA-values.

Table 1B: Example Data values: 0, 1, 1, 1, 2, 2, 2, 4, 5, 8. Totpop= 10.

	1	2	3	4	5	6	7	8	9	10
Slot interval	***-1 to 0***	***0-1***	***1-2***	***2-3***	***3-4***	***4-5***	***5-6***	***6-7***	***7-8***	***8-9***
Lower boundary	*-1*	*0*	***1***	***2***	***3***	***4***	***5***	***6***	***7***	*8*
Data Values		***0***	***1,1,1***	***2,2,2***		***4***	***5***			***8***
SA – Slot Frequency	***0***	***1***	***3***	***3***	***0***	***1***	***1***	***0***	***0***	***1***
SumSA	***0***	***1***	***4***	***7***	***7***	***8***	***9***	***9***	***9***	***10***
SumSAN – CDF	***0.00***	***0.10***	***0.40***	***0.70***	***0.70***	***0.80***	***0.90***	***0.90***	***0.90***	***1.00***

Our basic example: We want to investigate how the Slot method works on a large set of data. Since we do not have access to real data, we will use a known theoretical distribution to create a large set of unordered data. We have created 5 M (million) random values from a Gumbel distribution. We refer to this set of data as the EDS (the Experimental Data Set), with the data values *Xi, i = 1, 2, , , Totpop*. The Gumbel distribution is only used to generate the EDS. None of the conclusions below depend in any way on this choice of example.

The yearly salaries, including the social cost, in millions of SEK of all employees in Sweden, about 5 M persons, are distributed somewhat like the positive values of the Gumbel distribution, with *my*= 2 and *scale* (also called beta) = 1. This distribution is highly skewed, see (6). Typical values are mean= 2.577 216, stddev= 1.282 550, median= 2.366 513.

This choice of parameters means that negative values are included in the set of data and in the analysis. This is made on purpose and does not cause any problems in the Slot method. We want to show that the interpretation "Income for all employees in Sweden" is only an example.

We have chosen the Slot Range boundaries as -1 and 14, which gives the Slot Range 15, and corresponds to a span in yearly income of 15 M SEK. Then all but very extreme quantiles are included. The number of data values outside these limits is very small. Nslot is determined by the wanted accuracy. Some usable values are 7.5k, 30k or 60k, corresponding to the slotwidths 2000, 500 and 250 SEK. (See Table 3 below for details).

The Gumbel distribution – in theory and used in the Slot method: In Fig. 1A we show the Gumbel distribution, with *my*= 2 and *scale*= 1, theoretical curves for pdf (blue) and cdf (red). The 15 quantiles have been marked. In Fig 1B we show the Slot method applied to Gumbel random values. A detail is shown in Fig. 1C. The count of *Xi*-values in each element in the Slot Array SA has been plotted. The blue curve shows the actual number of values in each slot. The hidden red curve shows the theoretical Gumbel cdf slotted - each slot shows the upper cdf value minus the lower cdf value. These values have been scaled by Totpop. Also shown are 12 quantiles.

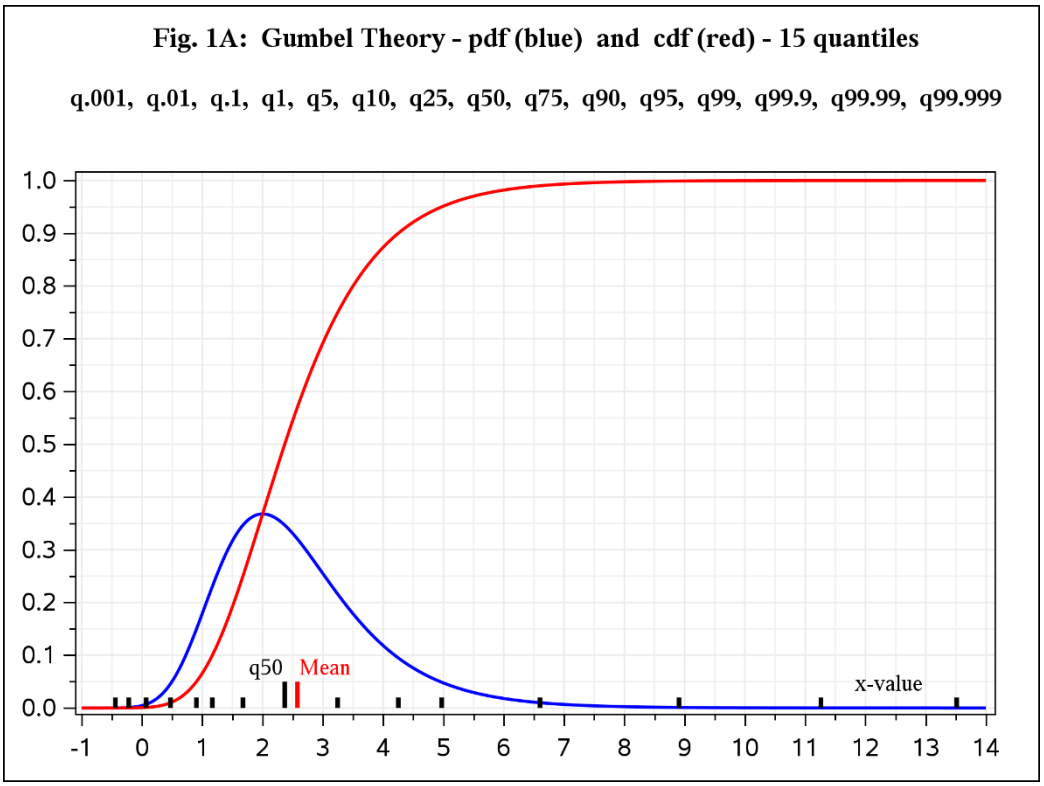

Fig. 1A: The Gumbel distribution – Theoretical curves for pdf (blue) and cdf (red), *my*= 2 and *scale*= 1. The 15 quantiles have been marked.

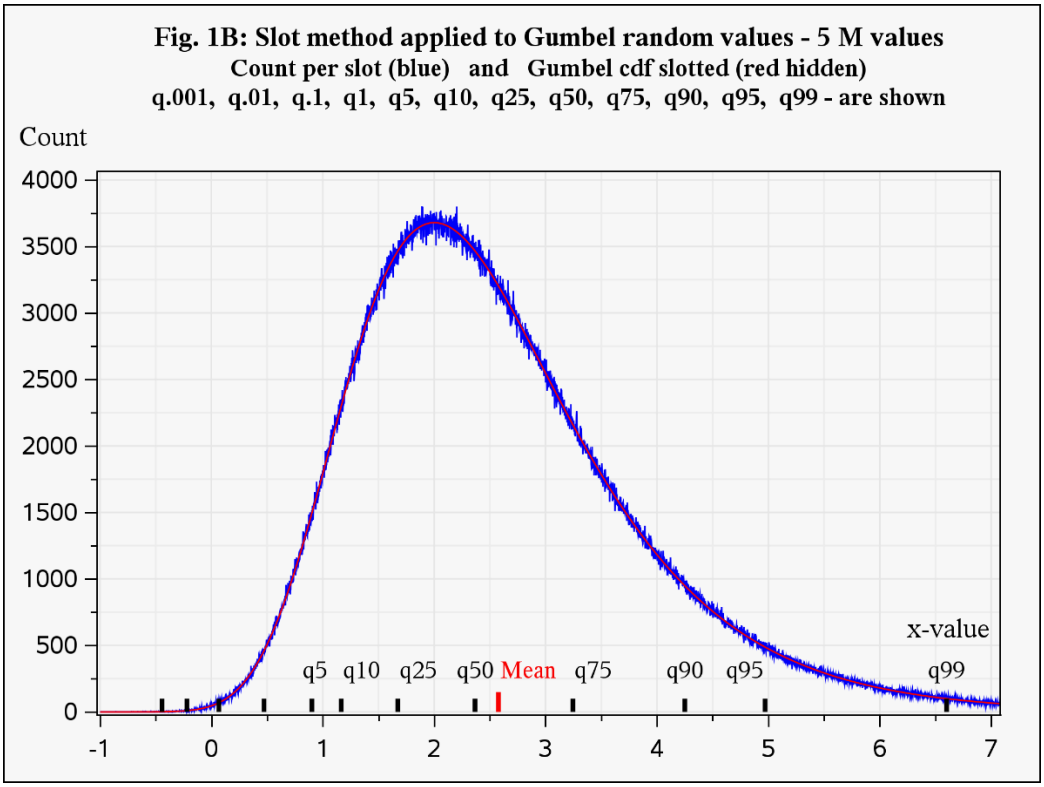

Fig 1B: The Slot method applied to Gumbel random values.

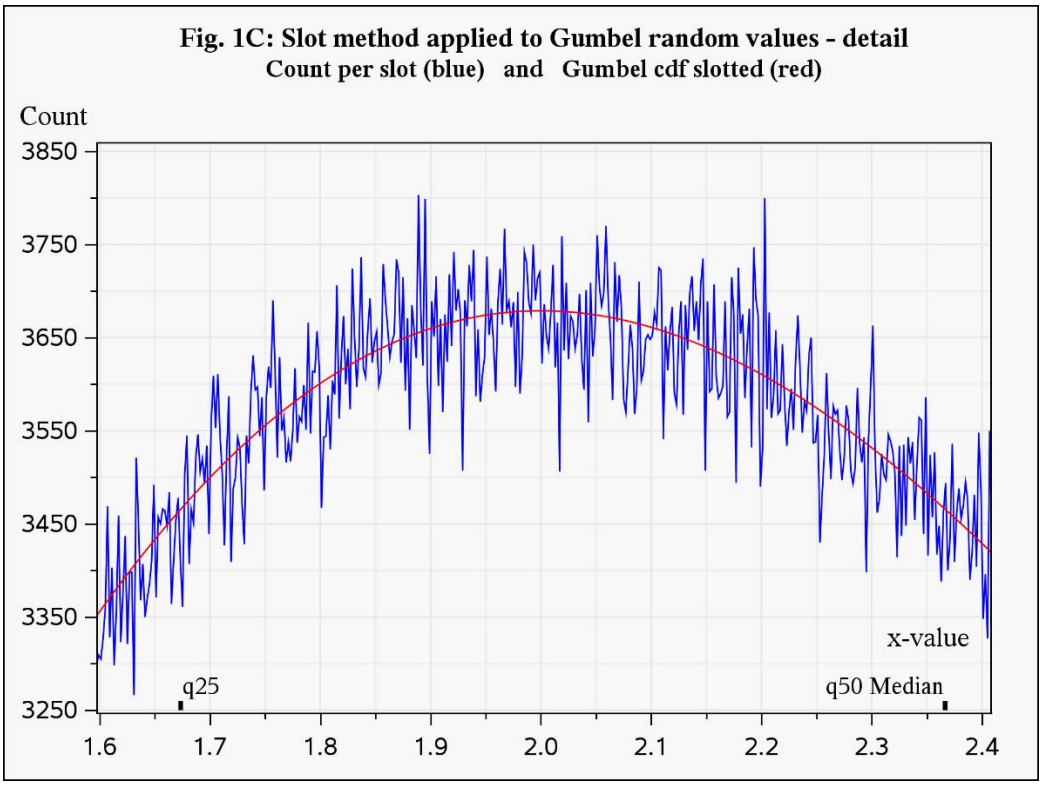

Fig. 1C: The central part of Fig. 1B. We clearly see the uneven distribution of the *Xi*-values.

<u>The linear transformation</u> – details: We want to associate each *Xi*-value (a random decimal value) with an integer slot number. Fetch/read the *Xi*-value from the EDS. Calculate `Xi - LLIM`.

Now multiply with the inverse of the Slotwidth, to get the slot number j. See Table 2 below for an overview of the lower and upper slots, and typical *Xi*-values. We have used SRange= 15 and Nslot= 7.5k (7500), which gives the Slotwidth= 0.002.

Table 2: Some values in the basic example, and the slot number.

	"Low"	M i d d l e					*"High"*
Slot Number	*<= 0*	1	2		NSLOT-1	NSLOT	*>= NSLOT+1*
Lower Boundary - inclusive	***Minus infinity***	-1 (LLIM)	-0.998		13.996	13.998	***14.0 (ULIM)***
Upper Boundary -non-inclusive	***-1 (LLIM)***	-0.998	-0.996		13.998	14 (ULIM)	***Plus infinity***
Typical x-value		-.99812	-0.99736		13.99634	13.99812	
No of values	SALow	SA(1)	SA(2)		SA(NSLOT-1)	SA(NSLOT)	SAHigh

We first once calculate `Inv_Slotwidth = NSLOT/(ULIM-LLIM)` and then use it many times to compute `SlotNo= 1 + FLOOR((Xi-LLIM)*Inv_Slotwidth)`. The function FLOOR rounds the value down to the closest lower integer, which is positive, zero or negative. See (7).

The value of SlotNo is an integer. If the *Xi*-value is less than LLIM, the value of SlotNo will be less than 1 and we increase the counter SALow by one. If the *Xi*-value is greater than or equal to ULIM, the value of SlotNo is greater than Nslot and we increase the counter SAHigh by one.

When the *Xi*-value is greater or equal to LLIM and less than ULIM, the variable SlotNo will have a value between 1 and Nslot. Now, we can use the SA array, and increase the counter in array element SA(SlotNo) by one.

The Accuracy: For every quantile, we can always find the first slot, where the accumulated normalized count is greater than or equal to the desired quantile value. We have primarily used the mid-point value of this slot as the estimate of the value of the quantile. This means that the absolute error, in general, is at most half the slotwidth. The exception is when the quantile falls outside the Middle Region.

We use SALow, the counts in the array SA, SAHigh and the information about the slot boundary values, to calculate the quantiles. Special rules may be implemented e.g., when the data values are very sparse, and some slots are empty.

SRange has been chosen as 15 (the limits -1 to +14), since "almost all" quantiles are then included. For the Gumbel distribution, e.g. q99 - q1= 6 and q99.999 - q.001= 13.95.

There are several ways to define the number of correct digits. The absolute quantile error is one half slotwidth. The relative error is one half slotwidth divided by SRange. An estimate of ***the relative accuracy*** is then: `log10(SRange/Half Slotwidth)` or better ***`log10(2*Nslot)`***, since "Half Slotwidth" = SRange/(2*Nslot). Below in this text accuracy always means relative accuracy.

This is often a rather conservative estimate. The error in our basic example is often smaller. We here assume that SRange is large enough to include all quantiles whose value we want to estimate. The errors in the calculated quantities also depend on which rule is used in the computations. To use the midpoint rule (use the midpoint of a slot) is very simple and safe.

Table 3 below shows that when Nslot is increased from 7.5k to 5000k (k= thousand), the relative accuracy (in red) increases from 4.2 to 7 digits. The absolute error (Half Slotwidth) is decreased from 1000 SEK to 1.5 SEK, since the income span is 15 M SEK.

With 60k slots the maximum absolute error in every quantile is less than 125 SEK, and the relative error is 5.1 digits. The relative accuracy (red) is computed as log10(2*Nslot).

In our basic example, when a user provides the data values, these are often rounded down to the closest 100 SEK. Thus, the descriptive statistics will all have uncertainties of about 100 SEK and the accuracy of this statistics cannot be increased by using larger values of Nslot.

Table 3: Relative and Absolute Errors for different values of NSLOT – 1k to 5000k

NSLOT	1k	**7.5k**	15k	30k	**60k**	125k	250k	**500k**	**1000k**	**5000k**
log10(2*NSLOT)	3.3	**4.2**	4.5	4.8	**5.1**	5.40	5.70	6.00	6.30	7.00
Half Slotwidth *1000	7.5	**1.**	0.5	0.25	**0.125**	0.06	0.03	0.015	0.0075	0.0015
Half Slotwidth	7500	**1000**	500	250	**125**	60	30	15	7.5	1.5

Total cpu-time - for calculating the descriptive statistics: Below we show the cpu-time for calculating the mean, the stddev and 15 quantiles (`q.001, q.01, q.1, q1, q5, q10, q25, q50, q75, q90, q95, q99, q99.9, q99.99 and q99.999`).

The Slot method has been tested for Totpop values from 0 up to 500M, without any kind of problem or malfunction. This holds also in test runs where Nslot is much bigger than Totpop.

In Fig. 2A, 2B and 2C below we have used the following **notations (in the order of the curves):**

`OS` = OS-method. System sort and own calculation of mean, stddev and 15 quantiles.
`5000k` = Slot method. Parameter Nslot= 5000k. Calculations of mean, stddev and 15 quantiles.
`1000k` = Slot method with parameter Nslot= 1000k (i.e. 1M slots). Calculations like 5000k.
`100k` = Slot method with parameter Nslot= 100k. Calculations like 5000k.
`OneMS` = Read all the *Xi*-values and calculate exact values for mean, stddev. No quantiles.

Cpu-time versus Totpop - Nslot is a parameter: We show how the cpu-time depends on Totpop for the OS-method and for the Slot method (for different values of the parameter Nslot). The cpu-times shall be considered as approximate values since all test runs have been made on a platform (8) with several users.

In Fig. 2A, 2B and 2C we show that the Slot method is usually much faster than the OS-method. The Slot method, with Nslot= 100k, produces very exact values for mean and stddev and 15 rather exact quantiles (5.3 digits), using only slightly more cpu-time than OneMS, for a range of Totpop from 0M up to 500M.

The maximum value for Totpop in the OS-method on (8) is 400 M, which depends on the amount of available computer memory. The cpu-time for the OS-method increases like `constant * Totpop`. The same applies to the Slot method, but the constant is much smaller.

No quantiles are calculated in OneMS, and neither the OS-method nor the Slot method is used. OneMS is called Test 4 in Table 4 below.

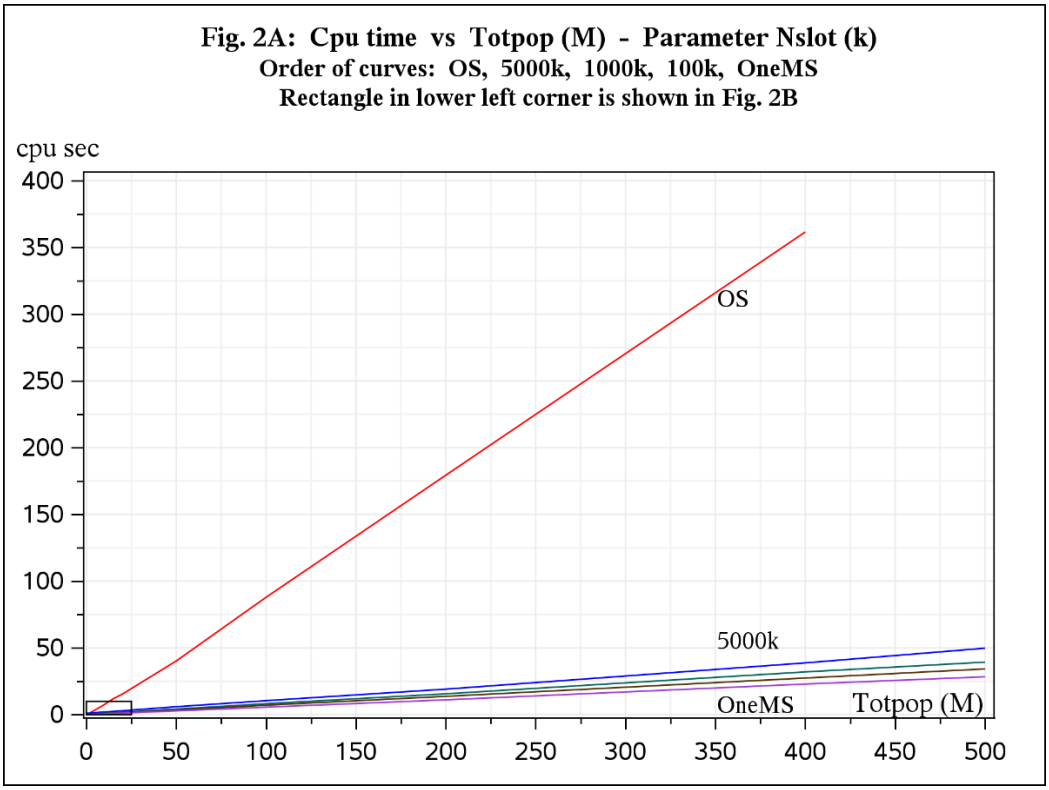

Fig. 2A: CPU – Totpop – The total cpu-time (in seconds) for the Slot method – versus Totpop (the size of the set of data in M), for various values of the parameter Nslot (in k).

In Fig. 2A and Fig. 2B the "cpu-lines" for Nslot= 100k and 1000k both go down to zero, when Totpop is small. When Nslot= 5000k (5M slots, 7 digits) the "cpu-line" for the Slot method does not go down to zero, when Totpop is small. Instead about 1 cpu second is used, when Totpop < 2.5M. See Fig. 2C.

In Fig. 2C we compare the cpu-times for the OS-method and the Slot method, for rather small values of Totpop. All cpu-times are less than roughly 1 second. The OS-method is fast when Totpop < 2 M. When we require slightly lower accuracy, say 5.3 digits (Nslot= 100k), the Slot method is faster than the OS-method, for all values of Totpop.

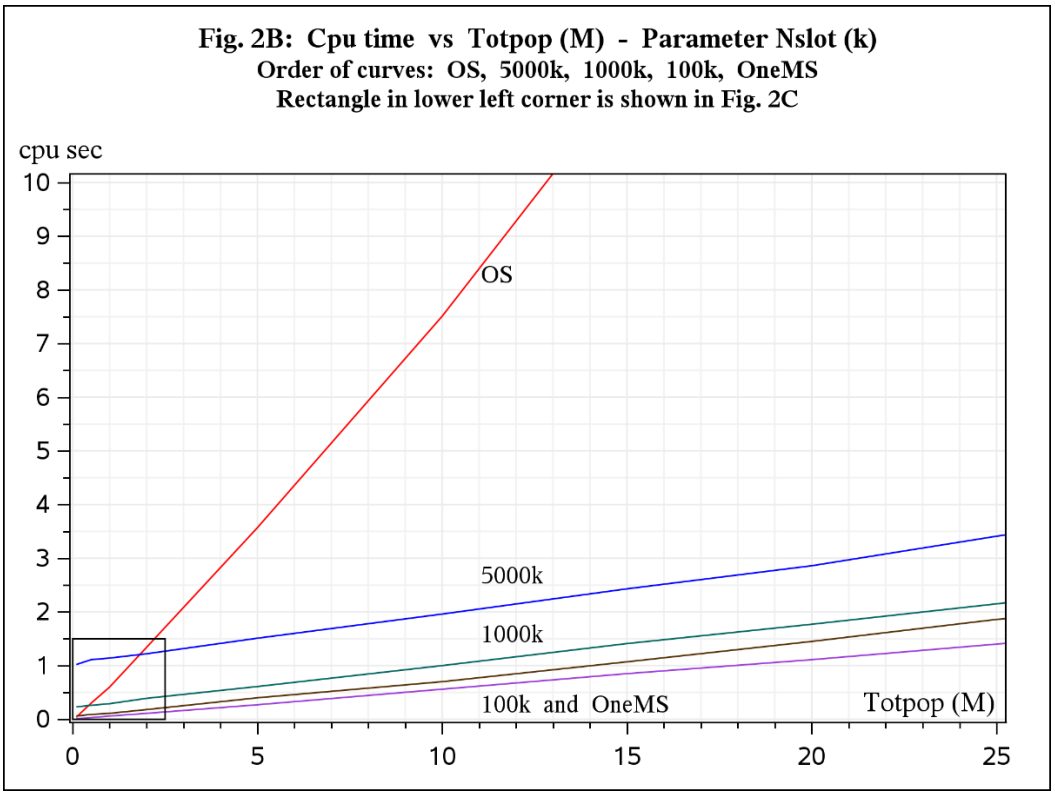

Fig. 2B: CPU – Totpop - The Slot method works well in all these test runs. The rectangle in the lower left corner corresponds to Totpop < 2.5M and cpu-time < 1.5 seconds.

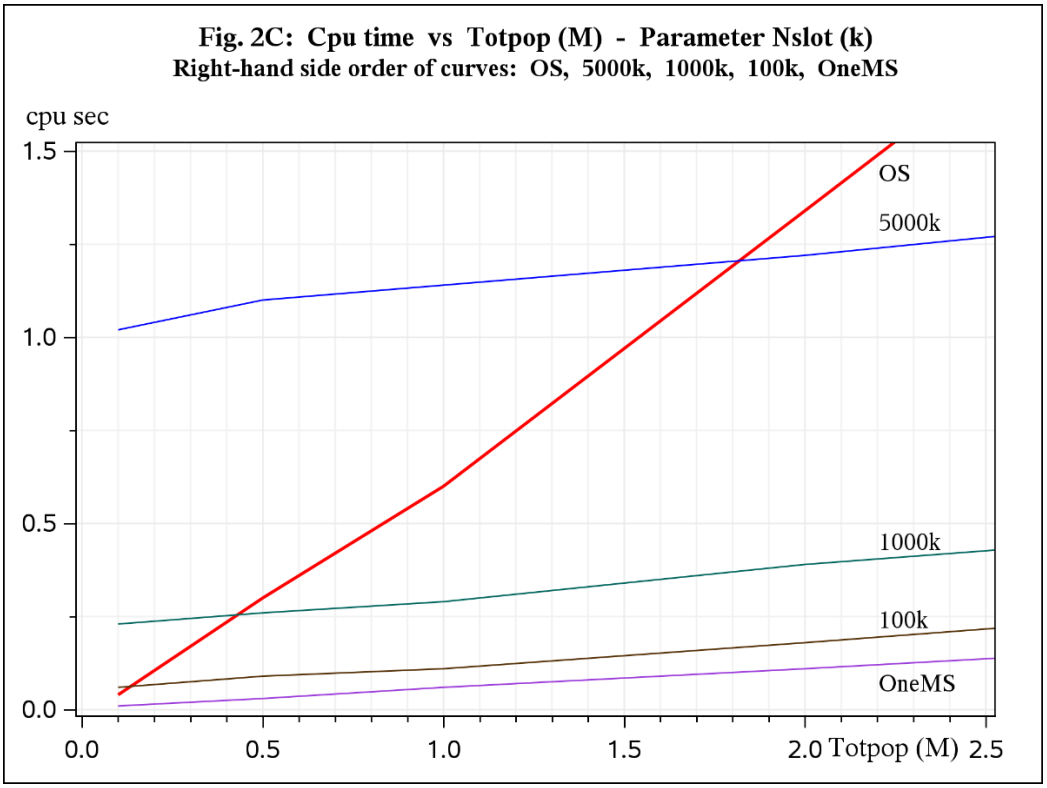

Fig. 2C: CPU – Totpop – The Slot method is useful for all values of Totpop, when 5 digits in accuracy is sufficient. For small Totpop < 2 M, the OS-method is fast.

Cpu-time versus Nslot - Totpop is a parameter: We show how the cpu-time depends on Nslot for different values of the parameter Totpop.

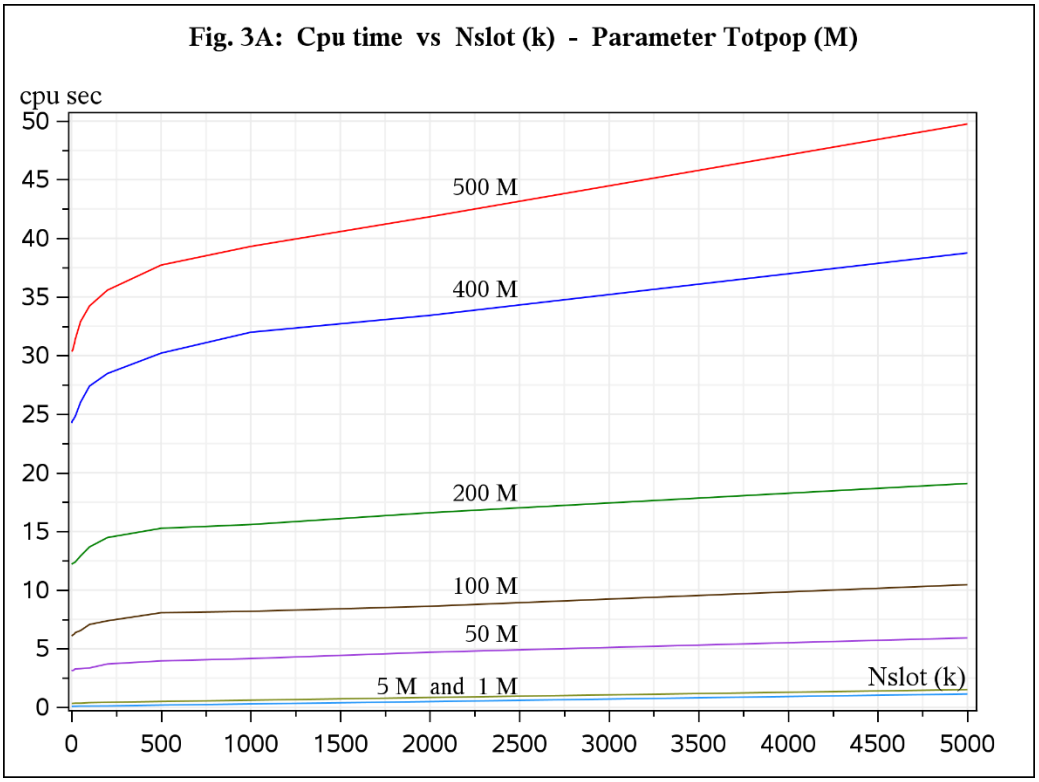

Fig. 3A: CPU – Nslot: Total cpu-time versus Nslot. Totpop is a parameter.

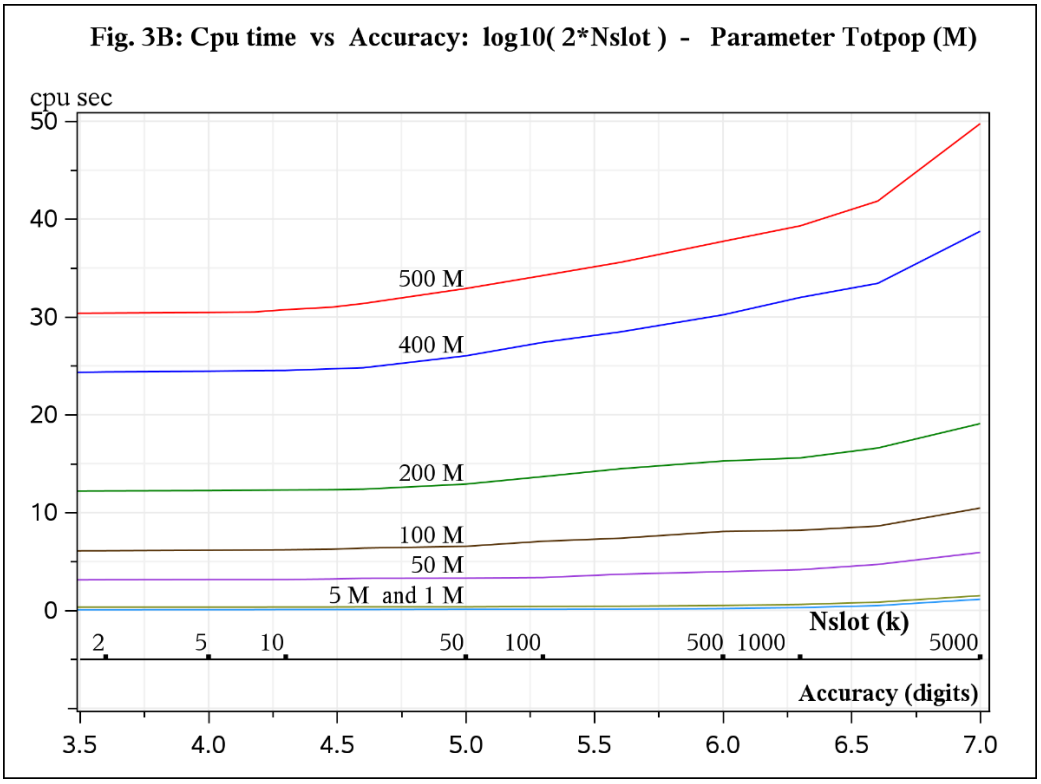

Fig. 3B: CPU – Accuracy: Total cpu-time versus accuracy. Totpop is a parameter.

The relative accuracy is computed as `log10(2*Nslot)`. Often the actual accuracy is higher. The cpu-time depends mainly on Totpop, and only a little on the total number of slots (Nslot). The cpu-time increases slowly with increasing Nslot. If Nslot is doubled once or twice, the cpu-time only increases a little. See Fig. 3A.

If Nslot is big, especially when Nslot is larger than Totpop, many slots will be empty. The Slot method still works very well. The quantiles are computed with high accuracy. The cpu-time depends only slightly on what statistics are calculated. If we decrease the number of quantiles (and/or omit mean and stddev) the cpu-time is only slightly reduced.

Compare cpu-times **-** We have calculated the cpu-time for several test cases, see 1-6 below, in Table 4. We show the total and the relative cpu-times, with Totpop= 2M, 20M and 200 M. In the Slot method we used Nslot= 7.5k (7500 slots), which gives at least 4 digits.

The cpu-time for calculating the descriptive statistic (the mean, the stddev and the 15 quantiles) is only a minor part of the total cpu-time. In the Slot method, reading/fetch a data value, calculating the slot number, and updating the counter in the array, take almost all the cpu-time. It depends mainly on Totpop, and only a little on Nslot.

The test cases are (MS15Q below is short for computing mean, stddev and 15 quantiles):

Test 1 - OS-method and all the MS15Q calculations.
Test 2 - Slot method and all the MS15Q calculations.
Test 3 - Fill the Slot Array. No calculation at all is made of descriptive statistics.
Test 4 - OneMS. Read once. Calculate the mean and stddev (MS). No quantile is computed.
Test 5 - Read all the data, but do not use them in any way. No results are computed.
Test 6 - Read raw data only (convert the numbers to internal representation) from a flat file.

A comparison of Test 2 and Test 3 shows that the calculation of the descriptive statistics in the Slot method, is very fast. (It is difficult to measure the cpu difference).

In Test 3 we can save the Slot Array SA to disc, and later in the post-processing choose what statistical quantities shall be calculated, and what rules shall be used.

Test 4 is called OneMS in Fig. 2A, 2B and 2C. Neither the OS nor the Slot method is used.

Table 4: The cpu-time for some test cases. Only Tests 1 and 2 produce MS15Q results.

	Totpop:	2 M data		20 M data		200 M data	
	Rel. = relative order of rows, for cpu sec	*Rel.*	*sec*	*Rel.*	*sec*	*Rel.*	*sec*
1	OS-method – MS15Q.	20.	1.38	24.	15.0	28.	175.
2	Slot method, calculate MS15Q.	2.1	0.15	2.0	1.24	2.0	12.1
3	Slot method, calculate no statistics.	1.9	0.13	2.0	1.24	2.0	12.1
4	OneMS: Read, calculate MS, write two values.	1.7	0.12	1.8	1.13	1.8	11.2
5	Read only – no calculations, no write.	1.0	0.07	1.0	0.63	1.0	6.3
6	*Read raw data only.*		*0.38*		*3.72*		*35.5*

Based on the values in Table 4 we do not see any opportunity to design a method for calculating the mean, the stddev and the 15 quantiles, for the specified accuracy, that are significantly faster than the Slot method. The reason is simply that any method must read the data, perform some rearrangement and calculations, and finally write the results.

Details of the Slot method*: Estimating the parameters – Quantiles not computed - Some rules for quantile calculations.*

Estimating the Middle region - In all the computations we assume that the Middle region, which is defined by the parameters LLIM and ULIM, is so big that only a small part of the *Xi*-values falls outside this region.

If we have little knowledge of the data values, we shall choose a very wide Middle region (small LLIM and large ULIM). Then choose Nslot based on the accuracy wanted. To double the value of Nslot once or twice, only means a slight increase in the cpu-cost. See Fig. 3A and Fig. 3B.

If we have NO knowledge of the data values, we first calculate some basic statistics (like q5, q50 and q95) using only a small random subset, say 1% of the data values and a very, very wide Middle region. The Slot method performs well also in this case. We will not discuss this any further since there are standard ways of estimating basic parameters.

Quantiles not computed: If we use a Middle region which is small or misplaced, the values for some quantiles may fall outside the Slot Range and cannot be estimated. When the quantile is less than *(SALow + 0.5*Slotwidth/ SA(1))/ Totpop*, (e.g. 0.01), the estimate will fall in the Low region. When the quantile is greater than *1 - (SAHigh – 0.5*Slotwidth/ SA(Nslot))/ Totpop* (e.g. 99.99) the estimate will fall in the High region. We have not computed these quantiles. See the Appendix.

These expressions are roughly *SALow/ Totpop* and *1 - SAHigh/ Totpop .* They depend only on LLIM and ULIM. If Nslot or/and Totpop is doubled, these quantities are not affected.

The general advice is to use a Middle region that covers all the wanted quantiles and use a large value of Nslot, that gives the desired accuracy.

Rules for the calculation of Quantiles: There are several ways to calculate the quantiles in detail, but no generally accepted rule. In (1), (5), (9) the nine most important rules are discussed. Let us call them R1 to R9. See (9) for a comparison, using the data values in Table 1B.

When the number of values (Totpop) is larger than some thousand values the differences between these rules are small (9), for all ordinary quantiles. For extreme quantiles the different rules do produce different values. There is no clear answer as to which rule to choose in different situations. For large values of Totpop the rules are reduced into the R1- and R2-rules.

Let us define the quantile rules QOS_R1 and QOS_R2, for the OS-method. We have sorted the *Xi*-values to get the set of data values *X(1) <= .. <= X(i) <= X(i+1) <= ..<= X(Totpop).* We want to find the largest Xq, the x-value for the quantile q, such that the total number of values up to and including Xq, is less than or equal to `Q = q*Totpop`. Xq need not be one of the data values.

Let *X(ii)* be the largest *Xi*-value from the EDS, such that the count of the number of values up to and including this value, is less than or equal to Q. Q need not be an integer. The next higher data value from the EDS is *X(ii+1)*.

Then *X(ii) <= Xq < X(ii+1)*. We denote the left value *X(ii*) by QOS_R1. We denote the average of the two values X(ii) and X(ii+1), by QOS_R2. These correspond to the R1- and R2-rules in (9).

The quantile definitions in the OS- and the Slot method are very similar but slightly different. Note that the rules described in (1), (5), (9) are not quite directly applicable in the Slot method, since they refer to discrete data points.

The *Xi*-data values in the EDS are unsorted and random. Using the Slot method, we create an ordered set of slots, where each slot contains the number of original *Xi*-values that fall between the slot boundaries. We want to define rules for calculating the x-values for the quantiles that correspond to the R1- and R2-rules. Below we have designed algorithms that resemble the traditional sorting and counting but use approximate and clustered values.

MP or Mid-Point rule – We want to identify the x-value, where the accumulated number of data values equals Q. We start with the number of values that belong to the Low region. At each slot boundary we know the exact number of *Xi*-values.

Let Sacc(J) be the accumulated number of *Xi*-values, up to and including slot J. Find the first slot JJ where `Sacc(JJ) >= Q`. This slot is said to correspond to the quantile q.

To estimate Xq, the x-value for quantile q, we use the midpoint of slot JJ and denote it by QSL_MP. `QSL_MP= LLIM + (JJ-0.5)*Slotwidth`.
The absolute error, compared to QOS_R1, is less than one half slotwidth.

Linear interpolation within a slot: Let us now assume that the original *data values Xi are evenly distributed within each slot.* Slot JJ contains SA(JJ) data values. Divide the slot into SA(JJ) subslots. Every subslot is assumed to contain exactly one data value. The midpoint of the k:th subslot of slot JJ is: `Z= Slot left boundary + (k-0.5)*Slotwidth/ SA(JJ)`

where Slotwidth/ SA(JJ) is the width of the local subslot.
We consider Z as an approximation of the k:th *Xi*-value within the slot. The number of *Xi*-values up to and including Z is then "the number of *Xi*-values including the previous slot" + k.

The R1- and R2-rules applied to slots: We know that the quantile q lies in slot JJ. Let SAccO= SAcc (JJ-1), the accumulated number of data values up to and including slot JJ-1. Q-SAccO is the number of additional data values needed to "reach the x-value for the quantile q".

We use the midpoint of subslot number Q-SaccO as an estimate of the x-value for q. It is similar to the QOS_R1-rule. We call it the "*Left rule" (R1)* and denote it by QSL_R1. We define it as `QSL_R1= LLIM + (JJ-1)*Slotwidth + (Q-SAccO – 0.5)*Slotwidth/SA(JJ).`

Table 5 – Overview – Different rules for quantile calculations.

Quantile name	**Description**	**Comment - Plus and Minus**
OS-method		
QOS_R1 – Left rule.	The **left** of the surrounding points X(j) and X(j+1).	Very exact. Requires sorting Expensive.
QOS_R2 – Average rule.	**Average** of the surrounding points X(j) and X(j+1).	Very exact. Requires sorting. Expensive.
Slot method		
QSL_MP – MP Compare with QOS_R1	*Mid-Point in the slot, where the quantile was found.*	*Simple to code and to understand. - Use big Nslot for accuracy.*
QSL_R1 – Left rule. Compare with QOS_R1	Use the subslot for Q.	Good accuracy. - More complex to program
QSL_R2 – Average rule. Compare with QOS_R2	Use the average of the subslots for Q and Q+1.	Good accuracy. - More complex to program

The x-value for Q+1 lies in the next non-empty subslot (in the current slot, or in the next non-empty slot). We use the average of the estimated x-values for Q and Q+1, as an estimate of the x-value for the quantile q. We denote it by QSL_R2 and call it the *"Average Rule (R2)".* This is similar to the QOS_R2 definition. In the Appendix we show the exact code for QSL_R1 and QSL_R2.

In regions where the data values are very sparse, the x-value for Q and the x-value for Q+1 may fall in different slots. Then QSL_R2 may be assigned to an empty slot – just as QOS_R2 may lie far from any data values when the *Xi*-values lie wide apart.

Comparison of the statistics: "Exact" – Slot method – for the basic example – 5 M.

In Table 6 we have calculated the statistics for the basic example EDS, using the OS-method. This is the "Exact" solution for the basic example. The values calculated using the Slot method shall be compared to these values. We have used the rules QSL_R1 and QSL_R2, which produce slightly different results. In Table 7 we used the Slot method, with Slotwidth= 0.002.

When we compare the values in Table 6 and Table 7, we find that the errors in the quantiles are all less than 0.001 (i.e. half the Slotwidth). The major errors are in q0.001 and q99.99.

The x-value of a quantile may lie anywhere in the slot, in the middle or near a boundary.

Table 6: Quantiles, Mean and StdDev - Basic example (5M values) – "Exact OS-method".

Rule	q.001	q.01	q.1	q1	q5	q10	q25	**q50**
QOS_R2	-0.460 101	-0.221 466	0.065 876	0.472 900	0.903 092	1.165 984	1.673 654	**2.367 106**
QOS_R1	-0.460 238	-0.221 780	0.065 848	0.472 898	0.903 092	1.165 984	1.673 656	**2.367 106**
		q75	q90	q95	q99	q99.9	q99.99	q99.999
QOS_R2		3.246 626	4.250 125	4.967 147	6.607 672	8.938 886	11.256 068	13.687 862
QOS_R1		3.246 626	4.250 124	4.967 144	6.607 671	8.938 831	11.256 067	13.678 904
Mean=	2.577	705 382			**StdDev=**	1.283	249 161	

Table 7: Quantiles, Mean and StdDev - Basic example (5M, Slotwidth= 0.002) – Slot method.

Rule	q.001	q.01	q.1	q1	q5	q10	q25	**q50**
QSL_R2	-0.459 667	-0.221 600	0.065 868	0.473 000	0.903 116	1.165 985	1.673 636	**2.367 110**
QSL_R1	-0.460 333	-0.221 700	0.065 855	0.472 998	0.903 116	1.165 985	1.673 636	**2.367 110**
QSL_MP	*-0.461*	*-0.221*	*0.065*	*0.473*	*0.903*	*1.165*	*1.673*	***2.367***
		q75	q90	q95	q99	q99.9	q99.99	q99.999
QSL_R2		3.246 636	4.250 108	4.967 182	6.607 603	8.938 909	11.256 667	13.688 000
QSL_R1		3.246 636	4.250 107	4.967 180	6.607 595	8.938 818	11.256 333	13.679 000
QSL_MP		*3.247*	*4.251*	*4.967*	*6.607*	*8.939*	*11.257*	*13.679*
Mean=	2.577	705 424			**StdDev=**	1.283	249 134	

Error analysis - the MP-rule: The error in QSL_MP is always less than half the slotwidth when we compare to QOS_R1.

Error analysis - the R1- and R2-rules: The errors in QSL_R1 and QSL_R2 are always less than one slotwidth. These rules are more accurate than the MP-rule, when Nslot is small or medium in size. The difference in accuracy is reduced for large Nslot. For very big Nslot, they give the same error as the MP-rule.

Use the Slot Array – MP-, R1-, R2-rules - Conclusions: There is no "correct" rule to calculate the quantiles using the OS-method or the Slot Method. The rules give slightly different results, and the user must decide which rule to use.

The value for the quantile q99.999 differs between the R1- and R2-rules with more than four slotwidths (0.008). The same applies in the Slot method for QSL_R1 and QSL_R2.

* The simple MP-rule gives very accurate results, when the slotwidth is very small.
* The R1- and R2-rules both give more accurate results for modest values of slotwidth, but the errors in the quantiles decrease slowly with decreasing slotwidth. If the count per slot is small, these rules are more sensitive, and the MP-rule is more robust.

Random deviation: The number of *Xi*-values which fall in a certain slot SA(J) is not "smooth" compared to the values in the neighboring slots. The random differences in the original *Xi*-values, mean that the accumulated values SA(j) all have *"random slot differences*", which will influence the results and the usefulness of the quantiles rules (MP, R1 and R2). The size depends on the slotwidth (Nslot) and the size of the set of data (Totpop).

The *local_stepsize*, used in QSL_R1 and QSL_R2, is the slotwidth divided by number of values in that slot, `Slotwidth/SA(j)`. When the Slotwidth is halved, SA(j) will roughly be halved and the *local_stepsize* and the quantile estimate may remain the same.

In Fig. 1B and Fig. 1C we see that the count (of *Xi*-values) in each slot differs between slots. We want to estimate the average difference between the cdf contribution from the Slot method and that from the theoretical distribution.

First, we define an interval from a to b on the x-axis, where we want to compute the average difference. Let NN denote the number of slots in this interval. See Fig. 4A.

Fig. 4A: Slot method applied to Gumbel random values - detail of Fig. 1C, at x= 2.
Normalized count per slot (blue) and Gumbel cdf slotted (red)
The vertical line = AveDev (Average Deviation) - per 1000 slots

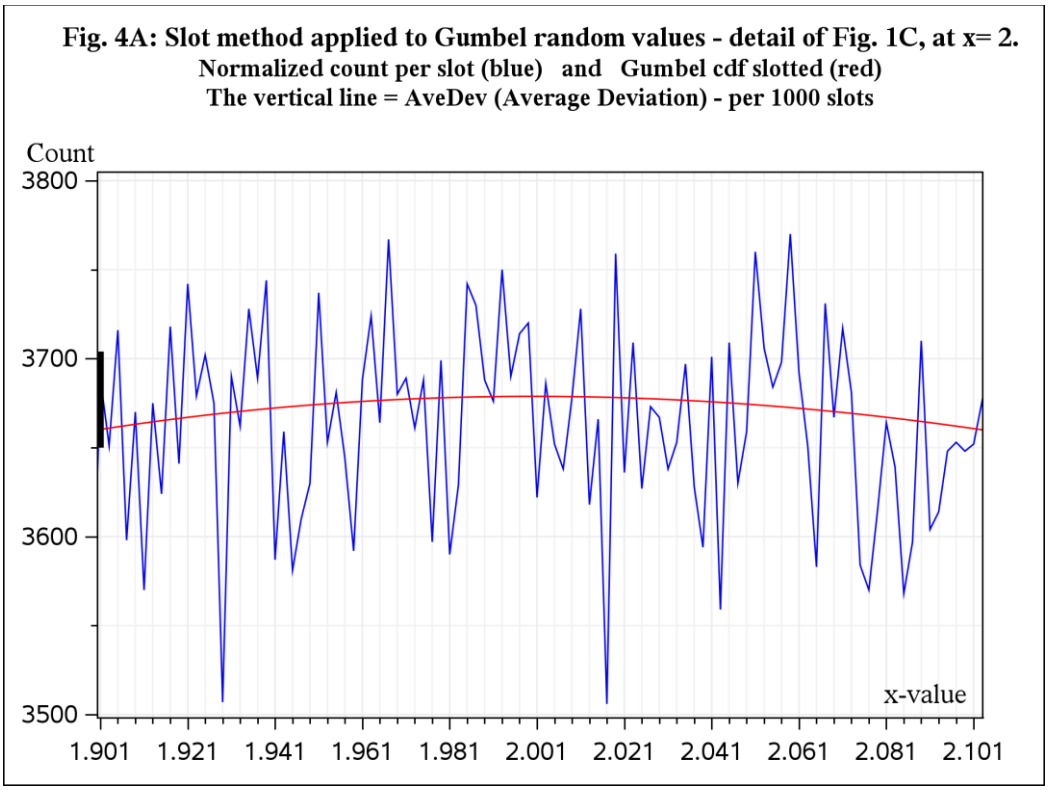

Fig. 4A: A detail of Fig. 1C, around x=2. The vertical line shows the size of *AveDev* (1000) = 54. The x-axis interval is 100 slotwidths wide.

`SA(J)/Totpop` is the normalized count in slot j. Compare this value to the difference in the theoretical Gumbel CDF, between the values in the upper and lower boundary of the slot.
We define the deviation in slot j as: **`Dev=`** `SA(j)/Totpop - (cdf(upper)-cdf(lower))`
We define the average deviation as: ***`AveDev(NN)=`*** `SQRT(Sum_over_NN_slots of Dev**2/ NN)`
This is a measure of the average deviation of SA(j) from the theoretical values, in each slot, measured over NN slots. *AveDev* depends on the placement on the x-axis, Nslot and Totpop.

AveDev (1000) = 10.8 E-6 is the normalized average deviation in each slot, in our basic example. We multiply with Totpop (5M) to get the average count deviation 54 in Fig. 4A, Fig. 4B.

The blue curve in Fig. 4A shows SA(j)/Totpop, for different x-values. The vertical line in Fig. 4A shows *AveDev*, for NN= 1000 slots. This is the average of the values in the blue curve.

We have found that *AveDev* only decreases by a factor of sqrt (2), when Nslot is doubled, for the Gumbel, Normal and Uniform distributions. Thus, the influence of the random effect increases with increasing Nslot. For very large values of Nslot we have only a few, say 0, 1, 2, 3 random *Xi*-values per slot. On the other hand, with small Nslot we do not see much of the random effect.

Linear interpolation (10) works well, when the slot is small, with an error (remainder) proportional to the square of the slotwidth used. Here we use inverse linear interpolation.

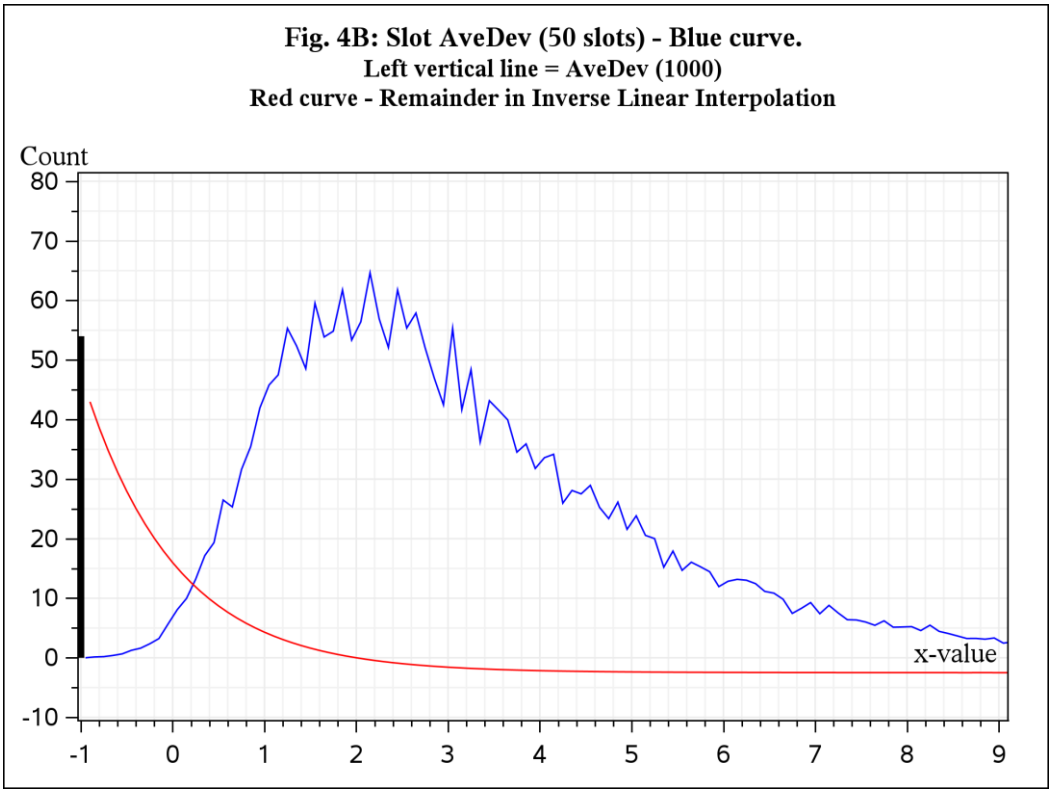

Fig. 4B: Blue *AveDev* (50) versus x. The vertical line shows *AveDev* (1000)= 54. See Fig. 4A. The red curve is the Remainder curve, scaled by 5E+6, versus x.

The red curve in Fig. 4B shows the x-value of the remainder, when computing quantiles using inverse linear interpolation, in the basic example. At x= -1 the Remainder is 2*Slotwidth**2.

The values, scaled by 5E+6, show the exact midpoint value minus the midpoint computed with inverse linear interpolation on exact boundary values, using the exact Gumbel distribution. At q75 (x= 3.25) we have a Remainder of 0.4E-6 and a *local_stepsize* of 1E-6.

In Fig. 4B we show the *AveDev* for 50 slots, for various x-values, for the basic example. The size of *AveDev* shows that there are limitations in the accuracy that can be obtained in QSL_R1 and QSL_R2 by increasing Nslot. This discussion has been included to explain the (lack of) improved accuracy for the QSL_R1- and QSL_R2-rules, when Nslot is increased.

We want to estimate the value for Nslot that makes the random errors in the R1- and R2-rules equal to the error in the MP-rule. The *local_stepsize* `Slotwidth/SAj)` depends on SA(j), which is affected by the randomness in our data.

We want to estimate the random part in `Slotwidth/SA(j)` and use it as the R1- and R2-random error. We replace `SA(j)` with `SA(j)+AveDev`. We use that *AveDev* decreases as sqrt (2) when Nslot is doubled. We also use that `Slotwidth/SA(j)` depends only a little on Nslot.

Finally, we use that the error in the MP-rule is half the slotwidth. By putting this together we arrive at a rough estimate Nslot= 1000k. This is a rough general estimate, which indicates that the R1- and R2- rules give improved accuracy for Nslot less than 1000k.

<u>Using the Slot method</u>: Truncated Mean, Total Income, More variables.

Truncated or Trimmed Mean: Using the Slot Array, we can determine what extreme slot values (outliers) shall be ignored, and compute e.g. Mean(q1-q99) and Mean(q5-q95). See (11).

Calculate the Total Income (Tot Inc) distribution – Let us calculate the quantity `SA(j)` `*(Midpoint Value of slot j),` i.e. (the no. of *Xi*-values in Slot j) * (the middle income in Slot j). This is a good estimate of the total income for all persons, whose income fall in slot j.

The red curve in Fig. 5A shows the total income in MSEK in each slot. The blue curve shows the count of persons in each slot. In Fig. 5B we show the corresponding accumulated and normalized values in the cdf curves.

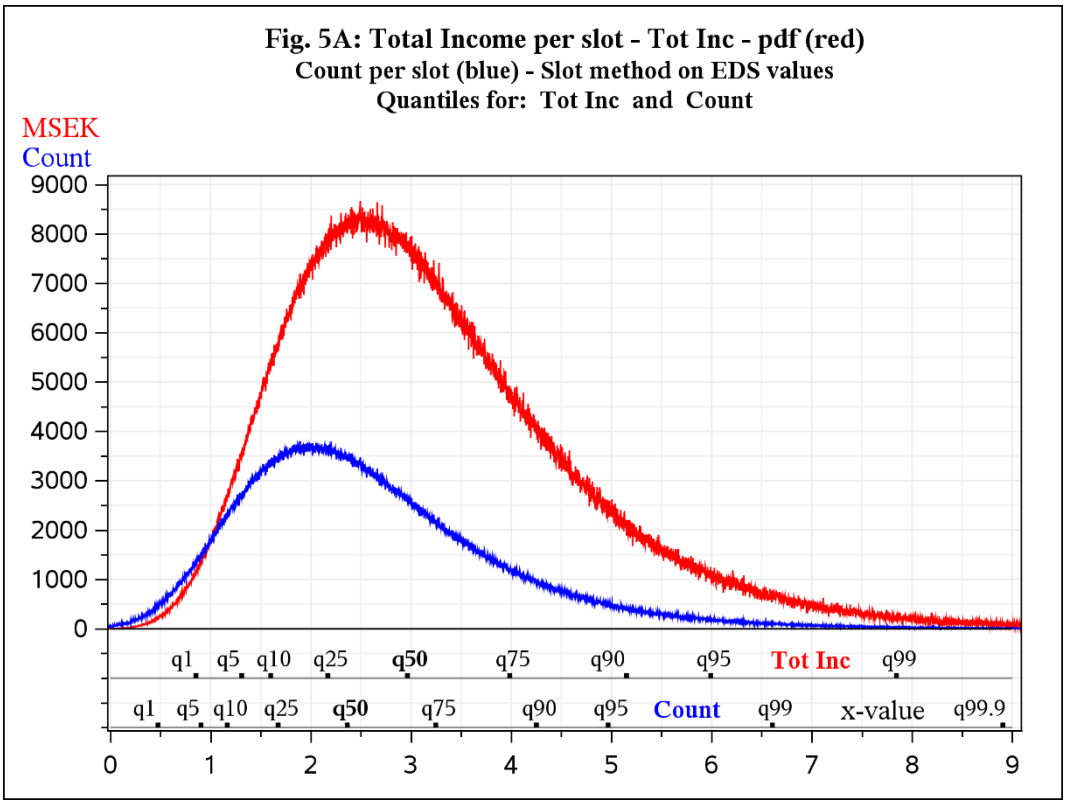

Fig. 5A: The total income per slot, in red. The blue curve shows the count of persons per slot.

We can note in Fig. 5A and Fig. 5B, that the quantiles for the total income are all greater than the quantiles for the count of persons.

The total incomes have been calculated exactly in the Low and High regions, to increase the accuracy. Summation from Slot 1 to Slot k now gives the total income of all employees, up to and including Slot k. Normalization gives the CDF for the total income of all employees.

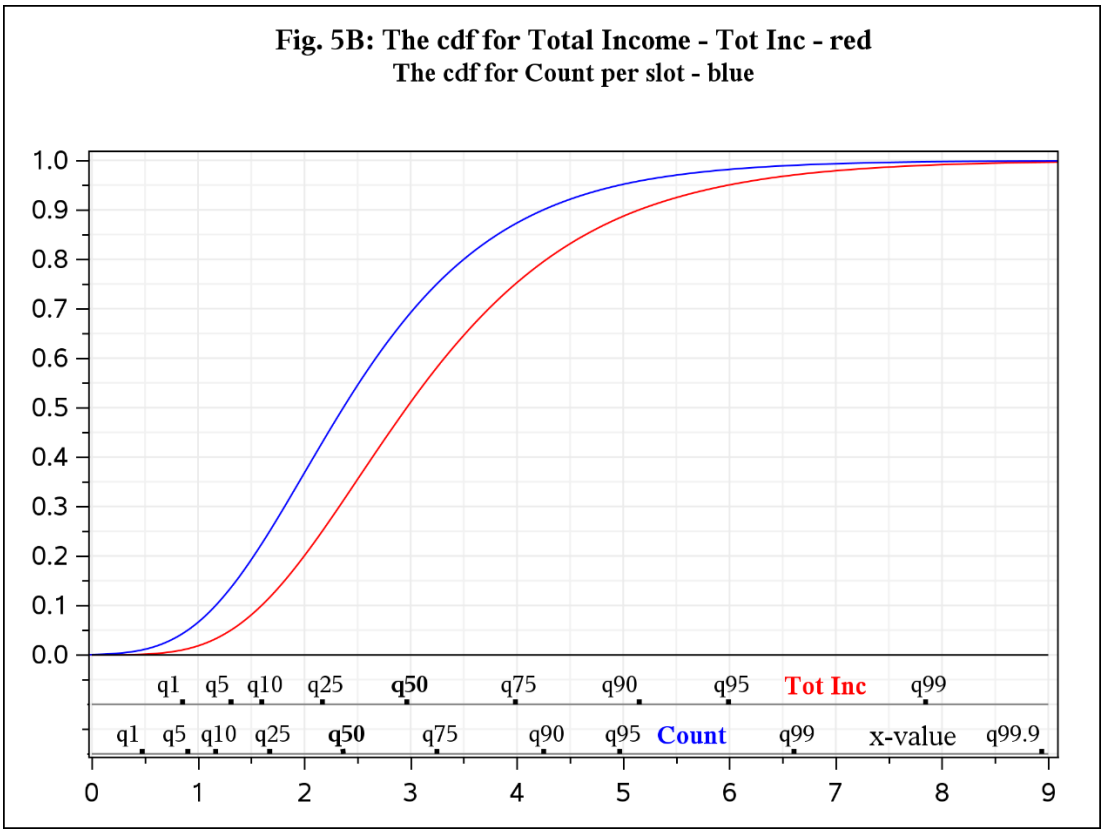

Fig. 5B: The cdf for the total income per slot, is shown in red.
The blue curve shows the cdf for the count of persons per slot.

More variables - Further analysis with more variables from the EDS - Several quantities can be calculated in one pass through the data. We can, for example, read Income and Gender from the EDS and use one Slot Array per gender (Male, Female).

In the postprocessing we can: Calculate the total Slot Array for All genders. Calculate the total income in each slot, in the arrays for Male, Female, All. One pass through the EDS, gives us the information for six pdf/cdf and six sets of quantiles: *Income per person: Male, Female, All* and *Total income: Male, Female, All.*

The persons providing the data for the EDS may provide additional information like "This specific data value represents 55 persons" and "Another group of values shall be corrected (for overtime)". In (2) there are two useful options for this type of problem, "Freq" and "Weight". "Freq" means that an *Xi*-value shall be considered several times and always included in Totpop. "Weight" means that some *Xi*-values have a weight different than 1.0 but shall only be included once in Totpop. We can implement this concept in the Slot method, in the way we want.

<u>Some aspects of the Slot method</u>: Combining Slot Arrays - Comparison to other methods - Implementation - User aspects - Cpu-time

<u>Combine Slot Arrays</u>: It is easy to combine or "merge" different Slot Arrays, originating from different times and/or different sources. The only requirement is that the Slot boundaries are the same or compatible.

Several methods are discussed in (12), (13) and (14), related to "extreme values", "extreme quantiles", "streaming data" and "dynamic data". The Slot method can perhaps be modified and used on these problems. We have not studied these methods.

If we want to analyze very extreme data values and/or very extreme quantiles, we may need to use an interval that is several hundred units wide. We can split this region into several regions and for each region define a unique linear transformation, and a unique value of Nslot, which are suitable for this region. The upper limit for one region must equal the lower limit of the next region. This makes it possible to calculate also very extreme data values and very extreme quantiles. In this paper we only discuss slots of equal length. The slot method should work well also with slots of unequal length. We have not studied extreme data or extreme quantiles.

Comparison to other methods: Above we have made several comparisons with the OS-method in Fig. 2A, 2B and 2C. In (15) there is an interesting discussion about different methods for calculating quantiles. Many of them have been implemented on a computer, but the program code is not available to us. Since the OS-method is straight forward to implement, we have used it above in our comparisons.

*Binary search – for intervals***:** An early version of the Slot method was published in (16). It was very SAS specific, using binary search, in SAS user-defined formats. These take a lot of cpu-time for big sets of data. The present method is more general, faster and does not use binary search.

Bin and Binning: In (17) and (18) bin and binning are discussed. In bins a lot of similar data values are not stored exactly, but rather assembled and reduced into bins. The number of bins to use, should be twice the third root of the number of data values (Rice rule, see Remark in (4)). In our general example this would mean 342 bins, which is far too small for accurate calculations of quantiles. Bins are usually designed to contain approximately the same number of data values.

Bins and the Slot method are similar but different. The slots discussed in this paper can be regarded as a kind of very, very small bins, each storing only the number of original values that fall into that slot. This number varies from very small or zero, up to several thousands. Slots and bins and can be used together and are useful in quite different ways.

Quantile interval for the estimated quantile: We start with a quantile q, $0 < q < 1$. E.g., q50, q75 or q95. Then we compute the x-value for this quantile and estimate the error in this x-value.

We can also estimate the error in the q-value corresponding to the computed x-value. We know the slot for the quantile. The accumulated and normalized sum of the Slot Array for the lower and upper slot boundary are the lower and upper quantile values for this slot. The width of this quantile interval is the normalized count of the slot.

For the basic example we arrive at 3500/5E6 = 0.7E-3 for q50, which gives the quantile interval 49.96 to 50.03. The quantile interval for q95 is smaller, it is 94.99 to 95.00.

Implementation: The cpu-time and memory used depend on the computer and on the implementation. A compiled version of the Slot method would be faster. Parallel processing and/or many threads will speed up the computations. The Slot method is well suited for this, since different "computer processes" can work independently, using compatible Slot Arrays, when reading different parts of the EDS. We have not studied this.

User aspects:

* Cpu-time and memory are always important when discussing computational methods. Other important aspects are: How fail-safe is the method, that is, will the method always produce adequate results (perhaps with warnings when needed), or can the results look nice but be bad.
* Are the accuracy estimates strict or are the results "accurate on a 95% confidence level." If the user has only one set of data, then he wants results for that set of data. A method that, on the average, will give very good results "for many sets of data", is perhaps not good enough.

Cpu-time: We have run several tests and made many comparisons. The conclusion is that the Slot method, with a certain wanted accuracy, only requires a little cpu-time. The measured cpu-times are not exact.

We note that the cpu-time for the OS-method in Fig. 2A increases like C1*Totpop. The cpu-time for a theoretical sort increases like C2*Totpop*log (Totpop), where C1 and C2 are constants. The system sort on (8) uses "parallel servers", to make it more efficient. The cpu-time for the Slot method also increases like C3 * Totpop.

According to Table 4 above, the Slot method calculates the mean, stddev and 15 quantiles, with a reasonable accuracy, using a total cpu-time that is roughly twice the cpu-time just to read the entire set of data.

SUMMARY:

Short description: We have in the current paper designed a method, which we call the Slot method, for calculating descriptive statistics, like the mean, stddev and 15 quantiles of a static set of data. The method makes it easy to calculate these statistics also for large sets of data. Fast internal arrays are heavily used. The search for the correct slot is performed using linear transformation of the data values. Slots of equal size are used.

We have applied this method to a problem that we have some knowledge of, "Calculation of basic statistics for salaries", and implemented it in a standard computational environment.

The Slot method has shown to be very stable, accurate and fast. We have tested the Slot method extensively for Totpop (number of data values) up to 500 M, and Nslot (number of slots) up 5000k (5 M values), without any problems or malfunction at all. The Nslot value of 5000 k means about 7 digits, or log10(2*Nslot), in relative accuracy. With Nslot= 75k we already reach the accuracy level of about 100 SEK in our example of salaries in Sweden.

We have also used the Slot method with Nslot 5 to 10 times larger than Totpop. The method works nicely and without any malfunction in all these cases, but many adjacent slots are empty.

Slot array: 200M → 100k - A large set of data, like 200 M records, can be described in the Slot Array using say 100k values, which enables fast further detailed analysis. The Slot Array is a compact description of the set of data.

We can save the Slot Array in a permanent set of data that will need rather little storage. Other statistics and measures can be added to the Slot method, in the step where the Slot Array is calculated, as well as in later steps. Some examples:

* The user can output selected values to special sets of data, to investigate extreme or odd data.
* We can choose to use additional methods for the quantile calculation.
* Data from several sources, using compatible slot boundaries, can easily be analyzed together.

Implementation and cpu-time: We have used the SAS System as a tool for implementing the Slot method and to calculate descriptive statistics. Many modern computer languages, which support fast handling of internal arrays and storage of data, can be used. The cpu-time used, increases slowly and linearly with the size of the set of data and the number of slots. See Fig. 2A to Fig. 3B.
Suitable Range: We must specify a suitable Slot Range, with lower and upper Slot Range boundaries, to define the Middle region, the data interval we want to investigate. If we have no good knowledge of the set of data, the EDS, we can choose very big the values for Slot Range and Nslot, since the cpu-time only increases slowly when increasing these values. See Fig. 3A, Fig. 3B.

Quantiles - The computed quantiles are not exact, but we can decide the wanted accuracy beforehand, say 3, 4 or 5 digits. The accuracy depends on the Slot Range and Nslot. See Table 3 above. Four correct digits require 7500 slots. We can always verify what quantiles are not calculated correctly.

Post processing of the Slot Array, and the values SALow and SAHigh give the user a lot of opportunities to perform extra analysis of the data and for estimating what quantiles have not been accurately computed. We have also computed *"slightly extreme"* quantiles, like q.001, q.01, q.1 and q99.9, q99.99 and q99.999, which worked well.

Generally the Left R1- and Average R2-rules give better accuracy than the MP-rule. When using larger and larger values for Nslot (with smaller and smaller slotwidths) the influence of the random fluctuations in the number of data values in each slot, is increased. The effect is that the accuracy of the Left R1- and Average R2-rules, does not increase when Nslot goes from medium to big value. The MP-rule is always a good and stable choice.

Disclaimer: The Slot method has not been designed and not tested for really big sets of data, for very extreme quantiles or for very extreme data values. The computer (8) limits the size of the set of data. The method has been implemented and tested on a standard non-parallel computer.

History and Future - The Slot method was not feasible 40 years ago, since Nslot= 7.5k (7500 slots) use about 75 kbyte of memory in internal arrays. In the future the memories will be faster and bigger. This will allow bigger Nslot values. The Slot method will work very well with the use of "parallel computers". This will lower the cpu-times.

Conclusions: The Slot method is a reliable and fast method for computation of any desired set of quantiles for large sets of data. The Slot Array makes it possible to apply other computational methods in the Post processing.

We do not see any opportunity to design other methods for calculating the quantiles, for a specified accuracy, that are significantly faster than the Slot method. The reason is simply that any method must read the data, perform some rearrangement and calculations, and finally write the results.

The cpu-time required for calculating the mean, the stddev and 15 quantiles with more than 4 digits accuracy, for 2M, 20M or 200M data values, are only twice the cpu-time for reading through the data values once. See Table 4 above.

About the Author: Anders Sköllermo, Ph.D. in "Applied Numerical Analysis", Actuary, IT- and SAS Specialist. email: anders.skollermo@one.se Stockholm.

Acknowledgement:

* Rolf Sundberg, professor emeritus of Mathematical Statistics at Stockholm University, encouraged me to explain the previous version of the Slot method in a better way.
* With all my heart I want to thank my wife Ph.D. Gunilla Sköllermo for all her support in this work and for many valuable discussions.
*SAS Institute and SAS Education who allowed me to use "SAS on Demand for Academics".

References:

(1) Quantile - https://en.wikipedia.org/wiki/Quantile
(2) The SAS System - www.sas.com
(3) SPSS: www.ibm.com/products/spss-statistics
(4) Histogram - https://en.wikipedia.org/wiki/Histogram
(5) Rick Wicklin, sample quantiles: A comparison of 9 definitions - https://blogs.sas.com/content/iml/2017/05/24/definitions-sample-quantiles.html
(6) Gumbel distribution – https://en.wikipedia.org/wiki/Gumbel_distribution
(7) https://en.wikipedia.org/wiki/Floor_and_ceiling_functions
(8) "SAS on demand for Academics" on a Linux platform.
(9) Rob J Hyndman and Yanan Fan - sample quantiles in statistical packages - https://www.amherst.edu/media/view/129116/original/sample+Quantiles.pdf
Rob J Hyndman and Yanan Fan - Errata: https://robjhyndman.com/publications/quantiles/
Rob J Hyndman - "sample quantiles 20 years later"
(10) Linear Interpolation - https://en.wikipedia.org/wiki/Linear_interpolation
(11) Truncated Mean - https://en.wikipedia.org/wiki/Truncated_mean
(12) C. Masson DDSketch: "A Fast and Fully-Mergeable Quantile Sketch with Relative-Error Guarantees" https://arxiv.org/pdf/1908.10693.pdf
(13) G. Carmode "Small Summaries for Big Data", Cambridge University Press 2020, http://dimacs.rutgers.edu/~graham/sBAd.html
(14) T.Dunning, O. Ertl "Computing Extremely Accurate Quantiles Using t-Digests" https://arxiv.org/abs/1902.04023
(15) Z. Chien et.al. "A Survey of Approximate Quantile Computation on Large-scale Data (Technical Report)" https://arxiv.org/pdf/2004.08255.pdf.
(16) A. Sköllermo "A Fast and Accurate One pass Method for Computing All Quantiles of Very Large Data Sets" – www.researchgate.net
(17) Bin - https://en.wikipedia.org/wiki/Data_binning
(18) SAS - https://blogs.sas.com/content/iml/2019/08/07/essential-guide-binning-sas.html
(19) Arrays in the SAS System: Search for SAS and "Array Processing" and "Array Statement".

Appendix: **Implementing the Slot method:** The exact details of the performance of the Slot method depend on the programming environment. There are many suitable candidates to choose from, but a deep knowledge of the computer language is needed.

We have implemented the Slot method, using the programming language and the data storage facilities in the SAS System. This is an efficient "Table handling software system", with fast read/write of data between the Data step and data stored in SAS internal format. In the Data step calculations are made on the current row or observation (obs for short). The Data step is a loop, through all the data. SAS data sets are often called tables for short.

Below "save a set of data" and "save in data set" always means "save in a SAS data set". This is faster than writing the values to an ordinary ASCII-file.

In the central part of the program, we use a table with only one single row, but with several thousand columns/variables. This is an unusual way of using the SAS System. It is very similar to an ordinary old-fashioned program (like Fortran) with internal arrays. The computations are very fast and very stable.

Two SAS procedures, Univariate and Means, use the OS-method, but they are not optimized for calculating descriptive statistics. The user written OS-method (using system sort and own calculation of the descriptive statistics) is much faster. The Slot method is even faster.

The parameters TOTPOP, NSLOT, LLIM, ULIM etc. are given values using macro variables, with the same name, before the start of the program. This is only a simple text editing feature.

Two ways to decrease the cpu-time:

First: The use of temporary arrays, like `Array SA(7500) 8 _Temporary_ (7500*0.);` decreases the cpu-time drastically. See (19). Temporary arrays are like vectors in other computer languages and exist only in the current row (obs) in the data step where they are used.

The values of elements SA(1), SA(2), , need to be transferred to ordinary variables, which are then saved in data sets. The maximum number of array elements depends on the memory of the computer. On (8) at least 250 M 8 bytes values can be used (2 Giga bytes).

Second: We use only one single Data step: The EDS data set is read. The computations of the wanted statistics (mean, stddev and quantiles) are made. The Slot Array is saved to a data set. The parameter information is saved in another data set.

The SAS Systems uses floating point numbers for integers. The SAS system uses 8 bytes as the default length for storage and calculations. If we change to 4 bytes above, the cpu-time is not reduced, and the maximum number of array elements is not increased.

Create Slot Array - using the SAS System: Our basic example. We use a Data-step with an array, the "Slot Array", with e.g., 7.5 k (i.e. 7500) slots. An *Xi*-value is fetched from the EDS, which has 5 M obs. A linear transformation translates the *Xi*-value into an integer j, which is the slot number. We will handle j values that are negative or zero, or greater than Nslot, separately. A counter SA(j) for slot j in the Slot Array SA is increased by one. The current *Xi*-value is not saved and is not used anymore. A new *Xi*-value is then fetched, and the process is repeated.

The central part of the SAS program is shown below. All the SA values now lie in the array. A data set work.SA with 7500 obs. is created.

```
Data work.SA;
   Array SA(&NSLOT) 8 _temporary_ (&NSLOT*0.);  /* Initialize Slot Array SA to zero.    */
   SALow= 0; SAHigh= 0;                         /* Initialize to zero.                  */
   Inv_Slotwidth = &NSLOT/(&ULIM-&LLIM);
   READ_EDS: Set work.EDS (keep= Xi)  end= last;
   slot = 1 + FLOOR((Xi-&LLIM)*Inv_Slotwidth);
   if slot LT 1 then do;
      SALow= SALow+1;               /* Include code for Low sum. */
      If not last then GOTO READ_EDS;
   end;
   if slot GT &NSLOT then do;
      SAHigh= SAHigh+1;             /* Include code for High sum. */
      If not last then GOTO READ_EDS;
   end;
   SA(slot) = SA(slot) + 1;
   If not last then GOTO READ_EDS;
       /* Include code below to - Save the values in SA to a SAS data set. */
Run;
```

Use the Slot Array - Calculate the descriptive statistics - in the SAS Data Step: When all the data values have been read, we use the Slot Array to calculate the statistics. The cpu-time is very small and only depends a little on Nslot. See Fig. 3A and 3B. The fastest way to calculate the statistics, is to include the calculations above, at the place `/* Include code below.. */`.

The SA values are summarized and normalized into a variable *SumSAN* (Summed SA and Normalized). This is the CDF, cumulative distribution function. We add the number of data values SALow that lie below LLIM and the number of data values SAHigh that lie above and including ULIM, to get correct values for the CDF. To speed up the calculations and to save space, we do not explicitly calculate the CDF, instead we sum the values "on the fly".

The original big data set has been transformed into a rather compact, and slightly approximate data set SA, with Nslot data rows (e.g. 7500). To create good estimates of the mean and the stddev we calculate partial sums of the *Xi*-values in the Low and High regions.

Initial - Create the EDS – Create the same of random values, by using `CALL Streaminit(1223456)` - when repeated, it will give the same stream of the Gumbel distribution. In the library name work, data can be saved. We can also use `Xi= &MY - &SCALE*LOG(-LOG(RAND('UNIFORM')));`

```
%LET DISTR= GUMBEL;   %LET MY= 2;   %LET SCALE=  1;  %LET TOTPOP= 500000;
Data work.EDS (keep= Xi );  /* Create EDS below. */
   CALL STREAMINIT(123456);
   do i= 1 to &TOTPOP;
      Xi= RAND("&DISTR", &MY, &SCALE);
      output;
   end;
run;
```

1 - Test1: **The OS-method - the "Exact" solution: System sort and user written code.**
The SAS-table `work.EDS_OS` with the columns `CV, QOS_R1, QOS_R2, Nfound_R2` is created. In rows 16-19 the values for Mean, StdDev and Totpop are displayed.

The quantiles wanted must be specified in increasing order. We need to find the biggest *Xi*-value less than or equal to the quantile. We call this QOS_R1.

Next, we determine the smallest *Xi*-value larger than the quantile. The R2-rule is the average of these two *Xi*-values, and we call it QOS_R2. The variable Nfound_R2 is the row number of the bigger *Xi* in the calculated quantile. The program has not been optimized completely.

```
 proc sort data = work.eds  out= work.eds;
   By Xi;
 run;
 data work.EDS_OS (keep= CV  QOS_R1  QOS_R2  Nfound_R2);
    ARRAY QQ(15)  8  _temporary_ (.00001 .0001 .001  .01   .05   .10    .25   .50
                                  .75    .90   .95   .99   .999  .9999  .99999);
    ARRAY QQTOT(15) 8  _temporary_ (15*0);
    Format CV $9.  QOS_R1  QOS_R2  15.10   Nfound_R2 10.;
    RETAIN CV QQ   QQTOT LastXi  QQNext TOTPOP  SumXi  SumXi2;
    LENGTH CV $9.;
    SET work.eds end=last;
    if _N_ EQ 1 then do;   /* Initialization. */
       CV     = " ";
       LastXi = Xi;
       QQnext = 1;
       SumXi  = 0;
       SumXi2 = 0;
      do i= 1 to DIM(QQ); /* DIM(QQ)= Length of the array QQ. */
         QQTOT(i) = QQ(i) * &TOTPOP;
      end;
   end;
   SumXi = SUM(SumXi, Xi);
   SumXi2= SUM(SumXi2,Xi**2);

   do i= QQnext to DIM(QQ);
      if _N_ LE QQTOT(i) then GOTO NextEDS;
      QOS_R1    = LastXi;             /* We found the QQTOT(i)=  QOS_R1 quantile. */
      QOS_R2    = SUM(LastXi, Xi)/2;  /* We found the QQTOT(i) quantile. */
      Nfound_R2 = _N_;
      CV        = PUT(QQ(i), 7.5);
      QQnext    = i+1;
      output work.EDS_OS;
      GOTO NextEDS;
   end;
   NextEDS:  /* Number of values is too small. Save Xi. Get a new values. */
   LastXi = Xi;
   if last then do;
      if QQnext LE DIM(QQ) then do;  /* Not all Quantiles were found. */
         do i= QQnext to DIM(QQ);
            CV      = PUT(QQ(i), 7.5);
            QOS_R1  = .;
            QOS_R2  = .;
            output work.EDS_OS;
         end;
      end;               /* Write out: Mean, StdDev and Totpop. */
      Mean      = SumXi/&TOTPOP;
      StdDev    = SQRT(SumXi2/&TOTPOP - Mean**2);
      Nfound_R2 = .;
      CV        = "Mean = ";
      QOS_R1    = Mean;
      QOS_R2    = Mean;
         output work.EDS_OS;
      CV        = "StdDev = ";
      QOS_R1    = StdDev;
      QOS_R2    = StdDev;
         output work.EDS_OS;
      CV        = "Totpop = ";
      QOS_R1    = .;
      QOS_R2    = .;
         output work.EDS_OS;
      CV        = LEFT(PUT(&TOTPOP,commaX12.));
         output work.EDS_OS;
   end;
Run;
```

2 - Test2 – This is the Slot method: The program has not been optimized completely.

```
LET LLIM= -1;   %LET ULIM= +14;   %LET NSLOT= 15000;   %LET TOTPOP= 5000000;
DATA WORK.QUANT_SLOT (KEEP= QQfound QQp    QQSAi  QQSlot QSL_MP  QSL_R1
                            QSL_R2  QQSANi REGION DiffJJ)
     WORK.SA_SLOT    (KEEP= SAi  SAccON)
     WORK.LOWHIGH    (KEEP= SALow  SAHigh SA_1  SA_NSLOT MEAN STDDEV);
   LENGTH REGION $6  DiffJJ $5  QQWARN $3;
   ARRAY SA(&NSLOT) 8  _temporary_ (&NSLOT*0);
   ARRAY QQ(15)     8  _temporary_ (.00001 .0001 .001  .01   .05   .10    .25    .50
                                    .75    .90   .95   .99   .999  .9999  .99999);
   ARRAY QQTOT(15)  8  _temporary_ (15*0);
   QQMAX     = DIM(QQ);
   Slotwidth = (&ULIM - &LLIM)/&NSLOT;
   INV  = 1/Slotwidth;
   SALow=  0;  SAHigh= 0;
   SAccO=  0;
   SumXi=  0;  SumXi2= 0;             /* Variables for Mean and StdDev. */
               /* Read EDS and populate SA, use linear transformation.  */
   READ_EDS: SET WORK.EDS(keep = Xi) end = last;
   Slot = 1 + FLOOR(( Xi - &LLIM) * INV);
   if Slot LE &NSLOT then do;
      if Slot LT  1 then do;   /* EXACT calculation of SumXi, SumXi2. */
         SALow  = SUM(SALow, 1);
         SumXi  = SUM(SumXi, Xi);
         SumXi2 = SUM(SumXi2, Xi**2);
      end;
      else SA(Slot)= SA(Slot) + 1; /* 1 <= Slot <= NSLOT. */
   end;
   else do;     /* Slot > NSLOT. EXACT calculation of SumXi, SumXi2. */
      SAHigh = SUM(SAHigh, 1);
      SumXi  = SUM(SumXi, Xi);
      SumXi2 = SUM(SumXi2, Xi**2);
   end;
   If not last then goto READ_EDS;
      /*-----------------------------------------------------------------*/
      /* Output SALow, SAHigh  AND  Calculate Mean StdDev below.       */
      /*-----------------------------------------------------------------*/
   SA_1      = SA(1);
   SA_NSLOT  = SA(&NSLOT);
    do j= 1 to &NSLOT;
       SumXi  = SUM(SumXi,  SA(j)*(&LLIM +(j-1)*Slotwidth +Slotwidth/2));
       SumXi2 = SUM(SumXi2, SA(j)*(&LLIM +(j-1)*Slotwidth +Slotwidth/2)**2);
   end;
   Mean      = SumXi/ &TOTPOP;
   StdDev    = SumXi2/ &TOTPOP;
   StdDev    = SQRT(StdDev - Mean**2);
   OUTPUT WORK.LOWHIGH;
      /* Create Slot Array SA&NAME data set below.                    */
   REGION  = "MIDDLE";  /*  MIDDLE */
   SAccO   = SALow;
   SAccON  = SALow/ &TOTPOP;
   Do j = 1 to &NSLOT;
      SlotNo = j;
      LowB   = &LLIM + (j-1)*Slotwidth;
      SAi    = SA(j);
      SANi   = SAi/&TOTPOP;
      Output WORK.SA_SLOT;
         /* SAccO  = Slot Array accumulated, including Previous slot. */
         /* SAccON = Slot Array accumulated, Normalized.              */
      SAccO  = SUM(SAccO, SAi);
      SAccON = SAccO/&TOTPOP;
   End;
```

```
   /*================================================================*/
   /* Calculate the Quantiles – Scale up the q values.               */
   /*================================================================*/
Do i = 1 to QQMAX; QQTOT(i)= QQ(i) * &TOTPOP;  End;
QQfound  = 0;
NextSlot = 1;
SAccO    = SALow;  /* All Xi-values less than LLIM. */
Do i = 1 to QQMAX;
   QQtoti = QQTOT(i);
   NextJJ = 0;   QQWARN = " ";   DiffJJ = " ";
          /* If QQtoti is NOT an integer, give a WArning in REGION. */
   if ABS( QQtoti - INT(QQ(i)*TOTPOP) ) GT 0.001 then QQWARN= "-WA";
   If QQtoti GE SALow then GOTO CHK_MIDDLE; /* <-- QQtot LT Upper   */
      /*-------------------------------------------------------------------------*/
      /* Check LOW REGION - QQTOTi LE SALow means that the quantile depends on values */
      /* smaller than the MIDDLE region, and is difficult to calculate in a good way. */
      /*-------------------------------------------------------------------------*/
   REGION   = '*LO'!!QQWARN;
   QQSlot   = 0;
   QSL_MP   = .;   QSL_R1   = .;   QSL_R2   = .;
   QQfound  = i;
   QQp      = QQ(i);
   QQSAi    = SALow;
   QQSANi   = SALow/&TOTPOP;
   OUTPUT WORK.QUANT_SLOT;
   GOTO NEXTI;  /* Find next Quantile. */

   *=======;  CHK_MIDDLE:          /* Check MIDDLE REGION           */
   If NextSlot GT &NSLOT then GOTO CHK_HIGH;
   NextJ= NextSlot;
   Do j = NextJ to &NSLOT;
           /* Fix – rounding. */
      If QQtoti > SUM(SAccO, SA(j)) + 1E-10 then GOTO NEWSLOT;
         /* QQtoti <= SAcc below and in Slot J. NextSLot= Next Slot to use   */
      NextSlot= j;
      QSL_MP = &LLIM + (j-0.5)*Slotwidth;  /* Quantile in Slot j*/
      QSL_R1 = &LLIM + (j-1)*Slotwidth + (QQtoti - SAccO -0.5)/SA(j)*Slotwidth;
                           /* Fix – rounding. */
      if QQtoti+1 <= SUM(SAccO, SA(j)) + 1E-10  then do;  /* QSL_R2 is also in Slot j. */
         QSL_R2 = &LLIM + (j-1)*Slotwidth + (QQtoti - SAccO)/SA(j)*Slotwidth;
         GOTO OUT_MID;
      end;
                        /* Fix – rounding. */
      if QQtoti+1 > SUM(SAccO, SA(j)) + 1E-10 then do;/* QSL_R2 in Next non-empty right slot. */
         if j = NSLOT then do;
            QSL_R2 = .;    PUT "Quantile= " QQtoti 10.6 "QSL_R2 in HI Region.";
            GOTO NEXTI;
         end;
         do JJNext= j+1 to &NSLOT;   /* J < NSLOT below.  */
            if SA(JJNext) NE 0 then do;
               QSL_Right  = &LLIM + (JJNext-1)*Slotwidth + 0.5/SA(JJNext)*Slotwidth;
               QSL_R2   = (QSL_R1 + QSL_Right)/2;
               DiffJJ   = PUT(JJNext-J, 5.);
               GOTO OUT_MID;
            end;
         end;
         QSL_R2 = .;   PUT "Quantile= " QQtoti 10.6 " QSL_R2 in HI Region.";
         GOTO OUT_MID;
      end;
      *=======; OUT_MID:
      REGION  = 'MID'!!QQWARN;
      QQSlot  = j;
      QQfound = i;
      QQp     = QQ(i);
      QQSAi   = SA(j);
```

```
      QQSANi  = QQSAi/&TOTPOP;
      OUTPUT WORK.QUANT_SLOT;
      GOTO NEXTI;  /* check next quantile, Starting in same slot?   */
      *======; NEWSLOT:        ** prepare test in next Slot **;
      SAccO    = SAccO + SA(j);
      NextSlot = j+1;
      If NextSlot GT &NSLOT then GOTO CHK_HIGH;
   End; ** end do-loop over slots **;

    ** All NSLOT checked **;
   If QQfound LT QQMAX then GOTO CHK_HIGH;
   GOTO ALL_Q_FOUND;

   *=======; CHK_HIGH:       ** check High REGION **;
   If QQfound EQ QQMAX then GOTO All_Q_FOUND;
      /* Quantile is in High Region. No estimate is done. */
     QSL_MP   = . ;    QSL_R1   = . ;    QSL_R2   = . ;
   REGION   = '*HI'!!QQWARN;
   QQSlot   = &NSLOT +1;
   QQfound  = i;
   QQp      = QQ(i);
   QQSAi    = SAHigh;
   QQSANi   = QQSAi/&TOTPOP;
   OUTPUT WORK.QUANT_SLOT;
   *====; NEXTI:
End;  ** end do-loop over quantiles **;
*==========; ALL_Q_FOUND:
If QQfound EQ QQMAX then put "All quantiles found";
Run;
```

3 - Test 3: Slot method, 15 k slots in Slot array – No statistics.

```
%LET LLIM= -1;   %LET ULIM= +14;   %LET NSLOT= 15000;   %LET TOTPOP= 5000000;
DATA  WORK.EDS_SA  (KEEP=  SAi SANi SlotNo LowB);
   ARRAY SA(&NSLOT) 8  _temporary_ (&NSLOT*0);
   Slotwidth = (&ULIM - &LLIM)/&NSLOT;
   INV      = 1/Slotwidth;
   SALow     = 0;
   SAHigh    = 0;
   READ_EDS: SET WORK.EDS(keep = Xi) end = last;
   Slot = 1 + FLOOR((Xi - &LLIM) * INV);
   if Slot LE &NSLOT then do;
      if Slot LT  1 then SALow= SAlow + 1;
      else SA(Slot)= SA(Slot) + 1;  /* 1 <= Slot <= NSLOT. */
   end;
   else SAHigh = SAHigh +1; /* Slot > NLSOT. */
   If not last then goto READ_EDS;
   SlotNo = 0;
   LowB   = &LLim + (j-1)*Slotwidth;
   SAi    = SALow;
   SANi   = SAi/&TOTPOP;
      Output WORK.EDS_SA;
   Do j = 1 to &NSLOT;
      SlotNo = j;
      LowB   = &LLim +(j-1)*Slotwidth;
      SAi    = SA(j);
      SANi   = SAi/&TOTPOP;
         Output WORK.EDS_SA;
   End;
   SlotNo = &NSLOT +1;
   LowB   = &ULIM;
   SAi    = SAHigh;
   SANi   = SAi/&TOTPOP;
      Output WORK.EDS_SA;
run;
```

4 - Test 4: Read - Calculate MS, write two values. – No quantiles.

```
data work.edsmean;
   set work.eds  end= last;
   RETAIN SumXi  SumXi2; /* RETAIN – to keep the value of the variables. */
   SumXi  = SUM(SumXi,  Xi);
   SumXi2 = SUM(SumXi2, Xi**2);
   if last then do;
      Mean  = SumXi/&TOTPOP;
      StdDev = SQRT(SumXi2/&TOTPOP - Mean**2);
      output work.edsmean;
   end;
run;
```

5 - Test 5: Read only – no calculations, no write.**:**

```
Data _NULL_; set work.eds; run;
```

Overview – the data sets (tables) created:

work.LowHigh: Only one row. The boundaries LLIM and ULIM shall be chosen so large, that SALow and SAHigh are close to zero.

SALow = Number of data values below LLIM.

SAHigh = Number of data values on or above ULIM.

MEAN, STDDEV = Computed value for Mean and StdDev (Standard Deviation). We have used exact calculation in the Low and High regions.

We can estimate the smallest and largest quantile to calculate using SA_1 and SA_NSLOT.

work.SA_SLOT: NSLOT rows.

SAi = Count of values in a slot.

SAccON = Normalized sum of all counts up to and including the previous slot. (N means divided by Totpop).

work.Quant_Slot: The major table. One row for each quantile wanted – 15 rows above.

REGION = "LO", "MID" or "HI". The text "-WA" has been added, when current q * TOTPOP is not an integer.

QQfound = Number of the quantile (e.g. the median is number 8 of 15 quantiles).

QQp = Value of the quantile (e.g. 0.25, 0.50 or 0.75).

QQSlot = Number of the slot, where the quantile was found. `0 <= QQSlot <= NSLOT+1.` The value NSLOT+1 means that the quantile lies in the High region, and no values is computed for this quantile.

QQSAi = Number of *Xi*-values (SAi) in the slot, where the quantile was found.

QQSANi = Normalized value (SANi) of QQSAi, (N stands for normalized. Divided by TOTPOP).

DiffJJ = A non-zero value specifies the number of subslots used in the quantile calculation.

QSL_MP = The x-value of the quantile, calculated using the MP-rule.

QSL_R1, QSL_R2 = The quantile values are calculated using the R1- and R2-rules.