Computational Statistics

Lecture 5: Simulating from arbitrary empirical random distributions

Raymond Bisdorff

University of Luxembourg

22 novembre 2019

Content of Lecture 5

1. Single-Pass estimation of arbitrary quantiles
   - Computing sample quantiles
   - Quantiles via selecting algorithms
   - Tracking the M largest in a single pass
2. Computing quantiles from binned data
   - Equally binned observation data
   - Linear interpolation formulas
   - Regular binned data quantiles
3. Incremental quantiles estimation : the IQ-agent
   - The incremental quantiles estimation algorithm
   - Using the `IQ-agent`
   - Monte Carlo simulations with the `IQ-agent`

1. Single-Pass estimation of arbitrary quantiles
   - Computing sample quantiles
   - Quantiles via selecting algorithms
   - Tracking the M largest in a single pass

2. Computing quantiles from binned data
   - Equally binned observation data
   - Linear interpolation formulas
   - Regular binned data quantiles

3. Incremental quantiles estimation : the IQ-agent
   - The incremental quantiles estimation algorithm
   - Using the `IQ-agent`
   - Monte Carlo simulations with the `IQ-agent`

Example of empirical data series

Consider the following empirical data series X gathering service time (in minutes) measured for 51 units of health care actions in a hospital :

$$\begin{aligned} x = [&21, 24, 24, 30, 30, 30, 30, 31, 31, 31,\\ &31, 32, 32, 33, 33, 34, 34, 34, 34, 34,\\ &36, 36, 36, 36, 37, 37, 38, 39, 40, 40,\\ &41, 41, 41, 42, 42, 43, 43, 45, 46, 46,\\ &46, 47, 48, 50, 51, 51, 55, 56, 56, 62,\\ &62] \end{aligned}$$

How to compute, for instance, quintiles :

$$[x_{0\%}, x_{20\%}, x_{40\%}, x_{60\%}, x_{80\%}, x_{100\%}]$$

where $x_{\alpha\%}$ corresponds to the service time such that $\alpha\%$ of the observed data in the series x are lower or equal to this value.

Example of empirical data series

Consider the following empirical data series X gathering service time (in minutes) measured for 51 units of health care actions in a hospital :

$$\begin{aligned} x = [&21, 24, 24, 30, 30, 30, 30, 31, 31, 31,\\ &31, 32, 32, 33, 33, 34, 34, 34, 34, 34,\\ &36, 36, 36, 36, 37, 37, 38, 39, 40, 40,\\ &41, 41, 41, 42, 42, 43, 43, 45, 46, 46,\\ &46, 47, 48, 50, 51, 51, 55, 56, 56, 62,\\ &62] \end{aligned}$$

How to compute, for instance, quintiles :

$$[x_{0\%}, x_{20\%}, x_{40\%}, x_{60\%}, x_{80\%}, x_{100\%}]$$

where $x_{\alpha\%}$ corresponds to the service time such that $\alpha\%$ of the observed data in the series x are lower or equal to this value.

Example computation of sample quintiles

Number of data intervals $n = length(x) - 1 = 50$ and cumulative probabilities $p = [0.0, 0.2, 0.4, 0.6, 0.8, 1.0]$.

$$x_{p[i]} = x_{(n*p[i]+1)} \quad \text{for} \quad i = 1 \ldots 6$$

Hence, $x_{0\%} = x_{(1)}$, $x_{20\%} = x_{(11)}$, $x_{40\%} = x_{(21)}$, $x_{60\%} = x_{(31)}$, $x_{80\%} = x_{(41)}$ and $x_{100\%} = x_{(51)}$

```
x = [ 21, 24, 24, 30, 30,  30, 30, 31, 31, 31,
      31, 32, 32, 33, 33,  34, 34, 34, 34, 34,
      36, 36, 36, 36, 37,  37, 38, 39, 40, 40,
      41, 41, 41, 42, 42,  43, 43, 45, 46, 46,
      46, 47, 48, 50, 51,  51, 55, 56, 56, 62,
      62 ]
```

Example computation of sample quintiles

Number of data intervals $n = length(x) - 1 = 50$ and cumulative probabilities $p = [0.0, 0.2, 0.4, 0.6, 0.8, 1.0]$.

$$x_{p[i]} = x_{(n*p[i]+1)} \quad \text{for} \quad i = 1...6$$

Hence, $x_{0\%} = x_{(1)}$, $x_{20\%} = x_{(11)}$, $x_{40\%} = x_{(21)}$, $x_{60\%} = x_{(31)}$, $x_{80\%} = x_{(41)}$ and $x_{100\%} = x_{(51)}$

```
x = [ 21, 24, 24, 30, 30,  30, 30, 31, 31, 31,
      31, 32, 32, 33, 33,  34, 34, 34, 34, 34,
      36, 36, 36, 36, 37,  37, 38, 39, 40, 40,
      41, 41, 41, 42, 42,  43, 43, 45, 46, 46,
      46, 47, 48, 50, 51,  51, 55, 56, 56, 62,
      62 ]
```

Example computation of sample quintiles

Number of data intervals $n = length(x) - 1 = 50$ and cumulative probabilities $p = [0.0, 0.2, 0.4, 0.6, 0.8, 1.0]$.

$$x_{p[i]} = x_{(n*p[i]+1)} \quad \text{for} \quad i = 1...6$$

Hence, $x_{0\%} = x_{(1)}$, $x_{20\%} = x_{(11)}$, $x_{40\%} = x_{(21)}$, $x_{60\%} = x_{(31)}$, $x_{80\%} = x_{(41)}$ and $x_{100\%} = x_{(51)}$

```
x = [ 21, 24, 24, 30, 30,  30, 30, 31, 31, 31,
      31, 32, 32, 33, 33,  34, 34, 34, 34, 34,
      36, 36, 36, 36, 37,  37, 38, 39, 40, 40,
      41, 41, 41, 42, 42,  43, 43, 45, 46, 46,
      46, 47, 48, 50, 51,  51, 55, 56, 56, 62,
      62 ]
```

Example computation of sample quintiles

Number of data intervals $n = length(x) - 1 = 50$ and cumulative probabilities $p = [0.0, 0.2, 0.4, 0.6, 0.8, 1.0]$.

$$x_{p[i]} \;=\; x_{(n*p[i]+1)} \quad \text{for} \quad i \;=\; 1...6$$

Hence, $x_{0\%} = x_{(1)}$, $x_{20\%} = x_{(11)}$, $x_{40\%} = x_{(21)}$, $x_{60\%} = x_{(31)}$, $x_{80\%} = x_{(41)}$ and $x_{100\%} = x_{(51)}$

```
x = [ 21, 24, 24, 30, 30,  30, 30, 31, 31, 31,
      31, 32, 32, 33, 33,  34, 34, 34, 34, 34,
      36, 36, 36, 36, 37,  37, 38, 39, 40, 40,
      41, 41, 41, 42, 42,  43, 43, 45, 46, 46,
      46, 47, 48, 50, 51,  51, 55, 56, 56, 62,
      62 ]
```

Example computation of sample quintiles

Number of data intervals $n = length(x) - 1 = 50$ and cumulative probabilities $p = [0.0, 0.2, 0.4, 0.6, 0.8, 1.0]$.

$$x_{p[i]} \;=\; x_{(n*p[i]+1)} \quad \text{for} \quad i \;=\; 1 \ldots 6$$

Hence, $x_{0\%} = x_{(1)}$, $x_{20\%} = x_{(11)}$, $x_{40\%} = x_{(21)}$, $x_{60\%} = x_{(31)}$, $x_{80\%} = x_{(41)}$ and $x_{100\%} = x_{(51)}$

```
x = [ 21, 24, 24, 30, 30,  30, 30, 31, 31, 31,
      31, 32, 32, 33, 33,  34, 34, 34, 34, 34,
      36, 36, 36, 36, 37,  37, 38, 39, 40, 40,
      41, 41, 41, 42, 42,  43, 43, 45, 46, 46,
      46, 47, 48, 50, 51,  51, 55, 56, 56, 62,
      62 ]
```

Example computation of sample quintiles

Number of data intervals $n = length(x) - 1 = 50$ and cumulative probabilities $p = [0.0, 0.2, 0.4, 0.6, 0.8, 1.0]$.

$$x_{p[i]} = x_{(n*p[i]+1)} \quad \text{for} \quad i = 1...6$$

Hence, $x_{0\%} = x_{(1)}$, $x_{20\%} = x_{(11)}$, $x_{40\%} = x_{(21)}$, $x_{60\%} = x_{(31)}$, $x_{80\%} = x_{(41)}$ and $x_{100\%} = x_{(51)}$

```
x = [ 21, 24, 24, 30, 30,  30, 30, 31, 31, 31,
      31, 32, 32, 33, 33,  34, 34, 34, 34, 34,
      36, 36, 36, 36, 37,  37, 38, 39, 40, 40,
      41, 41, 41, 42, 42,  43, 43, 45, 46, 46,
      46, 47, 48, 50, 51,  51, 55, 56, 56, 62,
      62 ]
```

Example computation of sample quintiles

Number of data intervals $n = length(x) - 1 = 50$ and cumulative probabilities $p = [0.0, 0.2, 0.4, 0.6, 0.8, 1.0]$.

$$x_{p[i]} = x_{(n*p[i]+1)} \quad \text{for} \quad i = 1 \ldots 6$$

Hence, $x_{0\%} = x_{(1)}$, $x_{20\%} = x_{(11)}$, $x_{40\%} = x_{(21)}$, $x_{60\%} = x_{(31)}$, $x_{80\%} = x_{(41)}$ and $x_{100\%} = x_{(51)}$

```
x = [ 21, 24, 24, 30, 30,  30, 30, 31, 31, 31,
      31, 32, 32, 33, 33,  34, 34, 34, 34, 34,
      36, 36, 36, 36, 37,  37, 38, 39, 40, 40,
      41, 41, 41, 42, 42,  43, 43, 45, 46, 46,
      46, 47, 48, 50, 51,  51, 55, 56, 56, 62,
      62 ]
```

Example computation of sample quintiles

Number of data intervals $n = length(x) - 1 = 50$ and cumulative probabilities $p = [0.0, 0.2, 0.4, 0.6, 0.8, 1.0]$.

$$x_{p[i]} = x_{(n*p[i]+1)} \quad \text{for} \quad i = 1 \ldots 6$$

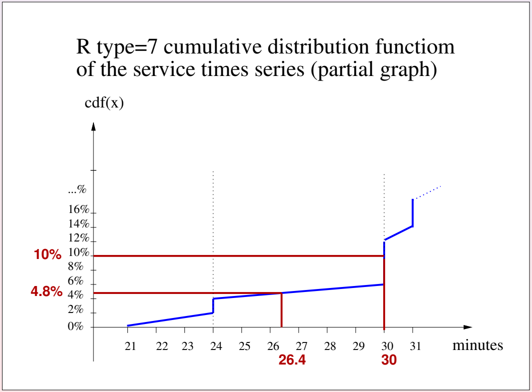
Hence, $x_{0\%} = x_{(1)}$, $x_{20\%} = x_{(11)}$, $x_{40\%} = x_{(21)}$, $x_{60\%} = x_{(31)}$, $x_{80\%} = x_{(41)}$ and $x_{100\%} = x_{(51)}$

```
x = [ 21, 24, 24, 30, 30,  30, 30, 31, 31, 31,
      31, 32, 32, 33, 33,  34, 34, 34, 34, 34,
      36, 36, 36, 36, 37,  37, 38, 39, 40, 40,
      41, 41, 41, 42, 42,  43, 43, 45, 46, 46,
      46, 47, 48, 50, 51,  51, 55, 56, 56, 62,
      62 ]
```

The method above corresponds to the default `quantile()` function (type=7) in R. Mind that there is *no standard definition* for sample quantiles computation ! !

R's default sample quantile function

```
> X = read.csv(
+  'minutes.csv')
> x = X{1:51,]
> quantile(x,
+   seq(0,1,0.2))
0% 20% 40% 60% 80% 100%
21  31  36  41  46  62
> quantile(x,0.048)
4.8% 26.4
> quantile(x,0.1)
10% 30
>
```



Selecting the kth smallest or $N - k$ largest

What is the kth smallest, equivalently the $m = N - k$th largest element out of N preordered (with possible ties) elements $x_{(i)}$ with $i = 1, ..., N$?

Here k may take on values between 1 and N, so $k = 1$ gives the minimum, and $k = N$ the maximum value (R *indexing rule*).

The most common use of selection is in statistical characterization of a set of data by quantiles.

The quartiles $Q_0 = x_{0\%}$, $Q_1 = x_{25\%}$, $Q_2 = x_{50\%}$ (the *median*), $Q_3 = x_{75\%}$, and $Q_4 = x_{100\%}$ are the quantiles used for *summaries* and *boxplots* for instance.

Selecting the kth smallest or $N - k$ largest

What is the kth smallest, equivalently the $m = N - k$th largest element out of N preordered (with possible ties) elements $x_{(i)}$ with $i = 1, ..., N$?

Here k may take on values between 1 and N, so $k = 1$ gives the minimum, and $k = N$ the maximum value (R *indexing rule*).

The most common use of selection is in statistical characterization of a set of data by quantiles.

The quartiles $Q_0 = x_{0\%}$, $Q_1 = x_{25\%}$, $Q_2 = x_{50\%}$ (the *median*), $Q_3 = x_{75\%}$, and $Q_4 = x_{100\%}$ are the quantiles used for *summaries* and *boxplots* for instance.

Selecting the kth smallest or $N - k$ largest

What is the kth smallest, equivalently the $m = N - k$th largest element out of N preordered (with possible ties) elements $x_{(i)}$ with $i = 1, \ldots, N$?

Here k may take on values between 1 and N, so $k = 1$ gives the minimum, and $k = N$ the maximum value (R *indexing rule*).

The most common use of selection is in statistical characterization of a set of data by quantiles.

The quartiles $Q_0 = x_{0\%}$, $Q_1 = x_{25\%}$, $Q_2 = x_{50\%}$ (the *median*), $Q_3 = x_{75\%}$, and $Q_4 = x_{100\%}$ are the quantiles used for *summaries* and *boxplots* for instance.

Selecting the kth smallest or $N - k$ largest

What is the kth smallest, equivalently the $m = N - k$th largest element out of N preordered (with possible ties) elements $x_{(i)}$ with $i = 1, \ldots, N$?

Here k may take on values between 1 and N, so $k = 1$ gives the minimum, and $k = N$ the maximum value (R *indexing rule*).

The most common use of selection is in statistical characterization of a set of data by quantiles.

The quartiles $Q_0 = x_{0\%}$, $Q_1 = x_{25\%}$, $Q_2 = x_{50\%}$ (the *median*), $Q_3 = x_{75\%}$, and $Q_4 = x_{100\%}$ are the quantiles used for *summaries* and *boxplots* for instance.

Selecting via partitioning

- The fastest method for selection, allowing rearrangement, is partitioning, exactly as is done in the Quicksort algorithm.
- Selecting a random element, one marches through the array, forcing smaller elements to the left and larger elements to the right.
- One can ignore one subset, and continue only with the subset containing the desired kth element. Selection therefore does not need a stack of pending operations and its operations count scales as N.
- For a C++/nr3 implementaton see the `sort.h` code.

Selecting via partitioning

- The fastest method for selection, allowing rearrangement, is partitioning, exactly as is done in the Quicksort algorithm.
- Selecting a random element, one marches through the array, forcing smaller elements to the left and larger elements to the right.
- One can ignore one subset, and continue only with the subset containing the desired kth element. Selection therefore does not need a stack of pending operations and its operations count scales as N.
- For a C++/nr3 implementaton see the `sort.h` code.

Selecting via partitioning

- The fastest method for selection, allowing rearrangement, is partitioning, exactly as is done in the Quicksort algorithm.
- Selecting a random element, one marches through the array, forcing smaller elements to the left and larger elements to the right.
- One can ignore one subset, and continue only with the subset containing the desired kth element. Selection therefore does not need a stack of pending operations and its operations count scales as N.
- For a C++/nr3 implementaton see the `sort.h` code.

Selecting via partitioning

- The fastest method for selection, allowing rearrangement, is partitioning, exactly as is done in the Quicksort algorithm.
- Selecting a random element, one marches through the array, forcing smaller elements to the left and larger elements to the right.
- One can ignore one subset, and continue only with the subset containing the desired kth element. Selection therefore does not need a stack of pending operations and its operations count scales as N.
- For a C++/nr3 implementaton see the `sort.h` code.

Tracking the M largest in a single pass

- The previous partitioning approach should not be used for finding the largest or smallest element in an array.
- When one is looking for the M largest elements, where M is modest compared to N, the number of elements of the array, a good approach is to keep a heap of the M largest values.
- This approach is implemented as a `HeapSelect` class with :
  - a constructor where you specify M, the size of the heap;
  - an `add` method allowing to add new incoming data values one by one, and
  - a `report` method for getting the kth largest seen so far $(1 \leqslant k \leqslant M)$.

Tracking the M largest in a single pass

- The previous partitioning approach should not be used for finding the largest or smallest element in an array.
- When one is looking for the M largest elements, where M is modest compared to N, the number of elements of the array, a good approach is to keep a heap of the M largest values.
- This approach is implemented as a `HeapSelect` class with :
  - a constructor where you specify M, the size of the heap,
  - an `add` method allowing to add new incoming data values one by one, and
  - a `report` method for getting the kth largest seen so far $(1 \leqslant k \leqslant M)$.

Tracking the M largest in a single pass

- The previous partitioning approach should not be used for finding the largest or smallest element in an array.
- When one is looking for the M largest elements, where M is modest compared to N, the number of elements of the array, a good approach is to keep a heap of the M largest values.
- This approach is implemented as a `HeapSelect` class with :
  - a constructor where you specify M, the size of the heap,
  - an add method allowing to add new incoming data values one by one, and
  - a `report` method for getting the kth largest seen so far ($1 \leqslant k \leqslant M$).

Tracking the M largest in a single pass

- The previous partitioning approach should not be used for finding the largest or smallest element in an array.
- When one is looking for the M largest elements, where M is modest compared to N, the number of elements of the array, a good approach is to keep a heap of the M largest values.
- This approach is implemented as a `HeapSelect` class with :
  - a constructor where you specify M, the size of the heap,
  - an `add` method allowing to add new incoming data values one by one, and
  - a `report` method for getting the kth largest seen so far ($1 \leqslant k \leqslant M$).

Tracking the M largest in a single pass

- The previous partitioning approach should not be used for finding the largest or smallest element in an array.
- When one is looking for the M largest elements, where M is modest compared to N, the number of elements of the array, a good approach is to keep a heap of the M largest values.
- This approach is implemented as a `HeapSelect` class with :
  - a constructor where you specify M, the size of the heap,
  - an `add` method allowing to add new incoming data values one by one, and
  - a `report` method for getting the kth largest seen so far ($1 \leqslant k \leqslant M$).

Tracking the M largest in a single pass

- The previous partitioning approach should not be used for finding the largest or smallest element in an array.
- When one is looking for the M largest elements, where M is modest compared to N, the number of elements of the array, a good approach is to keep a heap of the M largest values.
- This approach is implemented as a `HeapSelect` class with :
    - a constructor where you specify M, the size of the heap,
    - an `add` method allowing to add new incoming data values one by one, and
    - a `report` method for getting the kth largest seen so far $(1 \leqslant k \leqslant M)$.

Heap select –continue

- The heap has to be sorted when reporting, but all k values may be given without resorting when no new data value is added meanwhile.
- A special case is that getting the $M-1$st largest is always cheap, since it is always at the top of the heap.
- So if you look for a single favorite k, it is best to choose M sucht that $M-1=k$.
- For a C++/nr3 implementaton see the `sort.h` code.

Heap select –continue

- The heap has to be sorted when reporting, but all k values may be given without resorting when no new data value is added meanwhile.
- A special case is that getting the $M - 1$st largest is always cheap, since it is always at the top of the heap.
- So if you look for a single favorite k, it is best to choose M sucht that $M - 1 = k$.
- For a C++/nr3 implementaton see the `sort.h` code.

Heap select –continue

- The heap has to be sorted when reporting, but all k values may be given without resorting when no new data value is added meanwhile.
- A special case is that getting the $M-1$st largest is always cheap, since it is always at the top of the heap.
- So if you look for a single favorite k, it is best to choose M sucht that $M-1=k$.
- For a C++/nr3 implementaton see the `sort.h` code.

Heap select –continue

- The heap has to be sorted when reporting, but all k values may be given without resorting when no new data value is added meanwhile.
- A special case is that getting the $M-1$st largest is always cheap, since it is always at the top of the heap.
- So if you look for a single favorite k, it is best to choose M sucht that $M-1=k$.
- For a C++/nr3 implementaton see the `sort.h` code.

Equally binned empirical data series

Reconstider the pre ordered data series X showing the time in minutes measured for 51 units of health care actions in hospital :

```
[ 21, 24, 24, 30, 30, 30, 30,  31, 31, 31, 31, 32, 32,
  33, 33, 34, 34, 34, 34, 34, 36, 36, 36, 36, 37, 37,
  38, 39, 40, 40,   41, 41, 41, 42, 42,  43, 43, 45, 46,
  46, 46, 47, 48, 50,  51, 51, 55, 56, 56,  62, 62 ]
```

TABLE – Right-closed binning of data series X

bin	low	high	center	freq.	f%	F ↑	F* ↓
1	]20	30]	25	7	13.7%	13.7%	100%
2	]30	40]	35	23	45.1%	58.8%	86.7%
3	]40	50]	45	14	27.5%	86.3%	41.2%
4	]50	60]	55	5	9.8%	96.08%	13.7%
5	]60	70]	65	2	3.9%	100%	3.9%

Working hypothesis !
Observations are uniformly distributed in each bin.
How to compute quantiles from the binned data series ?

Notations

Notation :

- X : a finite data series,
- n : number of bins,
- $a_0 < a_1 < \ldots < a_m$: set of $n+1$ ordered real-valued breaks defined on $\mathbb{R}$. No observation in X may be lower than a_0 or higher than a_n.
- $]a_{i-1}, a_i]$: a partition of $\mathbb{R}$ into n non overlapping upper-closed bins with $i = 1, \ldots, n$,
- $F(x)$: cumulative distribution function (cdf), $x \in \mathbb{R}$,
- $F^*(x)$: complementary cdf : $1.0 - F(x)$,
- $F^{-1}(p)$: inverse cdf where $p \in [0, 1]$,
- x_p : quantile gathering in increasing order $p\%$ of the observation data.

Notations

Notation :

- X : a finite data series,
- n : number of bins,
- $a_0 < a_1 < ... < a_m$: set of $n+1$ ordered real-valued breaks defined on $\mathbb{R}$. No observation in X may be lower than a_0 or higher than a_n.
- $]a_{i-1}, a_i]$: a partition of $\mathbb{R}$ into n non overlapping upper-closed bins with $i = 1, ..., n$,
- $F(x)$: cumulative distribution function (cdf), $x \in \mathbb{R}$,
- $F^*(x)$: complementary cdf : $1.0 - F(x)$,
- $F^{-1}(p)$: inverse cdf where $p \in [0, 1]$,
- x_p : quantile gathering in increasing order $p\%$ of the observation data.

Notations

Notation :

- X : a finite data series,
- n : number of bins,
- $a_0 < a_1 < ... < a_m$: set of $n+1$ ordered real-valued breaks defined on $\mathbb{R}$. No observation in X may be lower than a_0 or higher than a_n.
- $]a_{i-1}, a_i]$: a partition of $\mathbb{R}$ into n non overlapping upper-closed bins with $i = 1, ..., n$,
- $F(x)$: cumulative distribution function (cdf), $x \in \mathbb{R}$,
- $F^*(x)$: complementary cdf : $1.0 - F(x)$,
- $F^{-1}(p)$: inverse cdf where $p \in [0, 1]$,
- x_p : quantile gathering in increasing order $p\%$ of the observation data.

Notations

Notation :

- X : a finite data series,
- n : number of bins,
- $a_0 < a_1 < ... < a_m$: set of $n+1$ ordered real-valued breaks defined on $\mathbb{R}$. No observation in X may be lower than a_0 or higher than a_n.
- $]a_{i-1}, a_i]$: a partition of $\mathbb{R}$ into n non overlapping upper-closed bins with $i = 1, ..., n$,
- $F(x)$: cumulative distribution function (cdf), $x \in \mathbb{R}$,
- $F^*(x)$: complementary cdf : $1.0 - F(x)$,
- $F^{-1}(p)$: inverse cdf where $p \in [0, 1]$,
- x_p : quantile gathering in increasing order $p\%$ of the observation data.

Notations

Notation :

- X : a finite data series,
- n : number of bins,
- $a_0 < a_1 < ... < a_m$: set of $n+1$ ordered real-valued breaks defined on $\mathbb{R}$. No observation in X may be lower than a_0 or higher than a_n.
- $]a_{i-1}, a_i]$: a partition of $\mathbb{R}$ into n non overlapping upper-closed bins with $i = 1, ..., n$,
- $F(x)$: cumulative distribution function (cdf), $x \in \mathbb{R}$,
- $F^*(x)$: complementary cdf : $1.0 - F(x)$,
- $F^{-1}(p)$: inverse cdf where $p \in [0, 1]$,
- x_p : quantile gathering in increasing order $p\%$ of the observation data.

Notations

Notation :

- X : a finite data series,
- n : number of bins,
- $a_0 < a_1 < ... < a_m$: set of $n+1$ ordered real-valued breaks defined on $\mathbb{R}$. No observation in X may be lower than a_0 or higher than a_n.
- $]a_{i-1}, a_i]$: a partition of $\mathbb{R}$ into n non overlapping upper-closed bins with $i = 1, ..., n$,
- $F(x)$: cumulative distribution function (cdf), $x \in \mathbb{R}$,
- $F^*(x)$: complementary cdf : $1.0 - F(x)$,
- $F^{-1}(p)$: inverse cdf where $p \in [0, 1]$,
- x_p : quantile gathering in increasing order $p\%$ of the observation data.

Notations

Notation :

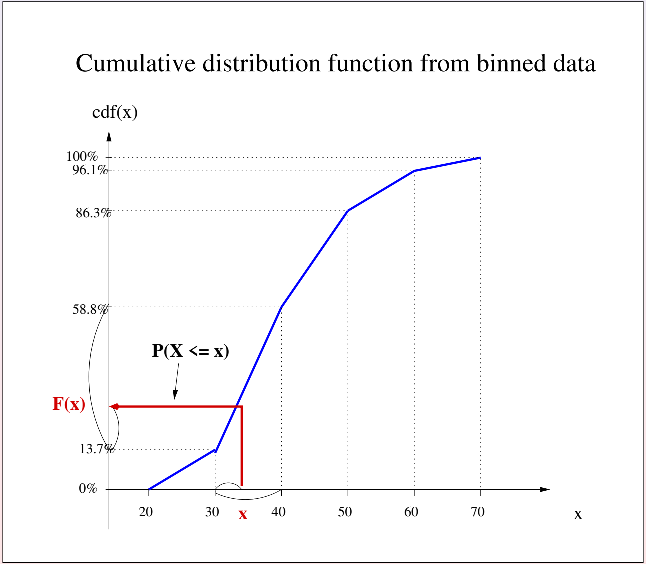
- X : a finite data series,
- n : number of bins,
- $a_0 < a_1 < ... < a_m$: set of $n+1$ ordered real-valued breaks defined on $\mathbb{R}$. No observation in X may be lower than a_0 or higher than a_n.
- $]a_{i-1}, a_i]$: a partition of $\mathbb{R}$ into n non overlapping upper-closed bins with $i = 1, ..., n$,
- $F(x)$: cumulative distribution function (cdf), $x \in \mathbb{R}$,
- $F^*(x)$: complementary cdf : $1.0 - F(x)$,
- $F^{-1}(p)$: inverse cdf where $p \in [0, 1]$,
- x_p : quantile gathering in increasing order $p\%$ of the observation data.

Notations

Notation :

- X : a finite data series,
- n : number of bins,
- $a_0 < a_1 < ... < a_m$: set of $n+1$ ordered real-valued breaks defined on $\mathbb{R}$. No observation in X may be lower than a_0 or higher than a_n.
- $]a_{i-1}, a_i]$: a partition of $\mathbb{R}$ into n non overlapping upper-closed bins with $i = 1, ..., n$,
- $F(x)$: cumulative distribution function (cdf), $x \in \mathbb{R}$,
- $F^*(x)$: complementary cdf : $1.0 - F(x)$,
- $F^{-1}(p)$: inverse cdf where $p \in [0, 1]$,
- x_p : quantile gathering in increasing order p% of the observation data.

Example of binned observation data



Linear cdf interpolation formula

Given x_p, we are looking for $p = F(x_p)$. Now,

$$x_p \in \,]a_{i-1}, a_i] \quad \text{iff} \quad (a_i \geq x_p > a_{i-1})$$

Interpolation principle :

$$\boxed{\frac{F(x_p) - F(a_{i-1})}{F(a_i) - F(a_{i-1})} = \frac{x_p - a_{i-1}}{a_i - a_{i-1}}}$$

Interpolated cumulative distribution function F :

$$p = F(x_p) = F(a_{i-1}) + \frac{x_p - a_{i-1}}{a_i - a_{i-1}} \times (F(a_i) - F(a_{i-1}))$$

Linear quantile interpolation formula

Given $p = F(x_p)$, we are looking for $x_p = F^{-1}(p)$. Again,

$$\begin{aligned} x_p \in \,]a_{i-1}, a_i] \quad & \text{iff} \quad \Big[\big(F(a_i) \geq p\big) \wedge \big(F^*(a_{i-1}) \geq 1-p\big)\Big] \\ & \text{iff} \quad \big(F(a_i) \geq p > F(a_{i-1})\big) \end{aligned}$$

Interpolation principle :

$$\boxed{\frac{x_p - a_{i-1}}{a_i - a_{i-1}} = \frac{F(x_p) - F(a_{i-1})}{F(a_i) - F(a_{i-1})}}$$

Interpolated quantile function F^{-1} :

$$x_p = F^{-1}(p) = a_{i-1} + \frac{p - F(a_{i-1})}{F(a_i) - F(a_{i-1})} \times (a_i - a_{i-1})$$

Interpolating quartiles from binned data

i	$]a_{i-1}$	$a_i]$	center	freq.	f%	F ↑	F* ↓
1	20	30	25	7	13.7%	13.7%	100%
2	30	40	35	23	45.1%	58.8%	86.7%
3	40	50	45	14	27.5%	86.3%	41.2%
4	50	60	55	5	9.8%	96.1%	13.7%
5	60	70	65	2	3.9%	100%	3.9%

$$(58.8\% > 25\% > 13.7\%) \quad \Rightarrow \quad Q_1 = x_{25\%} \quad \in \quad]30;40],$$

$$x_{25\%} \quad = \quad 30 + \frac{25\% - 13.7\%}{58.8\% - 13.7\%} \times (40-30) \; = \; 32.5 \text{ min.}$$

Similarly, $Q_2 = x_{50\%} = 38.0$ min. and $Q_3 = x_{75\%} = 45.9$ min.
By convention, $Q_0 = x_{0\%} = x_{(1)} = 21$ min. and
$Q_4 = x_{100\%} = x_{(51)} = 62$ min.

Interpolating quartiles from binned data

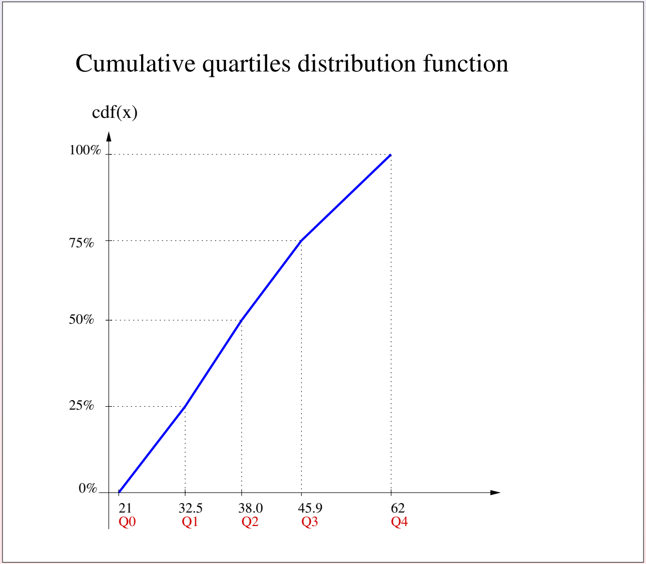
i	$]a_{i-1}$	$a_i]$	center	freq.	f%	F ↑	F* ↓
1	20	30	25	7	13.7%	13.7%	100%
2	30	40	35	23	45.1%	58.8%	86.7%
3	40	50	45	14	27.5%	86.3%	41.2%
4	50	60	55	5	9.8%	96.1%	13.7%
5	60	70	65	2	3.9%	100%	3.9%

$$(58.8\% > 25\% > 13.7\%) \quad \Rightarrow \quad Q_1 = x_{25\%} \quad \in \quad]30;40],$$

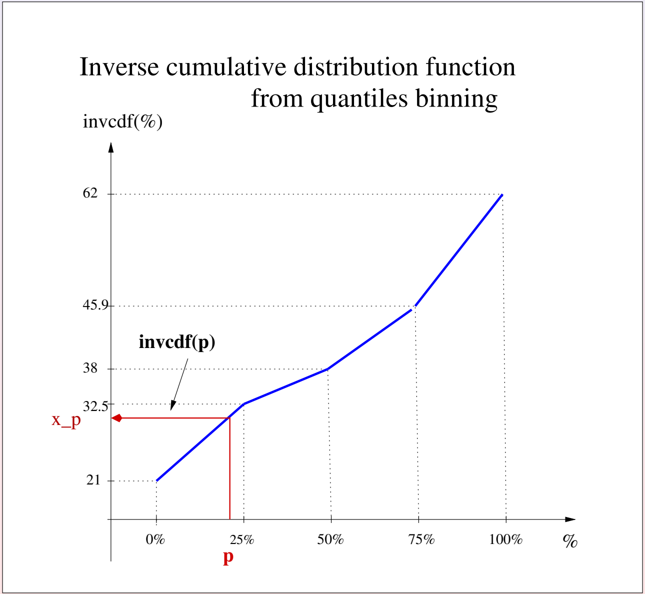
$$x_{25\%} \quad = \quad 30 + \frac{25\% - 13.7\%}{58.8\% - 13.7\%} \times (40 - 30) \; = \; 32.5 \text{ min.}$$

Similarly, $Q_2 = x_{50\%} = 38.0$ min. and $Q_3 = x_{75\%} = 45.9$ min.
By convention, $Q_0 = x_{0\%} = x_{(1)} = 21$ min. and
$Q_4 = x_{100\%} = x_{(51)} = 62$ min.

Regular quartiles binning



Regular quartiles binning



Enhancing exponential tail quantiles' accuracy

Exponential tail quantiles' accuracy may be improved by applying non linear, i.e. a logit interpolation.

1. The logit interpolation of cumulated probability $F(x_p)$ of quantile x_p, with $p \in [0,1]$, situated in bin $]a_{i-1}; a_i]$ with cumulated probabilities $F(a_{i-1})$ and $F(a_1)$ is defined as follows :

$$F(x_p) = g^{-1}\left(g(F(a_{i-1})) + (g(F(a_i)) - g(F(a_{i-1}))) \cdot \frac{x_p - a_{i-1}}{a_i - a_{i-1}}\right)$$

where $g(p) = \log(p/(1-p))$, and $g^{-1}(x) = (1 + \exp(x))^{-1}$.

2. Inversely, quantile x_p in bin $]a_{i-1}; a_i]$ with $F(a_{i-1}) < p < F(a_i)$, is defined as follows :
$x_p = \rho a_{i-1} + (1-\rho) a_i$, where

$$\rho = \frac{g(F(a_i)) - g(p)}{g(F(a_i)) - g(F(a_{i-1}))}$$

Enhancing exponential tail quantiles' accuracy

Exponential tail quantiles' accuracy may be improved by applying non linear, i.e. a logit interpolation.

1. The logit interpolation of cumulated probability $F(x_p)$ of quantile x_p, with $p \in [0,1]$, situated in bin $]a_{i-1}; a_i]$ with cumulated probabilities $F(a_{i-1})$ and $F(a_1)$ is defined as follows :

$$F(x_p) \; = \; g^{-1}\Big(g(F(a_{i-1})) + \big(g(F(a_i)) - g(F(a_{i-1}))\big) \cdot \frac{x_p - a_{i-1}}{a_i - a_{i-1}}\Big)$$

where $g(p) \; = \; \log(p/(1-p))$, and $g^{-1}(x) \; = \; (1 + \exp(x))^{-1}$.

2. Inversely, quantile x_p in bin $]a_{i-1}; a_i]$ with $F(a_{i-1}) < p < F(a_i)$, is defined as follows :
$x_p \; = \; \rho a_{i-1} + (1-\rho) a_i$, where

$$\rho \;\; = \;\; \frac{g(F(a_i)) - g(p)}{g(F(a_i)) - g(F(a_{i-1}))}$$

Enhancing exponential tail quantiles' accuracy

Exponential tail quantiles' accuracy may be improved by applying non linear, i.e. a logit interpolation.

1. The logit interpolation of cumulated probability $F(x_p)$ of quantile x_p, with $p \in [0,1]$, situated in bin $]a_{i-1}; a_i]$ with cumulated probabilities $F(a_{i-1})$ and $F(a_1)$ is defined as follows :

$$F(x_p) = g^{-1}\Big(g(F(a_{i-1})) + (g(F(a_i)) - g(F(a_{i-1}))) \cdot \frac{x_p - a_{i-1}}{a_i - a_{i-1}}\Big)$$

where $g(p) = \log(p/(1-p))$, and $g^{-1}(x) = (1 + \exp(x))^{-1}$.

2. Inversely, quantile x_p in bin $]a_{i-1}; a_i]$ with $F(a_{i-1}) < p < F(a_i)$, is defined as follows :
$x_p = \rho a_{i-1} + (1-\rho) a_i$, where

$$\rho = \frac{g(F(a_i)) - g(p)}{g(F(a_i)) - g(F(a_{i-1}))}$$

1. Single-Pass estimation of arbitrary quantiles
   - Computing sample quantiles
   - Quantiles via selecting algorithms
   - Tracking the M largest in a single pass

2. Computing quantiles from binned data
   - Equally binned observation data
   - Linear interpolation formulas
   - Regular binned data quantiles

3. Incremental quantiles estimation : the IQ-agent
   - The incremental quantiles estimation algorithm
   - Using the `IQ-agent`
   - Monte Carlo simulations with the `IQ-agent`

Single-pass estimation of a quantile

Working conditions :

1. The data values fly by in a stream.
2. You get to look at each value once, and do a constant-time process on it.
3. You only have a fixed amount of storage memory.
4. From time to time arbitrary quantiles of the data values seen so far have to be reported.

Single-pass estimation of a quantile

Working conditions :

1. The data values fly by in a stream.
2. You get to look at each value once, and do a constant-time process on it.
3. You only have a fixed amount of storage memory.
4. From time to time arbitrary quantiles of the data values seen so far have to be reported.

Single-pass estimation of a quantile

Working conditions :

1. The data values fly by in a stream.
2. You get to look at each value once, and do a constant-time process on it.
3. You only have a fixed amount of storage memory.
4. From time to time arbitrary quantiles of the data values seen so far have to be reported.

Single-pass estimation of a quantile

Working conditions :

1. The data values fly by in a stream.
2. You get to look at each value once, and do a constant-time process on it.
3. You only have a fixed amount of storage memory.
4. From time to time arbitrary quantiles of the data values seen so far have to be reported.

Single-pass estimation of a quantile

Working conditions :

1. The data values fly by in a stream.
2. You get to look at each value once, and do a constant-time process on it.
3. You only have a fixed amount of storage memory.
4. From time to time arbitrary quantiles of the data values seen so far have to be reported.

Single-pass estimation of a quantile

Working conditions :

1. The data values fly by in a stream.
2. You get to look at each value once, and do a constant-time process on it.
3. You only have a fixed amount of storage memory.
4. From time to time arbitrary quantiles of the data values seen so far have to be reported.

With conditions stated, only an approximate answer about the exact quantiles of the observed data may be given.

The incremental quantiles estimation algorithm

John Chambers et al. (see moodle resources) have given a robust, and extremely fast, algorithm they call IQ agent, that adaptively adjusts a set of bins so that they converge to the data values of specified quantiles (centiles, quartiles, etc).

The idea is to :

1. accumulate incoming data into batches,
2. update a stored, piecewise linear, cumulative distribution function (cdf) by :
   1. adding a batch's cdf, and
   2. interpolate back to the fixed set of quantiles
3. obtain arbitrary quantiles by linear interpolation from the inverse of the stored cdf.

For a C++/nr3 implementation see the `iqagent.h` code.

The incremental quantiles estimation algorithm

John Chambers et al. (see moodle resources) have given a robust, and extremely fast, algorithm they call IQ agent, that adaptively adjusts a set of bins so that they converge to the data values of specified quantiles (centiles, quartiles, etc).

The idea is to :

1. accumulate incoming data into batches,
2. update a stored, piecewise linear, cumulative distribution function (cdf) by :
   2.1 adding a batch's cdf, and
   2.2 interpolate back to the fixed set of quantiles.
3. obtain arbitrary quantiles by linear interpolation from the inverse of the stored cdf.

For a C++/nr3 implementation see the `iqagent.h` code.

The incremental quantiles estimation algorithm

John Chambers et al. (see moodle resources) have given a robust, and extremely fast, algorithm they call IQ agent, that adaptively adjusts a set of bins so that they converge to the data values of specified quantiles (centiles, quartiles, etc).

The idea is to :

1. accumulate incoming data into batches,
2. update a stored, piecewise linear, cumulative distribution function (cdf) by :
   2.1 adding a batch's cdf, and
   2.2 interpolate back to the fixed set of quantiles.
3. obtain arbitrary quantiles by linear interpolation from the inverse of the stored cdf.

For a C++/nr3 implementation see the `iqagent.h` code.

The incremental quantiles estimation algorithm

John Chambers et al. (see moodle resources) have given a robust, and extremely fast, algorithm they call IQ agent, that adaptively adjusts a set of bins so that they converge to the data values of specified quantiles (centiles, quartiles, etc).

The idea is to :

1. accumulate incoming data into batches,
2. update a stored, piecewise linear, cumulative distribution function (cdf) by :
   2.1 adding a batch's cdf, and
   2.2 interpolate back to the fixed set of quantiles.
3. obtain arbitrary quantiles by linear interpolation from the inverse of the stored cdf.

For a C++/nr3 implementation see the `iqagent.h` code.

The incremental quantiles estimation algorithm

John Chambers et al. (see moodle resources) have given a robust, and extremely fast, algorithm they call IQ agent, that adaptively adjusts a set of bins so that they converge to the data values of specified quantiles (centiles, quartiles, etc).

The idea is to :

1. accumulate incoming data into batches,
2. update a stored, piecewise linear, cumulative distribution function (cdf) by :
   2.1 adding a batch's cdf, and
   2.2 interpolate back to the fixed set of quantiles.
3. obtain arbitrary quantiles by linear interpolation from the inverse of the stored cdf.

For a C++/nr3 implementation see the `iqagent.h` code.

The incremental quantiles estimation algorithm

John Chambers et al. (see moodle resources) have given a robust, and extremely fast, algorithm they call IQ agent, that adaptively adjusts a set of bins so that they converge to the data values of specified quantiles (centiles, quartiles, etc).

The idea is to :

1. accumulate incoming data into batches,
2. update a stored, piecewise linear, cumulative distribution function (cdf) by :
   - 2.1 adding a batch's cdf, and
   - 2.2 interpolate back to the fixed set of quantiles.
3. obtain arbitrary quantiles by linear interpolation from the inverse of the stored cdf.

For a C++/nr3 implementation see the `iqagent.h` code.

The incremental quantiles estimation algorithm

John Chambers et al. (see moodle resources) have given a robust, and extremely fast, algorithm they call IQ agent, that adaptively adjusts a set of bins so that they converge to the data values of specified quantiles (centiles, quartiles, etc).

The idea is to :

1. accumulate incoming data into batches,
2. update a stored, piecewise linear, cumulative distribution function (cdf) by :
   - 2.1 adding a batch's cdf, and
   - 2.2 interpolate back to the fixed set of quantiles.
3. obtain arbitrary quantiles by linear interpolation from the inverse of the stored cdf.

For a C++/nr3 implementation see the `iqagent.h` code.

The incremental quantiles estimation algorithm

John Chambers et al. (see moodle resources) have given a robust, and extremely fast, algorithm they call IQ agent, that adaptively adjusts a set of bins so that they converge to the data values of specified quantiles (centiles, quartiles, etc).

The idea is to :

1. accumulate incoming data into batches,
2. update a stored, piecewise linear, cumulative distribution function (cdf) by :
    2.1 adding a batch's cdf, and
    2.2 interpolate back to the fixed set of quantiles.
3. obtain arbitrary quantiles by linear interpolation from the inverse of the stored cdf.

For a C++/nr3 implementation see the `iqagent.h` code.

Major steps in the IQ algorithm

1. Suppose that T data values have been processed so far.
2. The quantile buffer Q holds the estimated quantiles $[q_{p_1}, q_{p_2}, ..., q_{p_m}]$ for $p\%$ values : $[p_1 = 0.01, p_2 = 0.02, ..., p_m = 0.99]$.
3. Refill the data buffer $D = [d_1, d_2, ..., d_N]$ with N new data values.
4. If D is full or at prespecified times, D is converted into a discrete CDF $F_D(x)$ (a step function, see lecture 4).
5. The quantile buffer Q is converted into a CDF $F_Q(x)$.
6. For all values $x \in Q \cup D$, a weighted average CDF : $T/(T+N) \cdot F_Q(x) + N/(T+N) \cdot F_D(x)$ is computed.
7. Quantiles $[p_1 = 0.01, p_2 = 0.02, ..., p_m = 0.99]$ of the average CDF are used to update Q.

Major steps in the IQ algorithm

1. Suppose that T data values have been processed so far.
2. The quantile buffer Q holds the estimated quantiles $[q_{p_1}, q_{p_2}, ..., q_{p_m}]$ for $p\%$ values : $[p_1 = 0.01, p_2 = 0.02, ..., p_m = 0.99]$.
3. Refill the data buffer $D = [d_1, d_2, ..., d_N]$ with N new data values.
4. If D is full or at prespecified times, D is converted into a discrete CDF $F_D(x)$ (a step function, see lecture 4).
5. The quantile buffer Q is converted into a CDF $F_Q(x)$.
6. For all values $x \in Q \cup D$, a weighted average CDF : $T/(T+N) \cdot F_Q(x) + N/(T+N) \cdot F_D(x)$ is computed.
7. Quantiles $[p_1 = 0.01, p_2 = 0.02, ..., p_m = 0.99]$ of the average CDF are used to update Q.

Major steps in the IQ algorithm

1. Suppose that T data values have been processed so far.
2. The quantile buffer Q holds the estimated quantiles $[q_{p_1}, q_{p_2}, ..., q_{p_m}]$ for $p\%$ values : $[p_1 = 0.01, p_2 = 0.02, ..., p_m = 0.99]$.
3. Refill the data buffer $D = [d_1, d_2, ..., d_N]$ with N new data values.
4. If D is full or at prespecified times, D is converted into a discrete CDF $F_D(x)$ (a step function, see lecture 4).
5. The quantile buffer Q is converted into a CDF $F_Q(x)$.
6. For all values $x \in Q \cup D$, a weighted average CDF : $T/(T+N) \cdot F_Q(x) + N/(T+N) \cdot F_D(x)$ is computed.
7. Quantiles $[p_1 = 0.01, p_2 = 0.02, ..., p_m = 0.99]$ of the average CDF are used to update Q.

Major steps in the IQ algorithm

1. Suppose that T data values have been processed so far.
2. The quantile buffer Q holds the estimated quantiles $[q_{p_1}, q_{p_2}, ..., q_{p_m}]$ for $p\%$ values : $[p_1 = 0.01, p_2 = 0.02, ..., p_m = 0.99]$.
3. Refill the data buffer $D = [d_1, d_2, ..., d_N]$ with N new data values.
4. If D is full or at prespecified times, D is converted into a discrete CDF $F_D(x)$ (a step function, see lecture 4).
5. The quantile buffer Q is converted into a CDF $F_Q(x)$.
6. For all values $x \in Q \cup D$, a weighted average CDF : $T/(T+N) \cdot F_Q(x) + N/(T+N) \cdot F_D(x)$ is computed.
7. Quantiles $[p_1 = 0.01, p_2 = 0.02, ..., p_m = 0.99]$ of the average CDF are used to update Q.

Major steps in the IQ algorithm

1. Suppose that T data values have been processed so far.
2. The quantile buffer Q holds the estimated quantiles $[q_{p_1}, q_{p_2}, ..., q_{p_m}]$ for $p\%$ values : $[p_1 = 0.01, p_2 = 0.02, ..., p_m = 0.99]$.
3. Refill the data buffer $D = [d_1, d_2, ..., d_N]$ with N new data values.
4. If D is full or at prespecified times, D is converted into a discrete CDF $F_D(x)$ (a step function, see lecture 4).
5. The quantile buffer Q is converted into a CDF $F_Q(x)$.
6. For all values $x \in Q \cup D$, a weighted average CDF : $T/(T+N) \cdot F_Q(x) \; + \; N/(T+N) \cdot F_D(x)$ is computed.
7. Quantiles $[p_1 = 0.01, p_2 = 0.02, ..., p_m = 0.99]$ of the average CDF are used to update Q.

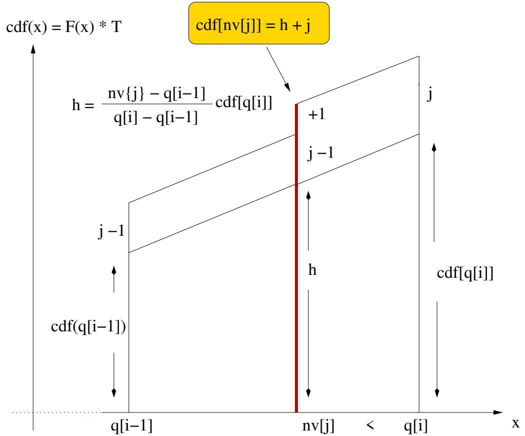
Major steps in the IQ algorithm

1. Suppose that T data values have been processed so far.
2. The quantile buffer Q holds the estimated quantiles $[q_{p_1}, q_{p_2}, \dots, q_{p_m}]$ for $p\%$ values : $[p_1 = 0.01, p_2 = 0.02, \dots, p_m = 0.99]$.
3. Refill the data buffer $D = [d_1, d_2, \dots, d_N]$ with N new data values.
4. If D is full or at prespecified times, D is converted into a discrete CDF $F_D(x)$ (a step function, see lecture 4).
5. The quantile buffer Q is converted into a CDF $F_Q(x)$.
6. For all values $x \in Q \cup D$, a weighted average CDF : $T/(T+N) \cdot F_Q(x) \; + \; N/(T+N) \cdot F_D(x)$ is computed.
7. Quantiles $[p_1 = 0.01, p_2 = 0.02, \dots, p_m = 0.99]$ of the average CDF are used to update Q.

Major steps in the IQ algorithm

1. Suppose that T data values have been processed so far.
2. The quantile buffer Q holds the estimated quantiles $[q_{p_1}, q_{p_2}, ..., q_{p_m}]$ for $p\%$ values : $[p_1 = 0.01, p_2 = 0.02, ..., p_m = 0.99]$.
3. Refill the data buffer $D = [d_1, d_2, ..., d_N]$ with N new data values.
4. If D is full or at prespecified times, D is converted into a discrete CDF $F_D(x)$ (a step function, see lecture 4).
5. The quantile buffer Q is converted into a CDF $F_Q(x)$.
6. For all values $x \in Q \cup D$, a weighted average CDF : $T/(T+N) \cdot F_Q(x) \ + \ N/(T+N) \cdot F_D(x)$ is computed.
7. Quantiles $[p_1 = 0.01, p_2 = 0.02, ..., p_m = 0.99]$ of the average CDF are used to update Q.

Adding the j-th new observation



Merging the jth observed value nv[j] before quantile q[i]

Incremental Python and/or R quantile agent

Exercise

1. *Re-implement the* `IQ-agent` *in Python and in R,*
2. *Design a suitable Monte Carlo simulation experience for verifying the re-implementations.*
3. *Compare the run times of your* `IQ-agent` *in Python and in R with the NR3C++ implementation.*

Empirical CDF agent

Exercise

Notice that the state of the incremental quantile agent represents in fact the empirical cumulated distribution function (cdf) constructed on the fly from an incoming random data stream.

1. *Add* `save` *and* `restore` *methods to the* `iqagent` *(C++/nr3 and R) that allow to save and restore the state of the agent in/from a file.*
2. *Add a* `cdf` *method to the iqagent in C++/nr3 and R rendering the propability $P(X \leqslant q)$ of a given quantile(q).*
3. *Add an inverse cdf method named* `cdfinv` *to the iqagent in C++/nr3 and R rendering the quantile $x_{u\%}$ gathering u% of the observation data.*

Use the `iqagent` for simulation problems

Exercise

Notice that previous incremental cdf agent may readily be used for Monte carlo simulation purposes :

1. *Save an empirical cdf from a sample of 10000 random normal numbers of mean 50 and standard deviation 20*
2. *Compare the previous cdf estimation with the theoretical random variable* $\mathcal{N}(50, 20)$.

Use the `iqagent` for simlation problems – continue

Exercise

The size of the data buffer has a certain influence on the accuracy of the iqagent estimations.

1. *Estimate quantiles with the iqagent from a continuous stream of values generated from a known probability distribution, by varying the size of the data buffer D.*
2. *What is the lowest size for D such that the accuracy stays within the 90% confidence interval of the χ^2 test of difference between the estimated and the real distribution.*