P² quantile estimator: estimating the median without storing values

November 24, 2020
Mathematics Statistics Research Quantiles Performance Telemetry Research: P² quantile estimator

Update: the estimator accuracy could be improved using a bunch of patches.

Imagine that you are implementing performance telemetry in your application. There is an operation that is executed millions of times, and you want to get its "average" duration. It's not a good idea to use the arithmetic mean because the obtained value can be easily spoiled by outliers. It's much better to use the median which is one of the most robust ways to describe the average.

The straightforward median estimation approach requires storing all the values. In our case, it's a bad idea to keep all the values because it will significantly increase the memory footprint. Such telemetry is harmful because it may become a new bottleneck instead of monitoring the actual performance.

Another way to get the median value is to use a sequential quantile estimator (also known as an online quantile estimator or a streaming quantile estimator). This is an algorithm that allows calculating the median value (or any other quantile value) using a fixed amount of memory. Of course, it provides only an approximation of the real median value, but it's usually enough for typical telemetry use cases.

In this post, I will show one of the simplest sequential quantile estimators that is called the P² quantile estimator (or the Piecewise-Parabolic quantile estimator).

The P² quantile estimator

This algorithm was initially suggested in [Jain1985]. Below you can find a short overview of this approach, notes about typos in the original paper, numerical simulation, and a C# implementation.

The main idea

Let's say we have a stream of observations $\{x_0, x_1, x_2, x_3, x_4, \ldots\}$ and we want to estimate p-quantile. The suggested approach introduces five markers that correspond to the estimations of

- q_0: The minimum
- q_1: The (p/2)-quantile
- q_2: The p-quantile
- q_3: The ((1+p)/2)-quantile
- q_4: The maximum

The q_i values are known as the marker heights.

Also, we have to maintain the marker positions $\{n_0, n_1, n_2, n_3, n_4\}$. These integer values describe actual marker indexes across obtained observations at the moment.

Next, we have to define the marker desired positions $\{n'_0, n'_1, n'_2, n'_3, n'_4\}$. For the first n observations, these real values are defined as follows:

- $n'_0 = 0$
- $n'_1 = (n-1)p/2$
- $n'_2 = (n-1)p$
- $n'_3 = (n-1)(1+p)/2$
- $n'_4 = (n-1)$

In order to speed up the algorithm, we can precalculate increments of the desired positions which should be added to the current values after each new observation:

- $dn'_0 = 0$
- $dn'_1 = p/2$
- $dn'_2 = p$
- $dn'_3 = (1+p)/2$
- $dn'_4 = 1$

Note that in the original paper, the authors use one-based indexing. I decided to adapt it to the zero-based indexing which is more convenient from the implementation point of view.

Initialization

Once we collected the first five elements, we should perform initialization logic:

$$\begin{cases} q_0 = x_{(0)}, & n_0 = 0, & n'_0 = 0, & dn'_0 = 0, \\ q_1 = x_{(1)}, & n_1 = 1, & n'_1 = 2p, & dn'_1 = p/2, \\ q_2 = x_{(2)}, & n_2 = 2, & n'_2 = 4p, & dn'_2 = p, \\ q_3 = x_{(3)}, & n_3 = 3, & n'_3 = 2+2p, & dn'_3 = (1+p)/2, \\ q_4 = x_{(4)}, & n_4 = 4, & n'_4 = 4, & dn'_4 = 1. \end{cases}$$

Marker invalidation

For each x_j for $j \geq 5$, we should invalidate our markers.

Firstly, we should adjust extreme marker heights (if $x_j < q_0$, we should update q_0; if $x_j > q_4$, we should update q_4) and find k such that $q_k \leq x_j < q_{k+1}$ (or $q_k \leq x_j \leq q_{k+1}$ for $k = 3$):

Condition	**q_i update**	**k**
$x_j < q_0$	$q_0 = x_j$	0
$q_0 \leq x_j < q_1$		0
$q_1 \leq x_j < q_2$		1
$q_2 \leq x_j < q_3$		2
$q_3 \leq x_j < q_4$		3
$q_4 \leq x_j$	$q_4 = x_j$	3

Secondly, we should update the marker positions and the marker desired positions:

$$\begin{aligned} n_i &= n_i + 1 && \text{for} \quad i = k+1, \dots, 4; \\ n'_i &= n'_i + dn'_i && \text{for} \quad i = 0, \dots, 4. \end{aligned}$$

Finally, we should adjust non-extreme marker heights (q_i) and positions (n_i) for $i \in \{1, 2, 3\}$ in the following way:

```
for (i = 1; i <= 3; i++)
{
    d = n'[i] - n[i]
    if (d >=  1 && n[i + 1] - n[i] >  1 ||
        d <= -1 && n[i - 1] - n[i] < -1)
    {
        d = sign(d)
        q' = Parabolic(i, d)
        if (!(q[i - 1] < q' && q' < q[i + 1]))
            q' = Linear(i, d)
        q[i] = q'
        n[i] += d
    }
}
```

The core equation of the algorithm is a piecewise-parabolic prediction (P²) formula that adjusts marker heights for each observation:

$$q'_i = q_i + \frac{d}{n_{i+1} - n_{i-1}} \cdot \left((n_i - n_{i-1} + d) \frac{q_{i+1} - q_i}{n_{i+1} - n_i} + (n_{i+1} - n_i - d) \frac{q_i - q_{i-1}}{n_i - n_{i-1}} \right).$$

Once we calculated q'_i, we should check that $q_{i-1} < q'_i < q_{i+1}$. If this condition is false, we should ignore the parabolic prediction and use the linear prediction instead:

$$q'_i = q_i + d \frac{q_{i+d} - q_i}{n_{i+d} - n_i}.$$

The result

Once you need the requested quantile estimation value, we should just take the value of q_2.

Typos in the original paper

A find a few typos in the original paper which may confuse readers who want to implement the algorithm from scratch:

- Page 1079, Box 1, B2: $i = k, \ldots, 5$ should be replaced by $i = k+1, \ldots, 5$
- Page 1079, Box 1, B3: $\textbf{THEN}\ q_i \leftarrow q_i$ should be replaced by $\textbf{THEN}\ q_i \leftarrow q'_i$

Numerical simulation

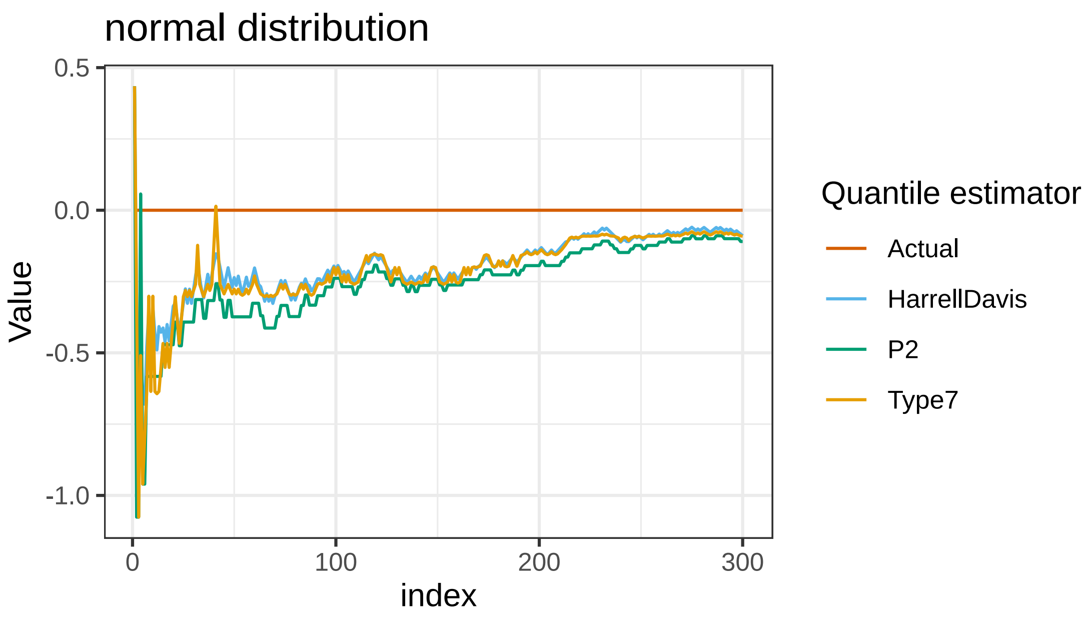
It's time to check how it works. I decided to visualize sequential values of the following quantiles estimator:

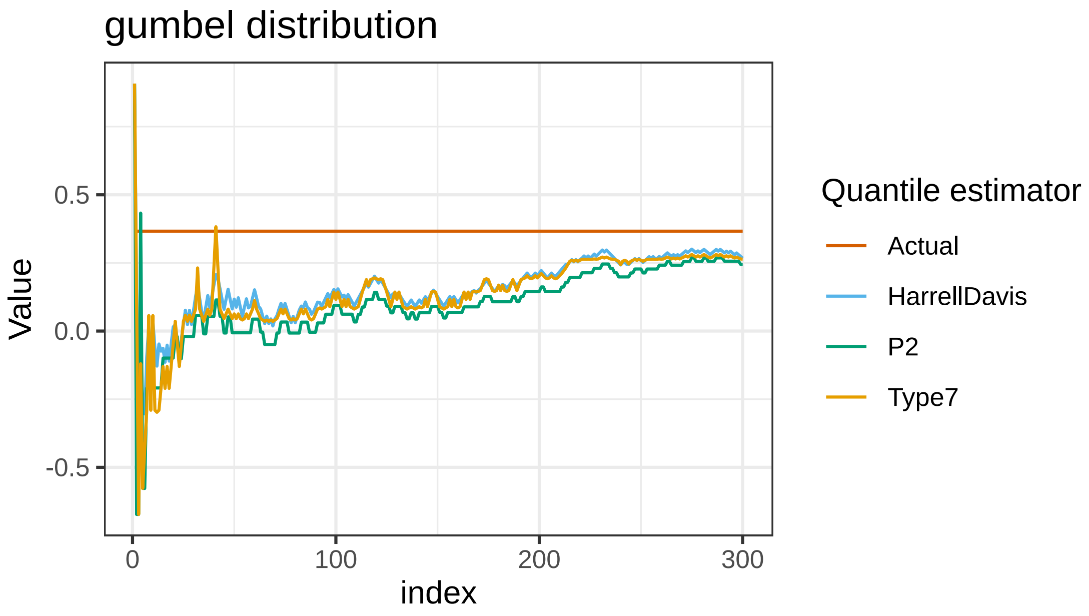
- **The P² quantile estimator**
  A sequential estimator that is described above.
- **The Type 7 quantile estimator**
  It's the most popular quantile estimator which is used by default in R, Julia, NumPy, Excel (`PERCENTILE`, `PERCENTILE.INC`), Python (`inclusive` method). We call it "Type 7" according to notation from [Hyndman1996], where Rob J. Hyndman and Yanan Fan described nine quantile algorithms which are used in statistical computer packages.
- **The Harrell-Davis quantile estimator**
  It's my favorite option in real life for non-sequential cases because it's more efficient than classic quantile estimators based on linear interpolation, and it provides more reliable estimations on small samples. This quantile estimator is described in [Harrell1982].
- **Actual**
  The true median value which is taken from the underlying distribution.

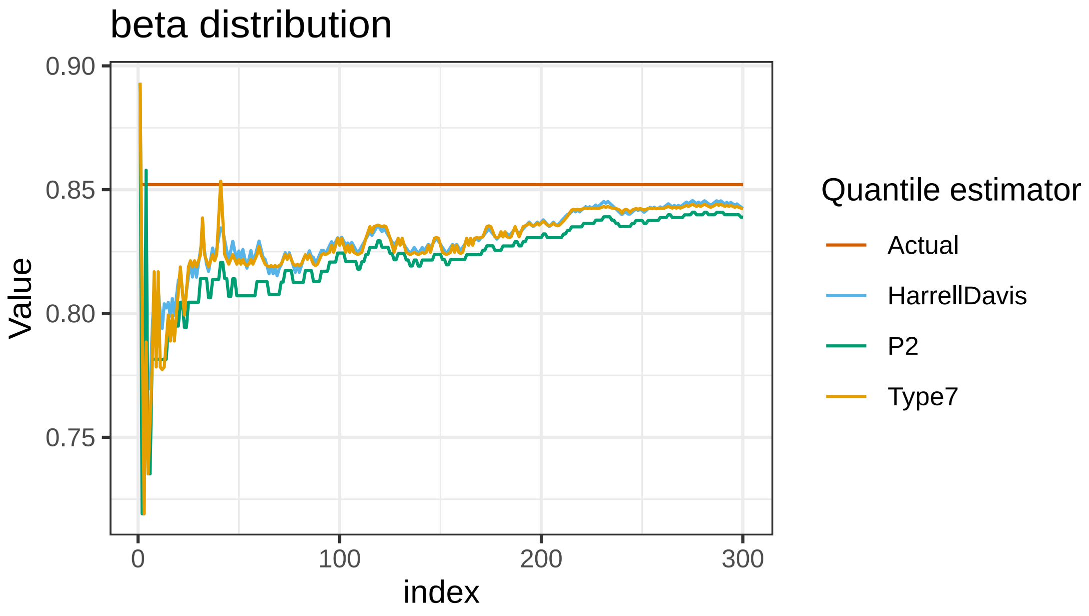
Below, you can find several plots for the following distributions:

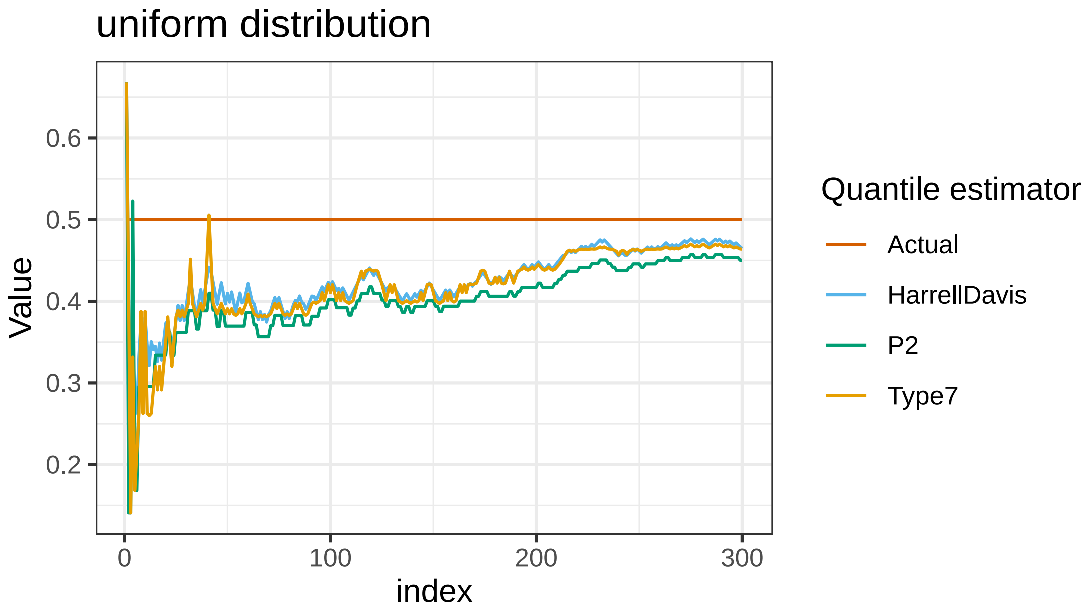
- **Normal distribution** $\mathcal{N}(0,1)$
- **Gumbel distribution** for $\mu = 0, \beta = 1$
- **Beta distribution** $\text{Beta}(10,2)$
- **Uniform distribution** $\mathcal{U}(0,1)$
- **Bimodal distribution** (mixture of $\mathcal{N}(10,1)$ and $\mathcal{N}(20,1)$)

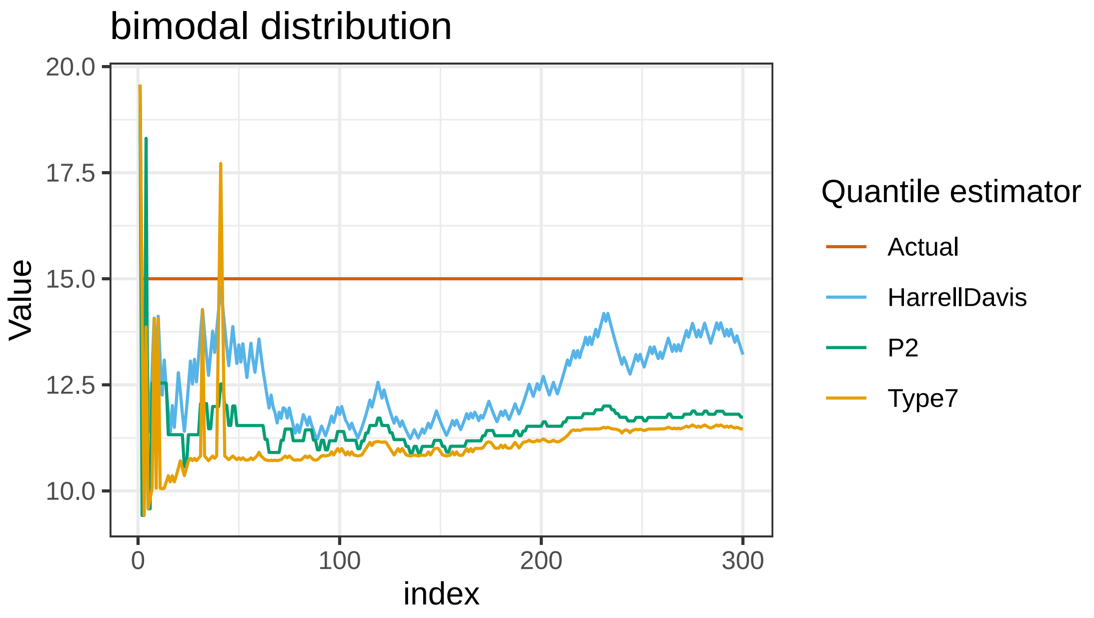
Here are the results:




gumbel distribution
Value
0.5
0.0
-0.5
0
100
200
300
index
Quantile estimator
Actual
HarrellDavis
P2
Type7


beta distribution
Value
0.90
0.85
0.80
0.75
0
100
200
300
index
Quantile estimator
Actual
HarrellDavis
P2
Type7


uniform distribution
Value
0.6
0.5
0.4
0.3
0.2
0
100
200
300
index
Quantile estimator
Actual
HarrellDavis
P2
Type7



As you can see, The P² quantile estimator produces reasonable median estimates. I also checked how it works on a considerable number of real data sets and I'm pretty satisfied with the results. You can also find a discussion about accuracy and the equation for the mean squared error in the original paper.

Reference implementation

Below you can find a C# implementation of the discussed algorithm. Also, you can use it via the latest nightly version (0.3.0-nightly.64+) of perfolizer.

Update: an updated implementation is available here.

```
public class P2QuantileEstimator
{
    private readonly double p;
    private readonly int[] n = new int[5]; // marker positions
    private readonly double[] ns = new double[5]; // desired marker positions
    private readonly double[] dns = new double[5];
    private readonly double[] q = new double[5]; // marker heights
    private int count;

    public P2QuantileEstimator(double probability)
    {
        p = probability;
    }

    public void AddValue(double x)
    {
        if (count < 5)
        {
            q[count++] = x;
            if (count == 5)
            {
                Array.Sort(q);

                for (int i = 0; i < 5; i++)
                    n[i] = i;

                ns[0] = 0;
                ns[1] = 2 * p;
                ns[2] = 4 * p;
                ns[3] = 2 + 2 * p;
                ns[4] = 4;
                dns[0] = 0;
                dns[1] = p / 2;
                dns[2] = p;
                dns[3] = (1 + p) / 2;
                dns[4] = 1;
            }

            return;
        }

        int k;
        if (x < q[0])
        {
            q[0] = x;
            k = 0;
        }
        else if (x < q[1])
            k = 0;
        else if (x < q[2])
            k = 1;
        else if (x < q[3])
            k = 2;
        else if (x < q[4])
            k = 3;
        else
        {
            q[4] = x;
            k = 3;
        }

        for (int i = k + 1; i < 5; i++)
            n[i]++;
        for (int i = 0; i < 5; i++)
            ns[i] += dns[i];

        for (int i = 1; i <= 3; i++)
        {
            double d = ns[i] - n[i];
            if (d >= 1 && n[i + 1] - n[i] > 1 || d <= -1 && n[i - 1] - n[i] < -1)
            {
                int dInt = Math.Sign(d);
                double qs = Parabolic(i, dInt);
                if (q[i - 1] < qs && qs < q[i + 1])
                    q[i] = qs;
                else
                    q[i] = Linear(i, dInt);
                n[i] += dInt;
            }
        }

        count++;
    }

    private double Parabolic(int i, double d)
    {
        return q[i] + d / (n[i + 1] - n[i - 1]) * (
            (n[i] - n[i - 1] + d) * (q[i + 1] - q[i]) / (n[i + 1] - n[i]) +
            (n[i + 1] - n[i] - d) * (q[i] - q[i - 1]) / (n[i] - n[i - 1])
        );
    }
    private double Linear(int i, int d)
    {
        return q[i] + d * (q[i + d] - q[i]) / (n[i + d] - n[i]);
    }

    public double GetQuantile()
    {
        if (count == 0)
            throw new InvalidOperationException("Sequence contains no elements");
        if (count <= 5)
        {
            Array.Sort(q, 0, count);
            int index = (int) Math.Round((count - 1) * p);
            return q[index];
        }

        return q[2];
    }
}
```

Conclusion

The P² quantile estimator allows estimating quantile values on a stream of numbers without storing individual values. There are many other sequential quantile estimators, but the P² quantile estimator is one of the most simple from the implementation point of view. Meanwhile, it provides good accuracy and low overhead. The algorithm can be used to introduce performance telemetry in your application without noticeable performance or memory footprint overhead.

References

- **[Jain1985]**
  Jain, Raj, and Imrich Chlamtac. “The P² algorithm for dynamic calculation of quantiles and histograms without storing observations.” Communications of the ACM 28, no. 10 (1985): 1076-1085.
  https://doi.org/10.1145/4372.4378
- **[Harrell1982]**
  Harrell, F.E. and Davis, C.E., 1982. A new distribution-free quantile estimator. *Biometrika*, 69(3), pp.635-640.
- **[Hyndman1996]**
  Hyndman, R. J. and Fan, Y. 1996. Sample quantiles in statistical packages, *American Statistician* 50, 361–365.
  https://doi.org/10.2307/2684934