STUDIA Z AUTOMATYKI I INFORMATYKI
TOM 38 – 2013

Piotr Kozierski,[*] Marcin Lis,[†] Joanna Ziętkiewicz[‡]

RESAMPLING IN PARTICLE FILTERING – COMPARISON

1. Introduction

The Particle Filter (PF) method is becoming increasingly popular. Is often used especially for complex objects where other methods fail. The origins of particle filter are associated with introduction of resampling [6]. Previously, some methods without resampling were known (SIS – Sequential Importance Sampling), but the drawback was degeneration (the algorithm can work for a few iterations only).

One can find a lot of resampling versions in the references. The purpose of this article is to bring together as many resampling methods as possible and their comparison.

In second Section, Particle Filter principle of operation has been described. In third Section, different resampling methods have been shown with pseudo-code for every method provided. Fourth Section describes how simulations have been performed and contains results. Conclusions and summary of the whole article are in last Section.

2. Particle Filter

PF principle of operation is based on Bayes filter

$$\overbrace{p\left(\mathbf{x}^{(k)}|\mathbf{Y}^{(k)}\right)}^{\text{posterior}} = \frac{\overbrace{p\left(\mathbf{y}^{(k)}|\mathbf{x}^{(k)}\right)}^{\text{likelihood}} \cdot \overbrace{p\left(\mathbf{x}^{(k)}|\mathbf{Y}^{(k-1)}\right)}^{\text{prior}}}{\underbrace{p\left(\mathbf{y}^{(k)}|\mathbf{Y}^{(k-1)}\right)}_{\text{evidence}}}\,, \tag{1}$$

where: $\mathbf{x}^{(k)}$ – vector of object state in k-th time step, $\mathbf{y}^{(k)}$ – vector of measurements in k-th time step, $\mathbf{Y}^{(k)}$ – set of measurements from beginning to k-th time step. Evidence in (1) is a normalizing coefficient, so it can be written as

$$\overbrace{p\left(\mathbf{x}^{(k)}|\mathbf{Y}^{(k)}\right)}^{\text{posterior}} \propto \overbrace{p\left(\mathbf{y}^{(k)}|\mathbf{x}^{(k)}\right)}^{\text{likelihood}} \cdot \overbrace{p\left(\mathbf{x}^{(k)}|\mathbf{Y}^{(k-1)}\right)}^{\text{prior}}\,, \tag{2}$$

[*]Poznan University of Technology, Faculty of Electrical Engineering, Institute of Control and Information Engineering, Piotrowo 3a street, 60-965 Poznań, e-mail: `piotr.kozierski@gmail.com`

[†]Poznan University of Technology, Faculty of Electrical Engineering, Institute of Electrical Engineering and Electronics, Piotrowo 3a street, 60-965 Poznań, e-mail: `mail.dla.studenta@gmail.com`

[‡]Poznan University of Technology, Faculty of Electrical Engineering, Institute of Control and Information Engineering, Piotrowo 3a street, 60-965 Poznań, e-mail: `joanna.zietkiewicz@put.poznan.pl`

where symbol $\propto$ means “is proportional to”. The main task of filter is estimation of posterior Probability Density Function (PDF).

The PF is one of possible implementations of Bayes filter. In this approach it has been assumed that posterior PDF is composed of particles. Each particle has some value $\mathbf{x}^i$ (state vector) and weight q^i. The greater the weight of the particle is, the greater becomes the probability that the value (state) of the particle is correct. It is thus seen that while the Bayesian filter is described by continuous distribution, the result of PF work is discrete distribution.

The accuracy of the PF depends on the number of particles N – the greater number of particles, the more accurate PDF. When N tends to infinity, it can be written that

$$p\left(\mathbf{x}^{(k)}|\mathbf{Y}^{(k)}\right) \overset{N\to\infty}{=} \hat{p}\left(\mathbf{x}^{(k)}|\mathbf{Y}^{(k)}\right) = \sum_{i=1}^{N} q^{i,(k)} \cdot \mathbf{x}^{i,(k)} \ . \tag{3}$$

Derivations and clear description of PF can be found in [1, 12], and below the final form of the algorithm has been only shown.

Algorithm 1.

1. Initialization. Draw particles from initial PDF $\mathbf{x}^{i,(0)} \sim p(\mathbf{x}^{(0)})$, set time step number $k = 1$.
2. Prediction. Draw N new particles from transition model $\mathbf{x}^{i,(k)} \sim p(\mathbf{x}^{(k)}|\mathbf{x}^{i,(k-1)})$.
3. Update. Calculate weights of particles base on measurement model $q^{i,(k)} = q^{i,(k-1)} \cdot p(\mathbf{y}^{(k)}|\mathbf{x}^{i,(k)})$.
4. Normalization. Normalize particle weights – their sum should be equal to 1.
5. Resampling. Draw N new particles based on posterior PDF, obtained in steps 2–4.
6. End of iteration. Calculate estimated value of state vector $\hat{\mathbf{x}}^{(k)}$, increase time step number $k = k + 1$, go to step 2.

The latter is not the generic form of the algorithm, since in step 2 one can draw from any PDF in general, however in this case, expression of calculating weights in step 3 would be more complicated. As one can see, this is one of many possible algorithms PF, but simultaneously it is one of the easiest to implement. Some variants of PF are, e.g., Marginal PF [11], Rao-Blackwellized PF [7], Likelihood PF [1]. There are many other algorithms PF, which can be found in the literature.

3. RESAMPLING METHODS

3.1. PRELIMINARIES

Resampling consists of drawing the N new particles from posterior PDF prepared in previous steps. One can ask for the purpose, since the same thing is done in next iteration (second step in Algorithm 1)? It should be noted that in the second step, each of the existing particle is moved according to the transition model $p(\mathbf{x}^{(k)}|\mathbf{x}^{i,(k-1)})$. On the other hand, as a result of the resampling, part of particles will be duplicated, and some rejected – the higher the weight, the greater the chance that the particle will be drawn several times. Thrun in [19] noted that resampling can be compared with a probabilistic implementation of Darwin's theory, which relates to adaptation by natural selection. In this case, the "adjustment" is the particle weight.

This Section describes the different types of resampling that can be found in references (but also methods proposed by authors) and their implementations (pseudo codes). In most cases, the basic arithmetic operations have been used, and among additional operations can be found such as:

- `rand()` – generate one random number from the uniform distribution,
- `SORT(vector)` – sorts values in vector from the smallest to the largest,
- `for i=N..1` – loop, wherein at each iteration there is decrement of variable i,
- $\mathbf{qq} = \frac{1}{N}$ – at the beginning of the code means that, regardless of the performed operations, weights of resampled particles always have values equal to $\frac{1}{N}$; in such a case, one can also modify Algorithm 1, in step 3 – method of weight calculating can be simplified to $q^{i,(k)} = p(\mathbf{y}^{(k)}|\mathbf{x}^{i,(k)})$,
- `u_s(i)^(1/i)` – this is the exponentiation (i-th vector element to the power $\frac{1}{i}$),
- `log2(x)` – base 2 logarithm,
- `prim=[7,11,13]` – declaration of specific vector values,
- `x mod y` – computes the remainder of dividing $\frac{x}{y}$,
- `floor(x)` – function round value x towards minus infinity,
- `max(vector)` – largest component of vector.

One can find another notation of resampling results in the literature. In this article, it was decided to provide information in its full form – values and weights of all particles. One can also transfer vector of N integers only, which inform how many times i-th particle has been drawn in the resampling.

It should also be noted that the subject of resampling is not exhausted in this article, because the possibility of different number of input and output particles has been omitted (and this is an important component of adaptive PF algorithms – see [17, 18]).

With PF the particles impoverishment phenomenon is also related. As a result of the drawing (step 2 in Algorithm 1), a lot of particles with weights close to zero are received, and only a few particles have significant weights. Some solutions of this problem assume suitable approach in the resampling step. However, this issue will not be discussed here, and interested readers are referred to the book [16].

3.2. Multinomial resampling

Multinomial resampling (MR), also called Simple Random Resampling [9], has been proposed together with first PF in [6]. It consists of drawing the N numbers from the uniform distribution

$$u^i \sim \mathcal{U}[0,1) \qquad i = 1, \ldots, N \ , \tag{4}$$

and selecting particle $\mathbf{x}^j$ for replication, such that

$$u^i \in \left[\sum_{p=1}^{j-1} q^p, \sum_{p=1}^{j} q^p \right) \ . \tag{5}$$

One can distinguish between the two implementations:

a) ascending sort of drawn numbers $\mathbf{u}$ to yield ordered set $\mathbf{u}_o$, and then compare with successive weight ranges,

b) creating a secondary set of numbers $\mathbf{Q} = [Q_1, \ldots, Q_N]$ based on expression

$$Q_j = \sum_{p=1}^{j} q^p = Q_{j-1} + q^j \ , \tag{6}$$

and then, using a binary search, select for replicate a particle $\mathbf{x}^j$ such that $u^i \in [Q_{j-1}, Q_j)$.

In both implementations, the complexity of the algorithm is $O(N \cdot \log_2 N)$. These algorithms are called $\mathrm{MR}_{1\mathrm{a}}$ and $\mathrm{MR}_{1\mathrm{b}}$.

Code 1.

$[\mathbf{xx}\ ,\ \mathbf{qq} = \frac{1}{N}] \ = \ \mathrm{MR}_{1\mathrm{a}}(\mathbf{x}\ ,\ \mathbf{q}\ ,\ N)$

```
for i=1..N
   u(i)=rand();
end
u_o=SORT(u);
sumQ=0; i=0; j=1;
while j<=N
   i=i+1;
   sumQ=sumQ+q(i);
   while (j<=N) && (sumQ>u_o(j))
      xx(j) = x(i);
      j=j+1;
   end
end
```

Code 2.

$[\mathbf{xx}\ ,\ \mathbf{qq} = \frac{1}{N}] \ = \ \mathrm{MR}_{1\mathrm{b}}(\mathbf{x}\ ,\ \mathbf{q}\ ,\ N)$

```
Q(0)=0;
for i=1..N
   Q(i)=Q(i-1)+q(i);
end
```

```
for i=1..N
   u=rand();
   begI=1; endI=N; stop=0;
   while !stop
      j = floor( (begI+endI)/2 );
      if (u>=Q(j-1)) && (u<Q(j))
         stop=1;
      else if (u<Q(j-1))
         endI=j-1;
      else
         begI=j+1;
      end
   end
   xx(i)=x(j);
end
```

However, in the literature, the approach utilizing linear complexity $O(N)$ is recommended – inverse CDF method [3, 5, 9]. In the first step, one must draw N random numbers u_s^i from uniform distribution, and then the numbers u^i are calculated. Thanks to this ordered random numbers are obtained. This transformation is described by the equations:

$$u^N = \sqrt[N]{u_s^N} \ , \tag{7}$$

$$u^i = u^{i+1} \sqrt[i]{u_s^i} \ . \tag{8}$$

Based on vector of values $\mathbf{u} = [u^1, \ldots, u^N]$, ranges (5) should be found and the corresponding particle $\mathbf{x}^j$ should be chosen for replication.

Code 3.
$[\mathbf{xx} \ , \ \mathbf{qq} = \frac{1}{N}] \ = \ \mathrm{MR}_2(\mathbf{x} \ , \ \mathbf{q} \ , \ N)$

```
for i=1..N
   u_s(i)=rand();
end
u(N)=u_s(N)^(1/N);
for i=N-1..1
   u(i)=u(i+1) * u_s(i)^(1/i);
end
sumQ=0; i=0; j=1;
while j<=N
   i=i+1;
   sumQ=sumQ+q(i);
   while (j<=N) && (sumQ>u(j))
      xx(j)=x(i);
      j=j+1;
   end
end
```

Another approach has been proposed in [4], although the target is similar to that of MR$_2$, i.e. to reduce the computational complexity to $O(N)$. This has been achieved by transformation of the random numbers received from uniform distribution $u_s^i \sim \mathcal{U}[0, 1)$

$$u^i = -\log_2(u_s^i) \ , \tag{9}$$

but the range of numbers is greater than previously by 1 ($i = 1, \ldots, N+1$). Afterwards numbers $T_1, \ldots, T_{N+1}$ should be prepared according to the expression

$$T_i = \sum_{p=1}^{i} u^p \ . \tag{10}$$

Values Q_j calculated from the formula (6) are also needed. Based on values T_i (for $i = 1, \ldots, N$) particles $\mathbf{x}^j$ are selected, for which the relation

$$T_i \in [Q_{j-1} \cdot T_{N+1} \ , \ Q_j \cdot T_{N+1}) \ , \tag{11}$$

is satisfied, which can be written as

$$\frac{T_i}{T_N} \in [Q_{j-1}, Q_j) \ . \tag{12}$$

It can be seen that there is a normalization of calculated values. Common sense dictates that certain actions are unnecessary, and this is true – transformation (9) is superfluous and, subsequently, values of T_i values could be calculated on the basis of $\mathbf{u}_s$. The authors in [4] propose another resampling, which is based on MR$_3$, but also uses additional parameters, obtained in the previous calculations (this is another PF version, which is not presented here). In Code 4, instead of creating a vector **Q**, the variable `sumQ` has been used, which is updated with the incrementation of `j`.

Code 4.

```
$[\mathbf{xx}\ ,\ \mathbf{qq} = \frac{1}{N}]\ =\ \mathrm{MR}_3(\mathbf{x}\ ,\ \mathbf{q}\ ,\ N)$
for i=1..N+1
   u_s(i)=rand();
end
for i=1..N+1
   u(i)=-log2(u_s(i));
end
T(1)=u(1);
for i=2..N+1
   T(i)=T(i-1)+u(i);
end
for i=1..N
   T(i)=T(i)/T(N+1);
end
i=1; j=1; sumQ=q(1);
while i<=N
   if sumQ>T(i)
      xx(i)=x(j);
      i=i+1;
   else
      j=j+1;
      sumQ=sumQ+q(j);
   end
end
```

3.3. Stratified resampling

Stratified resampling (StR) has been proposed in [10] for the first time. In the algorithm, it is assumed that division into strata (layers) is performed. In each stratum resampling can be performed simultaneously. However, also in this case appeared variations of method.

The approach, that can be easily implemented with complexity $O(N)$, assumes that the range $[0, 1)$ is subdivided into N equal parts, and the draw occurs in each such stratum [5, 14]

$$u^i \sim \left[\frac{i-1}{N}, \frac{i}{N}\right) \ . \tag{13}$$

Particles $\mathbf{x}^j$ are selected for replication in such a way that expression (5) is fulfilled.

Code 5.

$[\mathbf{xx}\ ,\ \mathbf{qq} = \frac{1}{N}] \ = \ \mathrm{StR}_1(\mathbf{x}\ ,\ \mathbf{q}\ ,\ N)$

```
j=1; sumQ=q(j);
for i=1..N
   u=(rand()+i-1)/N;
   while sumQ<u
      j=j+1;
      sumQ=sumQ+q(j);
   end
   xx(i)=x(j);
end
```

Another method is to split particles into n_s strata – in j-th stratum there are N_j particles with a total weight w_j. This is a more general approach that has been proposed in [8, 9]. However, it should be noted, that if the condition

$$\frac{w_i}{N_i} = \frac{w_j}{N_j} \tag{14}$$

is not satisfied for any numbers i and j, particle weights after resampling are different. It means that in step 3 of Algorithm 1, weights from the previous time step must be taken into account.

StR method should be chosen when it is possible to implement parallel computing. Division into strata can be performed according to the number of particles (however into stratum one may find particles only with zero or near-zero weights), or according to layer weight w_j. Below, the algorithm for splitting into layers with a similar number of particles is given.

Code 6.

$[\mathbf{xx}\ ,\ \mathbf{qq}] \ = \ \mathrm{StR}_2(\mathbf{x}\ ,\ \mathbf{q}\ ,\ N\ ,\ P)$

```
Tres=1; sum=0; i=1; j=0;
while (j<N)
   while (sum<Tres) && (j<N)
      sum=sum+P/N;
      j=j+1;
   end //stratum includes particles i:j
   Tres=Tres+1;
   sumQ(i)=q(i);
   for ii=i+1..j
```

```
10.       sumQ(ii)=sumQ(ii-1)+q(ii);
11.    end
12.    for ii=i..j
13.       qq(ii)=sumQ(j)/(j-i+1);
14.    end
15.    if sumQ(j)==0
16.       //PROBLEM!!! All weights in stratum are equal to 0
17.       for ii=i..j
18.          xx(ii)=x(ii);
19.       end
20.    else
21.       for ii=i..j
22.          sumQ(ii)=sumQ(ii)/sumQ(j);
23.       end
24.       jj=i;
25.       for ii=i..j
26.          u=(rand()+ii-i)/(j-i+1);
27.          while u>sumQ(jj)
28.             jj=jj+1;
29.          end
30.          xx(ii)=x(jj);
31.       end
32.    end
33.    i=j+1;
34. end
```

In Code 6, in 16-th line one can see the problem – possible case in which all weights in stratum are equal to zero. This makes the algorithm unable to continue (division by zero). But leaving the resampling for this layer (as in Code 6) is not a good solution, because in this case particles will be degenerated, such as in SIS algorithm (this method is quite similar to PF, but there is no resampling, so that after several iterations all particles except one have weight close or equal to zero). Authors of this article propose the following solution to this problem:

a) if sum of weights in stratum is equal to 0, leave the other calculations, but perform resampling StR$_1$ every K_R iterations of PF algorithm,

b) if sum of weights in stratum is equal to 0, leave the other calculations and compute estimated state vector; then, replace all particles with weights equal to zero, with the estimated state vector value and assign weights equal to $\frac{1}{N}$,

c) problem of strata degeneration is due to the fact that the same particles are in the same stratum in each iteration; so approach to random assignment to the layers has been proposed.

The implementation of these ideas are presented in the following three algorithms.

Code 7.

$[\mathbf{xx}\ ,\ \mathbf{qq}]\ =\ \mathrm{StR}_{2\mathrm{a}}(\mathbf{x}\ ,\ \mathbf{q}\ ,\ N\ ,\ P\ ,\ k\ ,\ K_R)$

```
if (k mod KR == 0)
   [xx , qq] = $\mathrm{StR}_1$(x , q , N)
else
   [xx , qq] = $\mathrm{StR}_2$(x , q , N , P)
end
```

Code 8. (StR_{2b} in PF algorithm)

1. Initialization.
2. Prediction.
3. Actualization.
4. Normalization.
5. Resampling:
   ```
   [xx , qq] = StR_2(x , q , N , P)
   ```
6. State $\hat{\mathbf{x}}^{(k)}$ estimation.
7. End of resampling:
   ```
   for i=1..N
      if qq(i)==0
         xx(i)=$\hat{\mathbf{x}}^{(k)}$
         qq(i)=1/N;
      end
   end
   ```
8. Increase iteration step. Go to step 2.

Code 9.

$[\mathbf{xx}\ ,\ \mathbf{qq}]\ =\ \mathrm{StR}_{2c}(\mathbf{x}\ ,\ \mathbf{q}\ ,\ N\ ,\ P)$

```
prim=[7,11,13,17,19,23,29,31,37,41];
do
   pr=prim( floor(rand()*10)+1 );
while (N mod pr == 0) //choice pr relatively prime to N
j=floor(rand()*N)+1;
for i=1..N
   id(i)=j; //in each iteration different indexes vector ’id’
   j=((j+pr-1) mod N)+1;
end
Tres=1; sum=0; i=1; j=0;
while (j<N)
   while (sum<Tres) && (j<N)
      sum=sum+P/N;
      j=j+1;
   end
   Tres=Tres+1;
   sumQ(i)=q(id(i));
   for ii=i+1..j
      sumQ(ii)=sumQ(ii-1)+q(id(ii));
   end
   for ii=i..j
      qq(id(ii))=sumQ(j)/(j-i+1);
   end
   if sumQ(j)==0
      //PROBLEM!!! All weights in stratum are equal to 0
      for ii=i..j
         xx(id(ii))=x(id(ii));
      end
   else
      for ii=i..j
         sumQ(ii)=sumQ(ii)/sumQ(j);
      end
      jj=i;
      for ii=i..j
         u=(rand()+ii-i)/(j-i+1);
         while u>sumQ(jj)
```

```
            jj=jj+1;
         end
         xx(id(ii))=x(id(jj));
      end
   end
   i=j+1;
end
```

However, apart from these three solutions another approach can be used. In presented algorithms, number of particles in stratum and the number of particles obtained from stratum are the same. The approach proposed in the literature [9] assumes that the number of drawn particles from the layer should be proportional to the total weight of the stratum. This case, however, also may have different solutions:

d) particle weights are calculated according to the theory, and therefore proportionally to the total weight of the layer,

e) authors of this article proposed approach, in which weights after resampling are approximate and equal to $\frac{1}{N}$. This eliminates the need to remember the weights for the next algorithm PF iteration.

Code 10.

$[\mathbf{xx}\ ,\ \mathbf{qq}]\ =\ \mathrm{StR_{2d}}(\mathbf{x}\ ,\ \mathbf{q}\ ,\ N\ ,\ P)$

```
Tres=1; sum=0; i=1; j=0;
to_draw=0; out=0;
while (j<N)
   while (sum<Tres) && (j<N)
      sum=sum+P/N;
      j=j+1;
   end
   Tres=Tres+1;
   sumQ(i)=q(i);
   for jj=i+1..j
      sumQ(jj)=sumQ(jj-1) + q(jj);
   end
   to_draw=to_draw + sumQ(j)*N;
   for ii=out+1..out+floor(to_draw)
      qq(ii)=sumQ(j)/floor(to_draw);
   end
   for jj=i..j
      sumQ(jj)=sumQ(jj)/sumQ(j);
   end
   jj=i;
   for ii=out+1..out+floor(to_draw)
      u=(rand()+ii-out-1)/floor(to_draw);
      while u>sumQ(jj);
         jj=jj+1;
      end
      xx(ii)=x(jj);
   end
   i=j+1;
   out=out+floor(to_draw);
   to_draw=to_draw-floor(to_draw);
end
```

Code 11.

$[\mathbf{xx}\ ,\ \mathbf{qq} = \frac{1}{N}]\ =\ \mathrm{StR}_{2\mathrm{e}}(\mathbf{x}\ ,\ \mathbf{q}\ ,\ N\ ,\ P)$

```
Tres=1; sum=0; i=1; j=0;
to_draw=0; out=0;
while (j<N)
   while (sum<Tres) && (j<N)
      sum=sum+P/N;
      j=j+1;
   end
   Tres=Tres+1;
   sumQ(i)=q(i);
   for jj=i+1..j
      sumQ(jj)=sumQ(jj-1) + q(jj);
   end
   to_draw=to_draw + sumQ(j)*N;
   for jj=i..j
      sumQ(jj)=sumQ(jj)/sumQ(j);
   end
   jj=i;
   for ii=out+1..out+floor(to_draw)
      u=(rand()+ii-out-1)/floor(to_draw);
      while u>sumQ(jj);
         jj=jj+1;
      end
      xx(ii)=x(jj);
   end
   i=j+1;
   out=out+floor(to_draw);
   to_draw=to_draw-floor(to_draw);
end
```

The division due to the particles weight is not considered, because the chance of degeneracy (one particle in several different strata) is too large.

Yet another approach to the division of particles has been presented in [2]. Authors assumed that the number of strata is equal to 3, and there are another calculations in each layer. However, the authors of this resampling named it separately, and as such it has been described in Section 3.7.

3.4. SYSTEMATIC RESAMPLING

Systematic resampling (SR) in all references is presented in the same way. Firstly, it has been proposed by Carpenter in 1999 and called by him “stratified”, whereas in the literature one can find also the name “universal” [5]. But definitively the name “systematic” has been adopted.

SR principle of operation is quite similar to StR$_1$, described in previous subsection. The difference is that in whole resampling step, random number is drawed only once:

$$u_s \sim \mathcal{U}\left[0, \frac{1}{N}\right)\ , \tag{15}$$

$$u^i\ =\ \frac{i-1}{N} + u_s\ . \tag{16}$$

Particles $\mathbf{x}^j$ for replication are selected based on expression (5).

This resampling has a complexity of $O(N)$, and is one of the more readily recommended, because of its simplicity and operation speed. However, it should be noted that a simple modification of the order before resampling can change the obtained PDF [5].

Code 12.

$[\mathbf{xx}\ ,\ \mathbf{qq} = \frac{1}{N}]\ =\ \mathrm{SR}(\mathbf{x}\ ,\ \mathbf{q}\ ,\ N)$

```
j=1; sumQ=q(j);
u=rand()/N;
for i=1..N
   while sumQ<u
      j=j+1;
      sumQ=sumQ+q(j);
   end
   xx(i)=x(j);
   u=u+1/N;
end
```

Authors of this article propose some variant of SR algorithm – Deterministic Systematic resampling (DSR). It has been assumed that, instead of drawing the random number, parameter u_s is constant in each particle filter iteration. The algorithm is then completely deterministic, what may be applicable, for example, in the implementation on the FPGA, where there is no `rand()` function. This will save memory and duration of action, because also in the later calculations, this function will not be needed (methods of drawing numbers from a Gaussian distribution without the use of pseudo-random numbers with uniform distribution are known, e.g. Wallace method [13]).

Code 13.

$[\mathbf{xx}\ ,\ \mathbf{qq} = \frac{1}{N}]\ =\ \mathrm{DSR}(\mathbf{x}\ ,\ \mathbf{q}\ ,\ N\ ,\ u_s)$

```
j=1; sumQ=q(j);
u=u_s/N;
for i=1:N
   while sumQ<u
      j=j+1;
      sumQ=sumQ+q(j);
   end
   xx(i)=x(j);
   u=u+1/N;
end
```

The authors recommend to take a small u_s value, for example 0.1 or 0.05. Thanks to this, in the last loop iteration (for `i=N`), the probability that some action could be omitted is greater (if the last particles have little weight).

3.5. Residual resampling

In Residual Resampling (RR), also called “remainder resampling” [5], it is assumed that for particles with large weight new particles can be assigned without drawing.

The algorithm is divided into two main parts. In the first part, particles with weights greater than $\frac{1}{N}$ are selected, and they are transferred for replication without draws. For these particles their input weights are reduced by a multiple of $\frac{1}{N}$. In the second part, the weight normalization and simple resampling is performed – one of the previously described. This causes that number of resampling variations can be as many as all other resampling methods. However, in this article stratified resampling StR_1 has been selected.

Code 14.

$[\mathbf{xx}\ ,\ \mathbf{qq} = \frac{1}{N}]\ =\ \mathrm{RR}_1(\mathbf{x}\ ,\ \mathbf{q}\ ,\ N)$

```
Nr=N; jj=0;
for i=1..N
   j=floor(q(i)*N);
   for ii=1..j
      jj=jj+1;
      xx(jj)=x(i);
   end
   q(i)=q(i)-j/N;
   Nr=Nr-j;
end
if Nr>0
   for i=1..N
      q(i)=q(i)*N/Nr;
   end
   j=1; sumQ=q(1);
   for i=1..Nr
      u=(rand()+i-1)/Nr;
      while sumQ<u
         j=j+1;
         sumQ=sumQ+q(j);
      end
      jj=jj+1;
      xx(jj)=x(j);
   end
end
```

In [2] interesting modification has been proposed, i.e., Residual-Systematic resampling (RSR). This algorithm combines two parts RR, so that only one loop is executed and the normalization is not necessary.

Code 15.

$[\mathbf{xx}\ ,\ \mathbf{qq} = \frac{1}{N}]\ =\ \mathrm{RSR}(\mathbf{x}\ ,\ \mathbf{q}\ ,\ N)$

```
jj=0;
u=rand()/N;
for i=1..N
   j=floor( (q(i)-u)*N )+1;
   for ii=1..j
      jj=jj+1;
      xx(jj)=x(i);
   end
   u=u+j/N-q(i);
end
```

It should be noted that the result of RSR operation is the same as for RR in combination with SR.

Authors of this article propose one more variation of RR. After weight normalization in second part of RR method, weights greater than $\frac{1}{N}$ can be found. Therefore, one can make the choice of particles for replication on similar terms as in the first part of method. Proposed method is deterministic, as well as DSR. This makes it suitable for implementation on a computational unit, which does not have implementation of the random number generator with uniform distribution.

Code 16.

$[\mathbf{xx}\ ,\ \mathbf{qq} = \frac{1}{N}]\ =\ \mathrm{RR}_2(\mathbf{x}\ ,\ \mathbf{q}\ ,\ N)$

```
jj=0;
while jj<N
   Nr=N;
   for i=1..N
      j=floor(q(i)*N);
      for ii=1..j
         if jj<N
            jj=jj+1;
            xx(jj)=x(i);
         end
      end
      q(i)=q(i)-j/N;
      Nr=Nr-j;
   end
   for i=1..N
      q(i)=q(i)*N/Nr;
   end
end
```

3.6. Metropolis resampling

Metropolis Resampling (MeR) has been shown in [14]. Among other resampling methods is characterized in that, there is no need to create distribution (6), and ranges (5) are not necessary. Instead, it requires a relatively large number of random numbers with uniform distribution.

For each weight N_c comparisons with randomly selected weights are executed. If the weight of randomly selected particle is greater, then it shall be selected for further comparisons. However, if the weight of the randomly selected particle is smaller, it shall be selected with probability equal to the ratio of the weights. After N_c comparisons, the current particle is passed for replication.

One can see that the number of comparisons N_c is an important parameter, because too small number causes that resampling does not meet its role (this may be the case that a lot of particles with insignificant weights will be selected for replication).

Code 17.

$[\mathbf{xx}\ ,\ \mathbf{qq} = \frac{1}{N}]\ =\ \mathrm{MeR}(\mathbf{x}\ ,\ \mathbf{q}\ ,\ N\ ,\ N_c)$

```
for i=1..N
   ii=i;
   for j=1..Nc
      u=rand();
      jj=floor(rand()*N)+1;
      if u<=q(jj)/q(ii)
         ii=jj;
      end
   end
   xx(i)=x(ii);
end
```

3.7. Rejection resampling

Rejection Resampling (ReR) has been proposed in [15]. Like the MeR, it is based on comparisons with a random value. Difference is that in the ReR random value is compared to the another ratio – random weight and the largest weight within the set. In addition, by using a while loop, the exact time of resampling operation can not be predicted.

Code 18.

$[\mathbf{xx}\ ,\ \mathbf{qq} = \frac{1}{N}]\ =\ \mathrm{ReR}_1(\mathbf{x}\ ,\ \mathbf{q}\ ,\ N)$

```
maxQ=max(q);
for i=1..N
   j=i;
   u=rand();
   while u>q(j)/maxQ
      j=floor(rand()*N)+1;
      u=rand();
   end
   xx(i)=x(j);
end
```

Some modification of ReR method has been proposed – in the calculated ratio the constant value is considered, instead of a maximum weight. Parameter g is used, which, however, has the inverse values (e.g. 1, 5, 10 instead of 1, $\frac{1}{5}$, $\frac{1}{10}$), and in the algorithm there is multiplication instead of division.

Code 19.

$[\mathbf{xx}\ ,\ \mathbf{qq} = \frac{1}{N}]\ =\ \mathrm{ReR}_2(\mathbf{x}\ ,\ \mathbf{q}\ ,\ N\ ,\ g)$

```
for i=1..N
   j=i;
   u=rand();
   while u>q(j)*g;
      j=floor(rand()*N)+1;
      u=rand();
   end
   xx(i)=x(j);
end
```

3.8. Partial resampling

Partial Resampling (PR) has been proposed in [2]. In this approach two thresholds have been assumed, i.e., high T_H and low T_L. All particles are divided into 3 strata, depending on their weight. Selection for replication is different, depending on the layer, to which the particle is assigned.

There are presented two variants of PR below, which have been proposed in [2]. Unfortunately, due to the large number of errors in the source article, the algorithms shown below may differ in detail from the original.

In Partial Stratified Resampling (PSR), it is assumed that calculations are performed only for the particles which weights satisfy the condition $q^i \geqslant T_H$ or $q^i < T_L$. Whereas particles with average weights are passed unchanged for replication, with the old weights. Therefore, the weights of individual particles after resampling may be different – it must be remembered due to weights calculation in next iteration (step 3 in Algorithm 1). In the second part of the resampling modified StR$_1$ algorithm has been selected.

Code 20.

$[\mathbf{xx}\ ,\ \mathbf{qq}]\ =\ \mathrm{PSR}(\mathbf{x}\ ,\ \mathbf{q}\ ,\ N\ ,\ T_L\ ,\ T_H)$

```
N_low=0; N_med=0; N_high=0; QTresSum=0;
for i=1..N
   if q(i)<T_L
      N_low=N_low+1;
      ind_low(N_low)=i;
      QTresSum=QTresSum+q(i);
   elseif q(i)>=T_H
      N_high=N_high+1;
      ind_high(N_high)=i;
      QTresSum=QTresSum+q(i);
   else
      N_med=N_med+1;
      ind_med(N_med)=i;
   end
end
if N_low>0
   j=1; sumQ=q(ind_low(j));
elseif N_high>0
   j=1; sumQ=q(ind_high(j));
end
for i=1..N_low+N_high
   u=(rand()+i-1)*QTresSum/(N_low+N_high);
   while sumQ<u
      j=j+1;
      if j<=N_low
         sumQ=sumQ+q(ind_low(j));
      else
         sumQ=sumQ+q(ind_high(j-N_low));
      end
   end
   if j<=N_low
      xx(i)=x(ind_low(j));
   else
      xx(i)=x(ind_high(j-N_low));
   end
   qq(i)=QTresSum/(N_low+N_high);
end
j=0;
for i=N_low+N_high+1..N
   j=j+1;
   xx(i)=x(ind_med(j));
   qq(i)=q(ind_med(j));
end
```

Another approach is Partial Deterministic Resampling (PDR) – particles which weighs are less than T_L can not be drawn for replication. Whereas all particles with high weights are selected for replication the same number of times (with error of one particle). It is therefore another example of resampling, which does not use a random number generator. Particles with average weights, as previously, remain unchanged.

Additionally, the assumption has been introduced, that both the number of particles with high weights and the particles number with low weights, must be greater than zero.

Code 21.

$[\mathbf{xx}\ ,\ \mathbf{qq}]\ =\ \mathrm{PDR}(\mathbf{x}\ ,\ \mathbf{q}\ ,\ N\ ,\ T_L\ ,\ T_H)$

```
N_low=0; N_med=0; N_high=0; QLowSum=0; QHighSum=0;
for i=1..N
   if q(i)<T_L
      N_low=N_low+1;
      ind_low(N_low)=i;
      QLowSum=QLowSum+q(i)
   elseif q(i)>=T_H
      N_high=N_high+1;
      ind_high(N_high)=i;
      QHighSum=QHighSum+q(i);
   else
      N_med=N_med+1;
      ind_med(N_med)=i;
   end
end
if (N_low>0) && (N_high>0)
   ii=0;
   Nn=(N_high+N_low) mod N_high;
   for i=1..Nn
      for j=1..floor(N_low/N_high)+2
         ii=ii+1;
         xx(ii)=x(ind_high(i));
         qq(ii)=q(ind_high(i)) / (floor(N_low/N_high)+2)
 * (QLowSum+QHighSum) / QHighSum;
      end
   end
   for i=Nn+1..N_high
      for j=1..floor(N_low/N_high)+1
         ii=ii+1;
         xx(ii)=x(ind_high(i));
         qq(ii)=q(ind_high(i)) / (floor(N_low/N_high)+1)
 * (QLowSum+QHighSum) / QHighSum;
      end
   end
   for i=1..N_med
      ii=ii+1;
      xx(ii)=x(ind_med(i));
      qq(ii)=q(ind_med(i));
   end
else
   for i=1..N
      xx(i)=x(i);
      qq(i)=q(i);
   end
end
```

4. SIMULATIONS

4.1. SIMULATION CONDITIONS

All simulations have been carried out for the same object and the same noise signals sequence. The object can be written by equations:

$$\begin{aligned} x_1^{(k+1)} &= x_1^{(k)} \cos\left(x_1^{(k)} - x_2^{(k)}\right) + v_1^{(k)} \ , \\ x_2^{(k+1)} &= x_2^{(k)} \sin\left(x_2^{(k)} - x_1^{(k)}\right) + \cos\left(x_1^{(k)}\right) + v_2^{(k)} \ , \\ y_1^{(k)} &= x_1^{(k)} x_2^{(k)} + n_1^{(k)} \ , \\ y_2^{(k)} &= x_1^{(k)} + x_2^{(k)} + n_2^{(k)} \ , \end{aligned} \tag{17}$$

where $x_1^{(k)}$ is a value of first state variable in k-th time step, y is a measured value, v is a value of system noise and n is a value of measurement noise (both are Gaussian).

The simulation length is $M = 100$ time steps. Simulations were repeated $1,000$ times for each number of particles N and, in some cases, for different values of other parameters. For methods which use operating particle filters independently (with division into strata), the value of N is the total number of particles.

To determine the quality of estimation, the coefficient

$$D = 10^6 \cdot \sum_{i=1}^{2} (\mathrm{MSE}_i)^2 \tag{18}$$

has been used, which utilise for calculation the Mean Square Errors (MSE) of each state variable. The D value has been obtained after each simulation run, and the graphs shows the average value of $1,000$ performances.

4.2. RESULTS

In Figure 1 all method MR (multinomial resampling) versions have been compared, as well as for methods StR$_1$ (Stratified resampling) and SR (Systematic resampling). Unfortunately, this scale chart is unintelligible.

It was decided to present a graph on an unusual scale in the vertical axis. For each number of particles the maximum and minimum values have been found, and chart in the Figure 2 is shown with relation to these values.

One can see that methods StR$_1$ and SR are slightly better than methods MR, for small values of N. This can be explained by the fact that values drawn from the interval $[0, 1)$ are more evenly distributed for methods SR and StR$_1$. For large N, this dependency is no longer visible.

Figure 3 shows comparison of obtained results for resampling StR$_{2a}$. Parameters are L_p (number of strata) and K_R (method StR$_1$ is activated each K_R iteration). For comparison, also results obtained by the StR$_1$ method have been shown. Whereas Figure 4 shows all the results again in a normalized scale.

It can be seen that the worst quality of these simulations provide methods in which StR$_1$ was the least likely. The smaller the number of strata, the higher becomes the estimation

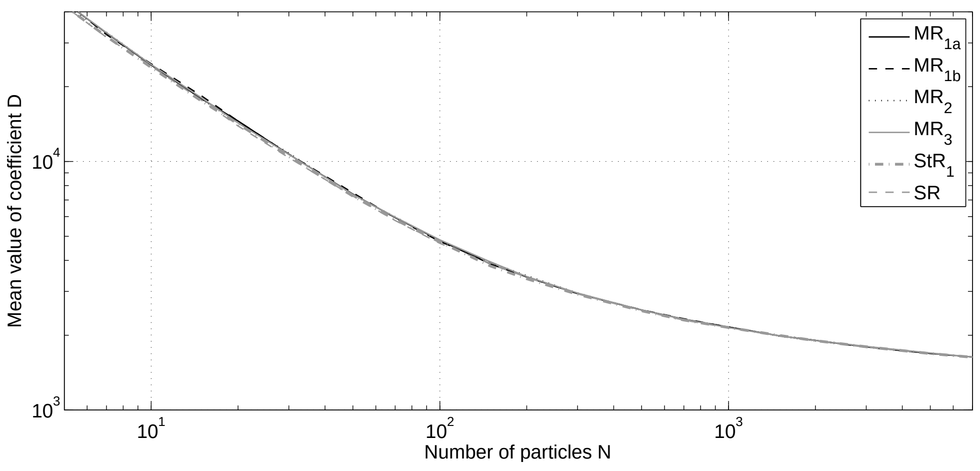

Fig. 1. Methods MR, StR_1 and SR comparison. Logarithmic scale

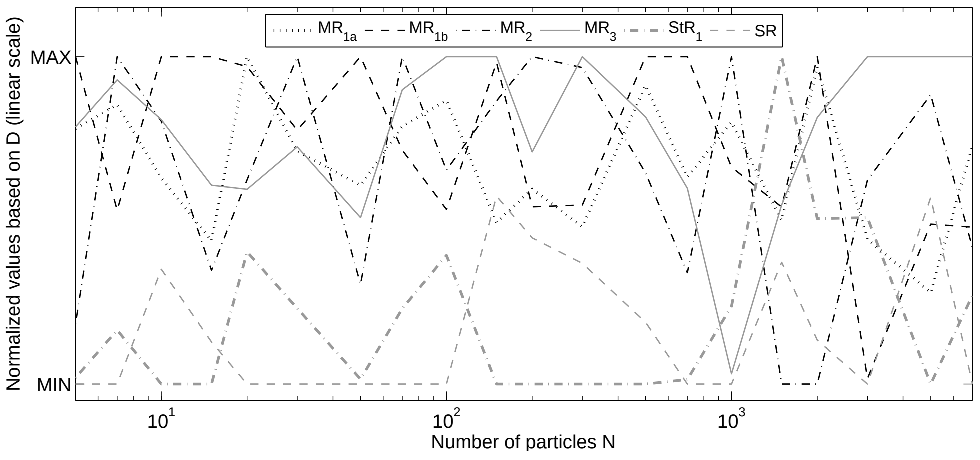

Fig. 2. Methods MR, StR_1 and SR comparison. Normalized vertical scale

quality. Method StR_{2a} has a very poor performance, is problematic to implement (2 different resampling methods) and should not be used.

In Figure 5 and Figure 6 a comparison of obtained results for methods StR_{2b} and StR_{2c} has been presented (with 3, 5 and 10 strata).

In the logarithmic scale one can see very poor results for the StR_{2b} method, and therefore these graphs are not shown in Figure 6. For StR_{2c} method it can be seen that the results are noticeably superior to the StR_1 method for small N values, regardless of the number of strata.

Figures 7 and 8 presents obtained results for methods StR_{2d} and StR_{2e}.

The obtained results confirm the earlier conclusion, i.e. for a small number of particles, division into strata has positive effect, the better, the higher number of L_p.

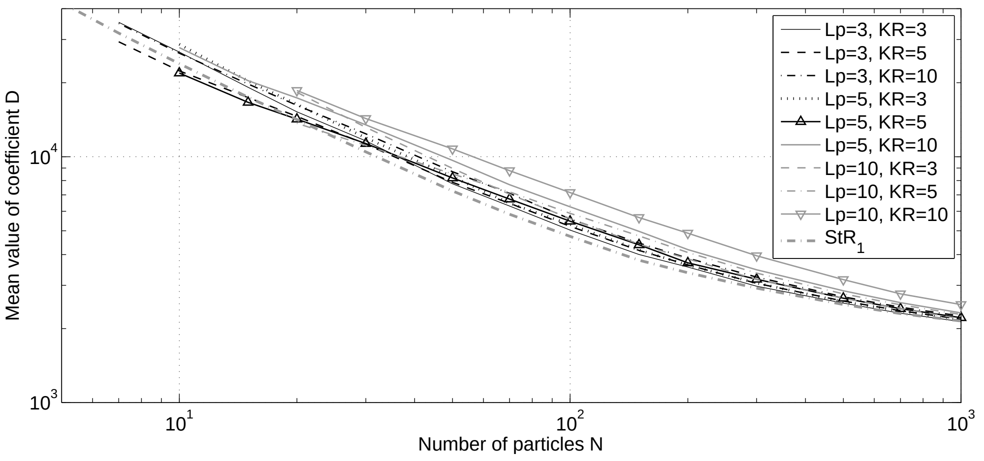

Fig. 3. Comparison for $\mathrm{StR}_{2\mathrm{a}}$ method with different values of L_p (number of strata) and K_R (method StR_1 is activated each K_R iteration). Logarithmic scale

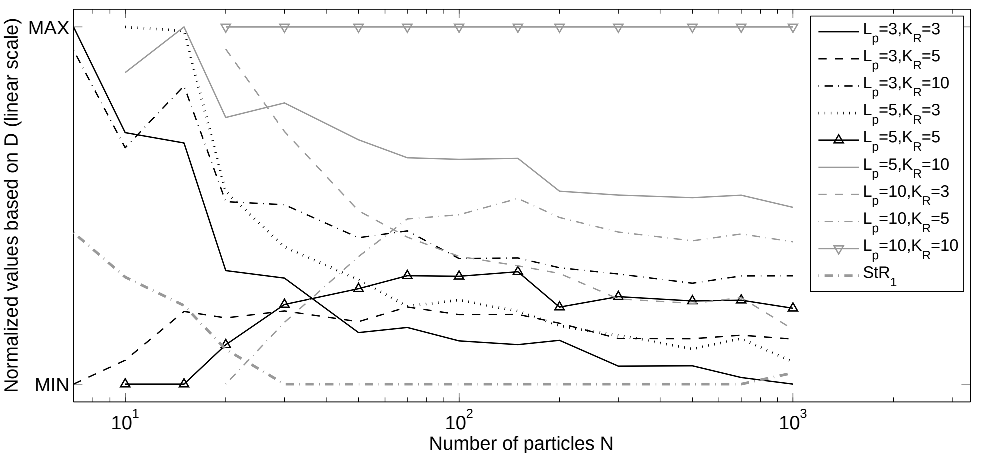

Fig. 4. Comparison for $\mathrm{StR}_{2\mathrm{a}}$ method with different values of L_p (number of strata) and K_R (method StR_1 is activated each K_R iteration). Normalized vertical scale

In Figure 9 additional comparison has been shown – 3 best of StR methods (with division into 10 strata).

Based on results, one can say that the $\mathrm{StR}_{2\mathrm{e}}$ method is the best choice, if someone wishes to implement the algorithm in parallel computing systems.

In Figures 10–11 results obtained for methods SR (Systematic resampling) and DSR (Deterministic Systematic resampling) have been presented.

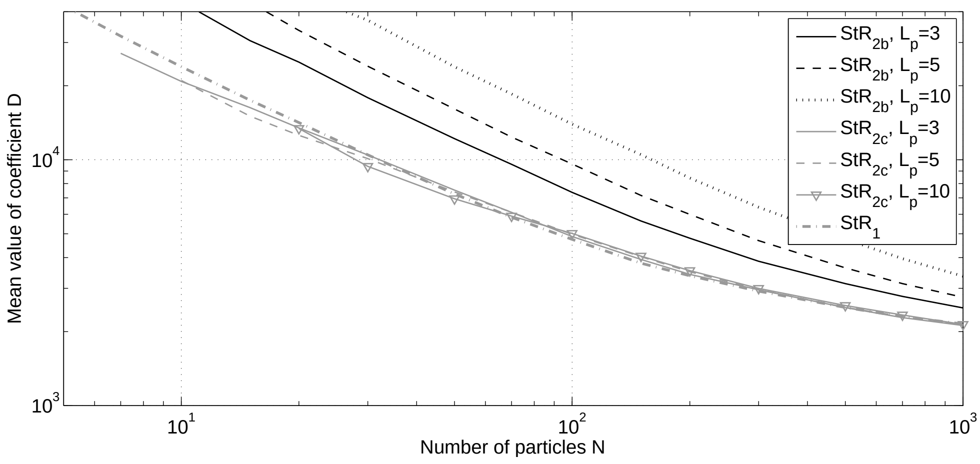

Fig. 5. Comparison of methods StR_{2b} and StR_{2c} with different number of strata (L_p). Logarithmic scale

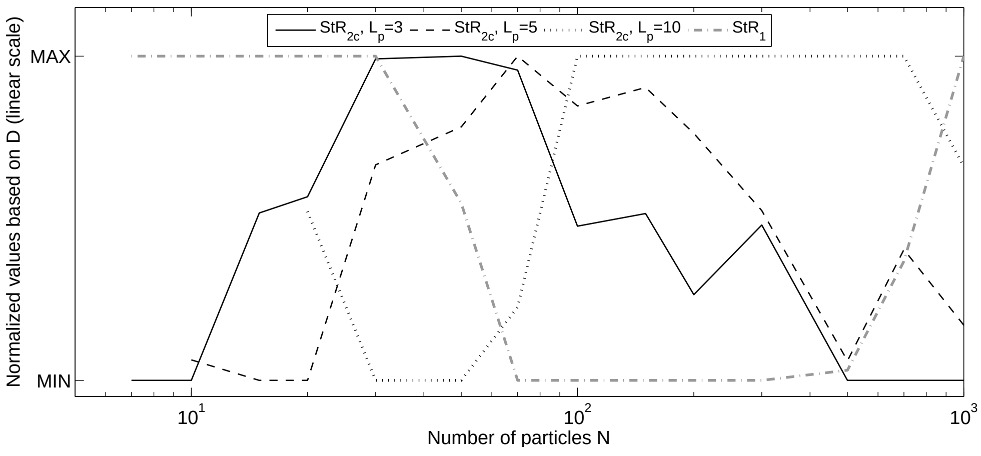

Fig. 6. Comparison of methods StR_{2b} and StR_{2c} with different number of strata (L_p). Normalized vertical scale

Based on obtained results one can see that these methods are similar in terms of quality. There is therefore no difference whether one perform the N draws (StR_1), 1 draw (SR), or none (DSR).

In Figure 12 and in Figure 13 a comparison for RR methods (Residual resampling) has been shown.

Worse quality of the proposed method RR_2 is immediately visible. Among the remaining methods, the best results have been obtained with RSR.

In Figures 14–16 results obtained for comparisons based methods have been presented, i.e. for MeR (Metropolis resampling) and ReR (Rejection resampling).

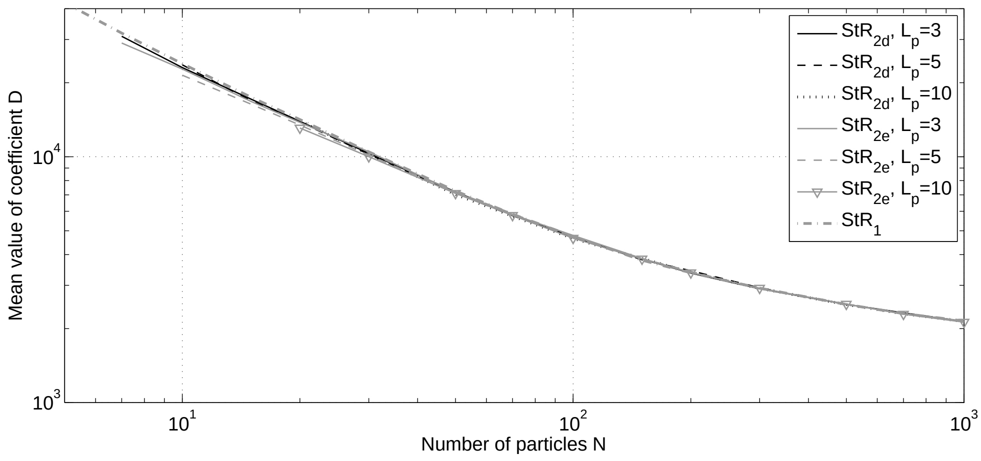

Fig. 7. Comparison of methods StR_{2d} and StR_{2e} with different number of strata (L_p). Logarithmic scale

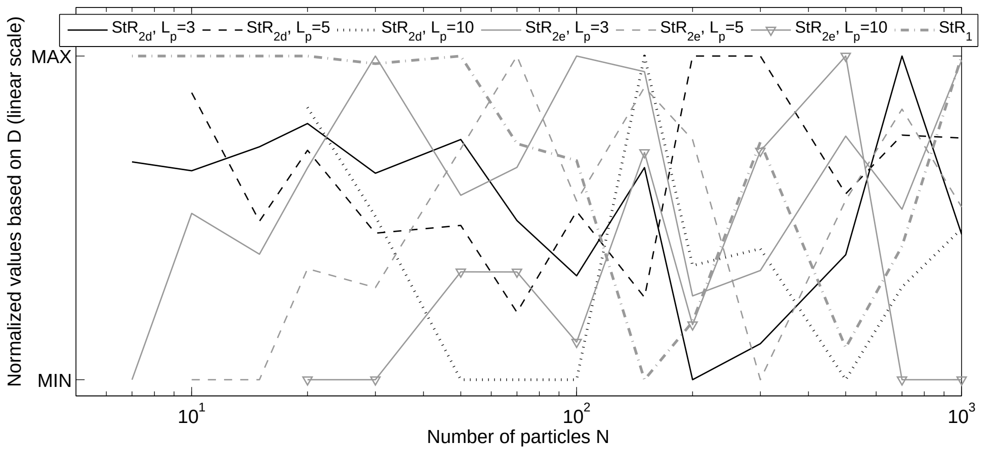

Fig. 8. Comparison of methods StR_{2d} and StR_{2e} with different number of strata (L_p). Normalized vertical scale

Already on a logarithmic scale one can see very poor quality of MeR method. Therefore, another graph has been presented after rejection of two worst results – see Figure 16.

For ReR_2 the smaller γ coefficient, the better the estimation quality. However, it should be noted that the case $\gamma = 1$ means that particle with weight $\frac{1}{N}$ has probability of being drawn equal to $\frac{1}{N}$. It means that average of N draws are needed to select one particle for replication. It is a large number, and in addition the results are not satisfactory.

In Figures 17–19 results for last two resampling methods (PSR and PDR) have been presented.

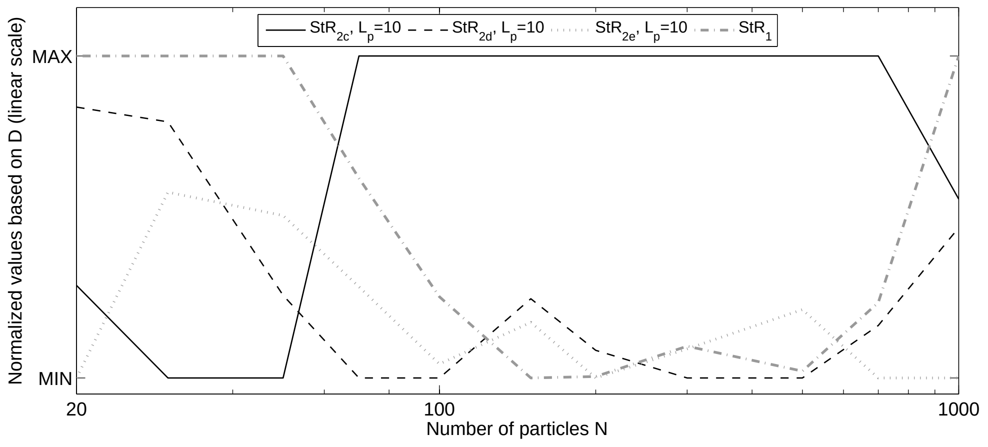

Fig. 9. Comparison of methods StR_{2c}, StR_{2d} and StR_{2e} with number of strata equal to 10. Normalized vertical scale

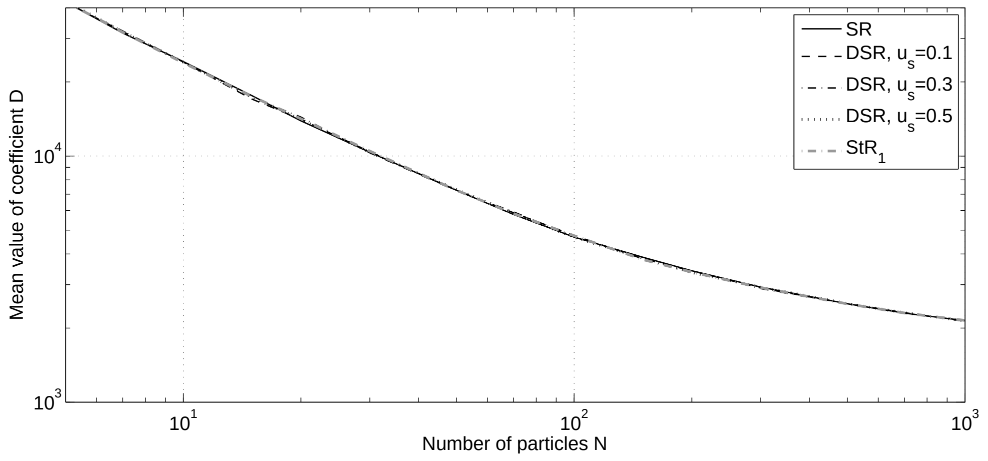

Fig. 10. Comparison of methods SR and DSR. Logarithmic scale

Estimation quality with resampling PDR is very poor. The comparison has been repeated only for PDR method – see Figure 19.

PSR method for the most closely related thresholds gives satisfactory results. This is due to the fact that for this case the method modification have least influence.

In Figure 20 and Figure 21 the last results have been shown – the best of StR methods (with division into 3 strata) and PSR method (it also has division into 3 layers).

Among presented methods the StR_{2d} seems to be the best, whereas StR_{2e} is slightly worse (with arbitrary tolerance).

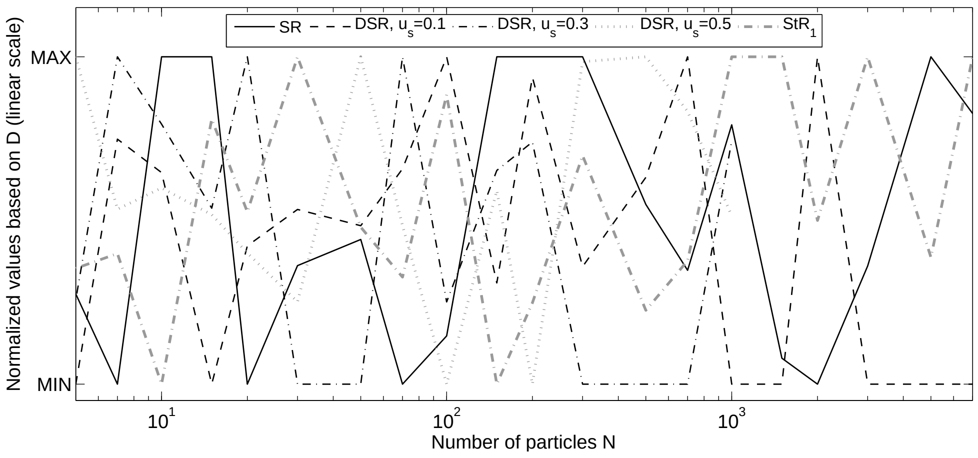

Fig. 11. Comparison of methods SR and DSR. Normalized vertical scale

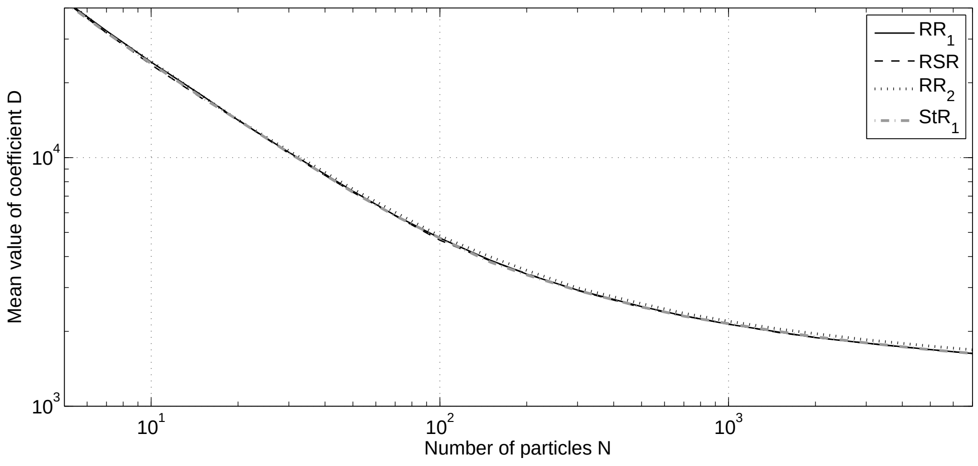

Fig. 12. Comparison of methods RR and RSR. Logarithmic scale

5. Conclusions

The article presents over 20 different types and variants of resampling methods. For each variant a code has been added and a series of simulations have been performed. Thanks to these simulations one can immediately discard certain methods, such as StR_{2a}, StR_{2b} or PDR.

Among the resampling methods that deserve attention are definitely SR, DSR and StR, which has been determined based on the results in Figures 10–11. Therefore, this means that a deterministic algorithm can be used for systems without built-in random number generator, with no loss of resampling quality.

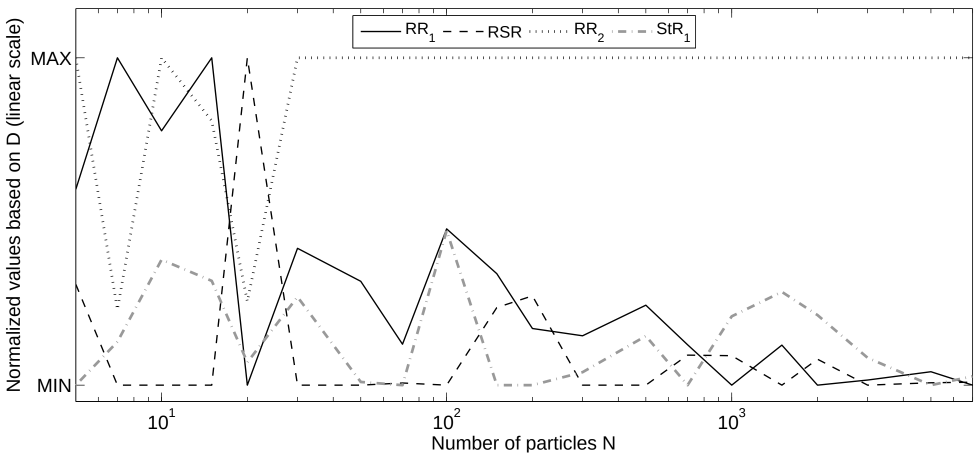

Fig. 13. Comparison of methods RR and RSR. Normalized vertical scale

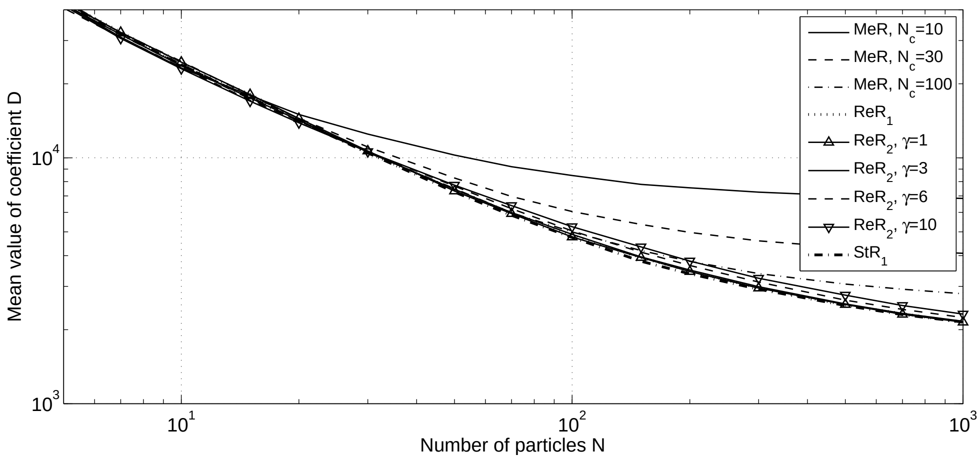

Fig. 14. Comparison of methods MeR (with different number of comparisons for one particle N_c), ReR$_1$ and ReR$_2$ (with different values of scale coefficient γ). Logarithmic scale

Among the methods which assume distribution of particles into strata, the most interesting with respect to the results, are StR$_{2\mathrm{d}}$ and StR$_{2\mathrm{e}}$. They allow for a parallel computing without loss of estimation quality. Among the methods based on the division into layers some relationship has been observed, i.e. noticeably better estimation quality for small values of N, as compared to the method without division.

In authors opinion, methods that are based on comparisons (MeR and ReR) computational requirements are too high, and the obtained results are inferior.

Further work will be related to the effect of the object dimension to the resampling operation.

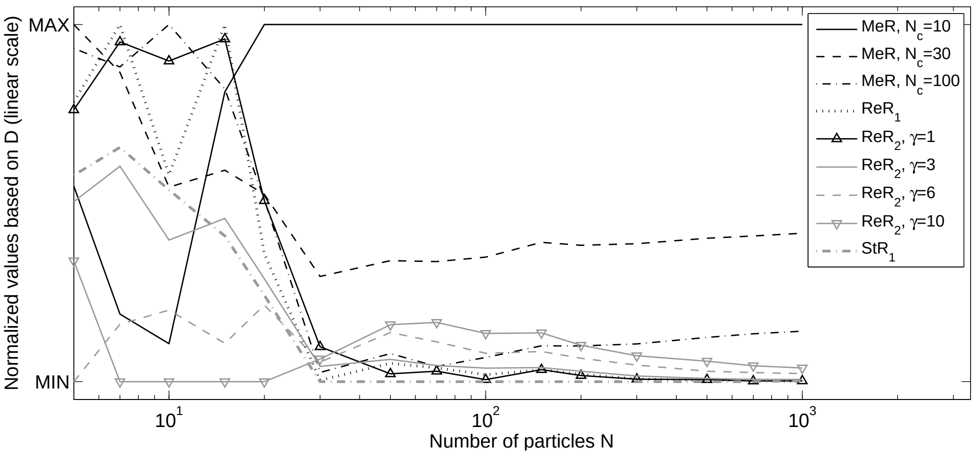

Fig. 15. Comparison of methods MeR (with different number of comparisons for one particle N_c), ReR$_1$ and ReR$_2$ (with different values of scale coefficient γ). Normalized vertical scale

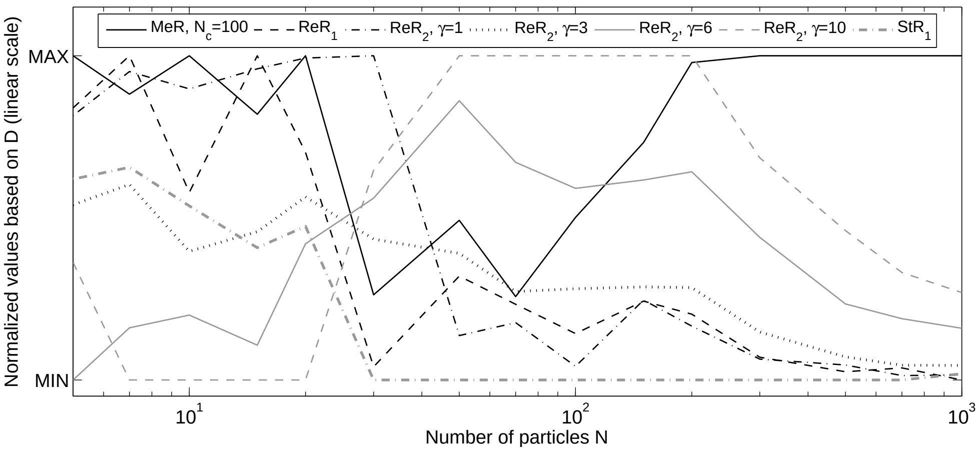

Fig. 16. Comparison of methods MeR (only for number of comparisons $N_c = 100$), ReR$_1$ and ReR$_2$ (with different values of scale coefficient γ). Normalized vertical scale

REFERENCES

[1] Arulampalam S., Maskell S., Gordon N., Clapp T., *A Tutorial on Particle Filters for On-line Non-linear/Non-Gaussian Bayesian Tracking*, IEEE Proceedings on Signal Processing, Vol. 50, No. 2, 2002, pp. 174–188.

[2] Bolic M., Djuric P. M., Hong S., New Resampling Algorithms for Particle Filters, *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, Vol. 2, April 2003, pp. 589–592.

[3] Candy J. V., *Bayesian signal processing*, WILEY, New Jersey 2009, pp. 237–298.

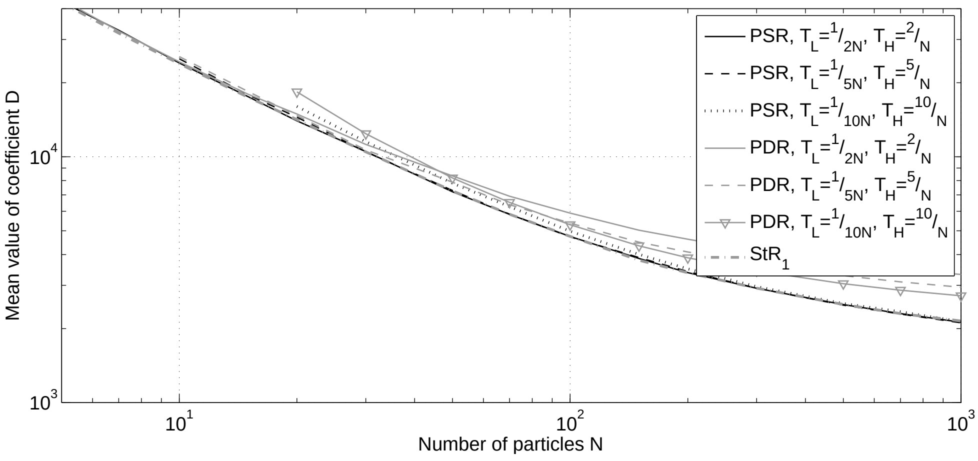

Fig. 17. Comparison of methods PSR and PDR with different values of thresholds (T_H and T_L). Logarithmic scale

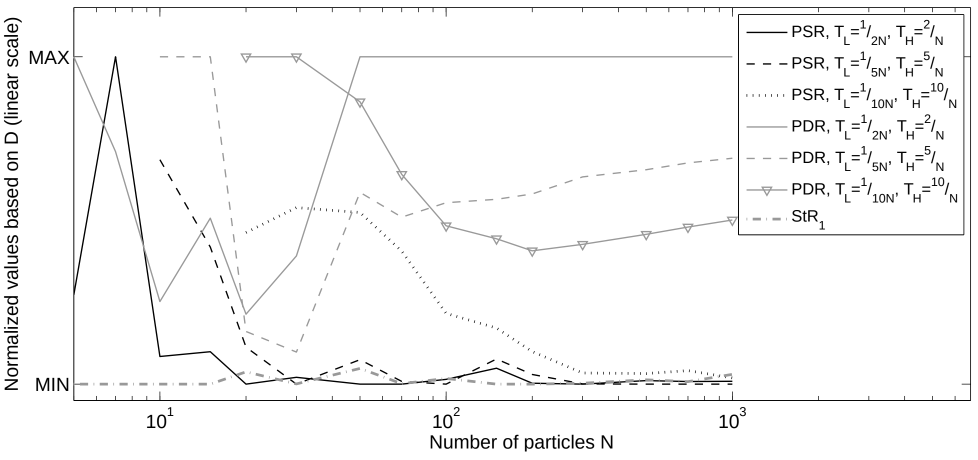

Fig. 18. Comparison of methods PSR and PDR with different values of thresholds (T_H and T_L). Normalized vertical scale

[4] Carpenter J., Clifford P., Fearnhead P., *Improved particle filter for nonlinear problems*, IEE Proceedings – Radar, Sonar and Navigation, Vol. 146, No. 1, 2/1999, pp. 2–7.

[5] Douc R., Cappe O., Moulines E., Comparison of Resampling Schemes for Particle Filtering, *Proceedings of the 4th International Symposium on Image and Signal Processing and Analysis*, September 2005, pp. 64–69.

[6] Gordon N. J., Salmond D. J., Smith A. F. M., *Novel approach to nonlinear/non-Gaussian Bayesian state estimation*, IEE Proceedings-F, Vol. 140, No. 2, 1993, pp. 107–113.

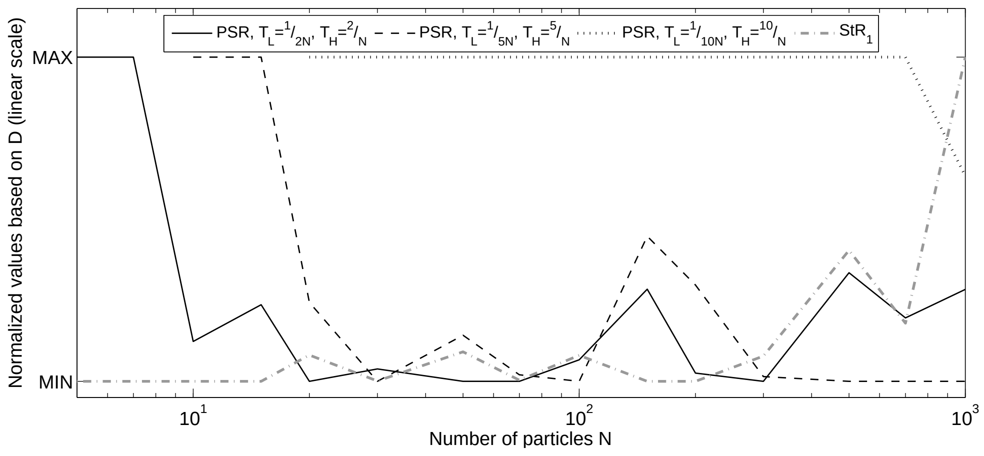

Fig. 19. Comparison of method PSR with different values of thresholds (T_H and T_L). Normalized vertical scale

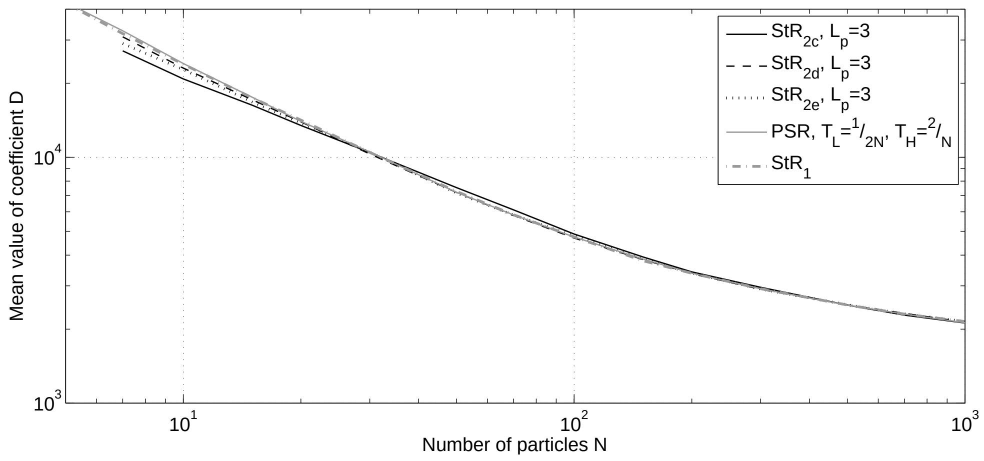

Fig. 20. Comparison of the best methods StR (with division into 3 strata) and PSR method for selected thresholds. Logarithmic scale

[7] Handeby G., Karlsson R., Gustafsson F., *The Rao-Blackwellized Particle Filter: A Filter Bank Implementation*, EURASIP Journal on Advances in Signal Processing, 2010, Article ID 724087, p. 10.

[8] Heine K., Unified Framework for Sampling/Importance Resampling Algorithms, *Proceedings of the 8th International Conference on Information Fusion*, July 2005.

[9] Hol J. D., *Resampling in Particle Filters*, report, Linkoping, 2004.

[10] Kitagawa G., *Monte Carlo Filter and Smoother for Non-Gaussian Nonlinear State Space Models*, Journal of computational and graphical statistics, Vol. 5, No. 1, 1996, pp. 1–25.

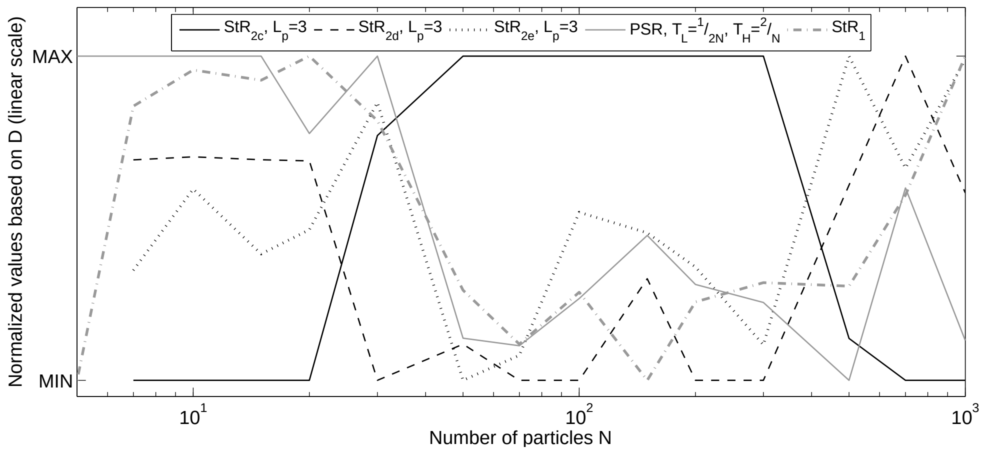

Fig. 21. Comparison of the best methods StR (with division into 3 strata) and PSR method for selected thresholds. Normalized vertical scale

[11] Klaas M., De Freitas N., Doucet A., *Toward Practical N2 Monte Carlo: The Marginal Particle Filter*, arXiv preprint, arXiv:1207.1396, 2012.

[12] Kozierski P., Lis M., *Filtr cząsteczkowy w problemie śledzenia – wprowadzenie*, Studia z Automatyki i Informatyki, Vol. 37, 2012, pp. 79–94.

[13] Lee D. U., Luk W., Villasenor J. D., Zhang G., Leong P. H. W., *A Hardware Gaussian Noise Generator Using the Wallace Method*, Very Large Scale Integration (VLSI) Systems, IEEE Transactions on, Vol. 13, No. 8, 2005, pp. 911–920.

[14] Murray L., *GPU Acceleration of the Particle Filter: the Metropolis Resampler*, arXiv preprint arXiv:1202.6163, 2012.

[15] Murray L., Lee A., Jacob P., *Rethinking resampling in the particle filter on graphics processing units*, arXiv preprint, arXiv:1301.4019, 2013.

[16] Simon D., *Optimal State Estimation*, WILEY-INTERSCIENCE, New Jersey 2006, pp. 461–484.

[17] Soto A., Self Adaptive Particle Filter, *Proceedings of the IJCAI*, July 2005, pp. 1398–1406.

[18] Straka O., Simandl M., *Particle Filter with Adaptive Sample Size*, Kybernetika, Vol. 47, No. 3, 2011, pp. 385–400.

[19] Thrun S., Burgard W., Fox D., *Probabilistic robotics*, MIT Press, Cambridge, MA, 2005, pp. 67–90.

ABSTRACT

The article presents over 20 different types and variants of resampling methods. Pseudo-code has been added for a description of each method. Comparison of methods has been performed using simulations (1,000 repetitions for each set of parameters). Based on the simulation results, it has been verified that among the methods for one processor implementation, the best working methods are those of Systematic resampling, one version of Stratified resampling and Deterministic Systematic resampling. The latter method does not require drawing numbers with uniform distribution. Among resampling methods for parallel computing, best quality is characterized by two variants of stratified resampling.

RESAMPLING W FILTRACJI CZĄSTECZKOWEJ – PORÓWNANIE

STRESZCZENIE

W artykule przedstawiono ponad 20 różnych rodzajów i odmian metod resamplingu. Do opisu każdej metody dodano pseudokod. Porównanie metod wykonano na podstawie przeprowadzonych symulacji (1000 powtórzeń dla każdego zbioru parametrów). Na podstawie przeprowadzonych symulacji stwierdzono, że wśród metod resamplingu przeznaczonych do implementacji na jednym procesorze, najlepiej działają Systematic resampling, jedna z odmian Stratified Resampling oraz Deterministic Systematic Resampling, przy czym ta ostatnia nie wymaga losowania liczb z rozkładu równomiernego. Wśród resamplingów przeznaczonych do obliczeń równoległych najlepszą jakością charakteryzowały się dwie odmiany Stratified resampling.