

Algorithms in faif library

Robert M. Nowak

March 2, 2012



Contents

1	DNA	1
1.1	Secondary structures	1
1.2	The Nussinov algorithm	2
2	Timeseries	3
2.1	Timeseries representation	3
2.2	Transformation	4
2.3	Discretizer	5
2.4	Prediction	5
2.5	Time series properties	7
3	Learning	7
3.1	Naive Bayes Classifier	7
3.2	Decision Tree Classifier	8
3.3	K Nearest Neighbours classifier	9
4	Utils	10
4.1	Random	10
4.2	Gauss eliminator	12
4.3	Power	12

1 DNA

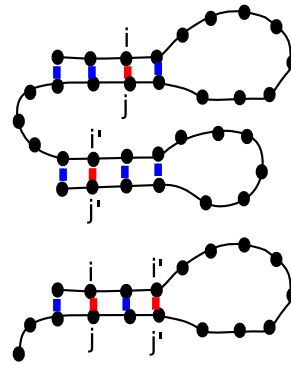
1.1 Secondary structures

The secondary structure is created mainly by hydrogen bonds (Watson-Crick). Nucleic acids secondary structures are generally divided into helices (contiguous base pairs) called stems, and various kinds of loops.

Secondary structures (DNA, RNA) of sequence $S = x_1x_2\dots x_n$ is the set of pair (i, j) , where $1 \leq i < j \leq n$, and:

- $j - i > 3$ (the hair-pin loops are longer than 3 nucleotides)

- if (i, j) and (i', j') are the pairs
 - (i, j) is before (i', j') $i < j < i' < j'$
 - or (i', j') is before (i, j) $i' < j' < i < j$
 - or (i, j) include (i', j') $i < i' < j' < j$
 - or (i', j') include (i, j) $i' < i < j < j'$



The structures when (i, j) and (i', j') are the pairs, and $i < i' < j < j'$ are called tertiary structures (e.g. pseudo-knots)

- pseudo-knots
 - kissing hair-pin loops
-

1.2 The Nussinov algorithm

This algorithm base on minimization the free energy, and it use the equation 1, where the $E(S)$ is the energy of structure as a sum of the connection energies for pairs, where the example of similarity matrix is given below.

$$E(S) = \sum_{i,j \in S} e(x_i, x_j) \quad (1)$$

$e(i,j)$	A	C	G	U
A	0	0	0	2
C	0	0	3	0
G	0	3	0	1
U	2	0	1	0

The Nussinov algorithm is the dynamic programming approach to find the maximum value of $E(S)$, and it calculates $F(i, j)$ value as:

$$F(i, j) = \max \left\{ \begin{array}{l} 0 \\ F(i+1, j-1) + e(x_i, x_j) \\ F(i+1, j) \\ F(i, j-1) \\ \max_{k:i \leq k < j} F(i, k) + F(k+1, j) \end{array} \right.$$

$j - i \leq 3$

connection

x_i unpaired
 x_j unpaired

division

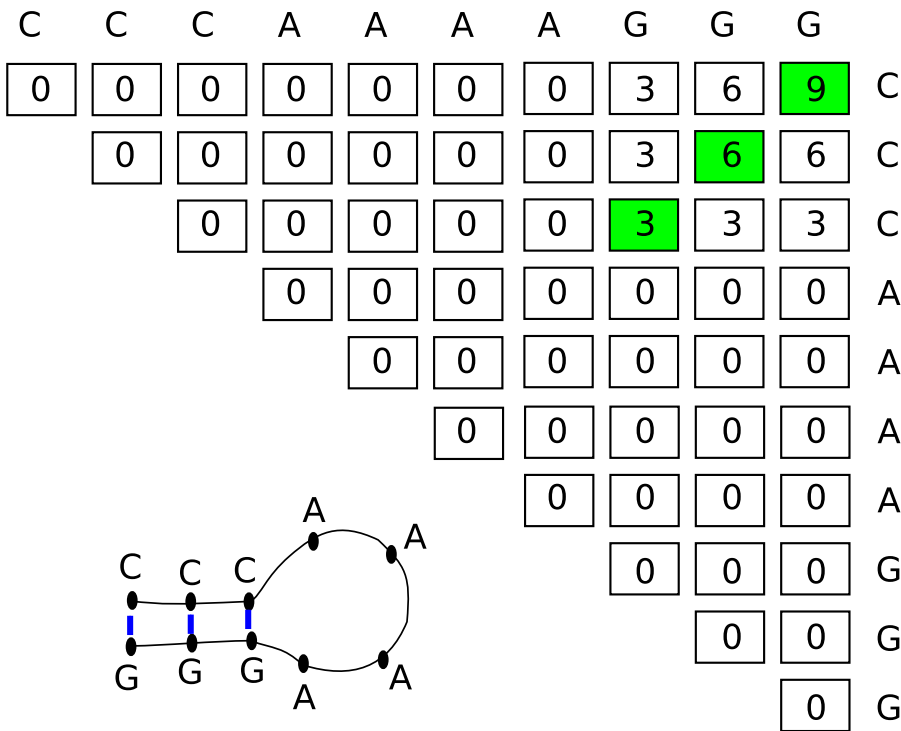


Figure 1: example the Nussinov algorithm for sequence CCCAAAAGGG

2 Timeseries

2.1 Timeseries representation

Timeseries is a collection of time-value elements. Time-value is the comprised of time-stamp, value, and quality.

Real timeseries

Such timeseries, known as real timeseries, are collections in which: the elements are sorted by timestamps (in ascending order), such that the first element has the oldest timestamp, and the last element has the newest timestamp.

There are no additional assumptions about timeseries, especially

- equal distance between contiguous timestamps is not required
- timestamp is not unique in a collection

name	type	description
timestamp	boost::posix_time::ptime	seconds since midnight January 1, 1970, posix time, 32 bit unsigned integer value
value	double	the average value (in the period of an adjacent timestamps) in <i>units</i> per second
quality	double	the average (in the period of an adjacent times-tamps), values from 0.0 to 1.0

Digit timeseries

During calculations timeseries is transformed into (so called) Digit Time Series. Digit timeseries is a collection of time-values, where

timestamp	32 bit signed integer value Interpretation:	* positive * zero (0) * negative	future timestamps present time past timestamps
value	average value (in the period of adjacent timestams)		
quality	average value (in the period of adjacent timestams), values from 0.0 to 1.0		

Digit timeseries properties

- are sorted by timestamps, the first element being the oldest, the last being the newest
- have every time-value from given range, for example if the first element has a timestamp of -3 and the last has a timestamp of 3, the collection must have 7 elements { -3, -2, -1, 0, 1, 2, 3 }
- the timestamps are unique
- timeseries must not be empty

2.2 Transformation

Create digit timeseries from real timeseries. Required parameters are:

present time	posix time	calculation the zero for digit timestamp
delta (resampling rate)	number of seconds	distance between continuous timestamps

The algorithm use linear approximation of the two closest elements in real time series, as showed in fig. 2

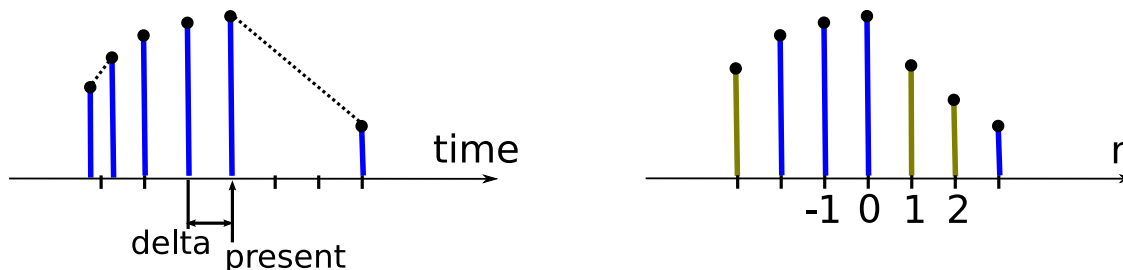


Figure 2: Transform real time series into digit time series by linear approximation algorithm. The values in function of time are presented

Example

when the real timeseries is (qualities ignored):

- $\{(1:00, .3), (1:15, .4), (2:00, .7), (5:00, .4), (6:00, .9)\}$,

and parameters are:

- present: 3:00
- delta 1:00

the digit timeseries is

- $\{(-2, .3), (-1, .7), (0, .6), (1, .5), (2, .4), (3, .9)\}$

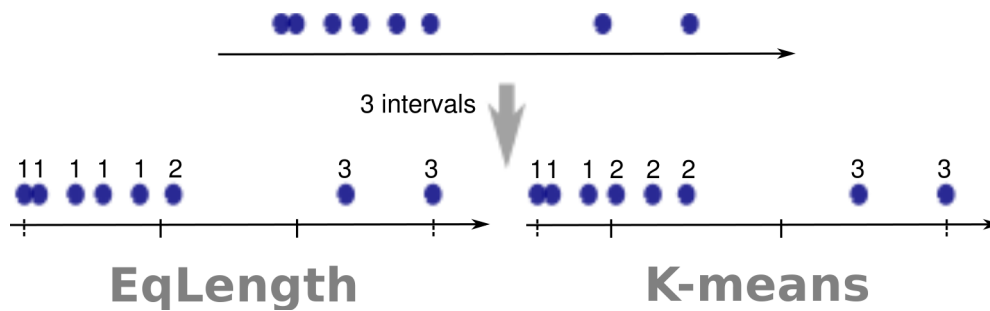
2.3 Discretizer

The converter from ratio values (eg real numbers) to ordered or interval values (eg integer numbers). This transformation is used when the algorithms require nominal or ordered values (eg classifier with only equality tests). The discretizer is a collection of disjoint intervals. Discretizer return the index of interval containing given value.

The following methods to calculate intervals are available:

- partition of K equal length intervals
- partition of K intervals, based on K-means algorithm

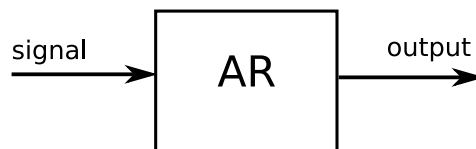
The difference is shown on picture.



2.4 Prediction

The regression-based and memory-based models are provided for prediction calculation. Regression-based are: auto-regressive (AR), auto-regressive moving average (ARMAX) Memory-based are: reference day (KNN) and reference days with inputs (KNNX).

Auto-regressive model (AR)



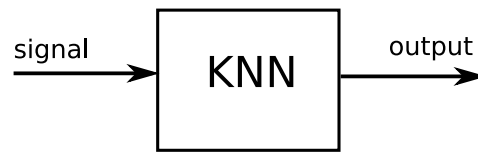
Auto-regressive model AR of order p called AR(p) calculates the prediction based on the history of the signal. The S is the signal, S_t the value in time t , $\alpha_1 \dots \alpha_p$ are parameters of the model.

$$S_t = \sum_{i=1}^p \alpha_i S_{t-i}$$

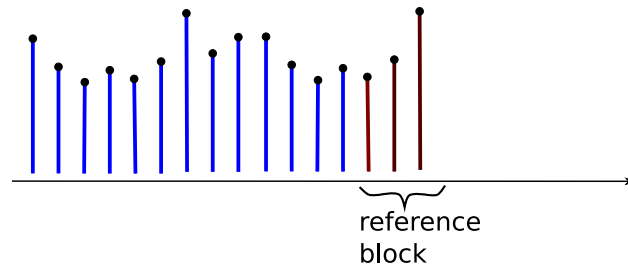
Examples:

- AR(2), $\alpha_1 = 1.0, \alpha_2 = 1.0$, signal = { 1.0, 1.0 }, predictions = { 2, 3, 5, 8, 13, 21, 34, ... };
- AR(4), $\alpha_1 = 0.25, \alpha_2 = 0.25, \alpha_3 = 0.25, \alpha_4 = 0.25$ for input { 1, 2, 3, 4 } gives predictions { 2.5, 2.875, 3.09375, ... }

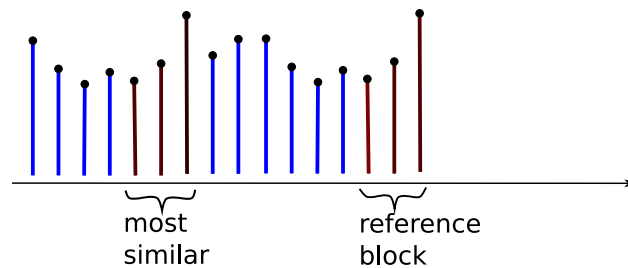
Memory-based model (KNN)



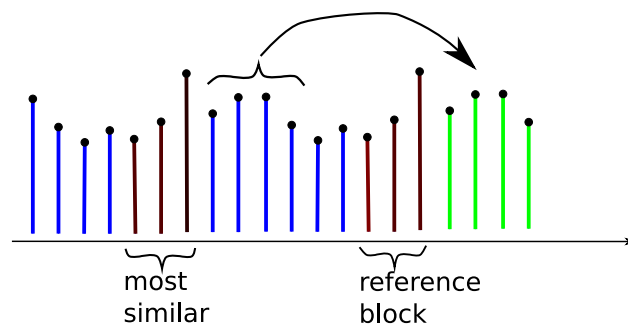
The KNN algorithm finds the closest (to the reference) sequence in past and then calculates the prediction. For example, when reference block (of length 3, timestamps {14,15,16}) is given in the figure below



the algorithm finds the nearest (most similar) sequence {5,6,7}



then the predictions are the time-values that follow the most similar sequence



Parameters of model:

name	description
reference block	the least time-values
reference block length	the length of reference block
prediction length	number of values in prediction
number of neighbours	in the given example only one closest block was found. Generally, the n closest could be considered, and the prediction is the average of predictions

2.5 Time series properties

Average

Correlation

Correlation, formally cross-correlation, finds repetitions in timeseries patterns, and is a measure of similarity between two signals. Autocorrelation is cross-correlation of a signal with itself.

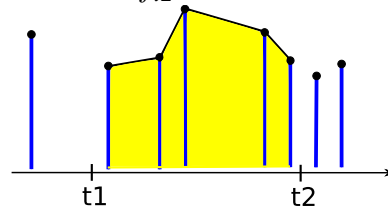
Definition:

$$(f \star g)[n] = \sum_j f[j]g[n+j]$$

Integral

In presented library the integration is calculated for real timeseries using the trapezoid rule for numerical integration, equation below, where f is the real timeseries, the f_i is the i -th element of timeseries f , $f_i.v$ is the value of i -th element, the $f_i.t$ is the timestamp of i -th element, the $i1$ is the element where $f_{i1}.t$ is near to $t1$, and the $i2$ is the element where $f_{i2}.t$ is near to $t2$.

$$P = \int_{t1}^{t2} f dt = \sum_{i=i1}^{i2} \frac{f_i.v + f_{i+1}.v}{2} (f_{i+1}.t - f_i.t)$$



3 Learning

3.1 Naive Bayes Classifier

Naive Bayes Classifiers calculates probability of the given categories for piece of input data. It uses the Bayes Rule, presented in equation 2.

$$P(c_i|x_1, x_2, \dots, x_n) = \frac{P(x_1, x_2, \dots, x_n|c_i)P(c_i)}{\sum_{c \in C} P(x_1, x_2, \dots, x_n|c)P(c)}, \quad (2)$$

where the $C = \{c_1, c_2, \dots, c_m\}$ is a set of categories; $x_1 \in X_1, x_2 \in X_2, \dots, x_n \in X_n$ are the known attribute values. The Naive Bayes Classifiers uses the assumption of attribute independence, therefore:

$$P(c_i|x_1, x_2, \dots, x_n) = \frac{P(x_1|c_i)P(x_2|c_i)\dots P(x_n|c_i)P(c_i)}{\sum_{c \in C} P(x_1, x_2, \dots, x_n|c)P(c)}.$$

In most cases the Naive Bayes Classifier is used to find the most probable category:

$$\arg \max_c P(c|x_1, x_2, \dots, x_n),$$

and because the denominator does not change for each $c \in C$, it could be omitted:

$$\arg \max_c P(c|x_1, x_2, \dots, x_n) = \arg \max_c P(x_1|c_1)P(x_2|c_1)\dots P(x_n|c_1)P(c_1).$$

The example calculation performed to obtain the most probable category, as well as the probability of all categories are showed below.

Lets say we have data with the information about people education level, age, gender and their answer for some question (yes/no):

education level	age	gender	answer
primary	young	female	yes
primary	young	male	no
primary	middle	male	no
primary	old	female	no
primary	old	male	no
college	young	female	yes
college	young	male	yes
college	middle	female	no
college	middle	male	no
college	old	male	no
university	young	male	yes
university	middle	male	yes
university	old	female	yes
university	old	male	yes

So here we have 7 yes answers, and 7 no answers, thus $P(\text{yes}) = 0.5$, $P(\text{no}) = 0.5$. For all 3 attributes the summaries are created, showing the conditional dependence of the answer given selected attribute:

education		
	answer = yes	answer = no
primary	1/7	4/7
college	2/7	3/7
university	4/7	0/7

Additionally $P(\text{primary}) = 5/14$, $P(\text{college}) = 5/14$, $P(\text{university}) = 4/14$ (this will speed-up computations). Similar tables are created for age and gender:

age		
	answer = yes	answer = no
young	4/7	1/7
middle	1/7	3/7
old	2/7	3/7

$P(\text{young}) = 5/14$, $P(\text{middle}) = 4/14$, $P(\text{old}) = 5/14$,

gender		
	answer = yes	answer = no
female	2/7	3/7
male	5/7	4/7

$P(\text{female}) = 5/14$, $P(\text{male}) = 9/14$.

The question is: what is the most probable answer for university educated female in middle age?

$$\arg \max_{c \in \{\text{yes}, \text{no}\}} P(\text{university, middle, female} \mid c) \Rightarrow \max \left\{ \frac{8}{686}, \frac{9}{686} \right\} \Rightarrow \text{no}$$

3.2 Decision Tree Classifier

Decision Tree Classifier, ID3 - Iterate Dichotomizer inspired algorithm.

Decision tree classifier uses tree structure, depicted in Fig. 3. Each node stores the probability for each category for objects that lead to it, so the root node describes all objects and is being the least precise, the leaf node are the most precise and describes very similar objects. In presented approach internal nodes stores the binary equality test, this gives the value of expression $\exists v \in E : v = v_i$, where E is set of feature values connected with given object (E is example), v_i is a test feature value.

Decision tree structure is travelled from root to leaf using Alg. 1, where next node is chosen based on test results for given object (i.e. set of feature values connected with object E). Algorithm

assumes false result of test (depicted in alg. 1) for missing attribute value.

Algorithm 1 Decision tree classification

```

procedure CLASSIFY((D, e)) ▷ decision tree D, example e
  v ← root(T)
  while ¬v.isLeaf() do
    if v.test(e) then
      v ← v.left
    else
      v ← v.right
    end if
  end while
  return v.cat
end procedure

```

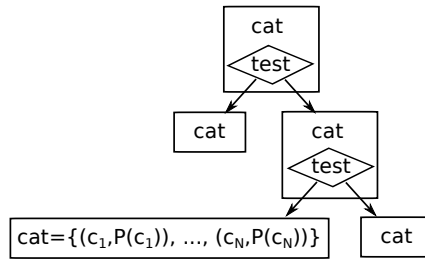


Figure 3: Decision tree data structure; node consists collection of probabilities for each category $c \in C$ (called *cat*) for objects lead to it, internal nodes have test.

The training of Decision Tree is depicted in alg. 2. It is recurrence procedure building the tree, where the decision if create a leaf or inner node is taken, based on properties of examples from training examples set supporting given node: if the number of examples is small or there exist only few examples with not major category the leaf is created, otherwise the inner node. The inner node creation involves best test searching, which is the test giving the best entropy gain, as described in eq. 3, where g is entropy gain for test t and example set E , C is set of categories. The examples are split using test results and left end right sub-node are created recursively.

$$g = \frac{en(E_p) + en(E_n) - en(E)}{\frac{|E_p|}{|E|} \log\left(\frac{|E_p|}{|E|}\right) + \frac{|E_n|}{|E|} \log\left(\frac{|E_n|}{|E|}\right)}$$

where

$$E_p = \{e \in E : t(e) = \text{true}\}, E_n = \{e \in E : t(e) = \text{false}\}$$

$$en(E) = \sum_{c \in C} \frac{|\{e \in E : e_c = c\}|}{|E|} \log\left(\frac{|\{e \in E : e_c = c\}|}{|E|}\right)$$
(3)

3.3 K Nearest Neighbours classifier

Memory based classifier, stores training examples.

Algorithm 2 Building decision tree using set of testing examples (training)

```
procedure BUILDTREE( $E$ ) ▷ set of training examples  
   $T \leftarrow \text{allTests}(E)$   
   $root \leftarrow \text{BuildNode}(T, E)$  return  $root$   
end procedure  
procedure BUILDNODE( $T, E$ )  
   $\hat{c} \leftarrow \arg \max_{c \in C} |\{e \in E : e_c = c\}|$   
  if  $|E| < E_m \vee \forall c \in C: c \neq \hat{c} |\{e \in E : e_c = c\}| < E_m$  then  
    return LeafNode( $E$ )  
  end if  
   $t \leftarrow \text{bestTest}(T, E)$   
   $E_p \leftarrow \{e \in E : t(e) = \text{True}\}$   
   $E_f \leftarrow \{e \in E : t(e) = \text{False}\}$   
   $v \leftarrow \text{InternalNode}(t, E)$   
   $v.\text{left} = \text{BuildNode}(T \setminus \{t\}, E_p)$   
   $v.\text{right} = \text{BuildNode}(T \setminus \{t\}, E_f)$   
  return  $v$   
end procedure
```

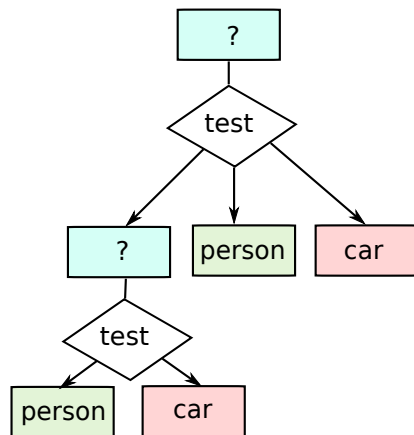


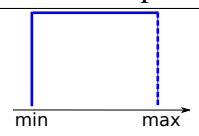
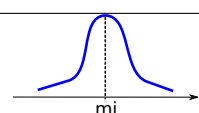
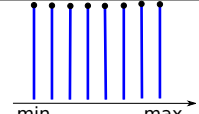
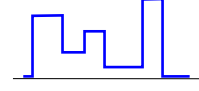
Figure 4: Example of decision-tree object

4 Utils

4.1 Random

Library provides a number of random generators. Each random generator uses the singleton with `boost::mt19937` (access to boost random generator is synchronized).

The following generators are available:

name	description	example
RandomDouble	<p>the uniform continuous distribution, parameters:</p> <ul style="list-style-type: none"> • $\mu(\text{mean}) = \frac{1}{2}(x_{min} + x_{max})$, • $\sigma(\text{standard deviation}) = \frac{1}{2\sqrt{3}}(x_{max} - x_{min})$ 	
RandomNormal	<p>the continuous normal distribution, parameters: mean (mi, μ), and standard deviation (sigma, σ)</p>	
RandomInt	<p>the uniform discrete distribution, in range <min,max> (parameters: min, max)</p>	
RandomCustomDistr	<p>this distribution is represented by segments, set of n disjoint uniform distributions (segments). The probability density function $h(x)$ is:</p> $h(x) = \begin{cases} p^1 & \text{for } x_{min}^1 \leq x < x_{max}^1 \\ p^2 & \text{for } x_{min}^2 \leq x < x_{max}^2 \\ \dots & \\ p^n & \text{for } x_{min}^n \leq x < x_{max}^n \\ 0 & \text{for } x < x_{min}^1 \text{ or } x \geq x_{max}^n \end{cases}$ <p>The p^i factors are normalized:</p> $\int_{-\infty}^{+\infty} h(x) dx = \sum_{i=1}^n p^i (x_{max}^i - x_{min}^i) = 1$ <p>Parameters of this distribution:</p> <ul style="list-style-type: none"> • $\mu = \frac{1}{2} \sum_{i=1}^n p^i (x_{min}^i + x_{max}^i)$ • $\sigma = \frac{1}{\sqrt{3}} \sqrt{\sum_{i=1}^n p^i ((x_{max}^i - \mu)^3 - (x_{min}^i - \mu)^3)}$ 	

The library could be used to Monte Carlo simulations which are parametrized by the accuracy i.e. number of steps when Monte Carlo is performed. The single step includes the calculation of value of each random variable (drawing) and the calculation of value of expression. Result of Monte Carlo could be the RandomCustomDistr, where the length of each segments (ϵ) is the same, and approximated by equation 4.

$$\epsilon = \frac{\sigma}{\sqrt{n} * \mathcal{N}_{0,1}(3)} \tag{4}$$

where:

- $\mathcal{N}_{0,1}$ is the pdf of normal distribution
- n is the number of steps in Monte Carlo
- σ is standard deviation of result distribution.

The standard deviation of result distribution for expression v is approximated by (5), given the approximated parameters (μ and σ) of sub-expression.

$$\sigma^2 = \begin{cases} \sigma_a^2 + \sigma_b^2 & \text{where } v = v_a + v_b \\ \mu_a^2 * \sigma_b^2 + \mu_b^2 * \sigma_a^2 + \sigma_a * \sigma_b & \text{where } v = v_a * v_b \end{cases} \tag{5}$$

The expression to calculate the mean and standard deviation the expressions from Tab. 1 are useful.

name	parameters	μ	σ
degenerate distribution	x_0	x_0	0
RandomNormal	μ, σ	μ	σ
RandomDouble	x_{min}, x_{max}	$\frac{1}{2}(x_{min} + x_{max})$	$\frac{1}{2\sqrt{3}}(x_{max} - x_{min})$
RandomCustomDistr	$p^1, p^2, \dots, p^n,$ $x_{min}^1, x_{min}^2, \dots, x_{min}^n,$ $x_{max}^1, x_{max}^2, \dots, x_{max}^n$	$\frac{1}{2} \sum_{i=1}^n p^i (x_{min}^i + x_{max}^i)$	$\frac{1}{\sqrt{3}} \sqrt{\sum_{i=1}^n p^i ((x_{max}^i - \mu)^3 - (x_{min}^i - \mu)^3)}$

Table 1: The mean and standard deviation for random generators

4.2 Gauss eliminator

4.3 Power

The integral power n of real number x could be calculated by expression

$$x^n = \underbrace{x * x * \dots * x}_n \quad (6)$$

If the n is big, the expression 6 is inefficient, and could be modified to use the square function. If $n = 2^m$ i.e n is power of two (2, 4, 8, 16, ...) the x^n could be calculated by expression 7, where $sqr(x)$ means square of x . For $n = 1024$ there is 10 multiplications instead of more than thousand used by expression 6.

$$x^n = \underbrace{(\dots((x^2)^2)\dots)^2}_m = \underbrace{sqr \circ sqr \circ \dots \circ sqr(x)}_m, \text{ where } n = 2^m \quad (7)$$

The x^n where n is a unsigned integer is calculated based on property described above. If the binary of $n = b_m * 2^m + b_{m-1} * 2^{m-1} + \dots + b_1 * 2 + b_0 = (b_m b_{m-1} \dots b_1 b_0)_2$, x^n is the multiplication of terms $\underbrace{sqr \circ sqr \circ \dots \circ sqr(x)}_m$, e.g. $x^{35} = x^{32+2+1} = (((x^2)^2)^2)^2 * (x^2) * x$.

The `faif/utils/Power.hpp` include the function templates to calculate the power for integer n .