# Modelling the Probability of Deadlocks in a Multithreading Process

Doina LOGOFĂTU, Felix DIETRICH, Evgeni PAVLIDIS, Dennis WILFERT
*Department of Computer Science and Mathematics*
*University of Applied Sciences*
*Lothstrasse 64, 80335, Munich*
*{doina.logofatu, felix.dietrich, evgeni.pavlidis, dennis.wilfert}@hm.edu*

***Abstract* – This paper addresses some aspects on modeling a problem in continuous probability theory. We start describing a common problem in current computer science, the deadlock. This is followed by a mathematical abstraction of the problem. Three solution models are presented for it, two of them designed for multidimensional cases. These models are then tested in experiments and compared against the exact solutions.**

***Index Terms* – Continuous Probability Theory, Geometry, Monte-Carlo Simulation, Parallelization, Threading**

## I. Introduction

Sharing is never easy. Most humans learn to share as a child, when the act of giving away a part of one's belongings results in getting back something, be it physically or emotionally. Problems with this approach arise when an object cannot be divided up and is desired by more than one party. In this case, a compromise can be found by defining disjunctive time spans where only a single party holds the object. When the time is over, it is handed on to another party. This approach directly leads to concurrency if the time spans are not set in advance [7, 9].

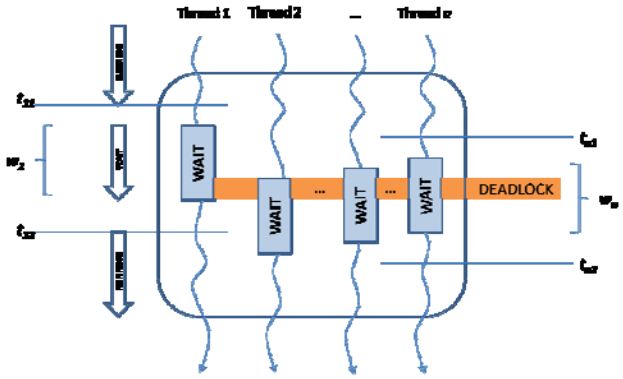



**Figure 1. Process containing several threads with the risk of a deadlock, thread *i* starts waiting for $w_i$ seconds between $t_{i1}$ and $t_{i2}$**

Concurrency in computer science means that several computations are executing simultaneously, potentially interacting with each other, and eventually sharing the same resources (memory, CPU). This often leads to a serialization of access, what implies a ranking of the participating events. In many cases it is unimportant in which order the events are executed or if they are processed simultaneously [2, 9].

Let us consider the scenario of two threads running in parallel (Fig. 2, state RUNNING) and further thread *A* waiting for thread *B* to release the lock of some object (Fig. 2, state WAIT). This waiting starts in a time span $[t_{11}, t_{12}]$ with equal probability distribution and takes some time $w_1$. Let now thread *B* also start waiting in a time span $[t_{21}, t_{22}]$ for an object thread *A* has to release. This takes the time $w_2$. It sometimes occurs that both threads are in the WAIT state at the same time. In this situation, none of them can proceed without the other, what is then called a *deadlock*. The application cannot operate any further and must at least shut down the two threads. The problem we will discuss in this paper is to minimize the probability of this event.

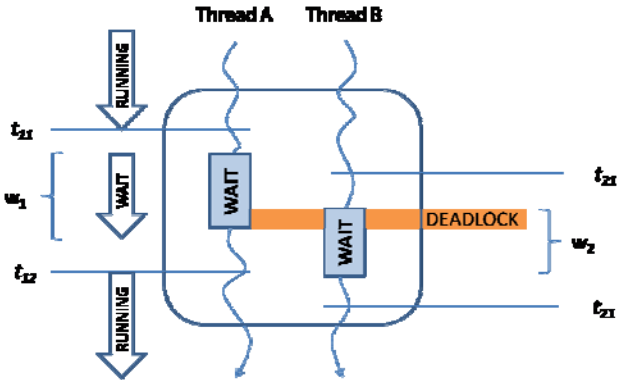



**Figure 2. Process containing two threads *A* and *B* with the risk of a deadlock**

To solve this problem, we first describe it in a more abstract, mathematical way. We start with $X_i$ being a continuous random variable with uniform distribution, representing the thread *i*. Let all $X_i$ have the following form:

$$X_i : \Omega \rightarrow [t_{i1}, t_{i2}] \tag{1}$$

with $t_{i1}$ and $t_{i2}$ being the first/last possible points in time where thread *i* might start waiting and

$$P(X_i = k) = \frac{1}{t_{i2} - t_{i1}} \forall k \in [t_{i1}; t_{i2}], k \in \mathbb{R} \quad (2)$$

Let then $w_i$ be the time thread $i$ needs to wait for another one to release the lock of an object. For simplicity, all $w_i$ are equal and constant for all $i$ (=$w$). To define the problem, we need a combination $C$ of the random variables. Let $n$ be the number of random variables and $i \in \{1, \ldots, n\}$ . Then $C$ can be defined as

$$C = \{c \in X_1, x_i \in X_i \mid \forall i: x_i - w \leq c \leq x_i + w\} \quad (3)$$

When $C \neq \emptyset$, the probability $p$ of a deadlock is nonzero and can be written as

$$p = \frac{|C|}{\left|\bigcup_i x_i\right|} \quad (4)$$

For two random variables $X_1$ and $X_2$, there is a geometric interpretation of this probability. Let $t_{11}$ and $t_{12}$ be the boundary points of the range of $X_1$ and let $t_{21}$ and $t_{22}$ the boundary points of the range of $X_2$. Then $C$ is the area of a rectangle $A$ in a two-dimensional space formed by $X_1$ and $X_2$ bordered by two lines $y_1 = x + w$ and $y_2 = x - w$ with $x \in X_1$ and $y_1, y_2 \in X_2$.

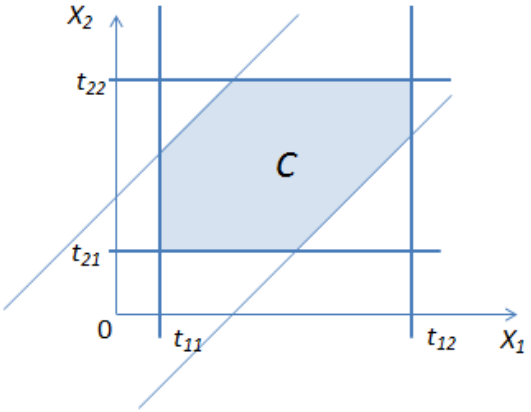



**Figure 3. *C* for two random variables**

The probability of a deadlock is

$$p = \frac{C}{A} \quad (5)$$

We will use this formula for our solutions in the next part.

## II. Solution Models

To calculate the area of $C$ in the two-dimensional case, we will now introduce three solution models – the Monte Carlo Simulation, a numerical method and the exact solution. The first two models were chosen because they can be extrapolated very easily. This will be an advantage when more than two random variables are chosen to generate $C$.

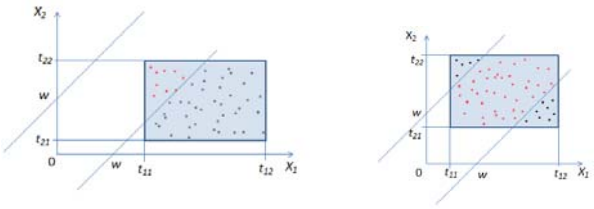
### *A. First solution model – Monte Carlo Simulation*

Monte Carlo methods rely on repeated random sampling to compute their results. The idea came from Enrico Fermi in the 1930s, when he used Monte Carlo in the calculation of neutron diffusion [13]. Monte Carlo methods are mostly used if finding an exact solution to a problem is very complex or impossible. A classic example of using the method is the approximation of π [18, 19, 20, 21]. There are many variants of Monte Carlo methods, but all of them follow a particular pattern:

1. For the originally mathematical model a stochastic model must be found which describes the problem.
2. A sequence of random numbers must be generated. These values should simulate possible real situations.
3. There must be found estimations from the random values for the original problem.

Monte Carlo methods are used in many different areas like mathematics, physics or in the financial sector [1, 4, 5, 6, 8, 12, 13]. In mathematics, they are often used for evaluating definite integrals with complicated boundary conditions. Especially for multidimensional integrals the Monte Carlo integration can be particularly efficient [14]. For example, two of the most used Monte Carlo methods for integration were compared by Hörmann and Leydold in 2005 [15]. In the financial sector, Monte Carlo methods are used, among others, to reduce the uncertainty involved in estimating future outcomes [16]. A wide area of application for Monte Carlo methods can also be found in physics [5, 6]. Zabenkov and Kochubey have used the Monte Carlo simulation to study the dependence of the spatial resolution of a luminescent object inside the skin [17]. Below, we are using Monte Carlo simulation to calculate the area of $C$.

In our solution model, we generate random points in the rectangle formed by the two random variables $X_1$ and $X_2$ as defined before.




**(*a*)** **(*b*)**

**Figure 4. Two samples for MC method**

The red points in (*a*) and (*b*) lie in $C$ and can be used to approximate the probability:

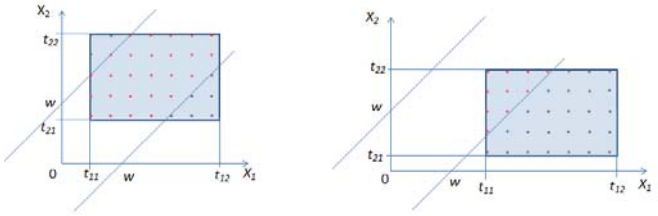
$$p \approx \frac{number\ of\ red\ points}{number\ of\ all\ points} \qquad (6)$$

```
void MCSimulation::Execute()
{
        double counter = 0;

        for(int i=0; i<points; i++)
        {
                if(InSolution(GeneratePoint()))
                        counter++;
        }
        result = counter;
}
```

**Figure 5. Code in C++ for Monte-Carlo method**

### B. *Second solution model – Numerical Integration*

If we do not choose random points but divide the rectangle in equally spaced subparts, we can approximate *C* numerically. The points lie on the edges between the parts. Again, the number of red points is used to approximate the probability the way we did it in (6).



**(*a*)** **(*b*)**

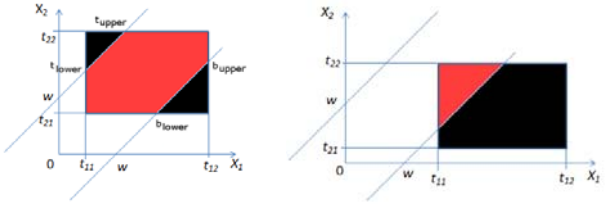
**Figure 6. Two Samples for the numerical method**

```
member this.solve (problem:Problem, solvers:int)=
        let w = problem.Bandwith
        let mutable counter = 0

        let xlist = [ t11 .. stepSize .. t12 ]
        let ylist = [ t21 .. stepSize .. t22 ]
        for x in xlist do
           for y in ylist do
              if x <= y+w && y<= x+w then
                  counter <- counter + 1

        (float)counter/
        (float)(xlist.Length*ylist.Length)
```

**Figure 7. Code in F# for the numerical integration**

### C. *Third solution model – Exact Integration*

As the problem is in two-dimensional space, we can use normal integration to solve it exactly.



**(*a*)** **(*b*)**

**Figure 8. Two Samples for the exact method**

The red area shown in figure 6 represents *C*. To get its value, some variables have to be defined first.

$$b_{lower} = \max(\min(t_{12}, t_{21} + w), t_{11}) \qquad (7)$$

$$b_{upper} = \max(\min(t_{12}, t_{22} + w), t_{11}) \qquad (8)$$

$$t_{lower} = \min(\max(t_{11}, t_{21} - w), t_{12}) \qquad (9)$$

$$t_{upper} = \min(\max(t_{11}, t_{22} - w), t_{12}) \qquad (10)$$

Then *C* can be computed by integrating the lines.

$$C = \int_{t_{lower}}^{t_{upper}} (x + w)dx - \int_{b_{lower}}^{b_{upper}} (x - w)dx \qquad (11)$$

The area *A* is $(t_{12}–t_{11}) \cdot (t_{22}–t_{21})$.

```
member this.solve (problem:Problem, solvers:int)=
 let w = problem.Bandwith

 let xBottom = Math.Max(Math.Min(t12,t21+w), t11)
 let xBottomTop=Math.Max(Math.Min(t12,t22+w),t11)

 let xTop = Math.Min(Math.Max(t11, t22-w), t12)
 let xTopBottom=Math.Min(Math.Max(t11,t21-w),t12)

 // integrate the upper line
 let A = 1.0 / 2.0*xTop*xTop +
         w * xTop - t21 * xTop -
         (1.0/2.0*xTopBottom*xTopBottom +
         w* xTopBottom - t21*xTopBottom) +
         Math.Max(0.0, (t12-xTop)*(t22-t21))

 // integrate the lower line
 let B = 1.0 / 2.0*xBottomTop*xBottomTop -
         w * xBottomTop - t21 * xBottomTop -
         (1.0/2.0*xBottom*xBottom -
         w* xBottom - t21*xBottom) +
         Math.Max(0.0,(t12-xBottomTop)*(t22-t21))

 // the area we search for is the difference
 // between A and B
 let area = A-B
 let completeArea = (t12-t11)*(t2-t1)
 // return the probability
 area / completeArea
```

**Figure 9. Code in F# for the exact integration**

## III. EXPERIMENTS

We will now compare the different solution models. In the experiments, only one parameter is changed at a time, the others remain constant. Interesting parameters are $t_{11}$, $t_{12}$, $t_{21}$, $t_{22}$, $w$, the number of MCS points and the step size of the numerical integration. $t_{11}$, $t_{12}$, $t_{21}$ and $t_{22}$ will remain constant in all experiments as they do not significantly change the performance of the methods. The parameter $w$ is called "bandwidth" in all experiments, as it defines the half-distance between the two lines in the two-dimensional space.

```
// define the random variables used in the experiments

let X1 = new RandomVariable(0.0,10.0)
let X2 = new RandomVariable(0.0,10.0)

let xList = [ X1; X2 ]

// define the solvers

let NISolver = new NumericalIntegration.Solver(0.02);
let MCSolver = new MonteCarloSimulation.Solver(500000);
let EISolver = new ExactIntegration.Solver();

// Experiment 1 - Changing the bandwith

let bandwidths = [ 0.0 .. 0.5 .. 10.0 ]

// Numerical Integration
let NIexperiments =
  [| for i in bandwidths
  do yield new Experiment(new Problem(xList,i), NISolver)
  |]
let NIresults =
  NIexperiments
  |> Array.map (fun exp -> exp.execute)

// Monte Carlo Simulations
let MCSexperiments =
  [| for i in bandwidths
  do yield new Experiment(new Problem(xList,i), MCSolver)
  |]

let MCSresults =
  MCSexperiments
  |> Array.Parallel.map (fun exp -> exp.execute)

// Exact Integration
let EIexperiments =
  [| for i in bandwidths
  do yield new Experiment(new Problem(xList,i), EISolver)
  |]

let EIresults =
  EIexperiments
  |> Array.map (fun exp -> exp.execute)
```

**Figure 10. Experimental scenario in F#**

For all experiments, random variables $X_1$ and $X_2$ are used, having the following continuous distribution:

$$P(X_1 = k) = 0.1 \; \forall k \in [0; 10], \; k \in \mathbb{R} \quad (12)$$

$$P(X_2 = k) = 0.1 \; \forall k \in [0; 10], \; k \in \mathbb{R} \quad (13)$$

### A. *First experiment: Changing the bandwidth*

This experiment uses all solution methods and compares them by applying different bandwidths problems. The graph below shows three solutions by the probability delta to the exact value. The solutions are two Monte Carlo Simulations with 5*e*5 points and the numerical integration method with step size 0.02. All methods are applied to problems with bandwidths varying from 0 to 10.

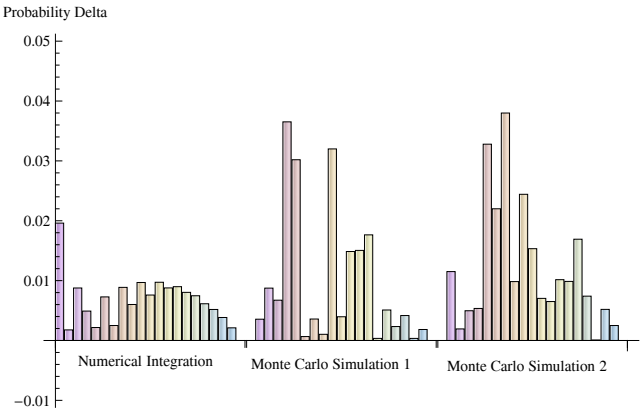



**Figure 11. Probability deltas for different bandwidths**

One can see that different Monte Carlo Simulations (MCS) produce very different output. Though, the overall difference to the exact value increases for bandwidths near the middle of the interval.

TABLE I. COMPUTED VALUES

| *w* | *Numerical* | *MCS1* | *MCS2* | *Exact* |
|---|---|---|---|---|
| 2 | 0.362168 | 0.323478 | 0.354648 | 0.36 |
| 4 | 0.633987 | 0.641032 | 0.64984 | 0.64 |
| 6 | 0.831219 | 0.855048 | 0.833494 | 0.84 |
| 8 | 0.953864 | 0.957670 | 0.952594 | 0.96 |
| 10 | 1.000000 | 1.000000 | 1.000000 | 1.00 |

### B. *Second experiment: changing the step size of the numerical method*

In this experiment we change the step size for the numerical integration method, leaving *w* constant at value 5. The approximation is getting better with increasing step size. Some values break out because the step size there fits well to the given problem. The exact value in this simulation is 0.75.

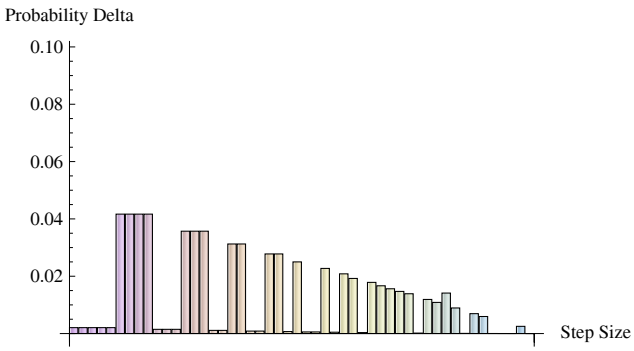
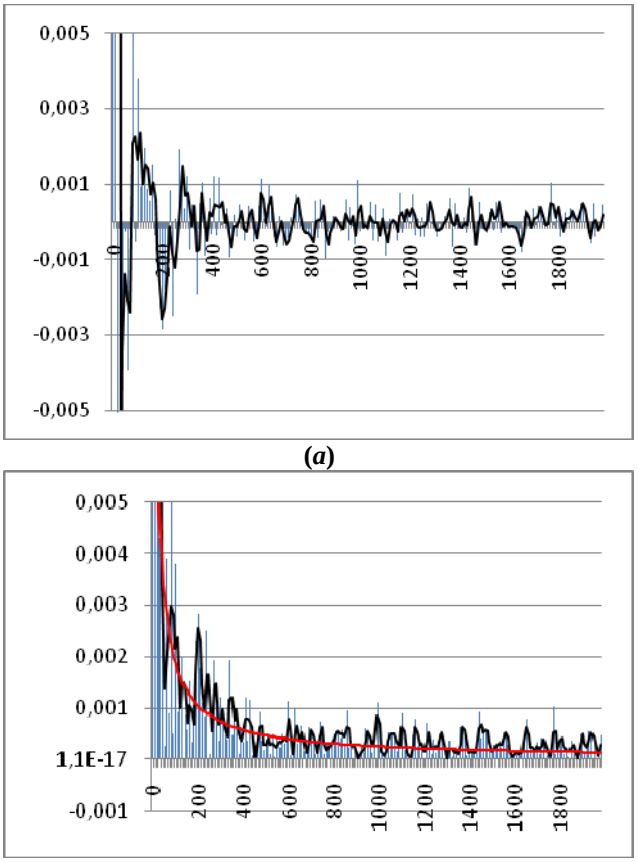



**Figure 12. Numerical method, step size from 1e1 to 2e-2**

### C. *Third experiment: changing the number of MCS points*

Now, the number of points used to approximate the probability is altered. *w* remains constant at value 5. Fig. 8 shows exact values of a MCS, Fig. 8 and Fig. 9 show the difference between the exact value (0.75) and the result of the experiments.




(*a*)

(*b*)

**Figure 13. MCS deltas with 0 to 2e7 points, step size 1e4 (a) delta (b) |delta|**

TABLE II. COMPUTED VALUES FOR THE EXPERIMENT FROM FIGURE 13

| *# points* | *Value* | *Delta* | *\|Delta\|* |
|---|---|---|---|
| 0 | 1.000000000 | 0.250000000 | 0.250000000 |
| 1e5 | 0.753811111 | 0.003811111 | 0.003811111 |
| 2e5 | 0.747168421 | -0.002831579 | 0.002831579 |
| 3e5 | 0.751206897 | 0.001206897 | 0.001206897 |
| 4e5 | 0.749664103 | -0.000335897 | 0.000335897 |
| 5e5 | 0.749600000 | -0.000400000 | 0.000400000 |
| 6e5 | 0.751137288 | 0.001137288 | 0.001137288 |
| 7e5 | 0.749449275 | -0.000550725 | 0.000550725 |
| 8e5 | 0.749494937 | -0.000505063 | 0.000505063 |
| 9e5 | 0.750067416 | 0.000067416 | 0.000067416 |
| 1e6 | 0.749761616 | -0.000238384 | 0.000238384 |
| 1.1e6 | 0.749096330 | -0.000903670 | 0.000903670 |
| 1.2e6 | 0.749974790 | -0.000025210 | 0.000025210 |
| 1.3e6 | 0.749975194 | -0.000024806 | 0.000024806 |
| 1.4e6 | 0.749824460 | -0.000175540 | 0.000175540 |
| 1.5e6 | 0.749869799 | -0.000130201 | 0.000130201 |
| 1.6e6 | 0.749827673 | -0.000172327 | 0.000172327 |
| 1.7e6 | 0.750348521 | 0.000348521 | 0.000348521 |
| 1.8e6 | 0.750415642 | 0.000415642 | 0.000415642 |
| 1.9e6 | 0.750494709 | 0.000494709 | 0.000494709 |
| 2e7 | 0.750215075 | 0.000215075 | 0.000215075 |

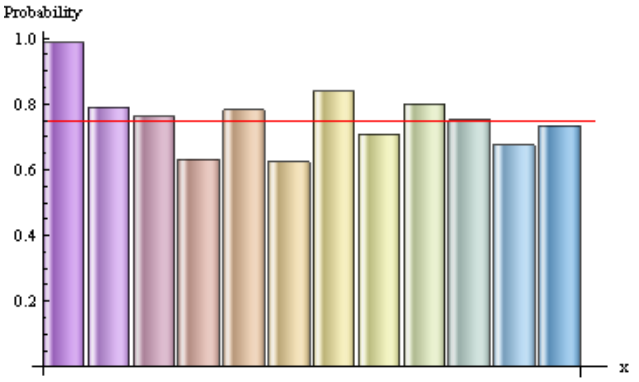



**Figure 14. MCS results with 1e4 to 6e5 points**

The figures 15 and 16 illustrate a detailed comparison between the two approximation methods. One can see the numerical method has, as expected, a smooth and faster convergence.

## IV. CONCLUSION

The two approximation models described in this paper do not have the same accuracy as the exact method, though they produce output very close to the desired one. This is getting important when exact solutions are not possible or much too expensive in needed computational power. Furthermore, if there are outside conditions, that have to be taken into account, the Monte Carlo method seems more flexible, as these can be simply added by modifying the simulation. As for the integration methods, in that case a new mathematical model has to be defined and implemented addressing these new conditions. A different probability distribution, for example, could not be easily modeled by numerical integration. In this paper, we gave a basic example, able to address the three solution methods. The described scenario can be extended in a variety of ways. For example, we can look at more than two parallel threads or consider different wait times ($w_i$) for the locking (Figure 1). In this case we would have to deal with multidimensional integrals for calculating the exact probability. The initial problem model can also be transferred in other areas such as database transactions or synchronization.

## V. USED TOOLS

All computation was performed on an Intel Core 2 Duo T7500 Processor. The code for the experiments was written in F# [3] and C++, using the .NET 4.0 Platform and Visual Studio 2010 Beta 2. The graphics were made in Mathematica 7.0 [11], GIMP 2.0 and Microsoft Excel 2010.

## REFERENCES


[1] R. E. Caflisch, "Monte Carlo and quasi-Monte Carlo methods", *Acta Numerica* 7, pp. 1-49, Cambridge University Press, 1998.

[2] B. Goetz, J. Bloch, J. Bowbeer, D. Lea, D. Holmes, T. Peierls, Java Concurrency in Practice, Addison-Wesley Longman, Amsterdam, 2006.

[3] J. Harrop, F # for Scientists, John Wiley & Sons, 2008.

[4] W. K. Hastings, Monte Carlo Sampling Methods Using Markov Chains and Their Applications, *Biometrica*, Vol. 57, No. 1, pp 97-109, 1970.

[5] D. P. Landau, K. Binder, A Guide to Monte Carlo Simulations in Statistical Physics, New York Cambridge University Press, 2005.

[6] J. S. Liu, Monte Carlo Strategies in Scientific Computing, Springer, New York, 2008.

[7] J. Magee, J. Kramer, State Models and Java Programs, John Wiley & Sons, 2nd Edition, 2006.

[8] N. Metropolis, S. Ulam, "The Monte Carlo Method", *Journal of the American Statistical Association* 44 (247), pp. 335-341, 1949.

[9] K. A. Robbins, S. Robbins, UNIX Systems Programming: Communication, Concurrency and Threads, Prentice Hall, 2003.

[10] C. P. Robert, G. Casella, Monte Carlo Statistical Methods, 2nd Edition, Springer, New York, 2005.

[11] H. Ruskeepaa, Mathematica Navigator: Mathematics, Statistics and Graphics, Academic Press, 3rd Edition, 2009.

[12] R. W. Shonkwiler, F. Mendivil, Explorations in Monte Carlo Methods, Springer, New York, 2009.

[13] CSM: Computer Science and Mathematics, "Impact of Monte Carlo methods on scientific research", URL: *http://www.csm.ornl.gov/ssi-expo/MChist.html*.

[14] MacKinnon, A.: Computational Physics, Monte-Carlo Integration, URL: *http://www.ipp.mpg.de/~rfs/comas/Helsinki/helsinki04/compphys/node87.html*.

[15] Hörmann, W., Leydold, J.: "Monte Carlo Integration using Importance Sampling and Gibbs Sampling", In: H. Dag and Y. Deng (eds.), *Proceedings of the International Conference on Computational Science and Engineering*, pp.92-97, Istanbul, 2005.

[16] Iordanova, T.: "Introduction to Monte Carlo Simulation", In: INVESTOPEDIA, URL: *http://www.investopedia.com/articles/07/monte_carlo_intro.asp*.

[17] Zabenkov, I.V., Kochubey, V.I.: "Monte Carlo simulation of the recording of fluorescent objects in the skin", In: Optics and Spectroscopy, vol. 107, nr. 6, pp. 898-902, Springer, 2009.

[18] Programming Praxis, "Calculating Pi", URL: *http://programmingpraxis.com/2009/10/09/calculating-pi/*.

[19] Gonsalves, G.J.: "Monte Carlo Calculation of $\pi$", URL: *http://www.physics.buffalo.edu/phy516/jan25.pdf*.

[20] Logofătu, D.: Bazele programării în C. Aplicații, pp. 229-246, Polirom, Iași, 2006.

[21] Logofătu, D.: Eine praktische Einführung in C, pp. 209-226, entwickler-press, München, 2008.

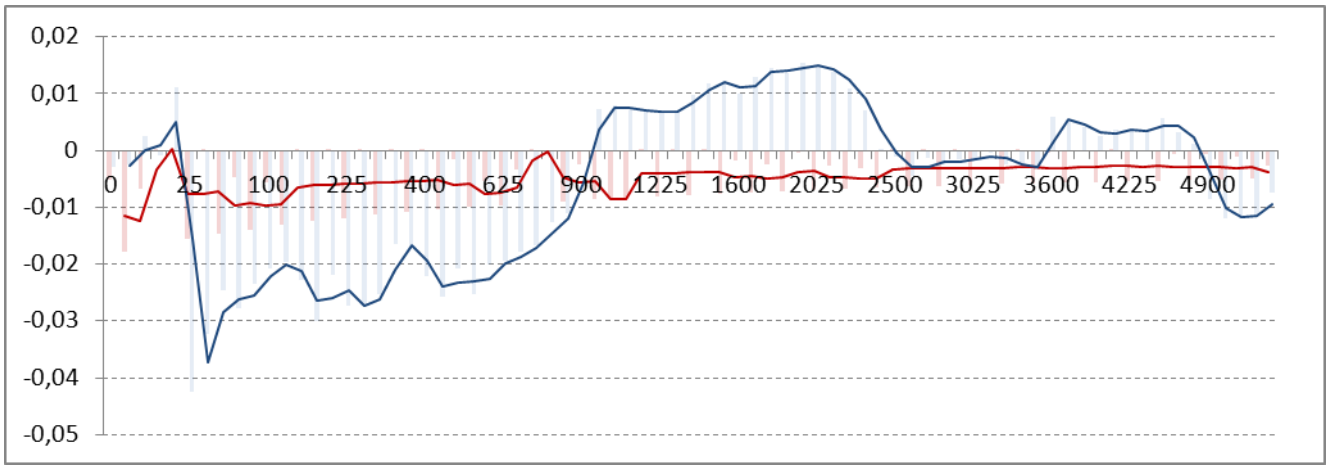
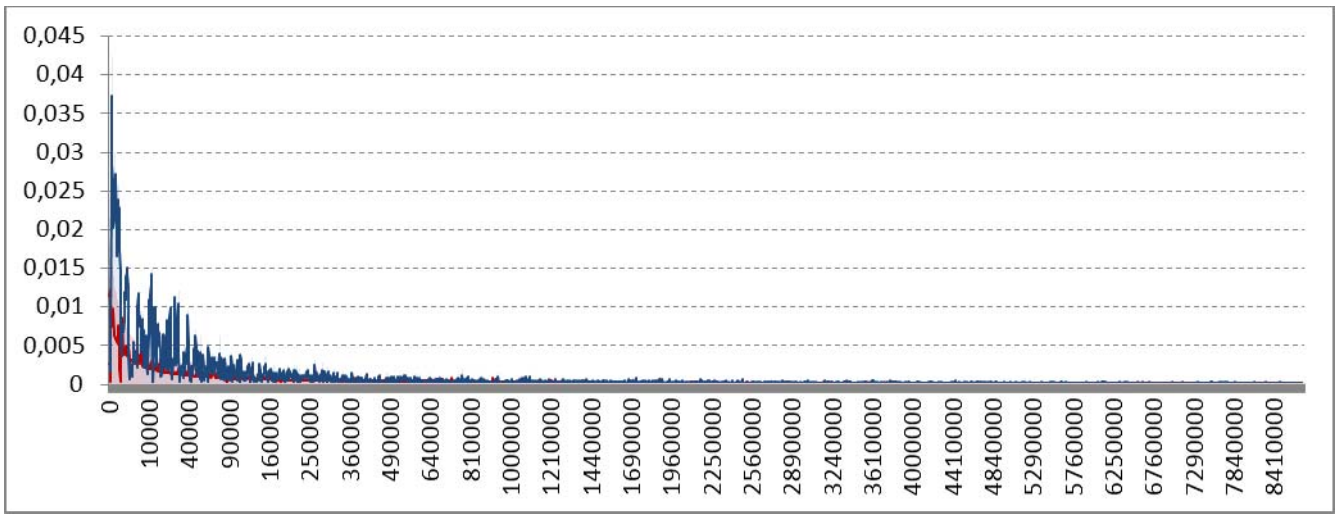
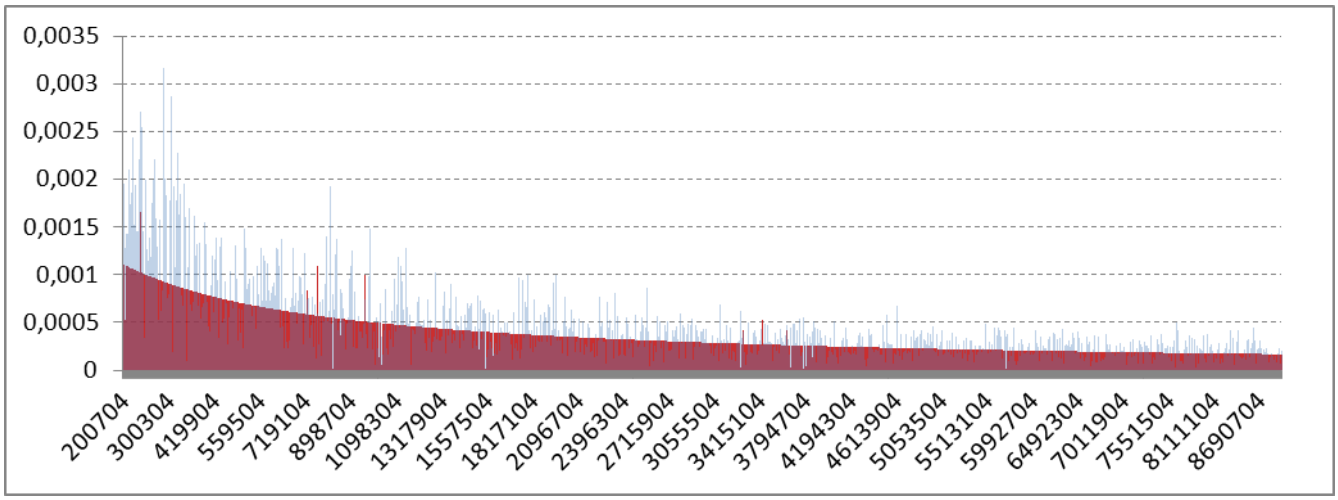



**Figure 15. Comparison of the error delta fluctuation for the two approximation methods (Monte Carlo Simulation – blue, Numerical Integration – red) ranging from 0 to 5e3**




**(*a*)**




**(*b*)**

**Figure 16. Comparison of the absolute error delta for the two approximation methods (Monte Carlo Simulation – blue, Numerical Integration – red)**
**(a) number of points increasing from 0 to 9e6**
**(b) number of points increasing from 2e5 to 9e6**