Correctly Rounded Newton-Cotes Quadrature

Laurent Fousse

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Correctly Rounded Newton-Cotes Quadrature

Laurent Fousse

N° 5605

Juin 2005

Thème SYM

Rapport de recherche

ISSN 0249-6399 ISRN INRIA/RR--5605--FR+ENG

Correctly Rounded Newton-Cotes Quadrature

Laurent Fousse

Thème SYM — Systèmes symboliques
Projet Spaces

Rapport de recherche n° 5605 — Juin 2005 — 20 pages

Abstract: Numerical integration is an important operation for scientific computations. Although the different quadrature methods have been well studied from a mathematical point of view, the analysis of the actual error when performing the quadrature on a computer is often neglected. This step is however required for certified arithmetics.

We study the Newton-Cotes quadrature scheme and give enough details on the algorithms and the error bounds to enable software developers to write a correctly-rounded Newton-Cotes quadrature.

Key-words: numerical integration, correct rounding, Newton-Cotes

Unité de recherche INRIA Lorraine
LORIA, Technopôle de Nancy-Brabois, Campus scientifique,
615, rue du Jardin Botanique, BP 101, 54602 Villers-Lès-Nancy (France)
Téléphone : +33 3 83 59 30 00 — Télécopie : +33 3 83 27 83 19

Intégration de Newton-Cotes avec arrondi correct

Résumé : L'intégration numérique est une opération importante pour le calcul scientifique. Bien que les différentes méthodes d'intégration aient été bien étudiées d'un point de vue mathématique, l'analyse de l'erreur commise lors d'un calcul numérique est souvent omise. Cette étape est cependant nécessaire dans le cadre de l'arithmétique certifiée des ordinateurs.

Nous étudions la méthode d'intégration numérique de Newton-Cotes en fournissant suffisamment de détails concernant les algorithmes et les bornes d'erreurs pour permettre une implémentation avec arrondi correct.

Mots-clés : intégration numérique, arrondi correct, Newton-Cotes

1 Introduction

Numerical integration is an operation that is common in mathematical software (`intnum` in Pari/GP, `quadrature` in MuPAD, `evalf(Int(...))` in Maple for example). At first glance this is a topic that seems well studied: several quadrature schemes with different convergence properties are known as well as strategies to use them (e.g. adaptative error analysis) or to combine them. When compared to the basic four operations (addition, subtraction, multiplication and division) one notices however the lack of correct rounding.

Let us illustrate the problem by an example, and compare the values returned respectively by Pari/GP and Maple for $I = \int_{10^6}^{10^6+\pi} \sin(\cos(t)) - \cos(\sin(t))dt$. On the one hand Pari/GP gives:

```
? \p19
? intnum(t = 10^6, 10^6+Pi, sin(cos(t)) - cos(sin(t)))
%1 = -1.810600390080270954
```

while Maple returns:

```
> Digits:=19: evalf(Int(sin(cos(t)) - cos(sin(t)), t=10^6..10^6 + Pi));
                          -1.810600390080269775
```

so clearly one software at least is wrong here.

The definition and normalization of rounding modes was a major advance in floating-point arithmetic with respect to the portability and reproducibility of computations. It is a challenging goal to extend this notion of correct rounding to more complex operations like numerical integration.

The error analysis of quadrature methods however is often limited to the mathematical error. In the context of correct rounding this is sadly not sufficient, because a precise bound on the total error is needed to be able to decide of the actual accuracy of the result. Previous works in the field includes the study of the adaptative `quadrature` function of MuPAD [9] and dynamic error control of simple or multiple integral [10, 11]. What differentiates our work is the careful study of the error term. It is often the case that the roundoff error is merely estimated or sometimes even dismissed by computing with a precision that is "good enough". Instead, our goal is to give a rigorous formula bounding the total error made in the computation (both the mathematical and roundoff error) in order to be able to guarantee the final result.

In this paper $f : [a, b] \to \mathbb{R}$ is the C^∞ function we want to integrate on a finite domain $[a, b]$, and n is the number of evaluation points in the Newton-Cotes method. Let

$$I = \int_a^b f(x)dx$$

be the exact value value of the integral, assumed to be finite.

n	Name	Formula	c_n
2	trapezes	$I_2 = (b-a)\frac{f_0+f_1}{2}$	$-\frac{1}{12}$
3	Simpson	$I_3 = \frac{b-a}{6}(f_0+4f_1+f_2)$	$-\frac{1}{90}$
4	Simpon's $\frac{3}{8}$ rule	$I_4 = \frac{3h}{8}(f_0+3f_1+3f_2+f_3)$	$-\frac{3}{80}$
5	Boole	$I_5 = \frac{2}{45}h(7f_0+32f_1+12f_2+32f_3+7f_4)$	$-\frac{8}{945}$

Figure 1: Newton-Cotes integration formulas for small n. To simplify the notations we define $f_i = f(x_i)$.

The Newton-Cotes method uses equally-spaced evaluation points in the integration domain, commonly refered to as "abscissas" $x_0, x_1, \ldots x_{n-1}$:

$$\text{for } 0 \leq i < n,\ x_i = a + ih \quad \text{where } h = \frac{b-a}{n-1} \quad \text{is the step.}$$

Since $x_0 = a$ and $x_{n-1} = b$, the bounds are used as abscissa and the method is said to be closed. The principle of the method is to approximate the function with the Lagrange interpolating polynomial with respect to the abscissas. The coming formulas follow directly from this statement.

For $i \in [0, n-1]$, let $l_i(x) = \prod_{j \neq i} \frac{(x-x_j)}{(x_i-x_j)}$ and $w_i = \frac{1}{h}\int_a^b l_i(x)dx$. The approximated integral is then

$$I_n = h \sum_{i=0}^{n-1} w_i f(x_i).$$

The mathematical error $E_n = I - I_n$ is of the form $E_n = c_n h^{n+1} f^{(n)}(\zeta)$ for n even and $E_n = c_n h^{n+2} f^{(n+1)}(\zeta)$ for n odd for some $\zeta \in]a, b[$ [8] (this will be detailed afterwards).

Firstly we describe the algorithms used in our implementation of the Newton-Cotes quadrature scheme. Then we establish some facts about the mathematical error of the method as well as a few useful lemmas relevant to floating-point arithmetic. These results allow us to proceed to a thorough study of the error made when using the Newton-Cotes quadrature scheme on a computer using floating-point arithmetic. Then we state our main theorem (Theorem 3.11). We conclude with some experiments and remarks about the quadrature scheme studied.

2 Algorithm for the computation of the Newton-Cotes coefficients

In the Newton-Cotes method we distinguish the computation of the coefficients from the quadrature itself, since the coefficients can be precomputed, and reused for several quadratures using the same number of points. For example the *composition* technique splits the initial integration interval in several parts and applies the same method on each part.

We describe here the algorithm for the computation of the coefficients. The full quadrature algorithm is explained in section 3.5 together with a discussion of the error.

First we show that the coefficients do not depend on the integration interval. This is true of every linear quadrature scheme, even with non-equally spaced abscissas simply because of the linearity of the integral. We include however the proof only for the Newton-Cotes case because we want to derive the formula for the coefficients.

Proposition 2.1. *The coefficients of the Newton-Cotes methods do not depend on the integration interval, and are symmetric with respect to the middle of the interval.*

PROOF: We transform the expression of w_i from section 1:

$$\begin{aligned} w_i &= \tfrac{1}{h}\int_a^b l_i(x)dx \\ &= \int_0^{n-1} l_i(a+xh)dx \\ &= \int_0^{n-1} \prod_{j\neq i} \tfrac{(a+xh-x_j)}{(x_i-x_j)}dx \\ &= \int_0^{n-1} \prod_{j\neq i} \tfrac{(a+xh-(a+jh))}{h(i-j)}dx \\ &= \int_0^{n-1} \prod_{j\neq i} \tfrac{(x-j)}{(i-j)}dx \\ &= \tfrac{(-1)^{n-1-i}}{i!(n-1-i)!}\int_0^{n-1} \prod_{j\neq i}(x-j)dx. \end{aligned}$$

The variable change $x \mapsto n-1-x$ shows that $w_{n-1-i} = w_i$. □

Let $l_i^\star(x) = \prod_{j\neq i}(x-j)$ and L_i the antiderivative of $l_i^\star$ such that $L_i(0) = 0$. Let $u_i = \frac{(-1)^{n-1-i}}{i!(n-1-i)!} = (-1)^{n-1-i}\frac{\binom{n-1}{i}}{n!}$.

Then we compute the weights as

$$w_i = u_i L_i(n-1) \tag{1}$$

From the formula one can notice the weights are rational. They can thus be computed exactly as $w_i = \frac{b_i}{\delta}$ by Algorithm 1 below.

Algorithm 1 Newton-Cotes coefficients

$\delta \leftarrow \mathrm{lcm}(2, 3, \ldots, n)$
$l_0^\star \leftarrow (x-1)(x-2)\ldots(x-n-1)\delta$
for $i \leftarrow 0$ to $n-1$ **do**
  $L_i \leftarrow \int_0^{n-1} l_i^\star(x)dx$ ▷ L_i is an integer
  $l_{i+1}^\star \leftarrow \frac{x-i}{x-(i+1)} l_i^\star$ ▷ update is done in place
  $b_i \leftarrow \binom{n-1}{i} L_i$
end for
return $(b_0, b_1, \ldots, b_{\lfloor n/2 \rfloor}, \delta \cdot n!)$

Timings were done on a Pentium-4 computer with MPFR[7] version 2.1.1 and are shown on figure 2. The time complexity is $\mathcal{O}(n \log^2 n)$ for each loop thus $\mathcal{O}(n^3 \log^2 n)$ in total.

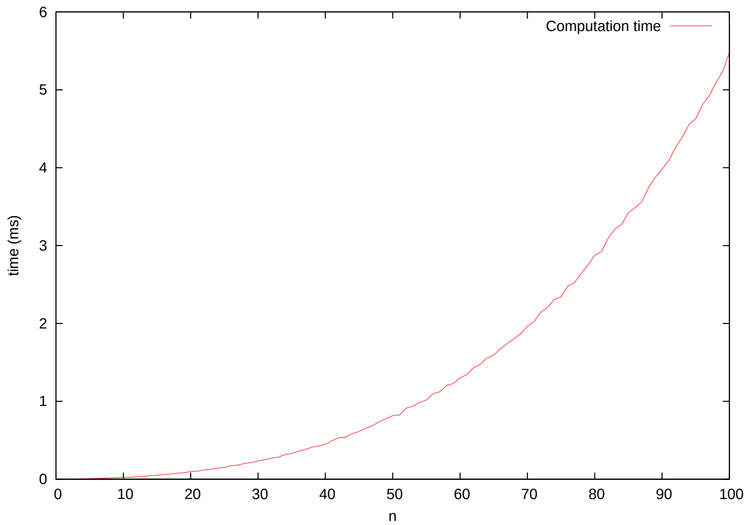

Figure 2: Coefficients computation time for small n.

3 Error bounds

When performing a numerical integration by means of a Newton-Cotes method, there are two sources of errors to consider: the mathematical error that comes from the method itself, and the roundoff error in the computation which depends on the way we implement the algorithm.

We first give bounds on the mathematical error with elementary proofs.

3.1 Bounds on the mathematical error

Theorem 3.1. *For n odd, the Newton-Cotes integration method on $[a, b]$ with n points is exact for polynomials of degree $\leq n$. For n even, it is exact for polynomials of degree $\leq n-1$.*

PROOF: For any n, the method is exact for polynomials of degree up to $n - 1$ because in that case the Lagrange interpolating polynomial is f exactly.

Let now n be odd. The choice of the evaluation points for the method gives $x_i + x_{n-1-i} = a+b$. Let $g(x) = (x-x_0)(x-x_1)\dots(x-x_{n-1})$.

$$\begin{aligned}
g\left(\frac{a+b}{2}-x\right) &= \prod_{i=0}^{n-1}\left(\frac{a+b}{2}-x-x_i\right)\\
&= \prod_{i=0}^{n-1}\left(\frac{a+b}{2}-x-(a+b-x_{n-1-i})\right)\\
&= \prod_{i=0}^{n-1}\left(-\frac{a+b}{2}-x+x_{n-1-i}\right)\\
&= (-1)^n\prod_{i=0}^{n-1}\left(\frac{a+b}{2}+x-x_{n-1-i}\right)\\
&= -g\left(\frac{a+b}{2}+x\right)
\end{aligned}$$

and then $\int_a^b g(x)dx = 0 = \sum_{i=0}^{n-1} w_i g(x_i)$ since $g(x_i) = 0$. The Newton-Cotes method is exact for polynomials of degree $n-1$ at most, and for g which has degree n, hence by linearity is exact for all polynomials of degree at most n.

3.2 Peano theorem

In this section we establish the expression of the mathematical error given in Section 1. For this the formalism of the Peano kernel of an integration method is a powerful tool.

For a quadrature method $I : C^{\nu+1}([a,b]) \to \mathbb{R}$ the error $E : f \mapsto \int_a^b f(x)dx - I(f)$ can be seen as a linear function $C^{\nu+1}([a,b]) \to \mathbb{R}$. We have the following result:

Theorem 3.2. *Define*

$$K_\nu(t) = \frac{1}{\nu!}E[x \mapsto (x-t)_+^\nu]$$

and

$$(x-t)_+^\nu = \begin{cases} (x-t)^\nu & \text{if } x > t \\ 0 & \text{otherwise.} \end{cases}$$

If $E[p] = 0$ for all polynomials p of degree at most ν then for $f \in C^{\nu+1}([a,b])$,

$$E[f] = \int_a^b f^{(\nu+1)}(t)K_n(t)dt.$$

K_ν is called the Peano kernel *of order ν of E.*

PROOF: writing the Taylor series associated with f at origin a:

$$\begin{aligned} f(x) &= p_\nu(x) + \int_a^x \frac{1}{\nu!}(x-t)^\nu f^{(\nu+1)}(t)dt \\ &= p_\nu(x) + \int_a^b \frac{1}{\nu!}(x-t)_+^\nu f^{(\nu+1)}(t)dt, \\ E[f] &= E\left[\int_a^b \frac{1}{\nu!}(x-t)_+^\nu f^{(\nu+1)}(t)dt\right] \\ &= \int_a^b E\left[\frac{1}{\nu!}(x-t)_+^\nu\right] f^{(\nu+1)}(t)dt. \end{aligned}$$

This theorem links the mathematical error with the maximal degree of the polynomial the method integrates exactly (its maximal order). In Theorem 3.1 we proved that the order of an n-points Newton-Cotes method is at least $n-1$ for n even and n for n odd. If we admit that this is optimal and that the Peano kernel of the Newton-Cotes method does not change sign on $[a,b]$, then we have a method to compute the coefficient c_n given in table 1. With the mean value theorem there exists $\zeta \in]a,b[$ such that:

$$E[f] = f^{(\nu+1)}(\zeta)\int_a^b K_\nu(t)dt \tag{2}$$

and we obtain

$$c_n := \begin{cases} \frac{1}{h^{n+1}}\int_a^b K_n(t)dt & \text{if n is even,} \\ \frac{1}{h^{n+2}}\int_a^b K_{n+1}(t)dt & \text{if n is odd.} \end{cases}$$

For example for the 3-points method known as *Simpson's rule*, we get

$$E[f] = \int_a^b f(x)dx - \frac{b-a}{6}\left[(f(a) + 4f(\frac{a+b}{2}) + f(b))\right],$$

$$K(t) = \frac{1}{6}E_x((x-t)_+^3),$$

$$
\begin{aligned}
6K(t) &= \int_a^b (x-t)_+^3 dx - \frac{b-a}{6}\left[(a-t)_+^3 + 4(\frac{a+b}{2}-t)_+^3 + (b-t)_+^3\right] \\
&= \int_t^b (x-t)^3 dx - \frac{b-a}{6}\left[4(\frac{a+b}{2}-t)_+^3 + (b-t)^3\right] \\
&= \begin{cases} \frac{(b-t)^4}{4} - \frac{b-a}{6}\left[4(\frac{a+b}{2}-t)^3 + (b-t)^3\right] & \text{if } t < \frac{a+b}{2} \\ \frac{(b-t)^4}{4} - \frac{b-a}{6}(b-t)^3 & \text{if } t \geq \frac{a+b}{2} \end{cases}, \\
\int_a^b K(t)dt &= \int_a^b \frac{(b-t)^4}{24} - \frac{b-a}{36}(b-t)^3 dt - \int_a^{\frac{a+b}{2}} \frac{(b-a)}{9}(\frac{a+b}{2}-t)^3 dt \\
&= \frac{(b-a)^5}{120} - \frac{(b-a)^5}{144} - \frac{(b-a)^5}{576} = -\frac{1}{90}\left(\frac{b-a}{2}\right)^5
\end{aligned}
$$

and we find the value $c_3 = -\frac{1}{90}$ given in the third row of Table 1.

3.3 Upper bound for the c_n coefficients

In order to be able to give an absolute bound on the mathematical error we need to bound the c_n coefficients. We detail here the bound and the proof for n even, and give the result only for n odd. Recall that for n even, the Newton-Cotes method is exact for all polynomials up to degree $n-1$.

Take for f a monic degree-n polynomial p in equation (2) to get:

$$E_n[p] = p^{(n)}(\zeta) \int_a^b K_{n-1}(t)dt$$

and then $c_n = \frac{E_n[p]}{n!h^{n+1}}$ since $p^{(n)}(\zeta) = n!$.

In particular for $p(x) = (x-x_0)(x-x_1)\dots(x-x_{n-1})$, we have

$$E_n[p] = \int_a^b p(x)dx - \sum_{i=0}^{n-1} w_i p(x_i) = \int_a^b p(x)dx$$

so it is enough to bound $\left|\int_a^b p(x)dx\right|$ in order to bound c_n. We will use repeatedly the following simple lemma:

Lemma 3.3. *For $(u,x,v) \in \mathbb{R}^3$ such that $u \leq x \leq v$, $|x-u||x-v| \leq \frac{(v-u)^2}{4}$.*

Proposition 3.4.

$$\forall x \in [a,b],\ |p(x)| \leq \frac{h^n(n-1)!}{4}.$$

PROOF: Let $x \in [a,b]$ such that $p(x) \neq 0$. Then there is $i_0 \in [0, n-2]$ such that $x \in]x_{i_0}, x_{i_0+1}[$, and thus

$$|x-x_{i_0}||x-x_{i_0+1}| \leq \frac{h^2}{4}. \quad \text{[Lemma 3.3]}$$

Then

$$\begin{aligned}
|p(x)| &\leq \prod_{i=0}^{n-1} |x - x_i| \\
&\leq \frac{h^2}{4} \prod_{i \neq \{i_0, i_0+1\}} |x - x_i| \\
&\leq \frac{h^2}{4} \left[\prod_{i > i_0+1} (i - i_0)h \right] \left[\prod_{i < i_0} (i_0 + 1 - i)h \right],
\end{aligned}$$

$$\begin{aligned}
\left[\prod_{i > i_0+1} (i - i_0)h \right] \left[\prod_{i < i_0} (i_0 + 1 - i)h \right] &\leq (n - 1 - i_0)!(i_0 - 1)!h^{n-2} \\
&\leq (n-1)!h^{n-2},
\end{aligned}$$

$$|p(x)| \leq \frac{h^n (n-1)!}{4}. \quad \square$$

On the other hand we have $b - a = (n-1)h$, which yields

$$|E_n[p]| = \left| \int_a^b p(x)dx \right| \leq \int_a^b |p(x)|dx \leq \frac{h^{n+1}(n-1)!(n-1)}{4}.$$

With $E_n[p] = c_n h^{n+1} n!$ we get

$$|c_n| \leq \frac{n-1}{4n} \leq \frac{1}{4}.$$

For n odd we take $p(x) = (x - x_0)(x - x_1)...(x - x_{n-1})(x - \frac{a+b}{2})$; we have the evaluation points and the middle of the interval as zeroes of p.

Similar computations give:

$$|p(x)| \leq \frac{h^{n+1}(n-1)!(n-1)}{8}$$

then

$$\left| \int_a^b p(x)dx \right| \leq \frac{h^{n+2}(n-1)!(n-1)^2}{8}$$

and with $E_n = c_n h^{n+2} p^{(n+1)}$ we get

$$|c_n| \leq \frac{(n-1)^2}{8n(n+1)} \leq \frac{1}{8}.$$

3.4 Ulp calculus

For the error analysis of Algorithm 2, we need a few useful lemmas concerning the "ulp[1] calculus", as well as some definitions. The floating-point numbers are represented with radix 2 (this could be generalized for any radix but radix 2 is simpler and is natural on computers). For this section, p is the working precision, and we assume all floating-point numbers are normalized, which means in our notations that the exponent range is unbounded.

Definition 3.5 (Exponent). *For a non-zero real number x we define $E(x) := 1+\lfloor \log_2 |x| \rfloor$, such that $2^{E(x)-1} \leq |x| < 2^{E(x)}$.*

Definition 3.6 (Ulp). *For a non-zero real number x we define* $\mathrm{ulp}(x) := 2^{E(x)-p}$.

For a real $x \neq 0$ and a working precision p we always have $2^{p-1}\mathrm{ulp}(x) \leq |x| < 2^p \mathrm{ulp}(x)$. If x is a floating-point number, then $\mathrm{ulp}(x)$ is the weight of the least significant bit — zero or not — in the p-bit mantissa of x. For all real x, $\mathrm{ulp}(x)$ is always greater than zero by definition.

Lemma 3.7. *If $c \neq 0$ and $x \neq 0$ then $c \cdot \mathrm{ulp}(x) < 2 \cdot \mathrm{ulp}(cx)$.*

Proof: if $c < 0$ it is void. By definition of $\mathrm{ulp}(x)$ we have for all $c > 0$:

$$2^{p-1}\mathrm{ulp}(x) \leq |x|$$

and

$$|cx| < 2^p \mathrm{ulp}(cx)$$

so

$$c \cdot 2^{p-1}\mathrm{ulp}(x) \leq |cx| < 2^p \mathrm{ulp}(cx). \quad \square$$

Lemma 3.8. *Assuming no underflow (flush to zero) occurs then in all rounding modes for a non zero real x we have:* $\mathrm{ulp}(x) \leq \mathrm{ulp}(\circ(x))$, *where $\circ(x)$ is the rounding of x in the chosen mode with an unbounded exponent range.*

Proof: we have $2^{E(x)-1} \leq |x| < 2^{E(x)}$ and $\mathrm{ulp}(x) = 2^{E(x)-p}$. After rounding we get $2^{E(x)-1} \leq |\circ(x)| \leq 2^{E(x)}$ since $2^{E(x)}$ and $2^{E(x)-1}$ are exactly representable, therefore $\mathrm{ulp}(\circ(x)) \geq 2^{E(x)-p} \geq \mathrm{ulp}(x)$. $\square$

Lemma 3.9. *Let x a non-zero real and $\circ(x)$ its rounding to nearest on p bits. Then $|x| \leq (1+2^{-p})|\circ(x)|$.*

Proof: by definition of rounding to nearest we have

$$|x - \circ(x)| \leq \frac{1}{2}\mathrm{ulp}(\circ(x)) \leq \frac{1}{2}2^{1-p}|\circ(x)|,$$

$$|x| \leq |\circ(x)| + 2^{-p}|\circ(x)|. \quad \square$$

[1] unit in the last place

Lemma 3.10. *Let a and b be two non-zero floating-point numbers of the same sign and precision p then in all rounding modes*

$$\mathrm{ulp}(a) + \mathrm{ulp}(b) \leq \frac{3}{2}\mathrm{ulp}(\circ(a+b)).$$

PROOF: It suffices to consider the case where a and b are positive. The definition of ulp gives:

$$2^{p-1}\mathrm{ulp}(a) \leq a < 2^p\mathrm{ulp}(a),$$

$$2^{p-1}\mathrm{ulp}(b) \leq b < 2^p\mathrm{ulp}(b)$$

thus

$$2^{p-1}[\mathrm{ulp}(a) + \mathrm{ulp}(b)] \leq a+b < 2^p[\mathrm{ulp}(a) + \mathrm{ulp}(b)].$$

If $\mathrm{ulp}(a) = \mathrm{ulp}(b)$ we get

$$2^p\mathrm{ulp}(a) \leq a+b < 2^{p+1}\mathrm{ulp}(a)$$

and therefore $\mathrm{ulp}(\circ(a+b)) \geq \mathrm{ulp}(a+b) \geq 2\mathrm{ulp}(a) = \mathrm{ulp}(a) + \mathrm{ulp}(b)$ (Lemma 3.8) and the lemma holds.

Otherwise we can assume without loss of generality that $\mathrm{ulp}(a) > \mathrm{ulp}(b)$, that is $\mathrm{ulp}(a) \geq 2 \cdot \mathrm{ulp}(b)$. We deduce:

$$\mathrm{ulp}(a) + \mathrm{ulp}(b) \leq \frac{3}{2}\mathrm{ulp}(a),$$

and together with $\mathrm{ulp}(\circ(a+b)) \geq \mathrm{ulp}(a+b) \geq \mathrm{ulp}(a)$ (Lemma 3.8) this concludes the proof. □

EXAMPLE: Let $p = 4$ and chose rounding to nearest: $a = 1.010$, $b = 0.1001$ in binary notation.
$a + b = 1.1101$, $\circ(a+b) = 1.110$,
$\mathrm{ulp}(a) + \mathrm{ulp}(b) = 2^{-3} + 2^{-4} = \frac{3}{2}2^{-3} = \frac{3}{2}\mathrm{ulp}(\circ(a+b))$.

3.5 Roundoff errors

In order to provide an error bound on the numerical result given by the Newton-Cotes method, we need to have a step-by-step look into Algorithm 2.

This step is often neglected when doing numerical integration, where error analysis stops right after stating the well known bound for the mathematical error. In fact, the experiment illustrated in Figure 3 shows that much remains to be done to control the error on the result.

For this section we denote by $\widehat{x}$ the value actually computed (i.e. with all roundoff errors) for a given "exact" value x, as would be computed with an infinite precision from the beginning of the algorithm.

In addition to the parameters of algorithm 2 we need an upper bound M of $|f^{(n)}|$ on $[a,b]$ if n is even, or an upper bound of $|f^{(n+1)}|$ if n is odd; p is the working precision expressed in

Algorithm 2 Newton-Cotes integration

INPUT: $\widehat{a}, \widehat{b}, f, n$.
OUTPUT: $\widehat{I}$.

for $i \leftarrow 0$ to $n-1$ **do**
    $t \leftarrow \circ(\widehat{a} * (n-i-1))$
    $u \leftarrow \circ(\widehat{b} * i)$
    $x \leftarrow \circ(t+u)$
    $x \leftarrow \circ(x/(n-1))$
    $y_i \leftarrow \circ(f(x))$
    $y_i \leftarrow \circ(y_i * n_i)$
end for
$S \leftarrow \mathrm{sum}(y_i, i = 0 \ldots n-1)$ $\triangleright$ with Demmel and Hida algorithm [2]
$U \leftarrow \circ(S/d(n-1))$
$D \leftarrow \circ(b-a)$
return $\circ(DU)$

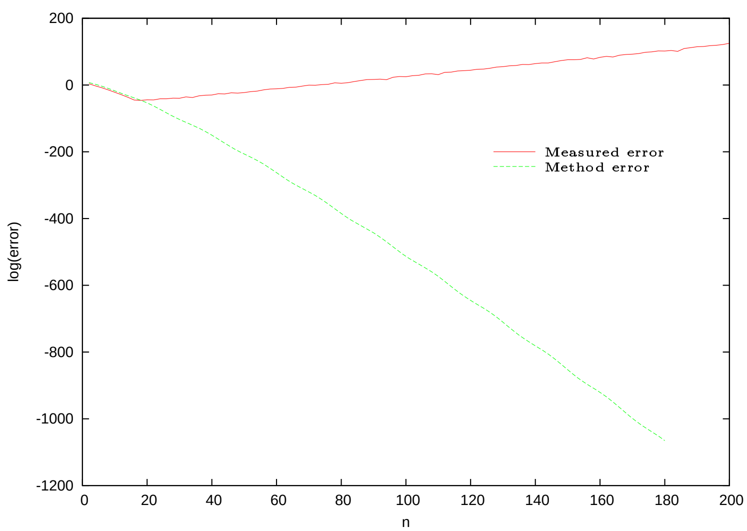

Figure 3: An example of error measurement for the Newton-Cotes method for $f : x \mapsto x^2$, $[a, b] = [0, 9]$. Computations were done with the default *double* precision of 53 bits, n the number of points is displayed on the abscissa, and in ordinates the base-2 logarithm of the error.

the number of bits of the mantissa, a and b rounded to nearest p-bit floating-point numbers giving $\widehat{a}$ and $\widehat{b}$; m an upper bound of $|f'|$ on $[a,b]$. In the rest of this section we will prove our main theorem:

Theorem 3.11. *When computing the numerical quadrature of f using Algorithm 2 the total error on the result is bounded by:*

$$\begin{aligned} E_{tot} &= \left(\frac{45}{2} + 21 \cdot 2^{-p}\right)\mathrm{ulp}(\widehat{I}) + \frac{1}{2}|\widehat{U}| \left(\mathrm{ulp}(\widehat{b}) + \mathrm{ulp}(\widehat{a})\right) \\ &\quad + 3\frac{(1+2^{-p})n \cdot \widehat{D}}{d(n-1)} \max(\delta_{\widehat{y}_i}) + \begin{cases} \frac{1}{8}\left(\frac{b-a}{n-1}\right)^{n+2} M & \textit{if } n \textit{ is odd,} \\ \frac{1}{4}\left(\frac{b-a}{n-1}\right)^{n+1} M & \textit{otherwise.} \end{cases} \end{aligned}$$

The algorithm can be analyzed in several steps:

1. the computation of the weights w_i, $i \in [0, n-1]$ of the method. For Newton-Cotes, those weights are rational and computed exactly : $w_i = \frac{n_i}{d}$ where $n_i, d \in \mathbb{Z}$, so no rounding error occurs at this step.

2. the computation of x_i. This is done at line 5 of Algorithm 2:

$$\widehat{x}_i = \circ\left(\frac{\circ(\circ((n-1-i)\cdot\widehat{a}) + \circ(i \cdot \widehat{b}))}{n-1}\right).$$

In order to simplify the notations we write $t = (n-i-1)a$, $u = i \cdot b$ and their inexact counterparts $\widehat{t} = \circ((n-i-1)\widehat{a})$, $\widehat{u} = \circ(i\,\widehat{b})$. If $b = 0$ or $i = 0$ the error on $\widehat{u}$ is zero. Otherwise the error estimation yields:

$$\begin{aligned} |\circ(i \cdot \widehat{b}) - ib| &\le \tfrac{1}{2}\mathrm{ulp}(\circ(i\widehat{b})) + \tfrac{i}{2}\mathrm{ulp}(\widehat{b}) \\ &\le \tfrac{3}{2}\mathrm{ulp}(\circ(i\widehat{b})) = \tfrac{3}{2}\mathrm{ulp}(\widehat{u}). \quad \text{[Lemmas 3.7 and 3.8]} \end{aligned}$$

Similarly if $a = 0$ or $n - i - 1 = 0$ the error on $\widehat{t}$ is zero, and otherwise we get $|\widehat{t} - t| \le \frac{3}{2}\mathrm{ulp}(\widehat{t})$.

Let us now assume that a and b have the same sign (which can be zero); this leads easily to the fact that $\widehat{t}$ and $\widehat{u}$ have the same sign (this assumption is necessary to be able to use Lemma 3.10). If this is not the case we can split the integration interval at 0. Moreover assume without loss of generality that $0 \le a < b$, which gives $0 \le \widehat{a} \le \widehat{b}$.

$$\begin{aligned} |\circ(\widehat{t} + \widehat{u}) - (t+u)| &\le \tfrac{1}{2}\mathrm{ulp}(\circ(\widehat{t}+\widehat{u})) + \tfrac{3}{2}(\mathrm{ulp}(\widehat{t}) + \mathrm{ulp}(\widehat{u})) \\ &\le \tfrac{11}{4}\mathrm{ulp}(\circ(\widehat{t}+\widehat{u})). \quad \text{[Lemma 3.10]} \end{aligned}$$

Taking into account the error coming from the division by $n-1$ we get:

$$\begin{aligned}\delta_{\widehat{x}_i} = |x_i - \widehat{x}_i| &\leq \tfrac{1}{2}\mathrm{ulp}(\widehat{x}_i) + \tfrac{11}{4(n-1)}\mathrm{ulp}(\circ(\widehat{c}+\widehat{d})) \\ &\leq \tfrac{1}{2}\mathrm{ulp}(\widehat{x}_i) + \tfrac{11}{2}\mathrm{ulp}(\widehat{x}_i) \quad \text{[Lemmas 3.7 and 3.8]} \\ &\leq 6 \cdot \mathrm{ulp}(\widehat{x}_i).\end{aligned}$$

3. the computation of $f(x_i)$. We assume we have an implementation of f with correct rounding, and we call the function f requesting the rounding to nearest of the exact value with precision p. Such correctly rounded implementations of mathematical functions with arbitrary precision on the result can be found for example in MPFR [7] for non-trivial functions like exp, sin, arctan and numerous others.

   With the already estimated error on $\widehat{x}_i$ we have:

   $$|f(\widehat{x}_i) - f(x_i)| = |f'(\theta_i)(\widehat{x}_i - x_i)|, \quad \theta_i \in [\min(x_i,\widehat{x}_i), \max(x_i,\widehat{x}_i)]$$

   and with an upper bound on f' we can bound this error absolutely. Let $\widehat{f}_i = \circ(f(\widehat{x}_i))$ be the floating-point number computed. At this step we now have:

   $$\begin{aligned}|\widehat{f}_i - f(x_i)| &\leq |f'(\theta_i)(\widehat{x}_i - x_i)| + \tfrac{1}{2}\mathrm{ulp}(\widehat{f}_i) \\ &\leq 6m \cdot \mathrm{ulp}(\widehat{x}_i) + \tfrac{1}{2}\mathrm{ulp}(\widehat{f}_i).\end{aligned}$$

4. computation of the $y_i = f(x_i) \cdot n_i$. The accumulated error so far:

   $$\begin{aligned}\delta_{\widehat{y}_i} = |\widehat{y}_i - y_i| &\leq |n_i| \cdot |\widehat{f}_i - f_i| + \tfrac{1}{2}\mathrm{ulp}(\widehat{y}_i) \\ &\leq 6|n_i| \cdot m \cdot \mathrm{ulp}(\widehat{x}_i) + \tfrac{|n_i|}{2}\mathrm{ulp}(\widehat{f}_i) + \tfrac{1}{2}\mathrm{ulp}(\widehat{y}_i) \\ &\leq 6|n_i| \cdot m \cdot \mathrm{ulp}(\widehat{x}_i) + \tfrac{3}{2}\mathrm{ulp}(\widehat{y}_i). \quad \text{[Lemmas 3.7 and 3.8]}\end{aligned}$$

   Remark: when bounding the error on $\widehat{x}_i$, $\widehat{f}_i$ as well as $\widehat{y}_i$, the term with $\mathrm{ulp}(\widehat{x}_i)$ vanishes if the error on $\widehat{x}_i$ is zero. One can easily show that with our assumption that no underflow occurs, if $\widehat{x}_i = 0$ then the error on $\widehat{x}_i$ is zero (i.e. $x_i = 0$) and the ill-defined quantity $\mathrm{ulp}(\widehat{x}_i)$ vanishes. For the error bound we keep track of only $\max(\delta_{\widehat{y}_i})$.

5. summation of the y_i's: this is done with Demmel and Hida summation algorithm[2], which guarantees an error of at most 1.5 ulp on the final result. This algorithm uses a larger working precision $p' \approx p + \log_2(n)$. Let $S = \sum_{i=0}^{n-1} y_i$.

   $$|\widehat{S} - S| \leq \frac{3}{2}\mathrm{ulp}(\widehat{S}) + n \cdot \max(\delta_{\widehat{y}_i}).$$

6. division of S by $d(n-1)$: $U = \frac{S}{d(n-1)}$. The computation of $d(n-1)$ is done with integer arithmetic and is exact. The error at this step is thus:

   $$\begin{aligned}|\widehat{U} - U| &\leq \tfrac{1}{2}\mathrm{ulp}(\widehat{U}) + \tfrac{3}{2d(n-1)}\mathrm{ulp}(\widehat{S}) + \tfrac{n}{d(n-1)}\max(\delta_{\widehat{y}_i}) \\ &\leq \tfrac{7}{2}\mathrm{ulp}(\widehat{U}) + \tfrac{n}{d(n-1)}\max(\delta_{\widehat{y}_i}). \quad \text{[Lemmas 3.7 and 3.8]}\end{aligned}$$

7. multiplication by $b-a$: $I=(b-a)U$. We note $D=b-a$ and $\widehat{D}=\circ(\widehat{b}-\widehat{a})$.

$$|\widehat{D}-D| \quad \leq \quad \tfrac{1}{2}\left[\mathrm{ulp}(\widehat{D})+\mathrm{ulp}(\widehat{a})+\mathrm{ulp}(\widehat{b})\right].$$

We have by hypothesis

$$\begin{array}{rcl}\widehat{b} & \geq & b-\frac{1}{2}\mathrm{ulp}(\widehat{b}),\\ \widehat{a} & \leq & a+\frac{1}{2}\mathrm{ulp}(\widehat{a})\end{array}$$

where $\mathrm{ulp}(0)=0$ by convention and therefore

$$\begin{array}{rcl}\widehat{b}-\widehat{a} & \geq & b-a-\frac{1}{2}\left(\mathrm{ulp}(\widehat{a})+\mathrm{ulp}(\widehat{b})\right)\\ & \geq & b-a-\mathrm{ulp}(\widehat{b}).\end{array}$$

On the other hand we know $\widehat{b}>\widehat{a}$, so we have $\widehat{b}-\widehat{a}\geq\max(\frac{1}{2}\mathrm{ulp}(\widehat{b}),b-a-\mathrm{ulp}(\widehat{b}))$ which gives $\mathrm{ulp}(\widehat{b})\leq 2(\widehat{b}-\widehat{a})$. Then

$$D\leq\widehat{b}-\widehat{a}+\mathrm{ulp}(\widehat{b})\leq 3(\widehat{b}-\widehat{a})\leq 3(1+2^{-p})\widehat{D}. \quad \text{[Lemma 3.9]} \tag{3}$$

If we put all the results and bounds gathered so far, we can reach the following final error on $\widehat{I}=\circ(\widehat{D}\widehat{U})$:

$$\begin{array}{rcl}|\widehat{I}-I| & \leq & \frac{1}{2}\mathrm{ulp}(\widehat{I})+|\widehat{D}\widehat{U}-D\cdot U|\\ & \leq & \frac{1}{2}\mathrm{ulp}(\widehat{I})+|\widehat{U}|\cdot|\widehat{D}-D|+|D|\cdot|\widehat{U}-U|\\ & \leq & \frac{1}{2}\mathrm{ulp}(\widehat{I})+|\widehat{U}|\cdot|\widehat{D}-D|+3(1+2^{-p})|\widehat{D}|\cdot|\widehat{U}-U| \quad \text{[Inequality (3)]}\\ & \leq & \frac{1}{2}\mathrm{ulp}(\widehat{I})+|\widehat{U}|\cdot|\widehat{D}-D|+\\ & & 3(1+2^{-p})|\widehat{D}|\left(\frac{7}{2}\mathrm{ulp}(\widehat{U})+\frac{n}{d(n-1)}\max(\delta_{\widehat{y}_i})\right)\\ & \leq & (\frac{43}{2}+21\cdot 2^{-p})\mathrm{ulp}(\widehat{I})+|\widehat{U}|\cdot|\widehat{D}-D|\\ & & +3\frac{(1+2^{-p})n\cdot\widehat{D}}{d(n-1)}\max(\delta_{\widehat{y}_i}) \quad \text{[Lemmas 3.7 and 3.8]}\\ & \leq & (\frac{45}{2}+21\cdot 2^{-p})\mathrm{ulp}(\widehat{I})+\frac{1}{2}|\widehat{U}|\left(\mathrm{ulp}(\widehat{b})+\mathrm{ulp}(\widehat{a})\right)\\ & & +3\frac{(1+2^{-p})n\cdot\widehat{D}}{d(n-1)}\max(\delta_{\widehat{y}_i}). \quad \text{[Lemmas 3.7 and 3.8]}\end{array}$$

This bound for the error is satisfactory for using it in the algorithm, because it is made of quantities that we can compute before the algorithm itself (p, n), or which are naturally computed in the flow of the algorithm ($\widehat{I}$, $\widehat{U}$, $\widehat{b}$, $\widehat{a}$, $\widehat{D}$, d, $\delta_{\widehat{y}_i}$).

For the final error bound we need to add a bound on the mathematical error:

$$E_{\mathrm{math}}\leq\begin{cases}\dfrac{1}{8}\left(\dfrac{b-a}{n-1}\right)^{n+2}M & \text{if } n \text{ is odd,}\\ \dfrac{1}{4}\left(\dfrac{b-a}{n-1}\right)^{n+1}M & \text{otherwise}\end{cases} \tag{4}$$

which is easily computed as well. In the course of the program we are able to chose the rounding mode for every computation. This allows us to use pessimistic rounding modes and thus avoid the problem of roundoff errors in the computation of the error bound itself. □

4 Experiments

Algorithm 2 was implemented using the MPFR library [7]. In addition to the result of the integration, the program gives an error bound on the computed result split in four terms:

1. the mathematical error, whose expression is given in equation 4,
2. the "static" error $E_{\text{stat}} = (\frac{45}{2} + 21 \cdot 2^{-p})\text{ulp}(\widehat{I})$,
3. the "difference" error $E_{\text{diff}} = \frac{1}{2}|\widehat{U}| \left(\text{ulp}(\widehat{b}) + \text{ulp}(\widehat{a}) + \text{ulp}(\widehat{D})\right)$,
4. the evaluation error $E_{\text{eval}} = 3\frac{(1+2^{-p})n \cdot \widehat{D}}{d(n-1)}\max(\delta_{\widehat{y}_i})$.

For our experiments we chose a function and an integration domain where the exact value is known, so that we can measure precisely the actual error of the computation (denoted by E_{meas}). Figure 4 shows the different errors when computing the integral $I = \int_0^3 e^x dx$ with 113 bits of working precision, the number of evaluation points varying from 2 to 100.

The dominating error source is always the evaluation error E_{eval}. The mathematical error decreases rapidly but it appears clearly that it is well compensated by the roundoff error as soon as more than about 10 evaluation points are used, for the considered function and parameters. The theoretical gain of increasing the order of the method is lost. Figure 5 gives the smallest value of the number of points for which the mathematical error is inferior to the sum of all other error terms, for different working precisions chosen. This is commonly interpreted as the optimal value of n in the following sense: for higher values of n the benefit of an higher order method is lost in the noise of the roundoff error, and for smaller values the accuracy on the evaluation of the function is not exploited to its fullest. Although the coefficients generating algorithm is slow for high values of n, no particular attempts were made to optimize it yet; this is motivated partly by the slow growth in Figure 5 (the other reasons being the numerical instability discussed below as well as the possibility to use composition).

The bound on the total error as given by the algorithm is somehow close to the measured error. In the experimental data we observe a maximal ratio of about 46000 — which seems to be huge, but with a logarithmic scale it means we lost a mere 16 bits of precision by our estimation. In particular this means our algorithm is not too grossly pessimistic.

The numerical instability of the method when n grows is not surprising, and no news either. The fact that negative coefficients appear in the formula as soon as $n \geq 8$ partly explains this fact which is demonstrated here. Considering the smoothness of the function chosen for the experiment, the instability is to be attributed to the method. Small values of n are therefore recommended for the Newton-Cotes quadrature method.

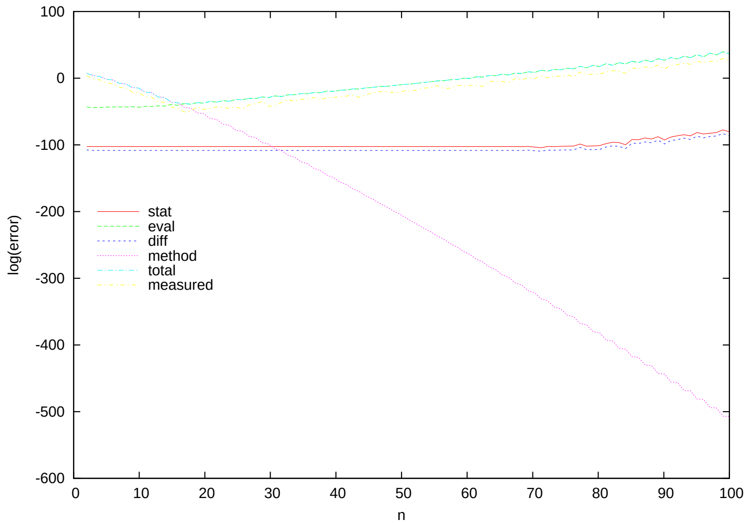

Figure 4: The different errors while computing $\int_0^3 e^x dx$ with 113 bits of precision.

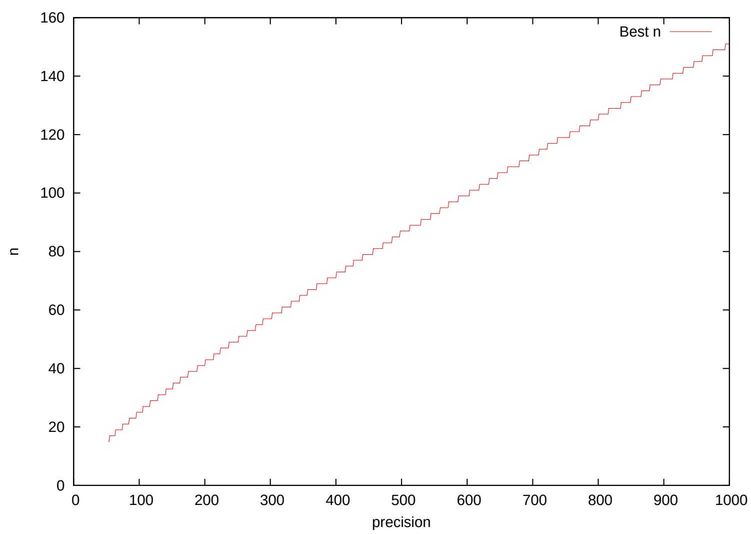

Figure 5: Optimal values of the number n of points for several working precisions (experimental data gathered with $\int_0^3 e^x dx$).

5 Conclusion and future work

The Newton-Cotes quadrature scheme is the simplest numerical quadrature method, which made it the natural candidate for a detailed study. We were able to provide a rigorous analysis of the method that is self-contained and covers every aspect that is relevant to an implementation, that is, the description of the algorithms and the establishment of proven error bounds.

However the Newton-Cotes family of quadrature methods were not a goal *per se* but rather a proof of concept that such a study of the error is feasible and indeed desirable. It is planned to perform the same kind of work with other quadrature schemes, notably the Gauss-Legendre methods (which have order $2n$ for n points and are numerically more stable). These analyses might serve as a mathematical foundation of a correctly rounded quadrature library.

References

[1] DEFOUR, D., DE DINECHIN, F., AND MULLER, J.-M. Correctly rounded exponential function in double precision arithmetic. Research Report 2001-26, Laboratoire de l'Informatique du Parallélisme, 2001. 21 pages.

[2] DEMMEL, J., AND HIDA, Y. Accurate floating point summation. `http://www.cs.berkeley.edu/~demmel/AccurateSummation.ps`, May 2002.

[3] FOUSSE, L., AND ZIMMERMANN, P. Accurate summation: Towards a simpler and formal proof. In *Proceedings of the RNC'5 conference (Real Numbers and Computers* (Sept. 2003). `http://www.ens-lyon.fr/LIP/Arenaire/RNC5`.

[4] IEEE standard for binary floating-point arithmetic. Tech. Rep. ANSI-IEEE Standard 754-1985, New York, 1985. approved March 21, 1985: IEEE Standards Board, approved July 26, 1985: American National Standards Institute, 18 pages.

[5] LEFÈVRE, V., AND MULLER, J.-M. The table maker's dilemma: our search for worst cases. `http://www.ens-lyon.fr/~jmmuller/Intro-to-TMD.htm`.

[6] LEFÈVRE, V., AND MULLER, J.-M. Worst cases for correct rounding of the elementary functions in double precision. In *Proceedings of the 15th IEEE Symposium on Computer Arithmetic (ARITH'15)* (2001), N. Burgess and L. Ciminiera, Eds., IEEE Computer Society, pp. 111–118.

[7] THE SPACES PROJECT. The MPFR library, version 2.1.1. `http://www.mpfr.org/`, 2005.

[8] PHILIP J. DAVIS AND PHILIP RABINOWITZ Methods of numerical integration. 1984

[9] WALTER OEVEL Numerical Computations in MuPAD 1.4 In *mathPAD vol 8 No 1*, 1998.

[10] F. Jezequel, M. Charikhi, J.-M. Chesneaux Dynamical control of computations of multiple integrals SCAN2002 conference, Paris (France) 23-27 september 2002.

[11] David H. Bailey, Xiaoye S. Li A Comparison of Three High-Precision Quadrature Schemes In Proceedings of Real Numbers And Computers (RNC5), Lyon (France) 3-5 september 2003

Unité de recherche INRIA Lorraine
LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399