A Dynamic Precision Floating-Point Arithmetic based on the Infinity Computer Framework*†

Pierluigi Amodio[1][0000−0003−3372−0162], Luigi Brugnano[2][0000−0002−6290−4107], Felice Iavernaro[1][0000−0002−9716−7370]✉, and Francesca Mazzia[3][0000−0003−1072−9578]

[1] Dipartimento di Matematica, Università di Bari, Italy
{pierluigi.amodio,felice.iavernaro}@uniba.it
[2] Dipartimento di Matematica e Informatica "U. Dini", Università di Firenze, Italy
luigi.brugnano@unifi.it
[3] Dipartimento di Informatica, Università di Bari, Italy
francesca.mazzia@uniba.it

Abstract. We introduce a dynamic precision floating-point arithmetic based on the Infinity Computer. This latter is a computational platform which can handle both infinite and infinitesimal quantities epitomized by the positive and negative finite powers of the symbol ①, which acts as a radix in a corresponding positional numeral system. The idea is to use the positional numeral system from the Infinity Computer to devise a variable precision representation of finite floating-point numbers and to execute arithmetical operations between them using the Infinity Computer Arithmetics. Here, numerals with negative finite powers of ① will act as infinitesimal-like quantities whose aim is to dynamically improve the accuracy of representation only when needed during the execution of a computation. An application to the iterative refinement technique to solve linear systems is also presented.

Keywords: Infinity Computer · Floating-point arithmetic · conditioning · iterative refinement.

1 Introduction

The Infinity Computer paradigm, patented in EU, USA, and Russia (see for example [16]), is based on a positional numeral system with the infinite radix ① (called grossone) representing, by definition, the number of elements of the set of natural numbers $\mathbb{N}$ [11, 14]. A number in this system is a linear combination of powers of ① with coefficients in the standard numeral system, such as

$$d_{p_m}①^{p_m}\dots d_{p_1}①^{p_1}d_{p_0}①^{p_0}d_{p_{-1}}①^{p_{-1}}\dots d_{p_{-k}}①^{p_{-k}},$$

* This work was funded by the INdAM-GNCS 2018 Research Project "Numerical methods in optimization and ODEs".
† Accepted version of the paper https://doi.org/10.1007/978-3-030-40616-5_22

with the usual meaning

$$d_{p_m}①^{p_m} + \ldots + d_{p_1}①^{p_1} + d_{p_0}①^{p_0} + d_{p_{-1}}①^{p_{-1}} + \ldots + d_{p_{-k}}①^{p_{-k}}.$$

The numerals $d_i \neq 0$ belong to a traditional numeral system and are called *grossdigits*, while numerals p_i are sorted in the decreasing order with $p_0 = 0$

$$p_m > p_{m-1} > \ldots > p_1 > p_0 > p_{-1} > \ldots p_{-(k-1)} > p_{-k},$$

and are called *grosspowers* (only finite grosspowers are considered in this contribution). Among the many fields of research this new methodology has been successfully applied, we mention *numerical differentiation and optimization* [4, 12, 17] and *numerical solution of differential equations* [9, 1, 10, 6].[4] First results on handling ill-conditioning using the Infinity Computer may be found in [5, 13].

Of particular interest in our study are grossnumbers consisting of a finite expansion of integer grosspowers such as, for example,

$$X = ①^P \sum_{j=0}^{T} x_j ①^{-j}, \qquad \text{with grossdigits} \quad x_j = \pm\beta^{p_j} \sum_{i=0}^{t} d_{ij}\beta^{-i}, \tag{1}$$

where $P, p_j \in \mathbb{Z}$ and T, t are given positive integers, while β stands for the base of the traditional floating-point arithmetic system (usually $\beta = 2$).

This representation suggests interesting applications of the Infinity Computer if now ① identifies a suitable prescribed finite value. The idea is to exploit the grossdigits x_i in order to store a large number of significant digits in a dynamic manner during the execution of an algorithm. This means that the accuracy may be increased/decreased on demand during the flow of computations by automatically activating/deactivating a number of negative grosspowers. Taking aside the technical aspects related to the hardware implementation of the Infinity Computer, our study explores this path of investigation and is addressed to the accurate solution of ill-conditioned/unstable problems [7, 3].

It should be noticed that, in principle, neither the user nor the programmer needs to know what the value of ① actually is. This assumption should be instead understood as an inherent feature of the machine architecture which, consistently with the Infinity Arithmetic methodology, will perceive the negative powers of ① as infinitesimal-like quantities if related to the classical floating-point system. Adopting this point of view, it turns out that changing the meaning of ① as we are going to do in the present work, does not affect that much the philosophical principles the grossone methodology is rooted in.

2 The framework

In this section, we discuss an application of the Infinity Arithmetic system which consists in devising a floating-point variable precision arithmetic that will be

[4] For further references and applications see the survey [15].

used later to overcome the intrinsic loss of accuracy experienced when solving an ill-conditioned problem in the standard floating-point arithmetic.

The implementation of multiple and, in particular, variable precision arithmetic has been largely explored and successfully implemented, as is testified by the rich literature on this topic (see, for example, the survey paper [2] and reference therein). Here, however, we explore a further generalization of this paradigm in that the number of significant digits needs not to be a priori specified and maintained fixed but may be dynamically changed during the flow of computations. In particular, depending on the specific problem at hand and the algorithm used to solve it, the involved variables may allocate a different and variable amount of memory during the execution of the algorithm. The final goal is to control the error and make sure that the desired accuracy in the output data is achieved. This dynamic precision arithmetic is introduced as a natural byproduct of the Infinity Computer architecture and thus is expected to be easy to handle and to perform efficiently, once a hardware implementation of this latter will be available.

Representation of a real number. Consider first the problem of storing a real non-zero number $x = \pm\beta^p \sum_{i=0}^{\infty} d_i \beta^{-i} \in \mathbb{R}$, $d_0 \neq 0$, by preserving $N+1 > 0$ significant digits. This task can be accomplished by setting in (1): $①\doteq\beta^{t+1}$ and assuming $p_j = p, j = 0, \ldots, T$, and $P = 0$.[5] The first $N+1$ digits of x may be gathered in adjacent groups of $t+1$ elements as follows:

$$x = \pm\beta^p \underbrace{d_0.d_1 \ldots d_t}\, \underbrace{d_{t+1} \ldots d_{2t+1}} \ldots \underbrace{d_{j(t+1)} \ldots d_{(j+1)(t+1)-1}} \ldots d_N. \tag{2}$$

Then, assuming that $(N+1) \leq (T+1)(t+1)$ and setting $d_i = 0$ for $i = N+1, \ldots, (T+1)(t+1)$, the floating-point grossnumber representing x takes the form

$$\mathrm{fl}(x) \;=\; \pm\beta^p \sum_{j=0}^{T} ①^{-j} \sum_{i=0}^{t} d_{j(t+1)+i} \beta^{-i}. \tag{3}$$

Notice that all grossdigits share the same exponent p which, therefore, could be stored only once. We call (3) the *normalized* machine representation of x and its uniqueness comes from the uniqueness of the standard normalized notation (2).

Renormalization after a computation. According to the Infinity Arithmetic methodology, the four basic operations over two grossnumbers follow the same rules of operations with polynomials. In fact, by definition, in this numeral system the radix ① is infinite while all digits d_{ij} are finite. For example, given the two grossnumbers (see (1))

$$X = x_1①^1 + x_2①^0 + x_3①^{-1}, \qquad Y = y_1①^0 + y_2①^{-1} + y_3①^{-2}$$

we get

$$X + Y = x_1①^1 + (x_2 + y_1)①^0 + (x_3 + y_2)①^{-1} + y_3①^{-2} \tag{4}$$

[5] $\doteq$ denotes the identification operation, so the meaning of ① remains unaltered.

and

$$X \cdot Y = x_1y_1①^1 + (x_1y_2 + x_2y_1)①^0 + (x_1y_3 + x_2y_2 + x_3y_1)①^{-1} \\ +(x_2y_3 + x_3y_2)①^{-2} + x_3y_3①^{-3}. \tag{5}$$

Now, due to the identification of ① with a finite number and the assumption that all grossdigits must share the same exponent, it follows that, in general, the result of an operation will be not normalized and thus it is necessary to carry forward or backward some digits along the powers of ① in order to obtain the result in the form (3). Without entering into details of this normalization procedure, which would go beyond the aims of this short paper, we consider an illustrative example.

Example 1. Set $\beta = 2$ (binary base) and $t = 3$ (four significant digits). Consider the sum of the two floating-point normalized grossnumbers

$$X = 2^0 \cdot (①^0 1.101 + ①^{-1} 1.010 + ①^{-2} 1.111),$$
$$Y = 2^{-2} \cdot (①^0 1.011 + ①^{-1} 1.110 + ①^{-2} 1.001).$$

The procedure, which moves along similar lines as for standard floating-point arithmetic, is summarized by the following steps of obvious meaning:

		$①^0$	$①^{-1}$	$①^{-2}$	
(a) alignment	2^0	1.101	1.010	1.111	
	2^0	0.010	1.111	1.010	01
(b) sum with carrying	2^0	10.000	1.010	1.001	01
(c) normalization	2^1	1.000	0.101	0.101	

Notice that, as explained above, the Infinity Computer would perform the addition without carrying. This means that step (b) needs to be suitably arranged, also considering how to manage the rounding effects in each floating-point grossdigit. This aspects needs a specific study and is not addressed here since we just intend to show the general lines of our apprach. As a further remark, one clear advantage arising from the use of the Infinity Computer is that the computation of the grossdigits outcoming from basic operations such as (4) and (5) may be carried out in parallel.

Dynamic precision usage. Among the features offered by the computational platform based on the Infinity Computer, we assume that the user may decide how many infinitesimals stored in a variable should be involved in a given computation. If X is chosen as in (1), we denote by $X^{(q)}$ the grossnumber obtained by neglecting, in the sum, all the infinitesimals of order greater than q, that is, $X^{(q)} = ①^P \sum_{j=0}^{q} x_j ①^{-j}$. For example, choosing X and Y as in Example 1, we see that $X^{(0)} + Y^{(0)} = 2^1 \cdot 1.000$ would become the standard floating-point addition, and one can improve the accuracy by letting the subsequent infinitesimals come into play. This possibility may be exploited in a dynamical manner to overcome ill-conditioning issues associated with a given problem. The following example has a heuristic purpose in this direction.

Example 2. Set $\beta = 10$ (decimal base) and $t = 3$ (four significant digits), without rounding. Consider the three grossnumbers

$$X = 10^0 \cdot (①^0 1.234 + ①^{-1} 5.678 + ①^{-2} 9.012 + ①^{-3} 3.456),$$
$$Y = 10^0 \cdot (①^0 1.234 + ①^{-1} 4.444 + ①^{-2} 4.444 + ①^{-3} 4.444),$$
$$Z = 10^{-4} \cdot (①^0 1.230 + ①^{-1} 1.234 + ①^{-2} 1.234 + ①^{-3} 1.234).$$

storing three corresponding decimal numbers, say $x, y, z \in \mathbb{R}$ with 16 significant digits. Then $X^{(0)}$,$Y^{(0)}$ and $Z^{(0)}$ may be interpreted as the single precision representation of x, y, z while, on the other side, X, Y and Z are their quadruple precision approximations. Consider the problem of computing $w = (x - y) - z$ with four significant digits, which would suggest the use of single precision. Unfortunately, the first subtraction $x - y$ is ill-conditioned in single precision:

$$\left| \frac{(x-y) - (X^{(0)} - Y^{(0)})}{x - y} \right| = \left| \frac{(X-Y) - (X^{(0)} - Y^{(0)})}{X - Y} \right| = 1. \tag{6}$$

The following scheme illustrates the procedure to obtain the correct result while minimizing the computational effort. It goes without saying that a cheap estimation of the relative error be available.[6] However, for simplicity of exposition, we evaluate the error by exploiting formulae similar to (6).

steps	error	action
(a) $X^{(0)} - Y^{(0)} = 0$	$1.9 \cdot 10^{-1}$	improve the accuracy
(b) $X^{(1)} - Y^{(1)} = 1.234 \cdot 10^{-4}$	$3.7 \cdot 10^{-4}$	accept the result
(c) $(X^{(1)} - Y^{(1)}) - Z^{(0)} = 4.000 \cdot 10^{-7}$	$7.7 \cdot 10^{-2}$	improve the accuracy
(d) $X^{(2)} - Y^{(2)} = 1.2344568 \cdot 10^{-4}$	$8.0 \cdot 10^{-9}$	
(e) $(X^{(2)} - Y^{(2)}) - Z^{(1)} = 4.333 \cdot 10^{-7}$	$5.1 \cdot 10^{-6}$	accept the result

Steps (a)-(b) produce four significant digits in the difference $X - Y$. However, a new cancellation phenomenon occurs at step (c). To overcome the loss of significant digits at this stage, a further improvement in accuracy of $X - Y$ is required (step (d)). The final step (e) reveals the coexistence of variables combined with different precisions.

In general, improving the accuracy of variables results in an increase of the overall computational complexity. However, it turns out that, in certain situations, algorithms may devised where only a marginal amount of computation needs to be performed with high accuracy. One such example is the iterative refinement and will be considered in the next section to illustrate the idea.

3 A case study

Citing Cleve Moler [8], *iterative refinement reduces the roundoff errors in the computed solution to a system of linear equations.* Starting from an initial approximated solution x_0 to a linear system $Ax = b$, this procedure consists of three steps iteratively executed. For $k = 0, 1, \ldots,$

[6] For example, one could take the values $X^{(k+1)}$, $Y^{(k+1)}$ and $Z^{(k+1)}$ as a reference solution with respect to $X^{(k)}$, $Y^{(k)}$ and $Z^{(k)}$, and change the procedure accordingly.

step 1: compute the residual $r_k = b - Ax_k$;
step 2: solve the system $Ad_k = r_k$;
step 3: add the correction $x_{k+1} = x_k + d_k$.

In absence of roundoff errors the iteration would converge after one step to the true solution $x^* = A^{-1}b$. As is well-known, the use of finite precision arithmetic causes an amplification of the representation errors of the input data A and b proportional to the condition number $\kappa(A)$ of the coefficient matrix A. It turns out that, if step 1 is performed using a higher precision arithmetic with respect to the standard precision used at steps 2-3, the accuracy of the approximation may be significantly improved. In particular, denoting by ε_1 and ε_2 the round-off units defining the accuracy of the evaluations of steps 2-3 and step 1 respectively, in [8] it is shown that

$$\frac{||x_k - x^*||_\infty}{||x^*||_\infty} \leq (\sigma\kappa_\infty(A)\varepsilon_1)^k + \mu_1\varepsilon_1 + n\mu_2\kappa_\infty(A)\varepsilon_2 \tag{7}$$

where n is the dimension of A and σ, μ_1, μ_2 are suitable positive quantities with $\mu_1, \mu_2 = O(1/(1 - \sigma\kappa_\infty(A)\varepsilon_1))$. Consequently, under the assumption $0 < \sigma\kappa_\infty(A)\varepsilon_1 \ll 1$, we see that μ_1 and μ_2 become of the order of unity and the relative error approaches the size of the greatest round-off unit (that is ε_1 if we assume $\varepsilon_2 \ll \varepsilon_1$). Usually, the LU factorization with partial pivoting of matrix A is used at step 2 to reduce the computational effort associated with the linear systems to be solved at each iteration. During the execution of the algorithm on the Infinity Computer, one can control the convergence of the scheme by looking at the norms of the residuals computed at step 1:

- In the unfortunate event that $||r_k||$ diverges, the algorithm should improve the overall accuracy of step 2 by involving suitable negative powers of ①, thus reducing ε_1 (see (7)). One alternative we adopt in the example below is to compute the LU factorization of A with a higher accuracy and then truncate L and U to the roundoff level ε_1.
- In the case where $||r_k||$ stagnates before the error reaches the desired size, an improvement of the accuracy in performing step 1 is needed to reduce the value of ε_2. This is again accomplished by introducing new negative powers of ① in the representation of the data A and b and in the computation of the residual r_k.

In the following example, the Infinity Computer arithmetic has been emulated in the Matlab environment by using $\beta = 2$ and $t = 52$, which means that the grossdigits associated with $①^0$ carry the 64-bit base-2 format of the IEEE 754 standard (double precision). This precision is doubled or tripled by involving the grossdigits associated with $①^{-1}$ and $①^{-2}$ respectively.

Example 3. Consider the system $Ax = b$, where A is the Vandermonde matrix of size 25 and coefficients $a_{ij} = (i-1)^{j-1}$, $i, j = 1, \ldots, 25$, while $b = Ae$ with $e = (1, \ldots 1)^\top$, so that $x^* := A^{-1}b = e$. The condition number of A is $\kappa_\infty(A) \approx 8.5 \cdot 10^{39}$. The iterative refinement procedure described above is executed starting

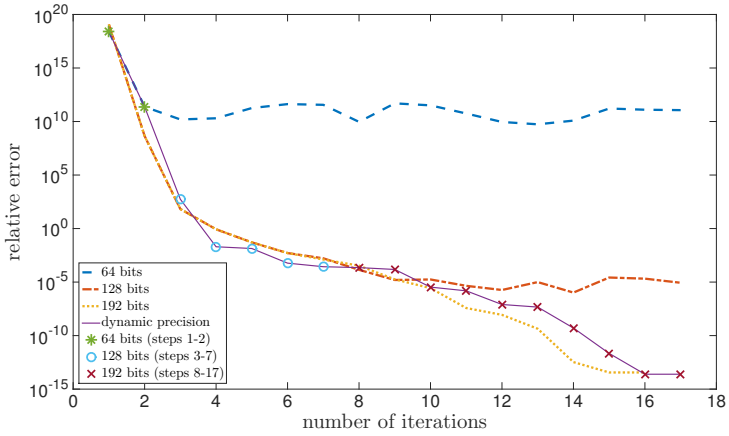

Fig. 1. Fixed versus dynamic precision implementation of the iterative refinement on a linear system with Vandermonde coefficient matrix of size 25.

from the initial guess $x_0 = (0, \ldots, 0)^\top$. We make the choice $\varepsilon_1 = \beta^{-t}/2$ (double precision) and we want to gain as many correct significant digits as possible while dynamically changing ε_2 in order to minimize the overall computational cost. A double precision accurate LU factorization with pivoting has been initially computed to solve the systems at step 2. As for the previous examples, we set $A^{(j)}$ and $b^{(j)}$ the truncations of A and b to the term j in their expansion along the negative grosspowers, and initially perform step 1 with $A^{(0)}$ and $b^{(0)}$ until an increase in $||r_k||$ is detected. In fact, the condition $||r_k|| > ||r_{k-1}||$ is symptomatic that a stagnation of the error is going to occur. This happens at the third iteration: here step 1 is run again with $A^{(1)}$ and $b^{(1)}$ (quadruple precision) in place of $A^{(0)}$ and $b^{(0)}$ and, consequently, the accuracy increases in the subsequent iterations. Notice that r_k has to be truncated at t digits, that is the standard double precision accuracy, before implementing step 2.

A further improvement in the accuracy of r_k is needed at the eighth iteration to avoid the saturation of the error at about 10^{-5}. Therefore $A^{(2)}$ and $b^{(2)}$ come into play at step 1 and assure a representation of A and b in sextuple precision (192 bits). This is enough to allow the error decrease at roundoff level ε_1 (see (7)). The results are illustrated in Figure 1. The dashed, dash-dotted and dotted lines refer to the execution of the iterative refinement using fixed accuracy $\varepsilon_2 = \beta^{-t}/2, \beta^{-2t}/2, \beta^{-3t}/2$, respectively, and reveal the error levels it is possible to reach with these choices. In particular, we see that sextuple precision is needed at step 1 in order to obtain a double precision accurate solution. The solid line refers to the dynamic precision implementation of the procedure illustrated above. The errors at the first two iterations executed with $\varepsilon_2 = \varepsilon_1$ are labeled with asterisks. The errors in the subsequent five iterations, executed with $\varepsilon_2 = \beta^{-2t}/2$ are labeled with circles. Finally, the remaining iterations are

executed with $\varepsilon_2 = \beta^{-3t}/2$ and the related errors are labeled with crosses. We see that the error decreases until the saturation level of the corresponding precision mode is attained. Consequently, the dynamic change of the accuracy is finely tuned for this example and guarantees a number of total iterations very close to those needed by directly working with the highest considered precision but, evidently, requiring a lower computational effort.

References

1. Amodio, P., Iavernaro, F., Mazzia, F., Mukhametzhanov, M.S., Sergeyev, Y.D.: A generalized Taylor method of order three for the solution of initial value problems in standard and infinity floating-point arithmetic. Math. Comput. Simulation **141**, 24–39 (2016)
2. Bailey, D.H., Borwein, J.M.: High-precision arithmetic in mathematical physics. Mathematics **3**(2), 337–367 (2015)
3. Brugnano, L., Mazzia, F., Trigiante, D.: Fifty years of stiffness, pp. 1–21. Recent Advances in Computational and Applied Mathematics, Springer, Dordrecht (2011)
4. De Cosmis, S., Leone, R.D.: The use of grossone in mathematical programming and operations research. Appl. Math. Comput. **218**(16), 8029–8038 (2012)
5. Gaudioso, M., Giallombardo, G., Mukhametzhanov, M.S.: Numerical infinitesimals in a variable metric method for convex nonsmooth optimization. Appl. Math. Comput. **318**, 312–320 (2018)
6. Iavernaro, F., Mazzia, F., Mukhametzhanov, M.S., Sergeyev, Y.D.: Conjugate-symplecticity properties of Euler–Maclaurin methods and their implementation on the Infinity Computer. Appl. Numer. Math. (2019). https://doi.org/10.1016/j.apnum.2019.06.011
7. Iavernaro, F., Mazzia, F., Trigiante, D.: Stability and conditioning in numerical analysis. Journal of Numerical Analysis, Industrial and Applied Mathematics **1**(1), 91–112 (2006)
8. Moler, C.B.: Iterative refinement in floating point. Journal of the ACM (JACM) **14**(2), 316–321 (1967)
9. Sergeyev, Y.D.: Solving ordinary differential equations by working with infinitesimals numerically on the Infinity Computer. Appl. Math. Comput. **219**(22), 10668–10681 (2013)
10. Sergeyev, Y.D., Mukhametzhanov, M.S., Mazzia, F., Iavernaro, F., Amodio, P.: Numerical methods for solving initial value problems on the Infinity Computer. International Journal of Unconventional Computing **12**(1), 3–23 (2016)
11. Sergeyev, Y.: A new applied approach for executing computations with infinite and infinitesimal quantities. Informatica **19**(4), 567–596 (2008)
12. Sergeyev, Y.: Higher order numerical differentiation on the Infinity Computer. Optimization Letters **5**(4), 575–585 (2011)
13. Sergeyev, Y., Kvasov, D., Mukhametzhanov, M.S.: On strong homogeneity of a class of global optimization algorithms working with infinite and infinitesimal scales. Commun. Nonlinear Sci. Numer. Simul. **59**, 319–330 (2018)
14. Sergeyev, Y.D.: Numerical computations and mathematical modelling with infinite and infinitesimal numbers. J. Appl. Math. Comput. **29**(1-2), 177–195 (2009)
15. Sergeyev, Y.: Numerical infinities and infinitesimals: Methodology, applications, and repercussions on two Hilbert problems. EMS Surveys in Mathematical Sciences **4**(2), 219–320 (2017)

16. Y.D., S.: Computer system for storing infinite, infinitesimal, and finite quantities and executing arithmetical operations with them (2010), uSA patent 7,860,914
17. Žilinskas, A.: On strong homogeneity of two global optimization algorithms based on statistical models of multimodal objective functions. Applied Mathematics and Computation **218**(16), 8131–8136 (2012)