Mathematical and Computational Applications

Article

Effectiveness of Floating-Point Precision on the Numerical Approximation by Spectral Methods

José A. O. Matos [1,2,†] and **Paulo B. Vasconcelos** [1,2,*,†]

[1] Center of Mathematics, University of Porto, R. Dr. Roberto Frias, 4200-464 Porto, Portugal; jamatos@fep.up.pt
[2] Faculty of Economics, University of Porto, R. Dr. Roberto Frias, 4200-464 Porto, Portugal
[*] Correspondence: pjv@fep.up.pt
[†] These authors contributed equally to this work.

Abstract: With the fast advances in computational sciences, there is a need for more accurate computations, especially in large-scale solutions of differential problems and long-term simulations. Amid the many numerical approaches to solving differential problems, including both local and global methods, spectral methods can offer greater accuracy. The downside is that spectral methods often require high-order polynomial approximations, which brings numerical instability issues to the problem resolution. In particular, large condition numbers associated with the large operational matrices, prevent stable algorithms from working within machine precision. Software-based solutions that implement arbitrary precision arithmetic are available and should be explored to obtain higher accuracy when needed, even with the higher computing time cost associated. In this work, experimental results on the computation of approximate solutions of differential problems via spectral methods are detailed with recourse to quadruple precision arithmetic. Variable precision arithmetic was used in `Tau Toolbox`, a mathematical software package to solve integro-differential problems via the spectral Tau method.

Keywords: floating-point arithmetic; variable precision arithmetic; IEEE 754-2008 standard; quadruple precision accuracy; spectral methods

MSC: 68N01; 68N19; 65M70; 47G20

check for updates

Citation: Matos, J.A.O.; Vasconcelos, P.B. Effectiveness of Floating-Point Precision on the Numerical Approximation by Spectral Methods. *Math. Comput. Appl.* **2021**, *26*, 42. https://doi.org/10.3390/mca26020042

Academic Editor: Fernando Casas

Received: 7 April 2021
Accepted: 23 May 2021
Published: 26 May 2021

1. Introduction

Two of the main goals when implementing numerical algorithms are correctness and speed—that is, to have the results with the required precision and as fast as possible. It is, in general, possible to improve one of these features at the cost of the other. Thus, in order to use better precision, we naturally lose performance because the number/complexity of computation is increased, with the opposite happening if we require less precision. From the 2008 revision [1], the IEEE 754 standard introduced a quadruple precision floating-point format (binary128).

Currently, this 128-bit floating-point type is mostly available only in software implementations. In general, there is a cost in computing performance when using higher precision numerical types (be it quadruple or others larger than double precision), because software-based solutions are being used, e.g., studies indicated that quadruple precision can be up to four orders of magnitude slower than double precision.

Seldom are stable numerical algorithms hindered from working within machine precision since double precision arithmetic is not sufficient. This is the case, among others, when facing ill-conditioned problems. By exploring floating point arithmetic with higher precision, for instance quadruple, the required accuracy can be achieved.

The different precision types can be available at the software or hardware level. Eventually it is fair to consider that hardware-based solutions are software written at a low level and fixed/imprinted on the processors, while software solutions can change and are, thus, more flexible since they use the general architecture and, therefore, are not as fast as the hardware supported precision formats/types. Other than the usual precision formats, like single or double IEEE precision, current hardware already supports other extended formats, such as the x87 80-bit precision format (that C/C++ refers as `long double`). In the near future, hardware supporting multiprecision arithmetic, not only higher than double but also half precision, will have a huge impact on the performance whenever it can be explored.

Even for multiprecision based in software, the more general the implementation is, like arbitrary precision, the slower it is. Implementations of fixed multiprecision formats, like the quadruple precision, can take advantage of this by trading some generality for speed (e.g., [2,3]). One example of implementing a quadruple precision type (that is similar but not equal to IEEE quadruple precision) uses two double precision numbers to represent one quadruple precision, also known as double-double arithmetic.

Software-based solutions that implement arbitrary precision arithmetic are available in several environments, such as MATLAB with the symbolic math toolbox [4], Octave with the symbolic package [5], or other multiprecision packages, some of which are based on GNU GMP [6] or GNU MPFR underlying libraries [7].

More importantly is to notice that the 128-bit, or higher, floating-point arithmetic is not only needed for applications requiring higher precision but likewise to allow the computation of double precision results and to more accurately mitigate the round-off errors at intermediate calculation steps. Depending on the type of the problem being studied, the higher precision only needs to be applied at selected precision bottlenecks and, thus, only paying the speed penalty for precision where strictly required. This already happens in libraries, such as the C math library (like the GNU libc) where some of the calculations for the functions are done internally in extended double precision or higher with the results of the calculations being returned in double precision.

Another interesting use of multiprecision implementations in numerical algorithms with higher precision than double, is to benchmark the accuracy of results obtained using different internal implementations as well as the speed of each. That allows us to assess the goodness of each implementation both in terms of the speed and accuracy and, thus, to select the best candidate even if the usual/production implementation will be performed exclusively using the standard double precision.

Relevant research lines include the possibility of using higher precision to overcome inherent ill-conditioning only in parts of the code (e.g., polynomial evaluations) working as mixed-precision arithmetic to minimize the computational efforts. In comparison to our approach, accuracy is improved only when needed, not affecting the overall computational effort as much. For this approach, the Infinity Computer is an adequate computational environment where this arithmetic can be implemented [8], and in [9], some work on the solution of initial value problems by Taylor-series-based methods have already been made within this setup.

The use of such an approach for spectral methods is clearly an interesting line to investigate. Closely related to this line of research is the sinking-point [10], a floating point numerical system that tracks the precision dynamically through calculations. This works in contrast with IEEE 754 floating-point where the numerical results do not, inherently, contain any information about their precision. With this tracking mechanism, the system ensures the meaningfulness of the precision in the calculated result. The detection of unreliable computations on recursive functions based on interval extensions was addressed in [11].

In this work, experimental results on the computation of approximate solutions of differential problems via spectral methods will be exposed with recourse to multiprecision arithmetic via the variable-precision arithmetic (arbitrary-precision arithmetic) freely available in MATLAB and Octave.

2. The Tau Spectral Method

Finding accurate approximate solutions of differential problems is of crucial importance, particularly when facing large integration domains or on dynamical systems. Spectral-type methods, like the Tau method, provide excellent error properties: when the solution is smooth, exponential convergence can be expected. For a detailed explanation on the Tau method, we suggest, e.g., [12].

The Tau method attempts to express the sought solution as a linear combination of orthogonal polynomials that form the base functions. The coefficients of such a combination are the exact solution of a perturbed differential problem. In the Tau method, we obtain an nth degree polynomial approximation y_n to the differential problem's solution y by imposing that y_n solves exactly the differential problem with a polynomial perturbation term τ_n in the differential equation, or system of differential equations. To achieve good minimization properties for the error, τ_n is projected onto an orthogonal polynomial basis.

Let $\mathcal{D} = \sum_{k=0}^{\nu} p_k \frac{d}{dx^k}$ represent an order ν linear differential operator acting on the space of polynomials $\mathbb{P}$, where $p_k = \sum_{i=0}^{n_k} p_{ki} x^i$ are polynomial coefficients, $n_k \in \mathbb{N}_0$, $p_{k,i} \in \mathbb{R}$, and we let $f \in \mathbb{P}$ with finite degree λ. An approximate polynomial solution y_n for the linear differential problem

$$\begin{cases} \mathcal{D}y = f \\ c_i(y) = s_i,\ i = 1, \ldots, \nu \end{cases}, \tag{1}$$

is obtained in the Tau sense by solving the perturbed system

$$\begin{cases} \mathcal{D}y_n = f + \tau_n \\ c_i(y_n) = s_i,\ i = 1, \ldots, \nu \end{cases}. \tag{2}$$

A matrix representation of (2) can be obtained as

$$\mathsf{T}\mathsf{a} = \mathsf{b},\ \text{with } \mathsf{T} = \begin{bmatrix} \mathsf{C} \\ \mathsf{D} \end{bmatrix} \text{ and } \mathsf{b} = \begin{bmatrix} \mathsf{s} \\ \mathsf{f} \end{bmatrix} \tag{3}$$

where

$$\mathsf{C} = [c_{ij}]_{\nu \times \infty} = c_i(P_{j-1}),\ i = 1, \ldots, \nu,\ j = 1, \ldots,$$

$$\mathsf{D} = \sum_{k=0}^{\nu} p_k(\mathsf{M})\mathsf{N}^k, \quad p_k(\mathsf{M}) = \sum_{i=0}^{n_k} p_{k,i}\mathsf{M}^i. \tag{4}$$

$\mathsf{s} = [s_1, \ldots, s_\nu]^T$ and $\mathsf{f} = [f_1, \ldots, f_{n-\nu}, 0, 0, \ldots]^T$ represent, respectively, the boundary conditions and the coefficients of the differential equation on the basis $\mathcal{P}$. Matrices M and N stand, respectively, for the multiplication and differentiation operators.

This is known as an operational formulation of the Tau method and represents a convenient framework for the implementation of the method. All operations are translated into matrix formulations, like the multiplication (M) of polynomials and derivatives (N). The solution of the differential problem is obtained by solving a linear system of equations, where the infinite system (3) is truncated to order equal to the wanted polynomial degree approximation. If the problem is nonlinear, a linearization process is built.

The `Tau Toolbox` [13–15] provides a robust and stable numerical library for the solution of integro-differential problems using the Tau method. In particular, the operational matrices M and N are computed directly on the orthogonal basis, thus, avoiding the usual similarity transformation. Indeed, building those matrices on the orthogonal basis is demanding and tricky in contrast with the power basis, which is intuitive and trivial. The drawback is that the latter requires a change of basis (twice) introducing stability issues. The `Tau Toolbox` provides these matrices via explicit and/or recursive relations.

The operations involving changes of the polynomial basis and powers of matrices must be numerically tackled with expertise, otherwise the overall approach may not be stable. Let

$\mathcal{P} = [P_0(x), P_1(x), \ldots]$ be an orthogonal basis satisfying $xP_j = \alpha_j P_{j+1} + \beta_j P_j + \gamma_j P_{j-1}$, $j \geq 0$, $P_0 = 1$, $P_{-1} = 0$.

- A proper `polyval` function is deployed for orthogonal evaluation. If $\mathcal{P}^*$ are the corresponding orthogonal polynomials shifted to $[a, b]$ and x is a vector, then the evaluation of $y_n(x) = \sum_{i=0}^{n} a_i P_i(x)$ is directly computed in P_n:

$$\begin{cases} P_0^* = [1, \ldots, 1]^T \\ P_1^* = \frac{c_1 x + (c_2 - \beta_0) P_0^*}{\alpha_0} \\ P_i^* = \frac{(c_1 x + c_2 - \beta_{i-1} P_0^*) \odot P_{i-1}^* - \gamma_{i-1} P_{i-2}^*}{\alpha_{i-1}}, \quad i = 2, 3, \ldots, n \end{cases}$$

  where $\odot$ is the element-wise product of two vectors, $c_1 = \frac{2}{b-a}$ and $c_2 = \frac{a+b}{a-b}$.
- No change of basis is used via matrix inversion. If V satisfies $\mathsf{a}_{\hat{\mathcal{P}}} = \mathsf{V}\mathsf{a}_{\mathcal{P}}$, where $\mathcal{P}$ and $\hat{\mathcal{P}}$ are the polynomial basis, then the coefficients of $\mathsf{W} = \mathsf{V}^{-1}$ are computed without inverting V by the recurrence relation

$$\begin{cases} \mathsf{w}_1 = \mathsf{e}_1 \\ \mathsf{w}_{j+1} = \mathsf{M}\mathsf{w}_j, \quad j = 1, 2, \ldots \end{cases},$$

  where M is such that $\mathcal{P}x = \mathcal{P}\mathsf{M}$, w_j is the jth column of M and e_1 the first column of the identity matrix.
- All similarity transformations are avoided to ensure numerical stable computations. Recurrence relations to compute the elements of the multiplication and differentiation operators (matrices M and N) are computed directly on the orthogonal basis:

$$\mathsf{M} = \left[\mu_{i,j}\right]_{i,j=1}^{n} = \begin{cases} \mu_{j+1,j} = \alpha_{j-1}, \quad \mu_{j,j} = \beta_{j-1}, \quad \mu_{j,j+1} = \gamma_{j-1} \\ \mu_{i,j} = 0, \quad |i-j| > 1 \end{cases},$$

$$\mathsf{N} = \left[\eta_{i,j}\right]_{i,j=1}^{n} = \begin{cases} \eta_{i,j+1} = \frac{\alpha_{i-1}\eta_{j,i-1} + (\beta_i - \beta_j)\eta_{j,i} + \gamma_{i+1}\eta_{j,i+1} - \gamma_j \eta_{j-1,i}}{\alpha_j} \\ \eta_{j,j+1} = \frac{\alpha_{j-1}\eta_{j,j-1} + 1}{\alpha_j} \end{cases}$$

These algorithms, among many others, are implemented in the `Tau Toolbox` library to ensure stability. The operational approach of the method, however, gives rise to operator matrices that can have increased condition numbers with the degree of the approximation. Solving ill-conditioned problems, even in the presence of a stable method, may lead to approximate solutions far from the required accuracy. It is at this point that variable precision can overcome the constraints imposed, inherently, by the data.

It is worth mentioning here that `Tau Toolbox` offers a post-processing phase based on the Frobenius–Padé approximation method to build rational approximations from the polynomial Tau approximation. This filtering extension improves the accuracy of the spectral approximation when working on the vicinity of solutions with singularities [16].

3. Numerical Experiments

In this section, we report the numerical results using variable precision arithmetic (VPA) in `Tau Toolbox`, mainly quadruple precision, emphasizing the complementary role that both quadruple and double precision can play in finding accurate approximate solutions.

In the first example, we illustrate the use of the `Tau Toolbox` to solve a boundary value problem, using double and quadruple precisions. The second example explores the properties of classical orthogonal polynomials to highlight the possibility of copying with more than the most usual Chebyshev basis and the use of high-level `Tau Toolbox` functions to overcome certain implementation technicalities. The third example shows that, for a set of initial value problems, the floating-point arithmetic together with the ill-conditioning of the data can lead to unsatisfactory accuracy results. The use of extended precision allows us to obtain machine double precision, which is a relevant aspect to emphasize, allowing the circumvention of accuracy bottlenecks.

All the errors illustrated in the examples are true errors, since we are comparing the results with a known analytical solution. For regular computations, the error is controlled via the Cauchy relative error ($\|y_n - y_{n-\ell}\| / \|y_n\|$, for a given ℓ).

The machine used for the computations was an AMD Ryzen 7 4800H with 32.0 GB RAM memory.

3.1. Example 1

In this first example, we consider the solution of a boundary value problem

$$(k^4 - 4k^3x + 4k^2x^2 + 2k^2 - 4kx + 1)\frac{d^2y}{dx^2}(x) - k(k^2 - 2kx + 1)\frac{dy}{dx}(x) - 2k^2y(x) = 0,$$

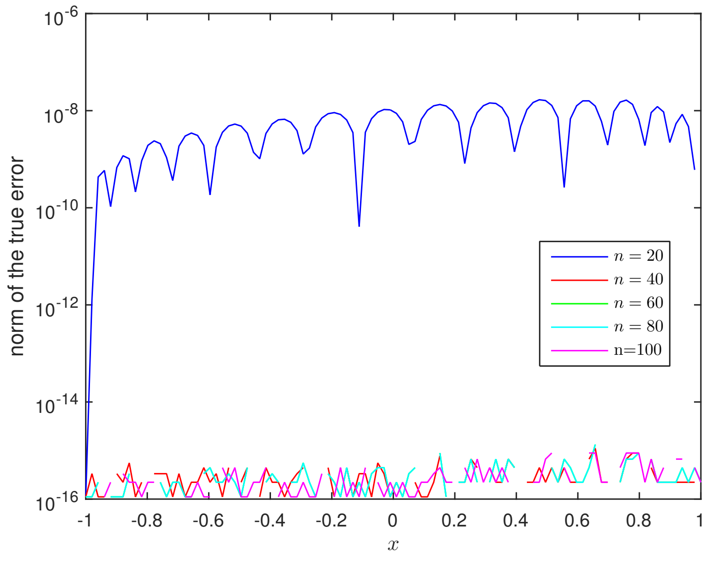
for $y(-1) = \frac{1}{1+k}$, $y(1) = \frac{1}{1-k}$, with the algebraic solution $y(x) = (1 - 2kx + k^2)^{-\frac{1}{2}}$.

The code below shows how to use the Tau method to solve the problem using either double or quadruple precisions and considering a Chebyshev (of first type) basis. It closely follows the theoretical framework presented in Section 2, using `Tau Toolbox` functions to build the necessary intermediate matrices, e.g., C and D, which internally process the M and N matrices described in (4).

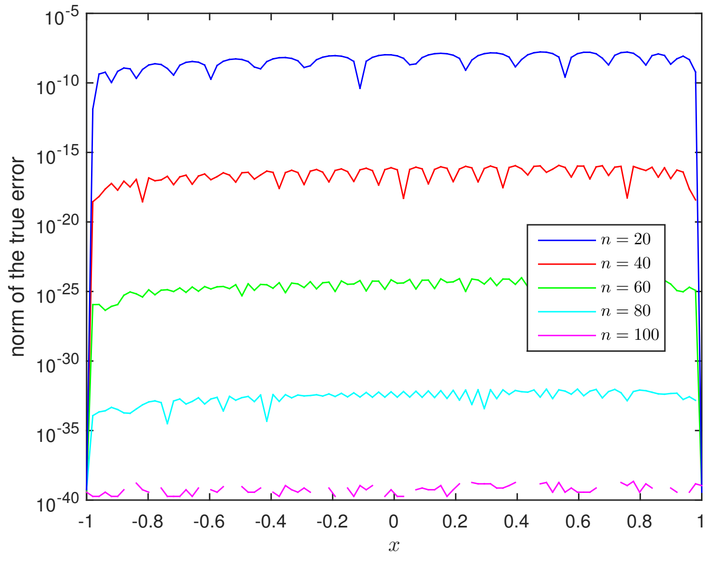
```
% differential problem
k = 0.4;
equation = @(x,y) (k^4-4*k^3*x+4*k^2*x^2+2*k^2-4*k*x+1)*...
                  diff(y, 2)-k*(-2*k*x+k^2+1)*diff(y)-2*k^2*y;
domain = [-1, 1];
conditions = @(y) {y(-1)-1/(1+k); y(1)-1/(1-k)};
options = tau.settings('degree', 20, 'quadPrecision', true);
problem = tau.problem(equation, domain, conditions, options);

% get the conditions and operator matrices
C = tau.matrix('condition', problem);
D = tau.matrix('operator', problem);

% build Tau matrix and solve the problem
nu = size(C, 1); % number of boundary conditions
T = [C(:, 1:n); D(1:n-nu, 1:n)];
b = zeros(n,1); b(1) = 1/(1+k); b(2) = 1/(1-k);
a = T\b;
```

The user can use quadruple precision just by setting the `quadPrecision` flag to be `true` (as shown). By default, the precision is double, and the `quadPrecision` is `false`.

Results for the error with respect to the known exact solution are shown in Figure 1. In Figure 1b, for double precision arithmetic, machine precision is almost reached for polynomial degrees of $n = 40$ or higher. The accuracy is kept near the maximum possible accuracy for higher values of n. With quadruple precision (Figure 1b) for $n = 40$, the accuracy is already under 10^{-16}, and, for increasing values of the degree n, the accuracy increases.



(**a**)



(**b**)

Figure 1. The true error for values of n ranging from 20 to 100: (**a**) Double precision. (**b**) Quadruple precision. The increase in the polynomial degree of the solution of the differential problem leads to an increase in the accuracy of the solution, which reflects the stability of the Tau method in the `Tau Toolbox`; for double precision, a moderate value for the polynomial approximation is sufficient to reach machine precision and after that, a stagnation occurs, while, for quadruple precision, more accurate results can be reached for higher polynomial degrees.

Figure 2 draws the norm of the true error for several values of n using double and quadruple precisions. Both provide similar results for polynomial approximations with degrees smaller than $n = 35$, as expected. On one hand, the double precision cannot improve the accuracy of the solution for values larger than 40. On the other hand, quadruple precision continues to ensure better results until reaching the degree $n = 95$. In both cases, it is important to notice the robustness of the implemented method since there is no degradation of the accuracy for higher values of the polynomial degree.

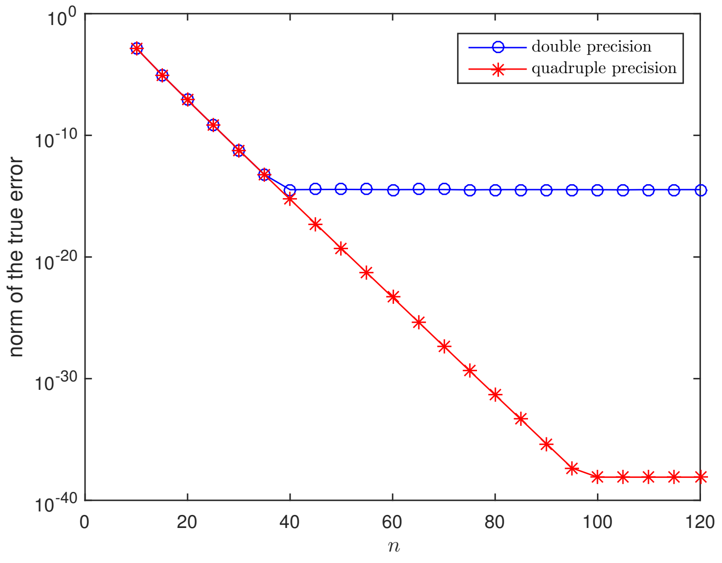


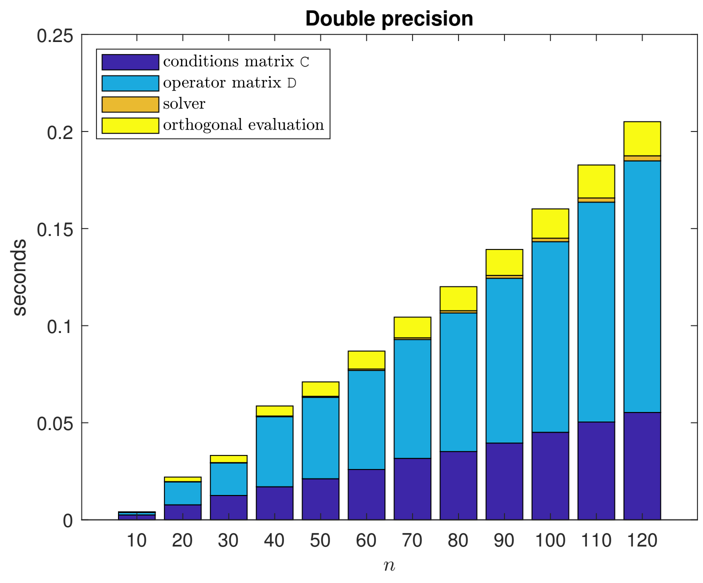
Figure 2. The true error for values of n ranging from 10 to 120. Notice the robustness of the implemented Tau method that faces no degradation in accuracy for increasing values of the polynomial degree with both double and quadruple precision.

The problem is not ill-behaved in terms of the data since the condition numbers of the Tau coefficient matrices are not very high (see Table 1). Later, we will deal with ill-conditioned problems.

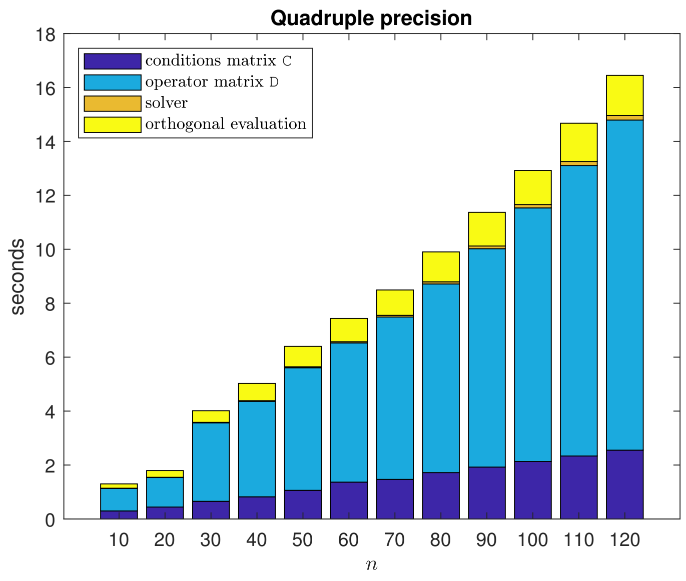
Table 1. Condition numbers for the coefficient Tau matrix T_n.

n	20	40	60	80	100
cond(T$_n$)	5.1×10^4	8.8×10^5	4.5×10^6	1.4×10^7	3.5×10^7

Considering the largest value for n, the times required for parsing and building the matrix problem were 6 ms and 15.900 s, while those for the solution phase were 6 ms and 1.788 s, respectively for double and quadruple precision. An order of magnitude of four was found for the parsing and building process and of three for the solution phase. The solution phase represents a minor cost and includes the evaluation of the polynomial coefficients on the orthogonal basis, which is, in turn, more costly than the solver itself. The most demanding stages are the generation of the building blocks for the matrix formulation, where finding the operator matrix is, as expected, marginally more costly than the conditions matrix. This is clearly illustrated in Figure 3.



(**a**)



(**b**)

Figure 3. Time (in seconds) for double and quadruple precision for values of n ranging from 10 to 120: (**a**) Double precision. (**b**) Quadruple precision. The pattern of the most time consuming parts of the code are similar, but building of the operator matrix D is, in relative terms, more demanding for quadruple precision.

Figure 4 depicts the cost factor for quadruple vs. double precision. Broadly speaking, this factor can be considered as constant independently of the polynomial degree approximation. The complexity of the algorithm, in all parts examined, is around the same for both precision types, while slightly changing the multiplicative constant.

This time, analysis is reproduced with other ordinary differential problems, of initial or boundary conditions, with similar conclusions. Even if the time required for the quadruple approach is much higher than for double precision, it is moderate and acceptable (bearing in mind that the double precision computations are very fast). This functionality is to be used on the limited number of cases when the double precision does not enable good approximations. For many cases, the double precision is sufficient to ensure almost machine epsilon double precision (say $10 \times eps$) in `Tau Toolbox`.

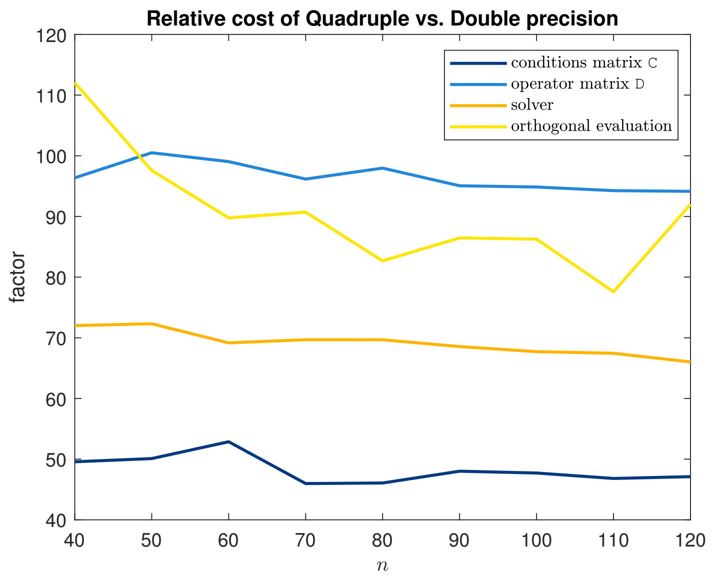


Figure 4. The relative cost of quadruple vs. double precision for values of n ranging from 40 to 120.

3.2. Example 2

Now, we consider a set of boundary value problems and show how to use high level `Tau Toolbox` functions to help formulate and solve the problems with ease.

The family of orthogonal polynomials $y_n(x)$ of degree n, in an interval $[a,b]$, satisfies the relation

$$g_2(x)\frac{d^2y_n}{dx^2}(x) + g_1(x)\frac{dy_n}{dx}(x) + a_n y_n(x) = 0 \tag{5}$$

where g_1 and g_2 are independent of n and the constant a_n only depends on n (see [17], Sections 22.1.3 and 22.6), and $y_k = P_k$, with $k = n-1$. For Chebyshev polynomials of the second type we have $g_1(x) = (1-x^2)$, $g_2(x) = -3x$, and $a_n = n(n+2)$, and for Legendre $g_1(x) = (1-x^2)$, $g_2(x) = -2x$, and $a_n = n(n+1)$. Problem (5) is fully specified in the interval $[-1,1]$ with $y(-1) = y(1) = 1$ as boundary conditions.

The norm of the characteristic Equation (5) using Chebyshev of the second type and Legendre basis, for $n = 40$ is, respectively, 0 and 4.7×10^{-13}. Thus, whereas for Chebyshev, the machine precision is reached, for Legendre, that is not the case. For quadruple precision, the error of the Chebyshev basis is still within the maximum accuracy and, for Legendre, it is 1.6×10^{-36}, which allows us to offer an approximate solution with accuracy below machine precision (double).

The code, using the high level `Tau Toolbox` function `tau.solve`, is:

```
% parameter
n = 10;

% differential problem
equation = @(x,y) (1-x^2)*diff(y,2)-2*x*diff(y)+n*(n+1)*y;
domain = [-1, 1];
conditions = @(y) {y(-1)-1; y(1)-1};
options = tau.settings('degree', n, 'basis', 'LegendreP');
problem = tau.problem(equation, domain, conditions, options);

% solution via tau method
yn = tau.solve(problem);
```

The user provides, in ordinary language, the parameters, the problem to be solved together with the conditions, and the degree of the wanted approximation. Then, the sought solution is found via `tau.solve`, which builds the required objects, sets the algebraic Tau formulation, and solves the problem in the Tau sense.

The solution is given by $y_n = \sum_{i=0}^{n} a_i P_{n,i}$, where P_n is an orthogonal (in the code shown Legendre) basis.

3.3. Example 3

This example shows that, for a set of initial value problems, the usual arithmetic together with the ill-conditioning of the data can lead to unsatisfactory results in terms of accuracy.

Let us consider the ordinary differential problem

$$(k-m)!x^m \frac{d^m y}{dx^m}(x) - k!y(x) = 0, \quad m \in \mathbb{N}, \quad m < k \text{ (fixed)}$$

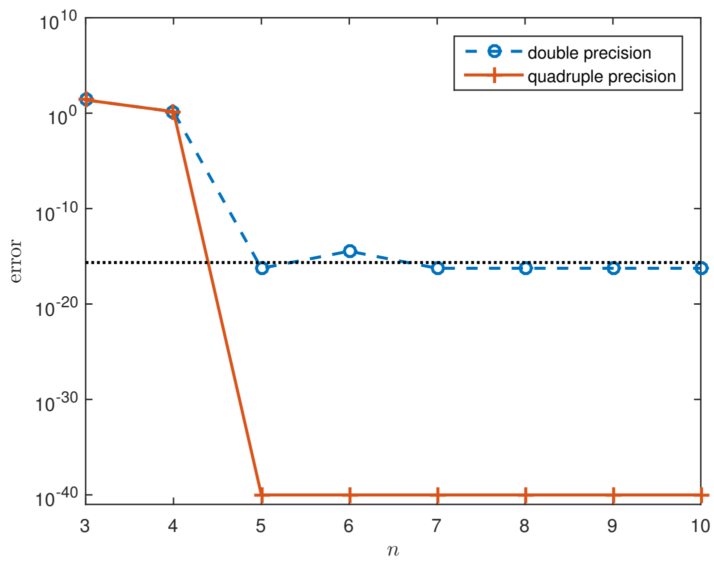
with the initial conditions

$$\begin{aligned} y(-1) &= (-1)^k \\ y'(-1) &= k\,(-1)^{k-1} \\ \cdots & \\ y^{(r)}(-1) &= k!/(k-r)!\,(-1)^{k-r}, \quad r = 0, \ldots, m-1. \end{aligned}$$

The analytical solution is x^k.

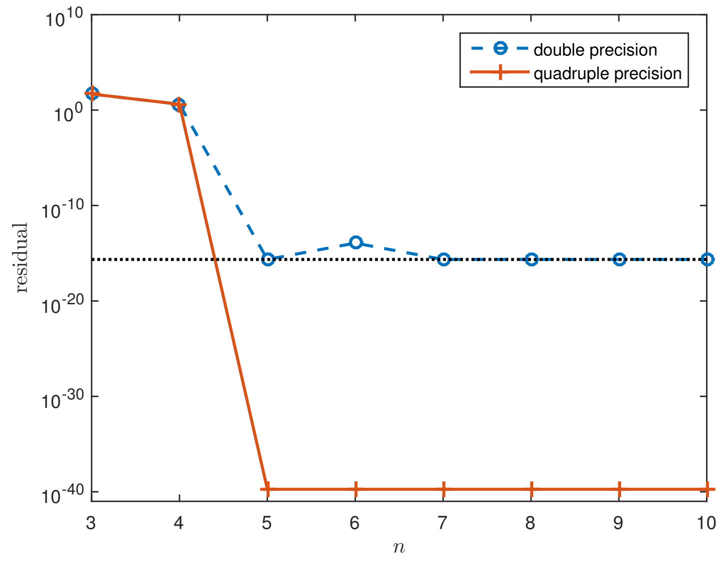
Since the solution is polynomial, the spectral method is expected to deliver the exact solution for the same polynomial degree approximation. However, this might not be the case due to the poor condition number of the linear system to be solved.

For this experiment, we tested the numerical approximation for $m = 4$ and $k = 5$. Since the derivative order along with the power exponent are small, a machine precision accuracy was expected for polynomial degree approximations equal to 5 and beyond. Figure 5 shows the true error (Figure 5a) and the residual (Figure 5b) for this specification and for several values of n, for double and quadruple precisions. Indeed, from $n = 5$ on, the solution is found within machine precision. The code is stable even when n grows.



(**a**)

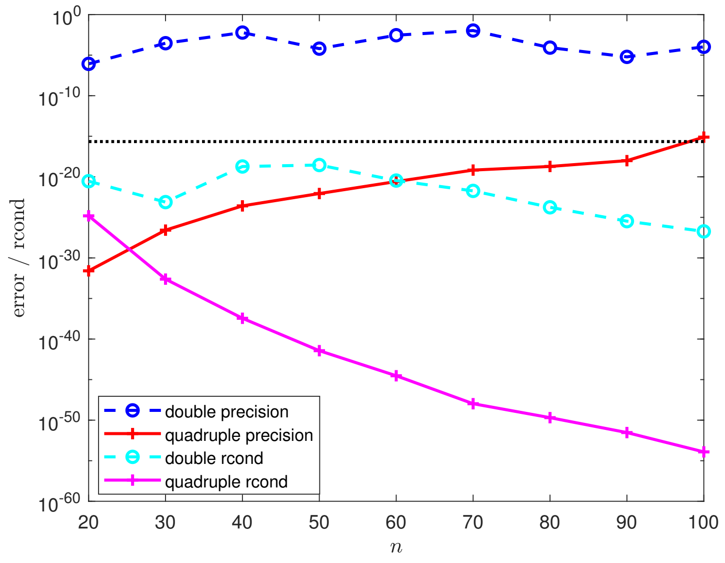
Figure 5. *Cont.*



(**b**)

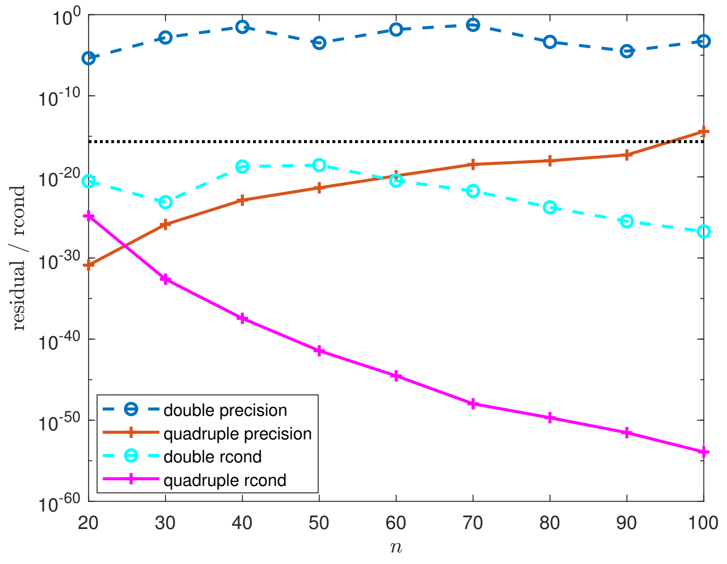
Figure 5. Convergence for values of n ranging from 3 to 10: $m = 4$, $k = 5$: (**a**) The error. (**b**) The residual. For well conditioned data, the stability of the Tau method and the robustness of its implementation allows for maximum accuracy according to the working precision.

For larger values of m and/or k, problems can occur, mainly due to ill-conditioning. Figure 6 shows the results of similar experiments but with considering $m = 11$ and $k = 13$. The reciprocal condition estimator of the condition number is also drawn for each n tested. The condition number of the problems to be solved is high, and thus the approximate solutions may not be computed accurately. It is clear that, for increasing values of n, the condition number increases (the reciprocal decreases).



(**a**)

Figure 6. *Cont.*



(**b**)

Figure 6. Convergence for values of n ranging from 20 to 100: $m = 11$, $k = 13$. The reciprocal condition estimators: (**a**) The error. (**b**) The residual. The double precision floating-point arithmetic operating on the ill-conditioning of data disables the ability to achieve accurate results; the use of enlarged precision allows us to obtain results with machine double precision.

For double precision arithmetic, the quality of the approximate solution is poor since the error is, for all cases considered, high: the approximation cannot be delivered with more than two or three significant digits, thus, strongly under single precision accuracy.

On the other hand, with quadruple precision arithmetic, the approximation was obtained with machine double precision (10^{-16}). Even for the larger n, where the condition number is higher than 10^{50}, an approximate solution can be obtained within machine double precision.

4. Conclusions

In this work, we extended the `Tau Toolbox` to work with variable precision arithmetic. This possibility is crucial to (i) accommodate ill-conditioned problems, which prevent stable algorithms from working within machine precision and (ii) distinguish between two different computational implementations of the same mathematical expression in terms of both the accuracy and speed. We compared both approaches for double and quadruple precision for several examples, including ill-conditioned problems. In those problems, the use of quadruple precision, used internally in the evaluations, allowed the method to achieve double precision, whereas lower than single precision was attained when the internal calculations were performed using double precision.

Spectral methods can deliver accurate approximation solutions, and thus the possibility to work with greater precision is a remarkable aspect. The `Tau Toolbox` allows the exploration of variable precision arithmetic with a single parameter specification. This is possible because the software package was built internally supporting different precision types and using naturally default double precision arithmetic.

The experimental results shown illustrate the efficiency of the use of quadruple precision on the computation of approximate solutions of differential problems via the spectral Tau method, in terms of the accuracy of the solution. Clearly, there is a time penalty that must be paid. In the near future, we expect that more widely used machine architectures will provide, natively, quadruple precision, which will mitigate the cost and, therefore, make its use more appealing. When that occurs, `Tau Toolbox` will be able to take immediate advantage since it is already prepared for this possibility.

Author Contributions: Conceptualization, J.A.O.M. and P.B.V.; methodology, J.A.O.M. and P.B.V.; software, J.A.O.M. and P.B.V.; writing—original draft preparation, J.A.O.M. and P.B.V.; writing—review and editing, J.A.O.M. and P.B.V. All authors have read and agreed to the published version of the manuscript.

Funding: The authors were partially supported by CMUP, which is financed by national funds through FCT—Fundação para a Ciência e a Tecnologia, I.P., under the project with reference UIDB/00144/2020.

Acknowledgments: We thank the reviewers for their careful reading of the manuscript and for providing very constructive comments that significantly improved the presentation quality.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. 754-2008 IEEE standard for floating-point arithmetic. *IEEE Comput. Soc. Std.* **2008**, *2008*, 517. [CrossRef]
2. Higham, N. A Multiprecision World. *SIAM News* **2017**. Available online: https://sinews.siam.org/Details-Page/a-multiprecision-world (accessed on 23 May 2021).
3. Higham, N.J. The rise of multiprecision arithmetic. In Proceedings of the 2017 IEEE 24th Symposium on Computer Arithmetic (ARITH), London, UK, 24–26 July 2017.
4. The MathWorks, Inc., Natick, MA, USA. Available online: https://www.mathworks.com/ (accessed on 23 May 2021).
5. Meurer, A.; Smith, C.P.; Paprocki, M.; Čertík, O.; Kirpichev, S.B.; Rocklin, M.; Kumar, A.; Ivanov, S.; Moore, J.K.; Singh, S.; et al. SymPy: Symbolic computing in Python. *PeerJ Comput. Sci.* **2017**, *3*, e103. [CrossRef]
6. Granlund, T. GNU MP: The GNU Multiple Precision Arithmetic Library. Available online: http://gmplib.org/ (accessed on 23 May 2021).
7. Zimmermann, P. Reliable computing with GNU MPFR. In Proceedings of the International Congress on Mathematical Software, Kobe, Japan, 13–17 September 2010; Springer: Berlin, Germany, 2010; pp. 42–45.
8. Amodio, P.; Brugnano, L.; Iavernaro, F.; Mazzia, F. On the use of the Infinity Computer architecture to set up a dynamic precision floating-point arithmetic. *Soft Comput.* **2020**, *24*, 17589–17600. [CrossRef]
9. Amodio, P.; Iavernaro, F.; Mazzia, F.; Mukhametzhanov, M.; Sergeyev, Y. A generalized Taylor method of order three for the solution of initial value problems in standard and infinity floating-point arithmetic. *Math. Comput. Simul.* **2017**, *141*, 24–39. [CrossRef]
10. Zorn, B.; Grossman, D.; Tatlock, Z. Sinking Point: Dynamic Precision Tracking for Floating-Point. In Proceedings of the Conference for Next Generation Arithmetic 2019 (CoNGA'19), Singapore, 13–14 March 2019; Association for Computing Machinery: New York, NY, USA, 2019. [CrossRef]
11. Nepomuceno, E.G.; Martins, S.A.; Silva, B.C.; Amaral, G.F.; Perc, M. Detecting unreliable computer simulations of recursive functions with interval extensions. *Appl. Math. Comput.* **2018**, *329*, 408–419. [CrossRef]
12. Ortiz, E.L. The Tau method. *SIAM J. Numer. Anal.* **1969**, *6*, 480–492. [CrossRef]
13. Trindade, M.; Matos, J.M.; Vasconcelos, P.B. Towards a Lanczos' τ-Method Toolkit for Differential Problems. *Math. Comput. Sci.* **2016**, *10*, 313–329. [CrossRef]
14. Vasconcelos, P.B.; Matos, J.M.; Matos, J.A. Tau Toolbox: Spectral Package for the Solution of Integro-Differential Problems. Available online: https://cmup.fc.up.pt/tautoolbox (accessed on 23 May 2021).
15. Vasconcelos, P.B.; Matos, J.M.; Trindade, M.S. Spectral Lanczos' Tau method for systems of nonlinear integro-differential equations. In *Integral Methods in Science and Engineering, Volume 1*; Springer: Berlin, Germany, 2017; pp. 305–314.
16. Matos, J.C.; Matos, J.A.; Rodrigues, M.J.; Vasconcelos, P.B. Approximating the solution of integro-differential problems via the spectral Tau method with filtering. *Appl. Numer. Math.* **2020**, *149*, 164–175. [CrossRef]
17. Abramowitz, M.; Stegun, I.A. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*; US Govt. Print: Washington, DC, USA, 2006.