

cdr

The numerical solution of fractional differential equations: Speed versus accuracy

| | |
|---------------|---|
| Item Type | Report |
| Authors | Ford, Neville J.;Simpson, A. Charles |
| Citation | Manchester: Manchester Centre for Computational Mathematics, 2003. |
| Publisher | Manchester Centre for Computational Mathematics |
| Download date | 2026-05-19 03:49:26 |
| Link to Item | http://hdl.handle.net/10034/13245 |



University of Chester

**This work has been submitted to ChesterRep – the University of Chester’s
online research repository**

<http://chesterrep.openrepository.com>

Author(s): Neville J Ford; A Charles Simpson

Title: The numerical solution of fractional differential equations: Speed versus
accuracy

Date: 2003

Originally published in:

Example citation: Ford, N. J., & Simpson, A. C. (2003). *The numerical solution of
fractional differential equations: Speed versus accuracy*. Numerical Analysis
Reports: 385. Manchester: Manchester Centre for Computational Mathematics

Version of item: Author’s post-print

Available at: <http://hdl.handle.net/10034/13245>



**The numerical solution of fractional differential
equations: speed versus accuracy**

Neville J. Ford and A. Charles Simpson

Numerical Analysis Report No. 385

A report in association with Chester College

Manchester Centre for Computational Mathematics
Numerical Analysis Reports

DEPARTMENTS OF MATHEMATICS

Reports available from: And over the World-Wide Web from URLs
Department of Mathematics <http://www.ma.man.ac.uk/MCCM/MCCM.html>
University of Manchester <ftp://ftp.ma.man.ac.uk/pub/narep>
Manchester M13 9PL
England

The numerical solution of fractional differential equations: speed versus accuracy

Neville J. Ford and A. Charles Simpson

May 23, 2003

Abstract

This paper is concerned with the development of efficient algorithms for the approximate solution of fractional differential equations of the form

$$D^\alpha y(t) = f(t, y(t)), \quad \alpha \in \mathbb{R}^+ - \mathbb{N} \quad (\dagger).$$

We briefly review standard numerical techniques for the solution of (\dagger) and we consider how the computational cost may be reduced by taking into account the structure of the calculations to be undertaken. We analyse the *fixed memory principle* and present an alternative *nested mesh* variant that gives a good approximation to the true solution at reasonable computational cost. We conclude with some numerical examples.

1 Introduction

The notion of a derivative which interpolates between the familiar integer order derivatives was introduced many years ago and has gained increasing importance in recent years in the development of mathematical models of a wide variety of situations in engineering, materials science, control theory and polymer modelling (see, for example, the book by Oldham and Spanier [17], and the more recent works by Diethelm and Freed [6], Podlubny [18], which incorporate model equations containing fractional derivatives).

There are several (non-equivalent) definitions of the fractional derivative in widespread use and we choose to focus on one particular form (the so-called Caputo version) in this paper. The version of the fractional derivative traditionally defined and analysed by mathematicians is the Riemann-Liouville fractional derivative, (see, for example, Samko et al [20]) but this is not always the most convenient definition for real applications. When used in mathematical models the Riemann-Liouville fractional derivative requires initial conditions to be expressed in terms of fractional integrals and their derivatives which have no obvious physical interpretation and therefore it is unlikely that the initial values required will be immediately available. The alternative definition of the fractional derivative given by Caputo [3] has the advantage of only requiring initial conditions given in terms of integer-order derivatives. These integer-order derivatives represent well-understood features of a physical situation and therefore their values can be measured accurately. In this paper we use the Caputo fractional derivative exclusively, precisely because of its applicability to real-world models.

Let $t > 0$, $n \in \mathbb{N}$, $\alpha \in (n - 1, n)$ then we define the Riemann-Liouville fractional integral of a function f as

$$I^\alpha f(t) = \frac{1}{\Gamma(\alpha)} \int_0^t \frac{f(s)}{(t-s)^{1-\alpha}} ds, \quad (1.1)$$

and the Caputo fractional derivative of f as

$$\begin{aligned} D^\alpha f(t) &= I^{n-\alpha}(f^{(n)})(t) \\ &= \frac{1}{\Gamma(n-\alpha)} \int_0^t \frac{f^{(n)}(s)}{(t-s)^{\alpha+1-n}} ds. \end{aligned} \quad (1.2)$$

A fractional differential equation is a differential equation where at least one of the differential operators is of fractional order. An example is given in the equation.

$$D^\alpha y(t) = f(t, y(t)), \quad t > 0, \quad \alpha \in \mathbb{R}^+ - \mathbb{N}, \quad (1.3)$$

where we assume initial conditions are given in the form

$$f^{(i)}(0) = f_0^{(i)}, \quad i = 0, \dots, n-1, \quad n \in \mathbb{N}, \quad \text{where } \alpha \in (n-1, n). \quad (1.4)$$

The Lipschitz condition

$$|f(t, y) - f(t, x)| \leq L|y - x|, \quad L > 0 \quad (1.5)$$

on the function f (together with continuity with respect to its first argument) is then sufficient to ensure existence of a unique solution to (1.3).

The basic analytical results on existence and uniqueness of solutions to fractional differential equations are given in Samko et al [20] and Podlubny [18]. For equations defined in terms of the Caputo fractional differential operator, further discussion of these matters is contained in the recent papers by Diethelm and Ford [7, 8, 9].

The increasing use of fractional differential equations in mathematical models motivates the desire to develop good quality numerical methods for their solution. Convergent and accurate methods for evaluating fractional integrals and for solving fractional differential equations have been developed and implemented by, amongst others, Lubich [13], Baker and Derakhshan [1], Blank [2], and Diethelm [4]. However, one problem with established methods is that they may be very slow, particularly where the application area requires a solution to be given over a long time interval.

It is clear from the definition of the fractional derivative that we require the approximation of a convolution integral and this is a computationally expensive problem since it requires us to sample and multiply the behaviour of two functions over the whole of the interval of integration. This leads to operation counts of $\mathcal{O}(n)$ at each step and $\mathcal{O}(n^2)$ overall, where n is the number of sampling points. Our aim is to exploit some of the structure within the integrand to produce a more-efficient approximation without significant loss of accuracy.

We can illustrate the problem with reference to the analysis presented by Diethelm and Ford in [7]. The authors show that the solutions of two equations with neighbouring orders will (under suitable conditions on their right hand sides f) lie close to one another. In particular the solution

$$D^1 y(t) = ky(t), \quad y(0) = y_0, \quad (1.6)$$

will be close to the solution of

$$D^{1-\epsilon} y(t) = ky(t), \quad y(0) = y_0, \quad \epsilon > 0, \quad (1.7)$$

for ϵ sufficiently small, however the amount of work required to compute their solutions may be quite different. If we consider the solution of system (1.6) over a fixed time interval $[0, L]$ with a step size of 10^{-3} , then using the trapezium method the solution takes $\mathcal{O}(10^3)$ flops whereas the computational cost of solving system (1.7), under the same condition using (for example) the method (of similar order) described in Diethelm [4], is $\mathcal{O}(10^6)$ flops. We find a similar increase in computational cost for any choice of fractional linear multistep method to solve (1.7).

The computational cost limits the usefulness of the fractional calculus as a modelling tool. The more sophisticated higher-order methods suffer from even greater increases in computational effort and this motivates us to consider how the computational effort can be reduced, without serious deterioration in the error of the method.

In this paper we explore existing techniques for reducing the amount of computational effort required to solve a fractional differential equation and assess the implications for the error in the approximate solutions. We present a new approach to reducing computational cost that keeps the error under control.

2 Methods for Accelerating the Computation

We have chosen to use the algorithm proposed by Diethelm [4] as a prototype and modify it to accelerate the computation. We could have used any fractional multistep method (see Lubich [13]) and we would have obtained the same pattern. One particular advantage of confining ourselves to a low-order method is that we do not have to concern ourselves with the computation of starting weights which would divert us from the task in hand: We are interested in the improvement in speed and the behaviour of the error compared to the solution calculated by the original algorithm.

Remark 1ex The recent thesis [16] contains a discussion of how properties of the integrand may be exploited in the adaptation of fractional linear multistep methods in order to gain a good order of convergence. The approach described in that thesis is quite different from the ideas we pursue here since we are especially concerned with reducing the computational cost of applying the algorithm.

The methods reviewed here for accelerating the computation rely on altering the quality of the approximation of the kernel of the fractional integral. In the standard method we see that the computational effort increases because we sample at equally spaced points. Our approach is to use equally spaced sample points for some interval of recent history $I_r = [t - T_r, t]$ and then to use some other approximation regime to fill in the remainder of the interval $R_r = [0, t - T_r)$. This seems reasonable since the kernel of the fractional derivative has a fading memory property ([4]) as time elapses. In [5] the possibility of varying the step length is considered, but our approach is quite different.

3 The Fixed Memory Principle

The simplest approach is to disregard the *tail* of the integral and to integrate only over a fixed period of recent history, I_r . In other words, we choose to approximate the integral over R_r as zero. This is commonly referred to as the *Fixed Memory Principle*, and is described in Podlubny [18], [19] where the error introduced for the Riemann-Liouville fractional derivative was analysed. Clearly the computational cost, at each step, is reduced to $\mathcal{O}(1)$ which is attractive. Podlubny [18] showed that the use of a fixed memory of length $T > 0$ introduces an error E (independent of the full interval of integration) that satisfies

$$E < \frac{MT^{-\alpha}}{\Gamma(1 - \alpha)}. \quad (3.1)$$

One can then show that the error introduced through the fixed memory principle can be controlled to preserve the order $\mathcal{O}(h^p)$ of some numerical method by choosing T suitably (of order $\mathcal{O}(h^{-\frac{p}{\alpha}})$).

We consider the error introduced for the Caputo fractional derivative, and the conclusions turn out to be a little different. For a fixed memory of length T and for $\alpha \in (0, 1)$, we make the analytic substitution of

$$\frac{1}{\Gamma(1 - \alpha)} \int_{t-T}^t \frac{y'(s)}{(t-s)^\alpha} ds \quad (3.2)$$

for

$$\frac{1}{\Gamma(1 - \alpha)} \int_0^t \frac{y'(s)}{(t-s)^\alpha} ds \quad (3.3)$$

and then attempt to make an accurate approximation of the integral in equation (3.2). This truncation introduces an error of

$$E = \left| \frac{1}{\Gamma(1 - \alpha)} \int_0^{t-T} \frac{y'(x)}{(t-x)^\alpha} dx \right|. \quad (3.4)$$

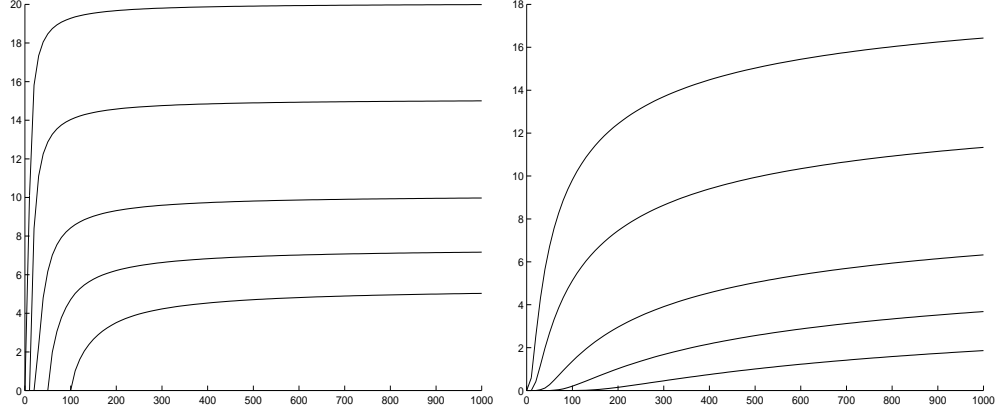


Figure 1: Percentage error occurring when using a memory of fixed length $T = 5, 10, 25, 50, 100$, in systems (3.6) and (3.7) respectively.

Let $\sup_{s \in [0, t]} |y'(s)| = M$ then

$$\begin{aligned} E &\leq \left| \frac{M}{\Gamma(1-\alpha)} \int_0^{t-T} \frac{1}{(t-x)^\alpha} dx \right| \\ &= \frac{M}{\Gamma(2-\alpha)} (t^{1-\alpha} - T^{1-\alpha}). \end{aligned}$$

As one would expect, the error $E = 0$ when $T = t$. However, for fixed T , one can see that this approach results in a loss of order in our numerical method. However the loss of order seems to be avoidable, if we allow the choice of T to vary with h (and the interval length). Thus for any given error bound $\epsilon > 0$ it is sufficient that we choose T to satisfy

$$T^{1-\alpha} \geq t^{1-\alpha} - \left(\frac{\epsilon \Gamma(2-\alpha)}{M} \right), \quad (3.5)$$

and therefore if we are employing a numerical scheme of $\mathcal{O}(h^p)$ over some interval I to preserve this accuracy we would need to use a T that satisfies (3.5) for all $t \in I$. As we shall see, this means that we will lose almost all of the computational benefits of the fixed memory principle and we explore this further here.

First we show the effect of a naive application of the fixed memory principle. In figure (1) we show the error occurring when applying the fixed memory principle (over an interval $I = [0, 1000]$) to the systems

$$D^{0.5}y = -y, \quad y(0) = 1, \quad (3.6)$$

and

$$D^{0.5}y = -y + t^2 + \frac{\Gamma(3)}{\Gamma(2.5)} t^{1.5}, \quad y(0) = 0 \quad (3.7)$$

respectively.

In Table 1 we compare the relative computational effort, over the same finite interval, normalising about the full calculation.

Figure 1 and Table 1 illustrate clearly the behaviour of the fixed memory principle: we see the percentage error growing rapidly as the length of the fixed interval I_r (and therefore the computational effort) decreases. This corresponds to loss of order of the original method through the application of the fixed memory. We can confirm that the main component of the error is due to the truncation of the integral. In Figure 2 we plot the error for systems (3.6) and (3.7) with a fixed memory window of 10 seconds, and in each case we show the errors for two different values of $h > 0$. The step size

| system | $T = \infty$ | $T = 100$ | $T = 50$ | $T = 25$ | $T = 10$ | $T = 5$ |
|--------|--------------|-----------|----------|----------|----------|---------|
| 3.6 | 1 | 0.1936 | 0.1015 | 0.0537 | 0.0243 | 0.0144 |
| 3.7 | 1 | 0.1988 | 0.1074 | 0.0598 | 0.0306 | 0.0208 |

Table 1: Relative computational cost, in floating point operations, for solving equations in systems (3.6) and (3.7).

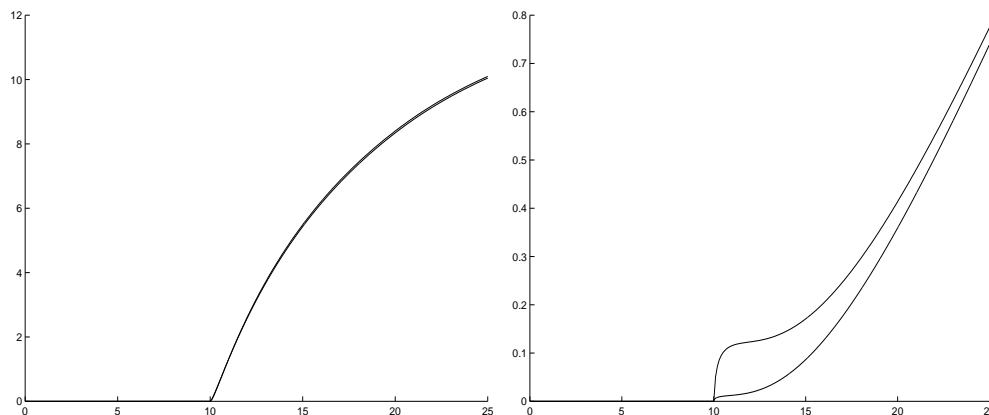


Figure 2: Percentage error occurring when using a memory of fixed length $T = 10$ with step sizes an order of magnitude different for systems (3.6) and (3.7) respectively.

used for the lower error curve is an order of magnitude smaller than that chosen for the upper error curve. It is clear that the overall error is much more dependent on the truncation of the integral than on the choice of step size.

In Table 2 we summarise the consequences of applying the theoretical approach described above (see equations (3.1) and (3.5)) to control the errors. We allow the finite memory interval to grow as the step length $h > 0$ decreases in order to preserve the order of the method. This time we can see that the error is kept under control, but at the expense of much greater computational cost.

The second row of the table gives the (uniform for all intervals of integration) values of T from (3.1) while the third row of the table records the values of T given by (3.5) for integration over a fixed interval $I = [0, 10000]$. We remark that, unless the interval over which we are finding the solution is very large indeed, the fixed memory principle with order preserved is unlikely to reduce significantly the computational effort compared with the full integral.

4 Nested Meshes and the Logarithmic Memory Principle

Clearly the problem with the fixed memory principle is that the approximation of the integrand by zero over the early part of the interval is too drastic. In this section we consider how the scaling property of the fractional integral can be used to control step size and use this to extend the fixed memory principle to a sequence of finite windows of differing step sizes which then cover (nearly) the whole of the interval of integration.

| | | | | |
|-------------------------------------|-----------|-----------|--------------|--------------|
| Error bound ϵ | 10^{-1} | 10^{-2} | 10^{-3} | 10^{-4} |
| T calculated from (3.1) | 56 | 5,600 | $5.6 * 10^5$ | $5.6 * 10^7$ |
| T from (3.5) for $I = [0, 10000]$ | 9987 | 9999 | 10000 | 10000 |

Table 2: Length of finite memory interval to guarantee error bound for system (3.6).

Nested Meshes Firstly we recall the scaling property of the fractional integral. Let $w > 0$ and f be a suitably integrable function then

$$I^\alpha f(t) = \int_0^t \frac{f(x)}{(t-x)^{1-\alpha}} dx, \quad (4.1)$$

and with w as above

$$I^\alpha f(wt) = \int_0^{wt} \frac{f(x)}{(wt-x)^{1-\alpha}} dx. \quad (4.2)$$

We make the substitution of x by wx to obtain

$$I^\alpha f(wt) = w^\alpha \int_0^t \frac{f(wx)}{(t-x)^{1-\alpha}} dx. \quad (4.3)$$

Additionally, for $p \in \mathbb{N}$ we have

$$I^\alpha f(w^p t) = w^{p\alpha} \int_0^t \frac{f(w^p x)}{(t-x)^{1-\alpha}} dx. \quad (4.4)$$

Thus if we use a numerical scheme, of the kind proposed by Lubich [15] or Diethelm [4] to obtain a discrete approximate of the convolution integral, the weights necessary to calculate $\Omega_h^\alpha f(nh) \approx I^\alpha f(nh)$ (using a step length h) can in principle be used to calculate $\Omega_{w^p h}^\alpha f(w^p nh) \approx I^\alpha f(nw^p h)$ (using a step length $w^p h$) for any $p \in \mathbb{N}$ simply by multiplying the resulting sum by $w^{p\alpha}$. That is

$$\Omega_h^\alpha f(nh) = \sum_{j=0}^n \omega_{n-j} f(jh) \Leftrightarrow \Omega_{w^p h}^\alpha f(nw^p h) = w^{p\alpha} \sum_{j=0}^n \omega_{n-j} f(jw^p h). \quad (4.5)$$

We define, for $h \in \mathbb{R}^+$, the mesh M_h by $M_h = \{hn, n \in \mathbb{N}\}$. If $w, r, p \in \mathbb{N}$, $w > 0$, $r > p$, then $M_{w^p h} \supset M_{w^r h}$. We then decompose the interval $[0, t]$, for fixed $T > 0$ in the following way:

$$[0, t] = [0, t - w^m T] \cup [t - w^m T, t - w^{m-1} T] \cup \dots \cup [t - wT, t - T] \cup [t - T, t]$$

where $m \in \mathbb{N}$ is the smallest integer such that $t < w^{m+1} T$.

Our approach is motivated by the fact that the singularity in the kernel of the fractional derivative (integral) occurs when $t = x$ and that $1/(t-x)^{1-\alpha} \rightarrow 0$ as $t \rightarrow \infty$ for $x \rightarrow 0$. This links with the scaling property to suggest that we could distribute our computational effort, over past time, logarithmically rather than uniformly. We use a step length of h over the most recent time interval $[t-T, t]$ and successively larger step lengths over earlier intervals in the following way.

We let $t, T, h \in \mathbb{R}$, $w^{m+1} T > t \geq w^m T$, $T > 1$, $h > 0$ with $t = nh$ for some $n \in \mathbb{N}$. We can rewrite the integral as

$$\begin{aligned} I_{[0,t]}^\alpha f(t) &= I_{[t-T,t]}^\alpha f(t) + \sum_{i=0}^{m-1} I_{[t-w^{i+1}T, t-w^i T]}^\alpha f(t) + I_{[0, t-w^m T]}^\alpha f(t) \\ &= I_{[t-T,t]}^\alpha f(t) + \sum_{i=0}^{m-1} w^{i\alpha} I_{[t-wT, t-T]}^\alpha f(w^i t) + w^{m\alpha} I_{[0, t-w^m T]}^\alpha f(w^m t). \end{aligned}$$

Here

$$I_{[t-a, t-b]}^\alpha f(t) = \frac{1}{\Gamma(\alpha)} \int_{t-a}^{t-b} \frac{f(x)}{(t-x)^{1-\alpha}} dx.$$

If in the discrete approximation of $\Omega_{[0,t]}^\alpha f(t)$, we use the approximation

$$\Omega_{h[t-w^{i+1}T, t-w^i T]}^\alpha f(t) \approx \Omega_{w^i h[t-w^{i+1}T, t-w^i T]}^\alpha f(t),$$

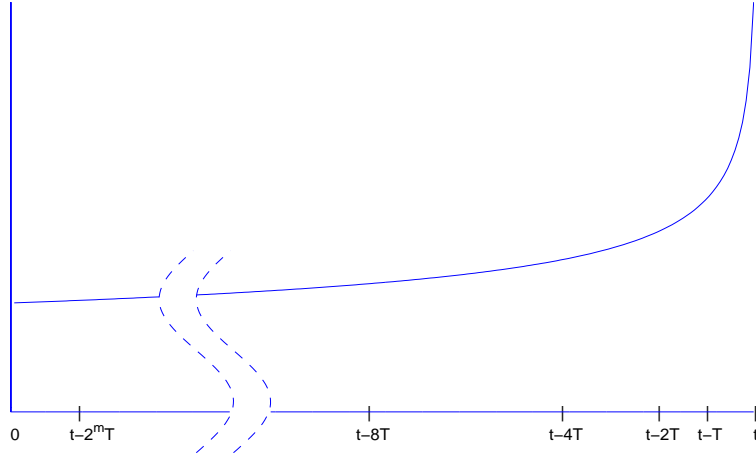


Figure 3: Example distribution of time intervals shown under graph of convolution kernel function $= 1/(t - x)^{0.25}$ as x varies from 0 to t .

then we can substitute

$$w^{i\alpha} \Omega_h^\alpha [t-wT, t-T] f(t) = \Omega_{w^i h}^\alpha [t-w^{i+1}T, t-w^i T] f(t)$$

using the scaling property described above.

We illustrate this approach in Figure 3 (drawn here for $w = 2$). We use the finest mesh over the right-most intervals with successively coarser meshes as we move to the left. The figure illustrates the fading memory of the integrand which justifies our choice of larger step-sizes as we move closer to the origin.

Figure 3 shows that there may be an initial part (depending on t) of the integrand that is not covered by our algorithm. (In the diagram this is part of the interval $[0, t - 2^m T]$ which may not be an exact multiple of the current step-length $2^m h$). We could use a special starting quadrature (similar in concept to those used in [14]) to evaluate the initial part of the integrand and to preserve the exact order of the method or we can simply evaluate this short initial interval with step length h . In either case we can prove that the order of the method is preserved. However, it turns out that the fading memory of the kernel means that the error introduced by simply truncating the integrand at this point will be small. We have undertaken numerical experiments that indicate that the small extra computational cost of calculating over this first step would not usually be necessary in practice.

We conclude this section with a Convergence Theorem for the nested mesh scheme.

Theorem 4.1 The nested mesh scheme preserves the order of the underlying quadrature rule on which it is based.

The proof of the Theorem is immediate: For integration over a fixed interval $[0, t]$ the choice of T fixes (independent of h) the number of subranges over which the integral is evaluated. One considers the underlying quadrature rule of order $\mathcal{O}(h^p)$ and writes down, assuming y to be sufficiently smooth, bounds on the errors over each of the subintervals used. (One could use, for example, the asymptotic error expansions given in [10] for one such quadrature scheme, but it is sufficient to write the sum of the fixed number of terms, each of $\mathcal{O}(h^p)$). It is now simple to see that the total error is $\mathcal{O}(h^p)$.

Estimation of Relative Computational Cost The method we propose here has two benefits in terms of computational cost. The first benefit derives from the fact that we evaluate a fixed number of quadrature coefficients and then re-use the coefficients over successive intervals. Therefore the cost of calculating the quadrature coefficients is fixed (once h and T are chosen), no matter how

| | Full | NestedMesh | FMP |
|--------|--------|-------------|-----|
| T | n | n | n |
| wT | wn | wn | n |
| w^2T | w^2n | $(2w - 1)n$ | n |
| w^3T | w^3n | $(3w - 2)n$ | n |
| w^4T | w^4n | $(4w - 3)n$ | n |

Table 3: Theoretical relative computational cost of methods

| | Interval Length L | | | | | |
|------------------------|---------------------|------|-------|-------|--------|--------|
| | 25 | 50 | 100 | 250 | 500 | 250* |
| Fixed Memory Principle | 2.3 | 4.8 | 10.0 | 25.3 | 50.8 | 100.1 |
| Logarithmic Memory | 5.5 | 14.6 | 36.6 | 115.8 | 268.9 | 462.3 |
| Full Memory | 6.3 | 25.0 | 100.1 | 625.6 | 2501.1 | 2501.2 |

Table 4: Comparison of computational cost for actual calculation (MFlops)

large the interval $[0, t]$ may become. Since the calculation of fractional quadrature weights is itself an expensive procedure, this gives an initial saving in computational cost. The second (and potentially more substantial) saving in computational cost comes when the algorithm is applied to solve the equation. In Table 4 we compare the theoretical computational costs of the Nested Mesh method introduced here with those of the full algorithm and the Fixed Memory Principle.

Let S_D be the number of times each sample point is referenced in time $[0, w^r T)$ in the full algorithm then $S_D = w^r n(w^n - 1)/2$. Let S_{NM} be the number of times each sample point is referenced in the Nested Mesh modification of the algorithm then

$$S_{NM} = \frac{w^r - 1}{w - 1} S_w + w^3 n(1 + 2w + \dots + rw^{r-1}), \quad (4.6)$$

where $S_w = wn(wn - 1)/2$. Therefore in terms of w the original method is $\mathcal{O}(w^{2m})$ whilst the Nested Mesh variant is $\mathcal{O}(w^{m+1})$.

Remark 1ex The value for S_{NM} is calculated in the following way: Over the interval $[0, wT)$ there are $\frac{wn(wn+1)}{2} - 1$ sample point references. On the interval $[wT, w^2T)$ there are $w((wn + 1) + (wn + 2) + \dots + 2wn) = w^3n + w^2n(wn + 1)/2$ sample point references. Over successive intervals $[w^{r-1}T, w^rT)$ there are $(r - 1)w^{r+1}n + w^r n(wn + 1)/2$ sample point references. The expression in equation (4.6) is derived by summing these terms.

In a particular example calculation, we found that the comparative CPU costs we measured were as shown in Table 4 (see [11] for a further example) and this shows the practical benefit of our proposed new method. We solved equation (3.6) over successive intervals $[0, L]$ using a step length $h = 10^{-2}$ and finite memory window $T = 5$. In the final column (marked *), we give an additional comparison for $h = 5 \times 10^{-3}$. The results are recorded in MFlops by Matlab 5.3.

Remark 1ex One needs to interpret the results in Table 4 in the light of the errors given by the methods. In Tables 5 and 6 we summarise the percentage errors and computational costs found in solving equation (3.7) by the three methods.

| | Method and Memory Length T | | | | | | |
|---------|------------------------------|-----------|-----------|-----------|--------|---------|---------|
| | Diethelm | LM T=5 | LM T=10 | LM T=20 | FM T=5 | FM T=10 | FM T=20 |
| Error % | 10^{-4} | 10^{-2} | 10^{-3} | 10^{-3} | 10 | 5 | 2 |
| MFlops | 101.1 | 37.5 | 59.3 | 89.0 | 10.9 | 20.2 | 37.2 |

Table 5: Comparison of percentage errors and computational costs for actual calculation (t=100)

| | Method and Memory Length T | | |
|---------|------------------------------|-----------|-----------|
| | Diethelm | LM $T=20$ | FM $T=20$ |
| Error % | 10^{-7} | 10^{-4} | 3 |
| MFlops | 628 | 316 | 99 |

Table 6: Comparison of percentage errors and computational costs for actual calculation ($t=250$)

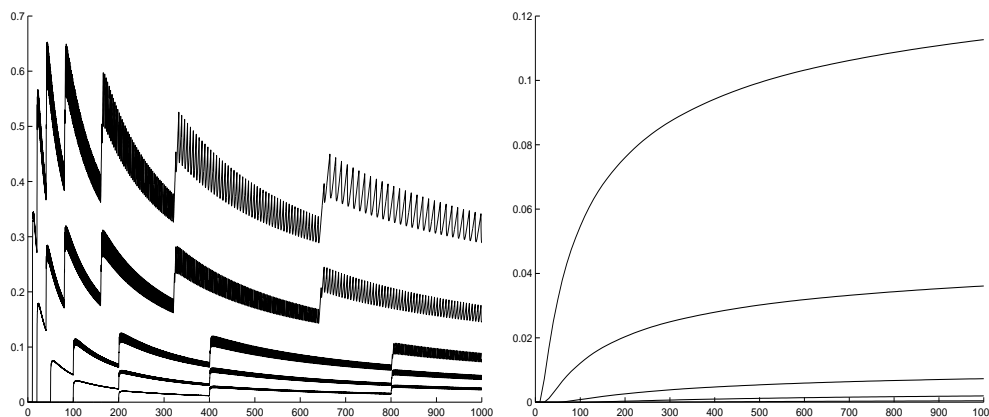


Figure 4: Percentage error occurring when using nested meshes with a memory of fixed length $T = 5, 10, 25, 50, 100$, in systems (3.6) and (3.7) respectively.

5 Improving Convergence by Extrapolation

Diethelm and Walz have shown in [10] that the numerical solution to simple equations of the form (1.3), given by the method in Diethelm [4], possesses an asymptotic expansion of the form

$$x_n = x(t_n) + \sum_{\mu=1}^{M_1} \gamma_{\mu} n^{-\lambda_{\mu}} + o(n^{-\lambda_{M_1}}) \quad (5.1)$$

where the values λ_k satisfy

$$\begin{aligned} \lambda_k &= 2j + 1 - \alpha, & k &= 3j, \\ \lambda_k &= 2j - \alpha, & k &= 3j + 1, \\ \lambda_k &= 2j, & k &= 3j + 2. \end{aligned}$$

This allows a sequence of improved solutions to be calculated by linear extrapolation from the relationship

$$y_i^k = y_{i+1}^{(k-1)} + \frac{y_{i+1}^{(k-1)} - y_i^{(k-1)}}{b^{\lambda_k} - 1} \quad (5.2)$$

where $n_{i+1} = bn_i$. We have applied the same extrapolation technique to system (3.6), with $b = 2$. In Table 7 we summarise the computational effort required to calculate the function values (for $t = 500$). In Table 8 we give the function values, for the same value of t , calculated with the full Diethelm algorithm, and their improved descendants. In Table 9 we give the function values calculated with the log-Diethelm algorithm, and their improved descendants. For the values of h chosen in these examples, the theory predicts an error in the extrapolated scheme of $\mathcal{O}(10^{-5})$. The difference in solutions obtained by the two methods is $< 3 \times 10^{-5}$ and this indicates that the extrapolation scheme has been successful. The extrapolation schemes provide rapid convergence where high accuracy is required and for reasonably accurate long term predictions (at lower computational cost) we believe our technique provides a particularly effective solution.

| <i>Time</i> | 10 | 20 | 30 | 40 | 50 | 100 | 250 | 500 |
|--------------------|-----|-----|-----|------|------|------|-------|--------|
| <i>Full</i> | 0.9 | 3.5 | 7.8 | 13.8 | 21.5 | 84.5 | 531.2 | 2126.7 |
| <i>Logarithmic</i> | 0.8 | 2.0 | 3.5 | 5.1 | 6.7 | 16.0 | 48.2 | 109.1 |

Table 7: Cost of calculating function values required for the extrapolation scheme (in MFlops)

| | | | | | | | | |
|-----------|-----------|-----------|-----------|-----------|-----------|--|--|--|
| 0.0252074 | | | | | | | | |
| | 0.0252064 | | | | | | | |
| 0.0252068 | | | 0.0252063 | | | | | |
| | | 0.0252063 | | | 0.0252062 | | | |
| 0.0252065 | | | | 0.0252062 | | | | |
| | | 0.0252062 | | | | | | |
| 0.0252063 | | | | | | | | |

Table 8: Extrapolation scheme for values with full algorithm

Remark 1ex The exact solution of (3.6) can be written in the closed form

$$y(t) = \sum_{n=0}^{\infty} \frac{(-t^{\frac{1}{2}})^n}{\Gamma(\frac{n}{2} + 1)}. \quad (5.3)$$

6 Acknowledgements

Charles Simpson was supported in this work by a Chester College Research Student bursary. The authors are glad to acknowledge the helpful comments of the referees.

References

- [1] C.T.H.Baker and M.S.Derakhshan, FFT techniques in the numerical solution of convolution equations, *JCAM*, 20, 5–24, 1987.
- [2] L.Blank, *Numerical treatment of differential equations of fractional order*, Numerical Analysis Report 287. Manchester Centre for Computational Mathematics, 1996.
- [3] M.Caputo, Linear models of dissipation whose Q is almost frequency independent II, *Geophys. J. Royal Astronom. Soc.*, 13, 529–539,1967.
- [4] K.Diethelm, An Algorithm for the Numerical Solution of Differential Equations of Fractional Order, *Elect. Trans. Num. Anal.*, 5, 1–1,1997.
- [5] K.Diethelm, Numerical approximation of finite-part integrals with generalised compound quadrature formulae, *IMA J. Num. Anal.*, 17, 479–493, 1997.

| | | | | | | | | |
|-----------|-----------|-----------|-----------|-----------|-----------|--|--|--|
| 0.0247011 | | | | | | | | |
| | 0.0250612 | | | | | | | |
| 0.0249339 | | | 0.0251450 | | | | | |
| | | 0.0251240 | | | 0.0251791 | | | |
| 0.0250568 | | | | 0.0251731 | | | | |
| | | 0.0251608 | | | | | | |
| 0.0251240 | | | | | | | | |

Table 9: Extrapolation scheme for values with log algorithm

- [6] K. Diethelm and A. Freed, On the solution of nonlinear fractional order differential equations used in the modelling of viscoplasticity, *Scient. Comp. in Chem. Eng. II — Computational Fluid Dynamics, Reaction Engineering, and Molecular Properties*, F. Keil, W. Mackens, H. Voß and J. Werther (eds), 217–224, Springer, Heidelberg, 1999.
- [7] K. Diethelm and N.J. Ford, Analysis of Fractional Differential Equations, *J. Math. Anal. Appl.*, to appear.
- [8] K. Diethelm and N.J. Ford, Numerical solution of the Bagley Torvik equation, to appear.
- [9] K. Diethelm and N.J. Ford, Fractional differential equations involving derivatives of several orders and their numerical solution, to appear.
- [10] K. Diethelm and G. Walz, Numerical Solution of Fractional Order Differential Equations by Extropolation, *Numer. Alg.*, 16, 231–253, 1997.
- [11] N.J. Ford and A.C. Simpson, Numerical and analytical treatment of differential equations of fractional order, *Proceedings of IMACS International Conference on Scientific Computing and Mathematical Modeling*, Milwaukee, 2000.
- [12] N.J. Higham, *Accuracy and Stability of Numerical Algorithms*, SIAM Publications, Philadelphia, 1996.
- [13] C. Lubich, *Discretized Fractional Calculus*, *SIAM J. Math. Anal.*, 17, 704–719, 1986.
- [14] C. Lubich, Fractional linear multistep methods for Abel-Volterra integral equations of the second kind, *Math. Comp.*, 45, 463–469, 1985.
- [15] C. Lubich, Convolution Quadrature and Discretized Operational Calculus. II, *Num. Math.*, 52, 413–425, 1988.
- [16] A.R. Nkamnang, *Diskretisierung von mehrgliedrigen Abelschen Integralgleichungen und gewöhnlichen Differentialgleichungen gebrochener Ordnung*, PhD Dissertation, Freie Universität Berlin, 1998.
- [17] K.B. Oldham and J. Spanier, *The Fractional Calculus*, Academic Press, San Diego, 1974.
- [18] I. Podlubny, *Fractional Differential Equations*, Academic Press, San Diego, 1999.
- [19] I. Podlubny, *Numerical Solution of Ordinary Fractional Differential Equations by the Fractional difference Method*, Proceedings of the Second International Conference in Difference Equations, Gordon and Breach Scientific Publishers, 507–515, 1997.
- [20] S.G. Samko, A.A. Kilbas, O.I. Marichev, *Fractional Integrals and Derivatives*, Gordon and Breach Science Publishers, 1993.