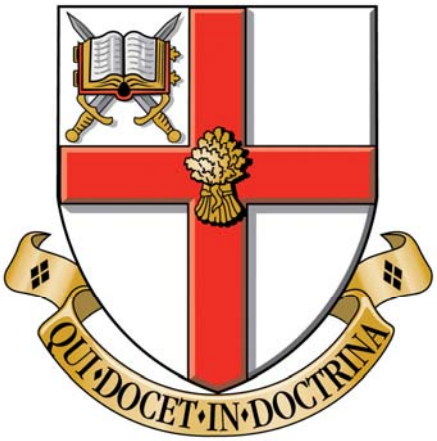

University of Chester

Author(s): Kai Diethelm; Judith M Ford; Neville J Ford; Marc Weilbeer

Title: Pitfalls in fast numerical solvers for fractional differential equations

Date: 2006

Originally published in: Journal of Computational and Applied Mathematics

Example citation: Diethelm, K., Ford, J. M., Ford, N. J., & Weilbeer, M. (2006). Pitfalls in fast numerical solvers for fractional differential equations. *Journal of Computational and Applied Mathematics*, 186, 482-503

Version of item: Author's preprint

Available at: http://hdl.handle.net/10034/11808

Pitfalls in fast numerical solvers for fractional differential equations

Kai Diethelm [1] and Judith M. Ford [2] and Neville J. Ford [3] and Marc Weilbeer [1]

Abstract

We consider the problem of implementing fast algorithms for the numerical solution of initial value problems of the form $x^{(\alpha)}(t) = f(t, x(t))$, $x(0) = x_0$, where $x^{(\alpha)}$ is the derivative of x of order α in the sense of Caputo and $0 < \alpha < 1$. We review some of the existing methods and explain their respective strengths and weaknesses. We identify and discuss potential problems in the development of generally applicable schemes.

Keywords: fractional differential equation, high order method, backward differentiation method.

AMS Subject Classifications: Primary 65L05; secondary 65L06, 65R20, 26A33.

1 Introduction

This paper considers the properties of high order methods for the solution of fractional differential equations. There is a growing demand for such methods from modellers whose work leads to linear and nonlinear equations involving derivatives of fractional order and yet there seems to be no well-understood method of reasonably high order that can be used to generate a reliable approximate solution.

[1] Institut für Angewandte Mathematik, Technische Universität Braunschweig, Pockelsstraße 14, 38106 Braunschweig, Germany ({k.diethelm, m.weilbeer}@tu-bs.de). Supported by US Army Medical Research and Material Command Grant No. DAMD-17-01-1-0673 to the Cleveland Clinic. Current address for KD: GNS Gesellschaft für Numerische Simulation mbH, Am Gaußberg 2, 38114 Braunschweig, Germany (diethelm@gns-mbh.com)

[2] Department of Mathematics, UMIST, PO Box 88, Manchester M60 1QD, United Kingdom (j.ford@umist.ac.uk). EPSRC Research Fellow supported by grant ref: GR/R95982/01. Current address: Royal Liverpool Children's NHS Trust, Eaton Road, Liverpool L12 2AP, UK

[3] Corresponding author, Department of Mathematics, University College Chester, Parkgate Road, Chester CH1 4BJ, United Kingdom (njford@chester.ac.uk).

Our investigations are motivated by a few classical and many very recent applications of fractional differential equations. Among the classical problems we mention areas like the modelling of the behaviour of viscoelastic materials in mechanics (studied since the 1980s [35]) and applications of Abel-Volterra equations in superfluidity [24]. More recently fractional calculus has been applied to continuum and statistical mechanics for viscoelasticity problems, Brownian motion and fractional diffusion-wave equations [27] and the description of the propagation of a flame [21,23]. Newer studies are also done, among others, in the area of modelling of soft tissues like mitral valves or the aorta in the human heart [15]. It is evident that these applications require not only fast but in particular *reliable* numerical methods.

In our earlier work we have presented (see [5,9,10,12,14]) several methods for the approximate solution of differential equations of fractional order. In the main these have been of low order, but they have nevertheless attracted interest because of the relative ease of application and the reliable results that we have been able to give relating to convergence and stability of the methods. We have also shown (see, for example, [10,13,14]) that the underlying order of our methods may be improved (through extrapolation schemes) leading to methods of higher order.

In the 1980s there was a surge of interest in developing higher order numerical methods for Abel-Volterra integral equations (of which fractional differential equations form a subclass) and detailed theoretical results were given for these methods at that time. However these so-called fractional multistep methods have proved to be of more theoretical than practical use over the intervening two decades (although they have been included in the NAG Fortran Library as a method to solve certain Abel-Volterra equations). One purpose of this paper is to assess why this has been the case, and to give a clear direction to further research that will lead to more practical methods for today's applications.

This paper is structured as follows: first we describe in greater detail the class of problems that we seek to solve and we set out clear objectives for a well-behaved numerical scheme, then we review the available algorithms for the solution of these equations against the objectives we have set. We consider the work published in the 1980s on fractional multistep methods and review its strengths from a theoretical viewpoint and show how the methods can be applied very effectively to the types of problems prevalent at that time. We consider more recent model equations and highlight some of the pitfalls in trying to implement fractional multistep methods in this case.

We conclude with some advice to users on the choice of numerical schemes for the solution of particular types of equation. We also give a statement of the issues that we regard as the most important for algorithm developers who wish to produce useful higher order methods for practical application.

2 Objectives

We consider the solution of fractional differential equations of the form

$$x^{(\alpha)}(t) = f(t, x(t)), \qquad x^{(k)}(x_0) = x_0^{(k)} \quad (k = 0, 1, \ldots, \lceil \alpha \rceil - 1), \tag{1}$$

where α is some positive non-integer number. Here the notation $x^{(\alpha)}$ is used for the Caputo type fractional derivative, defined by

$$x^{(\alpha)}(t) := \frac{1}{\Gamma(m-\alpha)} \int_0^t (t-\tau)^{m-\alpha-1} x^{(m)}(\tau)\, \mathrm{d}\tau$$

where $m := \lceil \alpha \rceil$.

One can also define the Caputo fractional derivative based on classical Riemann-Liouville differential operators of fractional order $\alpha > 0$ which are defined by,

$$D^{\alpha} x(t) := \frac{1}{\Gamma(m-\alpha)} \frac{d^m}{dt^m} \int_0^t \frac{x(u)}{(t-u)^{\alpha-m+1}} du$$

where m is the integer defined by $m - 1 < \alpha < m$ (see [28,34]). The standard (Riemann-Liouville) approach [34, §42], is then to define the initial conditions for solving the fractional differential equation in the form

$$\frac{d^{\alpha-k}}{dt^{\alpha-k}} x(t)|_{t=0+} = b_k, \qquad k = 1, 2, \ldots, m = \lfloor \alpha + 1 \rfloor,$$

with given values b_k. In other words we would need to specify some values of the fractional derivatives of the function x. In practical applications, these values are frequently not available, and it may not even be clear what their physical meaning is. By contrast Caputo [4] suggested that one should incorporate the classical derivatives (of integer order) of the function x, as they are commonly used in initial value problems with integer-order equations, into the fractional-order equation, giving the alternative (equivalent) formulation of the Caputo fractional differential equation as

$$x^{(\alpha)}(t) := D^{\alpha}(x - T_{m-1}[x])(t) = f(t, x(t)), \tag{2a}$$

where $T_{m-1}[x]$ is the Taylor polynomial of order $(m-1)$ for x, centered at 0. For $\alpha \in \mathbb{N}$, one simply defines $x^{(\alpha)}(t)$ to be the usual differential operator of order α. Then, one can specify the initial conditions in the classical form

$$x^{(k)}(0) = x_0^{(k)}, \qquad k = 0, 1, \ldots, m-1. \tag{2b}$$

For more details of the relationship between Caputo and Riemann-Liouville fractional differential operators see [29].

In the common cases (which we shall mainly be concerned with here), we have $\alpha \in (0, 1)$. The functions x and f may in general be vectors but for our purposes here we shall gain all

the appropriate insights by considering the scalar case. As we saw in our papers [7,8] the vector case permits us to analyze also approximate methods for the solution of multi-term fractional differential equations of the form

$$x^{(\alpha_k)}(t) = f(t, x(t), x^{(\alpha_1)}(t), \ldots, x^{(\alpha_{k-1})}(t)) \tag{3}$$

(in combination with appropriate initial conditions), and for this reason we can regard the insights presented in the current work as having wide application to linear and non-linear fractional differential equations of both single and multiple orders.

The existing literature on fractional differential equations (and indeed on Abel-Volterra integral equations) tends to focus on particular values for the order α. The value $\alpha = \frac{1}{2}$ is especially popular. This is because in classical fractional calculus, many of the model equations developed used these particular orders of derivatives. In modern applications (see, for example, [16]) much more general values of the order α appear in the equations and therefore one needs to consider a little more carefully how methods can be chosen to solve equations of more or less arbitrary order.

Perhaps the best way to set out our objectives for a good numerical scheme for fractional differential equations is to base them on the well-established desirable characteristics of solution methods for ordinary (integer-order) differential equations (see, for example, [19,22]). Considerable research effort has been invested in schemes for ordinary differential equations over many years and therefore to set our sights on the best features of existing schemes for ordinary differential equations sets challenging targets for the solution of fractional order equations where the underlying problems are less well understood and the exact solution that we are attempting to approximate is typically less well-behaved than in the integer order case.

Thus we desire numerical schemes that are *convergent*, *consistent* and *stable* (see, for example, [22, pp. 21ff.] or [19, pp. 391ff.]). We also desire that our scheme should be reasonably easy to program on a computer and that the resulting computer code should execute reliably and reasonably quickly.

Most elementary numerical schemes are based on the use of a time step of fixed length. Here the *order of convergence* of the numerical scheme is particularly important in that it helps us to know how quickly the approximate solution at some fixed point in time t will approach the exact solution as $h \to 0$. To be precise: a numerical scheme with fixed step length $h > 0$ has order $p \in \mathbb{N}$ if

$$E(T) := \sup_{t \in [0,T]} |\tilde{x}_h(t) - x(t)| = \mathcal{O}(h^p) \text{ as } h \to 0 \tag{4}$$

where $\tilde{x}_h(t)$ represents the approximate solution at $t \in [0, T]$ evaluated using the step length h.

In practice we can expect there to be some restriction on the order of convergence of methods if we insist that they also exhibit the required stability properties (see, for example, [19]). This is well documented. However the behaviour of the error in (4) is not

so often made precise although it is well understood. The expression (4) means that the error $E(T)$ has an expansion whose dominant term has the form $A_T h^p$. Now until we know something about the value of the constant A_T we cannot predict the size of the actual error. Typically the value of A_T grows as T increases and one can use a Gronwall-type inequality to estimate this value (see [19, p. 395]). In the case of an integer order equation this leads to an estimate of the form

$$A_T \leq \alpha e^{\kappa T} \tag{5}$$

illustrating the fact that the growth in the error as T increases is at an exponential rate even though the actual error at any fixed point T will have an $\mathcal{O}(h^p)$ convergence to zero as $h \to 0$. One can derive a corresponding Gronwall-type lemma for a fractional order equation (see for example [8]) and in this case we have the relation

$$A_T \leq \alpha E_\alpha(\kappa T^\alpha) \tag{6}$$

where α is the fractional order of the equation and where E_α represents the Mittag-Leffler function with parameter α.

One concludes from this overview that order alone does not necessarily give a good guide to the actual size of the error in a numerical approximation. Indeed, for some fixed $h > 0$ it can turn out that a lower order method gives a better approximation than a higher order method. This would be the case, for example, when the respective values of A_T were quite different in magnitude. The order of the method tells us rather how the error will improve as successively smaller values of $h > 0$ are used. We should note also that the theoretical order of the method is given only in the limit as $h \to 0$ and therefore this might not be seen in the values of h chosen for any specific experiment. Moreover we will demonstrate that certain other effects play a significant role in this context. As we shall see below, it is sometimes very difficult to calculate the weights of a numerical method with high accuracy, and then it may happen that the errors introduced in this way spoil the entire calculation. In particular it is possible that — in contradiction to what one would expect — the results get significantly worse as we let $h \to 0$ because the number of steps increases, and since such errors are accumulated over the number of steps, the effect is magnified.

Later on in this paper we shall consider also the expected execution time of a numerical method. This is typically expressed in terms of the expected number of calculations involved in running the algorithm to completion. We often therefore refer to a method as being of order N^k where we are using the value $N = T/h$ for some fixed $T > 0$. Now once again we tend to assume that the time taken to compute the solution using a particular method is dependent on the value of k and easily forget to mention that the constant multiplying the value N^k may well differ significantly between different methods.

We are now in a position to define our objectives. We seek a numerical scheme that is

(1) convergent,
(2) consistent of some reasonable order h^p,
(3) stable,

(4) reasonably inexpensive to run,
(5) reasonably easy to program.

In fact this last objective has been largely disregarded by previous authors. As we shall see, it turns out to have a key role to play in this particular paper.

3 Fractional multistep methods

Historically, the fractional multistep methods of Lubich [25,26] were among the first methods to be introduced. We will recall their derivation and state some important properties and then be concerned with their numerical implementation.

3.1 Analytical background

We first use the fact [6] that the initial value problem

$$x^{(\alpha)}(t) = f(t, x(t)), \qquad x(0) = x_0, \tag{7}$$

with $0 < \alpha < 1$ (we shall from now on restrict our attention to this particularly important special case) is equivalent to the weakly singular Volterra equation

$$x(t) = x_0 + \frac{1}{\Gamma(\alpha)} \int_0^t (t-\tau)^{\alpha-1} f(\tau, x(\tau))\, \mathrm{d}\tau. \tag{8}$$

Therefore, we will first look at a class of methods for the numerical approximation for convolution integrals of the form

$$\frac{1}{\Gamma(\alpha)} \int_0^t (t-\tau)^{\alpha-1} g(\tau)\, \mathrm{d}\tau. \tag{9}$$

The construction of these methods (see [26]) is based on the well known concept of linear multistep methods for first order equations, which we assume to be given in terms of their characteristic polynomials ρ and σ. Using these two polynomials in z (the backward difference operator) we can construct the generating function $\omega(z) := \sigma(1/z)/\rho(1/z)$ and look at the Taylor expansion of its αth power,

$$(\omega(z))^\alpha = \left(\frac{\sigma(1/z)}{\rho(1/z)}\right)^\alpha = \sum_{j=0}^{\infty} \omega_j z^j, \tag{10}$$

thus defining (in an implicit way) the values ω_j, $j = 0, 1, 2, \ldots$; it is evident from eq. (10) that they depend on the choice of α, but since α is constant, we have decided not to denote this explicitly in order to keep the notation simple.

The coefficients ω_j are called *convolution weights*. They can be used to construct a quadrature formula

$$\frac{1}{\Gamma(\alpha)}\int_0^{nh}(nh-\tau)^{\alpha-1}g(\tau)\,\mathrm{d}\tau \approx h^\alpha\sum_{j=0}^{n}\omega_{n-j}g(jh),$$

and it can be shown that this method gives $\mathcal{O}(h^p)$ accuracy if the underlying multistep method is of the order p and the function g is sufficiently smooth. However, in the application that we have in mind the integrand function g is typically *not* smooth. To be precise, if f is a smooth function then the solution of the fractional differential equation (7) will have an asymptotic expansion of the form

$$x(t)=\sum_{\gamma\in\mathcal{A}}c_\gamma t^\gamma+o(t^{\max\mathcal{A}}) \tag{11}$$

as $t\to 0$ where

$$\mathcal{A}:=\{\gamma=j+\ell\alpha : j,\ell\in\{0,1,2,\ldots\},\gamma\le p-1\}$$

with some suitable $p>0$. Therefore, in order to construct a reasonable numerical method for our problem, it is not sufficient only to look at the convolution weights. Rather we need a second set of weights w_{nj}, known as *starting weights*, which take into account the asymptotic behaviour of the exact solution x near the origin (which is more complicated than in the case of a first order equation), and are chosen in such a way that the quadrature rule

$$\frac{1}{\Gamma(\alpha)}\int_0^{nh}(nh-\tau)^{\alpha-1}g(\tau)\,\mathrm{d}\tau \approx h^\alpha\sum_{j=0}^{n}\omega_{n-j}g(jh)+h^\alpha\sum_{j=0}^{s}w_{nj}g(jh), \tag{12}$$

(with some fixed s and $n\ge s$) is exact whenever $g(t)=t^\gamma$ with $\gamma\in\mathcal{A}$. Evidently, this is (for fixed n) a linear system of equations that can be used to determine the starting weights w_{nj}, $j=0,1,\ldots,s$, since all the other quantities appearing in the equations are known. The total number of equations is equal to the cardinality of the set $\mathcal{A}$, and therefore it is evident that the (as yet unspecified) parameter s must be chosen as $s=\operatorname{card}\mathcal{A}$. We thus find that the starting weights w_{nj} are obtained by solving the linear system

$$h^\alpha\sum_{j=0}^{s}w_{nj}(jh)^\gamma=\frac{1}{\Gamma(\alpha)}\int_0^{nh}(nh-\tau)^{\alpha-1}\tau^\gamma\,\mathrm{d}\tau-h^\alpha\sum_{j=0}^{n}\omega_{n-j}(jh)^\gamma,\quad \gamma\in\mathcal{A}. \tag{13}$$

The matrix of coefficients $(a_{ij})=(h^\alpha(jh)^{\gamma_i})$ of this system is an exponential Vandermonde matrix (which is a generalized Vandermonde matrix with real exponents, see [31]) and hence regular but not well conditioned. Its precise condition number depends on the value of α in a very subtle way. For example, if $\alpha=1/M$ with some integer M then it can be rewritten by an obvious change of variables in the form of a classical Vandermonde matrix which is mildly ill-conditioned. If, however, $\alpha=1/M-\epsilon$ with some small $|\epsilon|$ and $p\ge 2$, then the set $\mathcal{A}$ will contain the elements 1 and $M\alpha=1-M\epsilon$, and hence the matrix will have two almost identical columns and therefore an extremely bad condition number. An additional aspect of this system is that, as already remarked in [26, §4.2], the evaluation of the right-hand side of (13) suffers from cancellation of digits. As we shall see in the later sections, the combination of these two problems may have serious adverse effects for the entire scheme.

Assuming that we have calculated the starting weights in some way, we can use the resulting quadrature formula as given in (12) to construct a scheme for the approximate solution of the Volterra equation (8) according to (see [25])

$$x_n = x_0 + h^\alpha \sum_{j=0}^{n} \omega_{n-j} f(jh, x_j) + h^\alpha \sum_{j=0}^{s} w_{nj} f(jh, x_j) \qquad (n = 1, 2, \ldots, N). \tag{14}$$

It is evident from eq. (14) that the entire calculation process can be decomposed into two phases, the starting phase $n \leq s$ and the main phase $n > s$.

The s equations of the starting phase all contain the unknown approximations $x_1, x_2, \ldots, x_s$, and so we are dealing with a fully coupled nonlinear system of s equations in s unknowns. There are essentially two different ways to get hold of the values $x_1, x_2, \ldots, x_s$: We can either try to solve the nonlinear system by a suitable algorithm (typically a Newton method, assuming that proper starting values can be determined), or we can revert to a different numerical scheme for the solution of the given fractional differential equation (19), use this for the approximation of the solution at the points h, $2h$, $\ldots$, sh and proceed to the main phase with these instead of the solutions of the nonlinear system.

For the main phase, we can proceed in the usual step by step manner because the equations are now uncoupled. That is, the nth equation contains x_n as the only unknown quantity because we have computed $x_1, x_2, \ldots, x_{n-1}$ in the previous calculations. Of course, in the general case the equations will still be nonlinear, and so we will have to use a (one-dimensional) Newton method to solve each of them individually.

Recalling our objectives 4 and 5 from §2 we next point out how the above theoretical scheme was used to construct the fast algorithm in [18] for the case $\alpha = \frac{1}{2}$ and how it can be transferred for other choices of α. We focus on the computation of the convolution and starting weights since those are the parts where most problems arise.

3.2 Computation of quadrature weights

In [18] fast algorithms are developed for the computation of the convolution and starting weights in the case $\alpha = \frac{1}{2}$. These methods are based in part on the Fast Fourier Transform which will only perform well if the number of weights is chosen accordingly. While this is not a major drawback, the use of Newton's method for formal power series in [18, §3] for the computation of the convolution weights is only applicable for the special cases where α is a unit fraction. Thus for general choices of $\alpha \in (0, 1)$ a different method needs to be developed.

Assuming that the generating function $\omega(z) = \sigma(1/z)/\rho(1/z)$ of the underlying non-fractional linear multistep method is analytic (which is true for the backward differentiation formulae (BDF) methods) one can prove using automatic differentiation techniques (we refer to [30, §5] for some basic principles) that the convolution weights of the fractional linear multistep method (i.e. the Taylor coefficients of the generating function $\omega(z)$

to its αth power) can be computed by

$$\omega_j = \frac{1}{j u_0} \sum_{i=0}^{j-1} [\alpha(j-i) - i] \omega_i u_{j-i}. \tag{15}$$

Here the values $u_j, j = 0, 1, 2, \ldots$ denote the Taylor expansion coefficients of the generating function $\omega(z)$ of the underlying non-fractional linear multistep method. In the case of the classical (integer) BDF method of order p the generating function $\omega(z)$ is a polynomial of order p, given by

$$\sum_{j=1}^{p} \frac{1}{j} (1-z)^j.$$

Formula (15) works equally well for all choices of α. In particular the evaluation of formula (15) is fast since all values u_j, $j = p+2, p+3, p+4, \ldots$ are zero for a given order p and therefore the sum in (15) consists of only $p+1$ non-zero summands. Thus we have a fast and easily implementable formula for the computation of the convolution weights. We next focus on the computation of starting weights.

Except for the remarks in [18, §3] and [26, §4.2] about the ill-conditioning and cancellation of digits in the equation system (13) not much is said about the computation of the starting weights. The reason for this is the fact that even though the system (13) is ill-conditioned and cancellation of digits occurs for larger numbers of mesh points, in the case $\alpha = \frac{1}{2}$ a simple linear system solver produces starting weights for which the *residual*, given by (18), is small. We shall see in §5 that even the case $\alpha = \frac{1}{2}$ exhibits some problems, and for choices of α different from $\frac{1}{2}$ the problem of solving the system (13) so that the residual stays small becomes more difficult. One therefore might try especially adapted algorithms for the computation of the starting weights.

The equation system (13) is ill-conditioned in general but it also exhibits a special Vandermonde-type structure. In the cases of α being a unit fraction the coefficient matrix is a classical Vandermonde matrix. Hence an algorithm exploiting this structure may prove useful. Given the fact that we have to solve the system (13) for as many right-hand sides as mesh points in our quadrature, the use of the algorithm by Björck and Pereyra [3] to obtain the inverse of the matrix seems well suited for cases where α is a unit fraction: Their algorithm is fast, requiring only $\mathcal{O}(n^2)$ arithmetic operations to solve a linear equation system with n variables. More importantly Higham showed in [20] that if the Björck-Pereyra algorithm is used to invert a classical Vandermonde matrix for which the defining elements are positive and monotonically ordered (which is true for our system (13)) the estimate

$$|\hat{V}^{-1} - V^{-1}| \leq 5n\epsilon_{\mathrm{M}} |V^{-1}| + \mathcal{O}(\epsilon_{\mathrm{M}}^2) \tag{16}$$

holds, where ϵ_{M} is the machine precision and $\hat{V}^{-1}$ is the inverse of the Vandermonde matrix V computed by the algorithm. The estimate (16) has to be understood componentwise using the modulus of matrices, defined by $|A| = (|a_{ij}|)$. Even though $\mathcal{O}(n^3)$ arithmetic operations are required for the computation of the inverse using the Björck-Pereyra algorithm, the method seems advantageous since the error bound (16) is independent of the condition number of the Vandermonde matrix V. However, while theoretically the

Björck-Pereyra algorithm seems to be well fitted for our problem, the practical implementation fails because the estimate (16) is dependent on the entries of the exact inverse V^{-1} whose absolute values are getting exceedingly large in our cases due to the structure of the exponential Vandermonde system.

A different approach to tackle the ill-conditioned equation system is the use of a non-stationary iterative solver suited for our problem. The *Generalized Minimum Residual method* (GMRES) by Saad and Schultz [33] seems to be the most promising (for further reading we refer to the book [32] by Saad). Our choice is based on the fact that we are primarily concerned with obtaining an approximate solution to (13) for which the residual is small (see §5). GMRES has the property that the norm of the residual is minimized over the current Krylov subspace at each iteration. In addition, the non-Hermitian nature of the system rules out many of the cheaper alternatives and its denseness means that GMRES will be less expensive to apply than methods, such as Conjugate Gradient Squared (CGS), that require more than one matrix-vector multiplication at each step (see [17, §5.7]). In exact arithmetic GMRES will converge to the exact solution in no more than n iterations, but its convergence behaviour in a finite-precision implementation is currently not well-understood, particularly for ill-conditioned problems, so we cannot predict in advance whether or not the method will provide solutions to (13) with suitably small residuals. A disadvantage of this approach compared with either direct solution by LU decomposition or computation of the inverse matrix is that the iteration has to be repeated for each different right-hand-side, rather than using the ready-computed LU factors or inverse matrix to solve each system. Thus we expect this method to be considerably more expensive in terms of computer time.

We investigated both standard GMRES and the slightly modified GMRES solver by Walker [36] where the Householder transformation is used for the computation of the Krylov space instead of the modified Gram-Schmidt orthonormalization. The justification of this concept lies in the fact that the modified Gram-Schmidt orthonormalization can fail to perform well if the vectors of the Krylov space are not sufficiently independent (as they are especially in cases where the choice of α results in two almost identical columns). Indeed, if $Q = \{q_0, q_1, \ldots, q_{k-1}\}$ denotes the orthonormalized basis of the Krylov space S computed by the modified Gram-Schmidt method with floating point arithmetic of precision ϵ_{M}, then the following estimate holds (see Björck [2]):

$$Q^{\mathrm{T}}Q = I + E, \qquad \|E\|_2 \approx \epsilon_{\mathrm{M}}\kappa_2(S), \tag{17}$$

where $\kappa_2(S)$ denotes the (2-norm) condition number of the matrix S. However using the Householder transformations yields under the same notation as in (17) the following estimate (see Björck [2]):

$$Q^{\mathrm{T}}Q = I + E, \qquad \|E\|_2 \approx \epsilon_{\mathrm{M}},$$

which is independent of the condition number $\kappa_2(S)$ of the original basis of the Krylov space and thus it may give better results for our system.

We did an experiment for the calculation of starting weights using the four methods described above. All calculations were done in Matlab Version 6.5 in double precision. First

we used a simple linear equation system solver (denoted by "lu" in the tables below) for the computation of the starting weights. The Matlab backslash operator "\" was used, which applies an LU decomposition on the coefficient matrix and then solves the corresponding systems. Secondly a Matlab implementation of the the Björck-Pereyra algorithm ("bp") for inverting the Vandermonde matrix was tested for the cases where α was a unit fraction. Last we did two tests using the GMRES algorithm: (a) we used the Matlab *gmres* function ("gmres"), which uses the Gram-Schmidt orthonormalization; and (b) we implemented the method of [36, Algorithm 2.2] ("gmresh") in Matlab to check the GMRES algorithm with Householder transformation. In both cases we used full (i.e. not re-started) GMRES without preconditioning and with a stopping tolerance of 1e−16 (which was never, in practice, achieved).

The following tables give the results of these experiments. The average residuals of the first 1000 starting weights for the different methods are given for various choices of α. The best value for each choice of α is marked in bold.

α	$\frac{1}{10}$	$\frac{1}{5}$	$\frac{1}{4}$	$\frac{1}{3}$	$\frac{1}{2}$
lu	1.02e-04	5.61e-06	2.59e-07	3.66e-11	2.31e-14
bp	1.34e+33	9.79e+02	5.00e-03	9.59e-08	4.95e-12
gmres	1.01e-05	**2.57e-06**	4.39e-07	3.75e-11	**1.56e-14**
gmresh	**3.85e-06**	2.63e-06	**1.33e-07**	**2.26e-11**	1.27e-13

Table 1
Average residuals for various numerical methods for the exponential Vandermonde system (13) with α being a unit fraction.

α	0.49	0.51	$\frac{2}{3}$	$\frac{4}{5}$	$\frac{9}{10}$
lu	1.12e-05	1.38e-08	9.87e-13	1.43e-11	**5.98e-12**
gmres	**1.75e-06**	**6.92e-09**	**7.82e-13**	**5.04e-12**	8.66e-12
gmresh	1.76e-06	6.93e-09	3.11e-12	2.76e-11	4.01e-11

Table 2
Average residuals for various numerical methods for the exponential Vandermonde system (13) with α not being a unit fraction.

Another important difference between using a standard solver or a GMRES method for the starting weight computation is the actual distribution of the residual over the different starting weights. We present a figure (Figure 1) showing the starting weights as well as their residuals for the case $\alpha = \frac{1}{10}$ and 10000 nodes for the two different system solvers.

On the basis of experiments that we have conducted with a number of different values for α for which the above tables are just an extract, the following conclusions can be drawn:

- The Björck-Pereyra algorithm should not be used for the computation of the starting weights. However, a different algorithm exploiting the special structure of the exponential Vandermonde matrix may give better results in the future.

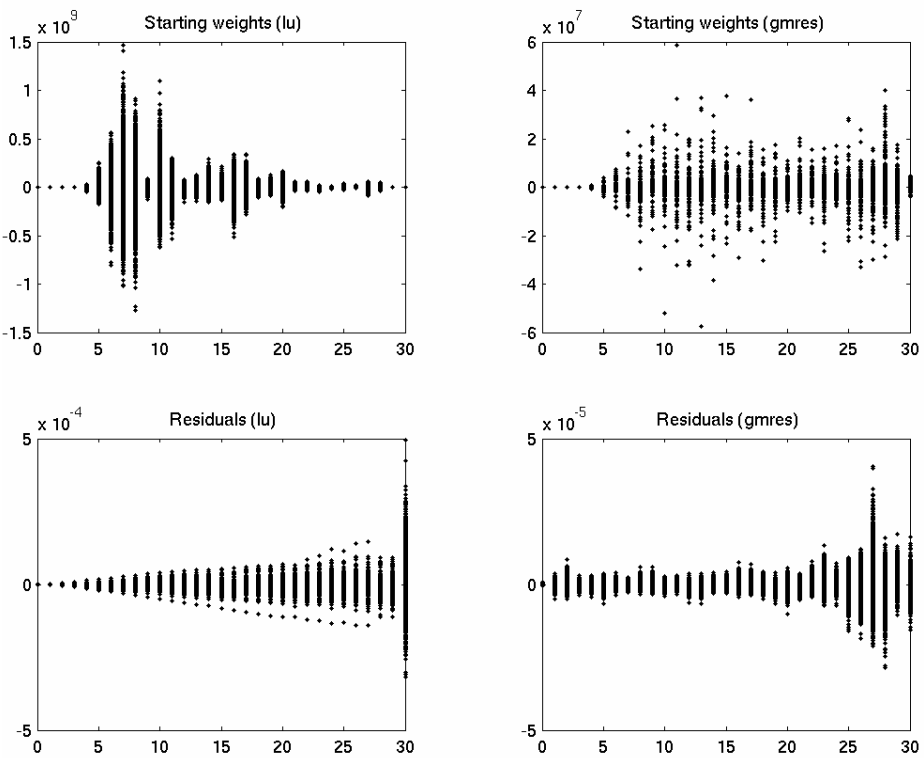

Fig. 1. Starting weights and residuals for $\alpha = \frac{1}{10}$ and $N = 10000$ computed by LU decomposition and GMRES method, respectively. Each dot represents one starting weight or residual. All 31 starting weights for the nodes $N = 1, 2, \ldots, 10000$ are shown.

- The LU decomposition method gives a slightly worse result for the starting weights than either of the GMRES methods. However, the computational time of the LU decomposition is far below that of the GMRES methods. Therefore it has advantages when attempting to implement a fast scheme.
- Both GMRES methods perform equally well. Each one has certain values of α where it is advantageous compared to the other one. However, the Householder transformation needs more computation time than the Gram-Schmidt orthonormalization. In cases where the "best" results are needed and computation time is not the most important factor, the GMRES method should be used.
- For almost all choices of α, *none* of the four methods produces starting weights which are exact to machine precision. Therefore, in general, problems will arise in using any of those weights in the quadrature as we will describe in more detail in Sections 5 and 6.

It is possible that the solution cost and/or the accuracy of the residuals computed using the GMRES iterations could be improved by using a preconditioner. However, our (so far rather limited) experiments using standard preconditioning techniques for dense matrices (e.g. incomplete LU decomposition, diagonal and band approximation, wavelet compression) have been unsuccessful and in some cases have *increased* both the residual norm and the computation time. Difficulties in designing an effective preconditioner for this system are to be expected, since most standard preconditioners are based on approximating the inverse of the system matrix, which we know cannot be done accurately in this case.

Moreover, although theoretical results are not available for GMRES, it is known that, for ill-conditioned systems, preconditioning is ineffective in improving the accuracy of other Krylov subspace methods, such as Conjugate Gradients (see [17]).

4 Review of other existing algorithms

A number of other schemes for the approximate solution of the initial value problem (7) has been proposed in the literature. In this section we shall briefly review those algorithms and identify their strengths and weaknesses. In particular we will see that the performance of most methods does not depend strongly on the precise choice of the order α; small changes in this parameter will usually give rise to insignificant changes in the behaviour of the algorithm. This observation is in striking contrast to what we will see below for the multistep methods.

As a first algorithm we mention the Adams-Bashforth-Moulton method introduced in [11,12] and investigated in a more detailed way in [10,9]. The method is a general purpose algorithm that is capable of handling any sort of function f on the right-hand side of eq. (7). As described in [9] it typically exhibits $\mathcal{O}(h^{1+\alpha})$ convergence (as above, h denotes the step size of the algorithm under consideration). Therefore this algorithm cannot be considered to be particularly fast (especially if α is close to 0), but it has its advantages in being very simple to implement (both for linear and for nonlinear equations) and reliable.

An alternative is the backward differentiation formula of [5]. Notice that this method is based on the idea of discretizing the differential operator in the given equation (7) by a certain finite difference. If we apply Lubich's approach described above to a generate the fractional version of a classical BDF (for first-order equations), then this amounts to using a different discretization of the differential operator, and so the two approaches are in effect not equivalent. The approach of [5] has been investigated very thoroughly. In particular, the main result of [5] was that under suitable assumptions we can expect an $\mathcal{O}(h^{2-\alpha})$ convergence behaviour (at least for linear problems, but the extension to the nonlinear case can be done along the usual lines). Thus we do not have very fast convergence here either, but now the most difficult case is if α is close to 1. In [13] we have seen how to improve the performance of the method by an application of extrapolation principles. The method itself has a simple structure (fully described in [5]) but it is implicit; therefore its application to nonlinear problems requires the use of an algorithm for the solution of nonlinear equations (such as, e.g., Newton's algorithm combined with a suitable technique to determine starting values for the iterative process).

5 Implementation of fractional multistep methods and examples

We now turn to the question of the effective practical implementation of the methods described in §3. It has long been recognised that the key problem in their implementation

is the calculation of the starting weights by solving the Vandermonde system (13) (see for example, [1,18,26]) since the Vandermonde matrix is notoriously ill-conditioned. However, the authors of previous works highlight that it is not the accurate calculation of the weights that is important, but rather the value of the residuals

$$h^\alpha \sum_{j=0}^{n} \omega_{n-j}(jh)^\gamma + h^\alpha \sum_{j=0}^{n} w_{n,j}(jh)^\gamma - \frac{1}{\Gamma(\alpha)} \int_0^t (t-s)^{\alpha-1} s^\gamma \, \mathrm{d}s, \quad \gamma \in \mathcal{A}, \tag{18}$$

which correspond to the errors in calculating the values of the integrals of the functions given in (11). To make this point clear, the Vandermonde system is ill-conditioned because it is nearly a matrix of deficient rank. Now any errors in the solution (the starting weights) that correspond to vectors in the kernel of the nearby matrix of deficient rank will lead to very small errors in the calculation of the integral. Therefore, it is argued, errors in the starting weights can be tolerated if the values of the residuals are small (i.e. to machine precision) (see [18,26]). In fact the authors of the earlier works indicate that the residuals may *reasonably be assumed always to be small.*

5.1 Two examples

We can see how this works quite effectively by means of the example problem

$$\begin{aligned} x^{(\alpha)}(t) = & \frac{40320}{\Gamma(9-\alpha)} t^{8-\alpha} - 3\frac{\Gamma(5+\alpha/2)}{\Gamma(5-\alpha/2)} t^{4-\alpha/2} + \frac{9}{4}\Gamma(\alpha+1) \\ & + \left(\frac{3}{2}t^{\alpha/2} - t^4\right)^3 - x(t)^{3/2} \end{aligned} \tag{19}$$

with initial condition $x(0) = 0$. We chose this equation as our test problem for this paper because it is a nonlinear equation that nevertheless has a known exact analytical solution of the form

$$x(t) = t^8 - 3t^{4+\alpha/2} + \frac{9}{4}t^\alpha \tag{20}$$

for every $\alpha \in (0, 1)$.

In the first special case we solved (19) with $\alpha = \frac{1}{2}$. This is the type of problem dealt with in the earlier literature and so we would hope that the fractional linear multistep method would be effective. We used a 4th order BDF method as the basis for the fractional multistep method. We used Matlab Version 6.5 in double precision for the calculations and compared the approximate and exact solutions over various numbers (N) of grid points on the interval $[0, 1]$. The starting values were obtained by iteration and can be assumed to contain small errors. We tabulated the absolute errors at $t = 1$ in each case and estimated the order of convergence of the method. For reference purposes later, an upper bound on the residuals (18) in this case was 2.3e−14 and so the residuals can be regarded as being to machine precision. The numerical results are given in the left part of Table 3. We can see immediately that the behaviour of the numerical method is exactly

what we would want. The estimated order of convergence is as close to 4 as could be expected and the method appears to perform well.

N	error	convergence order
40	4.1127e-05	
80	2.6325e-06	3.97
160	1.6624e-07	3.99
320	1.0435e-08	3.99
640	6.5334e-10	4.00

N	error	convergence order
80	2.5000e-01	
160	7.4594e-02	1.74
320	4.6450e-02	0.68
640	9.9320e-03	2.23
1280	1.9078e-04	5.70
2560	3.8214e-04	−1.00

Table 3
Errors at $t = 1$ for example problem (19) with $\alpha = 1/2$ (left) and $\alpha = 1/10$ (right).

In the second example we repeated the calculations, this time for $\alpha = \frac{1}{10}$. Now the residuals are roughly 1.1e−04 which is *not to machine precision*. Of course the earlier authors had no reason to check the situation for $\alpha = \frac{1}{10}$ since it was not at that time considered important for calculations.

We can see immediately from the numerical data presented in the right part of Table 3 that the method has lost its order 4. In fact this is hardly surprising because the errors in the integrals (represented by the residual values) are large compared to the overall error of the method.

This example gives us the first indication that the residuals in (18) cannot in general be relied upon to be small when we use values of α other than $\alpha = \frac{1}{2}$.

5.2 Investigating the magnitude of the starting weights

In the earlier papers the condition $|w_{nj}| = \mathcal{O}(n^{\alpha-1})$ is given as an important condition on the starting weights for stability of the numerical scheme. Baker and Derakhshan ([1], for example) point out that the starting weights will satisfy this condition for a range of numerical schemes, including the BDF methods that we have been using. However they assume that the Vandermonde system has been solved exactly. Therefore it is reasonable for us to consider the values of $|w_{nj}|$ as n varies as one way of testing the likely performance of our numerical schemes.

We present first a figure (Figure 2) showing how the calculated starting weights vary for up to 60 grid points and $\alpha = \frac{1}{2}$. This illustrates the phenomenon that we would hope to see and reflects the good performance of the BDF method in this case.

More surprising is the next figure (Figure 3). Here we present the starting weights for $\alpha = \frac{1}{2}$ but for much larger numbers of grid points. We draw attention to the way in which suddenly the method that is known to perform really well for small numbers of grid points

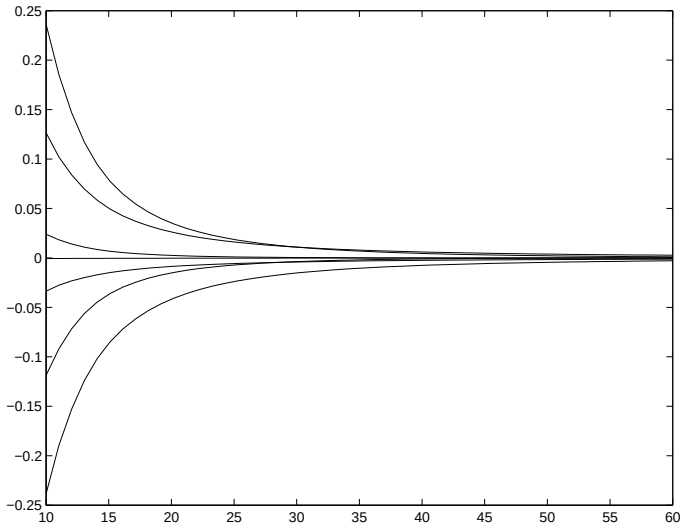

Fig. 2. Starting weights for $\alpha = \frac{1}{2}$ and $n = 10, 11, \ldots, 60$. Each line represents one column of starting weights.

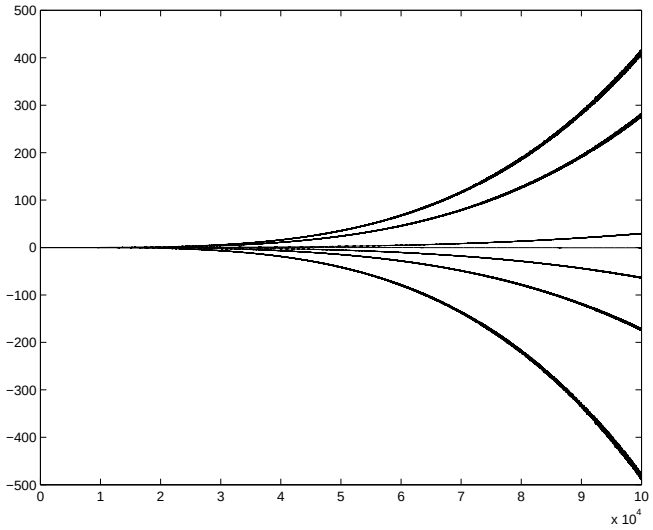

Fig. 3. Starting weights for $\alpha = \frac{1}{2}$ and $n = 1, 2, \ldots, 100000$. Each line represents one column of starting weights.

exhibits behaviour that would suggest a poor approximate solution for a larger number of grid points. We shall see below that this is indeed what will happen.

Now we present a figure similar to Figure 2 for $\alpha = \frac{1}{10}$ (Figure 4). This time we know that the scheme performs badly and this is again reflected in the figure, which shows the behaviour of a single starting weight. While the magnitude of the starting weight decreases in the beginning as one would expect from the theory, the explosion we have seen for the case $\alpha = \frac{1}{2}$ occurs for $\alpha = \frac{1}{10}$ much sooner. In addition the behaviour itself becomes chaotic. Similar behaviour is observed for all 31 starting weights.

Finally we want to draw attention to the fact that there is some interaction between

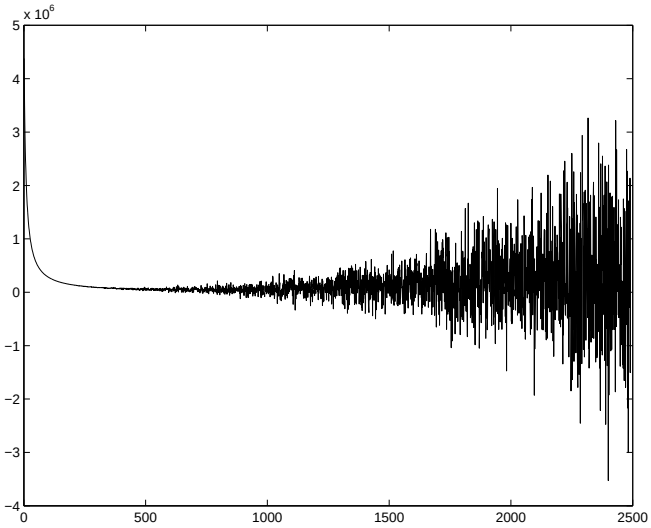

Fig. 4. Starting weight for $\alpha = \frac{1}{10}$ and $n = 10, 11, \ldots, 2500$. Only the graph of the 15th starting weight is drawn. The behaviour is typical of all 31 starting weights.

errors in the starting values and the errors in the starting weights which may be worthy of further investigation. We solved equation (19) with $\alpha = \frac{1}{10}$ first using starting values obtained in the usual iterative way and then we compared our solution with one calculated using *exact* starting values. The results (shown in Figure 5) indicate what we have found in several examples we tried, namely that a solution based on exact starting values may not display the same tendency to errors because of incorrect starting weights as would one based on inexact starting values. Furthermore the error at $t = 1$ produced by using exact starting values is 4.52e−7. This is about as good as the error produced by using the second order method with the same number of mesh points which is 6.05e−7 and thus the second order method would have been the more reasonable choice for this problem.

Another important drawback in the case where the starting values were obtained in the usual iterative way is that the algorithm stopped after 9850 steps since it returned a negative value at this step and thus the evaluation of the right-hand side in the next step would produce an imaginary number. The computation of the starting weights were done using the Matlab backslash operator "\". The GMRES method produced similar behaviour when we increased the number of mesh points. For $N = 10240$ however, the algorithm finished and produced an error of 2.29e−6 at $t = 1$ using starting weights computed by the Matlab "gmres" function.

A similar effect has been observed even in the much more well-behaved case $\alpha = \frac{1}{2}$ (see Fig. 6): Here we have used the starting values obtained by perturbing the exact data by a small amount (1e−4). The number of grid points was 100000. It turns out that in this case the numerical solution becomes negative at $t \approx 0.8$, and so the algorithm breaks down at this point.

We return to these drawbacks in our theoretical discussions of the next section.

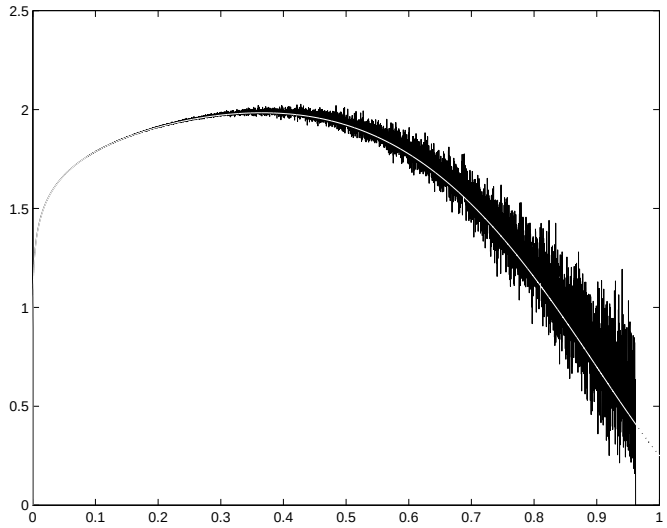

Fig. 5. Plot of the numerical solution (white line, dotted line) of example (19) with exact starting values against the numerical solution (black funnel) with computed starting values and $\alpha = \frac{1}{10}$, $p = 4$ and $N = 10240$.

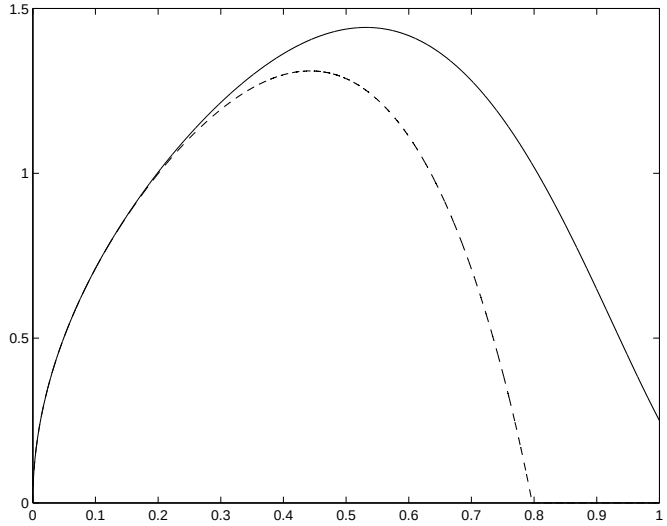

Fig. 6. Plot of the true solution (solid line) of example (19) against the numerical solution (dashed line) with parameters $\alpha = \frac{1}{2}$, $p = 4$ and $N = 100000$ and small (10^{-4}) perturbations in starting values.

6 Analysis of errors arising from starting weights and starting values

In this section we continue our discussion of errors in solutions based on errors in the starting weights and starting values. This time we approach the problem from a more theoretical viewpoint.

For the sake of simplicity we shall concentrate on the approximation of the convolution integral (9). This is in essence equivalent to solving a *linear fractional differential equation.*

As we discussed earlier, in the case of *nonlinear* fractional differential equations we need to employ, in addition, a nonlinear solver at each step. We do not concern ourselves with these details here.

We assume a fractional linear multistep method with s starting values. We propose to solve the equation over the interval $[0, T]$ where $T = Nh$ for some fixed $h > 0$.

The basic idea is as follows: we assume that the *exact* starting weights w_{nj} and convolution weights ω_{n-j} would be recorded in an $(N+1) \times (N+1)$ matrix A according to

$$A = \left(\begin{array}{c|c} I_s & 0 \\ \hline A_{21} & A_{22} \end{array} \right) \tag{21}$$

where I_s is an $s \times s$ identity matrix, A_{21} is the $(N+1-s) \times s$ matrix with $(A_{21})_{i,j} = w_{s+i,j} + \omega_{s+i-j}$ and A_{22} is the square $(N+1-s)$ matrix with $(A_{22})_{i,j} = \omega_{i-j}$ for $j \leq i$ and 0 otherwise. In practice we have perturbed weights leading to a matrix of the form $A + B$ where B takes the special form

$$B = \left(\begin{array}{c|c} 0 & 0 \\ \hline B_{21} & 0 \end{array} \right) \tag{22}$$

where B_{21} is the $(N+1-s) \times s$ matrix containing the errors in the starting weights $(B_{21})_{i,j} = (w_{s+i,j} - \tilde{w}_{s+i,j})$. This highlights the fact that only the starting weights contain errors.

The *exact* starting values are assumed to be stored as the first s elements in a solution vector $x \in \mathbb{R}^{N+1}$ and the errors in the starting values are assumed to be stored in the first s elements of the vector $e \in \mathbb{R}^{N+1}$.

Now we are in a position to formulate our calculations in terms of the matrices A, B and the vectors x, e: the approximate solution described in [25] is given by successive multiplication of vector x by the matrix $h^\alpha A$. Each successive multiplication by the matrix $h^\alpha A$ corresponds to evaluation of the next step in the convolution integral (starting from step s). In total we need to pre-multiply by $h^\alpha A$ a total of $N - s + 1$ times to complete the solution over $[0, T]$. Thus we wish to calculate

$$J = (h^\alpha A)^{N+1-s} x. \tag{23}$$

In fact, when we take the inevitable errors into account, we will actually evaluate

$$\tilde{J} = (h^\alpha (A + B))^{N+1-s} (x + e) \tag{24}$$

so the errors introduced by the starting values and starting weights we calculated are given by the expression $\tilde{J} - J$.

Lubich [25,26] defined the starting weights in A in such a way that the method integrates *exactly* a set of s functions (see also §3 above). Each of these functions can be sampled at

the values $0, h, 2h, 3h, \ldots, (s-1)h$ to give a vector in $\mathbb{R}^s$. It is simple to see that the set of s vectors of dimension s defined in this way spans $\mathbb{R}^s$. We extend each of these s-dimensional vectors to an $(N+1)$-dimensional vector by concatenating $N+1-s$ zeros in the last components to give us s linearly independent vectors that we shall call $v_1, v_2, \ldots v_s$.

By construction of the vectors v_i, we can see that constants α_j, β_j can be found so that $x = \beta_1 v_1 + \ldots + \beta_s v_s$ and $e = \gamma_1 v_1 + \ldots \gamma_s v_s$.

This shows (by linearity) that successive multiplication by the matrix $h^\alpha A$ evaluates exactly both the propagation of the values in x (which we want) and the propagation of the values in e (which we do not want).

Now we can turn our attention to the effect of multiplication by the matrix $h^\alpha B$. As we constructed B it consists of the errors in the starting weights which we evaluated in accordance with the methods of §3.2. In their paper [18], the authors say that the *residuals* in the calculation of

$$h^\alpha \sum_{j=0}^{n} \omega_{n-j}(jh)^\gamma + h^\alpha \sum_{j=0}^{s} w_{nj}(jh)^\gamma - \frac{1}{\Gamma(\alpha)} \int_0^t (t-s)^{\alpha-1} s^\gamma \, \mathrm{d}s, \quad \gamma \in \mathcal{A} \tag{25}$$

need to be small. They assert (see also [26]) that this can happen even when the errors in the starting weights themselves are not very small.

Now we can see that the residuals to which they refer are the same as the values obtained by multiplying rows of B by vectors v_j. Therefore the accuracy of the approximation of $A^{N+1-s}x$ by $(A+B)^{N+1-s}(x+e)$ hinges on the values of $B^\ell v_j$ for each vector v_j.

We recall that Baker and Derakhshan [1] have shown that, for the numerical methods of interest to us, the weights w_{nj} satisfy

$$\sup_{0 \le j \le n} |w_{nj}| = \mathcal{O}(n^{\alpha-1}) \text{ as } n \to \infty. \tag{26}$$

It follows that $\|A\|_1 = \mathcal{O}(N^\alpha)$ and $\|h^\alpha A\|_1 = \mathcal{O}(1)$. Thus we see that the calculation of the solution using the *exact* starting weights is *stable with respect to small errors in the starting values.*

We can readily obtain an estimate for the worst case behaviour by evaluating (estimating) $\|B\|_1$. We know that $\|A+B\| + \|A\| \ge \|B\| \ge \|A+B\| - \|A\|$ (for any norm) and that $\|A\|_1 = \mathcal{O}(N^\alpha)$. We have the matrix $A+B$ and so we can evaluate $\|A+B\|_1$ exactly. If $\|B\|_1$ is large then we know that certain combinations of v_j will be magnified by that factor. It is clear that $\|B\|_1$ will be small if and only if all the residuals are small.

The above discussion shows us that if the value $\|B\|_1$ is not small then the values $h^{\alpha\ell}B^\ell x$ and $h^{\alpha\ell}B^\ell e$ may become large. We wish to know whether they will in fact do this. For insight we turn to the *power method* for calculating eigenvalues of a square matrix based on repeatedly multiplying a starting vector by the given matrix. For the power method we see that if the starting vector is chosen randomly, there is a probability unity that the

dominant eigenvalue will be found. However if the starting vector is chosen so that there is no component in the direction of the eigenvector with dominant eigenvalue then some less prominent eigenvalue will be found. The situation in our problem is exactly parallel with this. If the vector x (the starting vector) is chosen so that there is no component in the direction along which the matrix B exhibits its dominant behaviour, then the error produced by the dominant behaviour will not be visible in the solution. On the other hand, the starting errors e are likely to be random and therefore with probability unity will show up the dominant behaviour of the matrix B. In general, in the examples we have been working with, $h^{\alpha}(A+B)$ leads to an unstable solution operator with respect to small changes in the starting values. As the reader will observe, we have been unable to implement the stable BDFp method (at least for $4 \leq p \leq 6$) developed theoretically by Lubich, but have been forced instead to implement an unstable approximation to it.

In the examples of §5.1 we saw a case where putting in the starting vector with exact initial values of the solution led to a good accurate solution, while putting in random starting errors destroyed accuracy (recall Figure 6). We can see how this can happen when we look at a section of the matrix of residuals. Almost all the residuals are quite small and therefore it is comparatively easy to find starting values that do not pick up the dominant (bad) behaviour. The random starting errors introduce all the dynamics of the solution.

7 Conclusions

We discussed at the start of the paper what we required of a good numerical scheme for fractional differential equations. The discussions of the two previous sections illustrate the pitfalls that can arise when we implement fractional multistep methods in practice, even though their good behaviour has been proved in theory. We have concentrated on BDF methods here as the basis for our investigations following the advice of Baker and Derakhshan [1] who indicated that they had not found any benefit in attempting to use other possible linear multistep formulas. It is known that BDFs of order up to 6 are $A(\phi)$-stable for some $\phi > 0$ and so these methods are the basis for our calculations here.

In the following diagrams we consider the computational cost of calculating the solution to equation (19) in terms of the time taken for the total calculation, compared with the error of the solution obtained. We present graphs for each of the methods BDF 1-6. In all cases we recorded the respective error at the point $t = 1$. The lines always display the correlation between the computation time (horizontal axis) and the numerical error (vertical axis; note the logarithmic scale). As can be expected the results of our previous discussion show up in some untypical shapes in the graphs.

The left part of Figure 7 shows that for $\alpha = \frac{1}{2}$ and a reasonably small number of nodes, BDF 1-4 methods are effective but BDF 5-6 show problems. The right part illustrates how even BDF 4 begins to lose accuracy as the number of nodes used increases.

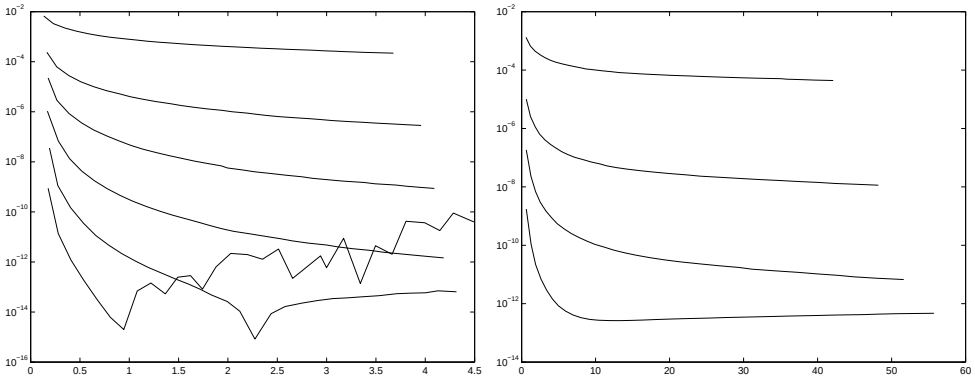

Fig. 7. Left: Error vs. time: Computation time (in seconds) and the numerical error for Lubich's BDF method of order 1 (top) to 6 (bottom), applied to test equation (19) with $\alpha = \frac{1}{2}$ and $N = 500, 600, \ldots, 3000$. Right: Error vs. time: Same condition as the left figure but $N = 2500, 5000, \ldots, 50000$ for methods of order 1 to 4.

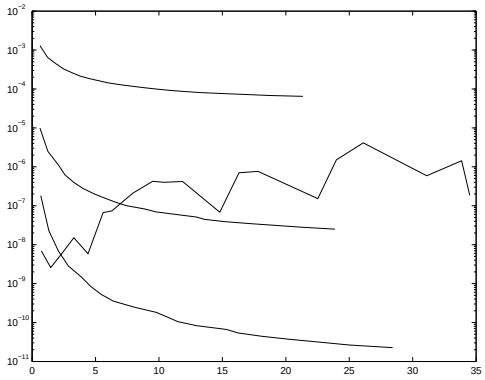

Fig. 8. Error vs. time: computation time and the numerical error for Lubich's BDF method of order 1 (top) to 4 (bottom), applied to test equation (19) with $\alpha = 0.49$ and $N = 500, 1000, 1500, \ldots, 10000$.

The corresponding behaviour for $\alpha = 0.49$ is shown in Figure 8. We draw attention to the fact that now that $\alpha \neq \frac{1}{2}$ the loss of good behaviour arises for much smaller numbers of nodes for BDF 4. BDF 5 and 6 are not included in the diagram since their results are even worse.

References

[1] C. T. H. Baker and M. S. Derakhshan, *Stability barriers to the construction of $\{\rho, \sigma\}$-reducible and fractional quadrature rules*, in Numerical Integration III, H. Braß and G. Hämmerlin, eds., no. 85 in Internat. Ser. Numer. Math., Basel, 1988, Birkhäuser, pp. 1–15.

[2] Å. Björck, *Solving linear least squares problems by Gram-Schmidt orthogonalization*, Nordisk Tidskr. Informations-Behandling, 7 (1967), pp. 1–21.

[3] Å. Björck and V. Pereyra, *Solution of Vandermonde systems of equations*, Math. Comp., 24 (1970), pp. 893–903.

[4] M. Caputo, *Linear models of dissipation whose Q is almost frequency independent, II*, Geophys. J. Royal Astronom. Soc., 13 (1967), pp. 529–539.

[5] K. Diethelm, *An algorithm for the numerical solution of differential equations of fractional order*, Elec. Transact. Numer. Anal., 5 (1997), pp. 1–6.

[6] K. Diethelm and N. J. Ford, *Analysis of fractional differential equations*, J. Math. Anal. Appl., 265 (2002), pp. 229–248.

[7] K. Diethelm and N. J. Ford, *Numerical solution of the Bagley-Torvik equation*, BIT, 42 (2002), pp. 490–507.

[8] ——, *Multi-order fractional differential equations and their numerical solution*, Appl. Math. Comput., 154 (2004), pp. 621–640.

[9] K. Diethelm, N. J. Ford, and A. D. Freed, *Detailed error analysis for a fractional Adams method*, Numerical Algorithms, 36 (2004), pp. 31–52.

[10] ——, *A predictor-corrector approach for the numerical solution of fractional differential equations*, Nonlinear Dynamics, 29 (2002), pp. 3–22.

[11] K. Diethelm and A. D. Freed, *The FracPECE subroutine for the numerical solution of differential equations of fractional order*, in Forschung und wissenschaftliches Rechnen 1998, S. Heinzel and T. Plesser, eds., no. 52 in GWDG-Berichte, Göttingen, 1999, Gesellschaft für wissenschaftliche Datenverarbeitung, pp. 57–71.

[12] ——, *On the solution of nonlinear fractional differential equations used in the modeling of viscoplasticity*, in Scientific Computing in Chemical Engineering II — Computational Fluid Dynamics, Reaction Engineering, and Molecular Properties, F. Keil, W. Mackens, H. Voß and J. Werther, eds., Heidelberg, 1999, Springer, pp. 217–224.

[13] K. Diethelm and G. Walz, *Numerical solution of fractional order differential equations by extrapolation*, Numer. Algorithms, 16 (1997), pp. 231–253.

[14] N. J. Ford and A. C. Simpson, *The numerical solution of fractional differential equations: Speed versus accuracy*, Numer. Algorithms, 26 (2001), pp. 333–346.

[15] A. D. Freed, K. Diethelm, and Y. Luchko, *Fractional-order viscoelasticity (FOV): Constitutive developments using the fractional calculus: First annual report*, Technical Memorandum, TM–2002-211914, NASA Glenn Research Center, Cleveland, 2002.

[16] R. Gorenflo, *Afterthoughts on interpretation of fractional derivatives and integrals*, in Transform Methods and Special Functions, Varna 96, P. Rusev, I. Dimovski, and V. Kiryakova, eds., Sofia, 1998, Bulgarian Academy of Sciences, Institute of Mathematics and Informatics, pp. 589–591.

[17] A. Greenbaum, *Iterative Methods for Solving Linear Systems*, SIAM, Philadelphia, 1997.

[18] E. Hairer, C. Lubich, and M. Schlichte, *Fast numerical solution of weakly singular Volterra equations*, J. Comput. Appl. Math., 23 (1988), pp. 87–98.

[19] E. Hairer, S. P. Nørsett, and G. Wanner, *Solving Ordinary Differential Equations I: Nonstiff Problems*, Springer, Berlin, 2nd revised ed., 1993.

[20] N. J. Higham, *Error analysis of the Björck-Pereyra algorithms for solving Vandermonde systems*, Numer. Math., 50 (1987), pp. 613–632.

[21] G. Joulin, *Point-source initiation of lean spherical flames of light reactants: An asymptotic theory*, Combustion Science and Technology, 43 (1985), pp. 99–113.

[22] J. D. Lambert, *Computational methods in ordinary differential equations*, John Wiley & Sons, London, 1973.

[23] C. Lederman, J.-M. Roquejoffre, and N. Wolanski, *Mathematical justification of a nonlinear integro-differential equation for the propagation of spherical flames*, C. R., Math., Acad. Sci. Paris, 334 (2002), pp. 569–574.

[24] N. Levinson, *A nonlinear Volterra equation arising in the theory of superfluidity*, J. Math. Anal. Appl., 1 (1960), pp. 1–11.

[25] C. Lubich, *Fractional linear multistep methods for Abel-Volterra integral equations of the second kind*, Math. Comp., 45 (1985), pp. 463–469.

[26] ——, *Discretized fractional calculus*, SIAM J. Math. Anal., 17 (1986), pp. 704–719.

[27] F. Mainardi, *Fractional calculus: Some basic problems in continuum and statistical mechanics*, in Fractals and Fractional Calculus in Continuum Mechanics, A. Carpinteri and F. Mainardi, eds., Wien, 1997, Springer, pp. 291–348.

[28] K. B. Oldham and J. Spanier, *The Fractional Calculus*, Academic Press, New York, 1974.

[29] I. Podlubny, *Fractional Differential Equations*, Academic Press, San Diego, 1999.

[30] L. B. Rall, *Automatic Differentiation: Techniques and Applications*, Springer, Berlin, 1981.

[31] J. W. Robbin and D. A. Salamon, *The exponential Vandermonde matrix*, Linear Algebra Appl., 317 (2000), pp. 225–226.

[32] Y. Saad, *Iterative Methods for Sparse Linear Systems*, SIAM, Philadelphia, 2nd ed., 2003

[33] Y. Saad and M. H. Schultz, *GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 856–869.

[34] S. G. Samko and A. A. Kilbas and O. I. Marichev, *Fractional Integrals and Derivatives: Theory and Applications*, Gordon and Breach, Yverdon, 1993.

[35] P. J. Torvik and R. L. Bagley, *On the appearance of the fractional derivative in the behavior of real materials*, J. Appl. Mech., 51 (1984), pp. 294–298.

[36] H. F. Walker, *Implementation of the GMRES method using Householder transformations*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 152–163.