Numerical Solution of Linear Volterra Integral Equations of the Second Kind with Sharp Gradients

Samuel A. Isaacson[a,1], Robert M. Kirby[b]

[a] *Department of Mathematics and Statistics, Boston University, 111 Cummington St., Boston, MA 02215 USA*
[b] *School of Computing, University of Utah, 72 S. Central Campus Drive, 3750 Warnock Engineering Building, Salt Lake City, UT 84112 USA*

Abstract

Collocation methods are a well developed approach for the numerical solution of smooth and weakly-singular Volterra integral equations. In this paper we extend these methods, through the use of partitioned quadrature based on the qualocation framework, to allow the efficient numerical solution of linear, scalar, Volterra integral equations of the second kind with smooth kernels containing sharp gradients. In this case the standard collocation methods may lose computational efficiency despite the smoothness of the kernel. We illustrate how the qualocation framework can allow one to focus computational effort where necessary through improved quadrature approximations, while keeping the solution approximation fixed. The computational performance improvement introduced by our new method is examined through several test examples. The final example we consider is the original problem that motivated this work: the problem of calculating the probability density associated with a continuous time random-walk in three-dimensions that may be killed at a fixed lattice site. To demonstrate how separating the solution approximation from quadrature approximation may improve computational performance, we also compare our new method to several existing Gregory, Sinc, and global spectral methods where quadrature approximation and solution approximation are coupled.

Keywords: Linear Volterra Integral Equation, collocation, partitioned quadrature, qualocation

1. Introduction

Let $I = [0, T]$, with $T < \infty$. In this paper we consider the numerical solution of the linear Volterra integral equation of the second kind for $y(t)$,

$$y(t) + \int_0^t K(t,s)\, y(s)\, ds = g(t), \quad t \in [0, T]\,. \tag{1}$$

Let $D = \{(t,s)|0 \leq s \leq t \leq T\}$. We restrict our attention to problems where $K(t,s) \in C^\infty(D)$ and $g(t) \in C^\infty(I)$. With these assumptions, the solution $y(t)$ to (1) exists, is unique, and $y(t) \in C^\infty(I)$ [8, Theorems 2.1.2 and 2.1.3].

The numerical solution of (1) is a well-studied problem, and a large variety of numerical methods have been developed to rapidly and accurately obtain approximations to $y(t)$. Overviews and references to the literature for many existing methods are available in [3, 8, 12]. Collocation methods [8, 5, 6, 33], Sinc methods [25], global spectral methods [34], methods for convolution equations [22], Newton-Gregory methods [12], Runge-Kutta methods [27, 28], qualocation methods [14, 11, 30, 32, 31], and Galerkin methods [2, 4, 9, 29] are several of the many approaches that have previously been considered.

*Corresponding Author
Email addresses: isaacson@math.bu.edu (Samuel A. Isaacson), kirby@cs.utah.edu (Robert M. Kirby)

One type of stochastic-reaction diffusion model that is used to model cellular processes, such as gene expression [20], involves the continuous time random walk of molecules on a lattice. Molecules occupying the same lattice site may then react with fixed probabilities per unit time. We recently studied the behavior of one such model, involving a molecule that can undergo a continuous time random walk with binding possible at one specific binding site, as the lattice spacing is varied [17, 18]. This model could be reformulated as a linear, convolution Volterra equation where the kernel, $K(t,s) = K(t-s)$, was smooth but contained sharp gradients [17]. In particular, the kernel could be considered a regularization of the non-integrable (in the Lebesgue sense) kernel

$$K(t,s) = \frac{1}{(t-s)^{3/2}}. \tag{2}$$

The large gradients that appear as the regularization approaches the non-integrable kernel (2) led to poor performance of several existing methods developed for smooth kernels.

To overcome this difficultly we developed an extension to the standard collocation method [8]. In the standard formulation the numerical approximation of the solution to (1) is coupled with the choice of quadrature rule used in evaluating the integral. We develop a qualocation method [14, 11, 30, 32, 31] by using partitioned quadrature to decouple the solution and quadrature approximations, allowing for more accurate evaluation of the integral in (1) when necessary. We show in Section 4.2 that for smooth kernels with sharp gradients partitioned quadrature can significantly increase the computational performance of the collocation method.

As commonly applied, the abstract qualocation method [14, 11, 30, 32, 31] begins with the Galerkin formulation of (1) and uses quadrature to approximate the resulting inner products within the weak formulation. Our qualocation method starts from the standard collocation approximation of (1) and then directly applies partitioned quadrature to the integral operator. Note, the standard collocation method can be obtained as a special case of the qualocation approximation of the Galerkin formulation of (1) (see, for example, [30]). As such, our method may be interpreted as applying the qualocation method twice; once to obtain the collocation equations, and then a second time to approximate the integrals within them.

To further illustrate the benefit of separating solution and quadrature approximation we also compare the performance of the new method to several methods where the two are coupled. In particular, the Sinc method of [25], the global Lagrange interpolating polynomial spectral method of [34], the standard sixth-order Gregory method [12], and an FFT-based, optimized sixth-order Gregory method for convolution equations are all examined. We stress that our comparison is meant to illustrate the benefit of separating solution and quadrature approximation, and is not meant to provide a comprehensive comparison of current state of the art methods for solving Volterra integral equations.

In Section 2 we give an overview of the standard collocation approach, and then present our qualocation-based extension of the method to support partitioned quadrature. For the reader unfamiliar with the standard collocation method, we first derive the method from the method of weighted residuals in Sections 2 and 2.1. It should be noted that there are many techniques for obtaining the collocation equations (10), including by applying the qualocation method as previously mentioned [30]. Moreover, the equivalence between Galerkin and collocation methods has been proven [1]. We conclude our introduction to the collocation method by presenting the standard quadrature approach used in approximating the integral operators in (1) within the collocation method in Section 2.1.1 (see, for example, [8]). For this method, $[0,T]$ is divided into a collection of disjoint "elements", and a set of collocation points are chosen on each element. The quadrature points used to evaluate the portion of the integral in (1) within a given element are chosen to coincide with the collocation points for that element. The approximation to the true solution to (1) is then given by the collection of Lagrange interpolating polynomials, each defined with support on one element and abscissas given by the collocation points. In Section 2.1.2 we introduce our qualocation-based generalization of this method, allowing for a complete separation between the choice of quadrature rule used in evaluating the integral in (1) and the piecewise polynomial approximation to the solution, $y(t)$. The notion of partitioned quadrature as used in qualocation [14, 11, 30, 32, 31] is introduced to allow for increased accuracy in the evaluation of the integral term in (1) for smooth kernels with sharp gradients. Note, similar approaches have also been used in the context of spectral element methods for the solution of partial

differential equations [21]. Finally, in Section 2.2 we give a brief summary of the Gregory, Sinc, and global Lagrange interpolating polynomial spectral methods. Our implementations of these three methods follow directly from the presentations of [17, 15] for the Gregory method, of [25] for the Sinc method, and [34] for the spectral method. We refer the interested reader to those references for full descriptions of the algorithms.

Each of the methods was implemented as a MATLAB m-file (available at [19]). The details of our particular implementations are presented in Section 3. We do not claim that our implementations are optimal, but an effort was made to incorporate natural optimizations for each method.

Section 4 presents a number of examples with smooth, well-behaved kernels and with smooth kernels containing sharp gradients. The smooth, well-behaved kernel examples are studied to gain a baseline understanding of the relative performance of the various methods. For each numerical method and example, a search of the method's numerical parameter space is performed to determine the "first" set of parameters where a specified absolute error tolerance is satisfied. This procedure is repeated for a collection of tolerances, and the median running times of each method are then calculated as a function of tolerance. We find that for the smooth, well-behaved examples of Section 4.1 all the methods perform quite well, with our implementations of the standard collocation method and the Gregory methods having the smallest median running times. The examples with smooth kernels containing sharp gradients in Section 4.2 prove more difficult to resolve. These examples require increased resolution in representing the solution, $y(t)$, in specific regions, increased resolution in evaluating the integral within (1), or a combination of both. In each of the global spectral method, the Sinc method, and the Gregory method the resolution of the solution representation and the accuracy of evaluating the integral term in (1) are linked. These methods all depend on one parameter: the number of basis functions in the Sinc and global spectral methods, and the number of time points in the Gregory method. This parameter must be increased to improve either the solution or integral evaluation accuracy. As such, these methods become less efficient when higher accuracy or resolution is needed in only one of the two components. In contrast, the separation of solution and integral approximations in the partitioned-quadrature qualocation method we present in Section 2.1.2 leads to significant computational improvements over the standard collocation method. We also find that the speed of the FFT-based convolution optimization of the Gregory method allows for good performance with very large numbers of time-steps on each example (though not as good as the partitioned-quadrature qualocation-method).

We conclude by examining the Volterra equation containing a smooth kernel with sharp gradients that motivated this work, first studied in [17]. This equation arises in studying stochastic reaction-diffusion models of gene expression and regulation [20]. Only the Gregory method with the FFT-based optimization and the partitioned-quadrature qualocation method with non-uniform element spacings are able to successfully resolve this example. The latter performs particularly well, requiring only a slight increase in the number of quadrature partitions, and no increase in the number of elements or number of collocation points per element as the regularization of the kernel is decreased. Moreover, we find that the method is limited solely by the increased time required to evaluate the kernel as the regularization is decreased. (In contrast, the Gregory method requires a significant increase in the number of time-steps, and spends a larger fraction of its computational time in determining the solution versus evaluating the kernel.)

2. Methods Studied

We begin by formulating the standard collocation method as a special case of the method of weighted residuals. As pointed out in the introduction, there are many ways to derive the collocation method. The material that follows, and in Subsections 2.1 and 2.1.1, is meant as background for the reader unfamiliar with collocation methods and as an introduction to our notation. In Section 2.1.2 we introduce our partitioned-quadrature qualocation-based approach.

The weak form of (1) is given by

$$\left(y(t) + \int_0^t K(t,s)y(s)\,ds, v(t) \right) = \left(g(t), v(t) \right), \quad \forall v(t) \in V, \tag{3}$$

where V is a vector space of functions, and the inner product of two functions is given by

$$(y(t), v(t)) = \int_0^T y(t)v(t)\,dt. \tag{4}$$

For now we leave the space V arbitrary. We denote by $y_h(t)$ our approximation to $y(t)$ and assume $y_h(t)$ may be written as a linear combination of some finite dimensional basis, $\Phi = \{\phi_1(t), \ldots, \phi_N(t)\}$,

$$y_h(t) = \sum_{n=1}^{N} Y_n \phi_n(t). \tag{5}$$

Denote by $R[y_h](t)$ the residual of the original integral equation (1) when $y(t)$ is replaced by $y_h(t)$,

$$R[y_h](t) = y_h(t) + \int_0^t K(t,s)y_h(s)\,ds - g(t).$$

For the actual solution, $y(t)$, $R[y](t) = 0$. In the method of weighted residuals approach we choose a collection of test functions $v_1(t), \ldots, v_N(t)$, and in the space given by their span try to minimize the residual, $R[y_h](t)$, under the inner product (4). That is, we impose $(R[y_h](t), v_j(t)) = 0$ for each $j = 1, \ldots, N$. Rearranging this equation we find

$$\sum_{n=1}^{N} Y_n \left(\phi_n(t) + \int_0^t K(t,s)\phi_n(s)\,ds, v_j(t) \right) = \left(g(t), v_j(t)\right), \quad j = 1 \ldots N. \tag{6}$$

Let $V_h = \text{span}\{v_1(t), \ldots, v_N(t)\}$. By linearity, these equations are equivalent to the approximate weak formulation,

$$\sum_{n=1}^{N} Y_n \left(\phi_n(t) + \int_0^t K(t,s)\phi_n(s)\,ds, v(t) \right) = \left(g(t), v(t)\right), \quad \forall v(t) \in V_h. \tag{7}$$

We have therefore replaced the exact weak formulation (3) of (1) by the approximate weak formulation (7). Specific choices for V_h, Φ, and quadrature approximations to the integral within the inner product then give the standard collocation method and our qualocation method.

2.1. Standard Collocation Method

The standard collocation method we consider is the same as that given in [8]. We look for a $C^0(I)$ piecewise polynomial approximation [16, 21], $y_h(t)$, to the true solution to (1), $y(t)$. Denote by

$$I_h = \{t_m | 0 = t_0 < t_1 < \cdots < t_M = T\}$$

a partition of $[0, T]$, and let

$$h_m = t_{m+1} - t_m, \quad \text{with } h = \max_{m=0,\ldots,(M-1)} h_m.$$

We will subsequently refer to the interval, $[t_m, t_{m+1}]$, as the mth element. For any arbitrary interval, $\tilde{I}$, we denote by $\mathbb{P}^l(\tilde{I})$ the space of polynomials of degree l on $\tilde{I}$. We assume that the restriction, $y_h^{(m)}(t)$, of $y_h(t)$ to the mth element is a polynomial of degree p_m. (*i.e.* $y_h^{(m)}(t) \in \mathbb{P}^{p_m}([t_m, t_{m+1}])$.) To ensure $y_h(t)$ is continuous, we require $y_h^{(m-1)}(t_m) = y_h^{(m)}(t_m)$, $m = 1, \ldots, M-1$.

We will represent $y_h^{(m)}(t)$ as a Lagrange interpolating polynomial through the abscissas,

$$t_m = t_{m,0} < t_{m,1} < \cdots < t_{m,p_m} = t_{m+1}. \tag{8}$$

Let $L_{m,j}(t)$ denote the corresponding jth local Lagrange basis function through those abscissas,

$$L_{m,j}(t) = \prod_{\substack{i=0,\\ i\neq j}}^{p_m} \frac{t - t_{m,i}}{t_{m,j} - t_{m,i}}, \quad t \in [t_m, t_{m+1}],$$

and define

$$Y_{m,j} = y_h^{(m)}(t_{m,j}) = y_h(t_{m,j}).$$

We then have the representation

$$y_h^{(m)}(t) = \sum_{j=0}^{p_m} Y_{m,j} L_{m,j}(t).$$

Denote by $\mathbf{1}_{\tilde{I}}(t)$ the indicator function on the set $\tilde{I}$. In analogy to (5), we may expand $y_h(t)$ in the basis functions $\phi_{m,j}(t)$,

$$\phi_{m,j}(t) = \begin{cases} L_{m,j}(t)\,\mathbf{1}_{[t_0,t_1]}(t), & m = 0, \\ L_{m,j}(t)\,\mathbf{1}_{(t_m,t_{m+1}]}(t), & m = 1,\ldots,M-1, \end{cases}$$

so that

$$y_h(t) = \sum_{m=0}^{M-1} \sum_{j=0}^{p_m} Y_{m,j}\phi_{m,j}(t). \tag{9}$$

Choosing the collocation space, $V_h = \{\delta(t - t_{m,j})\}_{m,j}$, the weak approximations (6) reduce to the collocation equations

$$Y_{m,j} + \sum_{m'=0}^{m} \sum_{j'=0}^{p_{m'}} Y_{m',j'} \int_0^{t_{m,j}} K(t_{m,j}, s)\phi_{m',j'}(s)\,ds = g(t_{m,j}),$$

$$j = 0,\ldots,p_m,\ m = 0,\ldots,M-1. \tag{10}$$

The points, $\{t_{m,j}\}$, at which $y_h(t)$ satisfies (6) in the collocation space are subsequently referred to as the collocation points.

Note that (10) will generate a block lower-triangular system of linear equations. To convert to this form, we define the column vectors, $Y^{(m)} = (Y_{m,0},\ldots,Y_{m,p_m})^{\mathrm{T}}$, $G^{(m)} = (g(t_{m,0}),\ldots,g(t_{m,p_m}))^{\mathrm{T}}$, and the p_m by $p_{m'}$ matrices $B_{m,m'}$, with entries given by

$$(B_{m,m'})_{j,j'} = \int_0^{t_{m,j}} K(t_{m,j}, s)\phi_{m',j'}(s)\,ds. \tag{11}$$

We may then rewrite (10) as

$$Y^{(m)} + \sum_{m'=0}^{m} B_{m,m'} Y^{(m')} = G^{(m)}, \quad m = 0,\ldots,M-1. \tag{12}$$

Alternatively, let Y denote the vector

$$Y = \begin{pmatrix} Y^{(0)} \\ \cdots \\ Y^{(M-1)} \end{pmatrix},$$

define G similarly, and let B denote the block lower-triangular matrix

$$B = \begin{pmatrix} B_{0,0} & 0 & \cdots & \cdots \\ B_{1,0} & B_{1,1} & 0 & \cdots \\ B_{2,0} & B_{2,1} & B_{2,2} & 0 \ \cdots \\ \cdots & \cdots & \ddots & \cdots \\ B_{M-1,0} & B_{M-1,1} & \cdots & B_{M-1,M-1} \end{pmatrix}. \tag{13}$$

Noting that the total number of collocation points, N, is given by

$$N = 1 + \sum_{m=0}^{M-1} p_m,$$

we find the N by N linear system

$$Y + BY = G. \tag{14}$$

One strong benefit of the qualocation approach is the separation between quadrature choices for evaluating the integrals (11) and the choice of piecewise-polynomial interpolant. While in the "standard" collocation approach, presented in the next section, quadrature nodes are chosen to correspond to interpolation abscissas, we shall see that the freedom to decouple quadrature and approximation can allow for significant computational improvements. In particular, in Subsection 2.1.2 we discuss how nonuniform spacings of the elements may be used to improve the approximation of $y(t)$, while non-uniformly spaced composite Gaussian quadrature rules may improve the evaluation of the integrals (11) for smooth kernels with sharp gradients. These ideas can also be applied in the context of *hp* methods for the numerical solution of PDEs, see [21] for more details.

2.1.1. Standard Element Spacings and Quadrature Rules

In this section we shall summarize the typical element spacings, polynomial basis functions, and quadrature choices used in the standard collocation approach. We refer the interested reader to [8] for a more detailed description of the relevant convergence theory in this case.

We begin with the choice of polynomial interpolants and quadrature rules. The basis functions, $\phi_{m,j}(t)$, are chosen to have the same polynomial order (*i.e.* for some integer, p, we take $p_m = p$, $m = 0, \ldots, M-1$). In practice, the integrals (11) can not be evaluated analytically and must therefore be approximated through the use of numerical quadrature. We subsequently refer to the system obtained from (14) by replacing the integrals (11) with quadrature approximations as the discretized collocation equations. The quadrature formulas used in the numerical evaluation of the integrals (11) are chosen to be order preserving. That is, the quadrature formulas are chosen to ensure that the rate of convergence of the solution of the discretized collocation equations to the true solution $y(t)$ is the same as the rate of convergence of the solution to the original collocation system (14), see [8].

The quadrature formulas can be made order-preserving by choosing the collocation points, $t_{m,j}$, and the quadrature abscissas to coincide [8]. In particular, we assume that for the mth element the collocation points are given by

$$t_{m,j} = t_m + \alpha_j h_m, \quad j = 0, \ldots, p.$$

Note, the definition of the $t_{m,j}$ (8) imply that $\alpha_0 = 0$ and $\alpha_p = 1$. Since the functions $L_{m,j}(t)$ are only needed for $t \in [t_m, t_{m+1}]$, we then have the simplification, for $s = (t - t_m)/h_m$, that

$$L_{m,j}(t) = L_j(s) = \prod_{\substack{i=0,\\ i\neq j}}^{p} \frac{s - \alpha_i}{\alpha_j - \alpha_i}, \quad t \in [t_m, t_{m+1}]\,.$$

The specific quadrature approximations to the integrals within (11) we choose make use of quadrature abscissas with relative spacing related to the α_j. In particular, the general quadrature rule for an arbitrary function, $g(s)$, for $s \in [-1, 1]$ is

$$\int_{-1}^{1} g(s)\, ds \approx \sum_{k=0}^{p} g\left(z_k\right) \omega_k, \tag{15}$$

where the quadrature weights, ω_k, are given by

$$\omega_k = \int_{-1}^{1} L_k(s)\, ds.$$

The α_k are related to the chosen quadrature abscissas, z_k, by

$$\alpha_k = \frac{z_k + 1}{2}.$$

One common choice for the quadrature rule is to use Gaussian quadrature. The results presented in Section 4 use Gauss-Lobatto quadrature rules and abscissas [10, 21] for (15).

Two classes of integrals are contained within the matrices $B_{m,m'}$. For those matrices where $m' < m$, that is matrices below the diagonal in (13), we have the simplification

$$\begin{aligned}(B_{m,m'})_{j,j'} &= \int_{t_{m'}}^{t_{m'+1}} K\left(t_{m,j}, s\right) \phi_{m',j'}(s)\, ds, \\ &= \frac{h_m}{2} \int_{-1}^{1} K\left(t_{m,j}, t_{m'} + \frac{1+s}{2} h_m\right) \phi_{m',j'}\left(t_{m'} + \frac{1+s}{2} h_m\right) ds, \end{aligned} \tag{16}$$

since $\phi_{m',j'}(t)$ is zero outside of the m'th element. The application of (15) to (16) implies

$$\begin{aligned}(B_{m,m'})_{j,j'} &\approx \frac{h_m}{2} \sum_{k=0}^{p} K\left(t_{m,j}, t_{m',k}\right) \phi_{m',j'}\left(t_{m',k}\right) \omega_k, \\ &= \frac{h_m}{2} K\left(t_{m,j}, t_{m',j'}\right) \omega_{j'}, \end{aligned} \tag{17}$$

as $\phi_{m',j'}(t_{m',k}) = \delta_{j',k}$. Here $\omega_{j'}$ refers to the j'th quadrature weight.

The second class of integrals in (13) occurs for matrices on the diagonal (*i.e.* when $m' = m$). For the integrals within these matrices, the domain of integration contains a portion of an element when simplified. As the support of $\phi_{m,j'}(t) \subseteq [t_m, t_{m+1}]$, equation (11) reduces to

$$\begin{aligned}(B_{m,m})_{j,j'} &= \int_{t_m}^{t_{m,j}} K\left(t_{m,j}, s\right) \phi_{m,j'}(s)\, ds, \\ &= \frac{\alpha_j h_m}{2} \int_{-1}^{1} K\left(t_{m,j}, t_m + \frac{1+s}{2} \alpha_j h_m\right) \phi_{m,j'}\left(t_m + \frac{1+s}{2} \alpha_j h_m\right) ds. \end{aligned} \tag{18}$$

The application of (15) to (18) then gives the quadrature approximation

$$(B_{m,m})_{j,j'} \approx \frac{\alpha_j h_m}{2} \sum_{k=0}^{p} K\left(t_{m,j}, t_m + \alpha_k \alpha_j h_m\right) \phi_{m,j'}\left(t_m + \alpha_k \alpha_j h_m\right) \omega_k. \tag{19}$$

Replacing the matrix (13) in (14) and the matrices, $B_{m,m'}$, in (12) by the corresponding matrices with entries given by (16) and (18), we obtain the discrete collocation approximation to the original Volterra integral equation (1).

Denote by $\hat{y}_h(t)$ the solution to the discretized collocation problem. That is, $\hat{y}_h(t)$ satisfies (9) with the coefficients $Y_{m,j}$ replaced by those obtained from solving (14) with the quadrature approximations to the matrices, $B_{m,m'}$. With these choices, one can show the following error estimate [8, Theorems 2.2.3 and 2.2.11].

Theorem 2.1. *Let $f(t) \in C^k(I)$, $K(t,s) \in C^k(D)$, and assume $p = k$. If the quadrature formula* (15) *is based on corresponding interpolatory $p+1$ point polynomials with abscissas given by $\{\alpha_j\}$ then*

$$\|y(t) - \hat{y}_h(t)\| = \sup_{t \in [0,T]} |y(t) - \hat{y}_h(t)| = O(h^p).$$

We refer the interested reader to [8] for more detailed convergence results.

Please note, while this theorem shows convergence as h is decreased and p kept fixed, in many situations increasing p before changing h can give significant gains in accuracy and computational performance. This is the general strategy we use on the examples of Section 4, where the partitioned quadrature qualocation extension of the standard collocation method proves quite successful.

For smooth problems the most common, and simplest, choice for the elements is uniform spacing. In this case we take $t_m = mh$ with $h = T/M$, for $m = 0, \ldots, M$. We shall subsequently refer to the standard collocation method with uniform polynomial interpolants of order p on each element, uniformly spaced elements, and Gauss-Lobatto interpolation abscissa and quadrature rules as the `HpStd` method. Alternatively, one can use Chebyshev interpolation abscissa and Clenshaw-Curtis quadrature. For integrating smooth functions, Clenshaw-Curtis quadrature rules will often exhibit comparable accuracy to Gaussian-quadrature [35]. We expect to see comparable accuracy when using Chebyshev interpolation and Clenshaw-Curtis quadrature instead of Gauss-Lobatto interpolation and Gaussian quadrature.

2.1.2. Element Spacings and Quadrature Approximations for Kernels with Sharp Gradients

The freedom to independently choose element spacing, polynomial interpolant order per element, and quadrature approximations to the basis function integrals (11) is a major strength of the partitioned quadrature qualocation approach we now adopt. In this section we will describe alternative choices for the element spacing and the quadrature rules used to evaluate the integrals (11) which can significantly improve computational efficiency and accuracy for kernels with sharp gradients. Examples demonstrating these improvements are given in Section 4.2.

Consider the choice of element spacing. For integral equations (1) where the solution requires higher resolution in the neighborhood of certain points graded element meshes are often used [8, 6, 7, 33]. For example, a graded mesh that provides higher resolution as one approaches the origin would be given by the choice

$$t_m = \left(\frac{m}{M}\right)^r T, \quad m = 0, \ldots, M, \tag{20}$$

for $r > 1$. These types of meshes are often used in the solution of weakly singular Volterra integral equations of the second kind where the kernel, $K(t, s)$, has an integrable singularity of the form $\log(t - s)$ or $(t - s)^{-\beta}$ with $0 < \beta < 1$ [8, 6, 7, 33]. The corresponding solutions, $y(t)$, are generally continuous, but not differentiable, as $t \to 0$ [8]. While uniform element spacings will only provide lower orders of convergence for these types of kernels, graded meshes can be employed to recover higher orders [8, 6, 7, 33]. For equations with smooth solutions that contain sharp gradients, uniform element spacings may in practice require a large number of elements before providing a good approximation to the underlying solution. In such situations a graded mesh may provide similar accuracy with a smaller number of elements.

Graded meshes can also be used to improve the accuracy of the quadrature approximations (11) when using the quadrature choices of the previous section. For example, consider the kernel

$$K(t, s) = \frac{t + \epsilon}{s + \epsilon},$$

for small ϵ. This kernel is smooth for all $\epsilon > 0$, but contains sharp gradients near $s = 0$ when ϵ is small and $t \gg \epsilon$. By concentrating elements near the origin, $s = 0$, the graded mesh (20) will provide better accuracy than a uniform mesh in evaluating the integrals (11) for this example.

One potential drawback to using graded meshes is the (possible) introduction of round-off error, which can decrease the accuracy of the solution approximation. When using graded meshes to study solutions to weakly singular Volterra integro-differential equations this problem has been observed [7, 33]. (Note, both references also discuss how to choose the grading exponent for such equations, with the later proposing modifications to help avoid round-off errors.) For the examples considered in Section 4 we did not observe this problem. We leave a more thorough investigation of how to best choose the number of elements and grading exponents so as to maximize accuracy to future work.

The choice of quadrature rules can also be used to significantly improve computational accuracy and performance. When evaluating the integrals (11) in regions where the kernel has sharp gradients, it may be

beneficial to significantly increase the local accuracy (or resolution) in certain regions of (t, s) space. For example, consider (1) with the choice $y(t) = t\exp(-t)$ and

$$K(t,s) = \frac{1}{(t-s+\epsilon)^2},$$

with $\epsilon > 0$, and $f(t)$ defined by (1). For small ϵ, the integrals (11) will require higher resolution when $s \approx t$. The approach we introduce is to use partitioned quadrature qualocation, replacing the Gauss-Lobatto quadrature rules (15) used in the `HpStd` method with composite Gauss-Lobatto rules. That is, we partition the interval, $[-1, 1]$, as

$$-1 = s_0 < s_1 \leq \cdots \leq s_L = 1,$$

and replace the quadrature rule (15) we apply to each integral (11) with

$$\int_{-1}^{1} g(s)\,ds = \sum_{l=0}^{L-1} \int_{s_l}^{s_{l+1}} g(s)\,ds \approx \sum_{l=0}^{L-1} \frac{s_{l+1}-s_l}{2} \sum_{k=0}^{p'} g\left((s_{l+1}-s_l)\frac{1+z'_k}{2} + s_l\right)\omega'_k. \tag{21}$$

Note, we subsequently refer to L as the "number of partitions", so that "one partition" refers to the case where the interval $[-1, 1]$ has not been subdivided. We again use Gauss-Lobatto quadrature rules for the quadrature nodes, z'_k, and weights, ω'_k, and, for simplicity, use the same Gauss-Lobatto rule on each partition. We do not, however, require that the choice of quadrature nodes correspond to the collocation points as in the `HpStd` method or that the number of quadrature nodes per partition, $p' + 1$, agree with the number of collocation points. In Section 4.2.1 we show how the use of graded mesh spacings for the partitions, $\{s_0, \ldots, s_L\}$, concentrated around the sharp gradients in a kernel can lead to significant computational improvements for a regularization of a kernel that is not Lebesgue integrable.

While the partitioned quadrature approach can significantly improve the accuracy in evaluating the integrals (11), the simplified formula (18) for off-diagonal matrices will no longer hold. As such, for certain kernels with sharp gradients it may not in practice prove more computationally efficient to use partitioned quadrature instead of using the `HpStd` method with a large number of elements.

2.2. Other Methods

There are many other numerical methods for the solution of Volterra integral equations of the second kind which have been proposed. In our study of the performance of the partitioned quadrature collocation methods for solving equations with smooth, "nice", kernels, and smooth kernels with sharp gradients in Section 4, we also consider several methods where the solution and quadrature approximations are coupled. We specifically consider a sixth-order Gregory method [17], denoted by `Greg6`, a global spectral method [34], denoted by `GLI`, and a double-exponential transform-based Sinc method [25], denoted by `Sinc`.

The `Greg6` method was previously described in the appendix to [17] and is a standard Gregory-type method [12]. A fixed number of time steps, N, are chosen, and the solution to (1) is sought at the discrete time points $t_n = nh$, where $n = 0, \ldots, N$ and $h = T/N$. For the general equation (1), the work required to solve for the numerical solution at all N time points is $O(N^2)$. As described in [17], we make use of the FFT-based optimization originally proposed in [15] for use with Runge-Kutta methods to rapidly solve equations with convolution kernels, $K(t, s) = K(t-s)$. In this special case, the asymptotic work to solve (1) by the `Greg6` method is $O(N\log^2(N))$. If Y_n denotes the numerical approximation to $y(t_n)$, the `Greg6` method requires the first ten values $Y_0, \ldots, Y_9$ as starting initial data. To obtain $Y_1, \ldots, Y_9$ we use a sixth order explicit Runge-Kutta method [12] (see [23] for its tableaux).

The `GLI` method of [34] corresponds to the `HpStd` method with only one element, $M = 1$. In [34] it is shown that for smooth kernels the `GLI` method should have spectral convergence. In particular, the error bound derived in [34] depends on $L^2(0, T)$ norms of the higher derivatives in the second argument of the kernel. Increasing the smoothness of $y(t)$ and $K(t, s)$ allows higher-order derivatives in the error bound, and also increases the convergence rate. Note, however, that if the $L^2(0, T)$ norm of the kernel's derivatives increases when higher derivatives are taken this may negate the faster convergence rate (for any

computationally tractable number of collocation points). We indeed see this problem for each of the smooth kernels with sharp gradients used in Section 4.2.

The Sinc method was developed for kernels, $K(t,s)$, smooth on the domain $D = \{(t,s) \mid 0 < t < T, 0 < s < T\}$ with singularities only allowed for $s = 0$ or $s = T$. In particular, let Ω_d denote the strip in the complex plane,

$$\Omega_d = \{z \in \mathbb{C} \mid |\mathrm{Im}\, z| < d\},$$

and

$$\Omega_{d(\varepsilon)} = \{z \in \mathbb{C} \mid |\mathrm{Re}\, z| < 1/\varepsilon,\, |\mathrm{Im}\, z| < d(1-\varepsilon)\}.$$

We shall subsequently denote the boundary of $\Omega_{d(\varepsilon)}$ by $\partial\Omega_{d(\varepsilon)}$. Exponential convergence of the `Sinc` method is expected, based on the results of [25], when

1. There is a known double-exponential transformation function [25], $\phi(s) : \mathbb{R} \to (0,T)$, such that the transform, $\psi(t,s)$, of the kernel, $K(t,s)$, in the s coordinate grows at most double exponentially, *i.e.*

$$\psi(t,s) = K\left(t, \phi(s)\right) \phi'(s) \leq C e^{-\alpha\, e^{|s|}}, \quad \forall s \in \mathbb{R},$$

   for some real $\alpha > 0$.
2. $\psi(t,z)$ is analytic for $z \in \Omega_d$.
3. $$\lim_{\varepsilon \to 0} \int_{\partial\Omega_{d(\varepsilon)}} |\psi(t,z)|\, |dz| < \infty,$$

with these conditions holding uniformly for all $t \in (0,T)$, see [25]. In addition, $y(\phi(z))$, where $y(t)$ is the solution to (1), must be analytic and bounded for $z \in \Omega_d$.

In [25] it is demonstrated that even when these conditions are not satisfied the `Sinc` method may still exhibit spectral convergence. We do not attempt to verify which of the examples of Sections 4.1 and 4.2 satisfy the preceding conditions. For all the examples of Section 4.1, and the first sharp gradient example of Section 4.2, the `Sinc` method performs quite well. In contrast, for the latter two examples of Section 4.2 the method is not effective.

Note, there are many other numerical methods for the solution of (1) with smooth kernels that we do not consider. Omitted methods include many fast and highly optimized methods. We refer readers interested in these methods to the many references given in the introduction.

3. Numerical Implementation and Optimizations

We have developed MATLAB m-file based implementations of each of the standard collocation method of Section 2.1.1, `HpStd`, variations of our partitioned quadrature qualocation method method from Section 2.1.2 (with and without graded element meshes), `HpPar`, the sixth-order Gregory method described in [17], `Greg6`, the double-exponential Sinc method of [25], `Sinc`, and the global Gauss-Lobatto spectral method of [34], `GLI`.

We wish to emphasize that we make no claim to having optimal implementations of each method. A number of computational speedups, described below, have been exploited where possible, but we have no doubt that further improvements could be made. Instead, we hope that our results give a representation of what performance to expect for a typical implementation of the basic algorithms with *some* thought given to optimization.

We developed two implementations of the `Greg6` method, one for convolution kernels and one for non-convolution kernels. Both follow the formulations described in Section 2.2 and [17], with the convolution kernel algorithm taking advantage of the FFT-based optimization to give improved running times. One drawback of our particular implementation of this optimization is that it currently can not handle a non-power of two number of time-steps (though it certainly could be modified to do so). In particular, if given a non-power of two number of time-steps as input our implementation currently uses the optimized FFT-based algorithm up to the nearest power of two preceding the desired number of time-steps, and then

uses the non-FFT version of the algorithm for the remainder of the time-steps. As such, when studying performance of the method in Section 4 for various error tolerances the number of time-steps were always taken in powers of two. For convolution kernels, we also pre-evaluated the kernel, $K(t-s)$, at each discrete time point in the beginning of the routine, and then used the cached values of the kernel in the subsequent calculations. This optimization relies on the discrete values of $t-s$ that are used in the course of a simulation corresponding to time points at which the solution is to be solved for. None of the other solution methods have this property, and so could not take advantage of this optimization. Finally, in evaluating the discrete convolution terms in the FFT-based method we found that MATLAB's built-in discrete convolution routine, `conv`, and the MATLAB Signaling Toolbox FFT-based discrete convolution routine, `fftfilt`, performed poorly for sufficiently large vectors. We instead made use of the `convfft` routine [26], which we found to perform significantly faster for large vectors.

A number of implementation optimizations have been exploited for both the `HpStd` and `HpPar` methods. We assumed that the polynomial order was constant on each element. This optimization allows the Gauss-Lobatto nodes to only be evaluated once in the `HpStd` method and twice in the `HpPar` method (once for the collocation points and once for the quadrature abscissas). Our particular implementation of the `HpPar` method was not optimal, instead evaluating the Gauss-Lobatto nodes three times. For the smooth, well-behaved kernels of Section 4.1 this optimization gave a significant improvement in the performance of the `HpStd` method. (Our initial implementation of the `HpStd` method evaluated the Gauss-Lobatto nodes once per element, which led to the worst performance of any of the methods we examined for well-behaved kernels.) Note, to evaluate the Gauss-Lobatto nodes and weights we made use of the `lglnodes` routine [36].

We also optimized both the `HpStd` and `HpPar` implementations through the use of barycentric Lagrange interpolation for evaluating the interpolating polynomials within the integrals $(B_{m,m})_{j,j'}$. The barycentric weights only needed to be calculated once for a given simulation. The ease of evaluating the barycentric polynomials led to a significant speedup versus repeated construction and evaluation of the Lagrange form of the interpolating polynomial. Pre-calculating and caching the values of the Lagrange interpolation polynomials at all the quadrature points in (21) also gave a noticeable speedup for the `HpPar` method. Since we assumed that the number of quadrature partitions was the same for each element, and number of quadrature points was the same for each partition, these values could be reused in evaluating the integrals $(B_{m,m'})_{j,j'}$ for $m < m'$ by the quadrature rule (21).

Our initial implementation of the `HpStd` method did not take advantage of the optimization given by equation (17) to simplify the determination of the entries of the $B_{m,m'}$ sub-matrices when $m' < m$. The addition of this specific formula dramatically improved the performance of this method.

Finally, we would like to point out one optimization we elected not to exploit. Instead of solving (10) by forward block substitution through the use of (12), we fully formed and then solved the linear system (14) (using the backslash operator in MATLAB).

Both the `Sinc` and `GLI` methods were implemented as described in [25] and [34] respectively. For the `GLI` method, we evaluated the Lagrange interpolation polynomials through the use of Legendre functions as suggested by the authors [34]. The sine integral,

$$\mathrm{Si}(x) = \int_0^x \frac{\sin(x')}{x'}\,dx',$$

in the `Sinc` method was evaluated through the use of a double-exponential transform as also suggested by the authors [25]. The evaluation of this integral took as much as half of the median running time for the nice, smooth examples of Section 4. Through the use of tabulation we believe we could have significantly reduced the time to evaluate this function, but did not take advantage of this potential optimization.

4. Examples

In this section we compare the computational performance for fixed absolute error tolerances of the standard collocation formulation of Section 2.1.1, `HpStd`, variations of the partitioned quadrature qualocation method with graded element meshes from Section 2.1.2, `HpPar`, the sixth-order Gregory method described in

Section 2.2, `Greg6`, the double-exponential Sinc method of [25], `Sinc`, and the global Gauss-Lobatto spectral method of [34], `GLI`.

For each example problem a collection of absolute error tolerances were specified. A systematic numerical parameter search was then performed for each method to determine a resolution where the method "first" satisfied a given tolerance. The specific search methodology used for each method is described in more detail in Sections 4.1 and 4.2.1. Once a set of numerical parameters were found where a method satisfied a given error tolerance, the MATLAB routine `timeit` [13] was used to estimate the method's running time. `timeit` has the benefit of "warming up" the method to avoid m-file initialization overhead (such as the initial loading of the program to memory and compiling of the m-file storing the routine). It also estimates and runs the program a minimum number of times to ensure at least one second of computing time is used. The timing estimate it provides is then given by the median time among each of these runs.

All examples given in this section were run within MATLAB 7.5 on a Sun Fire X4600 M2 x64 server running Red Hat Linux version 4.1.2-42, with kernel version 2.6.18-92.1.10.el5. The server was configured with four AMD Opteron Model 8220 processors (2.8GHz dual-core) and 16GB of RAM.

4.1. Smooth, Well-Behaved Examples

In this section we study the relative performance of the `HpStd`, `Greg6`, `Sinc`, and `GLI` methods for several "nice" smooth examples. As we will see, the basic `HpStd` method is sufficiently fast for these examples that we saw no need to consider other element spacings or our more general partitioned quadrature qualocation method. The results of this section provide a baseline to distinguish how the relative performance of the methods changes when we subsequently consider kernels with sharp gradients in Section 4.2.

The error metric we used was simply the discrete maximum norm error at the "natural" solution points returned by a given method. For each individual method, the specific absolute error metrics and numerical parameter search strategies used are as follows. Let $[0, T]$ denote the interval we wish to solve (1) on. For the `HpStd` method there are two numerical parameters that may be varied: the polynomial order, p, and the number of elements, $M_{\texttt{HpStd}}$. (Recall we assumed the elements are uniformly spaced with uniform polynomial order in the `HpStd` method.) The element boundary points are given by $t_m = mh$, for $h = T/M_{\texttt{HpStd}}$, so that the collocation points within the mth element are the Gauss-Lobatto points on the interval $[t_m, t_{m+1}]$, $t_{m,j} = mh + \alpha_j h$ (see Section 2.1.1). The error metric, $e_{\texttt{HpStd}}$, is given by

$$e_{\texttt{HpStd}}\left(M_{\texttt{HpStd}}, p_{\texttt{HpStd}}\right) = \max_{\substack{m=0,\ldots,M_{\texttt{HpStd}}-1 \\ j=0,\ldots,p_{\texttt{HpStd}}}} \left|y(t_{m,j}) - \hat{y}_h(t_{m,j})\right|.$$

To determine $M_{\texttt{HpStd}}$ and $p_{\texttt{HpStd}}$ such that $e_{\texttt{HpStd}}\left(M_{\texttt{HpStd}}, p_{\texttt{HpStd}}\right)$ satisfied a given error tolerance we employed the following search strategy. Starting with one element, $p_{\texttt{HpStd}}$ was increased from two to a maximum of ten. If the tolerance was not satisfied at this point, one additional element was added and $p_{\texttt{HpStd}}$ was reset to two. The procedure was then repeated until the error tolerance was satisfied or the number of elements reached a maximum of 500.

We denote by t_n, $n = 0, \ldots, N_{\texttt{Greg6}}$, the solution points for the `Greg6` method. Here $N_{\texttt{Greg6}}$ denotes the number of time-steps. The absolute error metric for the Gregory method, $e_{\texttt{Greg6}}$, is given by

$$e_{\texttt{Greg6}}\left(N_{\texttt{Greg6}}\right) = \max_{n=0,\ldots,N_{\texttt{Greg6}}} \left|y(t_n) - Y_n\right|,$$

where Y_n denotes the Gregory method solution at time t_n. We refined $N_{\texttt{Greg6}}$ by doubling until $e_{\texttt{Greg6}}\left(N_{\texttt{Greg6}}\right)$ was below a given tolerance. The first such value of $N_{\texttt{Greg6}}$ found was then used for determining running time of the `Greg6` method for that tolerance. We chose this strategy since our FFT-based implementation of the `Greg6` method for convolution kernels was only optimized to handle values of $N_{\texttt{Greg6}}$ that were powers of two, see Section 3. Note, however, this was simply done for simplicity in implementation; it is possible to apply the algorithm of [15] for non-power of two values of $N_{\texttt{Greg6}}$.

For the `Sinc` method only one numerical parameter, the number of terms of the Sinc expansion, was varied. Let $N_{\texttt{Sinc}}$ be defined such that the total number of terms used is $2N_{\texttt{Sinc}} + 1$. Denote by t_j,

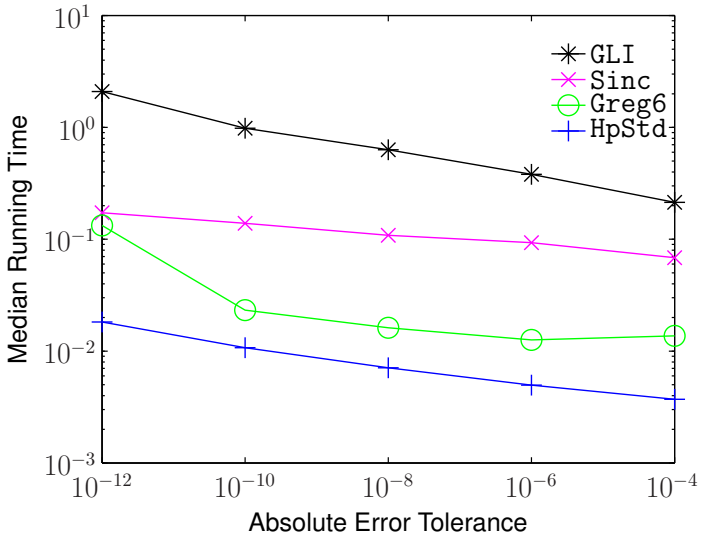

Figure 1: Median running times found for each method listed in the inset at given absolute error tolerances. $y(t) = \exp(-t)$, $K(t,s) = \exp(t-s)$, and $g(t) = 0.5(\exp(-t) + \exp(t-2))$. The corresponding numerical parameters found for each method to satisfy a given tolerance are listed in Table 1.

	Tolerances:				
	1e-04	1e-06	1e-08	1e-10	1e-12
$N_{\texttt{GLI}}$	10	12	14	16	20
$N_{\texttt{Sinc}}$	30	40	46	58	70
$N_{\texttt{Greg6}}$	128	128	256	512	4096
$M_{\texttt{HpStd}}$, $p_{\texttt{HpStd}}$	1, 10	2, 8	2, 10	3, 10	5, 10

Table 1: Numerical parameters associated with Figure 1. N gives the numerical size parameter used in the simulation (*i.e.* number of time points or basis functions, see the text for details). For the `HpStd` method the polynomial degree, denoted by p_{HpStd}, for each simulation is also listed.

$j = -N_{\texttt{Sinc}}, \ldots, N_{\texttt{Sinc}}$, the Sinc points and by Y_j the numerical approximation to $y(t_j)$, see [25] for details. The absolute error metric for the `Sinc` method, $e_{\texttt{Sinc}}$, is given by

$$e_{\texttt{Sinc}}\left(N_{\texttt{Sinc}}\right) = \max_{j=-N_{\texttt{Sinc}},\ldots,N_{\texttt{Sinc}}} \left|y(t_j) - Y_j\right|.$$

For a particular error tolerance, $N_{\texttt{Sinc}}$ was initially set to two, and then incremented by two until $e_{\texttt{Sinc}}\left(N_{\texttt{Sinc}}\right)$ was below that tolerance.

Finally, for the global Gauss-Lobatto spectral method, `GLI`, the one numerical parameter varied was the number of Gauss-Lobatto points. Let $t_0, \ldots, t_{N_{\texttt{GLI}}}$ denote the $N_{\texttt{GLI}}+1$ Gauss-Lobatto points on the interval $[0,T]$. We denote by $Y_j \approx y(t_j)$ the numerical solution at the jth Gauss-Lobatto point. The error metric for the spectral method, $e_{\texttt{GLI}}$, is given by

$$e_{\texttt{GLI}}\left(N_{\texttt{GLI}}\right) = \max_{j=0,\ldots,N_{\texttt{GLI}}} \left|y(t_j) - Y_j\right|.$$

For a specified tolerance, $N_{\texttt{GLI}}$ was incremented by two, beginning at two, until $e_{\texttt{GLI}}\left(N_{\texttt{GLI}}\right)$ was smaller than that tolerance.

The first example we examined was a convolution equation, where

$$y(t) = e^{-t}, \qquad K(t,s) = e^{t-s}, \qquad g(t) = \frac{e^{-t} + e^{t-2}}{2},$$

with $t \in [1, 10]$. Note, since the kernel has a convolution form we were able to use the faster FFT-based version of the `Greg6` method. Figure 1 and Table 1 show the computational performance results we found. All four methods perform quite well, taking at most on the order of one second to satisfy absolute error tolerances as small as 1e-12.

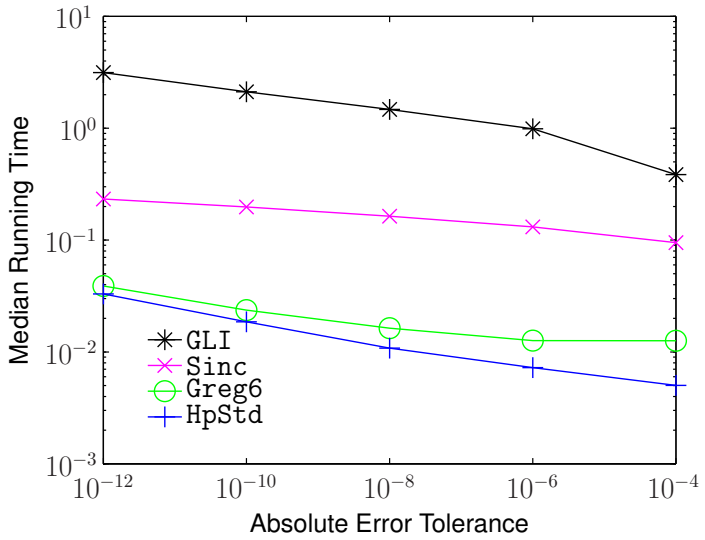

Figure 2: Median running times found for each method listed in the inset at given absolute error tolerances. $y(t) = 2\sqrt{3}\sin(\sqrt{3}t/2)\exp(-t/2)/3$, $K(t,s) = \cos(t-s)$, and $g(t) = \sin(t)$. The corresponding numerical parameters found for each method to satisfy a given tolerance are listed in Table 2.

	Tolerances:				
	1e-04	1e-06	1e-08	1e-10	1e-12
$N_{\texttt{GLI}}$	12	16	18	20	22
$N_{\texttt{Sinc}}$	40	54	66	78	90
$N_{\texttt{Greg6}}$	128	128	256	512	1024
$M_{\texttt{HpStd}}$, $p_{\texttt{HpStd}}$	2, 8	2, 10	3, 10	5, 10	7, 10

Table 2: Numerical parameters associated with Figure 2. N gives the numerical size parameter used in the simulation (*i.e.* number of time points or basis functions, see the text for details). For the `HpStd` method the polynomial degree, denoted by p_{HpStd}, for each simulation is also listed.

The second example we examined was again a convolution kernel with,

$$y(t) = \frac{2\sqrt{3}}{3}\sin(\frac{\sqrt{3}t}{2})e^{-t/2}, \qquad K(t,s) = \cos(t-s), \qquad g(t) = \sin(t),$$

and $t \in [0, 4\pi]$. Figure 2 and Table 2 show the computational performance results for this example. As in the first example, all four methods performed quite well, taking at most on the order of a few seconds. Note the performance of the `Greg6` and `HpStd` methods are very close.

The final example we examined was a non-convolution equation, with

$$y(t) = e^{4t}, \qquad K(t,s) = e^{ts}, \qquad g(t) = e^{4t} + \frac{e^{t(t+4)} + e^{-(t+4)}}{t+4},$$

and $t \in [-1, 1]$. Figure 3 and Table 3 show the computational performance results for this example. As in the previous examples, the methods all perform quite well, with running times of at most the order of one second.

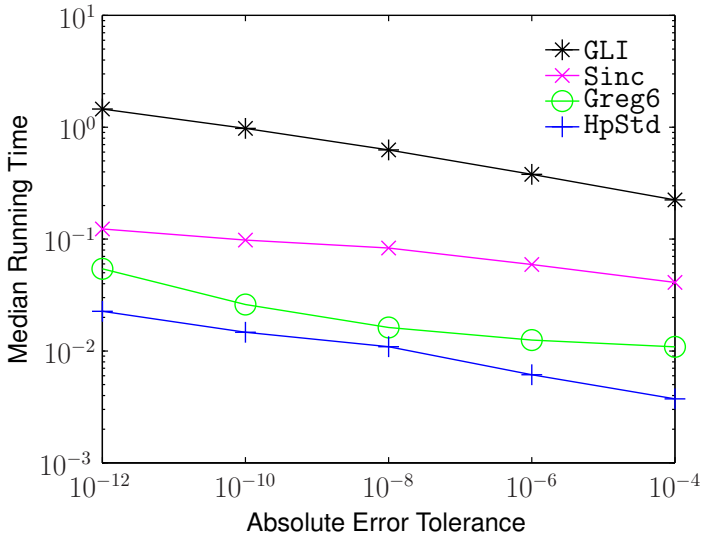

Figure 3: Median running times found for each method listed in the inset at given absolute error tolerances.$y(t) = \exp(4t)$, $K(t,s) = \exp(ts)$, and $g(t)$ is given in the text. The corresponding numerical parameters found for each method to satisfy a given tolerance are listed in Table 3.

	Tolerances:				
	1e-04	1e-06	1e-08	1e-10	1e-12
$N_{\texttt{GLI}}$	10	12	14	16	18
$N_{\texttt{Sinc}}$	18	26	36	42	52
$N_{\texttt{Greg6}}$	64	128	256	512	1024
$M_{\texttt{HpStd}}$, $p_{\texttt{HpStd}}$	1, 10	2, 9	3, 10	4, 10	6, 10

Table 3: Numerical parameters associated with Figure 3. N gives the numerical size parameter used in the simulation (*i.e.* number of time points or basis functions, see the text for details). For the `HpStd` method the polynomial degree, denoted by p_{HpStd}, for each simulation is also listed.

For our implementations, the `HpStd` method appeared to provide the best performance, though the `Greg6` method is quite close for most tolerances. Note, the `Greg6` method may in fact be able to obtain similar, or faster, performance than the `HpStd` method if we had allowed non-power of two values of $N_{\texttt{Greg6}}$. All four methods perform quite well, with sufficiently fast running times that any would be appropriate for general use with similar kernels.

With the exception of the `Greg6` method, for all three examples the methods required very few degrees of freedom to achieve high accuracy. For the finest tolerance of 1e-12, the `HpStd` method required at most seven elements with degree ten polynomials (71 collocation points), the `Sinc` method required 181 Sinc points, and the `GLI` method required 23 Gauss-Lobatto collocation points. Both the `GLI` and `Sinc` methods roughly double their number of degrees of freedom as the tolerance is increased from 1e-4 to 1e-12, while the `Greg6` and `HpStd` methods require substantially larger increases. This faster growth in numbers of degrees of freedom for the latter two methods appears to be compensated for by the relative efficiency of their implementations.

4.2. Examples with Sharp Gradients

In this section we examine the performance of the `HpStd`, `HpPar`, `Greg6`, `GLI`, and the `Sinc` methods for smooth kernels containing sharp gradients. Note, `HpPar` is our partitioned quadrature qualocation method. The examples we study are chosen to depend on a small parameter, and in the limit that the parameter

approaches zero the kernels become singular. That is, each example we consider is a regularization of a kernel with *non-integrable* singularities (in the Lebesgue sense). In particular, in Section 4.2.1 we consider the example that motivated this work, where the kernel approaches the non-integrable singularity,

$$\frac{1}{(t-s)^{3/2}},$$

in the limit that the small parameter approaches to zero.

When studying the convergence of the six numerical methods for these kernels, we found that our implementation of the global spectral method, `GLI`, was unable to solve any of the three test examples to the required tolerances in a computationally reasonable amount of time. Our implementation of the Sinc-based method was only able to resolve the first of the three examples of this section. This failure to resolve the other two examples is not unexpected as the method was not designed to handle the types of sharp gradients present in these problems. For the latter two examples, the singularities that appear lie on the line $t = s$, while the method assumes that any singularities occur only at $s = T$ or $s = 0$ (see Section 4.2 for details on the assumed regularity properties of the kernel for the Sinc method). When the regularization parameter is sufficiently small the `Sinc` method is therefore unable to efficiently solve the corresponding Volterra equations.

Our definitions of the error metrics remain the same as in the previous section. The strategies for determining the numerical parameters of a given method that "first" satisfy a specified absolute error tolerance also remain the same within this section, however, as we do not have an exact solution for the problem of Section 4.2.1 we use a different error metric there. (One that depends only on the numerical solution, see Section 4.2.1.)

We subsequently allow the elements in the `HpStd` method to be graded, with element endpoints given by (20). The local resolution of the solution representation may then be increased in regions with less smoothness (or sharper gradients). To indicate a specific grading exponent, r in (20), we will denote the method by `HpStdr`. For example, `HpStd4` will denote a grading exponent of $r = 4$. `HpStd` will continue to refer to the use of uniform element spacing, $r = 1$. Note, the particular grading formula (20) we use will cluster elements near $t = 0$. One could, of course, use a different grading that clusters near the other endpoint or a point inside the interval $[0, T]$ if necessary.

The partitioned quadrature qualocation-based `HpPar` method adds several additional numerical parameters from the `HpStd` method: the number of quadrature partitions per element, the degree of the quadrature rule per partition, and the choice of partition spacing. As for the `HpStd` method, we assume the polynomial degree is constant on each element. We further assume that the number of partitions is the same for each element, and the number of quadrature points is the same on each partition. For the first two examples of this section, the number of quadrature points per partition is taken to be the same as the number of collocation points per element, $p + 1$. With this choice, the `HpPar` method should reduce to the `HpStd` method when only one partition is used per element. In Section 4.2.1 we choose the number of quadrature points to be two more than the number of collocation points per element, $p + 3$. We allow both the element spacing and the quadrature partition spacing on each element to be graded. The element spacing is chosen as for the `HpStd` method, by (20). For the second example of this section and the example of Section 4.2.1 we found that grading the quadrature partition endpoints within each element could significantly improve computational performance. In both examples the kernels contain sharp gradients as $t \to s$. Recall that the integrals (11) simplify, as the support of $\phi_{m',j'} \subseteq [t_{m'}, t_{m'+1}]$, to (16) when $m' < m$ and to (18) when $m' = m$. Note that each of these integrals involves integration over only one complete, or partial, element. The particular graded partition spacing we chose in (21) uses the partition points,

$$s_l = 1 - 2\left(1 - \frac{l}{L}\right)^{r'}, \quad l = 0, \ldots, L,$$

where r' denotes the grading exponent. When applied to (16) and (18) this clusters points about the *end* of the integration range. We therefore obtain higher resolution in the vicinity of points where $s \approx t$, as appear in the integrals (18). One could continue to resolve the integrals (16) with uniform partitions, and

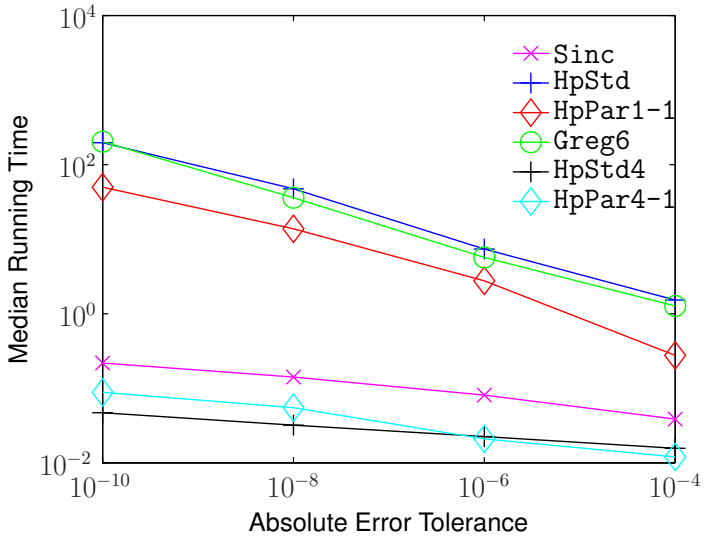

Figure 4: First smooth kernel with sharp gradients example (22) with $\varepsilon = 0.001$. The median running times found for each method listed in the inset are shown for the specified absolute error tolerances. The corresponding numerical parameters found for each method to satisfy a given error tolerance are listed in Table 4.

only use graded partitions for the integrals (18), however, the results presented herein use the same graded quadrature partition spacing for all the integrals (16) and (18). We will subsequently refer to the `HpPar` method with element grading exponent, r, and partition grading exponent, r', by the name `HpPar`r-r'. For example, `HpPar4-6` refers to using an element grading with $r = 4$ and a partition grading with $r' = 6$. The choice of uniform element and uniform partition spacing will then be denoted by the label `HpPar1-1`.

Note, how to best choose the two grading exponents, r and r' is an interesting question that we did not investigate herein. For problems using non-uniform element gradings we always chose $r = 4$, while for non-uniform partition spacing we choose $r' = 6$.

Our error metric for the `HpPar` method, denoted by $e_{\texttt{HpPar}}$, remains the same as for the `HpStd` method,

$$e_{\texttt{HpPar}}\left(M_{\texttt{HpPar}}, p\right) = \max_{\substack{m=0,\ldots,M_{\texttt{HpPar}}-1 \\ j=0,\ldots,p_{\texttt{HpPar}}}} \left|y(t_{m,j}) - \hat{y}_h(t_{m,j})\right|.$$

Note, $e_{\texttt{HpPar}}$ will also depend on the element spacing, quadrature order, number of quadrature partitions per element, and quadrature partition spacing. We will subsequently denote by $Q_{\texttt{HpPar}}$ the number of partitions per element. In addition to varying $p_{\texttt{HpPar}}$ and $M_{\texttt{HpPar}}$ when trying to reduce the error metric, $e_{\texttt{HpPar}}$, below a given tolerance, we now also vary $Q_{\texttt{HpPar}}$. When searching for the "first" set of $(p_{\texttt{HpPar}}, M_{\texttt{HpPar}}, Q_{\texttt{HpPar}})$ that satisfy a given tolerance, we initially set $p_{\texttt{HpPar}} = 2$, $Q_{\texttt{HpPar}} = 1$, and $M_{\texttt{HpPar}} = 1$. $p_{\texttt{HpPar}}$ is then incremented by one until reaching a maximum value of ten. If the error tolerance is not satisfied at this point then $p_{\texttt{HpPar}}$ is reset to two, and $Q_{\texttt{HpPar}}$ is incremented by five. If $p_{\texttt{HpPar}}$ reaches the value of ten and $Q_{\texttt{HpPar}}$ the value of 100, and the error tolerance is still not satisfied, then $M_{\texttt{HpPar}}$ is incremented by one, $p_{\texttt{HpPar}}$ is reset to two, and $Q_{\texttt{HpPar}}$ is reset to one. The process is then repeated.

The first example we consider is the Volterra integral equation (1) with

$$y(t) = e^t\left[1 + (t+\varepsilon)\log\left(1 + \frac{t}{\varepsilon}\right)\right], \qquad K(t,s) = -\frac{t+\varepsilon}{s+\varepsilon}, \qquad g(t) = e^t, \tag{22}$$

with $t \in [0,1]$. Note, as $\varepsilon \to 0$ this kernel becomes singular in s at $s = 0$, with a non-integrable singularity (in the Lebesgue sense). Moreover, the solution, $y(t)$ blows-up for all $t > 0$ as $\varepsilon \to 0$.

	Tolerances:			
	1e-04	1e-06	1e-08	1e-10
$N_{\texttt{Sinc}}$	16	32	52	74
$N_{\texttt{Greg6}}$	8192	16384	32768	65536
$M_{\texttt{HpStd}}$, $p_{\texttt{HpStd}}$	87, 10	168, 10	282, 10	433, 10
$M_{\texttt{HpStd4}}$, $p_{\texttt{HpStd4}}$	5, 9	7, 9	8, 10	11, 10
$M_{\texttt{HpPar1-1}}$, $p_{\texttt{HpPar1-1}}$, $Q_{\texttt{HpPar1-1}}$	13, 10, 11	43, 10, 6	95, 10, 6	174, 10, 6
$M_{\texttt{HpPar4-1}}$, $p_{\texttt{HpPar4-1}}$, $Q_{\texttt{HpPar4-1}}$	2, 10, 11	3, 10, 6	6, 9, 6	7, 10, 6

Table 4: Numerical Parameters associated with Figure 4 and equation (22). See text for definitions of the parameters. Note, the quadrature order of the `HpPar` methods was chosen such that the quadrature points agreed with the collocation points when only one partition was used per element.

	Tolerances:				
	1e-02	1e-03	1e-04	1e-05	1e-06
$N_{\texttt{Greg6}}$	131072	131072	131072	131072	131072
$M_{\texttt{HpStd}}$, $p_{\texttt{HpStd}}$	30, 10	63, 10	106, 10	158, 10	220, 10
$M_{\texttt{HpPar1-1}}$, $p_{\texttt{HpPar1-1}}$, $Q_{\texttt{HpPar1-1}}$	1, 10, 6	1, 10, 16	1, 10, 26	2, 10, 46	3, 10, 51
$M_{\texttt{HpPar4-6}}$, $p_{\texttt{HpPar4-6}}$, $Q_{\texttt{HpPar4-6}}$	1, 6, 11	1, 10, 11	1, 9, 16	2, 10, 16	3, 10, 21

Table 5: Numerical Parameters associated with Figure 5 and equation (23). See text for definitions of the parameters. Note, the quadrature order of the `HpPar` methods was chosen such that the quadrature points agreed with the collocation points when only one partition was used per element.

Figure 4 and Table 4 show the median running times and associated numerical parameters found for the various methods as the absolute error tolerance is varied. The best performing methods for this example were the `HpStd4` and `HpPar4-1` methods. Since the solution, $y(t)$, contains sharp gradients for ε small, it is not surprising that increasing the number of elements near $t = 0$ would allow the use of fewer elements (and hence increase the performance of these two methods). The number of elements required when a graded element spacing is used decreases by more than a factor of ten from the non-graded case for most tolerances. The use of uniformly spaced partitions does not help improve performance for this example. We might expect that using a quadrature partitioning that is graded towards the $s = -1$ end of the integrals (16) and (18) might be beneficial; particularly for the case that $m' = 0$. We did not, however, examine this case. Also, note that the `Sinc` method was faster than both the uniformly spaced `HpStd` and `HpPar` methods, and was within one order of magnitude of the performance of the graded `HpStd` and `HpPar` methods. Unlike the next two examples, the sharp gradient in the kernel near $s = 0$ can be handled by the `Sinc` method. The double-exponential transform used in the `Sinc` method changes the integration range in (1) so that $s = 0$ is moved off to negative infinity. Coupled with the sharp decay in the Jacobian of the transformation at negative infinity, the `Sinc` method is then able to resolve the sharp gradient.

The second example we consider is the Volterra integral equation (1) with

$$y(t) = te^{-t}, \qquad K(t,s) = \frac{1}{(t-s+\varepsilon)^2}, \tag{23}$$

$g(t)$ defined by the left-hand side of (1), and $t \in [0, 10]$. To evaluate $g(t)$ a cubic spline interpolation table was built with an absolute error tolerance of 1e-10. In building the table, the integral in (1) was evaluated through the use of MATLAB's built-in `quadgk` routine with an absolute error tolerance of one hundred times machine precision. Figure 5 and Table 5 show the median running times and associated numerical parameters found for the various methods as the absolute error tolerance is varied. Note, the `HpPar` methods are *slightly* increasing in running time as the tolerance is increased, but this is only visible on the expanded scale of the lower-left inset of Figure 5.

The singularity of the kernel as $\varepsilon \to 0$ now lies on the line $t = s$. As such, the `Sinc` method was

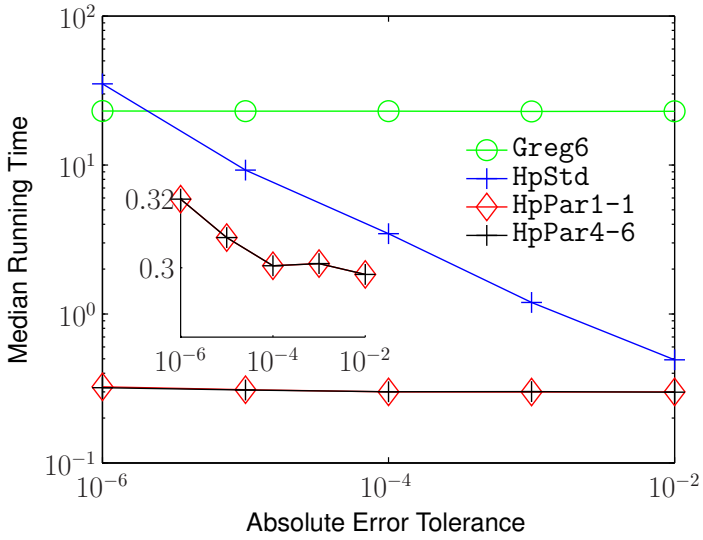

Figure 5: Second kernel with sharp gradient example (23) with $\varepsilon = 0.01$. The median running times found for each method listed in the upper-right inset are shown for the specified absolute error tolerances. The corresponding numerical parameters found for each method to satisfy a given error tolerance are listed in Table 5. The lower-left inset shows an expanded view of the two `HpPar` methods, showing their slight increase in running time as the absolute error tolerance is decreased.

no longer effective in solving this particular example for small ε (see Section 2.2 for a discussion of the regime of validity of the `Sinc` method). For the specified tolerances, the `Sinc` method did not converge in a computationally reasonable amount of time. In contrast, the use of the partitioned quadrature methods, `HpPar1-1` and `HpPar4-6`, are quite effective for this example, giving a large performance gain over the `HpStd` method. We found that grading the elements in either the `HpStd` method or the `HpPar` method had little effect on the median running times. The computational challenge is not in approximating the smooth solution, $y(t)$, in (23), but instead in accurately evaluating the integrals (16) and (18). As shown in Table 5, only a few elements are needed to accurately approximate the true solution, $y(t)$, however, a large number of partitions (for the `HpPar1-1` method) or a moderate number of *well-placed* partitions (for the `HpPar4-6` method) are necessary to accurately evaluate those integrals. While the graded partitioned quadrature method, `HpPar4-6`, did not significantly improve the computational running time, it did significantly reduce the total number of partitions required to satisfy a given tolerance. This example demonstrates one of the main benefits of the qualocation approach; separating the approximation of the integral in (1) from the numerical approximation of the solution to (1), $y(t)$.

4.2.1. Application to a Smooth Kernel with Sharp Gradients

The final example we consider arises from a problem one of the authors has previously investigated [17, 18], and was the original motivation for this work. We were interested in studying the numerical convergence of the solution to the discrete-space continuous-time diffusion equation with a sink term at the origin. This model arises as a special case when studying stochastic reaction-diffusion models of gene expression based on the reaction-diffusion master equation [20, 17, 18]. In particular, consider an infinite Cartesian lattice in three-dimensions comprised of cubic voxels of width h. We subsequently denote this space by $\mathbb{Z}^3 h = \{\boldsymbol{x} = h\boldsymbol{i} \mid \boldsymbol{i} \in \mathbb{Z}^3\}$, where $\mathbb{Z}^3$ denotes the space of all integer valued vectors with three components. Denote by $\mathbf{0} = (0, 0, 0)$ the zero vector. The problem we studied was that of a single molecule undergoing a continuous-time random-walk on the lattice, which in the $\mathbf{0}$ voxel could be removed from the lattice with probability per unit time k/h^3. (Here k should be interpreted as a bimolecular rate constant with units of `volume/time`.) Physically, this model can be thought of as approximating binding to a fixed target located

in the $\mathbf{0}$ voxel of the lattice.

Define $\boldsymbol{e}_d$ to be the unit vector along the dth coordinate axis of $\mathbb{R}^3$, and D to be the diffusion constant of the molecule (with units of area per time). We let $\delta_h(\boldsymbol{x})$ represent the discrete Dirac delta function,

$$\delta_h(\boldsymbol{x}) = \begin{cases} \frac{1}{h^3}, & \boldsymbol{x} = \mathbf{0}, \\ 0, & \text{otherwise.} \end{cases}$$

Denote by $p_h(\boldsymbol{x}, t)$, with $\boldsymbol{x} \in \mathbb{Z}^3 h$ and $t > 0$, the solution to the discrete-space continuous-time diffusion equation with sink term,

$$\frac{dp_h}{dt}(\boldsymbol{x}, t) = \frac{D}{h^2} \sum_{d=1}^{3} \Big[p_h(\boldsymbol{x} + \boldsymbol{e}_d, t) + p_h(\boldsymbol{x} - \boldsymbol{e}_d, t) - 2p_h(\boldsymbol{x}, t) \Big] - \frac{k}{h^3} \delta_h(\boldsymbol{x}) p_h(\mathbf{0}, t), \tag{24}$$

with the initial condition,

$$p_h(\boldsymbol{x}, 0) = \delta_h(\boldsymbol{x} - \boldsymbol{x}_0).$$

Here $\boldsymbol{x}_0$ denotes the initial location of the molecule.

Let $G_h(\boldsymbol{x}, t)$ denote the Green's function for the discrete-space continuous-time diffusion equation,

$$\frac{dG_h}{dt}(\boldsymbol{x}, t) = \frac{D}{h^2} \sum_{d=1}^{3} \Big[G_h(\boldsymbol{x} + \boldsymbol{e}_d, t) + G_h(\boldsymbol{x} - \boldsymbol{e}_d, t) - 2G_h(\boldsymbol{x}, t) \Big], \quad \boldsymbol{x} \in \mathbb{Z}^3 h, \quad t > 0,$$

with the initial condition,

$$G_h(\boldsymbol{x}, 0) = \delta_h(\mathbf{0}).$$

Using Duhamel's principle, we may rewrite (24) as the system of Volterra integral equations

$$p_h(\boldsymbol{x}, t) = G_h(\boldsymbol{x} - \boldsymbol{x}_0, t) - k \int_0^t G_h(\boldsymbol{x}, t - s) p_h(\mathbf{0}, s)\, ds, \quad \boldsymbol{x} \in \mathbb{Z}^3 h, \quad t \geq 0. \tag{25}$$

Notice, at $\boldsymbol{x} = \mathbf{0}$ this leads to a single Volterra integral equation of the second kind for $p_h(\mathbf{0}, t)$,

$$p_h(\mathbf{0}, t) = G_h(\boldsymbol{x}_0, t) - k \int_0^t G_h(\mathbf{0}, t - s) p_h(\mathbf{0}, s)\, ds. \tag{26}$$

Here we have made use of the property that $G_h(-\boldsymbol{x}_0.t) = G_h(\boldsymbol{x}_0, t)$. Once the solution to (26) has been obtained, $p_h(\boldsymbol{x}, t)$ may be evaluated at any point in space through (25).

The original question of interest, discussed in [17], was to determine what the solution to (25) converges to in the limit that $h \to 0$. We initially examined this numerically using the `Greg6` method [17], and were ultimately able to prove that for $\boldsymbol{x} \neq 0$ the solution converges point-wise to the solution to the discrete-space continuous-time diffusion equation, *i.e.*

$$\lim_{h \to 0} p_h(\boldsymbol{x}, t) = G_h(\boldsymbol{x} - \boldsymbol{x}_0, t), \quad \boldsymbol{x} \neq \mathbf{0}, \quad t > 0.$$

While the `Greg6` method was able to give insight into what the limit should be, we felt that its performance for small values of h limited what we could explore numerically. This motivated the present study.

There are several difficulties in numerically evaluating the solution to (26). First, the Green's function, $G_h(\boldsymbol{x}, t)$, may only be represented as an appropriate Bessel function or, as we choose to make use of, an inverse Fourier integral. The Bessel function representation is that

$$G_h(\boldsymbol{x}, t) = \frac{e^{-6Dt/h^2}}{h^3} \prod_{d=1}^{3} I_{(\boldsymbol{x})_d/h} \left(\frac{2Dt}{h^2} \right),$$

where by $(\boldsymbol{x})_d$ we mean the dth component of $\boldsymbol{x}$, and I_n denotes the modified Bessel function of the first kind of order n. The Fourier representation of G_h is given by

$$G_h(\boldsymbol{x},t) = \iiint_{\left[\frac{-1}{2h},\frac{1}{2h}\right]^3} e^{-4Dt\sum_{k=1}^{3}\sin^2(\pi h\xi_k)/h^2}\, e^{2\pi i\boldsymbol{\xi}\cdot\boldsymbol{x}}\, d\boldsymbol{\xi},$$

and in the appendix to [17] we explain our numerical method for its evaluation (a mix of tabulation for sufficiently large t values and direct evaluation by use of the double-exponential transform [24] and the trapezoidal rule for small t values). Note, as $h \to 0$ with $\boldsymbol{x}$ a fixed lattice point, $G_h(\boldsymbol{x},t)$ will converge point-wise to the Green's function of the free-space diffusion,

$$G(\boldsymbol{x},t) = \frac{1}{(4\pi Dt)^{3/2}} e^{-|\boldsymbol{x}|^2/(4Dt)}.$$

The kernel to (26), $G_h(\mathbf{0}, t-s)$, will then have a non-integrable singularity (in the Lebesgue sense) of the form

$$\frac{1}{(t-s)^{3/2}},$$

as $h \to 0$. It is sufficient for our purposes to subsequently assume that the evaluation of the kernel is an expensive operation, one that should be minimized in order to reduce the computational time in solving (26).

As we do not have an exact solution to (26), we assessed numerical error differently than in the preceding sections. For the `Greg6` method the absolute error was assessed by comparing the solution with $N_{\texttt{Greg6}}+1$ points to the solution with $N_{\texttt{Greg6}}/2+1$ points. That is, the error metric, $\tilde{e}_{\texttt{Greg6}}$, was

$$\tilde{e}_{\texttt{Greg6}}\left(N_{\texttt{Greg6}}\right) = \max_{n=0,2,4\ldots,N_{\texttt{Greg6}}} \left|\tilde{Y}_n - Y_{n/2}\right|.$$

Here $\tilde{Y}_n$ denotes the `Greg6` method solution at the nth time-step with $N_{\texttt{Greg6}}+1$ time points and $Y_{n/2}$ denotes the `Greg6` method solution with $N_{\texttt{Greg6}}/2+1$ time points. Our search strategy to find the value of $N_{\texttt{Greg6}}$ that reduces the error below a given tolerance was the same as the previous sections, but used $\tilde{e}_{\texttt{Greg6}}$ instead of $e_{\texttt{Greg6}}$ when comparing to the tolerance.

For the `HpPar` method, the modified error metric we used compared the current numerical solution, $\tilde{y}_h(t_{m,j})$, to the solution with one less polynomial degree but the same number of elements, $\hat{y}_h(t_{m,j})$, at the collocation points for the latter. That is, if $\tilde{y}_h$ uses degree $p_{\texttt{HpPar}}$ interpolants on each element and has $M_{\texttt{HpPar}}$ total elements, the collocation points where the two solutions are compared are the points,

$$t_{m,j} = t_m + \alpha_j h_m, \quad j = 0,\ldots,p_{\texttt{HpPar}}-1, \quad m = 0,\ldots,M_{\texttt{HpPar}}-1,$$

where the $\{\alpha_j\}_{j=0}^{p_{\texttt{HpPar}}-1}$ are the mapping of the $p_{\texttt{HpPar}}$ Gauss-Lobatto nodes to $[0,1]$. The new error metric, $\tilde{e}_{\texttt{HpPar}}$, is then

$$\tilde{e}_{\texttt{HpPar}}\left(M_{\texttt{HpPar}}, p_{\texttt{HpPar}}\right) = \max_{\substack{m=0,\ldots,M_{\texttt{HpPar}}-1\\ j=0,\ldots,p_{\texttt{HpPar}}-1}} \left|\tilde{y}_h(t_{m,j}) - \hat{y}_h(t_{m,j})\right|.$$

Our numerical procedure to search for the first set of $(p_{\texttt{HpPar}}, M_{\texttt{HpPar}}, Q_{\texttt{HpPar}})$ that satisfy a given absolute error tolerance remained the same as described in previous sections, with the exception of using $\tilde{e}_{\texttt{HpPar}}$ as the error metric, and incrementing $M_{\texttt{HpPar}}$ by two elements instead of one. For all solutions reported in this subsection, the number of quadrature nodes per partition were chosen to be two more than the number of collocation points per element.

Similar absolute error metrics were constructed for each of the `GLI`, `Sinc`, `HpStd`, and `HpStd4-1` methods. We found that none of these methods were able to find a set of parameters where they resolve the solution to (26) to the tolerance we use below (each was allowed to search for appropriate numerical parameters for over a day's worth of computing time). As such, we do not subsequently discuss these methods in the remainder. Note, for the `Sinc` method this result is not unexpected as the sharp gradient is near the line $t = s$, see Section 2.2.

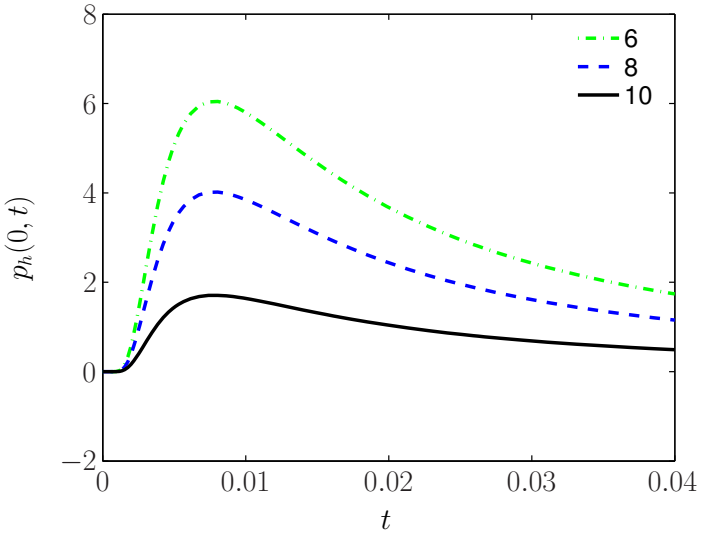

Figure 6: Numerical solutions to (26) using the `HpPar4-6` method. Note, for each curve $h = 2^{-n}$, where n is the number given in the inset. For all three solutions, $D = 1$, $k = 4\pi/1000$, and the absolute error tolerance was set to 1e-7.

	h		
	2^{-6}	2^{-8}	2^{-10}
$N_{\texttt{Greg6}}$	16384	524288	4194304
$M_{\texttt{HpPar4-6}}$, $p_{\texttt{HpPar4-6}}$, $Q_{\texttt{HpPar4-6}}$	10, 10, 21	10, 10, 21	10, 10, 26

Table 6: Numerical parameters associated with Figure 7. The number of quadrature points per partition in the `HpPar4-6` method were chosen to be two more than the number of interpolatory abscissas (*i.e.* $p + 3$). Associated physical parameters are given in Figure 6.

The numerical solutions to (26) obtained using the `HpPar4-6` method are shown in Figure 6 for $h = 2^{-6}$, 2^{-8}, and 2^{-10}. The absolute error tolerance for each solution was set to 1e-7, with $D = 1$, $k = 4\pi/1000$, and $\boldsymbol{x}_0 = (1/8, 1/8, 1/8)$. The corresponding median running times of the `HpPar4-6` and `Greg6` methods, and associated numerical parameters for each method, are shown in Figure 7 and Table 6. Grading both the element spacing near $t = 0$ and the partitions towards $t = s$ was necessary for the `HpPar` method to satisfy the specified error tolerance. For example, when just grading the element spacing our numerical search method was unable to find a set of parameters where the solutions satisfied the specified error tolerances over the course of several days of searching. (The required number of elements became so large that the calculation of the numerical solution to (26) was extremely time consuming.)

As shown in Figure 7, the `HpPar4-6` method outperforms the `Greg6` method as h is decreased. In particular, for the finest value of h, 2^{-10}, the difference between the two methods is approximately a factor of ten in median running time. Table 6 shows this arises in part as the `Greg6` method must increase the number of time-steps by a factor of 32 and then a factor of eight to satisfy the specified tolerance when h is decreased from 2^{-6} to 2^{-10}. In contrast, the numerical parameters of the `HpPar4-6` method barely change as h is decreased, with the number of elements and number of collocation points remaining constant. Given that the solution to (26) is decreasing as $h \to 0$, and apparently becoming smoother (see Figure 6), it is not surprising that the number of degrees of freedom needed to approximate the solution in the `HpPar4-6` method do not increase. A slight increase in the number of partitions is needed to resolve the finest value of h; this addition of five partitions per element corresponds to the addition of 650 quadrature points. Profiling shows that for the `HpPar4-6` method approximately 99% of the computing time was spent in the

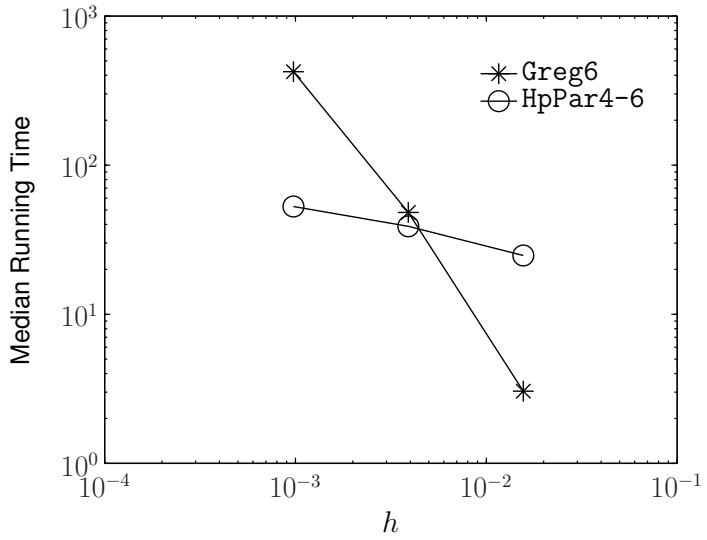

Figure 7: Median running times for the solutions obtained by the `HpPar4-6` method in Figure 6, and also by the `Greg6` method. Physical parameters and tolerance are given in Figure 6. Corresponding numerical parameters are given in Table 6.

evaluation of the kernel, $G_h(\mathbf{0}, t-s)$ for $h = 2^{-10}$. In contrast, the `Greg6` method spent approximately 39% of its running time in evaluating the kernel for $h = 2^{-10}$. This is not surprising since the `Greg6` method must increase the number of time-steps it uses to gain accuracy in either of the solution or quadrature approximations. By carefully placing the element boundaries and quadrature partitions, coupled with the use of higher order quadrature rules, the `HpPar4-6` method is able to use more than a factor of 1000 less quadrature points and 40000 less collocation points in resolving the solution to the specified tolerance.

5. Conclusions

A partitioned quadrature qualocation method was developed to improve the computational performance of the standard collocation method in solving linear Volterra integral equations with smooth kernels containing sharp gradients. The method was compared to the standard collocation approach where quadrature approximation and solution approximation are coupled. To demonstrate the benefit of decoupling the quadrature approximation from the solution approximation several additional methods where the two are coupled were also examined. We first examined the baseline performance of the methods for several equations with smooth, "well-behaved" kernels. For the examples under consideration all the methods performed quite well, with our particular implementations of the standard collocation method and the Gregory method having the fastest median running times. For smooth kernels that contained sharp gradients, we found the our partitioned quadrature qualocation method, using both non-uniformly spaced elements and partitioned Gauss quadrature rules performed the best. Only this method and the Gregory method were able to resolve all three examples we considered, and the partitioned quadrature qualocation method was generally faster with a better running time scaling as higher absolute error tolerances were considered. The benefit of decoupling the quadrature choice to evaluate the integral in (1) from numerical representation of the solution, $y(t)$, was readily apparent for these examples. The standard collocation formulation, the global spectral method, and the Sinc method, where the quadrature choices are coupled to the solution representation, were each unable to resolve all three of the smooth kernels with sharp gradient examples we considered. (The standard collocation and Sinc method did perform well on the first kernel with sharp gradient example.)

For the original Volterra equation, equation (26), containing a smooth kernel with sharp gradients that motivated this work, the qualocation-method of Section 2.1.2 substantially outperformed the Gregory convolution method. As the regularization parameter for the kernel was decreased, causing the kernel to become

sharper, the qualocation method with geometrically spaced elements and geometrically spaced quadrature partitions showed almost no increase in degrees of freedom need to represent the solution or evaluate the integrals involving the kernel. In contrast, the Gregory method need a factor of 256 more degrees of freedom to resolve the smallest value of the regularization parameter than for the coarsest value of the regularization parameter.

The MATLAB m-files we have developed for each of the methods will be made available online at [19].

[1] Arnold, D. N., Wendland, W. L., 1983. On the asymptotic convergence of collocation methods. Math. of Comp. 41 (164), 349–381.

[2] Atkinson, K. E., 1997. The Numerical Solution of Integral Equations of the Second Kind. Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press, New York, NY.

[3] Baker, C. T. H., 2000. A perspective on the numerical treatment of Volterra equations. J. Comput. Appl. Math. 125, 217–249.

[4] Bedivan, D. M., Fix, G. J., 1997. Analysis of finite element approximation and quadrature of Volterra integral equations. Numer. Methods Partial Differential Equations 13 (6), 663–672.

[5] Blom, J. G., Brunner, H., Jun 1991. Algorithm 689: Discretized collocation and iterated collocation for nonlinear Volterra integral equations of the second kind. ACM Trans. Math. Software 17 (2), 167–177.

[6] Brunner, H., 1985. The numerical solution of weakly singular Volterra integral equations by collocation on graded meshes. Math. Comp. 45 (172), 417–437.

[7] Brunner, H., 1986. Polynomial spline collocation methods for Volterra integro-differential equations with weakly singular kernels. IMA J. Numer. Anal. 6 (2), 221–239.

[8] Brunner, H., 2004. Collocation Methods for Volterra Integral and Related Functional Equations. Cambridge University Press, New York.

[9] Brunner, H., Lin, Y., Zhang, S., 1998. Higher accuracy methods for second-kind Volterra integral equations based on asymptotic expansions of iterated galerkin methods. J. Integral Equations Appl. 10 (4), 375–396.

[10] Canuto, C., Hussaini, M. Y., Quarteroni, A., Zang, T. A., 1987. Spectral Methods in Fluid Dynamics. Springer-Verlag, New York.

[11] Chandler, G. A., Sloan, I. H., 1990. Spline qualocation methods for boundary integral equations. Numer. Math. 58, 537–567.

[12] Delves, L. M., Mohamed, J. L., 1985. Computational Methods for Integral Equations. Cambridge University Press, Cambridge.

[13] Eddins, S., 2008. `timeit.m` available at `http://www.mathworks.com/matlabcentral/fileexchange`.

[14] Hagen, R., Silbermann, B., 1987. Operator equations and numerical analysis. Seminar Analysis. Karl-Weierstraß-Institut für Mathematik, Berlin, Ch. On stability of the qualocation method., pp. 43–52.

[15] Hairer, E., Lubich, C., Schlichte, M., 1985. Fast numerical solution of nonlinear Volterra convolution equations. SIAM J. Sci. Stat. Comput. 6 (3), 531–541.

[16] Hesthaven, J. S., Gottlieb, S., Gottlieb, D., 2007. Spectral Methods for Time-Dependent Problems. No. 21 in Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press, Cambridge UK.

[17] Isaacson, S. A., 2009. The reaction-diffusion master equation as an asymptotic approximation of diffusion to a small target. SIAM J. Appl. Math. 70 (1), 77–111.

[18] Isaacson, S. A., Isaacson, D., 2009. Reaction-diffusion master equation, diffusion-limited reactions, and singular potentials. Phys. Rev. E 80 (6), 066106 (9pp).

[19] Isaacson, S. A., Kirby, R., 2008. MATLAB routines for the numerical solution of linear Volterra equations of the second kind. Available at `http://www.math.bu.edu/people/isaacson`.

[20] Isaacson, S. A., Peskin, C. S., 2006. Incorporating diffusion in complex geometries into stochastic chemical kinetics simulations. SIAM J. Sci. Comput. 27 (1), 47–74.

[21] Karniadakis, G., Sherwin, S., 2005. Spectral/hp element methods for computational fluid dynamics. Oxford University Press, Oxford UK.

[22] López-Fernández, M., Lubich, C., Schädle, A., 2008. Adaptive, fast, and oblivious convolution in evolution equations with memory. SIAM J. Sci. Comput. 30 (2), 1015–1037.

[23] Luther, H. A., Apr. 1968. An explicit sixth-order Runge-Kutta formula. Mathematics of Computation 22 (102), 434–436.

[24] Mori, M., Sugihara, M., 2001. The double-exponential transformation in numerical analysis. J. Comput. Appl. Math. 127, 287–296.

[25] Muhammad, M., Nurmuhammad, A., Mori, M., Sugihara, M., 2005. Numerical solution of integral equations by means of the sinc collocation method based on the double exponential transformation. J. Comput. Appl. Math. 177, 269–286.

[26] Rosa, L., 2004. `fftconv.m` available at `http://www.mathworks.com/matlabcentral/fileexchange`.

[27] Sharp, P. W., Verner, J. H., 2000. Extended explicit Bel'Tyukov pairs of orders 4 and 5 for Volterra integral equations of the second kind. Appl. Numer. Math. 34 (2-3), 261–274.

[28] Sharp, P. W., Verner, J. H., 2000. Some extended explicit Bel'Tyukov pairs for Volterra integral equations of the second kind. SIAM J. Numer. Anal. 38 (2), 347–359 (electronic).

[29] Shaw, S., Whiteman, J. R., 2000. Numerical solution of linear quasistatic hereditary viscoelasticity problems. SIAM J. Numer. Anal. 38 (1), 80–97.

[30] Sloan, I. H., 1988. A quadrature-based approach to improving the collocation method. Numer. Math. 54, 41–56.

[31] Sloan, I. H., 2000. Qualocation. J. Comput. Appl. Math. 125, 461–478.

[32] Sloan, I. H., Wendland, W. L., 1989. A quadrature-based approach to improving the collocation method for splines of even degree. Zeit. f. Anal. u. ihre Anw. 8, 361–376.
[33] Tang, T., 1993. A note on collocation methods for Volterra integro-differential equations with weakly singular kernels. IMA J. Numer. Anal. 13 (1), 93–99.
[34] Tang, T., Xu, X., Cheng, J., 2008. On spectral methods for Volterra type integral equations and the convergence analysis. J. Comp. Math. 26 (6), 825–837.
[35] Trefethen, L. N., Jan 2008. Is Gauss quadrature better than Clenshaw-Curtis? SIAM Rev. 50 (1), 67–87.
[36] von Winckel, G., 2004. `lglnodes.m` available at `http://www.mathworks.com/matlabcentral/fileexchange`.