Journal of Computational and Applied Mathematics 310 (2017) 213–223

Contents lists available at ScienceDirect

Journal of Computational and Applied Mathematics

journal homepage: www.elsevier.com/locate/cam

# Analytical Jacobian-vector products for the matrix-free time integration of partial differential equations

Paul Tranquilli *, S. Ross Glandon, Arash Sarshar, Adrian Sandu

*Computational Science Laboratory, Department of Computer Science, Virginia Tech. Blacksburg, VA 24060, United States*

ARTICLE INFO

*Article history:*
Received 10 February 2016
Received in revised form 1 May 2016

*Keywords:*
Matrix-free
Implicit time integration
Jacobian-vector products
Numerical methods for PDEs
Numerical methods for ODEs

ABSTRACT

Many scientific and engineering applications require the solution of large systems of initial value problems arising from method of lines discretization of partial differential equations. For systems with widely varying time scales, or with complex physical dynamics, implicit time integration schemes are preferred due to their superior stability properties. These schemes solve at each step linear systems with matrices formed using the Jacobian of the right hand side function. For large applications iterative linear algebra methods, which make use of Jacobian-vector products, are employed. This paper studies the impact that the method of computing Jacobian-vector products has on the overall performance and accuracy of the time integration process. The analysis shows that the most beneficial approach is the direct computation of exact Jacobian-vector products in the context of matrix-free time integrators. This approach does not suffer from approximation errors, reuses the parallelism and data distribution already present in the right-hand side vector computations, and avoids storing or operating on the entire Jacobian matrix.



## 1. Introduction

The method of lines discretization of partial differential equations leads to initial value problems of the form:

$$\frac{dy}{dt} = F(t, y), \quad t_0 \le t \le t_F, \; y(t_0) = y_0; \; y(t), F(t, y) \in \mathbb{R}^N. \tag{1}$$

Existing numerical methods to solve (1) are either explicit, where future timesteps are dependent only on information at the current and past timesteps, or implicit methods, where future timesteps are dependent on information from both the current and future timesteps.

Explicit methods are relatively easy to analyze, construct, and implement. While the cost per timestep is minimal, they may suffer from stability issues for problems that exhibit stiff behavior. This means that while each timestep is cheap, a very large number of timesteps may be required to achieve a suitable solution.

While implicit methods are more complicated to analyze and implement they, ideally, do not suffer from as stringent of stability considerations as explicit methods. The flipside of implicit method's improved stability properties is the need to solve one or more linear, or non-linear, equations at each timestep.

The solution of these linear, or non-linear, equations requires the construction, and solution, of a linear system containing the Jacobian, $\mathbf{J}_n$, arising from (1)

$$\mathbf{J}_n = \left. \frac{\partial F(t, y)}{\partial y} \right|_{t=t_n, \, y=y_n}.$$

* Corresponding author.
*E-mail addresses:* ptranq@vt.edu (P. Tranquilli), rossg42@vt.edu (S.R. Glandon), sarshar@vt.edu (A. Sarshar), sandu@cs.vt.edu (A. Sandu).

http://dx.doi.org/10.1016/j.cam.2016.05.002
0377-0427/

For ordinary differential equation systems which naturally appear in the form of (1) this requirement may not be incredibly strenuous; however, for problems arising from a method of lines discretization of a partial differential equation system, such as earth system simulations, the dimension, $N$, of the system can be very large.

In the case of such systems it may be computationally infeasible to construct, or store, even a sparse representation of this system Jacobian, $\mathbf{J}_n$. Moreover, when the dimension of the system is large direct solution methods such as $LU$-decomposition may be computationally unstable, or lead to dense $\mathbf{L}$ and $\mathbf{U}$ matrices which pose an even greater storage problem than that with which we began.

This leads to a critical problem in the time evolution of such systems: explicit methods are a poor choice due to the need to take an extremely large number of timesteps to resolve the system dynamics, and implicit methods are similarly a poor choice due to an inability to deal with the very large, and costly to compute, Jacobian matrix. The solution to our problem lies in so called matrix-free methods, or methods which require only the action of the Jacobian on a vector and not the construction of the full Jacobian matrix.

The most common approach for constructing a matrix-free method is to simply replace any direct solution of linear systems with an iterative linear system solve such as GMRES. Applying this to a standard singly diagonally implicit Runge–Kutta (SDIRK) [1] method (2)

$$k_i = hF\left(t_n + c_i h, y_n + h\sum_{j=1}^{i} a_{i,j}k_j\right) \tag{2a}$$

$$y_{n+1} = y_n + \sum_{i=1}^{s} b_i k_i, \tag{2b}$$

leads to so called Jacobian-free Newton–Krylov methods [2], in which the nonlinear system in the stage Eq. (2a) is solved using a Newton iteration, and the linear solves at each iteration are solved using GMRES.

Alternatively, if we wish to avoid the solution of a non-linear system altogether, we can make use of a Rosenbrock [1] type method (3)

$$k_i = hF\left(t_n + c_i h, y_n + \sum_{j=1}^{i-1} \alpha_{i,j}k_j\right) + h\mathbf{A}_n \sum_{j=1}^{i} \gamma_{i,j}k_j + h^2\gamma_i \frac{\partial F}{\partial t}(t_n, y_n) \tag{3a}$$

$$y_{n+1} = y_n + \sum_{i=1}^{s} b_i k_i \tag{3b}$$

where $\mathbf{A}_n = \mathbf{J}_n$ for a classical Rosenbrock (ROS) method, or is an arbitrary approximation of $\mathbf{J}_n$ in the case of Rosenbrock-W (ROW) methods [1]. For more information on specific variations of Rosenbrock schemes see [3–5]. We also consider in Section 4 a relatively new family of Rosenbrock methods, called Rosenbrock–Krylov (ROK) [6], which treats $\mathbf{A}_n$ as a low rank Krylov based approximation of the Jacobian matrix.

The rest of the paper is laid out as follows: in Section 2 we describe several methods for computing Jacobian-vector products, as well as the advantages and disadvantages of each approach; in Section 3 numerical results illustrating the similar parallel scalability of right hand side (RHS) and direct Jacobian-vector computations in a shared memory setting are presented; in Section 4 we examine the interplay and effects of different Jacobian strategies and method choice on the accuracy and efficiency of the overall time integration process; finally, in Section 5 we give concluding remarks and a summation of the important points made throughout the manuscript.

## 2. Jacobian-vector products

There are several methods of computing, or approximating, Jacobian-vector products. The most common approach throughout the literature is interpreting this product as a directional derivative. The product is then approximated using a finite difference calculation so that

$$\mathbf{J}_n v \approx \frac{F(t_n, y_n + \varepsilon v) - F(t_n, y_n)}{\varepsilon}, \tag{4}$$

where $\varepsilon$ is chosen based on machine precision and $\|F(t_n, y_n)\|$ to minimize rounding and truncation errors of the finite difference approximation (4). A thorough discussion on the selection of $\varepsilon$ is given in [2].

This approach has the benefit of being computationally efficient. It has a similar cost to that of a single right-hand side computation since $F(t_n, y_n)$ is generally required for the method anyway. Similarly, it has the benefit of preserving any parallel features of the right-hand side computation.

Unfortunately, the finite difference approach has the downside of being inexact. The implications of this inexactness are different for each of the different methods: in the case of an SDIRK method (2) this can lead to convergence problems in the GMRES algorithm causing a significant increase in the number of Newton iterations required to solve the non-linear system, whereas for Rosenbrock methods in which the Jacobian matrix, $\mathbf{J}_n$, appears explicitly inexactness can lead to a dramatic loss

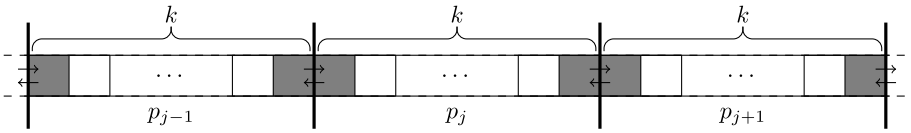



**Fig. 1.** An example parallel data decomposition of a 1D state vector on processors $p_{j-1}$, $p_j$, and $p_{j+1}$, with ghost cells requiring communication shaded. A total of $k$ elements are placed on each processor.

in the order of temporal convergence. The Rosenbrock-W and Rosenbrock–Krylov methods briefly introduced in Section 1, correct this behavior but may face other issues which are discussed more thoroughly in Section 4.

The naive, and most obvious, solution to problems caused by the inexactness of a finite difference approximation of the Jacobian-vector product is to explicitly construct the Jacobian matrix, and then compute products in the standard way when they are required. While this appears to run into the same storage and stability issues that plague standard implicit methods, the use of an iterative solver avoids the need to construct, and store, an *LU* decomposition as well as not encountering the accumulation of round off errors that causes the stability issues of direct methods for very large systems.

The true cost of explicitly constructing the Jacobian matrix, is the computational inefficiency of doing so. While the freedom from the need to store a (possibly full) *LU*-decomposition reduces some storage constraints, even a sparse representation of the Jacobian matrix may be too large to store for very large problems. Even worse for large problems, is the inability to exploit the built-in parallelism of the right-hand side; the distribution of data across many nodes that is beneficial to efficiently computing the right-hand side, $f(y_n)$, may not be as computationally advantageous for computing its derivative, $\mathbf{J}_n$.

Once again, we find an advantageous alternative to the common approaches: directly computing exact Jacobian-vector products. This approach, because it is exact, avoids the slow convergence of Newton iterations for SDIRK and the order reduction of Rosenbrock methods when using an inaccurate finite difference approximation. The basis of this approach comes from Eq. (5).

$$(\mathbf{J}_n v)_i = \sum_{j=1}^{N} \left.\frac{\partial F_i}{\partial y_j}\right|_{t=t_n, y=y_n} v_j. \tag{5}$$

Most importantly computing the product in this way allows a reuse of the parallelism and data distribution constructed for computing the right-hand side vector, $f(y_n)$. We briefly illustrate this idea by applying it to a straight forward example, the one-dimensional, diffusive Burger's equation (6).

$$\frac{\partial u}{\partial t} = \frac{\partial}{\partial x}\left(\frac{1}{2}u^2\right) + \varepsilon \frac{\partial^2 u}{\partial x^2}. \tag{6}$$

With the semi-discrete representation given in Eq. (7), using centered finite differences to approximate spatial derivatives

$$\left(\frac{du}{dt}\right)_i = \frac{1}{4\Delta x}\left(u_{i+1}^2 - u_{i-1}^2\right) + \frac{\varepsilon}{\Delta x^2}\left(u_{i+1} - 2u_i + u_{i-1}\right). \tag{7}$$

Making use of Eq. (5), we see that Jacobian-vector products can be computed as

$$(\mathbf{J}_n v)_i = \frac{1}{2\Delta x}\left(u_{i+1}v_{i+1} - u_{i-1}v_{i-1}\right) + \frac{\varepsilon}{\Delta x^2}\left(v_{i+1} - 2v_i + v_{i-1}\right). \tag{8}$$

From Eq. (8) it is clear that the vector $v$ can be distributed in memory in the same way that the state vector is distributed, since the Jacobian-vector product, $(\mathbf{J}_n v)$, has the same data dependencies on $u$ and $v$ as a right-hand side evaluation does on the state vector. Thus, any distributed memory parallelization scheme applied to the right-hand side computation can be reused in the Jacobian-vector product, as both operate on the same stencil. Additionally, both computations can share a memory access pattern, and thus, any shared-memory threading or GPU based parallelism. For example, Fig. 1 illustrates a method of placing the components of $u$ or $v$ across processors on a distributed system, including ghost cells to support the 3-point stencil used in (7) and (8). Direct computation of the Jacobian-vector product as in (5) can be implemented manually along with the method of lines discretization, or can be constructed using the tangent linear mode of an automatic differentiation tool, such as `TAPENADE` [7], `TAMC` [8], or `ADIFOR` [9].

## 3. Scalability experiments with the shallow water equations

In order to demonstrate some of the parallel properties of computing direct Jacobian-vector products, this section presents scalability results from an example model. Using a strong scaling experiment, we compare right-hand side evaluations and direct Jacobian-vector products, where the direct Jacobian-vector product computation was written to reuse the parallelization scheme of the model right-hand side. Analysis is performed by examining the timing data from the experiment, as well as the associated parallel speedups of both computations.

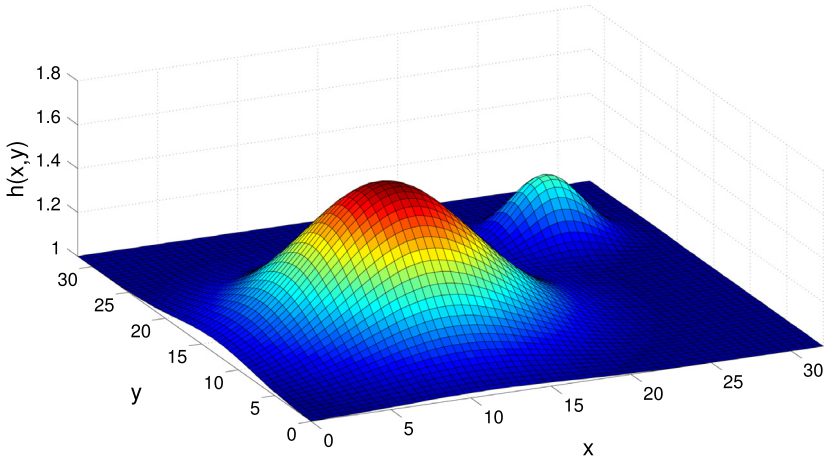



**Fig. 2.** Shallow water equations initial condition of the height components, $h(0, x, y)$ used for subsequent scalability experiments. All velocity components were set to zero.

The chosen model is a spatial discretization of the shallow water equations [10],

$$\frac{\partial}{\partial t}h + \frac{\partial}{\partial x}(uh) + \frac{\partial}{\partial y}(vh) = 0 \tag{9a}$$

$$\frac{\partial}{\partial t}(uh) + \frac{\partial}{\partial x}\left(u^2h + \frac{1}{2}gh^2\right) + \frac{\partial}{\partial y}(uvh) = 0 \tag{9b}$$

$$\frac{\partial}{\partial t}(vh) + \frac{\partial}{\partial x}(uvh) + \frac{\partial}{\partial y}\left(v^2h + \frac{1}{2}gh^2\right) = 0, \tag{9c}$$

with reflective boundary conditions, where $u(x, y, t)$, $v(x, y, t)$ are the flow velocity components and $h(x, y, t)$ is the fluid height. The spatial discretization's right-hand side is implemented using centered finite differences on an $N_x \times N_y$ grid, and the system (9) is brought to the standard ODE form (1) with

$$y = [u\ v\ h]^T \in \mathbb{R}^N, \qquad f_y(t, y) = \mathbf{J} \in \mathbb{R}^{N\times N}, \qquad N = 3 \times N_x \times N_y,$$

and parallelized using `OpenMP` [11]. Direct Jacobian-vector products are computed using code produced by the tangent linear mode of `TAPENADE` applied to the right-hand side to which `OpenMP` directives were manually re-added.

It is important to note that the re-addition of directives only involved copying from the right-hand side implementation after manual verification of the machine-generated Jacobian-vector product code. For these experiments, no time integrator is used, only the parallel computation of the right-hand side, as the primary expense of performing a finite difference, and direct Jacobian-vector products were performed and timed; for an analysis of the right-hand side scalability with the time integrator included, see [12].

All scalability experiments were performed on a dual-socket workstation with 128GB of memory. Both CPUs are Intel Xeon E5-2650v3 processors operating at 2.30 GHz with 10 physical cores, 25MB of last level cache, and Hyperthreading. Hyperthreading remained on for all experiments, resulting in a total of 20 physical cores and 20 virtual cores available for the experiments.

Strong scaling experiments were performed on four sizes of square grids:

$$N_x = N_y \in \{128,\ 256,\ 512,\ 1024\},$$

with the corresponding number of variables ranging from about 50 thousand to just over 3 million. Because of the relatively small size of the problems, for each size, the right-hand side and direct Jacobian-vector product were each computed 10,000 times in a loop and a CPU timer was used to measure the total time; average evaluation times were then computed from this total. For each test, the model (9) was initialized with the initial condition given in Fig. 2, scaled to the appropriate grid size.

The average times measured for each experiment are presented in Fig. 3, showing that the direct Jacobian-vector product computation requires only some small, essentially constant, multiple of the time taken for a RHS evaluation. By normalizing each set of times by the time for 1 core, we can more effectively see the similar scaling for both right-hand side evaluation and direct Jacobian-vector products, this parallel speedup graph is presented in Fig. 4. We see that the smallest grid size, $128 \times 128$, fails to gain any benefit from additional cores past 8, as expected for a small problem in a strong scaling test. Additionally, no problem size sees improvement with more than 20 cores, as we expect with only 20 physical cores. Looking at the speedup graph in Fig. 4, we can see that as problem size increases from $128 \times 128$ to $512 \times 512$, the speedup gets incrementally better, however, the $1024 \times 1024$ size shows reduced scaling. This reduction can be explained by considering the size in cache of the vectors computed by the model: the $512 \times 512$ size requires almost 800,000 8 byte floating point

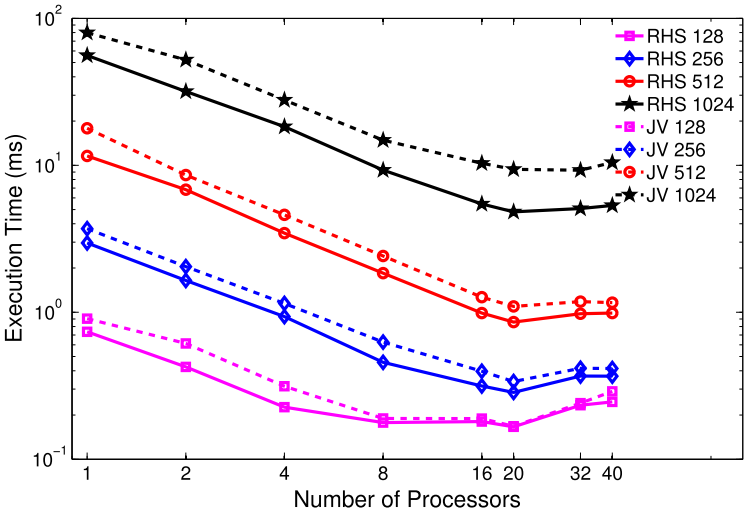



**Fig. 3.** Average right-hand side (RHS) and direct Jacobian-vector product (JV) evaluation times over 10,000 iterations of the shallow water equations model. Broken lines correspond to direct Jacobian-vector products. The compute time of a Jacobian-vector product is only slightly more expensive than the computation of the right-hand side, and the scalability is preserved.

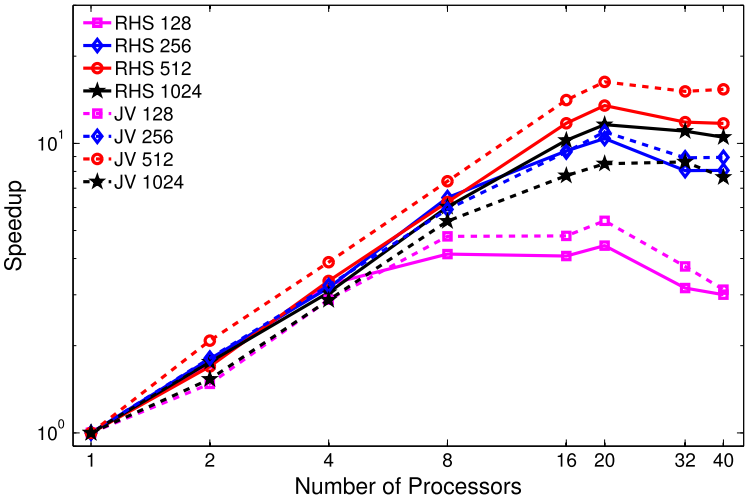


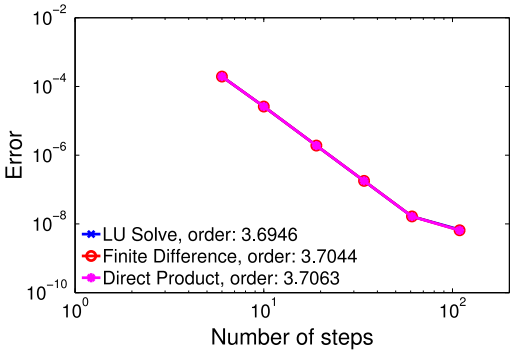

**Fig. 4.** Parallel speedup of the right-hand side (RHS) and direct Jacobian-vector product (JV) computations averaged over 10,000 iterations of the shallow water equations model. Broken lines correspond to direct Jacobian-vector products. Right-hand side and Jacobian-vector products show similar scaling behavior at corresponding problem sizes.

variables, coming to 6 MB. So, the 25 MB last-level cache can contain 3 or even 4 full-sized state vectors, whereas the $1024 \times 1024$ grid size requires 4 times the memory at 24 MB per vector, and must make further use of main memory during the computation. So, all of the smaller problems demonstrate the similar scalability of the right-hand side and Jacobian-vector product in the ideal case, when all calculations are performed in cache. Still, the speedup of both computations for the larger problem sees similar scaling reductions, continuing the trend of similar behavior. This effectively demonstrates the reliance of both implementations on the memory access pattern and data distribution they share.

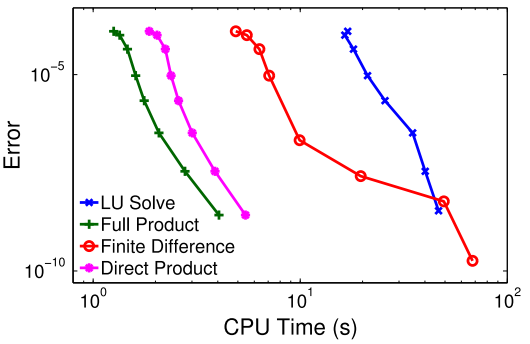
These results clearly demonstrate the natural scalability and minimal overhead of computing direct Jacobian-vector products as compared to evaluating the right-hand side, providing empirical evidence that this strategy is a viable replacement for computing finite difference approximations of the product, with only a small penalty in absolute performance and no degradation in parallel scalability. Most importantly, the parallelization scheme and other optimizations created for the model right-hand side can be directly reused when computing direct Jacobian-vector products, potentially simplifying implementation dramatically.

## 4. Impact of Jacobian-vector formulation on numerical solution accuracy

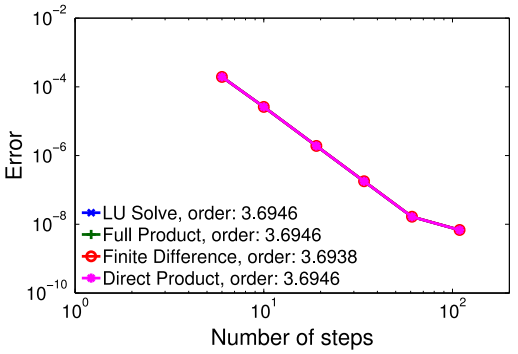
Here we present the complicated interplay between the choice of time integrator and Jacobian approach on the convergence and efficiency of the overall time integration strategy. We consider the two-dimensional Allen–Cahn problem,



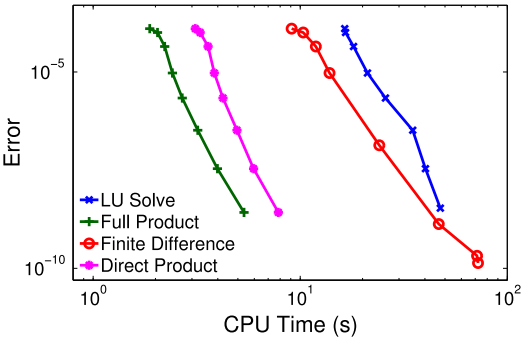

(a) Fixed stepsize, GMRES tolerance $1e-5$.




(b) Adaptive stepsize, GMRES tolerance $1e-5$.




(c) Fixed stepsize, GMRES tolerance $1e-8$.




(d) Adaptive stepsize, GMRES tolerance $1e-8$.

**Fig. 5.** Work-precision diagrams for SDIRK solutions of Eq. (10) on a $64 \times 64$ grid. Different Jacobian-vector computation strategies are applied. The Newton tolerance is $1e-7$ for all cases; the GMRES tolerances are varied. Matrix free methods outperform $LU$, with strategies employing exact Jacobian-vector information being most efficient. It is notable that Fig. 5(a) and (c) contain multiple overlapping lines.

a reaction–diffusion parabolic PDE used in materials science to study the evolution of phase boundaries in crystalline solids [13] defined by:

$$\frac{\partial u}{\partial t} = \alpha \nabla^2 u + \gamma \left(u - u^3\right), \quad u = [x, y]^T \in [0, 1] \times [0, 1], \; t \in [0, 1] \tag{10}$$

with parameters $\alpha = 0.01$ and $\gamma = 1.0$ subject to homogeneous Neumann boundary conditions and initial condition $u(t = 0) = 0.4 + 0.1(x + y) + 0.1 \sin(10x) \sin(20y)$. The spatial discretization used in the following experiments is a two-point central finite difference scheme applied to the Laplacian operator and the right-hand side of Eq. (10) on a rectangular grid of 64 by 64 nodes. We illustrate on four different time integrators: SDIRK, Rosenbrock, Rosenbrock-W, and Rosenbrock-K, the effects of four distinct Jacobian strategies:

- **LU** *Solve* — In this approach the full Jacobian is evaluated at each timestep, and all linear systems arising in the method are solved using a direct solve based on $LU$-decomposition.
- *Full Product* — In this approach the full Jacobian is evaluated at each timestep, and all linear systems arising in the method are solved using GMRES, with Jacobian-vector products evaluated through a matrix–vector multiplication of the already computed Jacobian.
- *Finite Difference* — In this approach the full Jacobian matrix is never computed, all linear systems arising in the method are solved using GMRES, with Jacobian-vector products approximated using Eq. (4).
- *Direct Product* — In this approach the full Jacobian matrix is never computed, all linear systems arising in the method are solved using GMRES, with Jacobian-vector products computed exactly as in Eq. (5).

Fixed timestepping is used to find convergence order and to detect any order reduction resulting from the approximations made, while adaptive timestepping is used to demonstrate the computational efficiency of each approach. Adaptive timestepping is performed by comparing, at each timestep, the result of the integrator with the result from an embedded lower order method. When the difference is too great, the step is rejected and retried with a reduced stepsize; similarly, if the difference is very small, the integrator increases the stepsize for the next timestep.

We first examine an SDIRK method of order four, using a Newton iteration to solve each nonlinear stage Eq. (2a), and GMRES to solve each linear system arising within these iterations. Fig. 5(a) and (c) display the benefit of a Newton iteration on the convergence of the method with all Jacobian strategies showing full order up to accuracies slightly below the tolerance of

**Table 1**
SDIRK with Relative tolerance $1e-7$, GMRES tolerance $1e-5$, and Newton tolerance $1e-7$.

| | RHS | Jacobian-vector | Accepted steps | Newton iterations |
|---|---|---|---|---|
| LU solve | 478 | | 36 | 514 |
| Full product | 470 | 12,814 | 36 | 506 |
| Finite difference | 2069 | 48,326 | 113 | 2069 |
| Direct product | 470 | 12,814 | 36 | 506 |

**Table 2**
Performance of ROS with relative tolerance $1e-7$ and GMRES tolerance $1e-5$.

| | RHS | Jacobian-vector | Accepted steps | Linear solves |
|---|---|---|---|---|
| LU solve | 147 | | 49 | 196 |
| Full product | 150 | 4661 | 50 | 200 |
| Finite difference | 150 | 4666 | 50 | 200 |
| Direct product | 150 | 4662 | 50 | 200 |

the Newton method. The use of inaccurate linear solutions caused by a GMRES iteration using either a loose error tolerance or approximate Jacobian-vector products leads to a quasi-Newton method that is likely to converge outside of pathological scenarios.

Fig. 5(b) and (d) tell a slightly different story. The difference between more or less restrictive GMRES tolerances is not tremendous when using exact Jacobian-vector products; however, the impact on efficiency when using a finite difference approximation is profound. Table 1 illustrates the cause of the dramatic decrease in performance when the computation of a suitably accurate GMRES solution is hampered by the use of approximate Jacobian information. We see that use of a finite difference leads to a significant increase in the number of required Newton iterations and therefore in the number of Jacobian-vector products and right hand side evaluations, resulting in a computational cost approaching that of the *LU* solution strategy.

Alternatively, it is clear that there is a decided advantage to making use of a matrix-free approach when exact Jacobian-vector products are available. In Fig. 5(b) and (d) the full and direct products are substantially faster than either *LU* or finite differences. Interestingly, the strategy employing an evaluation of the entire Jacobian at each timestep and then computing products outperforms directly computing this product. This behavior is easily explained by considering that the example presented here is run in `MATLAB` in which matrix vector products are computed using compiled code, while the direct computation of the Jacobian-vector products is computed as an interpreted `MATLAB` function and so will be inherently less efficient. Considering these facts in tandem leads to the conclusion that there exists a number of Jacobian-vector products after which it is cheaper to compute the matrix and then multiply. Due to the Newton iteration, SDIRK methods make use of a relatively large number of Jacobian-vector products at each timestep, making it unsurprising that the full product outperforms the direct product for this test problem.

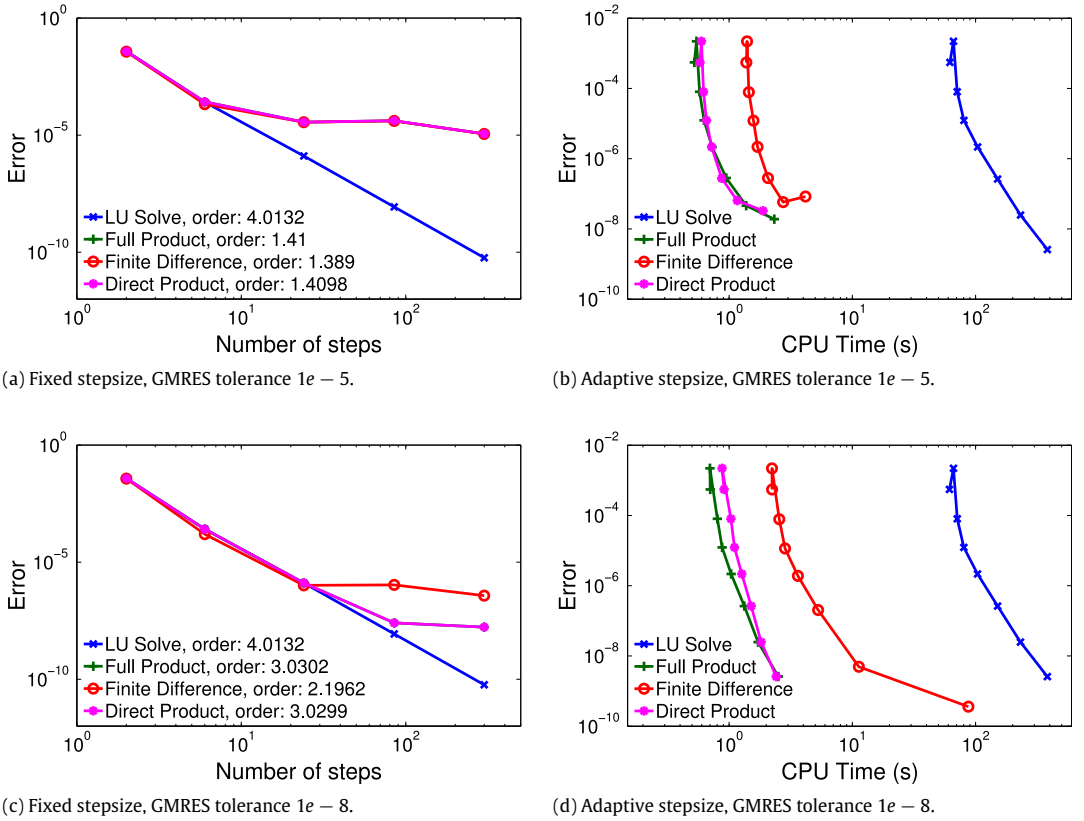
**Remark 1.** It is important to note that in more sophisticated applications in which both exact Jacobian-vector product strategies are on even footing, and where parallelism is essential, the break even point between the full and direct approaches is likely to be much greater than seen here, particularly considering the ability to reuse both memory distribution and access patterns for direct products as discussed in Section 3. With this in mind we expect that direct Jacobian-vector products are generally better suited for large scale applications.

We now examine a Rosenbrock (3) scheme in hopes of reducing the impact of inexact Jacobian-vector products through a linearization of (2) replacing the nonlinear solves in (2a) with the more computationally convenient linear system solves in (3a) and thus eliminating the need for a Newton iteration entirely. Fig. 6(b) and (d) show the benefit of avoiding the Newton iteration, with the reduced number of Jacobian-vector products required at each timestep (as seen in Table 2) leading to a dramatic improvement in the efficiency of the finite difference approximation and direct products relative to the *LU* solve and full products, respectively.

An unfortunate consequence of eliminating the Newton iteration is the restriction placed on the accuracy and convergence of the method by the quality of the obtainable linear system solutions. When the GMRES tolerance is loosened as in Fig. 6(a), or when finite differences are not sufficient to allow for accurate GMRES solutions as in Fig. 6(c), the convergence stagnates.

Also of note in Table 2 is the slight difference in the number of Jacobian-vector products performed by the Full and Direct Product approaches (and differences in the number of steps and linear solves in Table 3). While these approaches are mathematically equivalent, we can expect slight differences to appear with the use of finite precision arithmetic because of the significant difference in the order of operations.

Rosenbrock-W (ROW) methods have the potential to minimize the impact of inexact Jacobian strategies. These methods are based on the same general form as standard Rosenbrock methods (3), but make use of an order condition framework that guarantees full order of convergence when using an arbitrary approximation of the Jacobian matrix. Unfortunately, the cost of such a framework is a dramatic escalation in the number of order conditions as the order of the method is increased. For this reason methods of order four are not feasible, so a third order method derived in [14] is used here.




(a) Fixed stepsize, GMRES tolerance $1e-5$.

(b) Adaptive stepsize, GMRES tolerance $1e-5$.

(c) Fixed stepsize, GMRES tolerance $1e-8$.

(d) Adaptive stepsize, GMRES tolerance $1e-8$.

**Fig. 6.** Work-precision diagrams for ROS solutions for Eq. (10) on a $64 \times 64$ grid. Different Jacobian-vector computation strategies are applied. The GMRES tolerances are varied. Matrix free methods dramatically outperform $LU$, and convergence rate is influenced heavily by the quality of the linear system solution. It is notable that some of the plots contain overlapping data that may not be visible.

**Table 3**
Performance of ROW with relative tolerance $1e-7$ and GMRES tolerance $1e-5$.

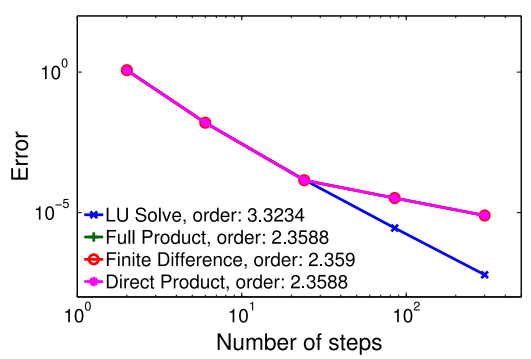
| | RHS | Jacobian-vector | Accepted steps | Linear solutions |
|---|---|---|---|---|
| LU solve | 308 | | 77 | 308 |
| Full product | 308 | 6916 | 77 | 308 |
| Finite difference | 308 | 6917 | 77 | 308 |
| Direct product | 312 | 6997 | 78 | 312 |

Fig. 7(a) and (c) show some improvement in convergence properties compared to Rosenbrock methods, but still suffer from order reduction when very inaccurate linear system solutions are used. This is because the independent GMRES solutions of each stage equation (3a) are in principle solving linear systems containing $s$ different, though "nearby", approximate Jacobian matrices, a fact not reconciled by the use of the more sophisticated order condition theory. We do, however, see an improvement in obtaining very accurate solutions relative to Rosenbrock methods as seen in Fig. 7(b) and (d).

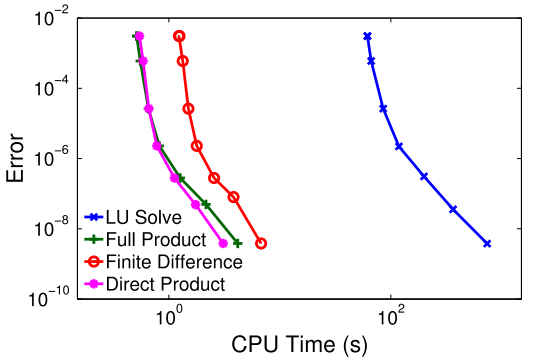
In response to the convergence issues encountered in Rosenbrock and Rosenbrock-W methods we examine Rosenbrock–Krylov (ROK) methods. In contrast to Rosenbrock-W methods which permit arbitrary approximations of the Jacobian matrix, Rosenbrock–Krylov methods make use of a specific Krylov based approximation of the Jacobian, dramatically reducing the number of order conditions required, and making fourth order schemes possible; a full derivation and discussion of K-methods is given in [6,15].

Rosenbrock–Krylov methods are based on a reduced space form of the standard Rosenbrock method (3) which considers the time integrator and linear system solves as a single computational process, and eliminates the need for accurate linear system solutions. This effect can be seen clearly in Fig. 8(b), in which the matrix-free approaches, with both accurate and approximate Jacobian-vector products, outperform the full product strategy which requires the computation of the full Jacobian matrix. Additionally, because each stage equation is solved in the same Krylov subspace the negative repercussions of inexact Jacobian-vector products observed on Rosenbrock and Rosenbrock-W methods, are eliminated entirely in the case of Rosenbrock–Krylov schemes, as can be seen in Fig. 8(a).

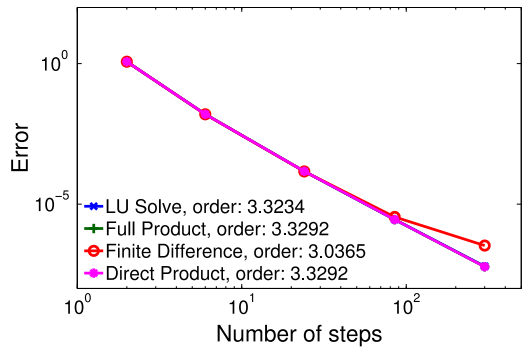
Finally, we conclude our numerical illustration of the choice of Jacobian strategy on the various time integration schemes by examining Fig. 9 which compares the most efficient choice for each of the various time integrators discussed. We note that



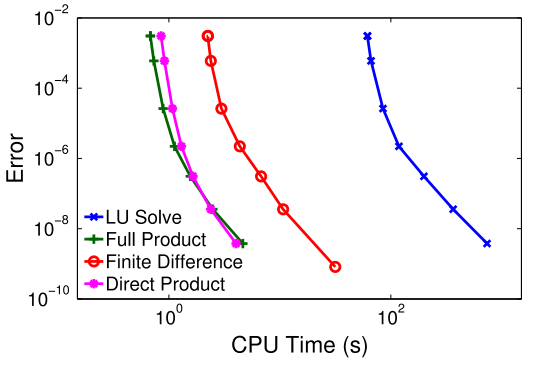

(a) Fixed stepsize, GMRES tolerance $1e-5$.




(b) Adaptive stepsize, GMRES tolerance $1e-5$.

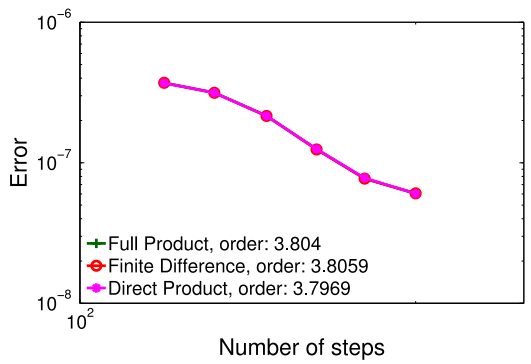



(c) Fixed stepsize, GMRES tolerance $1e-8$.

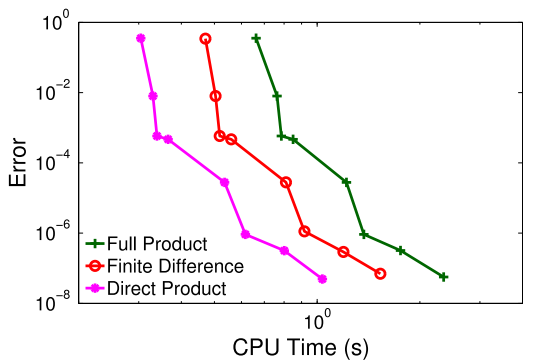



(d) Adaptive stepsize, GMRES tolerance $1e-8$.

**Fig. 7.** Work-precision diagrams for ROW solutions for Eq. (10) on a $64 \times 64$ grid. Different Jacobian-vector computation strategies are applied. The GMRES tolerances are varied. Relative efficiency of Jacobian-vector strategies is similar to ROS, with an improved convergence behavior with low accuracy linear system solutions. It is notable that some of the plots contain overlapping data that may not be visible.




(a) Fixed stepsize.




(b) Adaptive stepsize.

**Fig. 8.** Work-precision diagrams for ROK solutions for Eq. (10) on a $64 \times 64$ grid. Different Jacobian-vector computation strategies are applied. The method is inherently matrix free and convergence rate is unaffected by inexact Jacobian-vector products. It is notable that Fig. 8(a) contains multiple overlapping lines.

even for the relatively small test problem discussed here, a matrix-free approach outperforms *LU* in all cases, and similarly exact Jacobian-vector products outperform the finite difference approximation of these products.

## 5. Conclusions

The implicit time discretization of differential equations requires the solution of several linear, or non-linear, systems at each timestep. For large parallel applications direct solution methods based on *LU* decompositions are impractical due to their large computational and memory costs. For these applications iterative solution methods, like GMRES, are the preferred alternative to direct solution strategies.

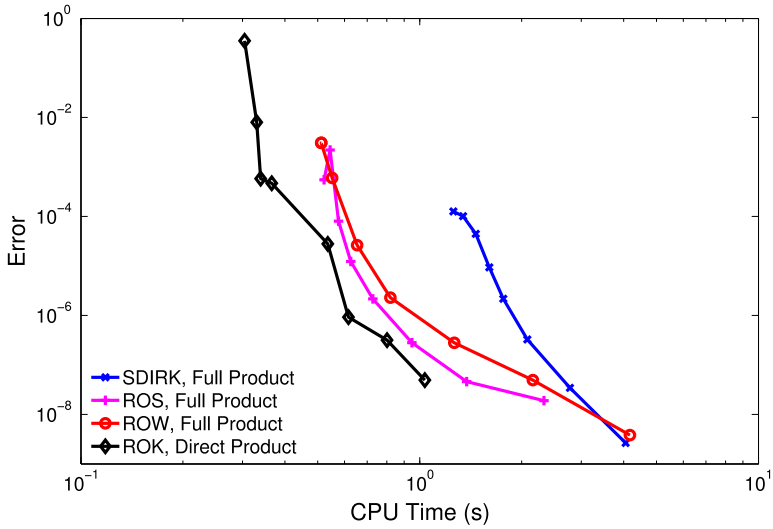



**Fig. 9.** Work-precision performance comparison of the fastest Jacobian-vector strategy for each time integration scheme. We see that matrix-free methods employing exact Jacobian-vector products outperform *LU* for all schemes.

Iterative methods require Jacobian-vector products. Several strategies to compute these products are used in practice. We have investigated here the impact and implications of several exact and approximate strategies for computing Jacobian-vector products on the overall accuracy and performance of different time integration schemes. The important conclusions of this investigation are as follows:

- Matrix-free methods show a reasonable efficiency improvement over direct solution approaches even for small, serial problems. This advantage is likely to increase as the problem size scales up and parallelism becomes critical.
- Finite difference approximations of Jacobian-vector products are widely used in practice, but the errors introduced have a considerable negative impact on the efficiency of SDIRK, and on the accuracy of Rosenbrock methods.
- The impact of inexact Jacobian-vector products on the accuracy of matrix-free methods can be minimized through the use of W- or K-type methods.
- The cost of directly computing exact Jacobian-vector products is nearly the same as the cost of a finite-difference approximation, and there are no approximation errors introduced.
- Exact Jacobian-vector products can be computed directly and efficiently in parallel by exploiting the memory distribution and access patterns used to optimize right hand side computations.

The ability of direct computation of Jacobian-vector products to reuse optimizations employed in computing the right hand side function reduces the programming effort as well as the memory footprint; these are considerable advantages when compared to the evaluation of the full Jacobian matrix. Direct Jacobian-vector products do not suffer from the approximation errors introduced by finite difference approximations. Consequently, matrix-free time integration schemes based on the direct evaluation of Jacobian-vector products provide the most robust choice for time discretization in large scale parallel simulations of partial differential equations.

## Acknowledgments

This work was funded in part by the awards NSF DMS-1419003, AFOSR 12-2640-06, and by the Computational Science Laboratory at Virginia Tech.

## References

[1] E. Hairer, G. Wanner, Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems, Springer, 2002.
[2] D. Knoll, D. Keyes, Jacobian-free Newton–Krylov methods: a survey of approaches and applications, J. Comput. Phys. 193 (2) (2004) 357–397. http://dx.doi.org/10.1016/j.jcp.2003.08.010. URL http://www.sciencedirect.com/science/article/pii/S0021999103004340.
[3] J. Lang, J. Verwer, Ros3p—an accurate third-order rosenbrock solver designed for parabolic problems, BIT 41 (4) (2001) 731–738.
[4] W. Liao, A strongly a-stable time integration method for solving the nonlinear reaction–diffusion equation, in: Abstract and Applied Analysis, Vol. 2015, Hindawi Publishing Corporation, 2015.
[5] R. Weiner, B. Schmitt, H. Podhaisky, ROWMAP — a ROW-code with Krylov techniques for large stiff ODEs, Appl. Numer. Math. 25 (2–3) (1997) 303–319. http://dx.doi.org/10.1016/S0168-9274(97)00067-6. URL http://www.sciencedirect.com/science/article/pii/S0168927497000676. Special Issue on Time Integration.
[6] P. Tranquilli, A. Sandu, Rosenbrock–Krylov methods for large systems of differential equations, SIAM J. Sci. Comput. 36 (3) (2014) A1313–A1338. http://dx.doi.org/10.1137/130923336.

[7] L. Hascoët, V. Pascual, The Tapenade automatic differentiation tool: principles, model, and specification, ACM Trans. Math. Software 39 (3) (2013) 20:1–20:43. http://dx.doi.org/10.1145/2450153.2450158.
[8] R. Giering, T. Kaminski, Recipes for adjoint code construction, ACM Trans. Math. Softw. 24 (4) (1998) 437–474. http://doi.acm.org/10.1145/293686.293695.
[9] C.H. Bischof, A. Carle, P. Khademi, A. Mauer, ADIFOR 2.0: Automatic differentiation of Fortran 77 programs, IEEE Comput. Sci. Eng. 3 (3) (1996) 18–32.
[10] R. Liska, B. Wendroff, Composite schemes for conservation laws, SIAM J. Numer. Anal. 35 (6) (1998) 2250–2271. http://dx.doi.org/10.1137/S0036142996310976.
[11] OpenMP Architecture Review Board, OpenMP application program interface version 4.0, 2013. URL http://www.openmp.org/mp-documents/OpenMP4.0.0.pdf.
[12] P. Tranquilli, R. Glandon, A. Sandu, CUDA acceleration of a matrix-free Rosenbrock-K method applied to the shallow water equations, in: Proceedings of the Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems, ScalA '13, ACM, New York, NY, USA, 2013, pp. 5:1–5:6. http://dx.doi.org/10.1145/2530268.2530273. URL http://doi.acm.org/10.1145/2530268.2530273.
[13] S.M. Allen, J.W. Cahn, A microscopic theory for antiphase boundary motion and its application to antiphase domain coarsening, Acta Metall. 27 (6) (1979) 1085–1095. http://dx.doi.org/10.1016/0001-6160(79)90196-2. URL http://www.sciencedirect.com/science/article/pii/0001616079901962.
[14] J. Rang, L. Angermann, New Rosenbrock methods of order 3 for PDAEs of index 2, Adv. Differ. Equ. Control Process. 1 (2) (2008) 193–217.
[15] P. Tranquilli, A. Sandu, Exponential-Krylov methods for ordinary differential equations, J. Comput. Phys. 278 (2014) 31–46. http://dx.doi.org/10.1016/j.jcp.2014.08.013. URL http://www.sciencedirect.com/science/article/pii/S0021999114005592.