Available online at www.sciencedirect.com
ScienceDirect
Journal of Computational Physics 220 (2006) 184–215
JOURNAL OF COMPUTATIONAL PHYSICS
www.elsevier.com/locate/jcp

The phase flow method

Lexing Ying *, Emmanuel J. Candès

Applied and Computational Mathematics, California Institute of Technology, Caltech, MC 217-50, Pasadena, CA 91125, United States

Received 4 October 2005; received in revised form 6 March 2006; accepted 5 May 2006
Available online 27 June 2006

Abstract

This paper introduces the phase flow method, a novel, accurate and fast approach for constructing phase maps for nonlinear autonomous ordinary differential equations. The method operates by initially constructing the phase map for small times using a standard ODE integration rule and builds up the phase map for larger times with the help of a local interpolation scheme together with the group property of the phase flow. The computational complexity of building up the complete phase map is usually that of tracing a few rays. In addition, the phase flow method is provably and empirically very accurate. Once the phase map is available, integrating the ODE for initial conditions on the invariant manifold only makes use of local interpolation, thus having constant complexity.

The paper develops applications in the field of high frequency wave propagation, and shows how to use the phase flow method to (1) rapidly propagate wave fronts, (2) rapidly calculate wave amplitudes along these wave fronts, and (3) rapidly evaluate multiple wave arrival times at arbitrary locations.
© 2006 Elsevier Inc. All rights reserved.

Keywords: The phase flow method; Ordinary differential equations; Phase maps; Phase flow; Interpolation; High-frequency wave propagation; Geometrical optics; Hamiltonian dynamics; Ray equations; Wave arrival times; Eulerian and Lagrangian formulations

1. Introduction

This paper is concerned with the problem of rapidly and accurately computing numerical solutions to ordinary differential equations (ODEs) of the form

$$\frac{d\boldsymbol{y}}{dt} = F(\boldsymbol{y}), \quad t > 0, \tag{1.1}$$

where $\boldsymbol{y}: \mathbf{R} \to \mathbf{R}^d$ and $F: \mathbf{R}^d \to \mathbf{R}^d$ is a smooth multivariate function. Note that the right-hand side of the equation does not depend on time and that we thus consider a class of so-called autonomous nonlinear differential equations. Given a time T and an initial condition $\boldsymbol{y}_0$, the basic problem is to integrate the system (1.1) and compute $\boldsymbol{y}(T, \boldsymbol{y}_0)$ which we may conveniently think as the position of a particle at time T, and with initial position $\boldsymbol{y}_0$.

* Corresponding author. Tel.: +1 626 395 5760.
E-mail address: lexing@acm.caltech.edu (L. Ying).

0021-9991/$ - see front matter © 2006 Elsevier Inc. All rights reserved.
doi:10.1016/j.jcp.2006.05.008

As is well known, standard methods for computing $\boldsymbol{y}(T,\boldsymbol{y}_0)$ select a time step and a local integration rule such as the explicit Euler method or the widely-used fourth order Runge–Kutta rule, and compute approximations of the solution by recursively applying the local integration rule at the discrete time points $n\tau$, where $n = 1,2,3,\ldots$. First, the accuracy of such strategies of course depends upon the local integration rule, and is usually of the form $O(\tau^\alpha)$, where α is called the order of the method. And second, the computation complexity of such strategies is of the order of $O(1/\tau)$ since one takes as many as $O(1/\tau)$ time steps and that each step has constant complexity. It goes without saying that there is a huge literature on the various methods for solving ODEs that we shall not review. Instead, we refer to [15,16,19] for a comprehensive exposition.

1.1. A new approach

In many situations, one needs to solve (1.1) for multiple initial conditions, and the standard approach is then to compute every solution independently which may be extremely costly. In this paper, we introduce a new strategy for computing numerical solutions to ODEs we call the *phase flow method*. The method operates by initially constructing the phase map for small times using standard ODE integrators and builds up the phase map for larger times with the help of a local interpolation scheme together with the group property of the phase flow. To the best of our knowledge, the phase flow method is radically different from standard ODE integration techniques.

To explain the key ideas underlying this new method, we need to introduce some elementary terminology. For a fixed time t, the map $g_t:\mathbf{R}^d \to \mathbf{R}^d$ defined by $g_t(\boldsymbol{y}_0) = \boldsymbol{y}(t,\boldsymbol{y}_0)$ is called the *phase map*, and the family $\{g_t, t \in \mathbf{R}\}$ of all phase maps the *phase flow*. The phase flow has a one parameter group structure, $g_{t'} \circ g_t = g_{t'+t}$, and is important in the study of autonomous ODEs. A manifold $M \subset \mathbf{R}^d$ is said to be *invariant* if $g_t(M) \subset M$. In many practical cases, one is interested in the behavior of (1.1) on an invariant manifold.

One such practical case may be the computations of solutions to Hamilton–Jacobi equations as those arising in the approximation of high-frequency waves, also known as geometrical optics. Here, one is interested in the trajectories $\boldsymbol{y}(t) = (\boldsymbol{x}(t),\boldsymbol{p}(t))$ in the phase space $\mathbf{R}^d \times \mathbf{R}^d$, $d = 2,3$, which are solutions to the Hamiltonian flow

$$\frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}t} = \nabla_{\boldsymbol{p}} H(\boldsymbol{x},\boldsymbol{p}), \qquad \frac{\mathrm{d}\boldsymbol{p}}{\mathrm{d}t} = -\nabla_{\boldsymbol{x}} H(\boldsymbol{x},\boldsymbol{p}), \tag{1.2}$$

where the Hamiltonian $H(\boldsymbol{x},\boldsymbol{p})$ is given by

$$H(\boldsymbol{x},\boldsymbol{p}) = c(\boldsymbol{x})|\boldsymbol{p}|,$$

and $c(\boldsymbol{x})$ is the medium sound speed. In Section 3, we shall briefly review the reason why the Hamiltonian flow describes the propagation of high-frequency waves. Given the initial position of a wave front—a smooth surface S in phase space—a problem of considerable interest is then to compute the location of the wave front at a later time. In other words, we wish to integrate the system (1.2) for each $\boldsymbol{y}_0 = (\boldsymbol{x}_0,\boldsymbol{p}_0) \in S$. Here, the invariant manifold M might be $\mathbf{R}^d \times \mathbf{R}^d$, or $\mathbf{R}^d \times \mathbf{S}^{d-1}$ where one substitutes $\boldsymbol{p}$ with the unit vector $\boldsymbol{v} = \boldsymbol{p}/|\boldsymbol{p}|$ which has the same orientation. In the case where the wave speed is periodic, M can even be reduced to a domain which is equivalent to the compact set $[0,1)^d \times \mathbf{S}^{d-1}$, see Section 3.

Instead of integrating (1.1) for each individual initial condition $\boldsymbol{y}_0$, the phase flow method constructs the complete phase map g_T at time T. The method is described in Section 2 but we now sketch the key ingredients. To do so, we fix a small time $\tau > 0$, and suppose that $T = 2^n\tau$.

1. *Discretization.* The phase flow methods assumes that we have available a uniform or quasi-uniform grid on M.
2. *Initialization.* We then start by computing an approximation of the phase map g_τ at time τ. The value of g_τ at each gridpoint is computed by applying a standard ODE integration rule with a single time step of length τ. The value of g_τ at any other point is defined via local interpolation.
3. *Loop.* For $k = 1,\ldots,n$, we then construct $g_{2^k\tau}$ using the group relation $g_{2^k\tau} = g_{2^{k-1}\tau} \circ g_{2^{k-1}\tau}$. For each gridpoint $\boldsymbol{y}_0$,

$$g_{2^k\tau}(\boldsymbol{y}_0) = g_{2^{k-1}\tau}(g_{2^{k-1}\tau}(\boldsymbol{y}_0)) \tag{1.3}$$

while $g_{2^k\tau}$ is defined via local interpolation at any other point.

When the algorithm terminates, one holds an approximation of the whole phase map at time $T = 2^n\tau$. In Section 2, we will see that the phase flow method is as accurate as the standard integration methods discussed above. What is remarkable, however, is that under certain assumptions, the asymptotic computational complexity of the phase flow algorithm is lower than that of a standard algorithm tracing out a *single* ray, i.e. integrating the solution for a single initial condition. On top of that, as soon as g_T is available, computing $\boldsymbol{y}(T,\boldsymbol{y}_0)$ for any arbitrary initial condition $\boldsymbol{y}_0 \in M$ only involves local interpolation, an operation with O(1) complexity.

As we pointed out earlier, this is only a rough sketch of the method and our aim here is to convey the main underlying ideas. Namely, the phase flow method avoids many computations by making a systematic use of what has actually been computed at previous steps. In details, the method uses a *repeated squaring type* algorithm as in (1.3) although the time increments are typically tuned differently as to better control the accuracy, see Section 2.

1.2. High frequency wave propagation

This paper then applies the phase flow method to solve important problems arising in computational high frequency wave propagation. The numerical methods we develop are for classical high frequency asymptotic expansions of the scalar wave equation

$$u_{tt}(\boldsymbol{x}) - c(\boldsymbol{x})^2\Delta u = 0, \quad t > 0, \ \ \boldsymbol{x} \in \mathbf{R}^d, \tag{1.4}$$

here, t is time, $\boldsymbol{x}$ is the spatial variable and $u(t,\boldsymbol{x})$ is the amplitude of the wave. Standard numerical methods such as finite difference, finite volume and finite element methods [21,24] require a few gridpoints or cells per wavelength in order to resolve the wave field accurately. When the wavelengths are small compared to the size of the computational domain, these methods are very expensive and approximations must be used. In the geometrical optics approximation, the solution to the wave equation (1.4) is expanded in inverse powers of ω

$$u(t,\boldsymbol{x}) = \mathrm{e}^{\mathrm{i}\omega\Phi(t,\boldsymbol{x})}\sum_{n\geqslant 0} A_n(t,\boldsymbol{x})(\mathrm{i}\omega)^{-n}, \tag{1.5}$$

where u is a complex solution of the wave equation, $\Phi(t,\boldsymbol{x})$ is the phase of the wave, and the A_n's are real amplitudes which are functions of time and space. Substituting this expansion into the wave equation and equating the coefficients of powers of ω gives a set of equations for the phase and the amplitudes. As is well known, high-frequency expansions such as the WKB ansatz (1.5) are good approximations to some of the linear waves which are solutions to (1.4) [37]. These approximations are nonlinear, however, and one of the goals of this paper is to introduce novel numerical methods based on such high-frequency approximations.

Specifically, we consider three problems of significant interest:

1. *Wave front tracking.* In the context of high-frequency waves, one can think of a wave front as the level set of the phase function Φ. Given the location of a wave front at time $t = 0$, e.g. the surface $\{\boldsymbol{x}:\Phi(0,\boldsymbol{x}) = 0\}$, we wish to compute the position of the wave front at an arbitrary later time t, e.g. the surface $\{\boldsymbol{x}:\Phi(t,\boldsymbol{x}) = 0\}$.
2. *Amplitude computations.* The problem here is to calculate an approximation of the amplitude of the wave (1.5) along a wave front of interest. For example, one may want to evaluate the leading amplitude term $A_0(t,\boldsymbol{x})$ along the surface $\{\boldsymbol{x}:\Phi(t,\boldsymbol{x}) = 0\}$.
3. *Multiple arrival times computations.* Given a collection of sources $\{x_\mathrm{s}\}$ and target points $\{x_\mathrm{r}\}$ (receivers), the problem is to compute the arrival times of waves at each point x_r. Of special interest is the situation where a single source generates multiple arrival times, a situation which often occurs when the medium is inhomogeneous.

As we shall see in Section 3, the phase flow method can be deployed in all three settings, hence offering new and elegant methods to solve each of these problems. Our solutions are efficient and accurate but we would like to emphasize that above all, they are markedly different from existing approaches, which we now briefly review.

Obviously, there is a vast literature on numerical methods for solving problems 1, 2 and 3; these methods are roughly divided between Eulerian and Lagrangian methods. For the problem of propagating wave fronts for example, a Lagrangian method often discretizes the initial wave front with Lagrangian markers. These markers are traced using the ray equations to obtain the wave front at a later time. Besides the fact that one typically needs to trace out many such rays, there is an additional difficulty stemming from shadow regions where rays diverge thus causing a substantial loss of resolution. Commonly discussed approaches to address this issue monitor the ray density as the wave front evolves and insert new rays whenever this is necessary [20,36]. This is problematic as standard methods for inserting rays are typically either computationally expensive or rather imprecise. As we shall see, the phase flow method easily overcomes these difficulties since as we have seen, it allows the tracing of new rays—i.e. compute new solutions—with O(1) complexity per ray. In other words, inserting new points is automatic and effortless.

Another line of work has pushed the Eulerian point of view to solve for the phase function or compute the evolution of the wave front. Methods based on the resolution of the time dependent eikonal equation (the nonlinear equation for the phase Φ) are imperfect because the superposition principle does not hold; waves are not allowed to cross since the ansatz postulates that there is at most one wave front at any given location. Such methods then only give the viscosity solution [34,35], i.e. only resolve the first arriving wave. Extensions to obtain multivalued solutions [1,2,12,26,29,31] are often complicated and often rely on discretionary choices. This is the reason why many modern computational strategies now operate in phase space [3,10]. The Eulerian viewpoint then assumes a discretization of phase space by means of a Cartesian grid. Physical quantities, e.g. the particle density function $f(t,\boldsymbol{x},\boldsymbol{p})$, are then computed by solving the PDE formulation of geometrical optics on the grid, e.g. the Liouville equation. Interesting work in this rapidly growing area includes the methods of Osher and his colleagues based on the level set formulation [5,22,23], the fast marching method of Fomel and Sethian [13], the moment-based method of Engquist and Runborg [9,25], and the segment projection method of Engquist et al. [11,33].

From a certain viewpoint, the phase flow method can be viewed as a hybrid method. On the one hand and much like in an Eulerian scenario, it starts with a discretization of a reduced version of phase space. On the other hand, however, the wave fronts and amplitudes are traced in a Lagrangian fashion using the ray equations.

2. The phase flow method

We are interested in solving the system of nonlinear differential equations (1.1) $\mathrm{d}\boldsymbol{y}/\mathrm{d}t = F(\boldsymbol{y})$, where we recall that $\boldsymbol{y}$ takes on values in $\mathbf{R}^d$ and $F:\mathbf{R}^d \to \mathbf{R}^d$ is a smooth function. Throughout this paper, $M \subset \mathbf{R}^d$ denotes a smooth, compact, $d(M)$-dimensional invariant manifold, and we will use $g_t: M \to M$ to denote the restriction of the phase map to M. The distance between any two points $\boldsymbol{y}_0$ and $\boldsymbol{y}_1$ on the manifold M is simply the usual Euclidean distance in $\mathbf{R}^d$ denoted by $|\boldsymbol{y}_0 - \boldsymbol{y}_1|$. Our methods can be adapted to handle other types of distances but we shall not consider such extensions in this paper. Finally, because we are only interested in the solutions near M, we assume without loss of generality that F is supported in an open convex and bounded domain U which contains M (the convexity assumption merely simplifies the proofs of the main results). This ensures that F satisfies the Lipschitz condition and thus the existence and uniqueness of the solution to (1.1) is guaranteed.

The phase flow algorithm introduced below (variants exist as we will see later) constructs an approximation $\tilde{g}_T : M \to M$ to the "true" phase map g_T so that the approximate solution, denoted by $\tilde{\boldsymbol{y}}(T,\boldsymbol{y}_0)$, is equal to $\tilde{g}_T(\boldsymbol{y}_0)$. We will assume throughout that an ODE integration rule and an interpolation scheme on M have been preselected. The extra assumptions on the interpolation scheme include that the construction of the interpolant has linear complexity in terms of the grid size and that the complexity of evaluating the interpolant at any point is O(1).

Algorithm 1 (The phase flow method)

- *Parameter selection.* Select a grid size $h > 0$, a time step $\tau > 0$, and an integer constant $S \geqslant 1$ such that $B = (T/\tau)^{1/S}$ is an integer power of 2.

- *Discretization.* Select a uniform or quasi-uniform grid $M_h \subset M$ of size h.
- *Burn-in.* Compute $\tilde{g}_\tau$. For a gridpoint $\boldsymbol{y}_0$, $\tilde{g}_\tau(\boldsymbol{y}_0)$ is calculated by applying the ODE integrator (single time step of length τ). Then construct an interpolant based on these sampled values, and for any other point $\boldsymbol{y}_0$, define $\tilde{g}_\tau(\boldsymbol{y}_0)$ by evaluating the interpolant at $\boldsymbol{y}_0$.
- *Loop.* For $k = 1, \ldots, S$, evaluate $\tilde{g}_{B^k\tau}$. For a gridpoint $\boldsymbol{y}_0$, $\tilde{g}_{B^k\tau}(\boldsymbol{y}_0) = (\tilde{g}_{B^{k-1}\tau})^{(B)}(\boldsymbol{y}_0)$ where $f^{(2)} = f \circ f$, $f^{(3)} = f \circ f \circ f$ and so on. Construct an interpolant based on these sampled values, and for any other point $\boldsymbol{y}_0$, define $\tilde{g}_{B^k\tau}(\boldsymbol{y}_0)$ by evaluating the interpolant at $\boldsymbol{y}_0$.
- *Terminate.* The algorithm terminates at $k = S$ since by definition $B^S\tau = T$ and hence $\tilde{g}_T = \tilde{g}_{B^S\tau}$. The approximate solution $\tilde{\boldsymbol{y}}(T, \boldsymbol{y}_0)$ is equal to $\tilde{g}_T(\boldsymbol{y}_0)$.

The phase flow algorithm relies on three components which have not been specified yet, namely, the ODE integration rule, the local interpolation scheme used in the *Burn-in* and *Loop* steps, and the selection of the discrete grid M_h. These components may of course be problem dependent and shall be specified in later sections when discussing concrete instances or applications. For the moment and at this level of generality, we would simply like to remark that both the ODE integrator and the local interpolation scheme may involve a projection step onto M since we require $\tilde{g}_t$ to be a mapping from M onto itself.

2.1. Error analysis

Algorithm 1 is amenable to rigorous error analysis and the theorem below enumerates some of its key properties.

Theorem 2.1. *Suppose that the ODE integrator is of order α and that the local interpolation scheme is of order $\beta \geqslant 2$ for sufficiently smooth functions. We shall also assume that the linear interpolation rule has h-independent L^∞ norm on continuous functions. Define the approximation error at time t by*

$$\varepsilon_t = \max_{\boldsymbol{b} \in M} |g_t(\boldsymbol{b}) - \tilde{g}_t(\boldsymbol{b})|. \tag{2.1}$$

Then Algorithm 1 enjoys the following properties:

(i) *The approximation error obeys*

$$\varepsilon_T \leqslant C \cdot (\tau^\alpha + h^\beta) \tag{2.2}$$

for some positive constant $C > 0$.

(ii) *The complexity is $\mathrm{O}(\tau^{-1/S} \cdot h^{-d(M)})$.*

(iii) *For each $y \in M$, $\tilde{g}_T(\boldsymbol{y})$ can be computed in $\mathrm{O}(1)$ operations.*

(iv) *For any intermediate time $t = m\tau \leqslant T$ where m is an integer, one can evaluate $\tilde{g}_t(\boldsymbol{y})$ for each $\boldsymbol{y} \in M$ in $\mathrm{O}(\log(1/\tau))$ operations.*

For a multi-index γ, we use $\partial^\gamma F(\boldsymbol{y})$ to denote the spatial derivative of F with respect to $\boldsymbol{y}$, while $\partial^\gamma \boldsymbol{y}(t, \boldsymbol{b})$ is the derivative of $\boldsymbol{y}$ with respect to the initial condition $\boldsymbol{b}$. The proof of Theorem 2.1 depends on the following two lemmas.

Lemma 2.2. *Suppose F has Lipschitz constant L on U. Then for any pair $\boldsymbol{b}_0, \boldsymbol{b}_1$ in M, we have*

$$|\boldsymbol{y}(t, \boldsymbol{b}_0) - \boldsymbol{y}(t, \boldsymbol{b}_1)| \leqslant \mathrm{e}^{Lt} |\boldsymbol{b}_0 - \boldsymbol{b}_1|.$$

This is a simple application of Gronwall's inequality and we omit the proof.

Lemma 2.3. *For each $s \geqslant 2$, there exists a constant C_s such that for any multi-index γ with $|\gamma| = s$ and any $\boldsymbol{b} \in M$*

$$|\partial^\gamma \boldsymbol{y}(t, \boldsymbol{b})| \leqslant C_s \mathrm{e}^{(2s-1)LT} \cdot t.$$

Proof. We use the componentwise notation for the vectors $\boldsymbol{y} = (y^1, \ldots, y^d)$ and $F = (F^1, \ldots, F^d)$, which allows rewriting (1.1) as

$$\frac{\mathrm{d}y^i}{\mathrm{d}t} = F^i(y^1, \ldots, y^d), \quad y^i(0) = y_0^i. \tag{2.3}$$

Since F is smooth and its support is contained in a bounded domain U, there exists a sequence of constants $\{C_s^1\}$ such that, for any multi-index γ with $|\gamma| \leqslant s$, we have

$$|\partial^\gamma F^i(\boldsymbol{y})| \leqslant C_s^1, \quad \boldsymbol{y} \in U.$$

The next step is to establish a bound for $\partial^\gamma \boldsymbol{y}(t, \boldsymbol{b})$. We claim that there exists a sequence of constants $\{C_s^2\}$ such that, for any γ with $|\gamma| = s$

$$|\partial^\gamma \boldsymbol{y}(t, \boldsymbol{b})| \leqslant C_s^2 \mathrm{e}^{(2s-2)LT+Lt} \leqslant C_s^2 \mathrm{e}^{(2s-1)LT}, \quad 0 \leqslant t \leqslant T.$$

We prove this by induction and start with $s = 1$. Differentiate (2.3) and obtain

$$\frac{\mathrm{d}y_k^i}{\mathrm{d}t} = \sum_a F_a^i y_k^a,$$

where the lower script denotes differentiation with respect to the kth variable. Because the norm of the d by d matrix F_a^i is the Lipschitz constant L of F, we have

$$\frac{\mathrm{d}|\boldsymbol{y}_k|}{\mathrm{d}t} \leqslant \left|\frac{\mathrm{d}\boldsymbol{y}_k}{\mathrm{d}t}\right| \leqslant L \cdot |\boldsymbol{y}_k|, \quad |\boldsymbol{y}_k(0)| = 1,$$

where the first inequality is a simple consequence of the triangle inequality. A direct application of Gronwall's inequality then shows

$$|\boldsymbol{y}_k(t)| \leqslant \mathrm{e}^{Lt}.$$

Now assume the claim is true for any multi-index of order less than s. Fix γ with $|\gamma| = s > 1$ and differentiate (2.3) on both sides to obtain

$$\frac{\mathrm{d}\partial^\gamma y^i}{\mathrm{d}t} = \sum_{p=1}^{s} \sum_{(\gamma_1, \ldots, \gamma_p): \gamma = \sum \gamma_j} \left(\sum_{a_1, \ldots, a_p} F^i_{a_1, \ldots, a_p} \prod_{k=1}^{p} \partial^{\gamma_k} y^{a_k} \right), \quad \partial^\gamma y^j(0) = 0; \tag{2.4}$$

the second summation is here over all different choices of $(\gamma_1, \ldots, \gamma_p)$ (up to permutations) where $\gamma = \gamma_1 + \cdots + \gamma_p$ and $|\gamma_j| > 0$. There is only a single term in this expansion corresponding to $p = 1$, namely, $\sum_a F_a^i \partial^\gamma y^a$. Every other term with $p > 1$ involves a product of different partial derivatives of y^i. First, by induction

$$\prod_{k=1}^{p} \partial^{\gamma_k} y^{a_k} \leqslant C \cdot \mathrm{e}^{(2|\gamma|-p)LT}$$

for $p > 1$. Second, the number of terms (which only depends on s and d) is finite. It then follows that

$$\frac{\mathrm{d}|\partial^\gamma \boldsymbol{y}|}{\mathrm{d}t} \leqslant L|\partial^\gamma \boldsymbol{y}| + C' \mathrm{e}^{(2s-2)LT},$$

where C' is a constant which depends on C_p^1 for $p \leqslant s$ and C_p^2 for $p < s$. Since $|\partial^\gamma \boldsymbol{y}(0)| = 0$, applying Gronwall's inequality gives

$$|\partial^\gamma \boldsymbol{y}(t, \boldsymbol{b})| \leqslant \frac{C'}{L} \mathrm{e}^{(2s-2)LT+Lt} \leqslant \frac{C'}{L} \mathrm{e}^{(2s-1)LT}.$$

Putting $C_s^2 = C'/L$ proves the claim.

To prove the lemma, observe that the right hand side of (2.4) is bounded by $C_s/L \cdot \mathrm{e}^{(2s-2)LT+Lt}$ where the constant C_s depends on C_s^1 and C_s^2. Integrating from time 0 to t gives

$$|\partial^\gamma \boldsymbol{y}(t,\boldsymbol{b})| \leqslant C_s \cdot \mathrm{e}^{(2s-2)LT} \cdot \frac{\mathrm{e}^{Lt}-1}{L} \leqslant C_s \cdot \mathrm{e}^{(2s-1)LT} \cdot t.$$

The proof is complete. □

Proof of Theorem 2.1. We begin by proving (i) and first develop an estimate for the error after the first time step of size τ. Suppose that $\boldsymbol{y}$ is a gridpoint, then

$$|\tilde{g}_\tau(\boldsymbol{y}) - g_\tau(\boldsymbol{y})| \leqslant C_0 \cdot \tau^{\alpha+1}$$

since the ODE integrator has order α. (The constant C_0 here depends on the behavior of g_t for $t \in [0,\tau]$.) Let I (resp. $\widetilde{I}$) be the interpolant constructed from the values of $g_\tau(\boldsymbol{b})$ (resp. $\tilde{g}_\tau(\boldsymbol{b})$) where $\boldsymbol{b}$ ranges through the discrete grid M_h. If $\boldsymbol{y}$ is not a gridpoint, $\boldsymbol{y} \in M \backslash M_h$, we have $\tilde{g}_\tau(\boldsymbol{y}) = \widetilde{I}(\boldsymbol{y})$ and

$$|\tilde{g}_\tau(\boldsymbol{y}) - g_\tau(\boldsymbol{y})| \leqslant |\widetilde{I}(\boldsymbol{y}) - I(\boldsymbol{y})| + |I(\boldsymbol{y}) - g_\tau(\boldsymbol{y})|.$$

Assuming N_I to be the h-independent norm of the interpolation operator, the first term is bounded by

$$|\widetilde{I}(\boldsymbol{y}) - I(\boldsymbol{y})| \leqslant N_I \cdot \max_{\boldsymbol{b}\in M_h} |\tilde{g}_\tau(\boldsymbol{b}) - g_\tau(\boldsymbol{b})| \leqslant C_0 \cdot N_I \cdot \tau^{\alpha+1},$$

while the second term obeys

$$|I(\boldsymbol{y}) - g_\tau(\boldsymbol{y})| \leqslant C_1 \cdot h^\beta \cdot \max_{|\gamma|=\beta} \sup_{\boldsymbol{b}\in M} |\partial^\gamma \boldsymbol{y}(\tau,\boldsymbol{b})| \leqslant C_2 \cdot h^\beta \cdot \tau$$

following Lemma 2.3. Therefore, the error term ε_τ obeys

$$\varepsilon_\tau \leqslant C_0 \cdot N_I \cdot \tau^{\alpha+1} + C_2 \cdot h^\beta \cdot \tau. \tag{2.5}$$

Consider now $\varepsilon_{B\tau}$. If $\boldsymbol{y} \in M_h$, then

$$|g_{2\tau}(\boldsymbol{y}) - \tilde{g}_{2\tau}(\boldsymbol{y})| \leqslant |g_\tau(g_\tau(\boldsymbol{y})) - g_\tau(\tilde{g}_\tau(\boldsymbol{y}))| + |g_\tau(\tilde{g}_\tau(\boldsymbol{y}) - \tilde{g}_\tau(\tilde{g}_\tau(\boldsymbol{y}))| \leqslant \mathrm{e}^{L\tau}\varepsilon_\tau + \varepsilon_\tau = \frac{\mathrm{e}^{2L\tau}-1}{\mathrm{e}^{L\tau}-1}\varepsilon_\tau,$$

where we applied Lemma 2.2. Repeating this procedure gives

$$|g_{B\tau}(\boldsymbol{y}) - \tilde{g}_{B\tau}(\boldsymbol{y})| \leqslant \frac{\mathrm{e}^{LB\tau}-1}{\mathrm{e}^{L\tau}-1}\varepsilon_\tau.$$

If $\boldsymbol{y}$ is not a gridpoint, the same decomposition as that used to bound the distance between $\tilde{g}_\tau(\boldsymbol{y})$ and $g_\tau(\boldsymbol{y})$ shows that

$$\varepsilon_{B\tau} \leqslant N_I \cdot \frac{\mathrm{e}^{LB\tau}-1}{\mathrm{e}^{L\tau}-1}\varepsilon_\tau + C_2 \cdot h^\beta \cdot B\tau.$$

In conclusion, for $1 \leqslant k \leqslant S$, our argument gives the following recurrence relation:

$$\varepsilon_{B^k\tau} \leqslant N_I \cdot \frac{\mathrm{e}^{LB^k\tau}-1}{\mathrm{e}^{LB^{k-1}\tau}-1} \cdot \varepsilon_{B^{k-1}\tau} + C_2 \cdot h^\beta \cdot B^k\tau.$$

It remains to expand the last inequality for $k = S$ which gives

$$\varepsilon_T \leqslant N_I^S \frac{\mathrm{e}^{LB^S\tau}-1}{\mathrm{e}^{L\tau}-1}\varepsilon_\tau + C_2 \cdot h^\beta \cdot \sum_{k=0}^{S} N_I^k \frac{\mathrm{e}^{LB^S\tau}-1}{\mathrm{e}^{LB^{S-k}\tau}-1} B^{S-k}\tau.$$

For the first term, $\mathrm{e}^{L\tau} - 1 \geqslant L\tau$ and, therefore, it follows from (2.5) that

$$N_I^S \frac{\mathrm{e}^{LB^S\tau}-1}{\mathrm{e}^{L\tau}-1}\varepsilon_\tau \leqslant \frac{\mathrm{e}^{LT}-1}{L}\left(C_0 \cdot N_I^{S+1} \cdot \tau^\alpha + C_2 \cdot N_I^S \cdot h^\beta\right).$$

For the second term, we observe that

$$\sum_{k=0}^{S} N_I^k \frac{\mathrm{e}^{LB^S\tau}-1}{\mathrm{e}^{LB^{S-k}\tau}-1} B^{S-k}\tau \leqslant \frac{\mathrm{e}^{LT}-1}{L}\sum_{k=0}^{S} N_I^k$$

and, therefore,

$$\varepsilon_T \leqslant C \cdot (h^\beta + \tau^\alpha),$$

where the constant C depends on C_0, C_2, N_I, S, T and L.

We now prove (ii) and let $|M_h|$ be the number of gridpoints. The one-step ODE integrator takes a constant number of operations per gridpoint and is thus of the order of $O(|M_h|)$. As we have seen, the construction of $\tilde{g}_T$ is divided into S stages, and each stage requires applying the approximate phase map B times. Hence each stage has complexity $O(B \cdot |M_h|)$. Since M is a $d(M)$-dimensional manifold, the size of the grid may not exceed $O(1/h^{d(M)})$ and, therefore, the total complexity is $O(SB/h^{d(M)}) = O(\tau^{-1/S} \cdot h^{-d(M)})$ since S is a fixed small integer.

The claim (iii) is immediate since for each $\boldsymbol{y} \in M$, computing $\tilde{g}_T(\boldsymbol{y})$ only uses local interpolation.

We finally prove (iv). Since B is a power of 2, the intermediate steps of the phase flow method build $\tilde{g}_{t_k}$ for all dyadic times of the form $t_k = 2^k\tau$. Fix a time $t = m\tau$, where m is an arbitrary integer, and consider the binary expansion $m = 2^{k_1} + 2^{k_2} + \cdots$. Clearly, we have available the approximation $\tilde{g}_t = \tilde{g}_{t_{k_1}} \circ \tilde{g}_{t_{k_2}} \circ \cdots$ where there are at most $\log_2(T/\tau)$ terms in the composition. The claim (iv) follows, and this concludes the proof of the theorem. □

To simplify the exposition, we have assumed that both M and F where smooth in the sense that F had derivatives up to any order, say. This is of course not necessary. We note from the proof that we only need that the mapping $\boldsymbol{y}(t, \boldsymbol{b})$ be $C^{\alpha+1}$ with respect to t and C^β with respect to $\boldsymbol{b}$. It is then sufficient to simply assume that M and F be r times continuously differentiable with $r = \max(\alpha + 1, \beta)$.

Specializing the grid size so that the two types of error be of the same magnitude gives:

Corollary 2.4. *Suppose one selects the grid size so that $h^\beta = \tau^\alpha$. Then the accuracy of the method is $\varepsilon_T = O(\tau^\alpha)$ and the complexity of the phase flow algorithm is $O(\tau^{-1/S - d(M)\alpha/\beta})$.*

One comment is in order. Suppose that M and F are sufficiently smooth, then one can choose β and S large enough so that $d(M)\alpha/\beta + 1/S < 1$. Admittedly, the claim in the corollary is then rather surprising. *In effect, it says that in an asymptotic sense, one can compute an approximation to the entire phase map g_T much faster than one computes—with the same order of accuracy—a single solution with the standard ODE integration rule.* This is of course an asymptotic statement and we have not studied instances where this might be 'practically' true. The construction of the phase maps in the numerical examples of Section 4 takes about the same amount of time as integrating a few dozen or perhaps a few hundred solutions. Therefore, whenever one needs to solve ODEs with thousands or even more initial conditions, the phase flow method offers significant speedup.

2.2. Variations

In Algorithm 1 we have chosen S to be a constant. Instead, one can also let B take on a small integer value, e.g. $B = 2$, and this results in a "time-doubling" type algorithm. However, the claims (i) and (ii) of Theorem 2.1 no longer hold since $S = \log_B(T/\tau)$ is no longer a fixed small integer. In this case, the complexity of the algorithm is reduced but there is a loss of accuracy due to an increase in the size of the interpolation errors. The reason is that the phase map is interpolated at the discrete time points $B^k\tau$ and the smaller B, the more interpolation steps—thus causing a loss of accuracy.

Corollary 2.5. *The time-doubling version of Algorithm 1 with $B = N_I$ (the h-independent interpolation norm) has complexity $O(\log(1/\tau) \cdot h^{-1/d(M)})$ but reduced accuracy $O((\tau^\alpha + h^\beta)/\tau)$. The time-doubling version of Algorithm 1 with $B = 2$ has accuracy $O((\tau^\alpha + h^\beta)/\tau^r)$, where $r = \log_2(N_I)$.*

The proof is a minor adaptation of that of Theorem 2.1. Although this version is less attractive from a theoretical viewpoint, it can still be useful for practical purposes.

For large times, g_T may become quite oscillatory while remaining smooth. In order to capture the variations of g_T, one would then need to use a tiny spacing for the discretization of M. In practice, the modified version below is often more efficient for large times T.

Algorithm 2 (*The phase flow method: modified version*)

- Choose $T_0 = \mathrm{O}(1)$ such that g_{T_0} remains non-oscillatory and pick h so that the grid is sufficiently dense to approximate g_{T_0} accurately. Assume that $T = mT_0$, where m is an integer (possibly but not necessarily a power of 2).
- Construct $\tilde{g}_{T_0}$ using Algorithm 1.
- For any $\boldsymbol{y}_0$, define $\tilde{g}_T(\boldsymbol{y}_0)$ by $\tilde{g}_T(\boldsymbol{y}_0) = (\tilde{g}_{T_0})^{(m)}(\boldsymbol{y}_0)$.

Since $T_0 = \mathrm{O}(1)$, the conclusions of Theorem 2.1 remain valid for Algorithm 2.

In practice, the choice of the parameters T_0 and h is often problem specific. The general rule for h is that the discretization grid M_h must be dense enough to give an accurate approximation of the map F restricted to the manifold M—preferably, significantly denser. One may want to select the value of T_0 along with the construction of the phase maps as follows: we monitor the error of $\tilde{g}_{B^k\tau}$ for increasing values of k, which can be done by tracing a dozen of rays accurately from random initial conditions and comparing with $\tilde{g}_{B^k\tau}$; we then terminate the algorithm at $k = K$ where the error of $\tilde{g}_{B^K\tau}$ is close to a prescribed error threshold, and set $T_0 = B^K\tau$.

We conclude this section with a pointer to Section 4 which complements the theoretical analysis with a series of numerical experiments showing that the phase flow method is practically very effective. Further, Section 4 is also informative in the sense that it proposes concrete choices for the discretization of the invariant manifold M.

3. High frequency wave propagation

This section applies the phase flow method to solve three important problems in computational high frequency wave propagation, namely, the tracking of wave fronts, the computations of wave amplitudes, and the computations of wave arrival times. The model we use is the standard geometrical optics formulation introduced in Section 1. To keep the exposition self-contained, we first review the basic tenets of this approximation.

3.1. Geometrical optics

Here, one assumes a classical high-frequency approximation to a wave field $u(t, \boldsymbol{x})$ of the form

$$u(t, \boldsymbol{x}) = \mathrm{e}^{\mathrm{i}\omega\Phi(t,\boldsymbol{x})} \sum_{n \geqslant 0} A_n(t, \boldsymbol{x})(\mathrm{i}\omega)^{-n},$$

where $\boldsymbol{x}$ is a two or three-dimensional spatial variable, t is time, and ω is a large base frequency. After substituting the approximation in the scalar wave equation (1.4), it follows that the phase function Φ must obey the eikonal equation

$$\Phi_t \pm c|\nabla\Phi| = 0, \tag{3.1}$$

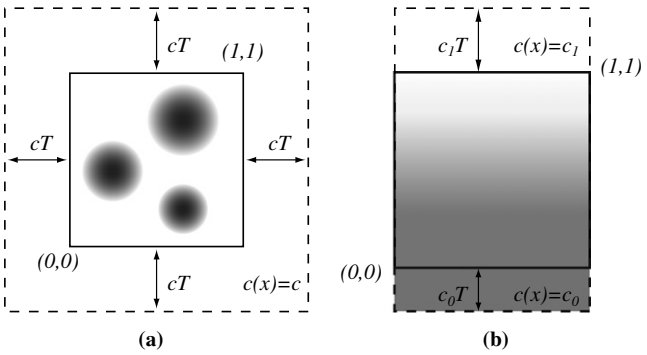


Fig. 1. Non-periodic wave speeds.

where we recall that $c(\boldsymbol{x})$ is the wave speed. Without loss of generality, consider the equation with the plus sign. Then the amplitudes obey transport equations, and we only give that for the first order term

$$A_{0,t} + c\frac{\nabla\Phi}{|\nabla\Phi|}\cdot\nabla A_0 + \frac{A_0(c^2\Delta\Phi - \Phi_{tt})}{2c|\nabla\Phi|} = 0. \tag{3.2}$$

Numerical methods based on geometric optics solve for these two functions.

The connections with systems of nonlinear ODEs are through the ray equations we introduced earlier

$$\frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}t} = c(\boldsymbol{x})\frac{\boldsymbol{p}}{|\boldsymbol{p}|}, \qquad \frac{\mathrm{d}\boldsymbol{p}}{\mathrm{d}t} = -|\boldsymbol{p}|\nabla c(\boldsymbol{x}), \tag{3.3}$$

where $\boldsymbol{p}$ is called the *slowness* vector. These are the equations of the Hamiltonian dynamics (1.2) with Hamiltonian $H(\boldsymbol{x},\boldsymbol{p}) = c(\boldsymbol{x})\boldsymbol{p}$, and the integral curves of (3.3) are often called *rays*. Now the ray equations (3.3) describe the evolution of the wave front because the phase function is actually *constant* along the rays; that is, setting $\boldsymbol{p}(0) = \nabla\Phi(0,\boldsymbol{x}(0))$ gives $\Phi(t,\boldsymbol{x}(t)) = \Phi(0,\boldsymbol{x}(0))$. This is why the system (3.3) is also called the *bicharacteristic flow* and the rays $(\boldsymbol{x}(t),\boldsymbol{p}(t))$ the bicharacteristics. In short, propagating a wave front—a level set of Φ—may be thought of as tracing rays. Once the phase of the wave is known, one can solve for the amplitude; in fact, the amplitude function is also transported along the rays as indicated by (3.2). Spreading and concentration of neighboring rays relate to the attenuation and amplification of the wave amplitude.

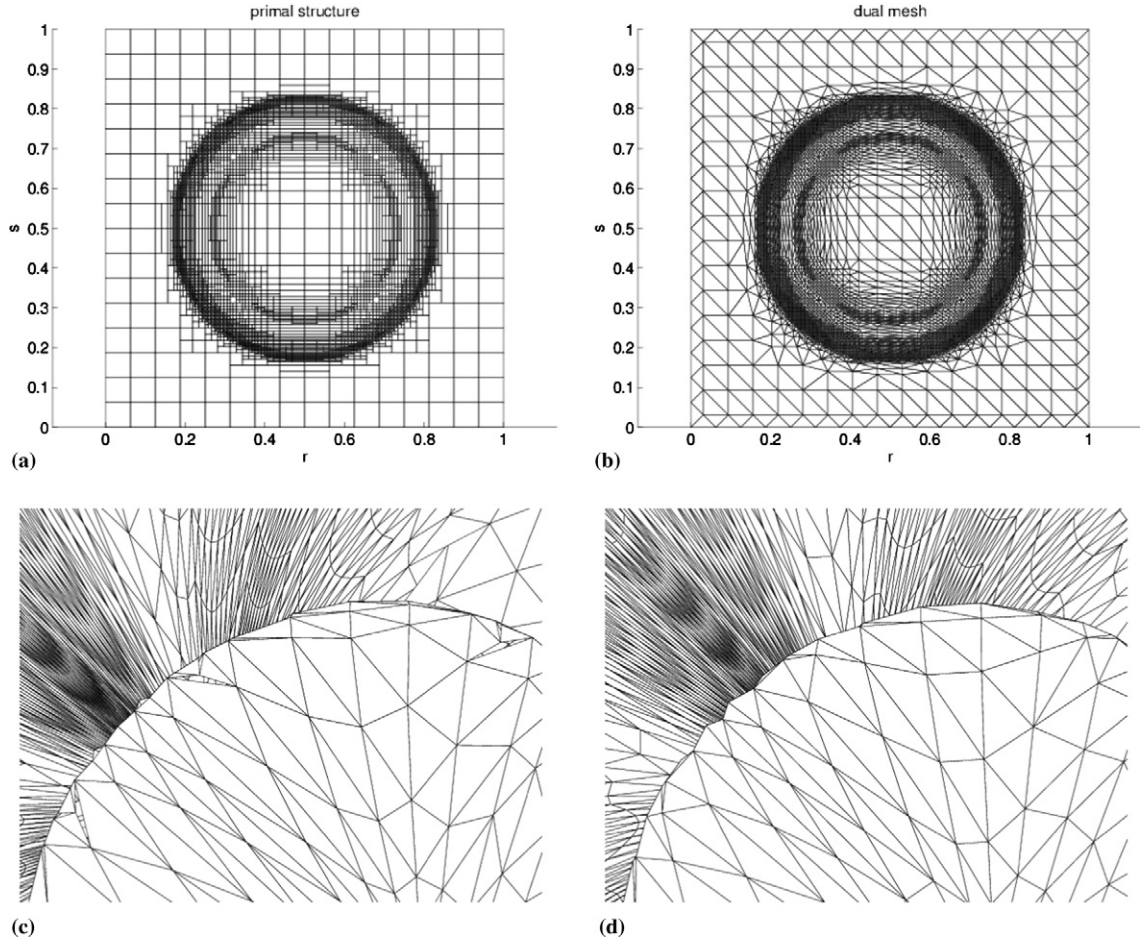


Fig. 2. Primal and dual structure in Example 3 of Section 4 at time $t = 2$. (a) and (b) primal and dual structure in the parametric domain. (c) and (d) zoom in on the mesh. The primal mesh has discontinuities ($\perp$ junctions) while the dual mesh is tight.

We conclude this short review by pointing out that there are other approaches to derive (3.3) from (1.4) by the detour of particle density functions in phase space and means of the Wigner transform [14,27,28,32].

3.2. Phase map construction

We assume in this section that $c(\boldsymbol{x})$ is a periodic function with period $[0,1]^d$. This assumption is valid in many applications (e.g. wave guide, micro structures) and will be relaxed later. Since (3.3) describes the Hamiltonian dynamics, $H(\boldsymbol{x},\boldsymbol{p}) = c(\boldsymbol{x})|\boldsymbol{p}|$ remains constant along the rays and it is thus possible to factor out the $|\boldsymbol{p}|$ dependence from (3.3). Setting $\boldsymbol{p} = |\boldsymbol{p}|\boldsymbol{v}$ gives a reduced version of the ray equations

$$\frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}t} = c(\boldsymbol{x})\boldsymbol{v}, \qquad \frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}t} = -\nabla c(\boldsymbol{x}) + (\nabla c(\boldsymbol{x}) \cdot \boldsymbol{v})\boldsymbol{v}. \tag{3.4}$$

We define $\boldsymbol{y}$ by $\boldsymbol{y} = (\boldsymbol{x},\boldsymbol{v})$ and will sometimes write (3.4) in the compact form

$$\frac{\mathrm{d}\boldsymbol{y}}{\mathrm{d}t} = F(\boldsymbol{y}). \tag{3.5}$$

In this setup, $M = \{(\boldsymbol{x},\boldsymbol{v}) \in \mathbf{R}^d \times \mathbf{S}^{d-1}\}$ is a smooth invariant manifold, and one is then in the position of applying the phase flow method.

Our discussion here focuses on the three components of the phase flow method: the discretization of M, the ODE integration rule and the local interpolation scheme.

*2*D. We parameterize the unit vector $\boldsymbol{v}$ with the angular variable θ and rewrite (3.4) as

$$\frac{\mathrm{d}x}{\mathrm{d}t} = c(x,y)\cos\theta, \qquad \frac{\mathrm{d}y}{\mathrm{d}t} = c(x,y)\sin\theta, \qquad \frac{\mathrm{d}\theta}{\mathrm{d}t} = c_x(x,y)\sin\theta - c_y(x,y)\cos\theta \tag{3.6}$$

with $M = \mathbf{R}^2 \times [0,2\pi)$. Since $c(\boldsymbol{x})$ is periodic, the dependence of g_t on the $\boldsymbol{x}$ variable is also periodic. Therefore, even though M is not compact, we only need to approximate the restriction of g_t on the fundamental domain $[0,1)^2 \times [0,2\pi)$ as the behavior of g_t elsewhere can be obtained by simple translation. We then simply discretize $[0,1)^2 \times [0,2\pi)$ by means of a uniform Cartesian grid. The fourth-order Runge–Kutta method is used as the ODE integrator for the initial step. Because $M = \mathbf{R}^2 \times [0,2\pi)$, no projection is necessary.

Instead of interpolating the phase map $g_t(\boldsymbol{y})$, we construct an interpolant for the shift $g_t(\boldsymbol{y}) - \boldsymbol{y}$ which is periodic. Hence, the problem simplifies to interpolating a periodic function on a uniform Cartesian grid. We use cardinal spline interpolation, which has h-independent interpolation norm [7], as our local scheme.

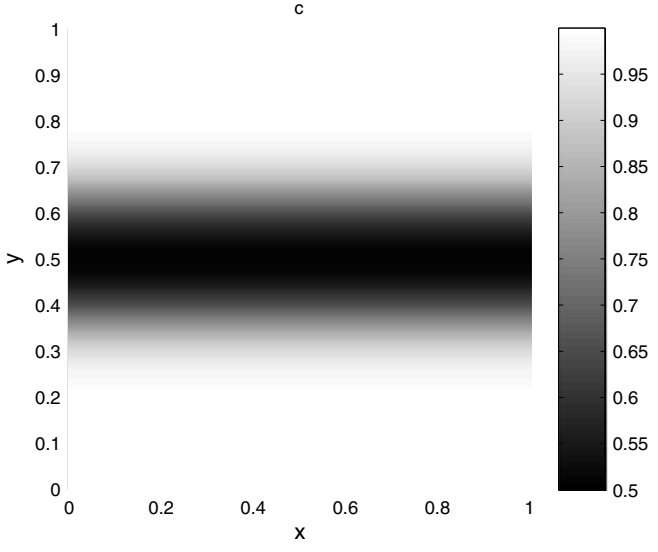


Fig. 3. The wave speed in Example 1.

Table 1
The error ε_{T_0} for different choices of initial grids and T_0

Discretization vs. T_0	0.0625	0.125	0.25	0.5
(16, 32)	4.991e−04	1.034e−03	2.316e−03	5.252e−03
(32, 64)	2.301e−05	4.563e−05	8.344e−05	3.787e−04
(64, 128)	1.274e−06	2.759e−06	5.195e−06	7.343e−06
(128, 256)	1.133e−07	1.755e−07	3.901e−07	6.016e−07

The rows indicate the number of gridpoints used to sample the reduced invariant manifold $M = [0,1]\times[0,2\pi]$, i.e. (16, 32) means that 16 and 32 points are used in the y and θ directions respectively. Each column corresponds to a different value of T_0.

Given the values of g_t on the Cartesian grid, we first build the interpolant in tensor product cardinal B-spline form. This is here equivalent to a simple deconvolution problem [7] which can be handled efficiently by FFTs. Once the interpolant is available in cardinal B-spline form, interpolating the function values at any fix point $\boldsymbol{y}$ has constant complexity.

3D. The invariant manifold M is now equal to $\mathbf{R}^3 \times \mathbf{S}^2 \subset \mathbf{R}^6$. For the same reason as in 2D, we only need to construct the interpolant on the domain $[0,1)^3 \times \mathbf{S}^2$. We discretize the unit cube $[0,1)^3$ (the $\boldsymbol{x}$ component) with a uniform Cartesian grid, and use the standard polar coordinates (θ,ϕ) to parameterize the unit sphere $\mathbf{S}^2$ (the $\boldsymbol{v}$ component)

$$\boldsymbol{v}(\theta,\phi) = (\cos\theta\sin\phi, \sin\theta\sin\phi, \cos\phi),$$

$\theta$and ϕ are then discretized with a uniform grid

$$(0, h, \ldots, 2\pi - h) \times (h/2, \ldots, \pi - h/2),$$

where we have assumed that π is a multiple of h. We again use the fourth order Runge–Kutta method as the ODE integration rule. After the ODE integration step, however, we need to reproject $\boldsymbol{v}$ onto $\mathbf{S}^2$. This is done by simply choosing $\boldsymbol{v}/|\boldsymbol{v}|$, the nearest point to $\boldsymbol{v}$ on $\mathbf{S}^2$. It is possible to show that this extra projection step does not affect the order of the ODE integrator.

Viewed as a function of $\boldsymbol{x}$ and of the pair (θ,ϕ), the shift $g_t(\boldsymbol{y}) - \boldsymbol{y}$ is periodic in $\boldsymbol{x}$ and θ. We extend the shift to make it periodic in ϕ as well in the following way. Suppose f is a smooth function on $\mathbf{S}^2$ parameterized with polar coordinates (θ,ϕ), we define its periodic extension $f^e : [0,2\pi)\times[-\pi,\pi) \to \mathbf{R}$ by

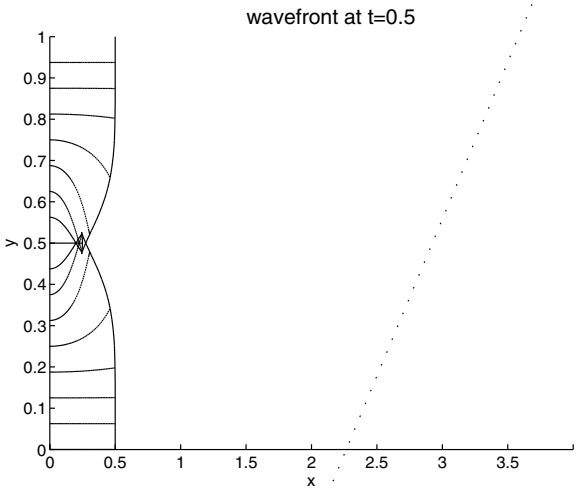
$$f^e(\theta,\phi) = \begin{cases} f(\theta,\phi), & \phi \in [0,\pi), \\ f(\theta+\pi,-\phi), & \phi \in [-\pi,0). \end{cases}$$

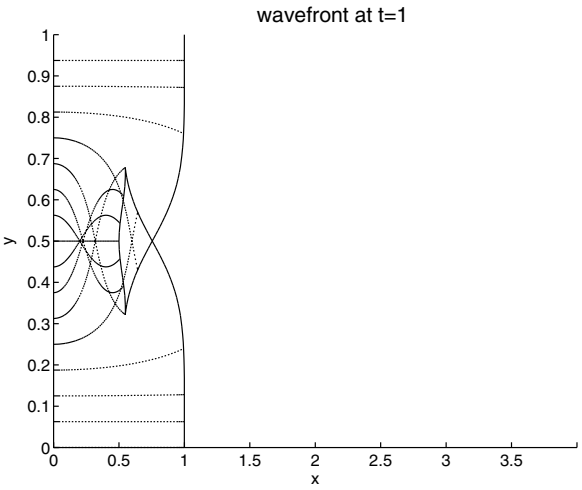
This allows to use the same interpolation procedure as that employed in the 2D example except that one uses the extension of $\tilde{g}_t$ as input.

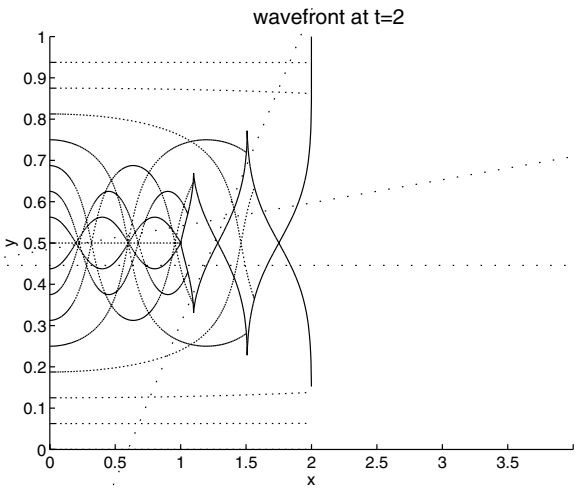
When applying Algorithm 2 to (3.4), we need to specify the 'mesh size' h. Suppose N is the frequency limit beyond which the magnitude of the Fourier coefficients of $c(\boldsymbol{x})$ is below a prescribed error threshold. Then $2N$ is the Nyquist sampling rate required to represent $c(\boldsymbol{x})$ accurately. In practice, we often select h as $1/3N$ or $1/4N$ and the value of T_0 is decided along the construction of the phase maps, see the discussion following Algorithm 2.

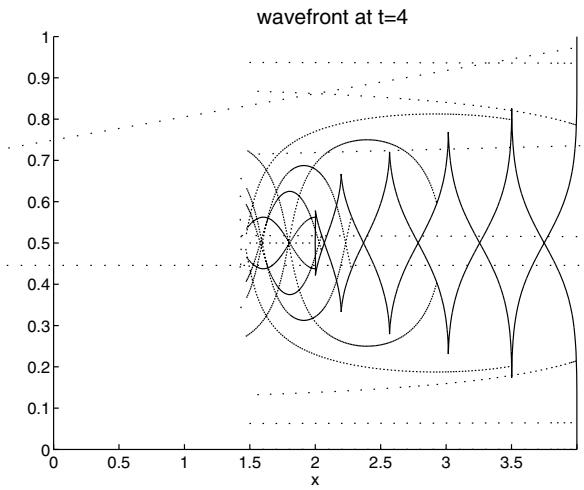
We would like to remark that in the case where $c(\boldsymbol{x})$ is independent of some of the coordinates, the invariant manifold M can be reduced. We refer to numerical examples in Section 4 for details.

The assumption imposing $c(x,y)$ to be periodic can be relaxed, and we illustrate this with two examples. In the first example, the inhomogeneity of the wave speed is restricted to $[0,1]^2$, and $c(x,y)$ is constant (equal to c) elsewhere, see Fig. 1(a). The invariant manifold M is again $\mathbf{R}^2\times\mathbf{S}^1$ but the behavior of g^t is not periodic anymore. However, for a given time T, we are still able to approximate g^T on M efficiently. Define $M_T := [-cT, 1+cT]^2 \times \mathbf{S}^1$ where the subscript indicates the time dependence. Although M_T is not an invariant manifold, for each $t \leqslant T$, g_t is trivial on the complement $\mathbf{R}^2\times\mathbf{S}^1\backslash M_T$ since the wave speed is constant in $\mathbf{R}^2\backslash[0,1]^2$. Therefore, one only needs to approximate the restriction of g_t (for $t \leqslant T$) on M_T instead of on the whole (reduced) phase space $\mathbf{R}^2\times\mathbf{S}^1$. This can be done efficiently by constructing the restriction of $\tilde{g}_{B^k\tau}$ $(k = 1,2,\ldots)$ on M_T by means of the phase flow method. In the second example, $c(x,y)$ is periodic in the x variable and obeys $c(x,y) = c_0$ for $y < 0$ and $c(x,y) = c_1$ for $y > 1$, see Fig. 1(b). For the same reason as before,









wavefront at t=16

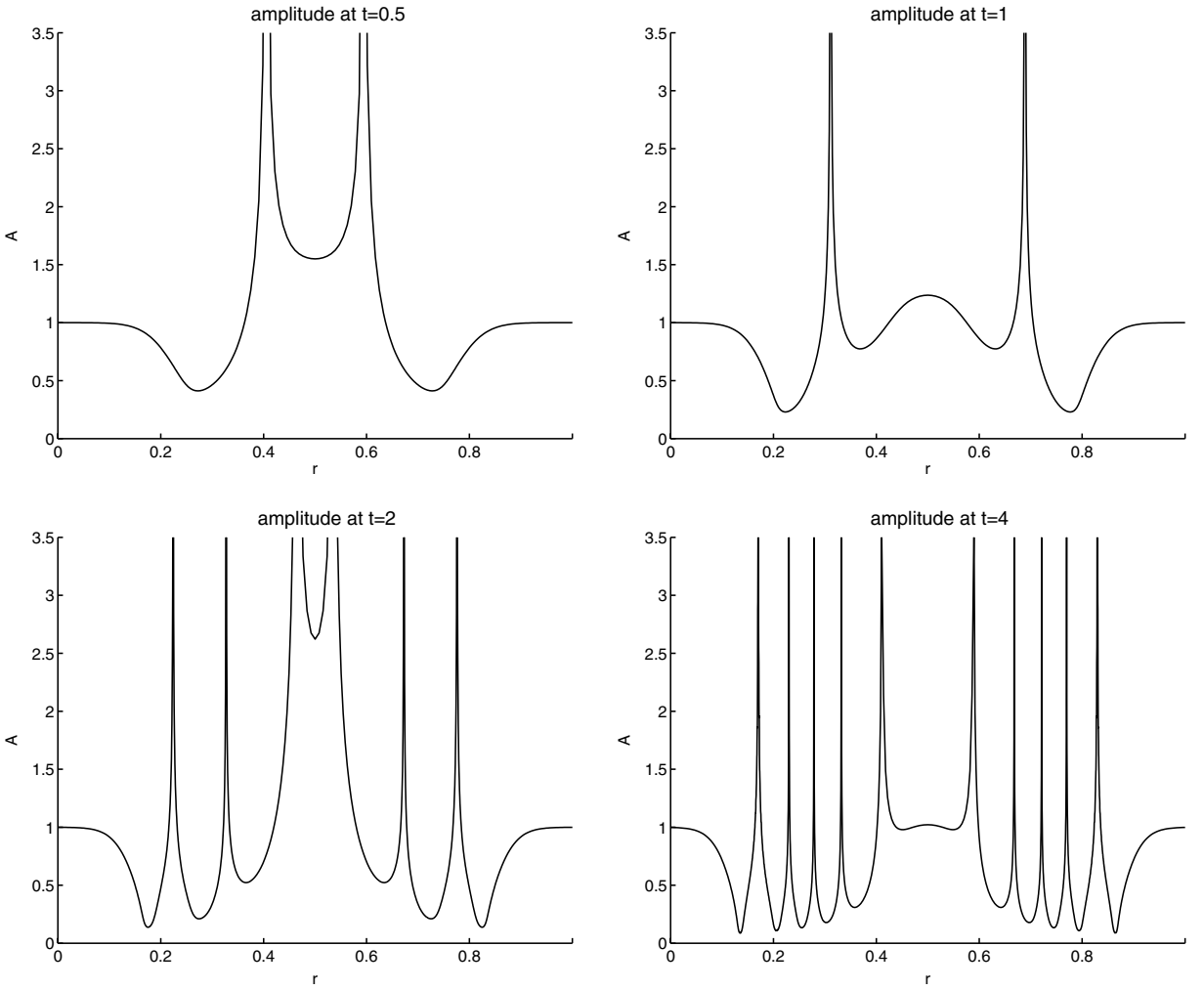


Fig. 5. Computed wave amplitude along the wave front (Example 1). The four frames show the amplitude at times $t = 0.5$, 1, 2 and 4. The horizontal axis is the parameterization variable. The positions of the singularities coincide with the locations of the caustic points in the first four frames of Fig. 4.

given a time T, we only need to approximate g_t for $t \leqslant T$ on $M_T = [0,1] \times [-c_0 T, 1 + c_1 T] \times \mathbf{S}^1$ since elsewhere, g_t is again trivial.

3.3. Wave front construction

In 2D, we shall consider one-dimensional wave front curves (taking values in M) parameterized by a real variable r, and with initial value $\boldsymbol{y}_0 = (\boldsymbol{x}_0, \boldsymbol{v}_0)$. Whenever convenient, we shall use the following notations to emphasize the dependence of the wave front curve on r

$$\boldsymbol{y}(t,r) := \boldsymbol{y}(t, \boldsymbol{y}_0(r)), \qquad \boldsymbol{x}(t,r) := \boldsymbol{x}(t, \boldsymbol{y}_0(r)), \qquad \boldsymbol{v}(t,r) := \boldsymbol{v}(t, \boldsymbol{y}_0(r)).$$

In 3D, we will consider two-dimensional wave front surfaces parameterized by the pair of variables (r,s). The corresponding notations are

$$\boldsymbol{y}(t,r,s) := \boldsymbol{y}(t, \boldsymbol{y}_0(r,s))$$

and likewise for $\boldsymbol{x}$ and $\boldsymbol{v}$. Finally, we use $\tilde{\boldsymbol{y}}$, $\tilde{\boldsymbol{x}}$ and $\tilde{\boldsymbol{v}}$ to denote numerical approximations to their corresponding quantities.

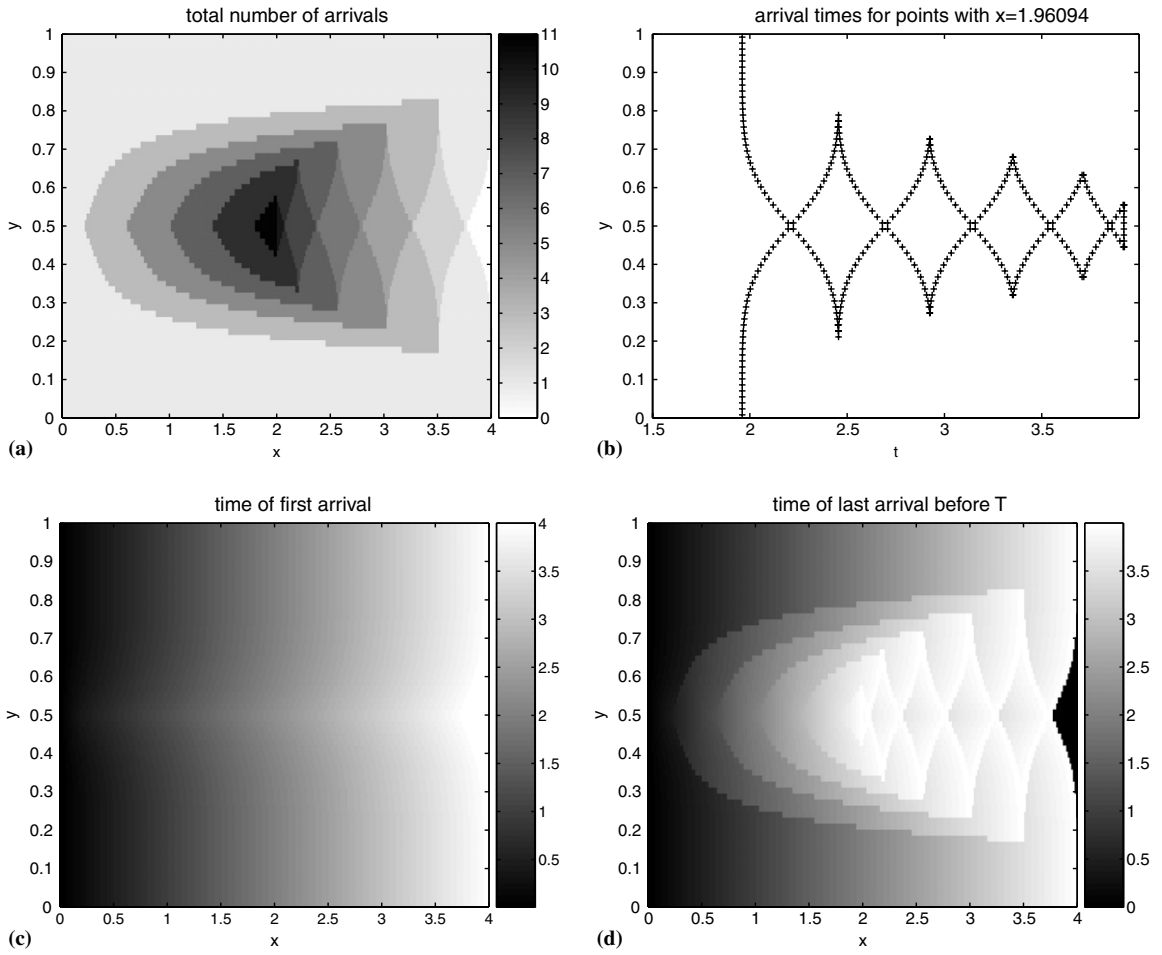


Fig. 6. Multiple arrival times in Example 1. The arrival information is sampled on a 256×64 Cartesian grid. (a) Total number of arrivals. (b) Arrival times at target points with a fixed x value. (c) and (d), first and last arrival times.

Given an initial wave front $\boldsymbol{y}_0$ in 2D, the basic algorithm to construct the wave front time T operates as follows:

- Choose T_0 according to Algorithm 2 and construct $\tilde{g}_{T_0}(\cdot)$.
- Discretize the wave front by sampling $\boldsymbol{y}_0(r)$ at points r_i.
- For each sample r_i, approximate $\boldsymbol{y}(T, r_i)$ with $\tilde{\boldsymbol{y}}(T, r_i) = (\tilde{g}_{T_0})^{(m)}(\boldsymbol{y}_0(r_i))$ where $T = mT_0$.
- Connect $\tilde{\boldsymbol{x}}(T, r_i)$—the spatial component of $\tilde{\boldsymbol{y}}(T, r_i)$—to construct the final wave front.

In the previous algorithm, we assumed a fixed and uniform discretization of the initial wave front. However, for large times T, the wave front may become rather complex and the rays may diverge. Unless the discretization of the initial wave front is extremely fine, uniform discretization may not be able to resolve the final wave front accurately. In a traditional Lagrangian approach, one often inserts new points on the wave front when the resolution deteriorates. Computing the position and slowness vector of these new points from corresponding initial conditions can be quite expensive. Instead, one typically computes these quantities by interpolating the values of nearby markers. Such interpolation procedures, however, are (1) quite complicated because the markers do not belong to a well structured grid, and (2) often have a low order of accuracy. In addition, interpolating the amplitude is certainly nontrivial.

We propose below an algorithm which adaptively discretizes the wave front based on its inherent complexity. In this adaptive approach, the position, slowness and even amplitude of the inserted points can be computed

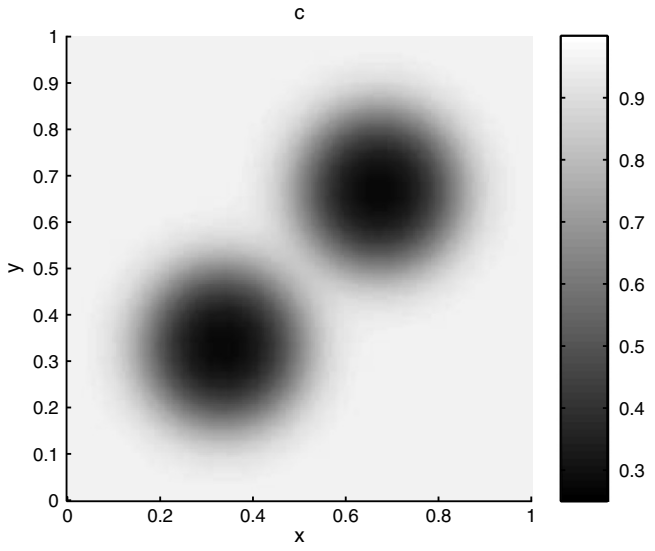


Fig. 7. The wave speed in Example 2.

directly from the initial conditions in O(1) operations (this is again one of the benefit of the phase flow method). Therefore, no complicated unstructured grid interpolation is required. We assume without loss of generality that the initial wave front is parameterized by a variable $r \in [0,1]$.

Algorithm 3 (*2D adaptive wave front construction*)

- Choose T_0 and construct $\tilde{g}_{T_0}$.
- Choose a tolerance λ. Discretize the initial wave front with samples $R = \{r_i\}$. The initial parameter domain $[0,1]$ is now partitioned into intervals and put $\tilde{\boldsymbol{y}}(0, r_i) = \boldsymbol{y}_0(r_i)$. We require that each pair of adjacent samples obeys $|\boldsymbol{y}_0(r_i) - \boldsymbol{y}_0(r_j)| \leqslant \lambda$.
- For $k = 1, \ldots, T/T_0$
  - For any $r_i \in R$, $\tilde{\boldsymbol{y}}(kT_0, r_i) = \tilde{g}_{T_0}(\tilde{\boldsymbol{y}}((k-1)T_0, r_i))$.
  - For each pair of adjacent samples (r_i, r_j), insert new samples $\{r_\ell\}$ evenly distributed in $[r_i, r_j]$ if $|\tilde{\boldsymbol{y}}(kT_0, r_i) - \tilde{\boldsymbol{y}}(kT_0, r_j)| > \lambda$. The number of new samples is equal to $\lceil |\tilde{\boldsymbol{y}}(kT_0, r_i) - \tilde{\boldsymbol{y}}(kT_0, r_j)|/\lambda \rceil$. The interval $[r_i, r_j]$ is thus partitioned into smaller intervals. The values $\tilde{\boldsymbol{y}}(kT_0, r_\ell)$ at the new points $\{r_\ell\}$ are computed by $\tilde{\boldsymbol{y}}(kT_0, r_\ell) = (\tilde{g}_{T_0})^{(k)}(\boldsymbol{y}_0(r_\ell))$.
- Connect $\tilde{\boldsymbol{x}}(T, r_i)$ for $r_i \in R$ to obtain the final wave front.

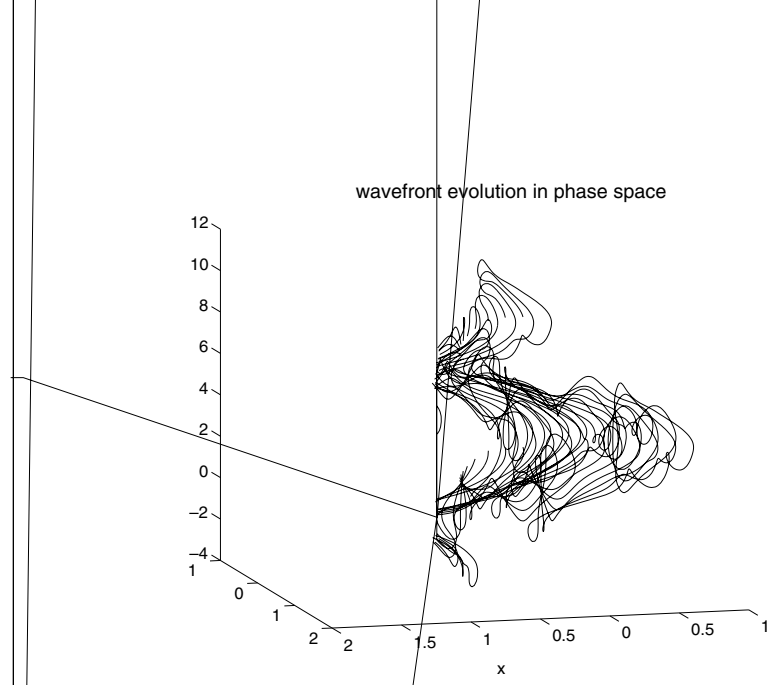
The algorithm in 3D is similar. Assuming the initial wave front is parameterized by $(r,s) \in [0,1] \times [0,1]$, the initial wave front is partitioned into patches, each of which is parameterized by a rectangular region in the (r,s) space. The diameter of each patch is less than λ. During the wave front construction, we maintain the wave front as a union of patches and keep track of the approximate solution $\tilde{\boldsymbol{y}}$ at the corners of the patches. Whenever the diameter of a patch is larger than λ, the patch is split in the parametric domain into smaller patches. The values at the corners of the new patches are computed from the initial value using $\tilde{g}_{T_0}$. A typical partition of the parametric domain is shown in Fig. 2(a).

If one only needs the spatial component of the wave front, one can replace the splitting rule

$$|\tilde{\boldsymbol{y}}(kT_0, r_i) - \tilde{\boldsymbol{y}}(kT_0, r_j)| > \lambda$$

with

$$|\tilde{\boldsymbol{x}}(kT_0, r_i) - \tilde{\boldsymbol{x}}(kT_0, r_j)| > \lambda.$$



Note that such a modification would cause problems in a standard Lagrangian approach. There, one needs to maintain an adequate approximation of the wave front in phase space for accurate interpolation. Inserting samples when there is a significant variation in the $\boldsymbol{x}$ component may not be capable of capturing the directional variation (the $\boldsymbol{v}$ component) of the wave front near the caustics, thus providing poor results when interpolating the wave front information at the newly inserted samples. In contrast, our approach is immune to such issues since no interpolation is used.

3.4. Amplitude computations

We are now concerned with the computation of the wave amplitudes along the rays. This requires additional information about the shape of the ray source as we now explain. Put A_0 to be the leading amplitude function in (1.5). Then the following relationships:

$$\frac{A_0(\boldsymbol{x}(t,r))}{A_0(\boldsymbol{x}(0,r))} = \sqrt{\frac{|\partial_r \boldsymbol{x}(0,r)|}{|\partial_r \boldsymbol{x}(t,r)|} \cdot \frac{c(\boldsymbol{x}(t,r))}{c(\boldsymbol{x}(0,r))}}, \tag{3.7}$$

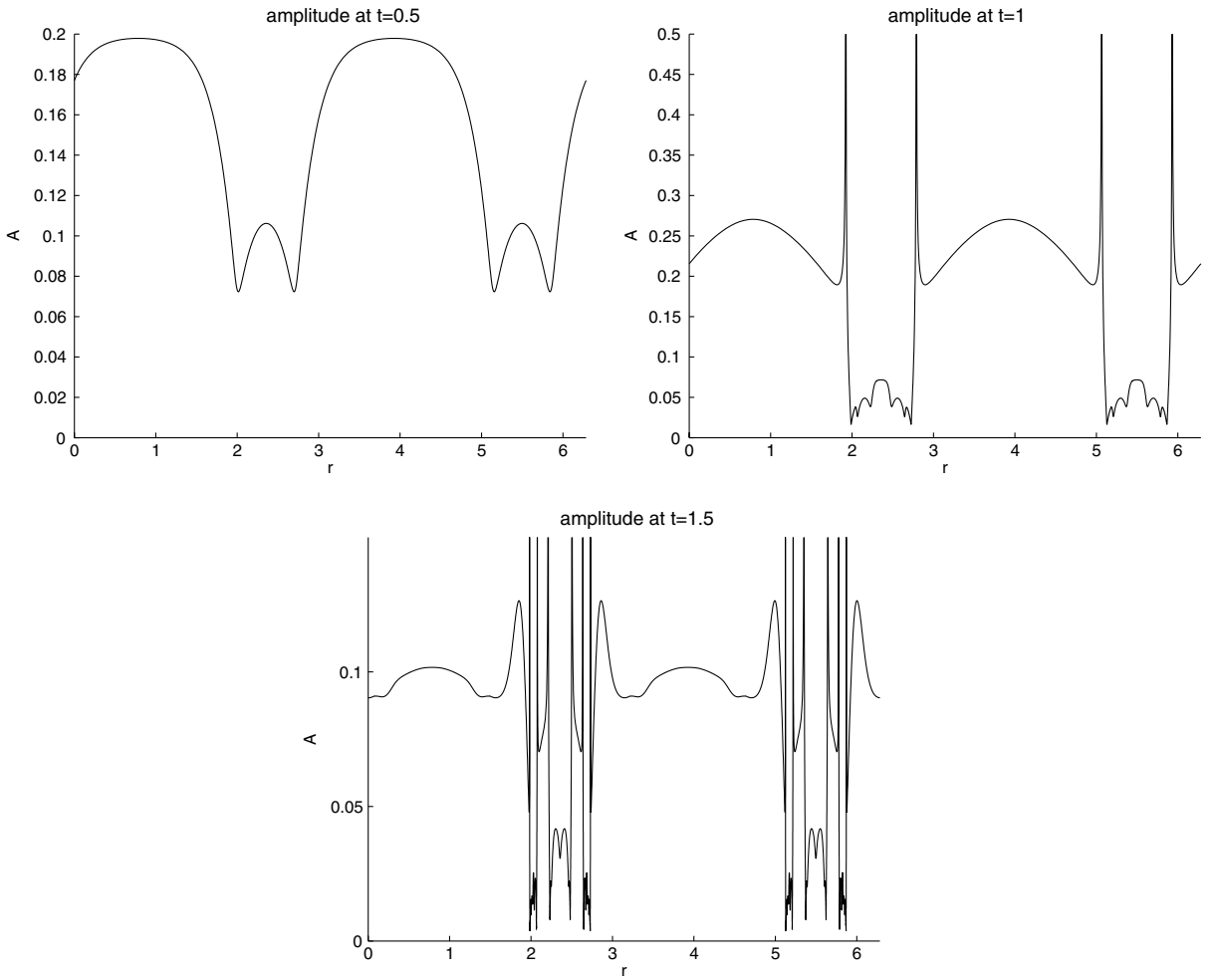


Fig. 9. Amplitude computation results (Example 2). The three frames show the amplitude at times $t = 0.5$, 1 and 1.5. The parameterization is that of the initial wave front. The spikes of the amplitude functions are collocated with the caustic points in Fig. 8.

$$\frac{A_0(\boldsymbol{x}(t,r,s))}{A_0(\boldsymbol{x}(0,r,s))} = \sqrt{\frac{|\partial_r \boldsymbol{x}(0,r,s) \times \partial_s \boldsymbol{x}(0,r,s)|}{|\partial_r \boldsymbol{x}(t,r,s) \times \partial_s \boldsymbol{x}(t,r,s)|} \cdot \frac{c(\boldsymbol{x}(t,r,s))}{c(\boldsymbol{x}(0,r,s))}} \tag{3.8}$$

hold in 2D and 3D respectively. The additional information we need to track are the spatial derivatives of the rays' sources, namely, $\partial_r \boldsymbol{x}(t,r)$ in 2D, and $\partial_r \boldsymbol{x}(t,r,s)$ and $\partial_s \boldsymbol{x}(t,r,s)$ in 3D. Using our compact notations, these quantities can of course be retrieved from the differential matrix $\nabla_{\boldsymbol{b}} \boldsymbol{y}(t,\boldsymbol{b})$. If we can track $\nabla_{\boldsymbol{b}} \boldsymbol{y}(T,\boldsymbol{b})$ for any $\boldsymbol{b}$, then we can easily compute A_0 using (3.7) and (3.8). The key point is that one can use the phase flow method to compute the flow $\nabla_{\boldsymbol{b}} \boldsymbol{y}(T,\boldsymbol{b})$.

Indeed, differentiating (3.5) with respect to the initial condition gives the following evolution equation for $\nabla_{\boldsymbol{b}} \boldsymbol{y}(t,\boldsymbol{b})$

$$\frac{\mathrm{d}\nabla_{\boldsymbol{b}} \boldsymbol{y}(t,\boldsymbol{b})}{\mathrm{d}t} = \nabla_{\boldsymbol{y}} F(\boldsymbol{y}(t,\boldsymbol{b})) \cdot \nabla_{\boldsymbol{b}} \boldsymbol{y}(t,\boldsymbol{b}), \quad \nabla_{\boldsymbol{b}} \boldsymbol{y}(0,\boldsymbol{b}) = I.$$

We use the phase flow method to compute the derivative of the phase map $G_t(\boldsymbol{b}) = \nabla_{\boldsymbol{b}} \boldsymbol{y}(t,\boldsymbol{b}) = \nabla_{\boldsymbol{b}} g_t(\boldsymbol{b})$. To achieve this, we approximate $G_t(\boldsymbol{b})$ along with $g_t(\boldsymbol{b})$ in Algorithms 1 and 2. The group relationship now reads

$$G_{2t}(\boldsymbol{b}) = G_t(g_t(\boldsymbol{b})) \cdot G_t(\boldsymbol{b}).$$

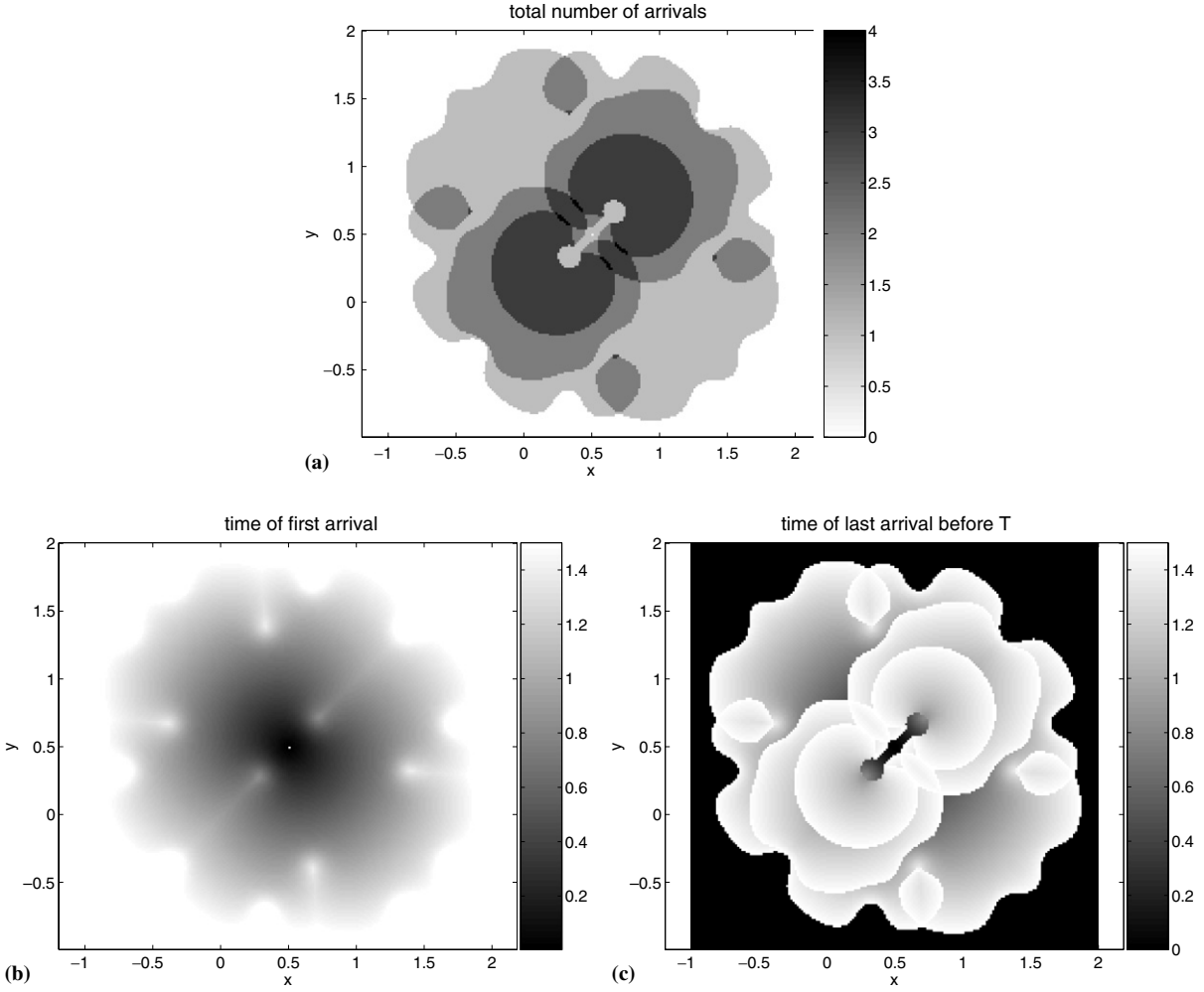


Fig. 10. Computed arrival times in Example 2: (a) total number of arrivals; (b) and (c), first and last time of arrival. The total number arrivals near the center is zero since the initial wave front is an expanding circle of radius 0.01 at $(1/2, 1/2)$.

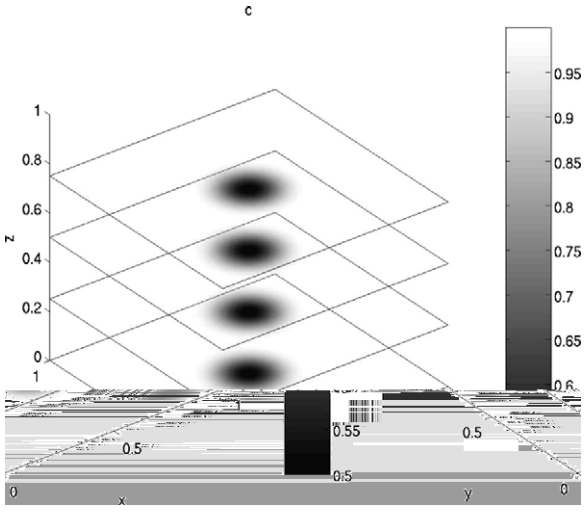


Fig. 11. Cross-section view of the wave speed in Examples 3 and 4.

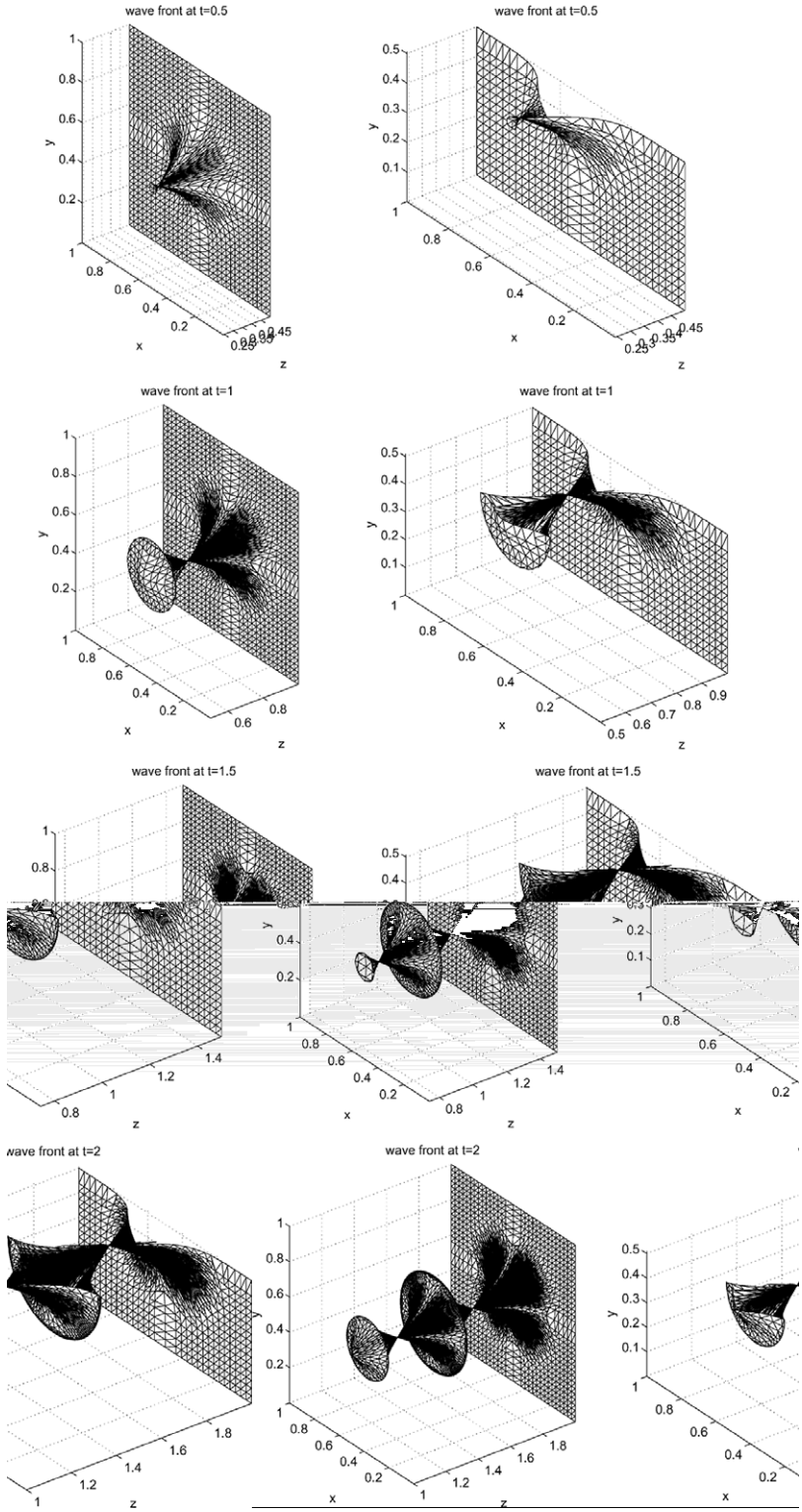


Fig. 12. Computed wave front in Example 3 (3D waveguide). The four rows show the wave front at times $t = 0.5$, 1, 1.5 and 2. In each row, the left frame plots the whole wave front, while the right frame only displays the lower half in order to show the interior. The mesh used to approximate the wave front is not uniform since the approximation is adaptively refined as the wave front evolves.

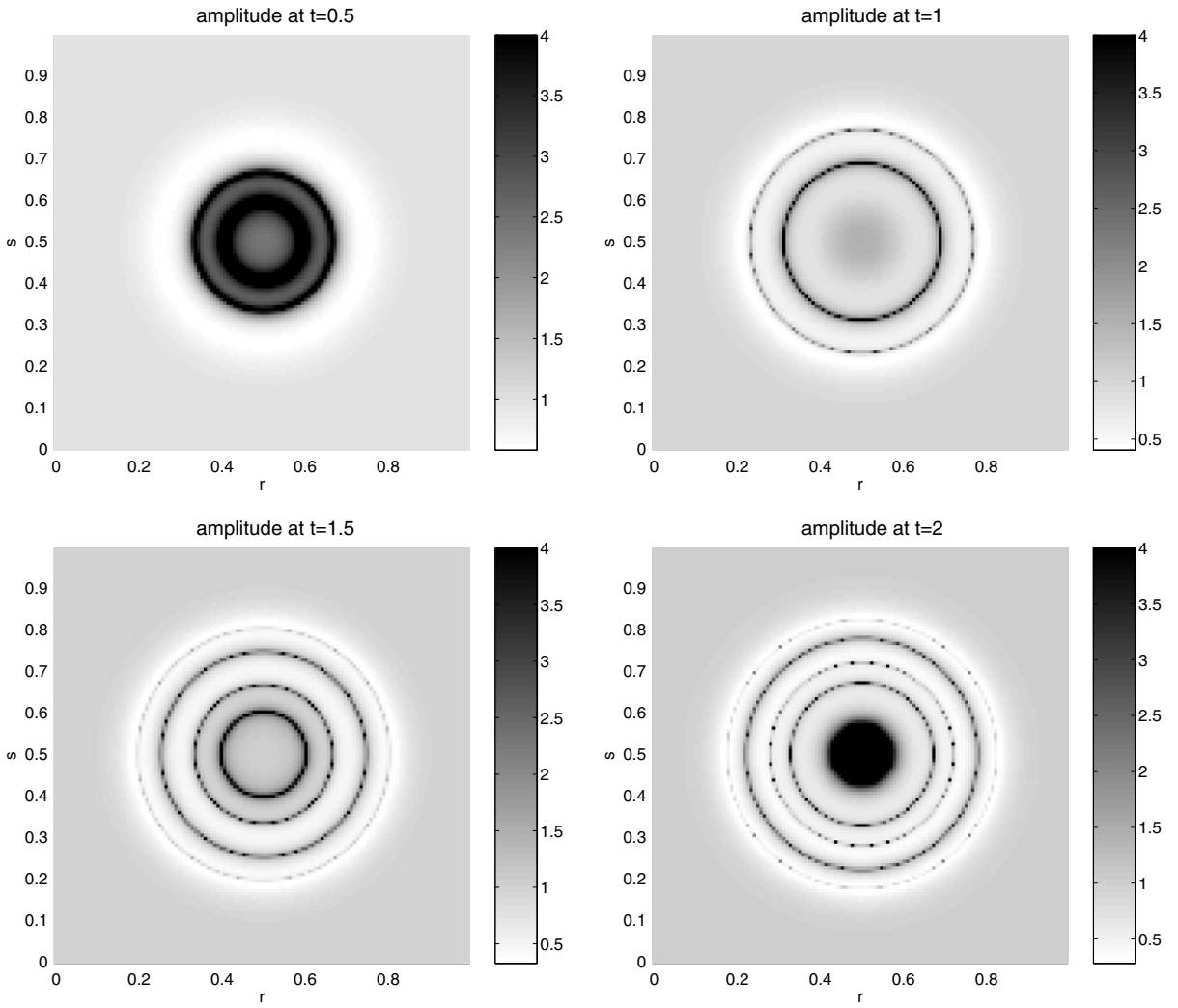


Fig. 13. Computed amplitude in Example 3. The four frames display the amplitude function with respect to the parameterization of the initial wave front at times $t = 0.5$, 1, 1.5 and 2. The positions of the singularities are collocated with the caustic rings in Fig. 12. The singularities are not fully resolved because of the low sampling rate.

Algorithmically, we compute $\widetilde{G}_{2t}(\boldsymbol{b})$, the numerical approximation to $G_{2t}(\boldsymbol{b})$, using

$$\widetilde{G}_{2t}(\boldsymbol{b}) = \widetilde{G}_t(\tilde{g}_t(\boldsymbol{b})) \cdot \widetilde{G}_t(\boldsymbol{b}).$$

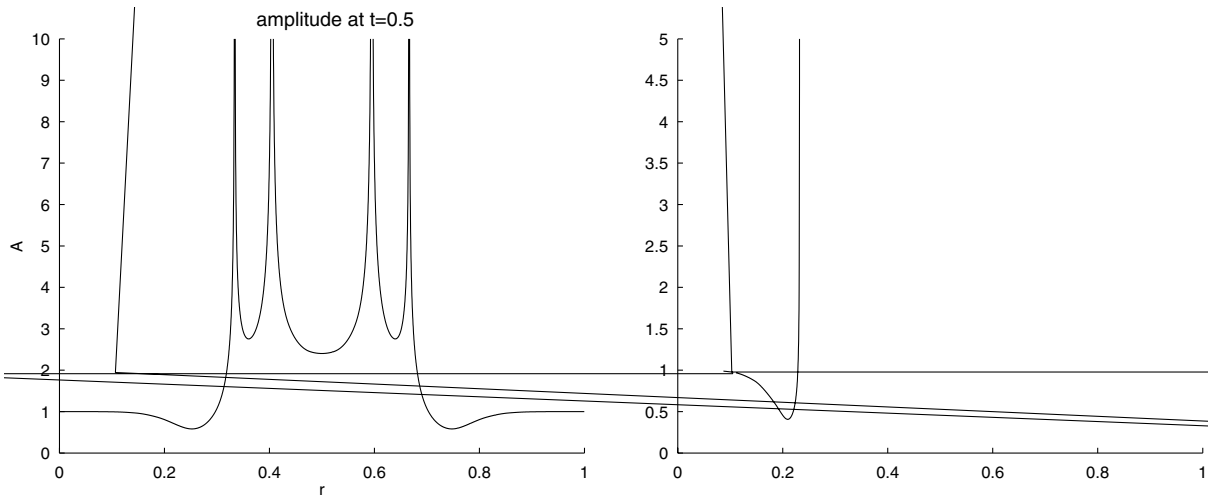
When $\tilde{g}_t(\boldsymbol{b})$ is not a gridpoint, $\widetilde{G}_t(\tilde{g}_t(\boldsymbol{b}))$ is evaluated by means of interpolating the values $\widetilde{G}_t$ at nearby gridpoints.

To be complete, we note that the first derivative of F in our problem (3.4) is given by

$$\nabla_y F(\boldsymbol{y}) = \begin{pmatrix} \boldsymbol{v}\nabla c^{\mathrm{T}} & cI \\ -\nabla^2 c + \boldsymbol{v}\boldsymbol{v}^{\mathrm{T}}\nabla^2 c & (\nabla c \cdot \boldsymbol{v})I + \boldsymbol{v}\nabla c^{\mathrm{T}} \end{pmatrix}.$$

In 2D for example, differentiation with respect to the (x, y, θ) parameterization gives a matrix of the form

$$\begin{pmatrix} c_x \cos\theta & c_y \cos\theta & -c\sin\theta \\ c_x \sin\theta & c_y \sin\theta & c\cos\theta \\ c_{xx}\sin\theta - c_{xy}\cos\theta & c_{xy}\sin\theta - c_{yy}\cos\theta & c_x\cos\theta + c_y\sin\theta \end{pmatrix}.$$



3.5. Multiple arrival times

We now consider the computation of multiple arrival times. We shall primarily be interested in two cases: single source/multiple targets and single source/single target. In 2D, the source is defined as a smooth curve and in 3D, as a smooth surface in the (reduced) phase space. For example, the source can be

$$x_0(r) = 0, \qquad y_0(r) = 0, \qquad \theta_0(r) = 2\pi r$$

for an expanding spherical wave initially centered at the origin or

$$x_0(r) = 0, \qquad y_0(r) = r, \qquad \theta_0(r) = 0$$

for a propagating planar wave. A *target* is a point in the *physical* space.

We define the *trace* of a wave front—from t_0 to t_1—as the union of the wave fronts (in the phase space) for all $t \in [t_0, t_1]$. The trace of a wave front in the 2D setup is a two-dimensional surface in the three-dimensional reduced phase space, while the trace of a wave front in the 3D setup is a three dimensional volume in the five dimensional reduced phase space. A trace is naturally parameterized by (t, r) in 2D and by (t, r, s) in 3D.

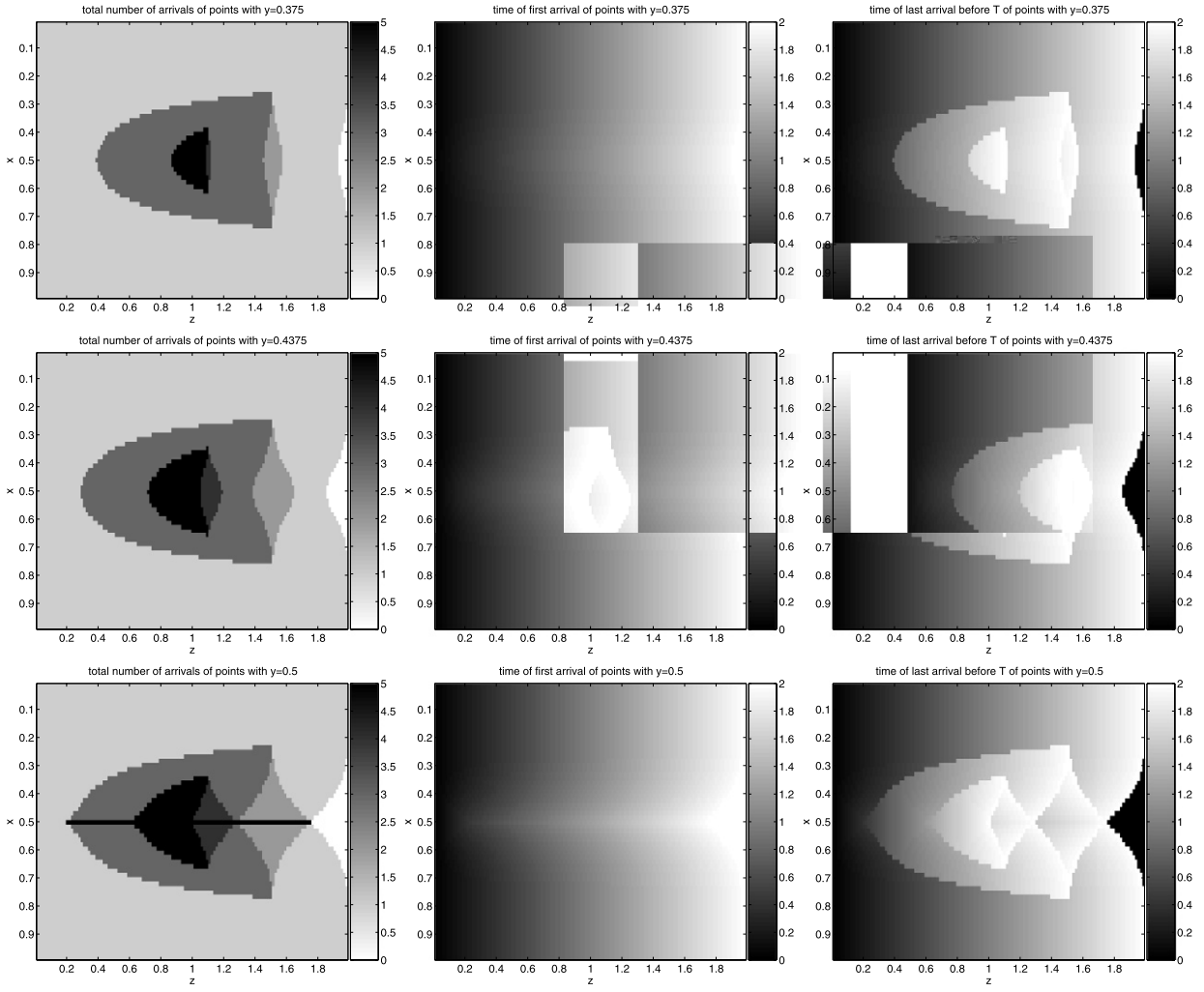


Fig. 15. Computed multiple arrivals in Example 4. The three rows show arrivals at target points with $y = 0.375$, 0.4375 and 0.5 respectively. Each row show the total number of arrivals, and the time of the first and last arrival.

For every target point, the number of wave arrivals is equal to the number of trace points which have the target point as its spatial component. The arrival times are the parameters t of these trace points. Therefore, computing multiple arrival times is equivalent to projecting the trace onto the physical domain. Assuming that one can construct the trace efficiently, this is a simple problem in computational geometry. To simplify the presentation, we assume that $c(\boldsymbol{x})$ is periodic.

Single source/multiple targets. Given a source and a time T, we wish to compute the arrivals generated by the source in the time interval $[0, T]$. Recall that the phase space curve of the source (resp. the trace) is parameterized by $r \in [0,1]$ (resp. $(t,r) \in [0,T] \times [0,1]$).

The parameters ΔT and λ in Algorithm 4 control the accuracy of the approximate trace and thus the accuracy of the computed multiple arrival times. Suppose that we are provided with a prescribed error tolerance ε for the approximation of the trace. Then ΔT and λ are set to be of order $\sqrt{\varepsilon}$ since the linear interpolation we use to approximate the trace is second order accurate.

Algorithm 4 (*2D single source/multiple targets*)

- Construct the wave trace with accuracy ε.
  - Choose a time step $\Delta T = \mathrm{O}(\sqrt{\varepsilon})$ and a tolerance $\lambda = \mathrm{O}(\sqrt{\varepsilon})$.

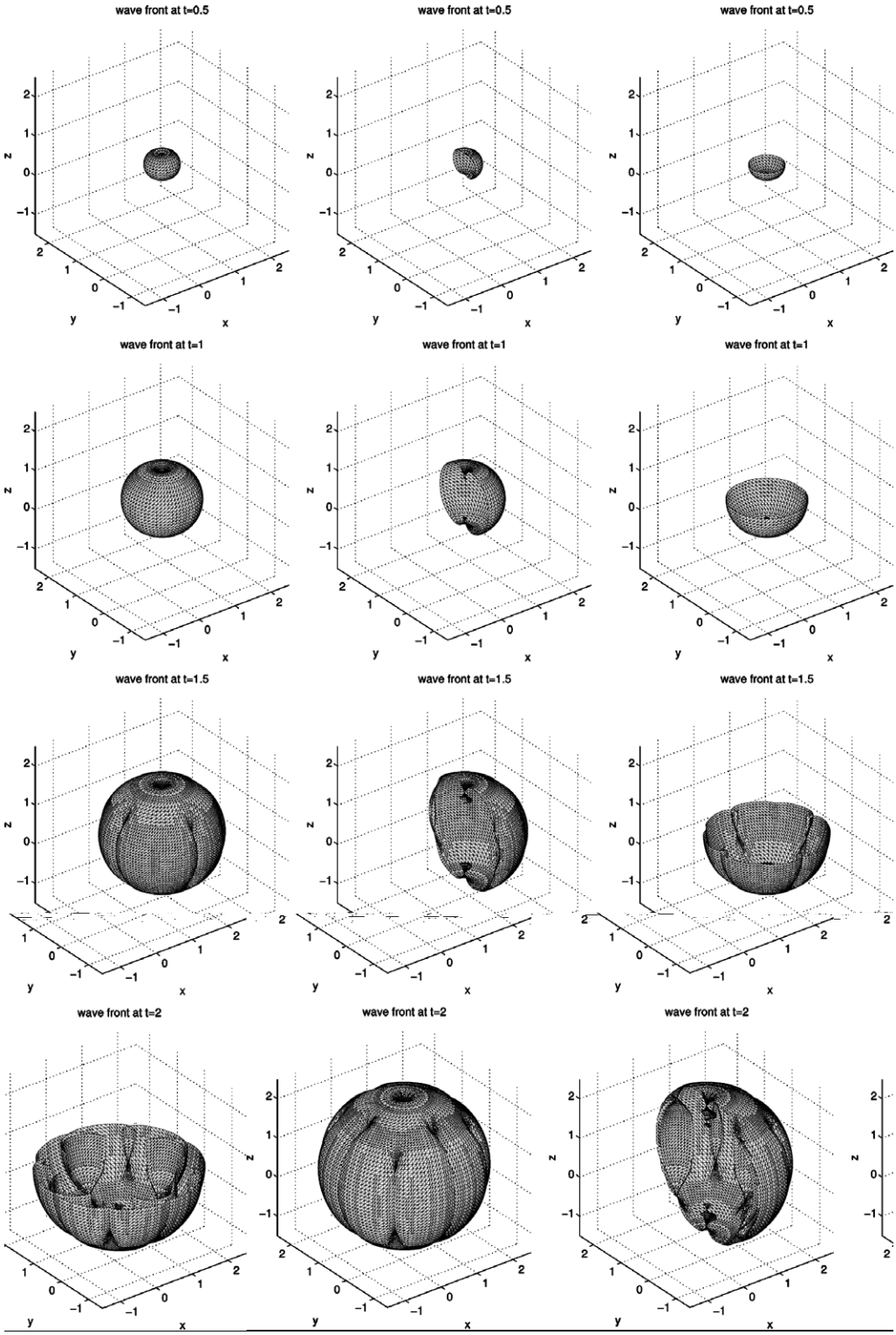


Fig. 16. Computed wave front in Example 4. The four rows show the wave front at times $t = 0.5$, 1, 1.5 and 2. In each row, the left frame plots the full wave front while the middle and right frames slice through the wave front to show the interior. The wave front is resolved adaptively with an increasing number of samples.

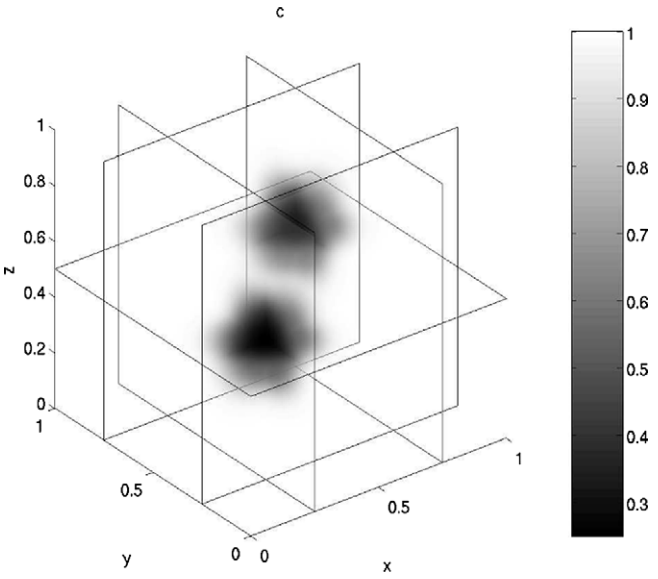


Fig. 17. The 3D wave speed in Example 5.

  - Apply Algorithm 3 to construct the final wave front at time T with time step $T_0 = \Delta T$ and tolerance λ.
  - The samples of the final wave front are parameterized by $\{r_i\}$ and the previous step provides the values $\tilde{\boldsymbol{y}}(k\Delta T, r_i)$ for $0 \leqslant k \leqslant T/\Delta T$. Approximate the trace by linearly interpolating the sampled values $\tilde{\boldsymbol{y}}(k\Delta T, r_i)$. This approximation is represented by a triangle mesh in phase space. The wave evolution is assumed to be linear in every triangle.
- Project the approximate trace $\tilde{\boldsymbol{y}}$ onto physical space (i.e. discard the $\boldsymbol{v}$ component).
- For each target point, we check whether it is covered by nearby projected triangles. If so, the arrival and arrival time are recorded (recall the arrival time is linearly interpolated on each triangle). The nearby triangles can be collected efficiently using a bounding box test, while inside/outside test for each triangle is carried out using the determinant test [6].

Away from the singularities—e.g. caustics, cusps and in general the points where the Jacobian of $\boldsymbol{x}(t,r)$ is degenerate—the one-to-one mapping from (t,r) to $\boldsymbol{x}$ is locally preserved since one uses a linear approximation. Therefore, away from the singularities, the total number of arrivals is computed exactly. Moreover, it follows from the inverse function theorem that the accuracy of the computed arrival times is of the order of ε.

Although the structure of Algorithm 4 is similar to that of the existing ray-tracing type methods for computing multiple arrival information, here, one can choose the time step ΔT to be much larger, and thus have a more efficient algorithm. The reason is that for existing methods, the magnitude of the time step is limited in order to ensure the stable and accurate construction of the wave front. In Algorithm 4, however, the construction of the wave front is decoupled and its accuracy is guaranteed by Algorithm 3. Therefore, the time step is only restricted by the prescribed accuracy ε for the multiple arrivals. As we will see in numerical examples (Section 4), ΔT may be chosen to be fairly large without compromising the accuracy of multiple arrival times computations.

The 3D algorithm has the same structure but is more complicated. Although linear interpolation within each rectangular domain of the (r,s) space provides a close approximation to the trace, the approximate trace is not "tight" in the sense that adjacent patches do not meet continuously along their common edge. This is due to the $\perp$ junctions which appear when the patch is partitioned (see Fig. 2(a)). This may result in errors and inconsistencies when computing multiple arrival times. Our solution constructs a *dual* triangle mesh associated with the *primal* structure. Every rectangle in the primal structure is associated with a vertex in the dual mesh

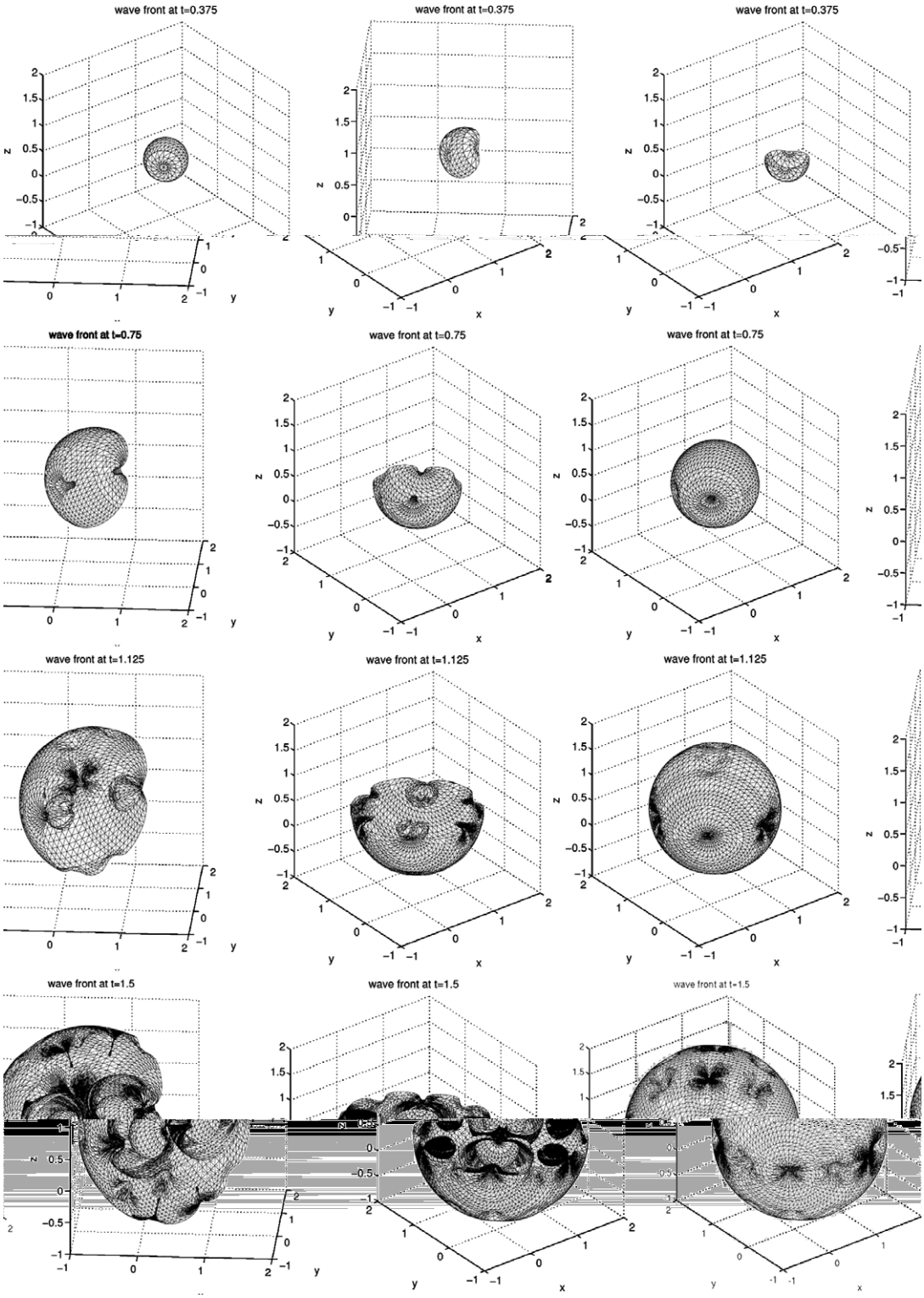


Fig. 18. Computed wave front in Example 5. The four rows show the wave front at times $t = 0.375$, 0.75, 1.125 and 1.5. In each row, the left frame plots the full wave front while the middle and right frames slice through the wave front to show its interior.

located at the center of the rectangle, while every vertex in the primal structure is associated to one or two triangles in the dual mesh. Fig. 2 shows the primal structure and the dual mesh we constructed in Example 3 of Section 4. The dual mesh is guaranteed to be continuous and tight. Moreover, it has the same approximation error as the primal patch structure. Therefore, in the adaptive version of the multiple arrivals algorithm, the trace is constructed by means of the dual mesh. It is further projected onto the physical space to derive the information about multiple arrivals.

Single source/single target. When there is a single target (or just a few targets), it would be wasteful to compute the full trace for $t \in [0,T]$ up to accuracy ε. We propose a more efficient algorithm based on the observation that along the computation, one can a priori discard large portions of the trace with no chance of ever hitting the target.

We make this rigorous and consider the part of the trace parameterized by $(t,r) \in [t_0,t_1]\times[r_0,r_1]$. Let $t_m = (t_0+t_1)/2$ and $r_m = (r_0+r_1)/2$. The distance between $\boldsymbol{y}(t,r)$ and $\boldsymbol{y}(t_m,r_m)$ obeys

$$\begin{aligned}|\boldsymbol{y}(t,r)-\boldsymbol{y}(t_m,r_m)| &\leqslant |\boldsymbol{y}(t,r)-\boldsymbol{y}(t,r_m)| + |\boldsymbol{y}(t,r_m)-\boldsymbol{y}(t_m,r_m)| \\ &\leqslant \mathrm{e}^{Lt}|\boldsymbol{y}(0,r)-\boldsymbol{y}(0,r_m)| + \max_{\boldsymbol{y}}|F(\boldsymbol{y})|\cdot|t-t_m|.\end{aligned}$$

Therefore, for any $(t,r) \in [t_0,t_1]\times[r_0,r_1]$,

$$|\boldsymbol{y}(t,r)-\boldsymbol{y}(t_m,r_m)| \leqslant \mathrm{e}^{Lt_1}\max_{r\in(r_0,r_1)}|\boldsymbol{y}(0,r)-\boldsymbol{y}(0,r_m)| + \max_{\boldsymbol{y}}|F(\boldsymbol{y})|\cdot\frac{t_1-t_0}{2}. \tag{3.9}$$

The trace parameterized by $(t,r) \in [0,T]\times[0,1]$ has a natural hierarchical structure corresponding to a quadtree structure on $[0,T]\times[0,1]$. The basic idea of the algorithm is to throw away large parts of the trace using (3.9) before one even attempts to construct them.

Algorithm 5 (*2D single source/single target*)

- Choose a time step ΔT and assume that $T = 2^n\Delta T$ for some integer n.
- Construct $\tilde{g}_T$ and store the intermediate phase maps $\tilde{g}_{2^k\Delta T}$, $-1 \leqslant k \leqslant n$, using Algorithm 2.
- Put $Q^0 = \{[0,T]\times[0,1]\}$.
- For $k = 1,\ldots,n$
  - Set Q^k to be empty.
  - For each $[t_0,t_1]\times[r_0,r_1]$ in Q^{k-1}, perform the following test: if the distance between the target and the spatial component of $\tilde{\boldsymbol{y}}\left(\frac{t_0+t_1}{2},\frac{r_0+r_1}{2}\right)$ is less than

    $$\mathrm{e}^{Lt_1}\max_{r\in(r_0,r_1)}\left|\boldsymbol{y}(0,r)-\boldsymbol{y}\left(0,\frac{r_0+r_1}{2}\right)\right| + \max_{\boldsymbol{y}}|F(\boldsymbol{y})|\cdot\frac{t_1-t_0}{2},$$

    then split $[t_0,t_1]\times[r_0,r_1]$ dyadically into four parts and put them into Q^k. Here $\tilde{\boldsymbol{y}}\left(\frac{t_0+t_1}{2},\frac{r_0+r_1}{2}\right)$ is computed by composing the appropriate maps $\tilde{g}_{2^k\Delta T}$ for $k = -1,\ldots,n$.
- For each $[t_0,t_1]\times[r_0,r_1]$ in Q^n, use linear interpolation to approximate the part of the trace parameterized by $[t_0,t_1]\times[r_0,r_1]$ and project the triangles (obtained from linear interpolation) onto the physical space. If the triangles cover the target point, then record the arrival and the arrival time.

Two features make this algorithm highly efficient: first, as we pointed out earlier, parts of the trace far away from the target are rejected and not constructed; and second, $\tilde{\boldsymbol{y}}\left(\frac{t_0+t_1}{2},\frac{r_0+r_1}{2}\right)$ is computed in $O(\log n)$ operations by composing the maps $\tilde{g}_{2^k\Delta T}$.

The extension to the 3D case is direct. We simply replace the quadtree structure on $(t,r) \in [0,T]\times[0,1]$ with an octree tree structure on $(t,r,s) \in [0,T]\times[0,1]\times[0,1]$.

4. Numerical results

We present a few examples to demonstrate the performance of the algorithms we introduced in Section 3. The inhomogeneous wave speeds $c(\boldsymbol{x})$ are here periodic in $[0,1]^d$. All of our examples involve caustics. The

algorithms are implemented in MATLAB. It is well known that a careful implementation in C or Fortran would typically yield a significant speedup and, therefore, for the timing results, we focus on the speedup over standard ray tracing methods. All the computations were performed on a desktop computer with a 2.6 GHz CPU and 1 GB of memory.

Example 1. The first example is a 2D waveguide whose wave speed is given by

$$c(x,y) = \frac{1}{1+\mathrm{e}^{-64(y-1/2)^2}}.$$

Because the exponential function decays rapidly, one can view $c(\boldsymbol{x})$ as periodic over $[0,1]^2$ (see Fig. 3). Further, since $c(x,y)$ is independent of x, the invariant manifold M can be reduced to a two dimensional structure with only the y and θ components. The initial wave front is a plane wave at $x=0$ and the wave propagates in the positive x direction: the parameterization of the initial wave front is then $\boldsymbol{x}_0(r)=(0,r)$ and $\theta_0(r)=0$. The initial amplitude is constant and equal to 1.

In Table 1, we report the error behavior of the approximate phase map. The initial time step τ used in the construction of $\tilde{g}_{T_0}$ is set at 2^{-10}. The error is estimated as follows: we first choose N (set to 200 in this case) random samples $\{\boldsymbol{y}_i\}$ from M; "exact" solutions $\boldsymbol{y}(T_0,\boldsymbol{y}_i)$ are computed by MATLABs adaptive ODE solver with a prescribed error accuracy equal to 10^{-9}; finally, we estimate the error by $\sqrt{\sum_{i=1}^{N}|\tilde{g}_{T_0}(\boldsymbol{y}_i)-\boldsymbol{y}(T_0,\boldsymbol{y}_i)|^2/N}$. The empirical accuracy is consistent with the theoretical estimates of Theorem 2.1.

Fig. 4 illustrates the results of the wave front construction algorithm. In this example, $T_0=0.25$, $\tau=2^{-10}$ and $S=4$. The two dimensional invariant manifold M is discretized uniformly with 64 points in y and 128 points in θ. The construction of the approximate phase map $\tilde{g}_{T_0}$ takes about 2 s. The computed phase map $\tilde{g}_{T_0}$ has about five digits of accuracy (see Table 1). We use Algorithm 3 to propagate the wave front adaptively until time $T=4$. We set the tolerance at $\lambda=1/32$ and the final wave front at $T=4$ has about 600 samples. With $\tilde{g}_{T_0}$ available, it takes 0.064 s to propagate the wave front. For comparison, the ray tracing algorithm using MATLABs ODE solver takes about 0.08 s to trace one ray up to time $T=4$. That is, the timing of a simple ray tracing algorithm would increase by a factor of 750, even with the knowledge of these well-placed 600 samples. (In practice, standard Lagrangian type algorithms would need to guess these locations by inserting rays as explained earlier, thereby increasing complexity.) The results show that our algorithm computes the solution accurately even for (very) large times. We have observed here and also in the following examples that for a fixed choice of T_0 (and hence of $\tilde{g}_{T_0}$) the error of the computed wave front grows linearly with respect to the final time T.

Fig. 5 shows the computed amplitude along the wave front at various times. Here λ is set at 1/64 in order to resolve the singularities in the amplitude. The wave front has about 1200 samples. The four frames here correspond to the first four times in Fig. 4.

We use the single source/multiple targets algorithm (Algorithm 4) to compute the multiple arrivals for $t\in[0,4]$. The target points belong to a 256×64 Cartesian grid $(x,y)\in[0,4]\times[0,1]$. The time step ΔT is equal to 1/16 and the wave front is again resolved adaptively with about 1200 samples. The results are presented in Fig. 6.

Example 2. The wave speed in this example is given by

$$c(x,y) = \frac{1}{1+3\mathrm{e}^{-64((x-1/3)^2+(y-1/3)^2)}+3\mathrm{e}^{-64((x-2/3)^2+(y-2/3)^2)}},$$

where again $c(\boldsymbol{x})$ is regarded as periodic in $[0,1]^2$ (see Fig. 7). The initial wave front is a circle centered at $(1/2,1/2)$ with radius $R=0.01$ and the wave propagates outward. The initial amplitude A is constant equal to one along the wave front. The initial wave front is parameterized with an angular variable $r\in[0,2\pi]$.

Fig. 8 displays the results of the wave front construction algorithm. We choose $T_0=0.25$ and $\tau=2^{-10}$. The invariant manifold M is discretized uniformly with 64 points in x, 64 points in y and 128 points in θ. The construction of the approximate phase map $\tilde{g}_{T_0}$ takes about 120 s. The computed phase map $\tilde{g}_{T_0}$ has an accuracy of about 10^{-5}. We then propagate the wave front adaptively until $T=1.5$. The tolerance λ in

Algorithm 3 is set to be 1/32. The final wave front is resolved with 1400 samples. The wave front construction takes about 0.3 s. For these 1400 samples, ray tracing takes 60 s and, hence, the speed up factor is here about 200.

Fig. 9 plots the amplitude along the wave front at times $t = 0.5$, 1 and 1.5. The tolerance in the adaptive algorithm is $\lambda = 1/128$. The final wave front has 6500 points as to resolve the spikes in the amplitude function which are in one-to-one correspondence with the caustic points in Fig. 8.

We apply Algorithm 4 to compute the arrivals information for $t \in [0, 1.5]$. The target points are fixed and belong to a 192×192 Cartesian grid in $[-1, 2] \times [-1, 2]$. The wave front is sampled adaptively with 3100 points. The results are shown in Fig. 10.

Example 3. This example is a 3D wave guide. The wave speed is

$$c(x, y, z) = \frac{1}{1 + e^{-64((x-1/2)^2+(y-1/2)^2)}},$$

which is again considered to be periodic in $[0, 1]^3$ (see Fig. 11). Since c is independent of z, the invariant manifold M is, therefore, four dimensional. The initial wave front is a plane wave at $z = 0$ and the wave is propagated in the positive z direction. We set the initial amplitude A to be one. The initial wave front is parameterized by two variables r and s with $\boldsymbol{x}(0, r, s) = (r, s, 0)$.

In the wave front construction algorithm, we choose $T_0 = 0.125$ and $\tau = 2^{-10}$. The invariant manifold M is discretized with a Cartesian grid with 24, 24, 48 and 24 points in x, y, θ and ϕ respectively. The phase map $\tilde{g}_{T_0}$ is constructed in about 300 s and the accuracy is of order 10^{-5}. We propagate the wave front using the adaptive algorithm up to $T = 2$. We choose the tolerance λ to be 1/16, and the final wave front is resolved with about 12 000 sampled values. After the construction of $\tilde{g}_{T_0}$, the wave front propagation takes less than 9 s. For comparison, a simple ray tracing algorithm would have taken about 360 s even with the knowledge of well-placed 12 000 samples. Therefore, the speedup factor in this example is around 40. Fig. 12 shows the wave front dynamics.

To display the amplitude information compactly, we discretize the initial wave front with a 128×128 Cartesian grid and compute the evolution of the amplitude function at these points. Fig. 13 shows the amplitude functions with respect to the (r, s) parameterization. In every frame, the circles with high amplitude values are in one-to-one correspondence with the caustic rings in Fig. 12. Because the 128×128 grid is not fine enough to capture the singular behavior of the amplitude function, one can observe aliasing effects. In Fig. 14, we plot the amplitude function for points with parameter value $s = 1/2$. We then sample r at 2000 points and the singular behavior of the amplitude function is accurately captured.

We apply the single source/multiple targets algorithm (Algorithm 4) to compute multiple arrivals for $t \in [0, 2]$, see Fig. 15. The target points belong to a uniform $64 \times 64 \times 128$ Cartesian grid in $[0, 1] \times [0, 1] \times [0, 2]$. The wave front is approximated with a dual mesh with 11 000 samples.

Example 4. The wave speed used in this example is the same as that in Example 3. However, the initial wave front is a sphere centered at $(1/2, 1/2, 1/2)$ with radius $R = 0.01$, and propagating out. We propagate the wave front adaptively until $T = 2$. The tolerance λ in Algorithm 4 is set at 1/12. The final wave front at $T = 2$ is resolved with 32 000 samples. Fig. 16 shows the evolution of the wave front.

Example 5. In this last example, the wave speed is given by

$$c(x, y, z) = \frac{1}{1 + 3e^{-64\cdot((x-1/4)^2+(y-1/4)^2+(z-1/2)^2)} + 3e^{-64\cdot((x-3/4)^2+(y-3/4)^2+(z-1/2)^2)}}$$

which is considered to be periodic in $[0, 1]^3$ (see Fig. 17). This is a full 3D example and the invariant manifold M is five dimensional. The initial wave front is again a sphere centered at $(1/2, 1/2, 1/2)$ and with radius $R = 0.01$. The wave propagates outward.

We choose $T_0 = 0.0625$ and $\tau = 2^{-10}$ in the wave front construction algorithm. The invariant manifold M is discretized with a Cartesian grid with 16, 16, 16, 32 and 16 points in x, y, z, θ and ϕ respectively. The approximate phase map $\tilde{g}_{T_0}$ is constructed in 900 s and its accuracy is around 5×10^{-4}. We propagate the wave

front adaptively until $T = 1.5$ by setting $\lambda = 1/8$ in Algorithm 3. The final wave front is resolved with about 37 000 sampled values, see Fig. 18.

We conclude our discussion of numerical results with a few remarks. In each example, even though M is discretized with only a few dozens of samples in each dimension, the computed phase map $\tilde{g}_{T_0}$ has an approximation error of order 10^{-5} (about 10^{-4} in the last example). Moreover, the error grows linearly with T. This demonstrates that the phase flow method is *practically* accurate and efficient besides being theoretically sound.

When the invariant manifold M is of large dimension, e.g. M is 5D for the 3D ray equations, memory emerges as a limitation simply because one needs to store an adequate discretization of the manifold. One way out is to parallelize the phase flow algorithm, which is straightforward. For the ray equations in 3D, the Cartesian grid may be partitioned into multiple rectilinear domains with possible overlapping regions and with each domain stored in a single processor. Parallel FFTs can then be used to construct the interpolant on each individual domain separately with the help of appropriate boundary conditions [7]. The essential step of the phase flow method is to compute $\tilde{g}_{2t}(\boldsymbol{y}_0) = \tilde{g}_t(\tilde{g}_t(\boldsymbol{y}_0))$ for a gridpoint $\boldsymbol{y}_0$ assuming that $\tilde{g}_t$ is available. To do this, we use $\tilde{g}_t(\boldsymbol{y}_0)$ to identify the processor which contains the interpolant of $\tilde{g}_t(\cdot)$ near the point $\tilde{g}_t(\boldsymbol{y}_0)$. This processor performs the interpolation and the result is sent back to the processor which owns $\boldsymbol{y}_0$.

5. Discussion

This paper introduced a novel approach to solve nonlinear autonomous ODEs on their invariant manifolds. The method computes an approximate phase flow by applying a local integrator for an initial small time step, and uses the group property of the phase flow and high order local interpolation procedures for larger times. As we have seen, the phase flow method is computationally efficient because it "pulls itself up by its bootstrap:" that is, the method makes repeated use of prior computations to calculate the phase map at the next (large) time step. There are related ideas in the literature as we note that repeated squaring strategies have been proposed to compute large powers of matrices, compare algorithms for computing matrix exponentials for example. Indeed, suppose we wish to compute a matrix A raised at a large power n of the form $n = 2^J$. Then a possible strategy is to define $A_0 = A$ and introduce the recurrence $A_{j+1} = A_j^2$ so that $A_j = A^{2^j}$. When the matrix A and its powers are sparse, this might be computationally attractive [8]. Besides computational efficiency, the phase flow method is also empirically and theoretically highly accurate; in our experiments, we computed phase maps with great precision from surprisingly coarse spatial grids.

As indicated in Section 3.2, the discretization of the invariant manifold depends upon the regularity of the ODEs. In the example of ray equations, the choice of h is governed by the decay rate of the Fourier coefficients of $c(\boldsymbol{x})$. If $c(\boldsymbol{x})$ is highly oscillatory, we are forced to use a fine grid and the phase flow method may or may not be the optimal choice in such settings. We note, however, that in many practical applications such as seismic imaging the wave speed $c(\boldsymbol{x})$—which often results from scale-separation arguments—is in fact quite non-oscillatory, see [30].

A more interesting question is whether the phase flow method extends to piecewise-smooth differential equations; that is, to systems in which the function F in (1.1) is now piecewise smooth or even discontinuous. For example, in the high-frequency wave propagation setup, one would like to be able to handle smooth wave speeds but with discontinuities along piecewise smooth surfaces. As we mentioned earlier, the validity of the phase flow method hinges on the smoothness of the phase maps g_t. The apparent problem is that the regularity of g_t for piecewise smooth functions F is in general unclear. Nevertheless, if g_t itself turns out to be piecewise smooth as well, the phase flow method can be extended provided the following three issues are resolved:

(a) The ODE integrator should integrate piecewise smooth systems with high order accuracy, as it decides the accuracy of $\tilde{g}_\tau$.
(b) One would need an efficient method for predicting the singularities of the phase map g_t at each value of t used in the phase flow method. These singularities partition M into several components such that g_t is smooth within each component.
(c) Finally, one would also need refined interpolation strategies to interpolate g_t accurately within each component.

In practice, the solutions to (b) and (c) can be combined into a single ENO-type interpolation scheme [17,18]. We are currently investigating this important extension and hope to report on our progress in a separate publication.

Finally, our method solves general ordinary differential equations and as such, is expected to find application niches in a variety of different fields. For example, the method may be directly applicable to compute geodesic flows. Suppose we are given a surface and that we wish to compute geodesic curves on that surface. Then, one could apply the phase flow method since it is possible to formulate geodesics as solutions to autonomous ordinary differential equations. Many problems with this flavor exist but there are also less traditional applications where the phase flow method might be useful and we conclude with one such opportunity. There exists a newly invented multiscale system based on parabolic scaling in which basis functions are called *curvelets* and are supported in elongated regions obeying the relation width $\sim$ length^2. This system is remarkable because it provides optimally sparse representations of the solution operators to large classes of wave equations (1.4) [4]. Curvelets are simple waveforms indexed by 'regularly' spaced points in phase space. Now the action of the wave propagator on a curvelet is well-approximated by a rigid motion along the Hamiltonian flow, and in order to compute efficiently the sparse curvelet matrix of the full wave propagator, one would need to know where the significant entries of the matrix would be. That is, one would need to trace as many rays as there are curvelets. Obviously, this is a problem for which the phase flow method is especially well suited.

Acknowledgments

E.C. is partially supported by a Department of Energy grant DE-FG03-02ER25529 and L.Y. is supported by that same grant. The authors are grateful to Laurent Demanet for inspiring conversations related to this project. We would also like to thank anonymous referees for their comments.

References

[1] J.-D. Benamou, Big ray tracing: next term multivalued travel time field computation using viscosity solutions of the eikonal equation, J. Comput. Phys. 128 (2) (1996) 463–474.
[2] J.-D. Benamou, Direct computation of multivalued phase space solutions for Hamilton–Jacobi equations, Commun. Pure Appl. Math. 52 (11) (1999) 1443–1475.
[3] J.-D. Benamou, An introduction to Eulerian geometrical optics (1992–2002), J. Sci. Comput. 19 (1–3) (2003) 63–93, Special issue in honor of the sixtieth birthday of Stanley Osher.
[4] E.J. Candès, L. Demanet, The curvelet representation of wave propagators is optimally sparse, Commun. Pure Appl. Math. 58 (2005) 1472–1528.
[5] L.-T. Cheng, H. Liu, S. Osher, Computational high-frequency wave propagation using the level set method, with applications to the semi-classical limit of Schrödinger equations, Commun. Math. Sci. 1 (3) (2003) 593–621.
[6] M. de Berg, M. van Kreveld, M. Overmars, O. Schwarzkopf, Computational Geometry, revised edition., Springer-Verlag, Berlin, 2000.
[7] C. de Boor, A Practical Guide to Splines, revised edition. Applied Mathematical Sciences, 27, Springer-Verlag, New York, 2001.
[8] B. Engquist, S. Osher, S. Zhong, Fast wavelet based algorithms for linear evolution equations, SIAM J. Sci. Comput. 15 (1994) 755–775.
[9] B. Engquist, O. Runborg, Multiphase computations in geometrical optics, in: Hyperbolic Problems: Theory, Numerics, Applications, vol. I (Zürich, 1998), Int. Ser. Numer. Math., vol. 129, pp. 273–284, Birkhäuser, Basel, 1999.
[10] B. Engquist, O. Runborg, Computational high frequency wave propagation, Acta Numer. 12 (2003) 181–266.
[11] B. Engquist, O. Runborg, A.-K. Tornberg, High-frequency wave propagation by the segment projection method, J. Comput. Phys. 178 (2) (2002) 373–390.
[12] E. Fatemi, B. Engquist, S. Osher, Numerical solution of the high frequency asymptotic expansion for the scalar wave equation, J. Comput. Phys. 120 (1) (1995) 145–155.
[13] S. Fomel, J.A. Sethian, Fast-phase space computation of multiple arrivals, Proc. Natl. Acad. Sci. USA 99 (11) (2002) 7329–7334 (electronic).
[14] P. Gérard, P.A. Markowich, N.J. Mauser, F. Poupaud, Homogenization limits and Wigner transforms, Commun. Pure Appl. Math. 50 (4) (1997) 323–379.
[15] E. Hairer, S.P. Nørsett, G. Wanner, Solving Ordinary Differential Equations I, second ed., Springer Series in Computational Mathematics, vol. 8, Springer-Verlag, Berlin, 1993 (Nonstiff problems).
[16] E. Hairer, G. Wanner, Solving ordinary differential equations II, second ed.Springer Series in Computational Mathematics, vol. 14, Springer-Verlag, Berlin, 1991, Stiff and differential-algebraic problems.

[17] A. Harten, B. Engquist, S. Osher, S.R. Chakravarthy, Uniformly high-order accurate essentially nonoscillatory schemes. III, J. Comput. Phys. 71 (2) (1987) 231–303.
[18] A. Harten, S. Osher, Uniformly high-order accurate nonoscillatory schemes. I, SIAM J. Numer. Anal. 24 (2) (1987) 279–309.
[19] A. Iserles, A First Course in the Numerical Analysis of Differential Equations, Cambridge Texts in Applied Mathematics, Cambridge University Press, Cambridge, 1996.
[20] G. Lambaré, P. Lucio, A. Hanyga, Two-dimensional multivalued traveltime and amplitude maps by uniform sampling of ray field, Geophys. J. Int. 125 (1996) 584–598.
[21] R.J. LeVeque, Finite Volume Methods for Hyperbolic Problems, Cambridge Texts in Applied Mathematics, Cambridge University Press, Cambridge, 2002.
[22] S. Osher, L.-T. Cheng, M. Kang, H. Shim, Y.-H. Tsai, Geometric optics in a phase-space-based level set and Eulerian framework, J. Comput. Phys. 179 (2) (2002) 622–648.
[23] J. Qian, L.-T. Cheng, S. Osher, A level set-based Eulerian approach for anisotropic wave propagation, Wave Motion 37 (4) (2003) 365–379.
[24] A. Quarteroni, A. Valli, Numerical Approximation of Partial Differential Equations, Springer Series in Computational Mathematics, vol. 23, Springer-Verlag, Berlin, 1994.
[25] O. Runborg, Some new results in multiphase geometrical optics, M2AN Math. Model. Numer. Anal. 34 (6) (2000) 1203–1231.
[26] S.J. Ruuth, B. Merriman, S. Osher, A fixed grid method for capturing the motion of self-intersecting wavefronts and related PDEs, J. Comput. Phys. 163 (1) (2000) 1–21.
[27] L. Ryzhik, G. Papanicolaou, J.B. Keller, Transport equations for elastic and other waves in random media, Wave Motion 24 (4) (1996) 327–370.
[28] C. Sparber, P.A. Markowich, N.J. Mauser, Wigner functions versus WKB-methods in multivalued geometrical optics, Asymptot. Anal. 33 (2) (2003) 153–187.
[29] J. Steinhoff, M. Fan, L. Wang, A new Eulerian method for the computation of propagating short acoustic and electromagnetic pulses, J. Comput. Phys. 157 (2) (2000) 683–706.
[30] W.W. Symes. Mathematical foundations of reflection seismology, Technical Report, Rice University, 1998..
[31] W.W. Symes, J. Qian, A slowness matching Eulerian method for multivalued solutions of eikonal equations, J. Sci. Comput. 19 (1–3) (2003) 501–526.
[32] L. Tartar, H-measures a new approach for studying homogenisation, oscillations and concentration effects in partial differential equations, Proc. Roy. Soc. Edinburgh Sect. A 115 (3–4) (1990) 193–230.
[33] A.-K. Tornberg, B. Engquist, The segment projection method for interface tracking, Commun. Pure Appl. Math. 56 (1) (2003) 47–79.
[34] J. van Trier, W.W. Symes, Upwind finite-difference calculation of traveltimes, Geophysics 56 (1991) 812–821.
[35] J.E. Vidale, Finite-difference calculation of traveltimes, Bull. Seismol. Soc. Am. 78 (1988) 2062–2076.
[36] V. Vinje, E. Iversen, H. Gjøystdal, Traveltime and amplitude estimation using wavefront construction, Geophysics 58 (1993) 1157–1166.
[37] G.B. Whitham, Linear and nonlinear waves, Pure and Applied Mathematics, John Wiley & Sons Inc., New York, 1999, Reprint of the 1974 original, A Wiley-Interscience Publication.