[Detection of conserved quantities for PDE schemes]An algorithm for the detection of polynomial conserved quantities of finite difference schemes for partial differential equations

Diogo A. Gomes King Abdullah University of Science and Technology (KAUST) [0]CEMSE Division [1]KAUST SRI, Center for Uncertainty Quantification in Computational Science and Engineering Thuwal 23955-6900 Saudi Arabia diogo.gomekaust.edu.sa Friedemann Krannich King Abdullah University of Science and Technology (KAUST) CEMSE Division Thuwal 23955-6900 Saudi Arabia friedemann.krannickaust.edu.sa Ricardo Ribeiro King Abdullah University of Science and Technology (KAUST) CEMSE Division Thuwal 23955-6900 Saudi Arabia ricardo.ribeirkaust.edu.sa

Symbolic computations; OTHER KEWYWORDS

[100]Networks

Many partial differential equations (PDEs) admit conserved quantities, that describe a physical quantity like mass or energy. When approximating PDEs by finite difference schemes, it arises the question whether those quantities remain conserved by the respective scheme. In this paper, we present an algorithm, that can check systematically, if a polynomial quantity is preserved under a finite difference scheme for a polynomial PDE. Our algorithm can handle a fairly large number of schemes, including implicit schemes, schemes with higher order time derivatives and an arbitrary number of variables. Additionally, we also present an algorithm that approaches the problem from the other side and, given a scheme, finds polynomial conserved quantities. Using our implementation of the algorithm, we examine several schemes that arise in applications for their properties regarding conserved quantities.

D. Gomes was partially supported by GN01KAUST baseline and start-up funds and GN02KAUST SRI, Uncertainty Quantification Center in Computational Science and Engineering. F. Krannich was partially supported by himself. R. Ribeiro was partially supported by himself.

Introduction.

Many partial differential equations (PDEs) admit integral quantities, that are conserved in time. For example, the advection equation preserves energy and the heat equation conserves mass. When approximating PDEs by finite-difference schemes, the questions arises whether such quantities, will still be conserved by the scheme. TO DISCUSS On the other hand one may ask if there are quantities, that are preserved by a scheme, that are actually not conserved by the original PDE. Such information can be crucial to determine whether a scheme approximates a PDE accurately. While such computations are often easy in simple cases, they get rather tedious for more complicated schemes or quantities. Hence, systematic algorithms that perform such computations and their implementations in computer algebra systems are desirable. An algorithm that finds conserved quantities for (continuous) differential equations was developed by Cheviakov, who also implemented this algorithm in Maple (see [1] and [1]). Work in this area has also been done by Hereman et al (see for example [1] and [1]), who proposed an algorithm to compute conserved densities for semi-discretized schemes for PDEs with first-order time derivative. Their algorithm

uses the scaling symmetries of the scheme to construct conserved quantities and calculates their coefficients by using the discrete variational derivative. Hereman et al also implemented their algorithm as part of a package in Mathematica. Gao et al extended this algorithm first to first-order time explicit schemes [1] and later generalized it to first-order time implicit schemes [1]. They also implemented it in the computer algebra system Reduce. In this paper, we approach the problem from the other side and propose an algorithm that checks, if a quantity is conserved in time under a scheme. Such quantities usually involve integral expressions or their discrete analog, sums. Algorithms for the simplification of sums have been proposed by Gomes et al in [1], where they also developed and implemented algorithms for detecting conserved quantities for PDEs and semi-discretized schemes. We will revise this techniques in Section [1]. To generalize those methods to situations where we do not sum over all arguments of the functions involved, we introduce the discrete partial variational derivative (Section [1]). Our algorithm detects conservation using the properties of Grbner basis (see Section [1]). It works for both schemes that are explicit and implicit in time and can treat schemes with arbitrary many and arbitrary orders of time derivatives and several space dimensions. Additionally we can treat systems of equations and equations with parameters. We have implemented this algorithm as part of a package in Mathematica [1]. We use our code to find conserved quantities and proper schemes for the time implicit and time explicit discretization of the Burgers equation and a system of PDEs arising in the study of mean-field games (see Section [1]).

Preliminaries and definitions.

Throughout this section we follow the ideas and computations in [1]. Here, subscripts denote indices related to coordinates or tuples while superscripts denote indices related to sequences.

Definitions.

Let [1], [1] and [1] be natural numbers and let the discrete torus [1] be defined as [1].

We define the space of functions on [1], as [1]

The periodicity allows us to extend all functions in [1] to [1]. Moreover, with periodicity we avoid discussing boundary terms.

Define the space of functionals [1] (not necessarily linear) on [1] as [1]

In the previous definition, the [1] are not necessarily unit vectors and can vary between functionals. Here we allow for arbitrary smooth [1]. However, later we require [1] to be a polynomial function.

Let [1], [1] and define [1]

For the simplification of sums, the discrete variational derivative is a useful tool: Let [1], [1] and [1]. Let [1] Then

[1]

We can shift the indices due to the periodicity of [1].

We call [1] the discrete variational derivative of [1].

Because we assumed all related functions to be sufficiently smooth, the discrete variational derivative always exists.

Let us return to the previous example and compute [1] and hence [1].

Basic theorem.

The algorithms for the simplification of sums presented in [1] rely on the following result:

Let [1]. Then [1] holds for all [1], if [1] is true for all [1] and if there exists [1] such that [1] holds. Conversely, [1] holds for all [1], if [1].

[1] and hence [1] follows.

Simplification of sums.

We use this results to simplify sums or examine if they sum to the same quantity.

Let [1] and consider the functionals [1] Although it is clear, that those functionals represent the same quantity (one shifts the index [1] due to the periodicity), we can now check this explicitly, using the discrete variational derivative: [1] so [1] and [1]. Hence both functionals sum to the same quantity.

Let [1]. It may not be obvious, that [1] sums to zero, but this can be checked by computing the discrete variational derivative and noticing that the whole expression is [1] when [1].

The discrete partial variational derivative.

We want to generalize the discrete variational derivative for situations where we keep one (or several) of the arguments of [1] constant and do not sum over them.

Let [1] be the vector of variables for the functions in [1]. We call them space variables. Let [1] and call [1] the time variables, that take values in [1]. When considering sums, we only sum over the space variables and not over the time variables and we keep the time variables constant when computing the discrete partial variational derivative. We also do not require periodicity in the time variables.

Define the function space [1]

We can extend the functions in [1] to [1] by periodicity in [1].

Define the function space [1]

Now we can consider functionals where we only sum over the space coordinates and keep the time variables constant.

We define the space of functions [1] as [1]

In the previous definition, [1] the concatenated vector of space and time variables. Throughout this paper, all quantities whose conservation in time we want to check will be functions in [1]. Using the Kroneckerdelta [1] we can rewrite [1] as

[1]

We rewrite the sum from the previous example: [1]

Let [1] and [1]. We compute

[1]

We define [1] as the discrete partial variational derivative of [1].

Let us consider the same example as before and compute the discrete partial variational derivative [1]

We implemented the discrete partial variational derivative as function in Mathematica:

We verify our computations from Example [1]:

It is obivous, that Theorem [1] also holds for the discrete partial variational derivative.

Detection of conserved quantities.

During the remainder of this paper we assume that there is only one time variable [1] ([1]) and we name the remaining space variables [1]. We propose an algorithm that checks, if a quantity (a functional [1] ) is conserved in time under a fully discretized scheme for a PDE.

Define the space of functions [1] by [1] A scheme is a function [1] or a set of functions [1] understanding, that this means [1] or [1] for all [1].

In this work, we only consider finite-difference schemes with fixed step sizes [1], but one can generalize our results and algorithms to other step sizes.

We say that a scheme [1] given by [1] is in time explicit form if we can rewrite the previous equation (eventually translating the scheme) as [1] with [1]. We call [1] the right-hand side of the scheme [1].

TODO: Maybe introduce a normal form of the scheme and also of a quantity, talking about equivalence classes to justify why whe can translate how we want?

Consider the forward difference scheme for the heat equation [1] [1] We can rewrite this scheme in time explicit form as [1]

The time explicit case.

Our algorithm is summarized in Algorithm [1].

TODO: DOES SAY SOMMETHING ABOUT MULTISTEP QUNTITIWS Notice, that after step 2 the resulting expression exists only at time [1], because all instances of [1] have been replaced. We have implemented this algorithm in Mathematica in the DiscreteConservedQOperator.

We check for conservation of mass under the time explicit scheme for the heat equation in one space dimension. For simplicity, we omit the time variable in the time explicit case, as all computations occur only at time [1].

The general case.

The algorithm for the general case is explained in detail below and summarized in Algorithm [1].

Build the discrete time derivative.

The algorithm builds the discrete time derivative by subtracting the quantity whose conservation we want to check at time [1] from the quantity at time [1]. This task is done in our code by the TimeDifferenceOperator.

Build the discrete time derivative for the quantity [1]

Our code also allows to check not only for conservation in time, but also if higher time derivatives are zero under the scheme by specifying a timederivativeorder in the variables.

Translate the scheme.

Let [1] and [1] be defined by

[1]

with [1] then we define the stencil of [1] as the tuple of sets of vectors [1]

We see that the stencil of an expression is not unique, it can vary when we change the order of the [1]. In applications it is useful to order the functions lexicographically, if they are not already indexed.

Let [1] be a stencil. We define its range as

[1]

i.e. as a tupel containing for every function a tupel containing for every coordinate a tupel with the minimum and the maximum value of that coordinate.

We calculate the translations of the scheme by elementwise subtracting range of the stencil of the scheme from the range of the stencil of the discrete time derivative. Then we replace every tupel [1] by the tupel [1]. Afterwards we replace every tupel of tupels (corresponding to a function) [1] by the outer product of those tupels [1] Every tupel corresponds to a translation of the scheme and every entry of the tupel denotes the translation in the respective variable.

Consider the discrete time derivative [1] and the scheme [1] Then the stencil of the discrete time derivative is [1] with range [1] The stencil of the scheme is [1] with range [1] and hence we get the translations by [1] We can not replace any of the tupels here, but we can do the second replacement resulting in [1] So all translations of the scheme are [1] and [1] corresponding to the scheme itself and [1]

Compute the Grbner basis and reduce the discrete time derivative.

Grbner bases: theoretical background.

In this subsection we will follow [1]. Consider a set of polynomials [1] in the variables [1], implicitly understanding that [1] for all [1]. Given another polynomial, [1], in the same variables, one wants to check if this polynomial belongs to the ideal generated by [1] or if at least one can reduce [1], i.e. write it in a simpler form using [1]. The remainder of [1] under [1], using polynomial division in several variables, may not be unique [page 14, Example 1.2.3]hibi13, it can even be non-zero, although [1] belongs to the ideal generated by [1]. Replacing [1] by its Grbner basis solves this problem, as the Grbner basis generates the same ideal but also guarantees uniqueness of the remainder of the polynomial division. To define the Grbner basis, one needs to define an order of the monomials involved. Depending on the order, one can get different remainders of the polynomial division. Grbner basis in our algorithm.

The algorithm takes the set of translated versions of the scheme, understanding each different instance of the functions as a polynomial variable.

The algorithm will understand the scheme [1] as the polynomial equation [1]

Our code tries to reduce the discrete time derivative in two different ways: First, the algorithm computes the Grbner basis of the translated scheme with respect to the default lexicographic order in Mathematica and then calculates the remainder of the discrete time derivative with respect to this Grbner basis. Second, the algorithm takes an ordered list (or several ordered lists) of the instances of functions that appear in the discrete time derivative and successively eliminates them from the time derivative by using building the weight matrix that leads to the desired order of elimination, calculates the related Grbner basis and reduces the discrete time derivative using polynomial division. Our code allows for as many different orderings as required by the user. By default it uses the explicit ordering (first eliminates instances of functions at later times and then earlier ones), but one can also use an implicit ordering (first eliminates earlier times and then later ones) or use the permutation ordering (gives a list of lists with all possible orders of the variables - logically the most desirable, but

computationally very costly). After those computations, the algorithm chooses the result with the least number of different instances of time. This is reasonable, as our quantity does not sum over the time and hence there are no possibilities besides the polynomial division to simplify the result with respect to time, while one can simplify in the space variables using our methods for simplification of sums.

Compute the discrete partial variational derivative and reduce again.

In the next step, the algorithm takes the resulting expression and computes its discrete partial variational derivative with respect to the time variable [1]. Then it calculates the translations of the range of the stencil again, and reduces the discrete partial variational derivative another time as described above.

The whole algorithm is implemented also in the DiscreteConservedQOperator, that can detect automatically if we are in the time explicit or implicit setting.

We want to verify our result from Example [1] by checking for the same scheme, but now written in the general, not time explicit form:

One should be aware, that the algorithm can only detect conservation, but can not check if something is not conserved for sure. So a resulting False should be understood as our algorithm not detecting conservation, not as there being no conservation at all.

A Basis for conserved quantities.

So far, we have discussed how to check if a quantity is preserved in time under a scheme. But it also desirable to have a systematic way to find conserved quantities for a scheme. The following Algorithm [1]. finds for a time explicit scheme all conserved quantities, that are generated by monomials up to a (total) degree.

We have implemented this algorithm in the FindDiscreteConservedQuantityBasisOperator.

We want to find conserved quantities up to degree 3 admitted by our discretization for the heat equation from Example [1], that are generated by [1] and [1].

Examples and applications.

Burgers equation.

To demonstrate our code, we test it with the Burgers equation [1] [1] in one space dimension.

We check for conservation of[1] using a forward difference discretization:

We want to find a scheme that preserves this quantity. Therefore we check for conservation under a class of schemes with a parameter, hoping to find an appropriate choice for this parameter. We have implemented parameter handling in our code, so that this task is performed automatically.

Now we discretize [1] using a central difference with parameter [1]

We indeed get conservation if we choose [1], corresponding to the standard central difference.

So far, we have only seen schemes that could be easily written in time explicit form. But our algorithm allows also for general (implicit) schemes.

We consider the same discretization as in the previous example, but using its time implicit version.

Conserved quantities for a mean-field game.

As a second application, we use our code to study conserved quantities admitted by the discretization of a system of PDEs.

In [1], Gomes et al derived the following system of PDEs, starting from a mean-field game with quadratic Hamiltonian and coupling by applying a substitution.

[1]

This system obviously admits the conserved quantities [1] Because this system was derived from a forward-forward mean-field game, it is natural to discretize it also forward in time. We check, using our code, if the scheme admits the same preserved quantities as the continuous system:

We can reproduce this result by rewriting the schemes in time explicit form:

Those are the only conserved quantities for this scheme, as we can check with our code:

The related backward-forward system with terminal condition for [1] reads

[1] Obviously this system admits the same conserved quantities as the forward-forward system, but it is natural to approximate this system explicit (forward) in time for [1] and implicit (backward) for [1]. Our code can also handle this setting, but we need to specify an order for the elimination of the variables, telling the code to use an explicit order for [1] and an implicit one for [1]:

Possible extensions and concluding remarks.

Polynomial treatment of non-polynomial expressions.

One can extend our algorithm to non-polynomial expressions by writing them in polynomial form. We can for example write the expression [1] in polynomial form as [1]

Translate the scheme more accurate.

It could be necessary to translate the scheme even more then we did in our algorithm to get cancellation, although we could not come up with an example where this actually happens. So therefore a more accurate method to determine the translations of the scheme could be desirable, one the one hand to achive better results and on the other hand to prevent the algorithm from becoming to costly in terms of computational resources needed.

Discrete conserved quantities that are not conserved by the PDE.

It may arise the question, if one can find conserved quantities, that are not preserved by the original PDE. Although we could not come up with an example, where this is actual the case, this is a natural question when one finds a conserved quantity. A criterion, that ensures that all discrete conserved quantities are also conserved by the original PDE is given by Gerdt et al [1]. They define the notion of s-consistency (see [1]), that guarantees that we only find discrete quantities that are also preserved in the continuous setting. S-consistency for a scheme can be easily checked by calculating the Taylor expansion of the Grbner basis of the scheme ([Theorem 3]gerdt11). In most of our examples (where we have only one single PDE and not a system) this is trivially the case.

Concluding remarks.

We have presented an algorithm, that allows to check for the conservation of a quantity under a finite difference scheme. Our algorithm allows for systems of equations, arbitrary time derivatives and both explicit and implicit schemes. It can also check for the conservation of time derivatives of higher order. For backward engineering, there also exists a function for finding conserved quantities admitted by a scheme. We used our implementation of the algorithm to analyze the conservation properties of several schemes for PDEs that arise in applications. The algorithm can be extended to non polynomial quantities, but implementing this remains as a task for the future.

acmnumeric ACM-Reference-Format.bst references