Estimation of the diffusion coefficient by interval methods, level curves and bicubic splines

O. Fogelklou[a,1,2,*]

[a]*Department of Mathematics, Uppsala University, P.O. Box 480, SE–751 06 Uppsala, Sweden*

Abstract

We propose a new method for estimating a solution dependent diffusion coefficient in the heat equation, given a numerical solution to the latter. The main idea is to use a set–valued approximation of the solution in order to construct constraints on the coefficient. These constraints enable us to obtain a cover of the graph of the diffusion coefficient for a discrete set of temperatures. We illustrate the pros and cons of our method on several examples.

Keywords: inverse problem, bicubic spline, interval analysis, heat equation

1. Introduction

Inverse problems are notoriously hard; inverse problems for the heat equation are no exception. Many variations of these type of problems have been investigated; for example, it is common to recover a coefficient in the heat equation. Here the problem can be linear with space dependent coefficient and one-dimensional as in [1], [2] and [3], linear with time dependent coefficient and one-dimensional as in [4], linear with space dependent coefficient and multidimensional as in [5], [6] and [7], and nonlinear with space dependent coefficient and one-dimensional as in [8]. The initial temperature can also be recovered as in the linear and one-dimensional problems described in [9] and [10]. Yet another variant of this problem is to consider the heat equation driven by a source which is to be determined as in the linear one-dimensional problems in [11], [12] and [13], and in the linear multidimensional problem [14]. The solution can be recovered on the boundary too as in [15]. The inverse problems can

*Corresponding author

Email address: oswald@math.uu.se (O. Fogelklou)

[1]Telephone: +46 18 471 3187

[2]Fax: +46 18 471 3201

also be a combination of the earlier mentioned problems. There is a search for the coefficient and the initial temperature in [16] and [17], and for the coefficient, the initial temperature and the boundary conditions in [18]. The four last problems are one-dimensional and linear. The problem we study comes from [19] and has some similarity to [8] due to the nonlinearity and the coefficient recovery. In our situation, however, we use a distinct method from that used in [19].

2. Problem formulation

Given an approximation of the solution u of the heat equation

$$\begin{cases} u_t = (a(u)u_x)_x,\ 0 < x < 1 \text{ and } 0 < t < T, \\ u(t,0) = f(t),\ u(t,1) = g(t),\ 0 < t < T, \\ u(0,x) = u_0(x), \end{cases} \tag{1}$$

our goal is to estimate the diffusion coefficient $a(u)$. Here a, f and g are positive functions and f and g are also continuously differentiable. To be more precise, what we are aiming for is a tabulation of a over a finite set of temperatures $u_1, \ldots, u_M$, taken from the range of the solution to (1).

It should be pointed out that this problem is hard even if the approximation errors $|u - \tilde{u}|$ are zero. One way of simplifying matters is to only consider diffusion coefficients having a (known) finite parameterization, e.g. $a(u) = a_1 u + a_2 \sin(a_3 u^2)$. This makes the problem finite dimensional, and easier to analyze.

2.1. Main strategy

Our approach to this problem begins by recasting (1) into the integral form:

$$a(u(t,x_2))u_x(t,x_2) - a(u(t,x_1))u_x(t,x_1) = \int_{x_1}^{x_2} u_t(t,x)dx. \tag{2}$$

Next, we consider two levels c_1 and c_2 and two level curves $\Gamma_1 = \{(t,x) \in [0,T] \times [0,1] \colon u(t,x) = c_2\}$ and $\Gamma_2 = \{(t,x) \in [0,T] \times [0,1] \colon u(t,x) = c_2\}$. Assuming that we can find two points on Γ_1 and Γ_2 having the same first variable value t_1', say (t_1', h_{11}) and (t_1', h_{12}) and two points on Γ_1 and Γ_2 having another same first variable value t_2', say (t_2', h_{21}) and (t_2', h_{22}), we can write (2) as the equation system

$$\begin{cases} a(c_2)u_x(t_1', h_{12}) - a(c_1)u_x(t_1', h_{11}) = \int_{h_{11}}^{h_{12}} u_t(t_1', x)dx \\ a(c_2)u_x(t_2', h_{22}) - a(c_1)u_x(t_2', h_{21}) = \int_{h_{21}}^{h_{22}} u_t(t_2', x)dx. \end{cases} \tag{3}$$

Assuming further that our approximation $\tilde{u}$ of u can be made C^1-close in the supremum norm, we have

$$\begin{cases} a(c_2)\tilde{u}_x(t_1', h_{12}) - a(c_1)\tilde{u}_x(t_1', h_{11}) \approx \int_{h_{11}}^{h_{12}} \tilde{u}_t(t_1', x)dx \\ a(c_2)\tilde{u}_x(t_2', h_{22}) - a(c_1)\tilde{u}_x(t_2', h_{21}) \approx \int_{h_{21}}^{h_{22}} \tilde{u}_t(t_2', x)dx. \end{cases} \tag{4}$$

Given a sufficient amount of high quality data, we can achieve this by approximating the solution by a piecewise bicubic spline. Repeating the process for different values of the levels c_1 and c_2 and solving (4) for $a(c_1)$ and $a(c_2)$, we obtain the desired tabulation of the diffusion coefficient a.

3. Set–valued considerations

Of course, in passing from (3) to (4), we must try to account for the effects of our approximations. To this end, we will extend our computations to the set–valued realm. We use sets to incorporate the fact that we are dealing with uncertainties; both in the given data, and in our subsequent estimations of the partial derivatives and integrals appearing in (4). We will use boldface to indicate that a entity is set–valued: the basic element is a compact interval $\boldsymbol{x} = [\underline{x}, \overline{x}] = \{x \in \mathbb{R} \colon \underline{x} \leq x \leq \overline{x}\}$. There exists a well developed theory for set–valued analysis [20, 21, 22], as well as sophisticated libraries for this purpose [23, 24, 25, 26]

Let us assume that the data provided comes in the following form:

$$\{t_i, x_j, \tilde{u}_{i,j}\}_{i \in \mathcal{I}, j \in \mathcal{J}},$$

where $\mathcal{I}$ and $\mathcal{J}$ are finite index sets. To begin with, we will use the given data set to produce a piecewise bicubic spline. This can be done by standard methods [27], given a (rectangular) partition of the global domain $\cup_{k=1}^{N} \mathcal{P}^{(k)} = [0, T] \times [0, 1]$. Thus, on each set $\mathcal{P}^{(k)}$ of the partition, we approximate the solution to (1) by a bicubic spline

$$s^{(k)}(t, x) = \sum_{0 \leq m,n \leq 3} b_{mn}^{(k)} (t - \check{t}^{(k)})^m (x - \check{x}^{(k)})^n. \tag{5}$$

Here $(\check{t}^{(k)}, \check{x}^{(k)})$ denotes the lower left corner of the partition element at hand $\mathcal{P}^{(k)}$. There are three kinds of errors for s approximating u.

- Errors are made by the solver of the forward problem (1).

- Errors are made in the finite difference approximation of $\tilde{u}_t$.

- Last but not least, the spline could be a bad fit of u at non-grid points.

We compensate for this by inflating the spline coefficients into intervals, producing a set–valued spline:

$$\boldsymbol{s}^{(k)}(t,x) = \sum_{0\le m,n\le 3} \boldsymbol{b}^{(k)}_{mn}(t-\check{t}^{(k)})^m(x-\check{x}^{(k)})^n. \tag{6}$$

Our aim is to enclose all data points in the, now set-valued, graph of the spline:

$$\tilde{u}_{i,j} \in \boldsymbol{s}^{(k)}(t_i,x_j) \quad \text{for all} \quad (t_i,x_j)\in\mathcal{P}^{(k)}, i\in\mathcal{I}, j\in\mathcal{J}. \tag{7}$$

Once we have achieved (7) for all $k = 1,\dots,N$, we form the spline for the global domain by gluing together the local pieces:

$$\boldsymbol{s}(t,x) = \sum_{k=1}^{N} \chi_{\mathcal{P}^{(k)}}(t,x)\boldsymbol{s}^{(k)}(t,x). \tag{8}$$

It is our hope that, by formally differentiating (8), we can obtain reasonable approximations to – or even enclosures of – the partial derivatives of u:

$$u_t(t,x)\in\boldsymbol{s}_t(t,x) \quad \text{and} \quad u_x(t,x)\in\boldsymbol{s}_x(t,x). \tag{9}$$

If this is indeed the case, we can mimic the strategy outlined in Section 2.1. The first difference is that we must now consider set–valued level curves: $\boldsymbol{\Gamma}_1 = \{(t,x)\in[0,T]\times[0,1]\colon c_1\in\boldsymbol{s}(t,x)\}$ and $\boldsymbol{\Gamma}_2 = \{(t,x)\in[0,T]\times[0,1]\colon c_2\in\boldsymbol{s}(t,x)\}$. This means for example that an intersection between $\boldsymbol{\Gamma}_1$ and the line $t = t'_1$ and an intersection between $\boldsymbol{\Gamma}_2$ and the line $t = t'_1$ will not consist of points (previously denoted h_{11} and h_{12}), but of intervals $\mathbf{h}_{11}$ and $\mathbf{h}_{12}$. These intervals can be found by a standard branch and bound procedure, in which the global x domain $[0,1]$ is bisected into smaller intervals. Once we have the four intervals $\mathbf{h}_{11}$, $\mathbf{h}_{12}$, $\mathbf{h}_{21}$ and $\mathbf{h}_{22}$ we can formulate a set–valued version of the constraint (4) for the diffusion coefficient, producing an interval enclosure of $a(c_1)$ and $a(c_2)$. Repeating the process for different values of the levels c_1 and c_2, we obtain an interval–valued tabulation of the diffusion coefficient a.

3.1. The set–valued level curves

We choose a vector with levels $c_1,\dots,c_M$ and a vector with time values $t'_1,\dots,t'_N$. Our goal is to estimate a on the set $\{c_1,\dots,c_M\}$. For every time value t'_n and every level c_m we check if $c_m\in\mathbf{s}(t'_n,[0,1])$. If that is the case, we check if $c_m\in\mathbf{s}(t'_n,[0,1/2])$

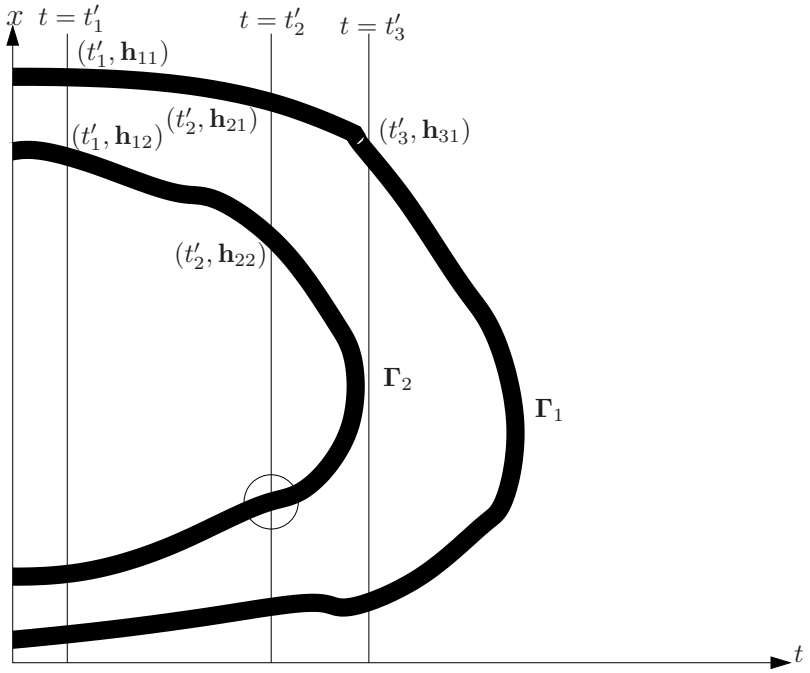

Figure 1: Some possible entries in the matrix $\mathbf{H}$.

or $c_m \in \mathbf{s}(t_n', [1/2, 1])$. We continue bisecting each interval with x values $\mathbf{I}_x$ and reevaluating the spline on its halves if $c_m \in \mathbf{s}(t_n', \mathbf{I}_x)$. We stop bisecting the intervals when their lengths are smaller than some given tolerance *tol*. Each interval $\mathbf{I}_x$ such that $c_m \in \mathbf{s}(t_n', \mathbf{I}_x)$ and such that its length is shorter than *tol* is put into a list. If the list is non–empty, the union of the entries in the list consists of one or several disjoint intervals. We let one of these intervals be the entry $\mathbf{h}_{nm}$ in a matrix $\mathbf{H}$. If the list is empty, we just set $\mathbf{h}_{nm}$ empty as well. In Figure 1 we show how $\mathbf{h}_{11}$, $\mathbf{h}_{12}$, $\mathbf{h}_{21}$, $\mathbf{h}_{22}$, $\mathbf{h}_{31}$ and $\mathbf{h}_{32}$ can be chosen for some set–valued level curves. Note that $\mathbf{h}_{32} = \emptyset$, since the line $t = t_3'$ does not cut the level curve $\mathbf{\Gamma}_2$ and that there is a subset of the circle in Figure 1 which could have been our chosen $\mathbf{h}_{22}$.

3.2. Estimation of the diffusion coefficient

Suppose that there are t_n', $t_p' \neq t_n'$, c_m and $c_q \neq c_m$ such that $\mathbf{h}_{nm}$, $\mathbf{h}_{nq}$, $\mathbf{h}_{pm}$ and $\mathbf{h}_{pq}$ are non–empty and such that $\mathbf{h}_{nm} \cap \mathbf{h}_{nq}$ and $\mathbf{h}_{pm} \cap \mathbf{h}_{pq}$ are empty. Then we look for solutions $\mathbf{a}(c_q)$, $\mathbf{a}(c_m)$ of the interval linear equation system

$$\begin{cases} \mathbf{s}_x(t_n', \mathbf{h}_{nq})\mathbf{a}(c_q) - \mathbf{s}_x(t_n', \mathbf{h}_{nm})\mathbf{a}(c_m) = \int_{\mathbf{h}_{nm}}^{\mathbf{h}_{nq}} \mathbf{s}_t(t_n', x)dx \\ \mathbf{s}_x(t_p', \mathbf{h}_{pq})\mathbf{a}(c_q) - \mathbf{s}_x(t_p', \mathbf{h}_{pm})\mathbf{a}(c_m) = \int_{\mathbf{h}_{pm}}^{\mathbf{h}_{pq}} \mathbf{s}_t(t_p', x)dx \end{cases}$$

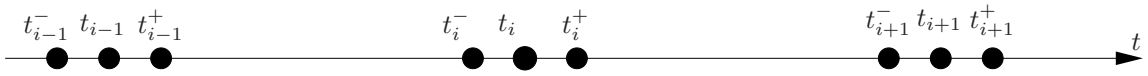

Figure 2: Two nodes, t_i^- and t_i^+, are set symmetrically around and very close to each time node t_i.

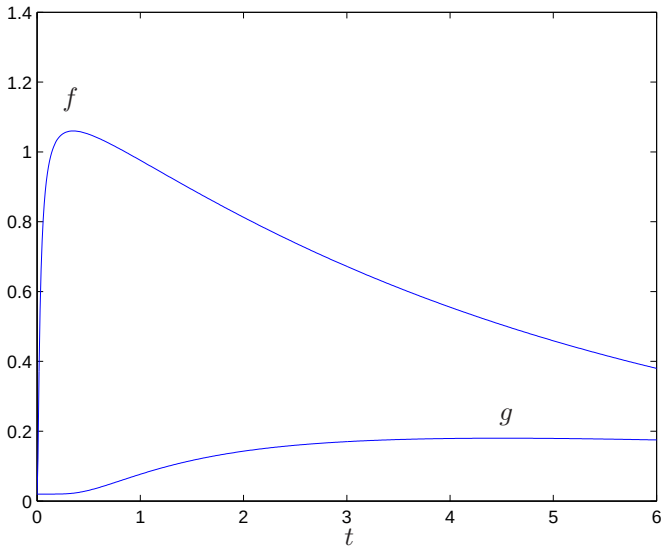

Figure 3: The boundary data $f(t)$ and $g(t)$.

corresponding to the constraint (4). Of course we can construct an interval linear equation system with time values t'_n and t'_p for any $p \neq n$ and unknowns $\mathbf{a}(c_m)$ and $\mathbf{a}(c_q)$ for any $q \neq m$. Therefore for each $\mathbf{a}(c_m)$ we gather all possible equation systems containing the unknown $\mathbf{a}(c_m)$ whose non–emptiness and emptiness properties mentioned above hold. In all these equation systems $\mathbf{a}(c_m)$ is a solution. Therefore we let $\mathbf{a}(c_m)$ be the intersection of all these solutions. Finally the diffusion coefficient is shown as a function of u on the set $\{c_1, \ldots, c_M\}$ in a plot.

4. Examples and results

The approximate solution to (1), $\tilde{u}$, and its interpolated derivative $\tilde{u}_x$ are computed by the MATLAB initial-boundary value problem solver `pdepe` until the time $t = T = 6$. The time derivatives also have to be approximated. In order to compute time derivative finite difference approximations, we set two nodes t_i^- and

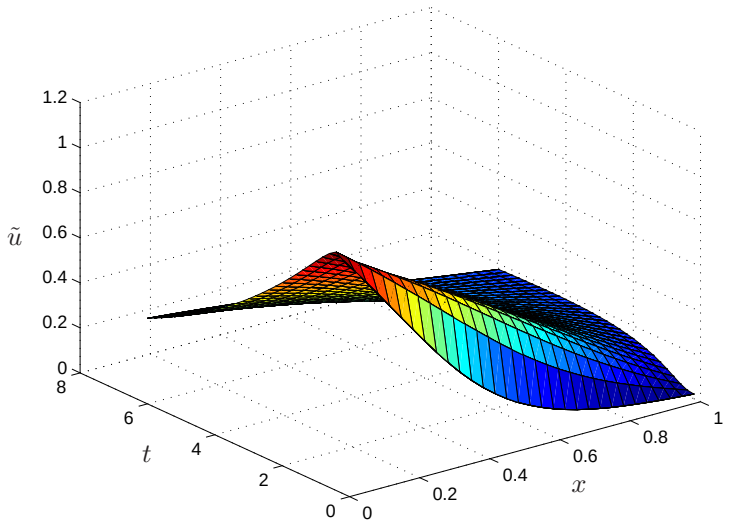

Figure 4: The solution to the forward problem $\tilde{u}(x,t)$.

t_i^+symmetrically around and very close to each time node t_i. See Figure 2. In our examples we choose the initial condition $u_0(x) = \omega_0$ and the boundary data

$$f(t) = \begin{cases} \alpha_1 e^{-\beta_1 t - \gamma_1/t} + \omega_0, & \text{if } t > 0 \\ \omega_0, & \text{if } t = 0 \end{cases}$$

and

$$g(t) = \begin{cases} \alpha_2 e^{-\beta_2 t - \gamma_2/t} + \omega_0, & \text{if } t > 0 \\ \omega_0, & \text{if } t = 0, \end{cases}$$

where

$$\begin{aligned} \alpha_1 &= 1.195779408199670 \\ \beta_1 &= 0.199396405798781 \\ \gamma_1 &= 0.024426059710351 \\ \alpha_2 &= 0.342798419030656 \\ \beta_2 &= 0.084663195505547 \\ \gamma_2 &= 1.714429708987323 \\ \omega_0 &= 0.02. \end{aligned}$$

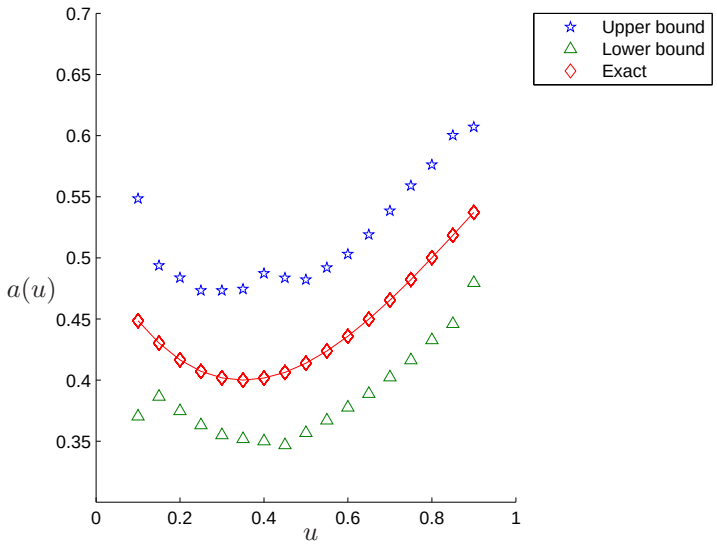

Figure 5: The diffusion coefficient $a(u)$. $r = 0.01$. Time values in $\{1, 1.5, 2, 2.5, \ldots, 5\}$. Computation time: 37 minutes.

The functions f and g given here resemble the upper and lower curve in FIG. 7.1 in [19] scaled by a factor 10^{-4} on the x–axis and 10^{-3} on the y–axis. They also satisfy the conditions (2.1) in [19] that $f(0) = g(0) = \omega_0$ and $f'(0) = g'(0) = 0$. The functions are shown in Figure 3. We also let $a(u) = 0.5 - 0.620761904761905u + 1.098231292517007u^2 - 0.402721088435375u^3$. This function resembles the function in FIG. 7.3 in [19] scaled by a factor 10^{-3} on the x–axis. The approximate solution of the forward problem $\tilde{u}$ is shown in Figure 4.

We set the number of time steps and the number of space grid intervals to 30 and the tolerance to 0.0005. The ratio between the time step and the small time step used for the finite differences is set to 10^4. We choose the set of levels to be $\{0.1, 0.15, 0.2, 0.25, \ldots, 0.9\}$. The spline coefficients are multiplied by the interval $[1-r, 1+r]$ to perform the transition from (5) to (6). We show the result for $r = 0.01$ and $r = 0.001$ and for time values in the set $\{1, 1.5, 2, 2.5, \ldots, 5\}$ and $\{1, 2, 3, 4, 5\}$. The four different cases and their computation times are shown in Figure 5–8. The triangles are the lower bounds of the estimation of $a(u)$, the diamonds and the curve are the exact function $a(u)$ and the pentagons are the upper bounds of the estimation of $a(u)$. All interval computations are performed using the free MATLAB package INTLAB [23].

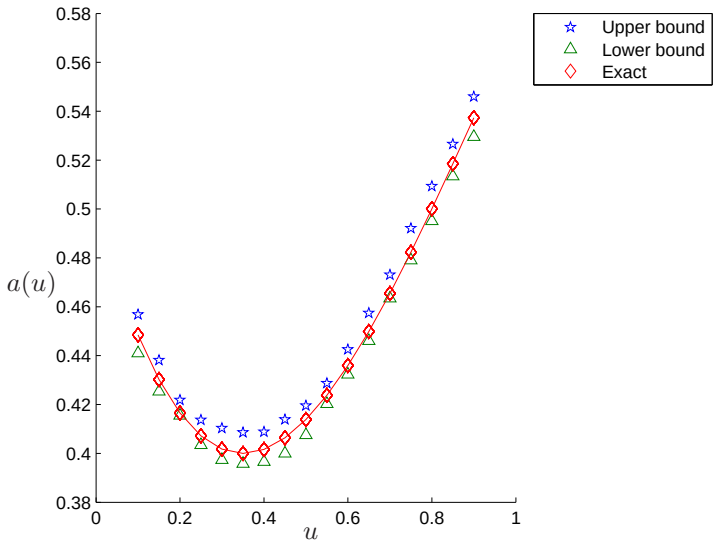

Figure 6: The diffusion coefficient $a(u)$. $r = 0.001$. Time values in $\{1, 1.5, 2, 2.5, \ldots, 5\}$. Computation time: 28 minutes.

5. Discussion

For very large or small data there is very little information. Therefore we cannot estimate a at 0 or 1.1 as in FIG. 7.3 in [19].
We find it natural to scale in the last section, in order to keep numbers not too large and not too small. The parameter values of f and g are determined by the locations of the critical points and the function values at $t = 0$, at $t = T$ and at the critical points. See FIG 7.1 in [19]. Similarly, the parameter values of a are determined by the location of the critical point and the function values at $u = 0$, at $u = 1.1$ and at the critical point. See FIG 7.3 in [19].
To decrease the computation time we can just decrease the inflation parameter r or above all the number of time values N. This is natural because if r decreases, the set–valued level curves become thinner and the chance of intersecting the line $t = t'_n$ decreases, so that more intervals are not bisected in the branch and bound procedure. If N decreases, we get a rougher estimation because less information is used as seen in Figure 8 and above all in Figure 7. In the figures one can see that there are no estimations for small and large values of u.
In the branch and bound procedure, we use queues in a time–consuming way. Since

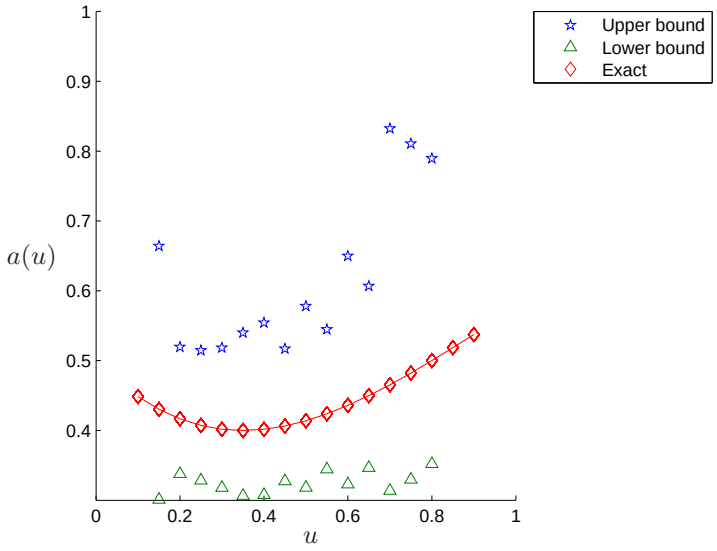

Figure 7: The diffusion coefficient $a(u)$. $r = 0.01$. Time values in $\{1, 2, 3, 4, 5\}$. Computation time: 13 minutes.

the bottleneck in the computations is solving all possible equation systems, we however do not gain much by improving the queue implementation.
The method presented in this article has the privilege that it is intuitively clear and that is easy to implement with exception of the bicubic spline part and the integrals with intervals as upper and lower limits. The drawback is that it is not rigorous so there is no guarantee that we can enclose the diffusion coefficient. It is also not clear how the levels and the time values should be chosen in a systematic way. Here we simply choose the levels and the time values to be uniformly distributed within the range of $\tilde{u}$ and the time domain respectively. It is harder to determine the appropriate value of the inflation parameter r. If r is too small, the information can get inconsistent so that we do not have any result at all. It could also happen that the interval which should include $a(u)$ do not. If r is too large, the intervals which include $a(u)$ get too wide.

Acknowledgements

The author thanks his supervisor Professor Warwick Tucker and Alexander Danis for reading many versions of the manuscript and suggesting many ideas.

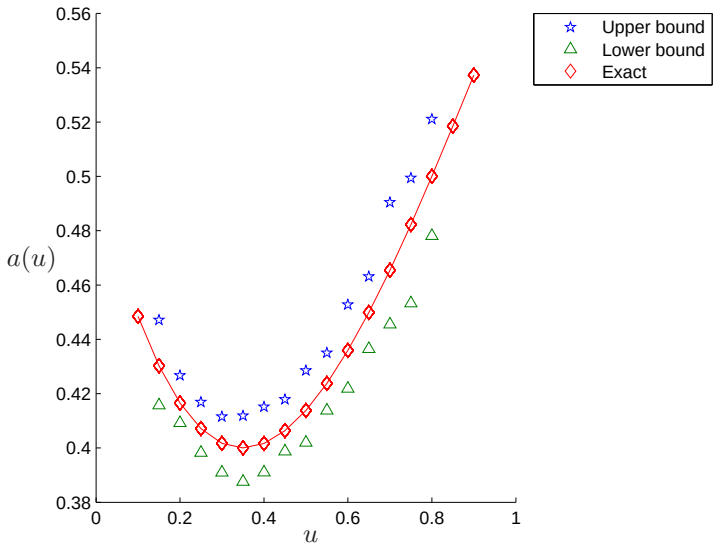

Figure 8: The diffusion coefficient $a(u)$. $r = 0.001$. Time values in $\{1, 2, 3, 4, 5\}$. Computation time: 9 minutes.

[1] A. G. Ramm, An inverse problem for the heat equation, Journal of Mathematical Analysis and Applications 264 (2) (2001) 691–697.

[2] N. S. Hoang and A. G. Ramm, An inverse problem for a heat equation with piecewise-constant thermal conductivity, Journal of Mathematical Physics 50 (6) (2009).

[3] A. Boumenir and V. K. Tuan, An inverse problem for the heat equation, Proceedings of the American Mathematical Society 138 (11) (2010) 3911–3921.

[4] M. I. Ivanchov and N. V. Saldina, Inverse problem for the heat equation with degeneration, Ukrainian Mathematical Journal 57 (11) (2005) 1563–1570.

[5] S.A. Avdonin, M. I. Belishev and Y. S. Rozhkov, The BC-method in the inverse problem for the heat equation, J. Inv. Ill-posed Problems 5 (4) (1997) 309–322.

[6] Y. V. Kurylev, N. Mandache and K. S. Peat, Hausdorff moments in an inverse problem for the heat equation: numerical experiment, Inverse Problems 19 (2) (2003) 253–264.

[7] A. Boumenir and V. K. Tuan, Inverse Problem for Multidimensional Heat Equations by Measurements at a Single Point on the Boundary, Numerical Functional Analysis and Optimization 30 (11-12) (2010) 1215–1230.

[8] E. G. Savateev and R. Riganti, Inverse Problem for the Nonlinear Heat Equation with the Final Overdetermination, Mathematical and Computer Modelling 22 (1) (1995) 29–43.

[9] K. Masood and F. D. Zaman Investigation of the Initial Inverse Problem in the Heat Equation, Journal of Heat Transfer 126 (2) (2004) 294–301.

[10] K. Masood, S. Messaoudi and F. D. Zaman Initial inverse problem in heat equation with Bessel operator, International Journal of Heat and Mass Transfer 45 (2002) 2959–2965.

[11] J. R. Cannon and S. Pérez Esteva, An inverse problem for the heat equation, Inverse Problems 2 (1986) 395–403.

[12] P. Wang and K. Zheng, Determination of the source/sink term in a heat equation, Electronic Journal o Differential Equations Conference 03 (1999) 119–125.

[13] Y. Fan and D. G. Li, Identifying the Heat Source for the Heat Equation with Convection Term, Int. Journal of Math. Analysis 3 (27) (2009) 1317–1323.

[14] M. Ikehata, An inverse source problem for the heat equation and the enclosure method, Inverse Problems 23 (2007) 183–202.

[15] A. G. Ramm, An inverse problem for the heat equation, Journal of Mathematical Analysis and Applications 123 (6) (1993) 973–976.

[16] T. Suzuki, On a certain inverse problem for the heat equation on the circle, Proc. Japan Acad. Ser. A Math. Sci. 58 (6) (1982) 243–245.

[17] T. Hattori, An inverse problem for one-dimensional heat equations with the Dirichlet boundary condition, J. Inv. Ill-posed Problems 2 (1) (1994) 33–48.

[18] T. Suzuki, Inverse problems for heat equations on compact intervals and on circles, I, J. Math. Soc. Japan 38 (1) (1986) 39–65.

[19] M. Hanke and O. Scherzer, Error analysis of an equation error method for the identification of the diffusion coefficient in a quasi-linear parabolic differential equation, Siam Journal of Applied Mathematics 59 (3) (1999) 1012–1027.

[20] Jaulin, L., Kieffer, M., Didrit, O., Applied Interval Analysis, Springer–Verlag, London, 2001.

[21] Moore, R. E., Kearfoot, B. R., Cloud, M. J., Introduction To Interval Analysis, SIAM Studies in Applied Mathematics, Philadelphia, 2009.

[22] Neumaier, A., Interval Methods for Systems of Equations. Encyclopedia of Mathematics and Its Applications 37, Cambridge Univ. Press, Cambridge, 1990.

[23] INTLAB – INTerval LABoratory Version 5.3. Available from www.ti3.tu–harburg.de/rump/intlab/.

[24] CXSC – C++ eXtension for Scientific Computation, version 2.0. Available from www.math.uni-wuppertal.de/org/WRST/xsc/cxsc.html

[25] M. Lerch, G. Tischler and J. Wolff von Gudenberg, FILIB++, a fast interval library supporting containment computations, ACM Trans. Math. Software (32) 2 (2006) 299–324.

[26] PROFIL/BIAS – Programmer's Runtime Optimized Fast Interval Library/Basic Interval Arithmetic Subroutines. Available from www.ti3.tu-harburg.de/Software/PROFILEnglisch.html

[27] W. H. Press, S. A. Teukolsky, W. T. Vetterling and B. P. Flannery, Numerical Recipes. The Art of Scientific Computing, 3rd Edition, Cambridge University Press, New York, 2007.

6. Appendix

Here we list all the MATLAB code used in implementing the algorithm. The sign # means that a row is terminated in the listing, but not in the corresponding M–file.

heatsplinelevelhanketime.m

```
function [A,h]=heatsplinelevelhanketime(K,J,Jsol,C,tol,tp,varargin)
% function [A,h]=heatsplinelevelhanke(K,J,Jsol,C,tol,tp,rir,microt,T)
% Solves the equation u_t=(aa(u)u_x)_x, u(0,x)=\omega_0
% u(0,t)=f(t), u(1,t)=g(t) with t\in[0,T] and x\in[0,1] and
% computes an approximation of the function aa(u) given the solution u.
% In the method a bicubic spline s=s(t,x) of u=u(x,t) is constructed. Then
% intervals with relative radius rir are set around the coefficients of the
% spline functions. For each level c_m and each time value t_i we look for
% intervals such that c_m belongs to the range of s(t_i,x) restricted to
% the interval. Those intervals are bisected and each half is also bisected
% if c_m belongs to the range of s(t_i,x) restricted to the half. The
% process is repeated. If it happens that the length of such an interval is
% less than the tolerance tol it is saved in the queue tolqueue. The union
% of the entries of the list is empty or consists of one or several
% disjoint intervals. We let the matrix element h_{im} be one of these
% disjoint interval (or we let h_{im} be empty if the union is empty).
% Then for every (non-ordered) time value pair t_1 and t_2 and every
% (non-ordered) level pair c_1 and c_2 we take the corresponding H
% component pairs with x values say h_{12} and h_{11} and h_{22}
% and h_{21}. Then we solve the equation system
% {aa(c_2)s_x(t_1,h_{12})-aa(c_1))s_x(t_1,h_{11})
% =\int_{h_{11}}^{h_{12}}s_t(t_1,x)dx.
% {aa(c_2)s_x(t_2,h_{22})-aa(c_1))s_x(t_2,h_{21})
% =\int_{h_{11}}^{h_{12}}s_t(t_2,x)dx.
% with respect to aa(c_1) and aa(c_2).
% Now after solving the equation system we know that aa(c_1) be-
% longs to some interval. After pairing with say aa(c_3) we get
```

```
% another interval which aa(c_1) should belong to. We then take the
% intersection of all such intervals. The same goes for aa(c_2)
% and all other. The intersection is also taken over all time values in tp.
% Input:
% K   = number of spline time steps
% J   = number of spline space steps
% Jsol= number of space steps in the solver
% C   = vector with levels
% tol = maximal length of the intervals in the queue tolqueue.
% tp  = the time values for which x intervals are found
% Optional input:
% rir    = relative interval radius. Default is 0.01.
% microt = ratio between a short time step for computing derivatives
%          with finite differences and the time step of the solution.
%          Default is 10^(-4).
% T      = the value of time in the end. Default is 6.
% Output:
% h = Matrix with x values of intervals which contain tp and such that
%     C is possibly assumed on them.
% A=  The first row contains some levels, the second contains
%     intervals which contain aa(u).
% Default values
rir=0.01;
microt=10^(-4);
T=6;
if nargin>=7
  rir=varargin{1};
end
if nargin>=8
  microt=varargin{2};
end
if nargin>=9
  T=varargin{3};
end
ts=linspace(0,T,K+1);
k=1;
t2=zeros(1,1+K*3);
% Adds 2 points to every point in ts for a second order
% finite difference time derivative approximation.
for j=3:3:length(t2)-1
  t2(j-1:j+1)=ts(k+1)+microt*(ts(k+1)-ts(k))*[-1 0 1];
  k=k+1;
end
xs=linspace(0,1,J+1); % The spline net
xsol=linspace(0,1,Jsol+1); % The solution net
options=odeset('RelTol',10^(-8),'AbsTol',10^(-8));
```

```
% The zero on the following line comes from slab symmetry.
disp('Solving the forward problem...')
tic
u=pdepe(0,@pdefun,@pdeini,@pdebc,xsol,t2,options); % pde solver
toc
disp('Constructing and evaluating splines...')
tic
% Removes one point in the beginning due to lack of regularity.
start=2;
t2=t2(3*start-4:end);
ts=ts(start:end);
us=zeros(length(t2),length(xs));
uxs=us;
for k=1:length(t2)
  % Interpolation of u and ux
  [us(k,:),uxs(k,:)]=pdeval(0,xsol,u(k+3*start-5,:),xs);
end
s=bicubictransform2interval(us,uxs,t2,xs,rir); % Makes bicubic splines.
figure
surf(xs,t2,us) % Plots the forward solution u.
drawnow
% The matrix H
h=intval(nan(length(tp),length(C)));
K=zeros(1,length(tp)); % Spline indices in time
for pp=1:length(tp)
  time_number=pp
  time=tp(pp)
  for c=1:length(C)
    value_number=c
    value=C(c)
    % Evaluates the spline on the whole space domain.
    [se,K(pp)]=bicubicevalintervaltime(s,ts,xs,tp(pp),infsup(0,1));
    if ~in(C(c),se)
      continue
    end
    queue=infsup(0,1);
    tolqueue=[];
    while ~isempty(queue)
      xq=queue(1);
      queue=queue(2:end);
      xq1=infsup(inf(xq),mid(xq));
      xq2=infsup(mid(xq),sup(xq));
      % Evaluates the spline on the halves.
      se1=bicubicevalintervaltime(s,ts,xs,tp(pp),xq1,K(pp));
      se2=bicubicevalintervaltime(s,ts,xs,tp(pp),xq2,K(pp));
      if in(C(c),se1)               % Is x at the set-valued level curve?
```

```
        if diam(xq1)>tol
          queue=[queue xq1];         % Bisects further.
        else
          tolqueue=[tolqueue xq1];
        end
      end
      if in(C(c),se2)                % Is x at the set-valued level curve?
        if diam(xq2)>tol
          queue=[queue xq2];         % Bisects further.
        else
          tolqueue=[tolqueue xq2];
        end
      end
    end
    if ~isempty(tolqueue)
      h(pp,c)=connect2first(tolqueue); % Chooses one of the disjoint
                                       % intervals.
    end
  end
end
hold off
disp('Solves the 2x2 equation systems...')
st=bicubicddtinterval(s,ts); % The time derivative of the spline
sx=bicubicddxinterval(s,xs); % The spacee derivative of the spline
A=intval(ones(1,length(C)))*infsup(-inf,inf);
B=intval(zeros(2));
b=intval(zeros(2,1));
% The first time value in the 2x2 equation system
for k1=1:length(tp)-1
  % The second time value in the 2x2 equation system
  for k2=k1+1:length(tp)
    % The first value of u
    for c1=1:length(C)-1
      if isnan(h(k1,c1)) || isnan(h(k2,c1))
        continue
      end
      % The second value of u
      for c2=1+c1:length(C)
        % Disjoint integration limits
        if isnan(h(k1,c2)) || isnan(h(k2,c2)) || ~isnan(intersect(h(k1,c1),h(k1,c2))) || #
        ~isnan(intersect(h(k2,c1),h(k2,c2)))
          continue
        end
        % The first row of the matrix in the 2x2 equation system
        B(1,:)=[bicubicevalintervaltime(sx,ts,xs,tp(k1),h(k1,c2),K(k1)) #
        -bicubicevalintervaltime(sx,ts,xs,tp(k1),h(k1,c1),K(k1))];
```

```
        % The first right hand side integral
        % Determines which limit is lower und which is upper.
        if sup(h(k1,c1))<sup(h(k1,c2))
          b(1)=intbicubicxinterval(st,ts,xs,tp(k1),h(k1,c1),h(k1,c2));
        else
          b(1)=-intbicubicxinterval(st,ts,xs,tp(k1),h(k1,c2),h(k1,c1));
        end
         % The second row of the matrix in the 2x2 equation system
        B(2,:)=[bicubicevalintervaltime(sx,ts,xs,tp(k2),h(k2,c2),K(k2)) #
        -bicubicevalintervaltime(sx,ts,xs,tp(k2),h(k2,c1),K(k2))];
        % The second right hand side integral
        % Determines which limit is lower und which is upper.
        if sup(h(k2,c1))<sup(h(k2,c2))
          b(2)=intbicubicxinterval(st,ts,xs,tp(k2),h(k2,c1),h(k2,c2));
        else
          b(2)=-intbicubicxinterval(st,ts,xs,tp(k2),h(k2,c2),h(k2,c1));
        end
        k1
        k2
        c1
        c2
        % Checks the determinant
        deter=B(1,1)*B(2,2)-B(1,2)*B(2,1);
        inf(deter)
        sup(deter)
        % Solves the equation system
        sol=B\b;
        if ~isnan(sol(1)) && ~isnan(sol(2))
          if ~in(0,deter)
            % Intersects the solution of the interval equation system from
            % INTLAB with the solution from Cramer's rule.
            sol(2)=intersect(sol(2),(B(1,1)*b(2)-b(1)*B(2,1))/deter);
            sol(1)=intersect(sol(1),(b(1)*B(2,2)-B(1,2)*b(2))/deter);
          end
          A(c1)=intersect(sol(2),A(c1));
          A(c2)=intersect(sol(1),A(c2));
        end
      end
    end
  end
end
A=[C;A];
figure
mi=mid(A(1,:));
hold on
% Plots the diffusion coefficient and the upper and lower bound of its
```

```
% estimation.
for l=1:length(A(2,:))
  if ~isnan(A(2,l))
    plot(mi(l),sup(A(2,l)),'p',mi(l),inf(A(2,l)),'^',mi,aa(mi),'d')
    legend('Upper bound','Lower bound','Exact','Location','BestOutside')
  end
end
plot(mi,aa(mi),'r')
hold off
toc
end

% Required information for the PDE solver.
function [c,h,s]=pdefun(~,~,u,dudx) % u_t=(aa(u)u_x)_x
c = 1;
h = aa(u).*dudx;
s = 0; % No source
end

function u0 = pdeini(x) % Initial data
% u0 = u(0,x)
u0=0.020;
end

function [pl,ql,pr,qr]=pdebc(~,ul,~,ur,t) % u(0,t)=f(t), u(1,t)=g(t)
pl = ul-f(t); % The left boundary
ql = 0;       % Dirichlet boundary condition on the left
pr = ur-g(t); % The right boundary
qr = 0;       % Dirichlet boundary condition on the right
end

% Diffusion coefficient a(u)
function A = aa(u)
a=[1 .7 3*.35^2;0.35 0.35^2 .35^3;1.1 1.1^2 1.1^3]\[0 -.1 .11]';
A=.5+a(1)*u+a(2)*u.^2+a(3)*u.^3;
end

% Left boundary function
function F = f(t)
% f(t)=u(t,0)
% f(0)=\omega_0.
b=log(1.04/0.36)/(2*0.35-6-0.35^2/6);
g=b*0.35^2;
a=log(0.36)-6*b-g/6;
F=exp(a+b*t+g./t)+0.02;
end
```

```
% Right boundary function
function G = g(t)
% g(t) = u(t,1)%
% g(0)=\omega_0.
b=log(0.16/0.155)/(2*4.5-6-4.5^2/6);
g=b*4.5^2;
a=log(0.155)-6*b-g/6;
G=exp(a+b*t+g./t)+0.02;
end
```

bicubictransform2interval.m

```
function a=bicubictransform2interval(us,uxs,t2,xs,rir)
% function a=bicubictransform2interval(us,uxs,t2,xs,rir)
% Creates splines for the function us defined on time grid ts and the
% space grid xs and with the derivative uxs. t2 is ts with two extra points
% around every points. Those points are used for a second order finite
% difference derivative approximation. The spline is stored in a
% (length(ts)-1)*(length(xs)-1)*16 tensor describing the piecewise defined
% polynomials with 16 coefficients each.
% Input:
% us  = the grid function whose spline will be created and which is
%       defined on the time grid t2 and space grid xs
% uxs = the approximate derivative of us with respect to xs
% t2  = the time grid of us and uxs with two uniformly spaced points
%       around every point in ts.
% xs  = the space grid of us and uxs
% rir = relative interval radius
% Output:
% a   = the spline described by a (length(ts)-1)*(length(xs)-1)*16 tensor
s=size(us);
if s(1)~=length(t2) || s(2)~=length(xs)
  error('The sizes of the arguments are not consistent.')
end
ts=t2(2:3:end-1); % ts is the original grid without added points.
% Initialization of the right hand sides
f=zeros(length(ts),length(xs));
fx=f;
ft=f;
fxt=f;
k=1;
for j=2:3:length(t2)-3
  f(k,:)=us(j,:);
  fx(k,:)=uxs(j,:);
  % Second order time derivative approximation
  ft(k,:)=(us(j+1,:)-us(j-1,:))/2/(t2(j+1)-t2(j));
```

```
  fxt(k,:)=(uxs(j+1,:)-uxs(j-1,:))/2/(t2(j+1)-t2(j));
  k=k+1;
end
% The inverse system matrix for the spline problem on the unit square
Ainv=bicubicinverse;
a=zeros(length(ts)-1,length(xs)-1,16);
b=zeros(16,1);
% Notice the time and space scale in the right hand side b.
for k=1:length(ts)-1
  for j=1:length(xs)-1
    b(:)=[f(k,j);f(k+1,j);f(k,j+1);f(k+1,j+1);(ts(k+1)-ts(k))*[ft(k,j);ft(k+1,j);ft(k,j+1);#
    ft(k+1,j+1)];(xs(j+1)-xs(j))*[fx(k,j);fx(k+1,j);fx(k,j+1);fx(k+1,j+1)];(ts(k+1)-ts(k))*#
    (xs(j+1)-xs(j))*[fxt(k,j);fxt(k+1,j);fxt(k,j+1);fxt(k+1,j+1)]];
    a(k,j,:)=Ainv*b;
  end
end
% An interval is set around every spline coefficient.
a=intval(a)*(1+infsup(-rir,rir));
```

bicubicinverse.m

```
function Ainv=bicubicinverse
% function Ainv=bicubicinverse
% Creates the inverse system matrix for the bicubic spline problem on
% the unit square.
% Output:
% Ainv = the inverse system matrix for the bicubic spline problem on
% the unit square
Ainv=zeros(16);
Ainv(1,1)=1;
Ainv(2,5)=1;
Ainv(3,1:6)=[-3 3 0 0 -2 -1];
Ainv(4,1:6)=[2 -2 0 0 1 1];
Ainv(5,9)=1;
Ainv(6,13)=1;
Ainv(7,9:14)=[-3 3 0 0 -2 -1];
Ainv(8,9:14)=[2 -2 0 0 1 1];
Ainv(9,1:3)=[-3 0 3];
Ainv(9,9:11)=[-2 0 -1];
Ainv(10,5:7)=[-3 0 3];
Ainv(10,13:15)=[-2 0 -1];
Ainv(11,:)=[9 -9 -9 9 6 3 -6 -3 6 -6 3 -3 4 2 2 1];
Ainv(12,:)=[-6 6 6 -6 -3 -3 3 3 -4 4 -2 2 -2 -2 -1 -1];
Ainv(13,1:3)=[2 0 -2];
Ainv(13,9:11)=[1 0 1];
Ainv(14,5:7)=[2 0 -2];
Ainv(14,13:15)=[1 0 1];
```

```
Ainv(15,:)=[-6 6 6 -6 -4 -2 4 2 -3 3 -3 3 -2 -1 -2 -1];
Ainv(16,:)=[4 -4 -4 4 2 2 -2 -2 2 -2 2 -2 1 1 1 1];
```

bicubicevalinterval.m

```
function f=bicubicevalinterval(A,tn,xn,t,x)
% function f=bicubicevalinterval(A,tn,xn,t,x)
% Evaluates the spline A defined on the time grid tn and the space grid xn
% in the points (t,x).
% Input:
% A  = the spline created by bicubictransform2interval defined on the time
%      grid tn and the space grid xn
% tn = the time grid of the spline A
% xn = the space grid of the spline A
% t  = the time values when the spline is evaluated
% x  = the sites where the spline is evaluated
% Output:
% f  = the value of the spline at the time t on the site x
s=size(A);
t=intval(t);
x=intval(x);
if s(1)~=length(tn)-1 || s(2)~=length(xn)-1
  error('The sizes of the arguments are not consistent.')
end
for k=1:length(tn)-1
  if tn(k)>=tn(k+1)
    error('The second argument is not an increasing vector.')
  end
end
for j=1:length(xn)-1
  if xn(j)>=xn(j+1)
    error('The third argument is not an increasing vector.')
  end
end
% The evaluation points must be in the grid.
for p=1:length(t)
  if inf(t(p))<tn(1) || sup(t(p))>tn(end)
    error('The spline is not defined for at least one of the time values.')
  end
end
for q=1:length(x)
  if inf(x(q))<xn(1) || sup(x(q))>xn(end)
    error('The spline is not defined on at least one of the sites.')
  end
end
% Searches in which rectangle the point should be evaluated.
% Loop over all points where the spline is evaluated
```

```
f=intval(zeros(length(t),length(x)));
a=intval(zeros(1,16));
for q=1:length(x)
  for p=1:length(t)
    % Loop over the time grid of the spline
    for k=1:length(tn)-1
      if inf(t(p))<=tn(k+1) % Finds the correct time to evaluate.
        Kinf=k;
        break
      end
    end
    for k=Kinf:length(tn)-1
      if sup(t(p))<=tn(k+1) % Finds the correct time to evaluate.
        Ksup=k;
        break
      end
    end
    % Loop over the space grid of the spline
    for j=1:length(xn)-1
      if inf(x(q))<=xn(j+1) % Finds the correct space to evaluate.
        Jinf=j;
        break
      end
    end
    for j=Jinf:length(xn)-1
      if sup(x(q))<=xn(j+1) % Finds the correct space to evaluate.
        Jsup=j;
        break
      end
    end
    % Special case for short t and x.
    for k=1:length(tn)-1
      if in(t(p),infsup(tn(k),tn(k+1)))
        Kinf=k;
        Ksup=k;
        break
      end
    end
    for j=1:length(xn)-1
      if in(x(q),infsup(xn(j),xn(j+1)))
        Jinf=j;
        Jsup=j;
        break
      end
    end
    if Kinf==Ksup && Jinf==Jsup
```

```
  K=Kinf;
  J=Jinf;
  a(:)=A(K,J,:);
  % Transformation to the unit square
  t01=(t(p)-tn(K))/(tn(K+1)-tn(K));
  x01=(x(q)-xn(J))/(xn(J+1)-xn(J));
  % Evaluation
  f(p,q)=hornersplineevaluate(a,t01,x01);
elseif Kinf==Ksup && Jinf<Jsup
  K=Kinf;
  t01=(t(p)-tn(K))/(tn(K+1)-tn(K));
  x01=(infsup(inf(x(q)),xn(Jinf+1))-xn(Jinf))/(xn(Jinf+1)-xn(Jinf));
  a(:)=A(K,Jinf,:);
  f(p,q)=hornersplineevaluate(a,t01,x01);
  x01=(infsup(xn(Jsup),sup(x(q)))-xn(Jsup))/(xn(Jsup+1)-xn(Jsup));
  a(:)=A(K,Jsup,:);
  f(p,q)=hull(f(p,q),hornersplineevaluate(a,t01,x01));
  x01=infsup(0,1);
  for j=Jinf+1:Jsup-1
     a(:)=A(K,j,:);
     f(p,q)=hull(f(p,q),hornersplineevaluate(a,t01,x01));
  end
elseif Kinf>Ksup && Jinf==Jsup
  J=Jinf;
  x01=(x(q)-xn(J))/(xn(J+1)-xn(J));
  t01=(infsup(inf(t(p)),tn(Kinf+1))-tn(Kinf))/(tn(Kinf+1)-tn(Kinf));
  a(:)=A(Kinf,J,:);
  f(p,q)=hornersplineevaluate(a,t01,x01);
  t01=(infsup(tn(Ksup),sup(t(p)))-tn(Ksup))/(tn(Ksup+1)-tn(Ksup));
  a(:)=A(Ksup,J,:);
  f(p,q)=hull(f(p,q),hornersplineevaluate(a,t01,x01));
  t01=infsup(0,1);
  for k=Kinf+1:Ksup-1
     a(:)=A(k,J,:);
     f(p,q)=hull(f(p,q),hornersplineevaluate(a,t01,x01));
  end
else
  x01=(infsup(inf(x(q)),xn(Jinf+1))-xn(Jinf))/(xn(Jinf+1)-xn(Jinf));
  t01=(infsup(inf(t(p)),tn(Kinf+1))-tn(Kinf))/(tn(Kinf+1)-tn(Kinf));
  a(:)=A(Kinf,Jinf,:);
  f(p,q)=hornersplineevaluate(a,t01,x01);
  t01=(infsup(tn(Ksup),sup(t(p)))-tn(Ksup))/(tn(Ksup+1)-tn(Ksup));
  a(:)=A(Ksup,Jinf,:);
  f(p,q)=hull(f(p,q),hornersplineevaluate(a,t01,x01));
  t01=infsup(0,1);
  for k=Kinf+1:Ksup-1
```

```
         a(:)=A(k,Jinf,:);
         f(p,q)=hull(f(p,q),hornersplineevaluate(a,t01,x01));
      end
      x01=(infsup(xn(Jsup),sup(x(q)))-xn(Jsup))/(xn(Jsup+1)-xn(Jsup));
      t01=(infsup(inf(t(p)),tn(Kinf+1))-tn(Kinf))/(tn(Kinf+1)-tn(Kinf));
      a(:)=A(Kinf,Jsup,:);
      f(p,q)=hull(f(p,q),hornersplineevaluate(a,t01,x01));
      t01=(infsup(tn(Ksup),sup(t(p)))-tn(Ksup))/(tn(Ksup+1)-tn(Ksup));
      a(:)=A(Ksup,Jsup,:);
      f(p,q)=hull(f(p,q),hornersplineevaluate(a,t01,x01));
      t01=infsup(0,1);
      for k=Kinf+1:Ksup-1
         a(:)=A(k,Jsup,:);
         f(p,q)=hull(f(p,q),hornersplineevaluate(a,t01,x01));
      end
      x01=infsup(0,1);
      for j=Jinf+1:Jsup-1
        t01=(infsup(inf(t(p)),tn(Kinf+1))-tn(Kinf))/(tn(Kinf+1)-tn(Kinf));
        a(:)=A(Kinf,j,:);
        f(p,q)=hull(f(p,q),hornersplineevaluate(a,t01,x01));
        t01=(infsup(tn(Ksup),sup(t(p)))-tn(Ksup))/(tn(Ksup+1)-tn(Ksup));
        a(:)=A(Ksup,j,:);
        f(p,q)=hull(f(p,q),hornersplineevaluate(a,t01,x01));
        t01=infsup(0,1);
        for k=Kinf+1:Ksup-1
          a(:)=A(k,j,:);
          f(p,q)=hull(f(p,q),hornersplineevaluate(a,t01,x01));
        end
      end
    end
  end
end
```

hornersplineevaluate.m

```
function p=hornersplineevaluate(a,t,x)
% function p=hornersplineevaluate(a,t,x)
% Evaluates the a spline at time t and place x. The variables
% t and x belong to [0,1]. The Horner scheme is used in two orders.
% Input:
% a = the vector with the spline coefficients
% t = the time
% x = the place
% Output:
% p = the value of the spline at time t and place x
pt=a(1)+t*(a(2)+t*(a(3)+t*a(4)))+x*(a(5)+t*(a(6)+t*(a(7)+t*a(8)))+#
x*(a(9)+t*(a(10)+t*(a(11)+t*a(12)))+x*(a(13)+t*(a(14)+t*(a(15)+t*a(16))))));
```

```
px=a(1)+x*(a(5)+x*(a(9)+x*a(13)))+t*(a(2)+x*(a(6)+x*(a(10)+x*a(14)))+#
t*(a(3)+x*(a(7)+x*(a(11)+x*a(15)))+t*(a(4)+x*(a(8)+x*(a(12)+x*a(16))))));
p=intersect(pt,px);
```

connect2first.m

```
function conint=connect2first(s)
% function conint=connect2first(s)
% This function makes a hull about all intervals in the vector s which are
% connected to the first interval in s, s(1),
% Input:
% s = vector of intervals
% Output:
% conint = interval which encloses all intervals connected to s(1)
bool=true;
while bool
  bool=false;
  d=2;
  while d<=length(s)
    if ~isnan(intersect(s(1),s(d)))
      s=[hull(s(1),s(d)) s(2:d-1) s(d+1:end)];
      bool=true;
      break;
    end
    d=d+1;
  end
end
conint=s(1);
```

bicubicddtinterval.m

```
function A=bicubicddtinterval(A,t)
% function A=bicubicddtinterval(A,t)
% Differentiate the bicubic spline A=A(t,x), which is piecewise
% polynomial, with respect to t. The spline was created by
% bicubictransform2interval.
% Input:
% A = a spline created by bicubictransform2interval defined on a time grid
% t and a space grid x
% t = the time grid where the spline A is defined
% Output:
% A = the partial derivative of the spline A with respect to t
s=size(A);
if s(1)~=length(t)-1
  error('The sizes of the arguments are not consistent.')
end
a=intval(zeros(1,16));
```

```
for j=1:s(2)
  for k=1:s(1)
    a(:)=A(k,j,:);
    A(k,j,:)=[a(2) 2*a(3) 3*a(4) 0 a(6) 2*a(7) 3*a(8) 0 a(10) 2*a(11) 3*a(12) 0 #
    a(14) 2*a(15) 3*a(16) 0]/(t(k+1)-t(k));
  end
end
```

bicubicddxinterval.m

```
function A=bicubicddxinterval(A,x)
% function A=bicubicddxinterval(A,x)
% Differentiates the bicubic spline A=A(t,x), which is piecewise
% polynomial, with respect to x. The spline was created by
% bicubictransform2interval.
% Input:
% A = a spline created by bicubictransform2interval defined on a time grid
% t and a space grid x
% x = the space grid where the spline A is defined
% Output:
% A = the partial derivative of the spline A with respect to x
s=size(A);
if s(2)~=length(x)-1
  error('The sizes of the arguments are not consistent.')
end
a=intval(zeros(1,16));
for j=1:s(2)
  for k=1:s(1)
    a(:)=A(k,j,:);
    A(k,j,:)=[a(5) a(6) a(7) a(8) 2*[a(9) a(10) a(11) a(12)] 3*[a(13) a(14) a(15) #
    a(16)] zeros(1,4)]/(x(j+1)-x(j));
  end
end
```

intbicubicxinterval.m

```
function I=intbicubicxinterval(A,t,x,tp,xmin,xmax)
% function I=intbicubicxinterval(A,t,x,tp,xmin,xmax)
% Computes the integral of the spline A=A(t,x) at time value tp
% from xmin to xmax.
% The spline was created by bicubictransform2interval.
% Input:
% A  =   a spline created by bicubictransform2interval defined on a time
%        grid t and a space grid x
% x  =   the space grid where the spline A is defined
% tp =   time value for the integral to be computed (number)
% xmin = lower integral limit (interval)
```

```
% xmax = upper integral limit (interval)
% Output:
% I = the integral of A from xmin to xmax,
s=size(A);
a=intval(zeros(1,16));
xmin=intval(xmin);
xmax=intval(xmax);
if sup(xmin)>inf(xmax)
  error('Wrong integration interval.')
end
for k=1:length(t)-1
  if t(k)>=t(k+1)
    error('The second argument is not an increasing vector.')
  end
end
for j=1:length(x)-1
  if x(j)>=x(j+1)
    error('The third argument is not an increasing vector.')
  end
end
if inf(xmin)<x(1) || sup(xmax)>x(end) || tp<t(1) || tp>t(end)
  error('The integral is not defined outside the spline grid.')
end
if s(2)~=length(x)-1 || s(1)~=length(t)-1
  error('The sizes of the arguments are not consistent.')
end
% Determines in what spline interval the limits of integration are.
for j=1:length(x)-1
  if inf(xmin)<=x(j+1) % The spline for the infimum of the lower limit
    Linf=j;
    break
  end
end
for j=Linf:length(x)-1
  if sup(xmin)<=x(j+1) % The spline for the supremum of the lower limit
    Lsup=j;
    break
  end
end
for j=Lsup:length(x)-1
  if inf(xmax)<=x(j+1) % The spline for the infimum of the upper limit
    Uinf=j;
    break
  end
end
for j=Uinf:length(x)-1
```

```
  if sup(xmax)<=x(j+1) % The spline for the supremum of the upper limit
    Usup=j;
    break
  end
end
for k=1:length(t)-1
  if tp<=t(k+1)
    t01=intval((tp-t(k))/(t(k+1)-t(k)));
    break
  end
end
% Lower part, zero if the lower limit is thin.
% The thin part is due to the fact that the integral domain is connected.
a(:)=A(k,Lsup,:)*(x(Lsup+1)-x(Lsup));
x2=(sup(xmin)-x(Lsup))/(x(Lsup+1)-x(Lsup));
if Linf==Lsup
  x1=(xmin-x(Lsup))/(x(Lsup+1)-x(Lsup));
  LI=hornersplineintegral(a,t01,x1,x2,true);
else
  x1=(infsup(x(Lsup),sup(xmin))-x(Lsup))/(x(Lsup+1)-x(Lsup));
  LI=hornersplineintegral(a,t01,x1,x2,true);
  LIthin=hornersplineintegral(a,t01,0,x2,false);
  x1=infsup(0,1);
  for j=Lsup-1:-1:Linf+1
    a(:)=A(k,j,:)*(x(j+1)-x(j));
    LI=hull(LI,LIthin+hornersplineintegral(a,t01,x1,1,true));
    LIthin=LIthin+hornersplineintegral(a,t01,0,1,false);
  end
  a(:)=A(k,Linf,:)*(x(Linf+1)-x(Linf));
  x1=(infsup(inf(xmin),x(Linf+1))-x(Linf))/(x(Linf+1)-x(Linf));
  LI=hull(LI,LIthin+hornersplineintegral(a,t01,x1,1,true));
end
% Upper part, zero if the upper limit is thin.
% The thin part is due to the fact that the integral domain is connected.
a(:)=A(k,Uinf,:)*(x(Uinf+1)-x(Uinf));
x1=(inf(xmax)-x(Uinf))/(x(Uinf+1)-x(Uinf));
if Uinf==Usup
  x2=(xmax-x(Uinf))/(x(Uinf+1)-x(Uinf));
  UI=hornersplineintegral(a,t01,x1,x2,true);
else
  x2=(infsup(inf(xmax),x(Uinf+1))-x(Uinf))/(x(Uinf+1)-x(Uinf));
  UI=hornersplineintegral(a,t01,x1,x2,true);
  UIthin=hornersplineintegral(a,t01,x1,1,false);
  x2=infsup(0,1);
  for j=Uinf+1:Usup-1
    a(:)=A(k,j,:)*(x(j+1)-x(j));
```

```
    UI=hull(UI,UIthin+hornersplineintegral(a,t01,0,x2,true));
    UIthin=UIthin+hornersplineintegral(a,t01,0,1,false);
  end
  a(:)=A(k,Usup,:)*(x(Usup+1)-x(Usup));
  x2=(infsup(x(Usup),sup(xmax))-x(Usup))/(x(Usup+1)-x(Usup));
  UI=hull(UI,UIthin+hornersplineintegral(a,t01,0,x2,true));
end
% Middle part of the integral (thin)
a(:)=A(k,Lsup,:)*(x(Lsup+1)-x(Lsup));
x1=(sup(xmin)-x(Lsup))/(x(Lsup+1)-x(Lsup));
if Lsup==Uinf
  x2=(inf(xmax)-x(Uinf))/(x(Uinf+1)-x(Uinf));
  MI=hornersplineintegral(a,t01,x1,x2,false);
else
  MI=hornersplineintegral(a,t01,x1,1,false);
  for j=Lsup+1:Uinf-1
    a(:)=A(k,j,:)*(x(j+1)-x(j));
    MI=MI+hornersplineintegral(a,t01,0,1,false);
  end
  x2=(inf(xmax)-x(Uinf))/(x(Uinf+1)-x(Uinf));
  MI=MI+hornersplineintegral(a,t01,0,x2,false);
end
I=LI+UI+MI;
```

hornersplineintegral.m

```
function I=hornersplineintegral(a,t,x1,x2,b)
% function I=hornersplineintegral(a,t,x1,x2,b)
% Computes the integral of a spline at time value t from x1 to x2. The variables
% t, x1 are x2 belong to [0,1]. The Horner scheme is used if l is true.
% Input:
% a  = the vector with the spline coefficients in the patch
% t  = the time value (number)
% x1 = lower limit of the integral (interval)
% x2 = upper limit of the integral (interval)
% b  = should be set false if x1 and x2 are numbers to save computation
% time.
% Output:
% I = the integral of the spline from x1 to x2 for the time value t
if ~b
  I=(a(1)+t*a(2)+t^2*a(3)+t^3*a(4))*(x2-x1)+(a(5)+t*a(6)+t^2*a(7)+t^3*a(8))*(x2^2-x1^2)/2+#
  (a(9)+t*a(10)+t^2*a(11)+t^3*a(12))*(x2^3-x1^3)/3+#
  (a(13)+t*a(14)+t^2*a(15)+t^3*a(16))*(x2^4-x1^4)/4;
else
  I2=x2*(a(1)+t*a(2)+t^2*a(3)+t^3*a(4)+x2*((a(5)+t*a(6)+t^2*a(7)+t^3*a(8))/2+#
  x2*((a(9)+t*a(10)+t^2*a(11)+t^3*a(12))/3+x2*(a(13)+t*a(14)+t^2*a(15)+t^3*a(16))/4)));
  I1=x1*(a(1)+t*a(2)+t^2*a(3)+t^3*a(4)+x1*((a(5)+t*a(6)+t^2*a(7)+t^3*a(8))/2+#
```

```
  x1*((a(9)+t*a(10)+t^2*a(11)+t^3*a(12))/3+x1*(a(13)+t*a(14)+t^2*a(15)+t^3*a(16))/4)));
  I=(x2-x1)*(a(1)+t*a(2)+t^2*a(3)+t^3*a(4)+(a(5)+t*a(6)+t^2*a(7)+t^3*a(8))*(x2+x1)/2+#
  (a(9)+t*a(10)+t^2*a(11)+t^3*a(12))*(x2^2+x2*x1+x1^2)/3+#
  (a(13)+t*a(14)+t^2*a(15)+t^3*a(16))*(x2^3+x2^2*x1+x2*x1^2+x1^3)/4);
  I=intersect(I2-I1,I);
end
```