REPARAMETRIZATION OF PDES

DAVID LUTTERKORT

ABSTRACT. This report deals with simplifying the geometry underlying a PDE problem with curved boundaries by reparametrizing the domain over a simpler (rectangular) domain and transforming the original PDE into one with an identical solution over the new domain.

The aim of this project was to develop a program that transforms the given PDE *symbolically* using Maple. The result of these efforts is the Maple program `repde.mpl` , which is included at the end of this report.

CONTENTS

1. Problem Statement 1
2. Some particular reparametrizations 2
3. Extension to 3D problems 6
4. Using the Maple Program 7
5. A Brief Tour of the Maple Code 8

1. PROBLEM STATEMENT

We are given a PDE problem on the domain Ω in the (s,t)–plane,

$$\begin{aligned} Lw &= f \quad \text{on } \Omega \\ w &= g \quad \text{on } \partial\Omega \end{aligned}$$

where $w(s,t)$ is the desired solution, L is a linear differential operator, f is the forcing function and g gives the (Dirichlet) boundary conditions. Assume that we are given a reparametrization $\Phi(x,y) = (s(x,y),\, t(x,y))$ of Ω over the unit square $R = [0,1] \times [0,1]$. We can now rewrite the original PDE problem as

$$\begin{aligned} \bar{L}u &= \bar{f} \quad \text{on } R \\ u &= \bar{g} \quad \text{on } \partial R. \end{aligned}$$

The new operator $\bar{L}$, the new forcing function $\bar{f}$ and the new boundary conditions $\bar{g}$ have to be chosen so that the solutions u and w correspond to each other by

$$u(x,y) = w(\Phi(x,y)).$$

Date: May 7, 1998.

1.1. **Details of rewriting the PDE.** It is clear what $\bar{f}$ and $\bar{g}$ should be: we get them by simply plugging the reparametrization into f and g,

$$\begin{aligned}\bar{f}(x,y) &= f(\Phi(x,y))\\ \bar{g}(x,y) &= g(\Phi(x,y)).\end{aligned}$$

The only remaining difficulty then is finding the new differential operator $\bar{L}$. Since L is linear, it has the form

$$L(u)(s,t) = \sum_i c_i(s,t)\, D_i(u)(s,t),$$

where the c_i are arbitrary functions independent of u, and the D_i are partial derivatives with respect to the variables s and t.

There are two tasks to be performed to get $\bar{L}$: we have to replace the c_i by functions $\bar{c}_i = c_i \circ \Phi$ and we have to express the partials D_i, which are a combination of $\partial/\partial s$ and $\partial/\partial t$ by partials $\bar{D}_i$, which are a combination of $\partial/\partial x$ and $\partial/\partial y$. It is easy to obtain the $\bar{D}_i$ from the D_i if we know the inverse Φ^{-1} of Φ: we only need to apply the chain rule. This is not practical since inverting Φ symbolically is impossible even for very simple reparametrizations. We can, though, exploit the linearity of differentiation to compute the $\bar{D}_i$ from the D_i and Φ. Applying the chain–rule yields a linear equation in the tangent vectors of the solution of the PDE,

$$\begin{aligned}\frac{\partial}{\partial x} &= \frac{\partial}{\partial s} s_x + \frac{\partial}{\partial t} t_x\\ \frac{\partial}{\partial y} &= \frac{\partial}{\partial s} s_y + \frac{\partial}{\partial t} t_y.\end{aligned}$$

The derivatives s_x, t_x, etc., are just the partials of the first and second component of Φ. Solving this linear system for $\partial/\partial s$ and $\partial/\partial t$ gives us explicit expressions for the partials of the old solution w in terms of the partials of the new solution u. The system has always a unique solution, since Φ is a reparametrization and has therefore a non–singular Jacobian everywhere.

We can compute higher order derivatives of w, say w_{st}, simply by applying the expression for w_s to the one for w_t. This is all we need to compute the $\bar{D}_i$ from the D_i and Φ. Note that the resulting operator $\bar{L}$ will still be linear and of the same order as L, although the coefficients $\bar{c}_i$ will be more complicated and the change of operator might produce non–zero coefficients $\bar{c}_i$ even if the corresponding c_i is zero.

2. Some particular reparametrizations

2.1. **Domains with two straight sides.** Let Ω be bounded by $s = 0$ and $s = 1$ on the left and on the right; let the top and bottom of Ω be given by two functions $a(s)$ and $b(s)$ such that

$$\Omega = \{\,(s,t) \mid 0 \leq s \leq 1, b(s) \leq t \leq a(s)\,\}\,.$$

We can reparametrize Ω simply by

$$\begin{aligned} s(x,y) &= x \\ t(x,y) &= y\,a(x) + (1-y)\,b(x). \end{aligned}$$

This means that we do not change the parametrization in the s–direction and that t varies for a fixed x along a straight line joining points on a and b with the same parameters.

Application to domains with four curved boundaries. If Ω has four curved boundaries and is given by

$$\Omega = \{\,(s,t) \mid b(s) \leq t \leq a(s), c(t) \leq s \leq d(t)\,\}\,,$$

we could apply the previous reparametrization twice. First, we find numbers C and D such that $C \leq c(t)$ and $D \geq d(t)$for all values of t that actually occur in points that are inside Ω. We reparametrize the domain enclosed by $s = C$, $s = D$ and a and b as in Section 2.1, obtaining a reparametrization Φ_1. In this new coordinate system, the curves $s = c(t)$ and $s = d(t)$ are expressed by

$$\begin{aligned} x &= c(y\,a(x) + (1-y)\,b(x)) \\ x &= d(y\,a(x) + (1-y)\,b(x)). \end{aligned} \tag{2.1}$$

We need to solve each of these two equations symbolically for x. This yields two equations $x = \bar{c}(y)$ and $x = \bar{d}(y)$ to which we can apply the strategy from Section 2.1 giving us a reparametrization Φ_2. The final reparametrization that maps R to Ω can now be obtained as $\Phi = \Phi_2 \circ \Phi_1$.

Unfortunately, this approach is not very practical. It is strongly limited by the fact that the equations (2.1) need to be solved explicitly, which causes big difficulties even for simple boundaries. If all four boundaries, for example, are quadratic polynomials, the equations (2.1) will both be of degree four in x and have no explicit, closed–form solution. Maple can represent the solutions $\bar{c}$ and $\bar{d}$ using `RootOf` expressions, which represent the roots of polynomials for which it can't find expressions in terms of radicals. When these expressions are evaluated numerically, Maple cannot decide which of the four roots in our example it should choose, leading to complex solutions even though we know that real solutions exist.

It seems that the approach discussed in this subsection can only be applied to very simple geometries or in the case where an automatic equation solver can be told "by hand" how to select the right solutions to the equations.

It is therefore not possible to solve the equations (2.1) both explicitly and exactly. But if we drop the demand for an exact solution, we can get an approximate and explicit solution by approximating the exact solution numerically, e.g. by B–splines. This approach was not pursued any further in the implementation of `repde.mpl` .

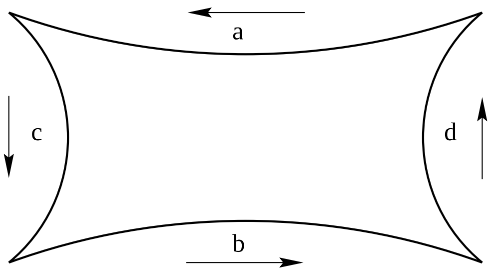

FIGURE 2.1. A non–convex foursided domain

2.2. **A simple linear interpolation scheme.** A very simple idea can be applied to foursided domains, for which we have a parametrization of the boundary curves a, b, c, and d. Let

$$\Omega = \{\, (s,t) \mid b(p) \leq t \leq a(p), c(q) \leq s \leq d(q) \; for\ all\ 0 \leq p, q \leq 1 \,\} .$$

We can use simple linear interpolation in both coordinate directions to get a reparametrization Φ:

$$\begin{aligned} s(x,y) &= (1-x)\; c(y) + x\; d(y) \\ t(x,y) &= (1-y)b(x) + ya(x). \end{aligned}$$

Φ is only a valid reparametrization of Ω, if all the lines that are used for s and t lie within Ω; otherwise we add extraneous points to the domain.

This is surely the case if Ω is convex, but is by no means necessary. Whether Φ is a valid reparametrization of Ω also depends on the parametrization of the boundary curves. If, for example, Ω is star–shaped and the boundary curves are parametrized in such a way that all the lines connecting points on opposite sides at equal parameter values pass through a point from which all of Ω is visible, then Φ does not add extraneous points to Ω. The domain in Figure 2.1 is a good example of this fact: if both a and b are parametrized from left to right we will clearly add points to the domain if we use the above Φ. By simply reversing the direction in which we walk through b as indicated in Figure 2.1, this problem disappears.

This approach has of course one big drawback: it leads to singular parametrizations, since Φ fails to be one–to–one at all points where s (resp. t) parameter lines intersect.

2.3. **Using polar coordinates.** If Ω is a threesided domain with two straight–line boundaries ℓ_0, ℓ_1 and one curved piece c, we can use polar coordinates to reparametrize it over the domain R. (see Figure 2.2) For simplicity, assume that the two straight sides intersect in the origin of the (s,t) plane, that ℓ_i encloses the angle φ_i with the x–axis and that c is given by an implicit equation $c(s,t) = 0$. Let us change to polar coordinates,

$$\begin{aligned} s(r,\varphi) &= r\cos\varphi \\ t(r,\varphi) &= r\sin\varphi \end{aligned} \tag{2.2}$$

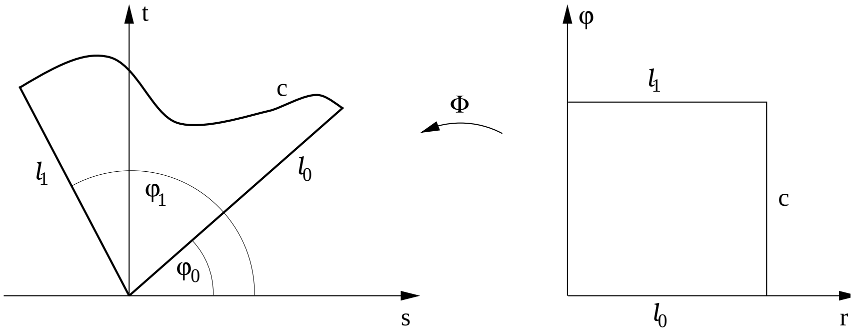

FIGURE 2.2. Threesided domain.

and substitute them into c to obtain an expression $c(r,\varphi)$. If c is not too complicated, we can solve $c(r,\varphi) = 0$ for r symbolically, which gives an expression $r(\varphi)$ for the distance from points on c to the origin parametrized by polar angle. With this expression and the correspondence

$$\begin{array}{rcl} x \in [0,1] & \leftrightarrow & \varphi \in [\varphi_0, \varphi_1] \\ y \in [0,1] & \leftrightarrow & r \in [0, r(\varphi)] \end{array}$$

in mind we substitute

$$\begin{array}{rcl} \varphi(x) & = & x\varphi_0 + (1-x)\varphi_1 \\ r(x,y) & = & y\, r(\varphi(x)) \end{array}$$

into equation (2.2) to obtain the reparametrization Φ.

The main limitation of this approach is that we have to parametrize the curved boundary c the by polar angle, which will only be possible in very simple cases.

More seriously, we are mapping a foursided domain to a threesided domain and should therefore expect a singularity in the parametrization: it is exactly the line $r = 0$ that gets mapped into the origin of the (s,t)–plane. The Jacobian of Φ is singular on this line.

Example. Let us look at the domain Ω given by

$$\begin{array}{rcl} 0 \leq & t & \leq s \\ s^2 + t^2 & \leq & (1-s)^2. \end{array}$$

The first line immediately tells us that we should choose $[0, \pi/4]$ as the range for the angle φ. The second line implies that we should use $c(s,t) = s^2 + t^2 - (1-s)^2$ as the third boundary. After changing to polar coordinates and solving for r, we get $r(\varphi) = 1/(\cos(\varphi) + 1)$. Putting this all together

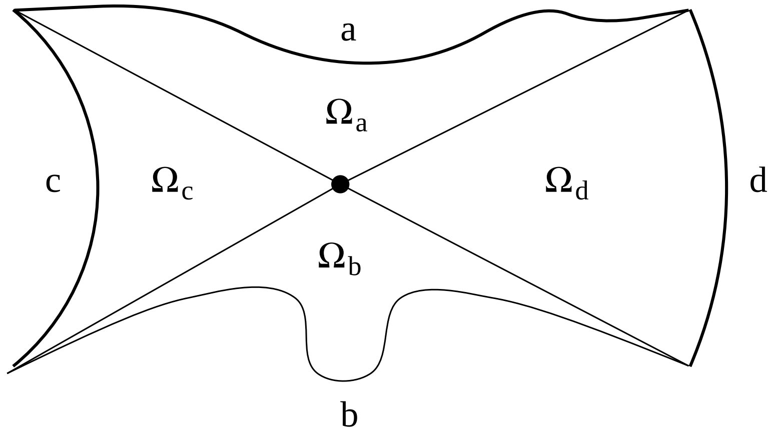

FIGURE 2.3. A star–shaped domain split into four triangular domains.

gives the reparametrization Φ as

$$\begin{aligned} s(x,y) &= \frac{y\cos(\pi x/4)}{\cos(\pi x/4)+1} \\ t(x,y) &= \frac{y\sin(\pi x/4)}{\cos(\pi x/4)+1}. \end{aligned}$$

2.4. **Star–shaped domains.** This idea can be applied to a star–shaped domain Ω by splitting Ω into four triangular pieces Ω_a, Ω_b, Ω_c and Ω_d and reparametrizing each piece with polar coordinates as indicated in Figure 2.3. We can either solve the PDE over each of the subdomains independently or piece the four subdomains together to one big rectangular domain.

Either way, we need to define suitable boundary conditions for the sides of the rectangular domain corresponding to $\varphi = \varphi_0$ and $\varphi = \varphi_1$, which are boundaries of R but map to a line in the interior of Ω. Since these two sides are mapped into the same line in Ω, the new boundary conditions have to be identical. For a steady state problem we can simply adjoin the Neumann conditions $\partial w/\partial n = 0$. For other problems, it might be hard to come up with the right boundary conditions.

3. EXTENSION TO 3D PROBLEMS

Both the general approach to simplifying the geometry underlying a PDE problem and most of the specific reparametrizations can be easily applied to 3D problems, where the solution is a trivariate function $u(r, s, t)$.

The implementation contains one function, `parametrize_trilin` to parametrize a 3D domain with curved boundaries by linear interpolation, analogous to the one described in Section 2.2.

The output routines in `repde.mpl` output a `.e`–file that cannot be used with Ellpack, since there is no discretization module in Ellpack that can handle a PDE over a rectangular domain in 3D with a general second order differential operator.

The code does perform the rewriting of the PDE though. It computes the new operator $\bar{L}$, a new forcing function $\bar{f}$ and new boundary conditions $\bar{g}$. All that is missing to use `repde.mpl` to solve 3D problems is a suitable output routine.

4. Using the Maple Program

The program `repde.mpl` offers two main functions for reparametrizing a PDE: `reparam_pde_true` for reparametrizing a PDE with a prescribed true solution and `reparam_pde` for reparametrizing a PDE with arbitrary Dirichlet boundary conditions.

Both functions need as input parameters a differential operator L, the name of the old PDE–function $u(s,t)$, the name of the new PDE–function $w(x,y)$, a parameter substitution of the form $\{s = s(x,y), t = t(x,y)\}$, and a gridsize for the finite difference discretization. In addition to this, `reparam_pde_true` needs a true solution from which it generates a forcing function f and boundary conditions g for each of the four sides of the domain. The function `reparam_pde` requires the specification of f and g directly.

The differential operator L is specified in symbolic Ellpack form as a Maple expression. This means that the derivatives of the solution are given as `wss`, `wt`, etc.

Both functions generate a `.e`–file that can be used as Ellpack input without further editing — one should check it though for lines longer than 80 characters, which are sometimes generated and instigate Ellpack to produce nasty core dumps.

Apart from the two PDE–transforming routines, `repde.mpl` contains several functions that can be used to construct reparametrizations Φ explicitly. These are `parametrize_xy`, `parametrize_bilin` and `parametrize_trilin`.

The latter function is used for 3D–problems. All of the code in `repde.mpl` can be used for 2D– as well as for 3D–problems without a change, although it could not be tested for 3D–problems since Ellpack does not support the solution of PDEs in 3D in a convenient way. It would require a major change in the routines that generate the `.e`–file to provide for the solution of 3D problems.

Example. Here is a simple input file for `repde.mpl` :

```
read(`/homes/lutterdc/class/615/proj/three.mpl`);
# Reparametrize boundary
a := (s,t) -> sqrt(1/4-(s-1/2)^2)+1;
b := (s,t) -> -(s-1/2)^2;
para := parametrize_xy(a, b, [s,t], [x,y]);
# Define operator, forcing function and b.c.
L := (s^2+t^2)*wss + (s^2-t^2)*wtt;
```

```
f := exp(s*t);
g := table([left=t^2, top=4*s^2, right=t^2, bottom=-2+2*s]);
# Create output in file test1.e
fname := `test1.e`;

reparam_pde(L, f, g, w(s,t), u(x,y), para, [10,10]);
```

This example is indeed very simple: the domain is bounded by the lines $s = 0$ and $s = 1$ on the left and on the right and by a circular arc a at the top and a parabolic piece b at the bottom. The reparametrization is computed using `parametrize_xy` which implements the idea described in Section 2.1. The call of `reparam_pde` specifies in the last argument that the new PDE is to be discretized using a 10×10 grid.

I tested `repde.mpl` with several examples with prescribed true solutions. In all cases the observed error between the computed and prescribed solution was satisfactory, giving between 4 and 6 digits of accuracy.

5. A Brief Tour of the Maple Code

I will explain the workings of `repde.mpl` by following the steps that the subroutine `reparam_pde_true (L, sol, w(s,t), u(x,y), para, grid)` takes.

It computes a forcing function `f` from the true solution `sol` using the procedure `force`, which simply applies `L` to `sol`. The boundary conditions `g` are set to be the same as `sol`. With the knowledge of `f` and `g`, `reparam_pde` is used to do the rewriting of the PDE. `reparam_pde_true` adds to the output of `reparam_pde` the true solution and the instruction for Ellpack to compute the error between the true and the computed solution.

`reparam_pde` generates the new forcing function $\bar{f}$ and the new boundary conditions $\bar{g}$ by substituting the reparametrization `para` into f and g. The change of L to $\bar{L}$ is done in `reparam_op` and the `.e`–file is printed using `ellpack_print`, which generates the `.e`–file whose name is in the variable `fname`.

`reparam_op` performs the change from L to $\bar{L}$. It computes the first partial derivatives with respect to the old variables in terms of the first partial derivatives with respect to the new variables using using the procedure `implicit_diff` as described in Section 1 and calls the procedure `make_deriv_subs` to change the higher–order derivatives in L from the old to the new variables. Applying the substitution `para` to the coefficients of the partials in L finishes the change from L to $\bar{L}$.

Much of the remaining code in `repde.mpl` deals with syntactic manipulations on expressions. The main purpose of these routines is to change back and forth between Ellpack's symbolic notation for derivatives and the equivalent Maple expressions. For example, an Ellpack–style `wss` has to be changed to Maple's `diff(w(s,t), s$2)` and vice versa.

The printing of the `.e`–file is performed by the procedures whose names start with `ellpack_print`. There is one procedure corresponding to each section of a `.e`–file. The biggest problem is the output of the new differential operator $\bar{L}$. The Maple expression for $\bar{L}$ has to be processed by the function `fortranify_eq`. This function replaces the coefficients of the derivative symbols in $\bar{L}$ by calls to FORTRAN functions; the coefficient of `uxx` is computed in the FORTRAN function `huxx(x,y)`. It does not generate any output itself. Instead, `fortranify_eq` puts the MAPLE expressions for each of the FORTRAN functions in the table `ellpack\_subroutine\_table`, which is indexed by the names of the FORTRAN functions. In addition, it adds entries for the boundary conditions, e.g., `bcleft`, and an entry for the forcing function `force` to this table. This table is later used by `ellpack_print_subpr` to generate the FORTRAN code for these functions.

Department of Computer Science, Purdue University, West Lafayette IN
E-mail address: `lutter@cs.purdue.edu`