COMPUTING PSEUDOSPHERICAL SURFACES FROM GEOMETRIC CAUCHY DATA

DAVID BRANDER

This note explains briefly how to use Matlab to compute constant Gauss curvature $K = -1$ surfaces, (also called K-surfaces, or pseudospherical surfaces) using a numerical implementation of the generalized d'Alembert method for such surfaces given by M. Toda [3]. With further recent results about the geometric Cauchy problem for these surfaces, one can compute a K-surface containing a prescribed curve with the surface normal prescribed along the curve [2], or a K-surface containing a prescribed curve as a cuspidal edge. Choosing appropriate curves makes it possible to create K-surfaces with interesting properties.

To compute the examples here, you need to put the files ksurf.m, kgcpotprinc.m, Asing.m, SGCP.m and charSGCP.m, which can be found at this link: http://davidbrander.org/software.html - into a folder where Matlab can find them. If you have a C++ compiler installed then try the function *ksurfX*, which calls some C++ mex functions. It's about twice as fast.

1. Simple examples: Prescribed singularities

We will look at some very simple examples first, and then explain the code in more detail in the following section.

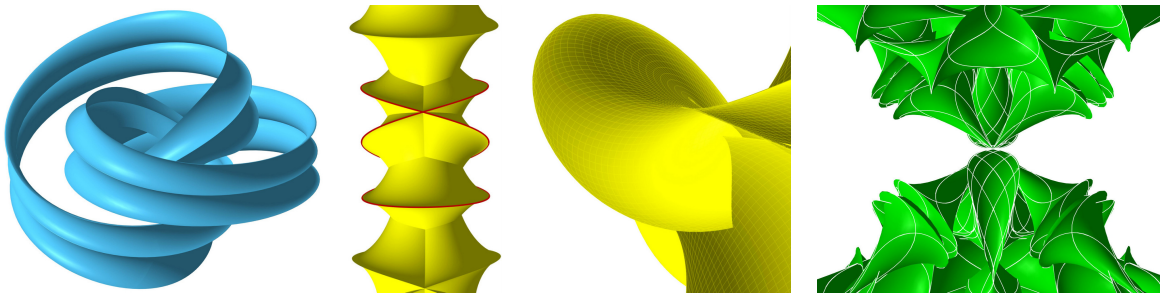

FIGURE 1. Left: Part of the unique K-surface generated by a given torus knot, which appears as the cusp line in the center. Middle: The surface generated by the red curve. It has swallowtail singularities as well as cuspidal edges. Right: A K-surface with a cone singularity at the center.

There are no complete immersed K-surfaces in $\mathbb{R}^3$. If you move far enough along a K-surface, you will come to a cusp line (like the sharp edges in Figure 1), or a more complicated singularity. A natural generalization of immersed constant curvature -1 surfaces is derived from Lorentz harmonic maps, and this type of surface is globally defined but has singularities here and there, mostly cuspidal edges. An interesting thing about these singularities is that, except for a few special cases, a singular curve determines the entire surface uniquely. More precisely, given a space curve with non-vanishing curvature κ and torsion τ satisfying $|\tau| \neq 1$, there is a *unique* generalized K-surface containing this curve as a cuspidal edge. This makes singular curves interesting as *generators* of K-surfaces.

The function *ksurf* computes a surface from a given pair of function handles *Am* and *Ap* (sometimes *Am* and *Ap* are the same) called *potentials*. The article [1] gives very simple potentials for producing pseudospherical surfaces with prescribed singularities. The Matlab functions *Asing*, *SGCP* and *charSGCP* return the potentials for the corresponding data.

Date: February 18, 2015.

1.1. **Non-characteristic singularities.** Most singularities are of this type. The singular curve determines the surface uniquely.

1.1.1. *Arbitrary non-characteristic singularities.* The function *Asing* produces the potential for a K-surface with a generic singularity. Enter

$$X = Asing(@(t)\beta(t), @(t)A(t), @(t)B(t)),$$

where $\beta(t)$, $A(t)$ and $B(t)$ are arbitrary non-vanishing functions. (The singular curve will be degenerate, i.e. not a regular curve in the coordinate domain, at a point where any of these functions vanishes). Then the command

$$ksurf(X,X,eye(2),eye(2),xinterval,yinterval,looporder),$$

where *xinterval* of the form *[x0, stepsize, stepsleft, stepsright]* describes an interval with center $x0$, and *yinterval* is similar, and *looporder* is the order of the polynomial approximation of the Fourier series involved, produces the surface with singularity determined by X. The choice of *looporder* depends on the problem and the size of the rectangle computed. Large values are more accurate but slower.

From Theorem 4.2 of [1] , the singularity is:

(1) a cuspidal edge if $A(0)+B(0) \neq 0$,
(2) a swallowtail if $A(0)+B(0) = 0$ and $A'(0)+B'(0) \neq 0$.
(3) a cone singularity if $A(t)+B(t) = 0$ for all t.

The images on the left in Figure 2 are produced with the following commands:

```
X = Asing(@(t)2, @(t)1+0.5*t,@(t)-1+0.5*t);
f = ksurf(X,X, eye(2), eye(2), [0 0.035 60 60],[0 0.035 60 60], 6);
f = ksurf(X,X, eye(2), eye(2), [0 0.035 80 80],[0 0.035 80 80], 7);
```

and those on the right by:

```
X = Asing(@(t)1, @(t)2+t,@(t)-2-t);
f = ksurf(X,X, eye(2), eye(2), [0 0.015 80 80],[0 0.015 80 80], 6);
```

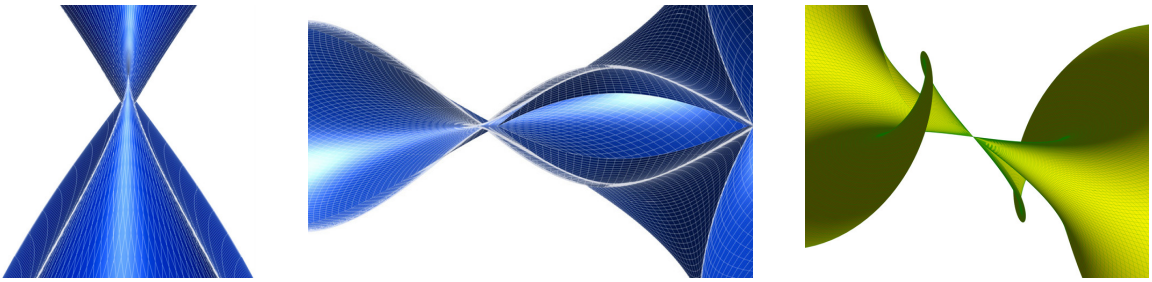

FIGURE 2. Left, center: Swallowtail. Right: Cone

1.1.2. *The unique K-surface corresponding to a generic space curve.* The command

$$X = SGCP(@(s)\kappa(s), @(s)\tau(s))$$

produces the potential for a pseudospherical surface which contains an arc-length parameterized cuspidal edge curve with the prescribed curvature κ and torsion τ, (see [1] Theorem 4.3). It degenerates at points where $\kappa = 0$ or $\tau = \pm 1$. If you have the curvature and torsion of a given curve $\gamma(t)$ which is not arc-length parameterized, then use:

$$X = SGCP(@(t)\kappa(t), @(t)\tau(t), @(t)dsdt(t),$$

where $dsdt(t) = |\gamma'(t)|$, instead.

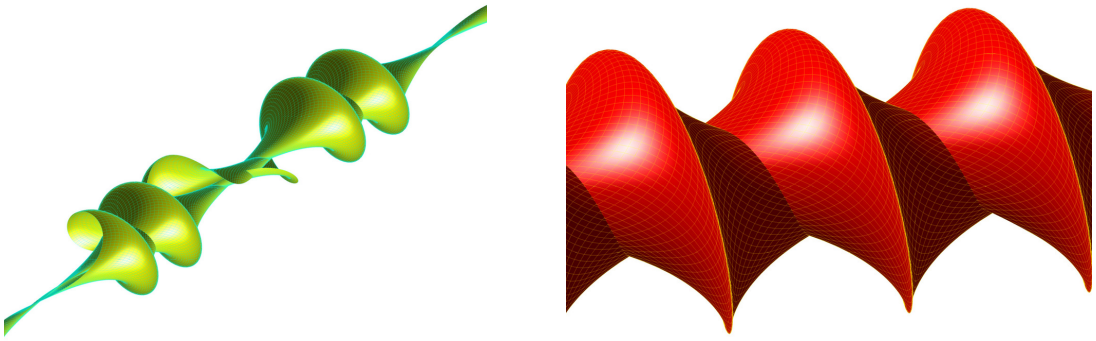

FIGURE 3. Helix surface

The commands:

```
X = SGCP(@(t)1/2, @(t)5) ;
f = ksurf(X,X, eye(2), eye(2), [0 0.025 80 80],[0 0.025 80 80], 10);
f = ksurf(X,X, eye(2), eye(2), [0 0.025 120 120],[0 0.025 120 120], 14);
```

produce the two different sized sub-domains of a helix surface in Figure 3.

The commands:

```
X = SGCP(@(t)t^2, @(t)2)
f = ksurf(X,X, eye(2), eye(2), [0 0.03 120 120],[0 0.03 120 120], 6);
```

produce the surface in Figure 4. Because the curvature $\kappa(t) = t^2$ grows in both directions, the central singular curve curves more and more tightly as $|t| \to \infty$. The surface has more and more cuspidal edges the further you move away from the center, and the whole surface appears to be contained in a bounded set, because of this spiraling effect.

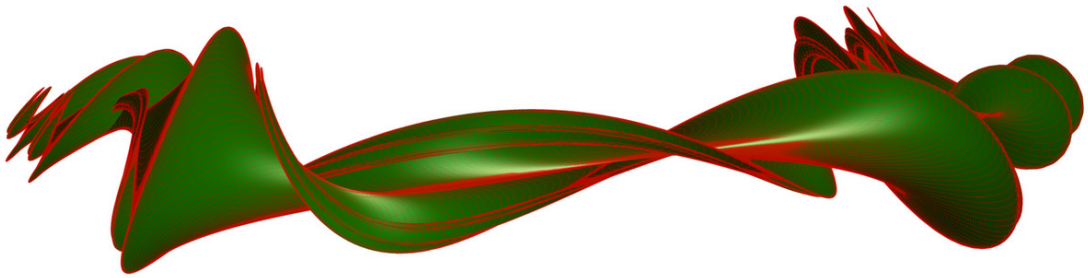

FIGURE 4. $\kappa(t) = t^2$, $\tau(t) = 2$.

1.2. **Characteristic singularities.** Characteristic singularities are special singularities where the singular curve is tangent to a characteristic direction (null direction with respect to the Lorentz structure of the surface). A regular characteristic singular curve is either a straight line segment or a curve with non-vanishing curvature and constant torsion $\tau = \pm 1$. For a curve of this type, there are infinitely many pseudospherical surfaces that have this curve as a characteristic singularity. The function *charSGCP* computes all of these, given the right input. Enter

$$[X,Y] = charSGCP(@(x)\kappa(x), @(y)\alpha(y));$$

where (from [1] Thereom 5.1) either:

(1) $\kappa(x) = 0$ for all x and α is an arbitrary function, or
(2) $\kappa(x) \neq 0$ for all x and α is a function that must satisfy $\alpha(0) = 0$.

Entering

```
[X,Y] = charSGCP(@(t)0, @(t)1);
f = ksurf(Y,X, eye(2), eye(2), [0 0.04 60 60],[0 0.04 60 60], 6);
```

produces a surface with a straight line cuspidal edge (Figure 5).

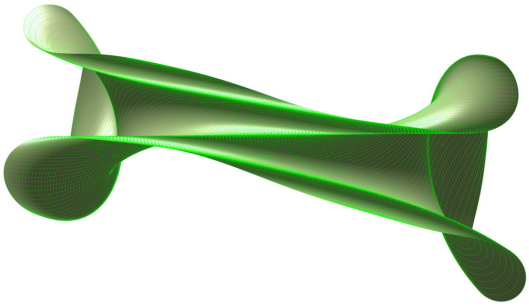

FIGURE 5. $\kappa(t) = t^2$, $\tau(t) = 2$.

2. USING THE FUNCTION *ksurf*

Here is a more detailed description of how *ksurf* works The function ksurf computes the surface corresponding to the potentials Ap and Am. These are loop valued function handles. All loops are entered untwisted and as Laurent polynomials of the form $\sum_{-n}^{n} A_i \lambda^i$. For example the loop $A_0 + A_1\lambda$ should be thought of as $0.\lambda^{-1} + A_0 + A_1\lambda$, and this is entered as a 2×6 matrix $[0\ A_0\ A_1]$. Untwisting a twisted loop is done as follows:

$$\begin{pmatrix} a(\lambda) & b(\lambda) \\ c(\lambda) & d(\lambda) \end{pmatrix} \mapsto \begin{pmatrix} a(\sqrt{\lambda}) & B_{-1}(\sqrt{\lambda}) \\ C_{+1}(\sqrt{\lambda}) & d(\sqrt{\lambda}) \end{pmatrix}, \quad B_{-1}(\lambda) := \lambda^{-1} b(\lambda), \quad C_{+1}(\lambda) := \lambda c(\lambda).$$

For example if Ap is the twisted potential $\begin{pmatrix} 0 & i \\ i & 0 \end{pmatrix} \lambda \, dx$, the loop $\begin{pmatrix} 0 & i \\ i & 0 \end{pmatrix} \lambda$ untwists to

$$\begin{pmatrix} 0 & i \\ \lambda i & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \lambda^{-1} + \begin{pmatrix} 0 & i \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ i & 0 \end{pmatrix} \lambda.$$

The potential would then be entered as

$$Ap = @(t)[0,0,0,i,0,0;\ 0,0,0,0,i,0]$$

Similarly the twisted potential $Am = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \lambda^{-1}$ untwists to $\begin{pmatrix} 0 & \lambda^{-1} \\ -1 & 0 \end{pmatrix}$.

We can use these to compute the Amsler surface, (which contains two straight lines) by entering in Matlab:

```
» Ap=@(t)[0, 0, 0, i, 0, 0;0, 0, 0, 0, i, 0];
» Am=@(t)[0 ,1, 0, 0, 0, 0; 0, 0, -1, 0, 0, 0];
» f=ksurf(Am, Ap, eye(2), eye(2), [0, 0.05, 40, 40], [0, 0.05, 40, 40], 8);
```

Which produces some text output and the image shown in Figure 6.

Of the text output, the important thing is at the bottom

Max error:3.8e-06. Mean error: 1.3e-06

These errors are estimated at each point by checking if the matrix computed is in $\mathfrak{su}(2)$. If the maximum error is less than about 10^{-2}, then the image should be accurate. If the maximum error is greater than 10^{-1}, you can try again with a higher order of polynomial approximation for the loops - see example below.

To use the function ksurf, type:

$$f = ksurf(Am, Ap, Fm0, Fp0, Iy, Ix, looporder).$$

Am and *Ap* are the function handles corresponding to the plus and minus potentials, as described above, $Fm0$ and $Fp0$ are loops which are to be the initial conditions, and can usually be taken to be the 2×2 identity ($eye(2)$ in

```
x-axis:  Left endpoint Error 3.08e-07. Right endpoint Error 3.08e-07
y-axis:  Left endpoint Error 3.08e-07. Right endpoint Error 3.08e-07
Row 1 Max Error 3.8e-06. Errors: 3.8e-06  3.8e-06  3.3e-06  3.1e-06
Row 11 Max Error 2.9e-06. Errors: 2.8e-06  2.8e-06  2.4e-06  2.2e-06
Row 21 Max Error 2.3e-06. Errors: 1.9e-06  1.9e-06  1.6e-06  1.4e-06
Row 31 Max Error 1.3e-06. Errors: 1.1e-06  1.1e-06  7.8e-07  6.4e-07
Row 41 Max Error 2.2e-06. Errors: 9.2e-07  9.3e-07  6.8e-07  5.7e-07
Row 51 Max Error 1.3e-06. Errors: 1.1e-06  1.1e-06  7.8e-07  6.4e-07
Row 61 Max Error 2.3e-06. Errors: 1.9e-06  1.9e-06  1.6e-06  1.4e-06
Row 71 Max Error 2.9e-06. Errors: 2.8e-06  2.8e-06  2.4e-06  2.2e-06
Row 81 Max Error 3.8e-06. Errors: 3.8e-06  3.8e-06  3.3e-06  3.1e-06

Max error:3.8e-06. Mean error: 1.3e-06. Corner Errors:
3.8e-06  3.8e-06
3.8e-06  3.8e-06
```

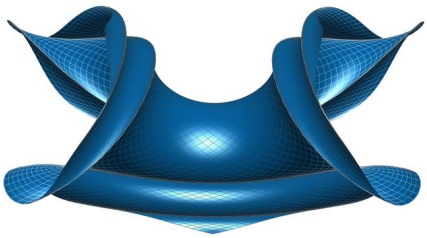

FIGURE 6. Amsler's surface

Matlab), and $Iy = [y_0, ystep, leftpoints, rightpoints]$ describes the interval of integration for Am, that is, Am will be integrated over the interval $[y_0 - ystep * leftpoints, y_0 + ystep * rightpoints]$ with stepsize $ystep$. Ix is similar w.r.t. Ap. Finally the *looporder* is the order of the Laurent polynomial approximation you want to use for the loops.

To get a K-surface as output, loops Ap amd Am have to have the appropriate properties for the potentials for this problem, described in [2], (Definition 5.2).

We can compute one quarter of the Amsler surface above, with the command:

```
f=ksurf(Am, Ap, eye(2), eye(2), [0 0.05 0 40], [0 0.05 0 40], 6);
```

to get the left image in Figure 7.

A somewhat random example:

```
» Ap=@(t)[3*i, 0, 0, 1, i, i*cos(t),0,0,0,0; 0, -3*i, 0,0,-1,-i,i*cos(t),0,0,0];
» Am=@(t)[0,(1+t^2)*(1+i),0,0,0,0; 0,0,(1+t^2)*(-1+i),0,0,0];
» f=ksurf(Am, Ap, eye(2), eye(2), [0 0.03 40 40], [0 0.03 40 40], 8);
```

produces the right image in Figure 7.

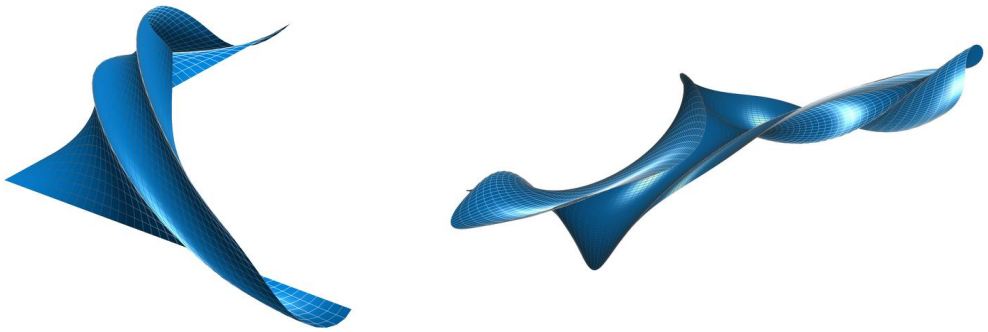

FIGURE 7. Left: Amsler's surface again. Right: Random example.

3. COMPUTING A SURFACE FROM REGULAR GEOMETRIC CAUCHY DATA

Theorem 5.3 in [2] can be used to compute the surface which contains an initial curve $f(t)$ and with prescribed unit normal $n(t)$ along the curve, provided the inner product $\frac{df}{dt} \cdot \frac{dn}{dt}$ is nowhere vanishing. (If this inner product vanishes then uniqueness is lost).

There are two cases that the theorem applies to, the case that $\frac{df}{dt}$ and $\frac{dn}{dt}$ are parallel for all t, and the case that they are linearly independent for all t. The function *kgcppotprinc* will compute the potentials for the first case,

given function handles for the following functions

$$fp = \frac{\mathrm{d}f}{\mathrm{d}t}, \quad fpp = \frac{\mathrm{d}^2 f}{\mathrm{d}t^2},$$
$$np = \frac{\mathrm{d}n}{\mathrm{d}t}, \quad E = \frac{\mathrm{d}f}{\mathrm{d}t} \times n.$$

(A similar function can be written to compute the potentials for the second case).

The parabola $y = x^2$ is parameterised as $f(t) = (t, t^2, 0)$ with derivative $f'(t) = (1, 2t, 0)$ and $f''(t) = (0, 2, 0)$. Using the curves own unit normal $n(t) = \frac{1}{(1+4*t^2)^{3/2}}(2, 4t, 0)$ we can compute the pseudospherical surface that contains this parabola as a principal geodesica curve with the commands:

```
A=kgcppotprinc( @(t)[1;2*t;0], @(t)(1+4*t^2)^(-3/2)*[2;4*t;0], @(t)[0;2;0] , @(t)[0;0;0] );
f=ksurf( @(t)-A(-t), A, eye(2), eye(2), [0 0.05 40 40], [0 0.05 40 40], 5);
```

To get the first image in Figure 8.

A similar process gives a pseudospherical surface containing an ellipse:

```
fp=@(t)[cos(t);-2*sin(t);0];
fpp=@(t)[-sin(t);-2*cos(t);0];
np=@(t)(cos(t)^2+4*sin(t)^2)^(-3/2)*[2*sin(t)^2*cos(t)+2*cos(t)^3; -4*cos(t)^2*sin(t)-4*sin(t)^3; 0];
E=@(t)[0;0;0];
f=ksurf(@(t)-A(-t), A, eye(2), eye(2), [0 pi/50 50 50], [0 pi/50 50 50], 4);
```

This produced the second image in Figure 8, which is not accurate because we used to low an order of approximation for the loops (order 4). The errors were:

Max error:1.4. Mean error: 0.051,

Which indicates we need to raise the loop order significantly. Computing with order 7:

```
f=ksurf( @(t)-A(-t), A, eye(2), eye(2), [0 pi/50 50 50], [0 pi/50 50 50], 7 );
```

We have:

Max error:0.0015. Mean error: 2.4e-05

and the third image in Figure 8. With a maximum error of 10^{-3}, the image is accurate.

FIGURE 8. Left: A pseudospherical surface containing a parabola. Middle: Inaccurate image. Right: A surface containing an ellipse.

These examples are described in [2].

REFERENCES

1. D Brander, *Pseudospherical frontals and their singularities*, arXiv:1502.04876 [math.DG].
2. D Brander and M Svensson, *The geometric Cauchy problem for surfaces with Lorentzian harmonic Gauss maps*, J. Differential Geom. **93** (2013), 37–66.
3. M Toda, *Initial value problems of the sine-gordon equation and geometric solutions*, Ann. Global Anal. Geom. **27** (2005), 257–271.

DEPARTMENT OF MATHEMATICS, MATEMATIKTORVET, BUILDING 303 S, TECHNICAL UNIVERSITY OF DENMARK, DK-2800 KGS. LYNGBY, DENMARK

E-mail address: D.Brander@mat.dtu.dk