Scientific Visualization
Project III

Milian Wolff

February 18, 2012

In our fourth and last project for the the scientific visualization class by Eugene Zhang we mainly investigated second order symmetric tensor fields. Similar to the last exercise we write a JavaView based application that offers an interface to design a tensor field using design elements. Furthermore we extract degenerate points and visualize the separatrices.

The two other tasks in this project deepen our knowledge about vector field topology by investigating the effects of a reverse stereographic projection. Additionally we get acquainted with Conley indizes.

The source code of my exercise solutions can be found online under

`https://github.com/milianw/scivi`

Contents

1 Reverse Stereographic Projection

Given a vector field $\vec{f}(x, y)$ in R^2 we project it onto the sphere S^2. All three given vector fields have only a single singularity at $(x, y) = (0, 0)$, which corresponds to the South Pole on S^2. To have a well-defined behavior of the field for $x, y \to \pm\infty$, we modify the magnitude of the field without creating new singularities in R^2 by multiplying with a scaling factor. This is just like we did in the last project for the vector field design elements. Here we can use for example

$$\sigma = \exp(-x^2 - y^2). \tag{1}$$

Through this modification we can safely apply the reverse stereographic projection and will now always encounter a properly defined singularity at the North Pole.

Since we know the euler characteristic of a sphere to be $\chi(S) = 2$, and that this equals the sum of Poincarè-indizes p_i, i.e. $\chi(S) = 2 = \sum p_i = p_{\mathrm{NP}} + p_{\mathrm{SP}}$, we find

$$p_{\mathrm{NP}} = 2 - p_{\mathrm{SP}}. \tag{2}$$

As such we can compute the Poincarè-index p_{NP} at the North Pole directly from the index p_{SP} of the South Pole.

Note: To visualize the fields at the poles I wrote a small application, `Ex4_1`, which shows the field-direction. The higher-order design elements, required to visualize the singularities at the North Pole can be computed using eq. 10 from [1], p. 106, with $a = 1$, $b = 0$, $c = 0$, $d = 1$, and $N = 2 \cdot p_{\mathrm{NP}}$.

1.1 Saddle

The first vector field we investigate is that of a saddle:

$$\vec{f} = \sigma \begin{pmatrix} x \\ -y \end{pmatrix} \tag{3}$$

A saddle has an index $p = -1$. As such there will be a quadrupole at the North Pole with an index of $p = 3$. See also fig. 1 for images of the field at both poles.

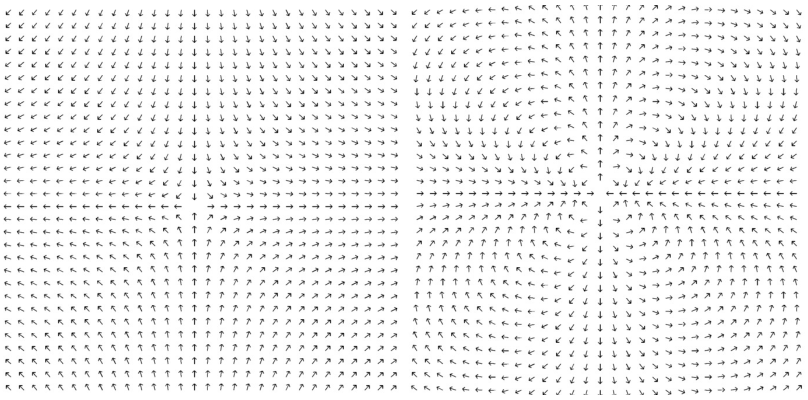

(a) Saddle at South Pole (b) Quadrupole at North Pole

Figure 1: vector field from eq. 3

1.2 Monkey-Saddle

The second vector field we investigate is that of a monkey-saddle:

$$\vec{f} = \sigma \begin{pmatrix} x^2 - y^2 \\ -2xy \end{pmatrix} \tag{4}$$

A monkey-saddle has an index $p = -2$. As such there will be a hexapole at the North Pole with an index of $p = 4$. See also fig. 2 for images of the field at both poles.

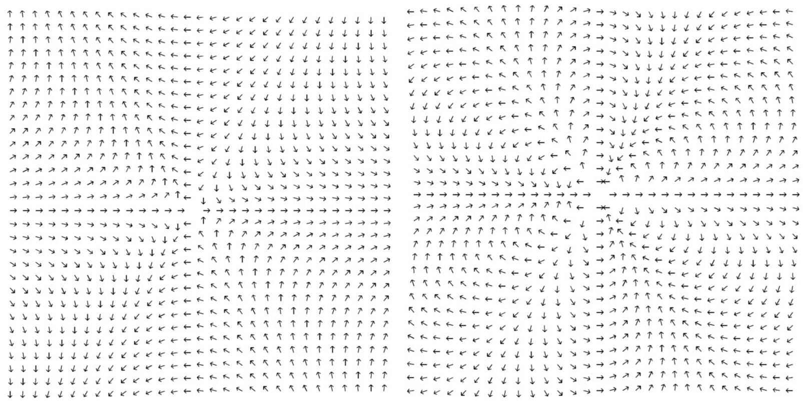

(a) Monkey-Saddle at South Pole (b) Hexapole at North Pole

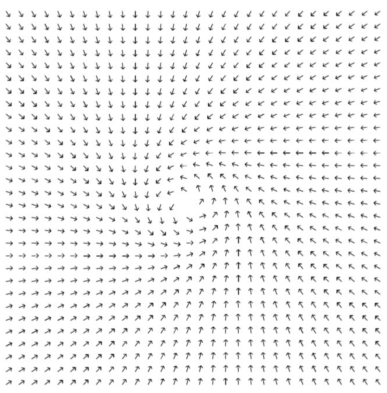
Figure 2: vector field from eq. 4

1.3 Attractive CCW Rotation

The third vector field represents an attractive clockwise rotation:

$$\vec{f} = \sigma \left(\begin{array}{c} x(1-\sqrt{x^2+y^2}) - y \\ x + y(1-\sqrt{x^2+y^2} \end{array} \right) \tag{5}$$

Such a singularity has an index $p = 1$ and hence the singularity at the Nort Pole will have the same index. Due to flow and symmetry conservation the singularity must be a repulsive clockwise rotation. See also 3.


(a) CCW-Rotation at South Pole

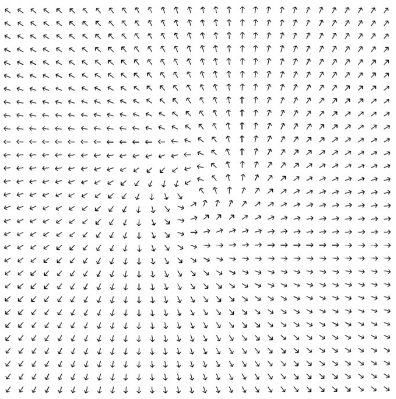

(b) CW-Rotation at North Pole

Figure 3: vector field from eq. 5

2 Conley Indizes

0, 0, 0

Any vector field without a singularity will do the trick, see e.g. fig. 4.

1, 0, 0

A sink or a sink-like vortex has a conley index of $(1, 0, 0)$, see fig. 5.

1, 1, 0

An attractive periodic orbit has an index of $(1, 1, 0)$ and might induce some curl, see fig. 6.

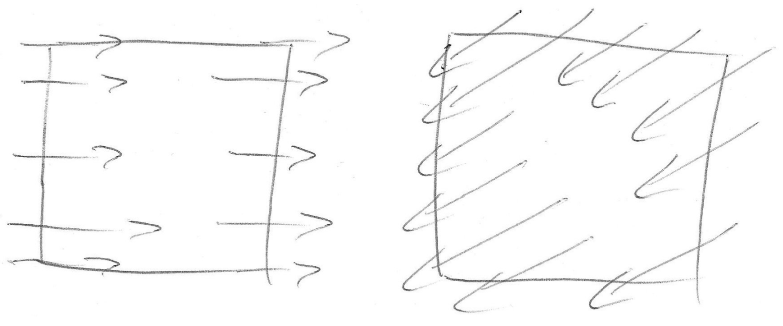


Figure 4: vector fields for Conley index $\beta_0 = 0$, $\beta_1 = 0$ and $\beta_2 = 0$

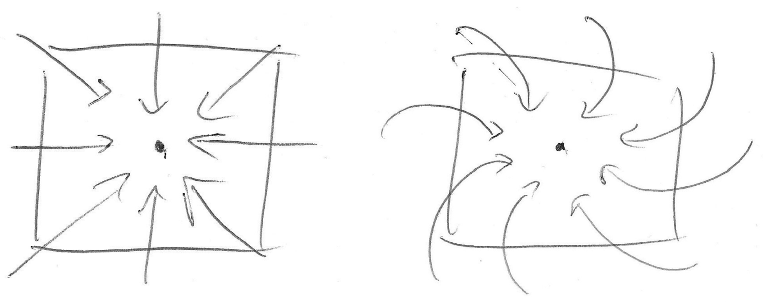


Figure 5: vector fields for Conley index $\beta_0 = 1$, $\beta_1 = 0$ and $\beta_2 = 0$

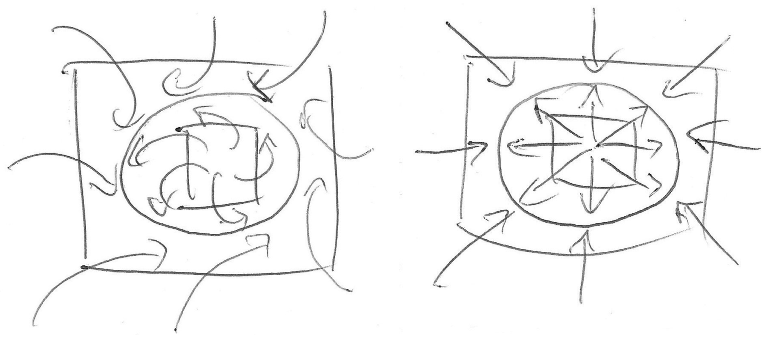


Figure 6: vector fields for Conley index $\beta_0 = 1$, $\beta_1 = 1$ and $\beta_2 = 0$

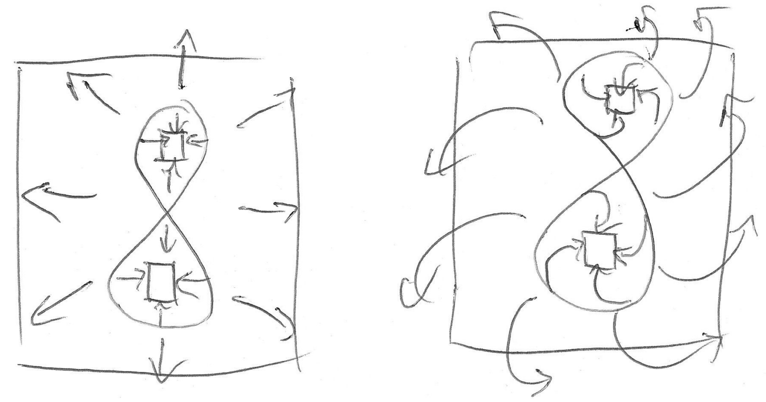


Figure 7: vector fields for Conley index $\beta_0 = 0$, $\beta_1 = 2$ and $\beta_2 = 1$

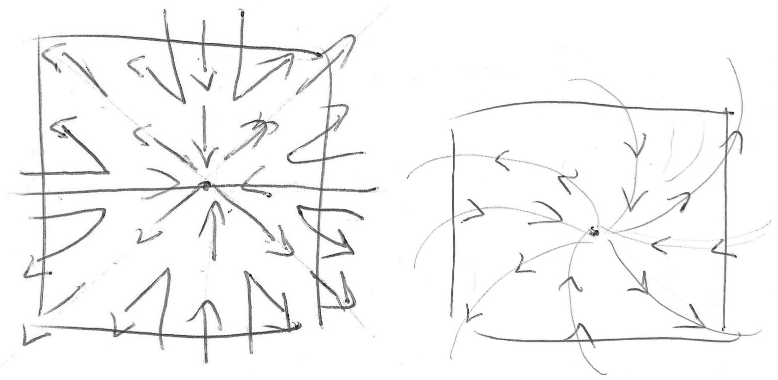


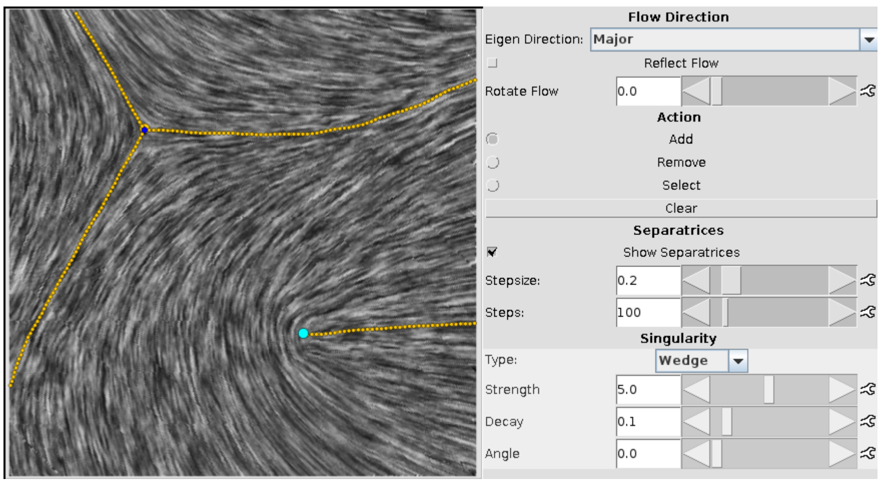
Figure 8: vector fields for Conley index $\beta_0 = 0$, $\beta_1 = 3$ and $\beta_2 = 0$

0, 2, 1

I assume we can add two repulsive periodic orbits such that they create a field whose combined conley-index is $(0, 2, 1)$. See also fig. 7.

0, 3, 0

Again, we should be able to superimpose three saddles to create an octupole with a conley index of $(0, 3, 0)$. See fig. 8.


Flow Direction

Eigen Direction: Major

☐ Reflect Flow

Rotate Flow 0.0

Action

[✓ Add] Remove Select

Clear

Separatrices

☑ Show Separatrices

Stepsize: 0.2

Steps: 100

Singularity

Type: Wedge

Strength 5.0

Decay 0.1

Angle 0.0

Figure 9: Tensor Field Design Application

3 Tensor Field Design

Our programming task consisted this time of adapting our previous vector field design application to visualize symmetric second order tensor fields. We followed the algorithms and ideas outlined by Zhang, Hays and Turk in [1].

A screenshot of my implementation can be found in fig. 9.

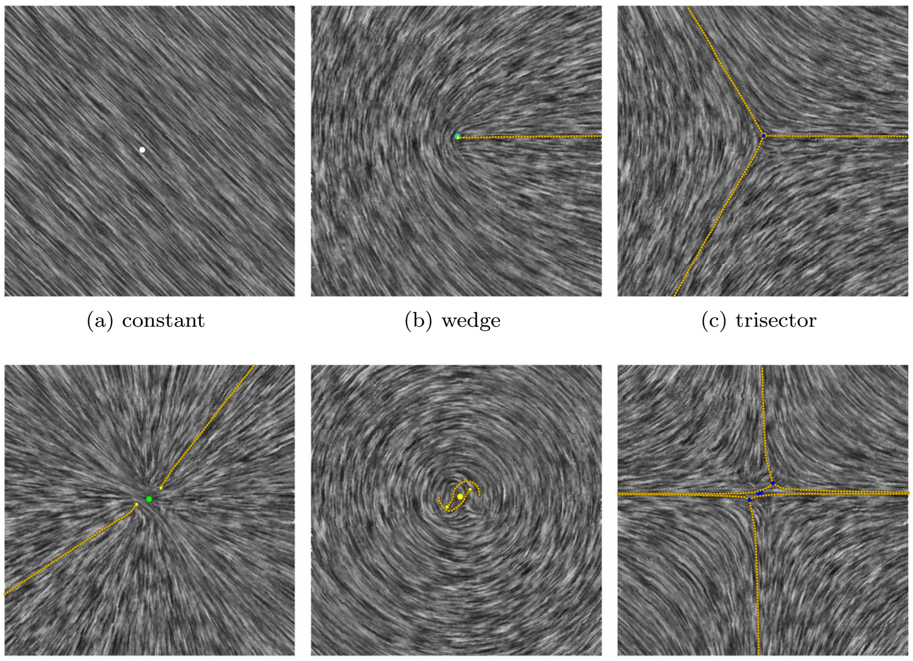
3.1 Design Elements

The application allows the user to interactively create a tensor field. You choose one of six design elements (see fig. 10 and [1] p. 99f) and then click into the view space to place it. The elements can be moved by drag and drop or be removed again as well.

The total second order tensor field is then defined through

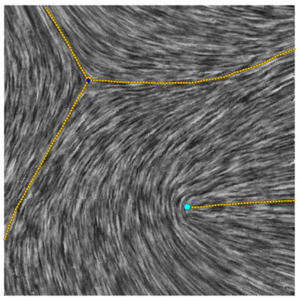
$$\mathbf{T}(\vec{r}) = \sum_{i}^{N} \mathbf{T}_i(\vec{r}). \tag{6}$$

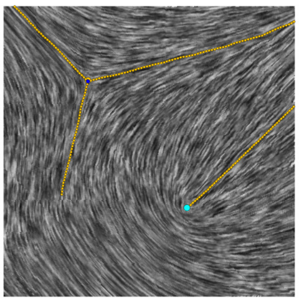
The class `TensorField` and the inheritors classes of `TensorTerm` provide the code implementation of this.

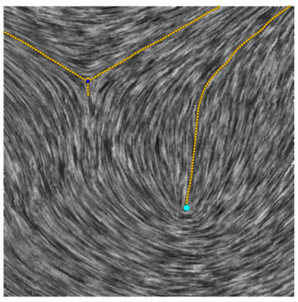

(a) constant (b) wedge (c) trisector

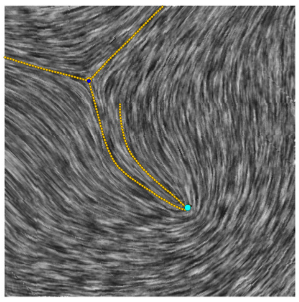
(d) node (e) center (f) saddle

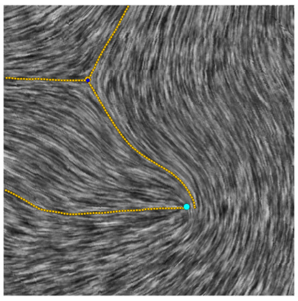
Figure 10: tensor field design elements


(a) original


(b) 45° rotation


(c) 90° rotation


(d) 135° rotation


(e) 180° rotation

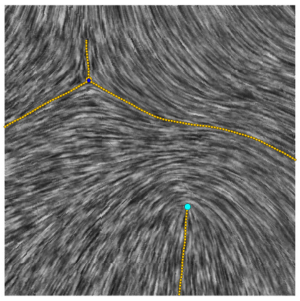

(f) 270° rotation

Figure 11: tensor field rotation

3.2 Rotation and Reflection

To rotate the tensor field by an angle θ, we use the reflection matrix

$$\mathbf{R} = \begin{pmatrix} \cos(\theta/2) & -\sin(\theta/2) \\ \sin(\theta/2) & \cos(\theta/2) \end{pmatrix} \tag{7}$$

The rotated tensor field is then obtained from eq. 6 using

$$\mathbf{T}'(\vec{r}) = \mathbf{R}\mathbf{T}(\vec{r})\mathbf{R}^T. \tag{8}$$

To additionally reflect the tensor field along the axis defined by θ, we can use a reflection matrix $\mathbf{R}'$ instead of $\mathbf{R}$:

$$\mathbf{R}' = \begin{pmatrix} \cos(\theta/2) & \sin(\theta/2) \\ \sin(\theta/2) & -\cos(\theta/2) \end{pmatrix} \tag{9}$$

Take a look at fig. 11 for tensor field rotation and at fig. 12 for tensor field reflection.

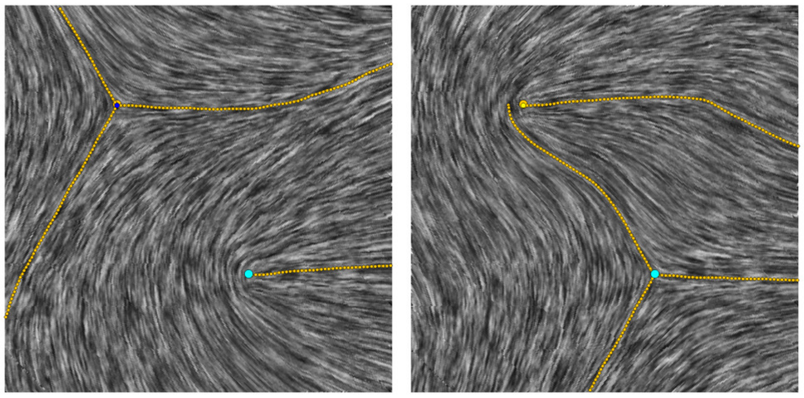

(a) original (b) reflected

Figure 12: tensor field reflection

3.3 Discretization and Interpolation

The continous tensor field in eq. 6 must be discretized for numerical analysis and visualization. First, we use a triangulated domain and compute the tensor field at each vertex. Then we interpolate the tensor field in each triangle from the three fields $\mathbf{T}_{1,2,3}$ at the vertices $\vec{v}_i = (x_i, y_i)$, $i = 1, 2, 3$, by solving the two equation systems

$$\left(\begin{array}{ccc|cc} a/c & b/d & c/f & \alpha & \beta \\ x_1 & y_1 & 1 & T_{1_{0,0}} & T_{1_{1,0}} \\ x_2 & y_2 & 1 & T_{2_{0,0}} & T_{2_{1,0}} \\ x_3 & y_3 & 1 & T_{3_{0,0}} & T_{3_{1,0}} \end{array}\right) \tag{10}$$

yielding the symmetric second order interpolated tensor field

$$\mathbf{T}_I(\vec{r}) = \begin{pmatrix} ax + by + e & cx + dy + f \\ cx + dy + f & -ax - by - e \end{pmatrix}. \tag{11}$$

Note that this interpolation is of course only valid inside the triangle spanned by the vertices $\vec{v}_{1,2,3}$.

The class `InterpolatedTensorField` implements the interpolation scheme outlined above.

3.4 Eigenvector Field Visualization

To visualize the tensor field we use the image based approach outlined in [1] p. 98f, which allows us to reuse the LIC code from JavaView. For the major eigenvector field, we can directly follow the equations outlined there. For the minor eigenvector field though, we have to make small adjustments: The minor eigenvector $\vec{E}_2$ is orthogonal to $\vec{E}_1$, i.e.

$$\vec{E}_2 = \begin{pmatrix} -\sin\theta \\ \cos\theta \end{pmatrix}. \tag{12}$$

We create $\vec{V}_x$ such that the x-component is positive:

$$\vec{V}_x = \begin{cases} \vec{E}_2 & \text{if } -\sin\theta > 0 \\ -\vec{E}_2 & \text{otherwise} \end{cases} \tag{13}$$

Analogously we create $\vec{V}_y$ such that its y-component is positive. The weighting factors $W_{x,y}$ use the x- and y-component of $\vec{E}_2$. Hence they are

$$W_x = \sin^2\theta \quad \text{, and} \tag{14}$$

$$W_y = 1 - W_x = \cos^2\theta. \tag{15}$$

You can see an example of the above code in fig. 13. If you are interested in the code that implements the calculations above, take a look at `Ex4_3.updateLICImage` and `Ex4_3.computeWeight`.

3.5 Degenerate Points

To find the degenerate points of a symmetric second order tensor field we iterate over all interpolated tensor fields (see eq. 11) in our domain and solve the equation system

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \vec{r}_0 = \begin{pmatrix} -e \\ -f \end{pmatrix}. \tag{16}$$

If $\vec{r}_0$ lies in the triangle for which the interpolation is valid, then we found a degenerate point. We can furthermore classify it by calculating the determinant of the coefficient matrix in the equation above. The degenerate point is a wedge if the determinant is positive, a trisector if it is negative and a higher-order degenerate point if the determinant is zero.

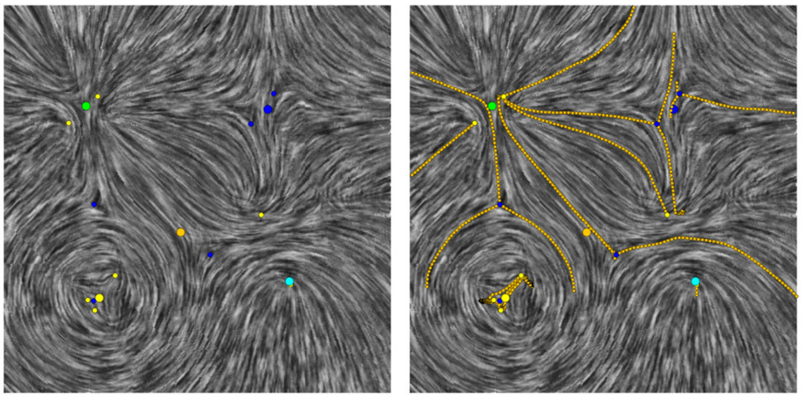
You can see the results of this algorithm in the figures of this report, e.g. fig. 13. Wedges are represented by small yellow dots, while trisectors are blue. Higher-order terms are usually not found, but if so they would be colored white.

`InterpolatedTensorField.findDegeneratePoints` implements the above.

3.6 Separatrices

Another important feature of the tensor topology beside the degenerate points are the separatrices. Following the scheme that was shown to us in-class we can extract the directions of the separatrices at a wedge or trisector, by finding the roots of the cubic polynomial

$$du^3 + (c + 2b)u^2 + (2a - d)u - c = 0, \qquad \text{with } u = tan\theta. \tag{17}$$


(a) major

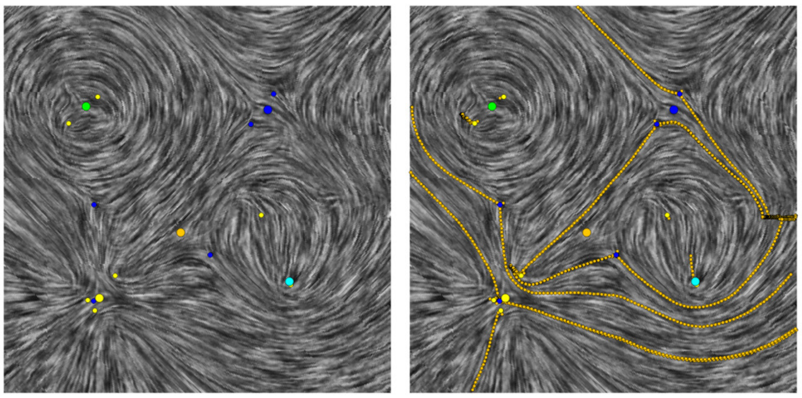

(b) minor

Figure 13: minor and major eigenvector fields

The coefficients a, b, c and d are again those from eq. 11. Solving the equation yields up to three real-valued roots u. It has to be noted that the algorithm applied is not very numerically stable and hence imaginary parts below a threshold of $1E-6$ are ignored. For each value of u we can then find two values of θ. Again it can be observed that the *atan* calculation is very unreliably for expected values near 90° or 270°.

To figure out whether a given θ points into a minor or major direction, we compare the eigen values, i.e.

$$\lambda_1 = \mathbf{T}_I(\vec{r}_0 + \vec{e}_\theta)\vec{e}_\theta \cdot \vec{e}_\theta \tag{18}$$
$$\lambda_2 = \mathbf{T}_I(\vec{r}_0 + \vec{e}_\theta)\vec{e}_{\theta+\pi/2} \cdot \vec{e}_{\theta+\pi/2}, \tag{19}$$

where $\vec{e}_\phi$ is a unit vector in direction of an angle ϕ:

$$\vec{e}_\phi = \begin{pmatrix} \cos\phi \\ \sin\phi \end{pmatrix} \tag{20}$$

If now $\lambda_1 > \lambda_2$, then θ points into the direction of a major separatrix. Otherwise it points into a minor direction. Based on this we can start a classical Runge-Kutta line tracer in the selected direction, using the eigenvector field direction as seed values for the integration scheme.

The above was already applied to find the separatrices (the orange lines) in the previously shown images, see e.g. fig. 13. The application allows the user to define the numer of steps and the stepsize. The tracer stops near degenerate points, i.e. when the distance to a degenerate point is less than one tenth of the stepsize.

The code that visualizes the separatrices of wedges and trisectors is implemented in `Ex4_3.findSeparatrices`, `TensorFieldFunctor` and `ClassicalRungeKuttaTracer`.

References

[1] Eugene Zhang, James Hays, and Greg Turk. Interactive Tensor Field Design and Visualization on Surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 13(1):94–107, January 2007.