Implementation Issues in Vector Field Topology

Al Globus[1]
Computer Sciences Corporation
Applied Research Branch
NAS Systems Division
NASA Ames Research Center

IEEE Visualization ‘93
26 October 1993

1. This work is supported through NASA contract NAS2-12961

Goal

Present a set of techniques and algorithms sufficient to implement vector field topology visualization of 2D and 3D computational results.

Motivation

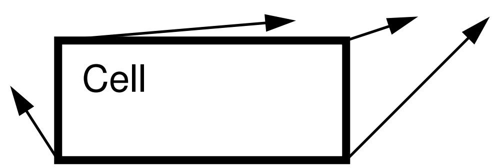
There is a lack of vector field topology visualization software available to users.

Outline

- Observations and Assumptions
- Stationary Points
    - Finding
    - Classifying by eigenvalues
    - Finding and displaying eigenvectors and eigenplanes
    - Finding stable and unstable manifolds
- Important secondary issues
    - Stationary curves, surfaces, and volumes
    - Computational space calculation
- Conclusions

Observations and Assumptions

- Graphics issues are straightforward - focus on scientific data manipulations.
- Near stationary points accuracy is crucial because high gradient and curvature is common.
- Noisy data and numerical inaccuracies require fuzz factors.
- Computational space computation simplifies coding.
- Domain divided into cells with known vectors at nodes. Vector field between nodes determined by interpolation.

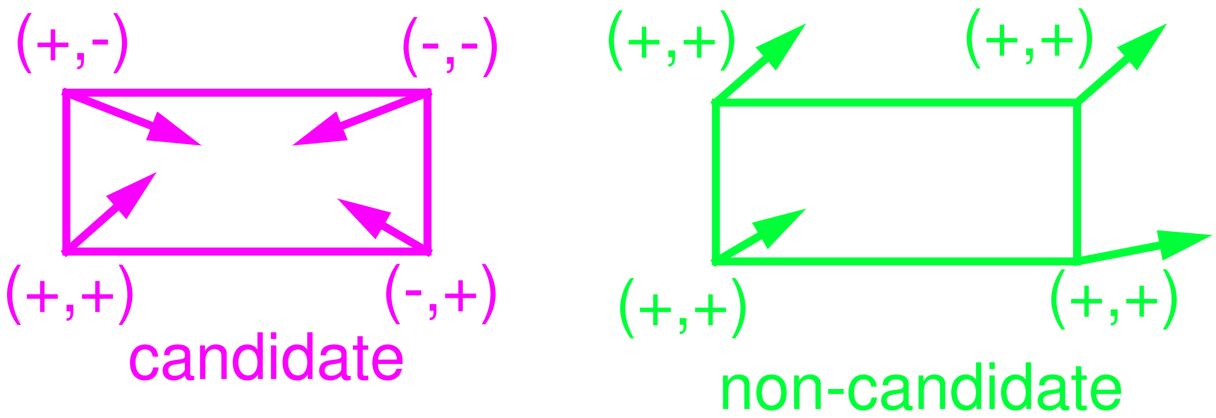


Finding Stationary Points

- To find cells that may contain stationary points (candidates) use a fast, simple necessary test.
- In candidate cells, solve simultaneous non-linear interpolation equations: one equation for each vector component.
    - 2D cells:
        - analytic solution
    - 3D cells:
        - Recursively bisect the cell
        - Make candidate test
        - Polish location with Newton-Raphson iteration

Candidate Cells

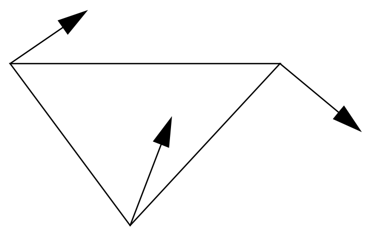
- Necessary, but not sufficient, condition: examine each component of the vector at each cell node. There must be a sign change in every component for a stationary point to exist in the cell.



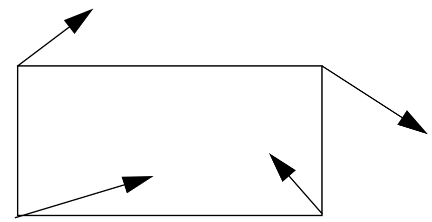
- Note: assumes monotonic interpolation.

Candidate Test Problem

- Cell with a collapsed edge (two nodes at the same physical space location) can give false positive candidate test in computational space.



non-candidate cell with collapsed edge

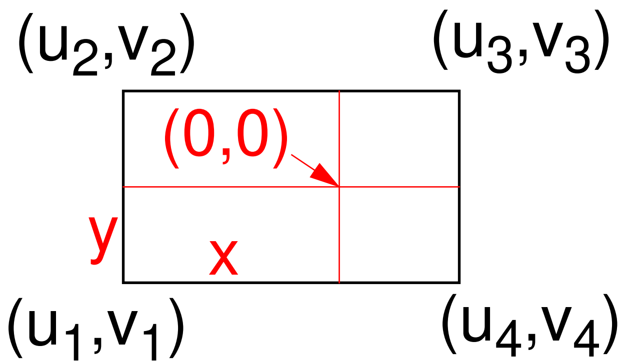


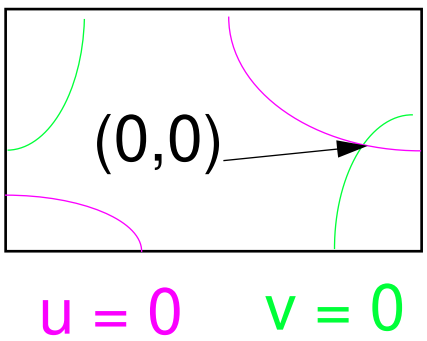
same cell in computational space - false candidate

- Apply test in physical space AND computational space.

2D Cells

- Solve simultaneous bilinear equations ($\bar{x} = 1 - x$, $\bar{y} = 1 - y$):





$$f(x, y) = u_1\bar{x}\bar{y} + u_2\bar{x}y + u_3xy + u_4x\bar{y} = 0 \quad (1)$$

$$g(x, y) = v_1\bar{x}\bar{y} + v_2\bar{x}y + v_3xy + v_4x\bar{y} = 0 \quad (2)$$

Analytic Solution

- Solve (1) for y in terms of x.

$$y = \frac{u_1 + (-u_1 + u_4)\,x}{u_1 - u_2 + (-u_1 + u_2 - u_3 + u_4)\,x} = \frac{A + Bx}{C + Dx}$$

- Use this result to substitute for y in (2).

$$(-v_1 + v_4)\,x + (-v_1 + v_2)\,y + (v_1 - v_2 + v_3 - v_4)\,xy = 0$$

$$E + Fx + Gy + Hxy = 0$$

$$(D + BH)\,x^2 + (D + B + AH)\,x + (EC + GA) = 0$$

- Solve this quadratic equation for x. See references for solution techniques.
- Substitute x into the solution of (1) to get y.

Analytic Solution Problems

- x,y may be outside of the cell
- if u = v there are infinite solutions
- if u is almost v the solution can be ill-behaved as the solution curves can lie arbitrarily close together

3D Cells

- Must search for solution to simultaneous trilinear equations. Considering just one component:

$$v_{000}\bar{x}\bar{y}\bar{z} + v_{001}\bar{x}\bar{y}z + v_{010}\bar{x}y\bar{z} + v_{011}\bar{x}yz + v_{100}x\bar{y}\bar{z} + v_{101}x\bar{y}z + v_{110}xy\bar{z} + v_{111}xyz = 0$$

v_{ijk} = vector component in the i,j,k location of the unit cell

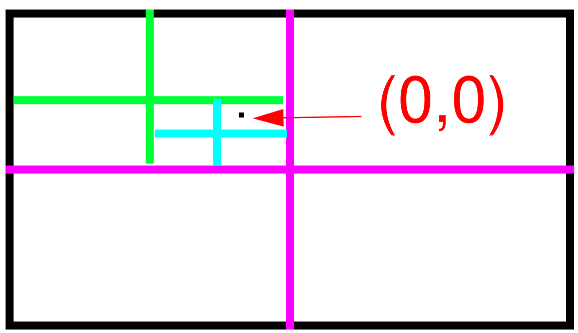
$$\bar{a} = (1 - a)$$

- Like bilinear interpolation, trilinear interpolation is non-linear. Solving simultaneous non-linear equations is difficult.
- Two iterative solution techniques often used are bisection and Newton-Raphson.

3D Cells (cont.)

- Bisection with candidate test is slow to converge, but sure to succeed if a solution exists with the cell.

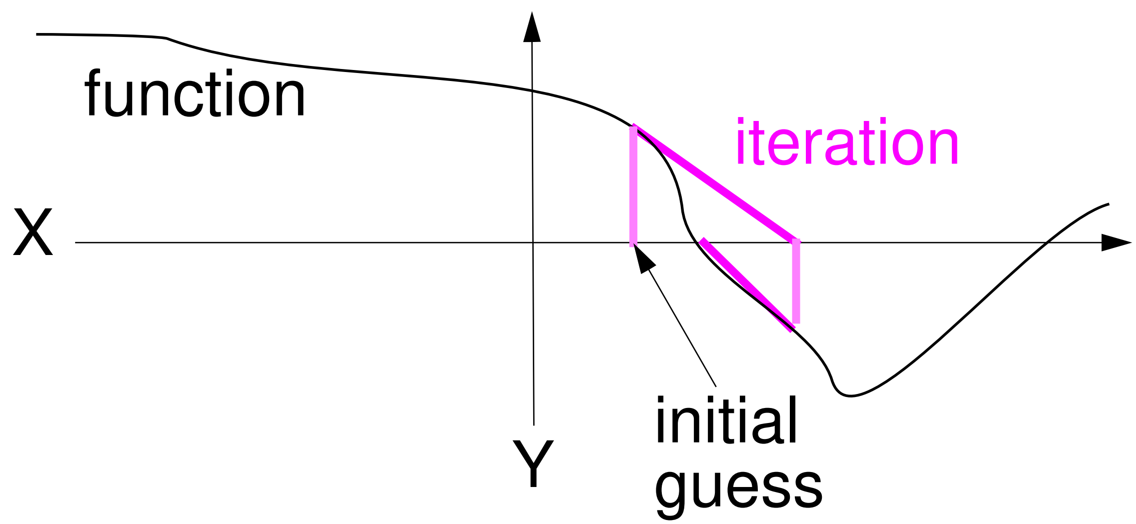
In 2D:



- Newton-Raphson (described below) can converge quickly, but requires good initial point to converge at all.
- Solution: bisect first, polish with Newton-Raphson.

Newton-Raphson

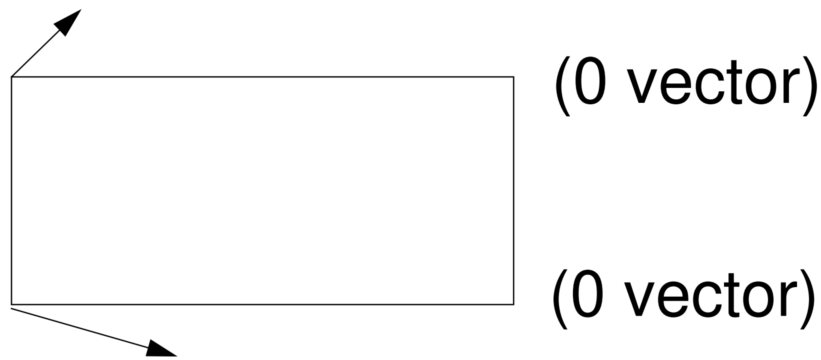
- Make a good initial guess ($\vec{x}_0$).
- Iteratively solve $\vec{x}_{n+1} = \vec{x}_n - J^{-1}\vec{v}$ until close enough. J is the jacobian at x_n.
- In one dimension:



3D Cells: Problems

- Stationary curves and surfaces
  - Check for stationary cell edges

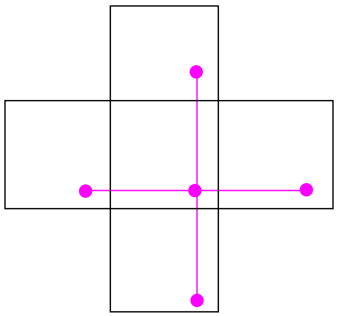
In 2D:



  - Enforce a maximum number of solutions
- Multiple discrete solutions are possible, but Newton-Raphson only finds one. Initial bisection steps help.
- Newton-Raphson may not converge. Increasing the number of bisections can help.

Classifying Stationary Points

- Use differences to calculate matrix.
    - Differencing within cell is inaccurate near corners because this approaches a one sided difference. Two equivalent solutions:
        - Difference cell nodes and interpolate, or
        - Interpolate points within neighbors and difference.



- Find eigenvalues.

Classifying Stationary Points (cont.)

- Remember that det(J - λI) = 0 (λ = eigenvalue)

- Solve resulting cubic equation (3D)

$$-\lambda^3 + Tr(J)\lambda^2 + \sum_i Cof_{ii}(J)\lambda + det(J) = 0$$

- See numerical methods references for solution techniques.

Selecting Sets of Stationary Points

- It is frequently useful to select a subset of the stationary points for various operations. Try selection by:
    - User pointing
    - Type, e.g., saddles
    - Location, e.g., within a bounding box
    - Field criteria, e.g., where pressure > 1.5
    - Set operations: and, or, xor, not
- Use linked lists for flexibility

Eigenvectors (3D)

- Solve $A\vec{x} = \lambda\vec{x}$ for each eigenvector
- Now $(A - \lambda I)\,\vec{x} = 0$ is underdetermined
- Set one component of $\vec{x}$ to 1

$$\begin{bmatrix} (a_{11}-\lambda) & a_{12} & a_{13} \\ a_{21} & (a_{22}-\lambda) & a_{23} \\ a_{31} & a_{32} & (a_{33}-\lambda) \end{bmatrix} \begin{bmatrix} 1 \\ x_2 \\ x_3 \end{bmatrix} = 0$$

$$\begin{bmatrix} (a_{22}-\lambda) & a_{23} \\ a_{32} & (a_{33}-\lambda) \end{bmatrix} \begin{bmatrix} x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} a_{21} \\ a_{31} \end{bmatrix}$$

or $\quad M\vec{x}' = \vec{b}$

Eigenvectors (3D) (cont.)

- Now $M\vec{x}' = \vec{b}$ can be solved if M is non-singular. Since there is a choice of M depending on the component of $\vec{x}$ set to 1, this can usually be accomplished.

Eigenvector Display

- Display of real eigenvectors is straightforward.
  - Scale normalized eigenvectors by a root of the eigenvalue to show relative eigenvalue sizes.
- Displaying 3D complex eigenvectors is less obvious.
  - Find real eigenplane spanned by the eigenvectors. Any two REAL eigenvectors created by linear combinations of the complex eigenvectors define this plane:
    - Now the eigenvectors are a conjugate pair, let eigenvectors $\vec{e}_1 = \vec{u} + i\vec{u}'$ and $\vec{e}_2 = \vec{u} - i\vec{u}'$
    - Thus, $im(\vec{e}_1 + \vec{e}_2) = 0$ and $im(i\vec{e}_1 - i\vec{e}_2) = 0$; i.e., they are real vectors.

Trajectory Integration

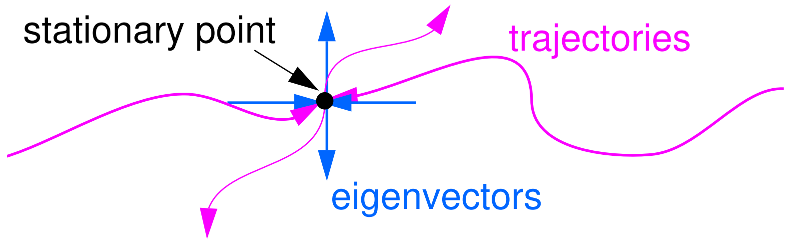
- High curvature and gradients near stationary points requires accurate integration techniques, e.g.:
  - 4th order Runge-Kutta (Δt = time step, x_n = point)
    - $\vec{k}_1 = \Delta t \vec{v}(\vec{x}_n)$
    - $\vec{k}_2 = \Delta t \vec{v}\left(\vec{x}_n + \frac{\vec{k}_1}{2}\right)$
    - $\vec{k}_3 = \Delta t \vec{v}\left(\vec{x}_n + \frac{\vec{k}_2}{2}\right)$
    - $\vec{k}_4 = \Delta t \vec{v}(\vec{x}_n + \vec{k}_3)$
    - $\vec{x}_{n+1} = \vec{x}_n + \frac{\vec{k}_1}{6} + \frac{\vec{k}_2}{3} + \frac{\vec{k}_3}{3} + \frac{\vec{k}_4}{6} + O(\Delta t^5)$

Trajectory Integration (cont.)

- Adaptive time step
  - Compare one full step vs. two half steps
  - If close, increase ∇t and continue; otherwise, decrease ∇t and try again. See Numerical Recipes in C.
- Problems:
  - ∇t can get too large or small
    - bracket it
  - Integration can stop making forward progress
    - Stop if no motion and no change in ∇t

Trajectories

- Trajectories started very near a stationary point on the eigendirections are important.

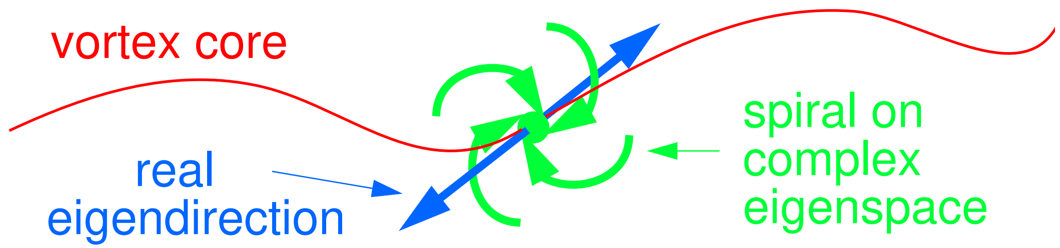


- In 2D the trajectories on saddle eigendirections divide the domain into basins of attraction
    - These trajectories end in nodes, Helman has constructed stationary point connectivity graphs on this basis
- In 3D vortex cores can be found

Vortex Core Detection

- We define a vortex core as the streamline within a vortex with minimum curvature.

- At a saddle focus, trajectories spiral in a plane, but not along the real eigendirection. Thus, the trajectory starting on the real eigendirection close to the stationary point will be the vortex core.



Vortex Core Detection Problems

- Numerical trajectories may not stay on the core
- Disturbances such as shocks make cause core discontinuities.
- Partial solution: in a vortex, two eigenvalues of J at each step must be complex. Stop if they aren't.

Stationary Curves, Surfaces, and Volumes

- Detection
  - Stationary points on adjacent nodes.
  - (Very) many critical points in a cell.
- Stationary Surfaces
  - Construct the derivative of the vector field on a stationary surface.
  - Remove the component normal to the surface.
  - Find 2D topology in this field.

Special Stationary Grid Surface Trick

- Set the surface to the vector field one cell away in the normal direction.
- Zero the vector component normal to the surface.

in 2D:

- Can now use 3D stationary point location code.
- Off surface eigenvalue and eigenvector can also be found.

Computation Space Calculation

- Simplifies code and improves performance.
- Regular cells have simpler interpolation functions.
- Trajectory integration does not require conversion from physical to computational space.
- Finding sufficiently accurate cell face normals for stationary grid surface trick is difficult in physical space.

Conclusions

- Topology programming is tricky.
- Most of the techniques are well known.
- Consult Numerical Recipes in C (Press, et. al.), Numerical Methods (Dahlquist and Bjoerck), or other reputable numerical analysis book frequently.
- Future work
    - Performance.
    - User interface.
    - Visualization systems other than FAST.
    - Stable and unstable surfaces.
    - Unstructured (e.g., finite element) grids.