Force Inference

In[2]:=

```
ClearAll[segmentImage];
segmentImage[binarizedMask_?ImageQ, border : True | False] :=
  Which[! border,
   Print["border not present, segmenting with morphological components"];
   ArrayComponents@DeleteBorderComponents@
     MorphologicalComponents[ColorNegate@binarizedMask, CornerNeighbors -> False],
   border,
   Print["border exists, segmenting with watershed components"];
   WatershedComponents[binarizedMask, CornerNeighbors → False]
  ];
```

In[4]:=

```
bresenhamLine[p0_, p1_] := Module[{dx, dy, sx, sy, err, newp},
   {dx, dy} = Abs[p1 - p0];
   {sx, sy} = Sign[p1 - p0];
   err = dx - dy;
   newp[{x_, y_}] := With[{e2 = 2 err}, {If[e2 > -dy, err -= dy;
       x + sx, x], If[e2 < dx, err += dx;
       y + sy, y]}];
   NestWhileList[newp, p0, # =!= p1 &, 1]
  ];
```

In[5]:=

```
closeImage[image_] := Module[{morphImage, morphImagePos, posConv, ptsOrdered},
   morphImage = MorphologicalTransform[image, "SkeletonEndPoints"];
   morphImagePos = PixelValuePositions[morphImage, 1];
   posConv = With[{mean = Mean@N@morphImagePos},
     ptsOrdered = Append[#, First@#] &[SortBy[morphImagePos, ArcTan @@ (mean - #) &]];
     DeleteDuplicates@Flatten[bresenhamLine @@@ Partition[ptsOrdered, 2, 1, 1], 1]
    ];
   ReplacePixelValue[image, posConv → 1]
  ];
```

In[6]:=
```
Options[associateVertices] = {"stringentCheck" -> True};
associateVertices[img_, segt_, OptionsPattern[]] :=
  With[{dim = Reverse@ImageDimensions@img, stringentQ = OptionValue["stringentCheck"]},
   Module[{pts, members, vertices, nearest, vertexset, likelymergers, imagegraph,
     imggraphweight, imggraphpts, vertexpairs, posVertexMergers, meanVertices, Fn},
    pts = ImageValuePositions[MorphologicalTransform[img,
       {"Fill", "SkeletonBranchPoints"}], 1];
    (* finding branch points *)members = ParallelMap[Block[{elems},
        elems = Dilation[ReplaceImageValue[ConstantImage[0, Reverse@dim], # -> 1], 1];
        DeleteCases[Union@Flatten@ImageData[elems * Image[segt]], 0 | 0.]] &, pts];

    vertices = Cases[Thread[Round@members -> pts],
      HoldPattern[pattern : {__} /; Length@pattern >= 2 -> _]];
    (* finding vertices with 2 or more neighbouring cells *)
    nearest = Nearest[Reverse[vertices, 2]];
    (* nearest func for candidate vertices *)Fn = GroupBy[MapAt[Sort,
        (# -> nearest[#, {All, 3}] & /@ Values[vertices]), {All, 2}], Last -> First, #] &;
    Which[Not@stringentQ,
     (* merge if candidate vertices are 2 manhattan blocks away. Not a stringent
      check for merging *)KeyMap[Union@*Flatten]@Fn[List@*N@*Mean] // Normal,
     stringentQ,
     (* a better check is to see the
      pixels separating the vertices are less than 3 blocks *)
     vertexset = Fn[Identity];
     (* candidates for merging*)likelymergers = Cases[Normal[vertexset],
       PatternSequence[{{__Integer} ..} -> i : {__List} /; Length[i] >= 2]];
     (*defining graph properties of the image *)
     imagegraph = MorphologicalGraph@MorphologicalTransform[img, {"Fill"}];
     imggraphweight = AssociationThread[(EdgeList[imagegraph] /.
          UndirectedEdge -> List) -> PropertyValue[imagegraph, EdgeWeight]];
     imggraphpts = Nearest@Reverse[Thread[VertexList[imagegraph] ->
          PropertyValue[imagegraph, VertexCoordinates]], 2];
     (* corresponding nodes on the graph *)
     vertexpairs = Union@*Flatten@*imggraphpts /@ (Values[likelymergers]);
     (* find pairs < than 3 edgeweights away, take a mean of vertices
      and update the association with mean position *)posVertexMergers =
      Position[Thread[Lookup[imggraphweight, vertexpairs] < 3], True];
     If[posVertexMergers != {},
      meanVertices =
       MapAt[List@*N@*Mean, likelymergers, Thread[{Flatten@posVertexMergers, 2}]];
      Scan[(vertexset[#[[1]]] = #[[2]]) &, meanVertices]];
     KeyMap[Union@*Flatten]@vertexset // Normal]
    ]
];
```

In[8]:=
```
formAndComputeMatrices[vertexCoordinatelookup_, inds_, colsOrder_, edgenum_,
   delV_, vertexToCells_, vertexvertexConn_, maxcellLabels_, filteredvertices_,
   vertexAssoc_] := Module[{tx, ty, tensinds, filteredvertexnum, relabelvert,
spArrayTx, spArrayTy, spArrayPx, spArrayPy, spArrayX, spArrayY, $filteredvertices},
```

```
{tx, ty} = Transpose[
  With[{target = vertexCoordinatelookup[#[[2]]],
      source = vertexCoordinatelookup[#[[1]]]},
     (target - source) / Norm[target - source]
    ] & /@ inds];
Print["Tension coefficients computed: ", Style["☑", 20]];
MapThread[Print[Style["counts of zero coefficients " <> #1, Red], Count[#2, 0.]] &,
 {{"Tx: ", "Ty: "}, {tx, ty}}];
$filteredvertices = Complement[filteredvertices, delV];
filteredvertexnum = Length@$filteredvertices;
relabelvert =
 AssociationThread[$filteredvertices -> Range[Length@$filteredvertices]];
tensinds = Thread[{Lookup[relabelvert, Part[inds, All, 1]], colsOrder}];
spArrayTx = spArrayTy = SparseArray@ConstantArray[0, {filteredvertexnum, edgenum}];
MapThread[(spArrayTx[[Sequence @@ #1]] = #2) &, {tensinds, tx}];
MapThread[(spArrayTy[[Sequence @@ #1]] = #2) &, {tensinds, ty}];
spArrayPx =
 spArrayPy = SparseArray@ConstantArray[0, {filteredvertexnum, maxcellLabels}];
MapThread[Print[Style[#1 <> "coefficients stats: ", Blue], Counts@
    Map[Total@*Unitize, Normal[#2]]] &, {{"Tx ", "Ty "}, {spArrayTx, spArrayTy}}];
Print[Style["Tension coefficients dist: ", Bold], Histogram[{{}, Join[spArrayTx[
      "NonzeroValues"], spArrayTy["NonzeroValues"]]}, 20, ImageSize -> Small]];

Block[{neighbouringCells, bisectionlabels, bisectpts, centroid,
  orderingT, vertexcoords, orderptsT, orderIndT, orderCells, kk = 0, px, py},
 With[{cellToVertexLabelsT = Reverse[vertexToCells, 2],
   edgeVertexPart = GroupBy[vertexvertexConn~Flatten~1 , First -> Last]},
  With[{cellToAllVerticesT =
     GroupBy[Flatten[Thread /@ cellToVertexLabelsT], First -> Last]},
   Do[
    neighbouringCells = vertexToCells[[i, 2]];
    (* for vertex, the neighbouring cell labels *)bisectionlabels =
     Intersection[cellToAllVerticesT[#], edgeVertexPart[i]] & /@ neighbouringCells ;
    If[Length[neighbouringCells] > 2 && MatchQ[bisectionlabels,
       {Repeated[{_, _}, {3, ∞}]}], (vertexcoords =
       DeleteDuplicates[vertexCoordinatelookup[#] & /@ Flatten@bisectionlabels];
      centroid = Mean@vertexcoords;
      orderingT = Ordering[
        ArcTan[Last@# - Last@centroid, First@# - First@centroid] & /@ vertexcoords];
      orderptsT = vertexcoords[[orderingT]];
      orderIndT =
       Partition[Lookup[vertexAssoc, Append[orderptsT, First@orderptsT]], 2, 1];
      orderCells  = Last @@@ Position[(x ↦ Map[Intersection[x, #] &, orderIndT]) /@
          (cellToAllVerticesT[#] & /@ neighbouringCells), x_ /; Length[x] == 2];
      neighbouringCells = neighbouringCells[[orderCells]];
      bisectpts = Map[vertexCoordinatelookup, orderIndT, {2}];
      {px, py} = Transpose[
        {(#[[2, 2]] - #[[1, 2]]) / 2, - (#[[2, 1]] - #[[1, 1]]) / 2} & /@ bisectpts];
      If[MemberQ[px, 0.] || MemberQ[py, 0.], kk++];
      {px, py})
```

```
        ];
        Scan[(spArrayPx[[Sequence @@ #[[1]]]] = #[[2]]) &,
         Thread[Thread[{relabelvert@i, neighbouringCells}] -> px]];
        Scan[(spArrayPy[[Sequence @@ #[[1]]]] = #[[2]]) &, Thread[
          Thread[{relabelvert@i, neighbouringCells}] -> py]], {i, $filteredvertices}]
     ]
    ];
   Print["Pressure coefficients computed: ", Style["☑", 20]];
   Print[Style["Pressure coefficients zero: ", Red], kk];
  ];
  MapThread[Print[Style[#1 <> "coefficients stats: ", Blue], Counts@
      Map[Total@*Unitize, Normal[#2]]] &, {{"Px ", "Py "}, {spArrayPx, spArrayPy}}];
  Print[Style["pressure coefficients dist: ", Bold], Histogram[{{}, Join[
      spArrayPx["NonzeroValues"], spArrayPy["NonzeroValues"]]}, ImageSize -> Small]];
  spArrayX = Join[spArrayTx, spArrayPx, 2];
  spArrayY = Join[spArrayTy, spArrayPy, 2];
  {spArrayX, spArrayY, Dimensions[spArrayTx], Dimensions[spArrayPx]}
 ];
```

In[9]:=
```
(* maximize Log-likelihood function *)
maximizeLogLikelihood[spArrayX_, spArrayY_, dimTx_, dimPx_] :=
  Module[{range = 10.0^Range[-1.5, 1.5, 0.1], sol, spA, spg, spB, n, m, spb, τ, SMatrix,
    Q, R, H, h, logL, μ, p}, Print[Style["\nwith maximum likelihood", Bold, 18]];
   sol = With[{ls = range},
     (*spA=SparseArray@(Join[spArrayX,spArrayY]);*)
     spA = SparseArray@(Flatten[Transpose[{spArrayX, spArrayY}], 1]);
     spg = SparseArray@
       (ConstantArray[1., Last@dimTx]~Join~ConstantArray[0., Last@dimPx]);
     spB = SparseArray@DiagonalMatrix[spg];
     n = First@Dimensions@spA;
     m = (Length[#] - Total@#) &@Diagonal[spBᵀ.spB];
     With[{DimspB = First[Dimensions@spB]},
      spb = SparseArray@ConstantArray[0., First[Dimensions@spA]];
      Table[(τ = Sqrt[μ];
        SMatrix = SparseArray@
          (Map[Flatten]@Transpose[{Join[spA, τ spB], Join[spb, τ spg]}, {2, 1}]);
        {Q, R} = SparseArray /@ QRDecomposition[SMatrix];
        R = DiagonalMatrix[Sign[Diagonal@R]].R;
        H = R[[ ;; #,  ;; #]] &@DimspB;
        h⃗ = R[[ ;; #, # + 1]] &@DimspB;
        h = R[[# + 1, # + 1]] &@DimspB;
        logL =
         - (n - m + 1) * Log[h^2] + Total[Log[Diagonal[μ (spBᵀ.spB)]["NonzeroValues"]]] -
          2 * Total[Log[Diagonal[H[[1 ;; -2, 1 ;; -2]]]["NonzeroValues"]]]), {μ, ls}]
      ]
     ];
   Print[ListPlot[{sol, sol}, Joined -> {True, False}, PlotStyle ->
      {{Red, Thick}, {PointSize[0.02], Black}}, AxesStyle -> {{Black}, {Black}},
     AxesLabel -> {"index μ", "LogLikelihood"}, Background -> LightBlue]];
   μ = Keys @@ MaximalBy[Thread[range -> sol], Values, 1];
   Print["optimized value of μ: ", μ];
   τ = Sqrt[μ];
   With[{DimspB = First[Dimensions@spB]},
    SMatrix = SparseArray@
      (Map[Flatten]@Transpose[{Join[spA, τ spB], Join[spb, τ spg]}, {2, 1}]);
    {Q, R} = SparseArray /@ QRDecomposition[SMatrix];
    R = DiagonalMatrix[Sign[Diagonal@R]].R;
    H = R[[ ;; #,  ;; #]] &@DimspB;
    h⃗ = R[[ ;; #, # + 1]] &@DimspB;
   ];
   p = PseudoInverse[H].h⃗; p
  ];
```

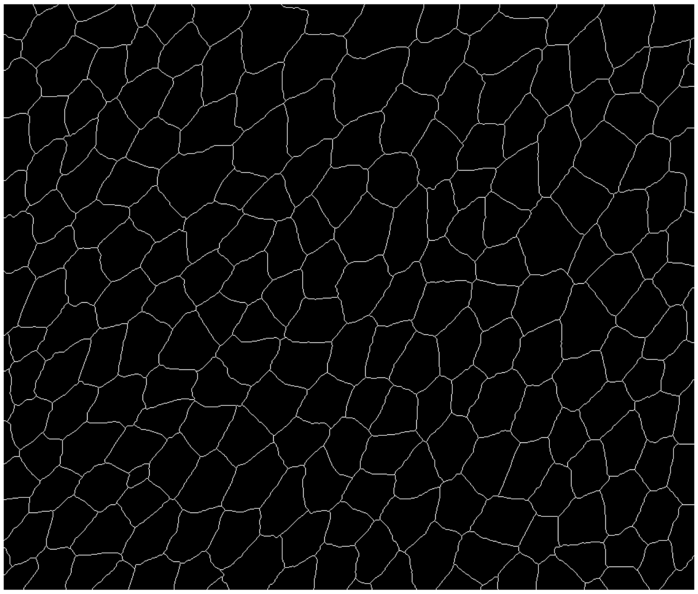
In[10]:=
```
plotMaps[p_, segmentation_, edgeImg_, maxcellLabels_, dimTx_,
   vertexToCells_, vertexCoordinatelookup_, edgeLabels_, border_] := Module[
   {cellToVertexLabels, cellToAllVertices, ptsEdges, k, v, ord, edgeptAssoc, poly, pts,
    mean, ordering, orderpts, tvals, cols, pvals, removecollabels, collabels,
    pressurecolours}, cellToVertexLabels = Reverse[vertexToCells, 2];
   cellToAllVertices = GroupBy[Flatten[Thread /@ cellToVertexLabels], First -> Last];
   (* polygons *)
   ptsEdges = {{1, 1}, Reverse@Dimensions[segmentation],
     {Last[Dimensions@segmentation], 1}, {1, First[Dimensions@segmentation]}};
   {k, v} = {Keys@#, Values[#][[All, 2]]} &@ComponentMeasurements[
      segmentation, {"AdjacentBorderCount", "Centroid"}, # == 2 &];
   ord = Flatten[Function[x, Position[#, Min[#]] &@
        Map[EuclideanDistance[#, x] &, ptsEdges]] /@ v];
   edgeptAssoc = Association[Rule @@@ Thread[{k, ptsEdges[[ord]]}]];
   poly = (
       pts = vertexCoordinatelookup /@ cellToAllVertices[#];
       If[MemberQ[k, #], AppendTo[pts, edgeptAssoc[#]], pts];
       mean = Mean[pts];
       ordering = Ordering[ArcTan[Last@# - Last@mean, First@# - First@mean] & /@ pts];
       orderpts = pts[[ordering]];
       Polygon@Append[orderpts, First@orderpts]
      ) & /@ Range[maxcellLabels];

   tvals = Rescale@p[[1 ;; Last@dimTx]];
   cols = ColorData["Rainbow"][#] & /@ tvals;
   Print["Tension map:"];
   Print[Graphics[{Thickness[0.005], Riffle[cols, Line /@ Values@edgeLabels]}]];

   pvals = p[[ Last[dimTx] + 1 ;;]];
   If[border,
    removecollabels =
     Keys@ComponentMeasurements[segmentation, "AdjacentBorders", Length[#] > 0 &],
    removecollabels = Values@Last@ComponentMeasurements[
        Dilation[segmentation, 1] /. 0 → (maxcellLabels + 1), "Neighbors"]
   ];
   collabels = Complement[Range@maxcellLabels, removecollabels];
   pressurecolours = ColorData["Rainbow"][#] & /@ Rescale[(pvals[[collabels]])];
   Print["Pressure map:"];
   Print@Show[Graphics@Riffle[pressurecolours, poly[[collabels]]], edgeImg]
  ];
```

In[11]:=
```
ForceInference[filename_, Imgborder : True | False : True] :=
  Module[{Img, ImgC, segmentation, maxcellLabels, cellsToVertices,
    vertexnum, edges, smalledges, maxedgeLabels, edgeEndPoints, nearest,
    nearestedgeEndPoints, edge2pixLabels, pos, oldCoords, vertexAssoc,
    vertexToCells, filteredvertices, filteredvertexnum, relabelvert,
    edgeLabels, edgenum, spArrayTx, spArrayTy, vertexCoordinatelookup,
    vertexpairs, vertexvertexConn, inds, edgelabelToVert, delV, vertToedges,
    edgeImg, colsOrder, p, spArrayX, spArrayY, dimTx, dimPx}, LaunchKernels[];
```

```
ImgC = Img = Binarize@ColorConvert[Import[filename], "Grayscale"];
If[Not@Imgborder, Img = closeImage@Img];
Print[Image[Img, ImageSize -> Medium]];
segmentation = segmentImage[Img, Imgborder];
Print[Colorize@segmentation];
Print["Image segmented: ", Style["☑", 20]];
maxcellLabels = Max@Values@ComponentMeasurements[segmentation, "LabelCount"];
cellsToVertices = associateVertices[Img, segmentation];
Print["vertices found and associated: ", Style["☑", 20]];
vertexnum = Length@cellsToVertices;

edges = MorphologicalComponents@ImageFilter[
   If[#[[3, 3]] == 1 && Total[#[[2 ;; -2, 2 ;; -2]], 2] == 3, 1, 0] &, ImgC, 2];
(* associate vertices with all edges. for pixel value 1 edge find
  two nearest pts. for all edges <3, merge the pts together;
make changes to the cellToVertex *)
(* edges to be deleted *)
smalledges =
 Flatten[Position[Values@ComponentMeasurements[edges, "Count"], 1 | 2]];
maxedgeLabels = Max@edges;
edgeEndPoints = With[{comp = Values@ComponentMeasurements[edges, "Mask"]},
  ParallelTable[
   If[Total[#] == 1, ImageValuePositions[#, 1],
      ImageValuePositions[MorphologicalTransform[#, "SkeletonEndPoints"], 1]] &@
    Binarize@Image[comp[[i]]],
   {i, maxedgeLabels}]
 ];
(* for small edge: if one pixel delete *)
edges =
 Fold[If[Length@edgeEndPoints[[#2]] == 1, #1 /. #2 -> 0, #1] &, edges, smalledges];
nearest = Nearest@Flatten[Values[cellsToVertices], 1];
nearestedgeEndPoints = Map[Flatten@*nearest, edgeEndPoints, {2}];
(* if edge is two pixels then put
 average value in the cellsToVertices: *)edge2pixLabels =
 Keys@Cases[ComponentMeasurements[edges, "Count"], HoldPattern[_ -> 2]];
If[edge2pixLabels != {},
  (oldCoords = nearestedgeEndPoints[[#]];
    pos = Position[cellsToVertices, #, Infinity] & /@ oldCoords;
    cellsToVertices =
     Fold[ReplacePart[#1, #2 -> Mean@oldCoords] &, cellsToVertices, pos]
   ) & /@ edge2pixLabels
];
edges = ArrayComponents[edges /. Thread[edge2pixLabels -> 0]];
Print["edges found and associated: ", Style["☑", 20]];

cellsToVertices = Normal@
  AssociationMap[Reverse, GroupBy[cellsToVertices, Last -> First, Union@*Flatten]];
vertexnum = Length@cellsToVertices;
(* Length@Flatten[Values@cellsToVertices,1]; *)
nearest = Nearest@Flatten[Values@cellsToVertices, 1];
edgeEndPoints = Delete[edgeEndPoints, Partition[smalledges, 1]];
```

```
    nearestedgeEndPoints = Map[Flatten@*nearest, edgeEndPoints, {2}];
    vertexAssoc =
     AssociationThread[Flatten[Values@cellsToVertices, 1], Range@vertexnum];
    vertexToCells = Reverse[MapAt[vertexAssoc[#] &,
       MapAt[Flatten, cellsToVertices, {All, 2}], {All, 2}], 2];
    filteredvertices = Keys@Select[<|vertexToCells|>, (Length[#] > 2 &)];
    filteredvertexnum = Length@filteredvertices;
    (* till above we have isolated all vertices that share three or more edges;
    we can relabel those filtered vertices to be the rows of the matrix *)
    relabelvert = AssociationThread[filteredvertices -> Range[Length@filteredvertices]];
    (* all edges are relabeled to have a unique identity *)
    edgeLabels = AssociationThread[Range[Length@#] -> #] &[nearestedgeEndPoints];
    edgenum = Max[Keys@edgeLabels];
    vertexCoordinatelookup = AssociationMap[Reverse, vertexAssoc];
    (* given the vertex label → get the coordinates from the original lookup *)
    vertexpairs = Map[vertexAssoc, nearestedgeEndPoints, {2}];
    (* edge coordinates to vertex label. take vertices one by one and
     find all the edges it is a part of. None should be less than 3 *)
    vertexvertexConn = ParallelTable[
      Cases[vertexpairs, {OrderlessPatternSequence[i, p_]} :> {i, p}],
      {i, filteredvertices}];

    delV = Cases[vertexvertexConn, {{p_, _}, {p_, _}} :> p];
    vertexvertexConn = DeleteCases[vertexvertexConn, {_, _}];
    inds = Flatten[vertexvertexConn, 1];
    (* edgelabel → vertices *)
    edgelabelToVert = Map[vertexAssoc, edgeLabels, {2}];
    (* vertices → edgelabel *)
    vertToedges = Normal@AssociationMap[Reverse, edgelabelToVert];
    colsOrder = Flatten[Cases[vertToedges, PatternSequence[
          {OrderlessPatternSequence @@ #} -> p_] :> p, Infinity] & /@ inds];
    edgeImg = Graphics[{Thickness[0.005], Line@Lookup[vertexCoordinatelookup,
          edgelabelToVert[#]] & /@ colsOrder}];

    {spArrayX, spArrayY, dimTx, dimPx} =
     formAndComputeMatrices[vertexCoordinatelookup, inds, colsOrder, edgenum, delV,
      vertexToCells, vertexvertexConn, maxcellLabels, filteredvertices, vertexAssoc];
    p = maximizeLogLikelihood[spArrayX, spArrayY, dimTx, dimPx];
    plotMaps[p, segmentation, edgeImg, maxcellLabels,
     dimTx, vertexToCells, vertexCoordinatelookup, edgeLabels, Imgborder];
   ];
```

In[13]:=
```
Quiet[AbsoluteTiming@
  ForceInference["C:\\Users\\aliha\\Desktop\\scripts-codes\\github codes\\wolfram
     force inference\\image.tif"]
]
```



border exists, segmenting with watershed components

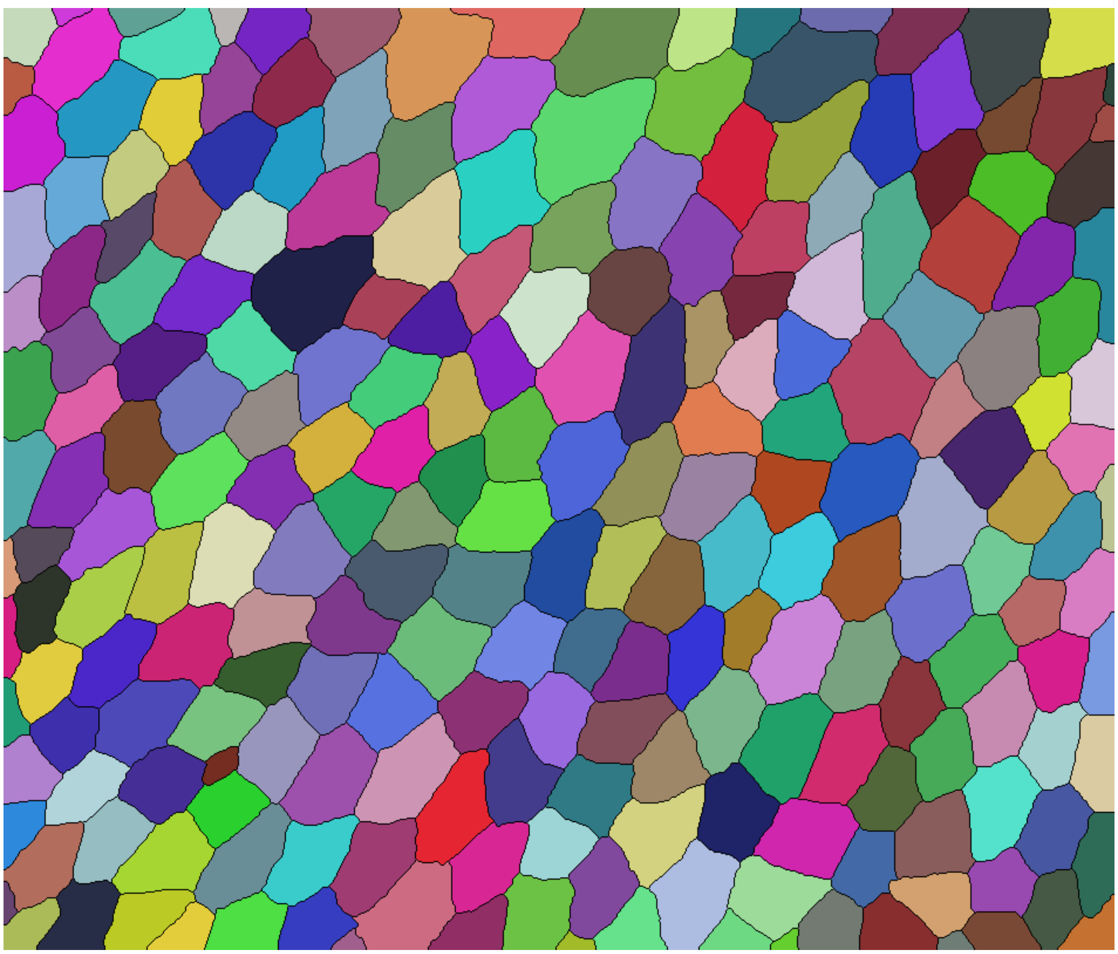


Image segmented: ☑

vertices found and associated: ☑

edges found and associated: ☑

Tension coefficients computed: ☑

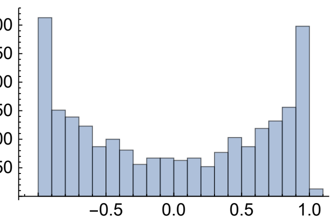
counts of zero coefficients Tx: 16

counts of zero coefficients Ty: 9

Tx coefficients stats: <|3 → 376, 2 → 16, 4 → 3|>

Ty coefficients stats: <|3 → 383, 2 → 9, 4 → 3|>

Tension coefficients dist:



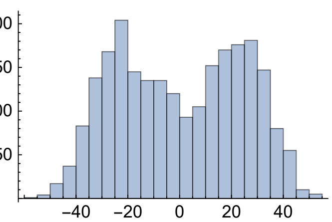
Pressure coefficients computed: ☑

Pressure coefficients zero: 15

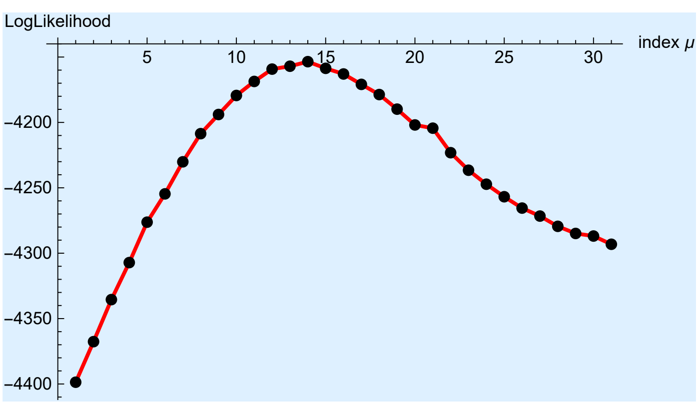
Px coefficients stats: <|3 → 387, 2 → 5, 4 → 3|>

Py coefficients stats: <|3 → 382, 4 → 3, 2 → 10|>

pressure coefficients dist:

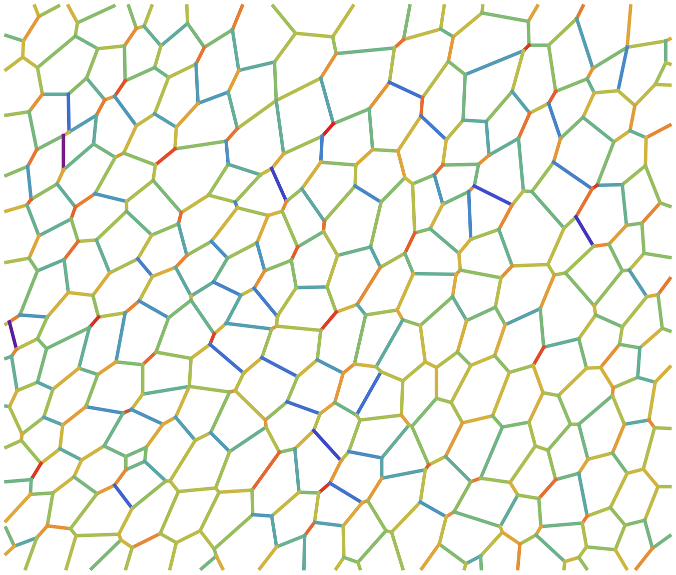


with maximum likelihood

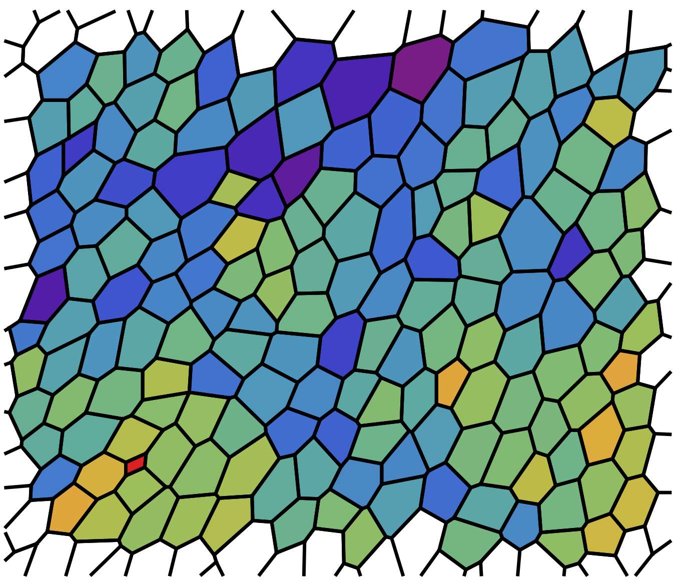


optimized value of μ: 0.630957

Tension map:

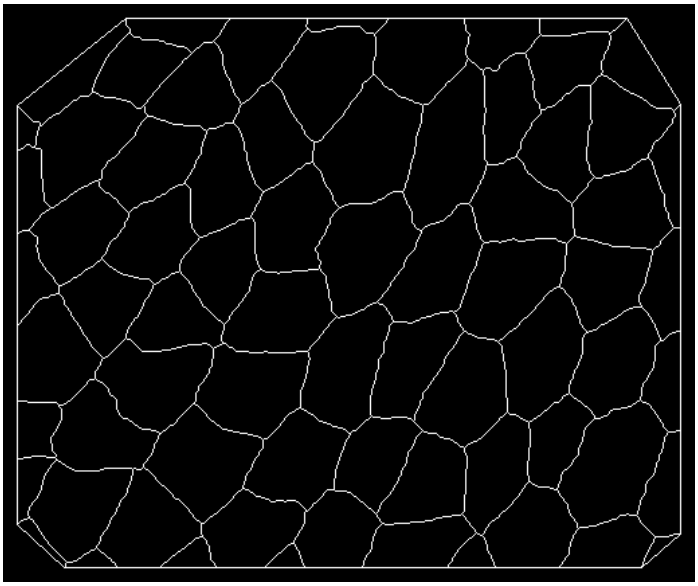


Pressure map:



Out[13]= {64.7283, Null}

In[14]:=
```
Quiet[AbsoluteTiming@
  ForceInference["C:\\Users\\aliha\\Desktop\\scripts-codes\\github codes\\wolfram
     force inference\\smallimagepadded.tif", False]
]
```



border not present, segmenting with morphological components

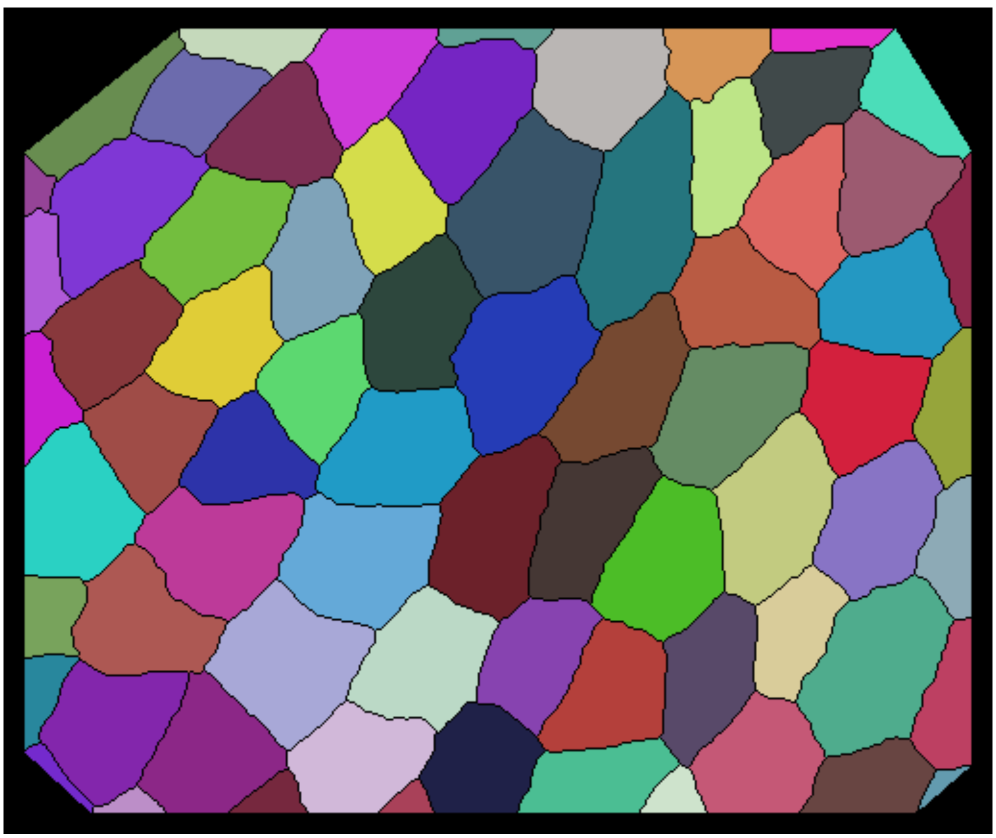


Image segmented: ☑

vertices found and associated: ☑

edges found and associated: ☑

Tension coefficients computed: ☑

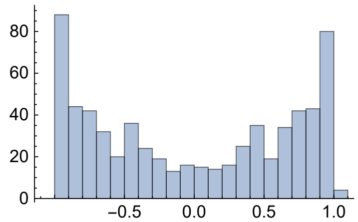
counts of zero coefficients Tx: 5

counts of zero coefficients Ty: 2

Tx coefficients stats: ⟨| 2 → 5, 3 → 105, 4 → 1 |⟩

Ty coefficients stats: ⟨| 3 → 110, 2 → 1 |⟩

Tension coefficients dist:



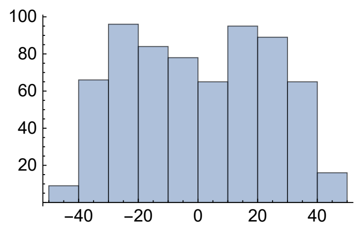
Pressure coefficients computed: ☑

Pressure coefficients zero: 5

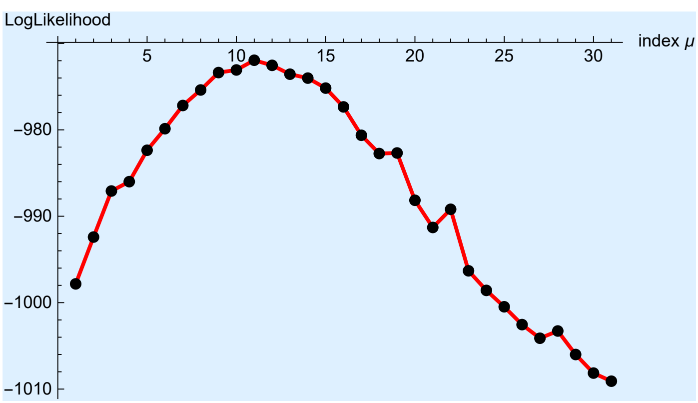
Px coefficients stats: ⟨| 3 → 107, 4 → 1, 2 → 3 |⟩

Py coefficients stats: ⟨| 3 → 108, 2 → 2, 4 → 1 |⟩

pressure coefficients dist:

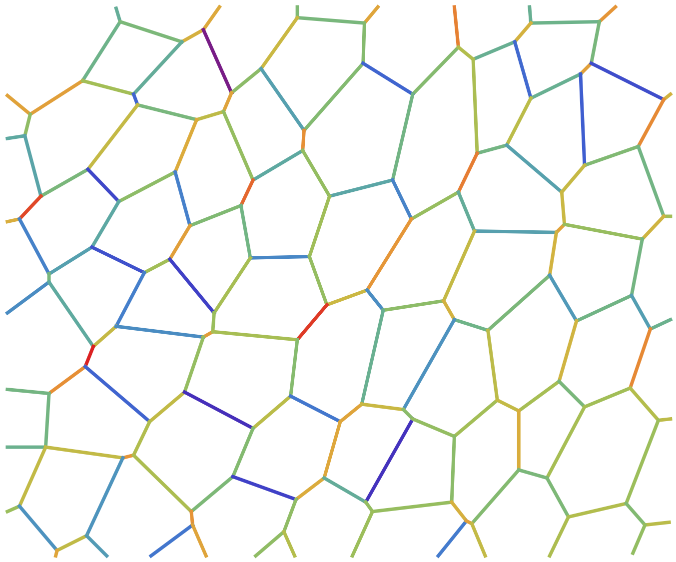


with maximum likelihood

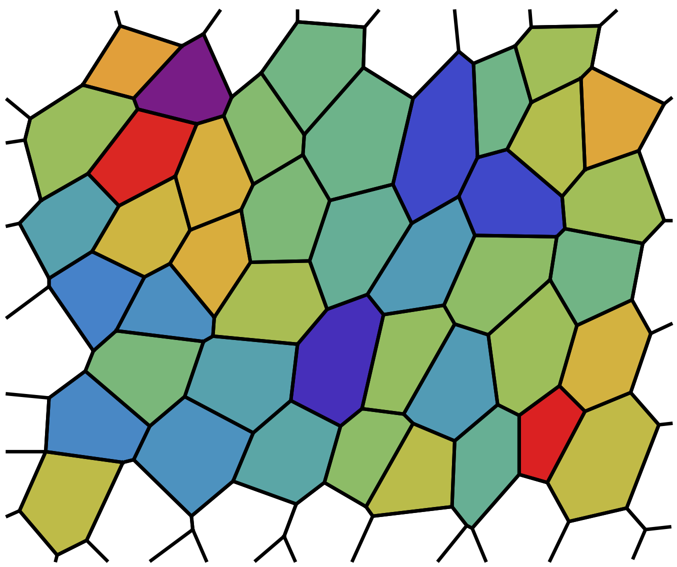


optimized value of μ: 0.316228

Tension map:



```
Pressure map:
```



Out[14]= {6.43651, Null}