order-reduction-model-of-Cahn-Hilliard-eq-03.ipy

Download this notebook in HTML

In [1]:

```
using VoronoiDelaunay
using VoronoiCells
using GeometryBasics
using LinearAlgebra
using Plots
using ProgressMeter
using JLD2
```

VoronoiDelaunay pkg で 連結行列と，接する generator 間距離を求める関数

In [2]:

```
# model constant.

# 距離の α乗に反比例して力が働くとする
α = 2

# ball の半径
rb = 0.02
```

Out [2]: 0.02

In [3]:

```
# generator 情報は以下のようなものを想定する

xmin, xmax = 0.0, 1.0
width = xmax - xmin

rect = Rectangle(Point2(xmin, xmin), Point2(xmax, xmax))
# points = [Point2(width*rand()+xmin, width*rand()+xmin) for _ in 1:numpts]
```

Out [3]: Rectangle{Point2{Float64}}([0.0, 0.0], [1.0, 1.0])

In [4]:

```
function make_pts(num)
    pts = [ Point2(width*rand()+xmin, width*rand()+xmin) ]

    for i in 2:num
      isadd = false
      while !(isadd)
          candidate = Point2(width*rand()+xmin, width*rand()+xmin)
          for j in 1:length(pts)
              l = norm(pts[j] - candidate)
              if l < 2*rb
                  isadd = false
                  break
              else
```

```
                    isadd = true
                end
            end
            if isadd
                push!(pts, candidate)
            end
        end
    end

    return pts
end
```

Out [4]: make_pts (generic function with 1 method)

In [5]:
```
points = make_pts(300)
```

Out [5]:
```
300-element Vector{Point2{Float64}}:
 [0.1467417454081682, 0.8384822025743587]
 [0.8429442100220145, 0.48986181641763116]
 [0.6501540805178629, 0.28184831351467543]
 [0.7335981690618417, 0.9865946956051994]
 [0.2791547890476116, 0.7418780561674926]
 [0.5002828519044215, 0.5230998885924372]
 [0.6567539665543662, 0.16812918954722333]
 [0.19831163051079548, 0.7753341165493735]
 [0.7415845415895999, 0.2538938903083774]
 [0.48204541731790607, 0.8343384901872863]
 [0.26258940164824407, 0.8325571148595905]
 [0.7252753343396348, 0.416670222104351]
 [0.08514728942780037, 0.47504496336924196]
 ⋮
 [0.666992986798021, 0.0651253483778903]
 [0.36838575171483734, 0.9372408291118163]
 [0.7043891737110506, 0.5903670025523076]
 [0.840119554925697, 0.724664233792082]
 [0.7788101010739399, 0.7239392416385924]
 [0.22814569740406376, 0.5484240287495462]
 [0.45174620386193, 0.19608793807613478]
 [0.7406714527690292, 0.7601646713893868]
 [0.7836599533996346, 0.8949465125196109]
 [0.07256541672627048, 0.43197371550065655]
 [0.4204845132254954, 0.2801589352611128]
 [0.9995475105941775, 0.4192680475166525]
```

In [6]:
```
# VoronoiDelauney を含め，幾何的ライブラリは、座標が 1+ϵ 以上 2 - 2ϵ 以下であるように
# 制限されていることが多い．
# さらに，VD は余計な4隅の点を generator として勝手に加えてしまうので，
# その影響を避けるため全体を小さくする変換関数，と逆変換を用意する

function limit_to_12sp(x, xmin, xmax)
  return (x - xmin)/(3*(xmax - xmin)) + 4.0/3
end

# 逆に戻す.
function expand_from_12sp(y, xmin, xmax)
  return 3*(xmax - xmin)*(y - 4.0/3) + xmin
end

# 座標 (x,y) を渡しての変換.
limit_to_12sp(v::Point2, xmin, xmax) =
  Point2D( limit_to_12sp(v[1], xmin, xmax),limit_to_12sp(v[2], xmin, xmax) )

expand_from_12sp(v::Point2, xmin, xmax) =
```

```
  Point2D( expand_from_12sp(v[1], xmin, xmax), expand_from_12sp(v[2], xmin, xmax)
```

Out [6]: expand_from_12sp (generic function with 2 methods)

In [7]:

```
import LinearAlgebra: norm
norm( x::Point2D ) = norm( [ x._x, x._y ])

function close(x,y)
    dist = norm( x - y )
    if dist < 1.5*eps()
        return true
    else
        return false
    end
end

# Point list の中の，どの point に該当するかを返す関数
function findpt(pts, pt)
    for i in 1:length(pts)
        if close(pts[i], pt)
            return i
        end
    end
    return false
end
```

Out [7]: findpt (generic function with 1 method)

In [8]:

```
function make_cx_l(pts)
    num = length(pts)
    pts_12 = [ limit_to_12sp( pts[n], xmin, xmax ) for n in 1:num ]
    tess = DelaunayTessellation()
    push!( tess, copy(pts_12) )
    # DelaunayTesselation は渡した点の順番を変えてしまうのでコピーを渡す.

    # 一応，お絵かきのために.
    x, y = getplotxy(voronoiedges(tess))
    ox = [ expand_from_12sp( x[i], xmin, xmax) for i in 1:length(x) ]
    oy = [ expand_from_12sp( y[i], xmin, xmax) for i in 1:length(y) ]
    # この ox, oy を返しておく.

    # お絵かきは次のような感じで.
    # plot(ox, oy, xlims = (0,1.0), ylims = (0,1.0), aspect_ratio = 1.0)
    # scatter!(pts, markersize = 4, label = "generators", aspectratio = 1.0)
    # annotate!([(pts[n][1] + 0.02, pts[n][2] + 0.03, Plots.text(n)) for n in 1:n

    edgepts = [ (geta(edge), getb(edge)) for edge in delaunayedges(tess)]
    # delaunay edge の端点を全て列挙する.

    omatc = zeros(Bool, num, num)
    for n in 1:length(edgepts) # 連結行列を作る
        pa, pb = edgepts[n][1], edgepts[n][2]
        i, j = findpt(pts_12, pa), findpt(pts_12, pb)
        omatc[i,j] = omatc[j,i] = true
```

```
    end

    omatl = zeros(num, num)
    # 本来の座標系での真の格子点間距離
    for i in 1:num
        for j in (i+1):num
            if omatc[i,j]
                xi, yi = pts[i][1], pts[i][2]
                xj, yj = pts[j][1], pts[j][2]
                omatl[i,j] = omatl[j,i] = sqrt( (xi-xj)^2 + (yi-yj)^2 )
            end
        end
    end

    return (ox, oy, omatc, omatl)
end
```

Out [8]: make_cx_l (generic function with 1 method)

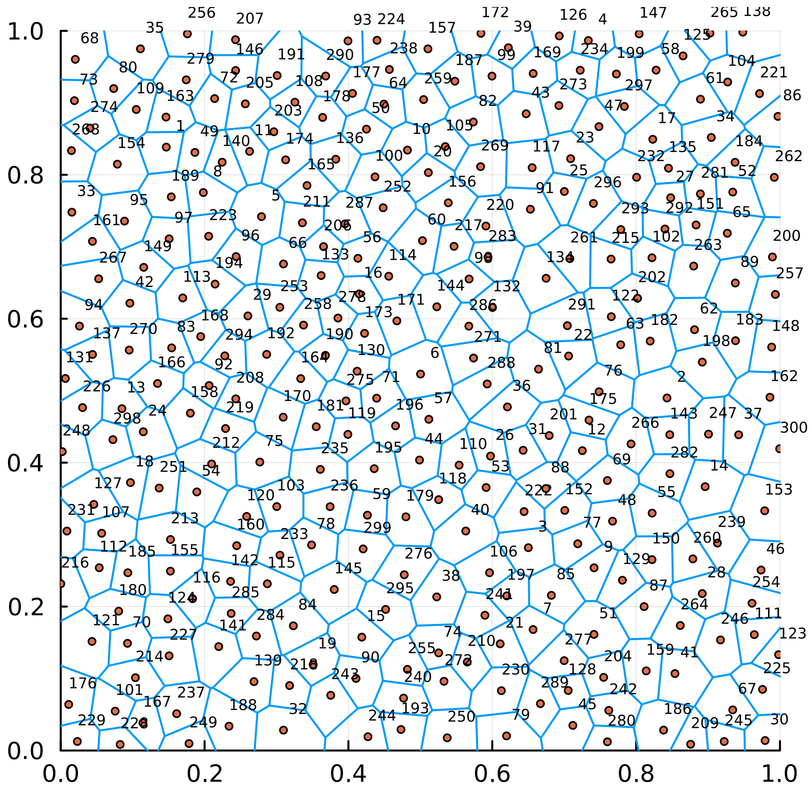
In [9]:

```
(ox, oy, omatc, omatl) = make_cx_l(points);
```

In [10]:

```
plot(ox, oy, xlims = (0,1.0), ylims = (0,1.0), aspect_ratio = 1.0, legend = false
scatter!(points, markersize = 2, label = "generators", aspectratio = 1.0)
annotate!([(points[n][1] + 0.02, points[n][2] + 0.03, Plots.text(n, 5)) for n in
```

Out [10]:



In [16]:

```
omatc
```

Out [16]:

```
100×100 Matrix{Bool}:
 0  0  0  0  0  0  0  0  0  0  0  0  0  …  0  0  1  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0     0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0     0  0  0  0  0  0  0  1  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0     0  0  0  0  0  0  1  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  1  0     0  1  0  0  0  0  0  0  0  0  0  0
```

```
 0  0  0  0  0  0  0  0  0  0  0  0  0  …  0  0  0  0  0  0  0  0  0  0  1  0
 0  0  0  0  0  0  0  0  0  0  0  0  0     0  0  0  1  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0     0  0  0  0  0  1  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0     0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  1  0  0     0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  1  0  0  0  …  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  1  0  0  0  0  0  0  0  0     0  1  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0     0  0  0  0  0  0  0  0  1  0  0  0
 ⋮              ⋮              ⋮        ⋱        ⋮              ⋮
 0  0  0  0  0  0  0  0  0  0  0  0  0     0  0  0  0  0  0  0  1  0  0  0  0
 0  0  0  0  1  0  0  0  0  0  0  1  0     0  0  0  0  0  0  0  0  0  0  0  0
 1  0  0  0  0  0  0  0  0  0  0  0  0  …  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  1  0  0  0  0  0  0     0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0     0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  1  0  0  0  0  0     0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  1  0  0  0  0  0  0  0  0  0     0  0  0  0  0  0  0  0  0  0  0  0
 0  0  1  0  0  0  0  0  0  0  0  0  0  …  1  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  1     0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0     0  0  0  0  0  0  0  0  0  0  1  0
 0  0  0  0  0  1  0  0  0  0  0  0  0     0  0  0  0  0  0  0  0  0  1  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0     0  0  0  0  0  0  0  0  0  0  0  0
```

In [17]:

```
omatl
```

Out [17]:

```
100×100 Matrix{Float64}:
 0.0       0.0  0.0       0.0       …  0.0        0.0       0.0        0.0
 0.0       0.0  0.0       0.0          0.0        0.0       0.0        0.0
 0.0       0.0  0.0       0.0          0.0        0.0       0.0        0.0
 0.0       0.0  0.0       0.0          0.0        0.0       0.0        0.0
 0.0       0.0  0.0       0.0          0.0        0.0       0.0        0.0
 0.0       0.0  0.0       0.0       …  0.0        0.0       0.0545016  0.0
 0.0       0.0  0.0       0.0          0.0        0.0       0.0        0.0
 0.0       0.0  0.0       0.0          0.0        0.0       0.0        0.0
 0.0       0.0  0.0       0.0          0.0        0.0       0.0        0.0
 0.0       0.0  0.0       0.0          0.0        0.0       0.0        0.0
 0.0       0.0  0.0       0.0       …  0.0        0.0       0.0        0.0
 0.0       0.0  0.0       0.0          0.0        0.0       0.0        0.0
 0.0       0.0  0.0       0.0          0.0448986  0.0       0.0        0.0
 ⋮                                  ⋱
 0.0       0.0  0.0       0.0          0.0        0.0       0.0        0.0
 0.0       0.0  0.0       0.0          0.0        0.0       0.0        0.0
 0.093083  0.0  0.0       0.0       …  0.0        0.0       0.0        0.0
 0.0       0.0  0.0       0.0          0.0        0.0       0.0        0.0
 0.0       0.0  0.0       0.0          0.0        0.0       0.0        0.0
 0.0       0.0  0.0       0.0          0.0        0.0       0.0        0.0
 0.0       0.0  0.0       0.148708     0.0        0.0       0.0        0.0
 0.0       0.0  0.135236  0.0       …  0.0        0.0       0.0        0.0
 0.0       0.0  0.0       0.0          0.0        0.0       0.0        0.0
 0.0       0.0  0.0       0.0          0.0        0.0       0.159645   0.0
 0.0       0.0  0.0       0.0          0.0        0.159645  0.0        0.0
 0.0       0.0  0.0       0.0          0.0        0.0       0.0        0.0
```

generator i に働く力(ベクトル)を計算

$f = \sum_{j:\mbox{neighbor}} (l_{i,j})^{-\alpha} v_{i\mbox{to}j}$

In [11]:

```
function force(i, pts, mat_conn, mat_len)
    cri_r = 2 *(π/4) * rb # 体積排除距離の2倍が本来だが…
    # cri_r = 2.5 *(π/4) * rb # 体積排除距離の2倍が本来だが…

    coeff_f = 2000 * (0.05)^(2-α)  # 力 = 1 / (coeff * l^alpha) とする.

   #  is_exc = false # 体積排除が働いたか

    cri_rev = 10 * (eps())^(1/α) # 逆数を安全にとるための臨界値

    v = Point2(0.0, 0.0)

    num = length(pts)
```

```
    for j in 1:num
      #   evs = [ Point2(0.0, 0.0)] # dummy
        if mat_conn[i,j]
            ll = mat_len[i,j]
            vl = pts[j] - pts[i]
            nvl = norm( vl )
            if ll > cri_r
                v = v + (1.0/ ( (ll^α) * nvl ) )* vl
            else
                is_exc = true
                if ll > cri_rev
                    v = v - (2.0/ ((ll^α) * nvl)) * vl # 体積排除効果
                else
                    v = v - (2.0/ (cri_rev^α * nvl) ) * vl
                end
                # push!(evs, (1/norm(vl)) * vl )
            end
        end
    end
         # if length(evs) == 2 # 境界にくっついてその方向へは動けない.
           # v -= dot( v, evs[2] ) * evs[2]
           # v = 0.5 * v # 横への動きは摩擦で少し弱く.
        # end
        # if length(evs) > 2 # 多角形角にハマって動けない.
           # v = Point2(0.0, 0.0)
        # end
       # for i in 1:length(evs)
       #    v -= dot( v, evs[i] ) * evs[i]
       # end
       # if length(evs) > 1 # 横へずれるケースは力を少し弱くする(摩擦的に)
       #     v = 0.5*v
       #  end


       v = (1/coeff_f) * v

        nv = norm(v)
      # if (nv > 0.9) && !(is_exc) # 流体的な動きなので, あまり強い効果はサチる.
      if (nv > 0.9)  # 流体的な動きなので, あまり強い効果はサチる.
           v = ( tanh(nv)/nv ) * v
       end

    return v
end
```

Out [11]: force (generic function with 1 method)

In [12]:

```
# force vector from points
function fv(pts)
    num = length(pts)
   (vx, vy, mat_c, mat_l) = make_cx_l(pts)
    return [ force(i, pts, mat_c, mat_l) for i in 1:num ]
end
```

Out [12]: fv (generic function with 1 method)

時間発展は Runge-Kutta で.

In [13]:
```
r(u) = fv(u)
```

Out [13]: r (generic function with 1 method)

In [14]:
```
function RK(u)
    r1 = r(u)
    r2 = r(u + Δt/2 * r1)
    r3 = r(u + Δt/2 * r2)
    r4 = r(u + Δt * r3)
    return u + Δt * (r1 + 2*r2 + 2*r3 + r4)/6
end
```

Out [14]: RK (generic function with 1 method)

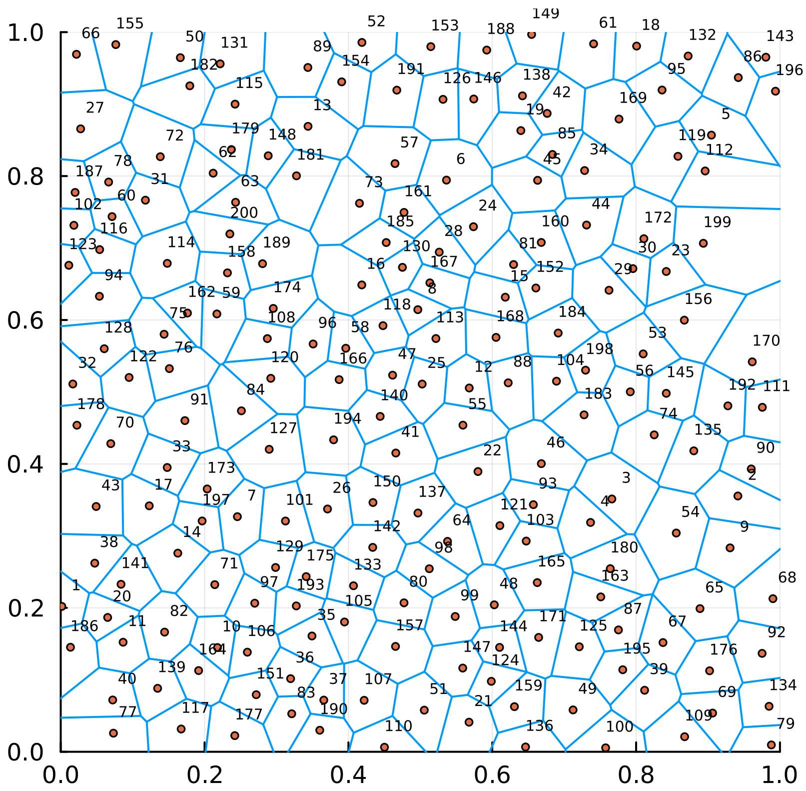
In [15]:
```
Δt = 0.0001
u0 = copy(points)

u = u0  # 最初の値はもちろん初期値
u_sq = [ u ] # 初期値を配列に入れておいて…

@showprogress for n in 1:400000
    u = RK(u)  # Runge-Kutta 法で新しい値をいきなり求める.
    push!(u_sq, u) # その値を配列に追加していく.
end
```

```
[32mProgress:  11%|████                                        |  ETA: 6:57:47[39mm58[39mm
```

In [16]:
```
function view_v(pts)
  (ox, oy, Connection, mat_l) = make_cx_l(pts)
  plot(ox, oy, xlims = (0,1.0), ylims = (0,1.0), aspect_ratio = 1.0, legend = fal
  scatter!(pts, markersize = 2, label = "generators", aspectratio = 1.0)
  annotate!([(pts[n][1] + 0.02, pts[n][2] + 0.03, Plots.text(n,5)) for n in 1:len
end
```

In [20]:
```
view_v( u_sq[1] )
```

Out [20]:



In [17]:

```
view_v( u_sq[end] )
```

In [18]:

```
using Printf # すぐ下の @sprintf を使いたいので.

function figure(dir, num)
    true_num = num * n_skip
    s = @sprintf("%8.7f", true_num * Δt)

    (ox, oy, Connection, mat_l) = make_cx_l(u_sq[true_num+1])
    plot(ox, oy, xlims = (0,1.0), ylims = (0,1.0), aspect_ratio = 1.0, legend = f
    scatter!(u_sq[true_num+1], markersize = 4, label = "generators", aspectratio
    # 時間をタイトルに表示

    savefig( dir * "/" * @sprintf("%06d", num) * ".png" )
    # 6桁の数字.png というファイル名で保存
end
```

In [ ]:

In [23]:

```
dir = "figures-ptcl-200"
run(`cmd /k mkdir $dir`)
```

```
c:\home\julia-programs\v1.9>
```

Out [23]:

```
Process(`[4mcmd[24m [4m/k[24m [4mmkdir[24m [4mfigures-ptcl-200[24m`, ProcessExited(0))
```

In [30]:

```
dir = "figures-ptcl-250"
run(`cmd /k mkdir $dir`)
```

```
c:\home\julia-programs\v1.9>
```

Out [30]: Process(`[4mcmd[24m [4m/k[24m [4mmkdir[24m [4mfigures-ptcl-250[24m`, ProcessExited(0))

In [19]:
```
dir = "figures-ptcl-300-2"
run(`cmd /k mkdir $dir`)
```

In [20]:
```
n_skip = 100
@showprogress for n in 0:div(length(u_sq), n_skip)
    figure(dir, n)
end
```

特殊な初期値を

In [16]:
```
using JLD2
@load "pi_curve_data.jld2" pi_dat
# 値は -1 から 1. 1 のところが π の形になっている.
```

Out [16]:
```
1-element Vector{Symbol}:
 :pi_dat
```

In [17]:
```
size(pi_dat)
```

Out [17]: (1000, 1000)

In [18]:
```
pi_dat
```

Out [18]:
```
1000×1000 Matrix{Float64}:
 -1.0  -1.0  -1.0  -1.0  -1.0  -1.0  …  -1.0  -1.0  -1.0  -1.0  -1.0  -1.0
 -1.0  -1.0  -1.0  -1.0  -1.0  -1.0     -1.0  -1.0  -1.0  -1.0  -1.0  -1.0
 -1.0  -1.0  -1.0  -1.0  -1.0  -1.0     -1.0  -1.0  -1.0  -1.0  -1.0  -1.0
 -1.0  -1.0  -1.0  -1.0  -1.0  -1.0     -1.0  -1.0  -1.0  -1.0  -1.0  -1.0
 -1.0  -1.0  -1.0  -1.0  -1.0  -1.0     -1.0  -1.0  -1.0  -1.0  -1.0  -1.0
 -1.0  -1.0  -1.0  -1.0  -1.0  -1.0  …  -1.0  -1.0  -1.0  -1.0  -1.0  -1.0
 -1.0  -1.0  -1.0  -1.0  -1.0  -1.0     -1.0  -1.0  -1.0  -1.0  -1.0  -1.0
 -1.0  -1.0  -1.0  -1.0  -1.0  -1.0     -1.0  -1.0  -1.0  -1.0  -1.0  -1.0
 -1.0  -1.0  -1.0  -1.0  -1.0  -1.0     -1.0  -1.0  -1.0  -1.0  -1.0  -1.0
 -1.0  -1.0  -1.0  -1.0  -1.0  -1.0     -1.0  -1.0  -1.0  -1.0  -1.0  -1.0
 -1.0  -1.0  -1.0  -1.0  -1.0  -1.0  …  -1.0  -1.0  -1.0  -1.0  -1.0  -1.0
 -1.0  -1.0  -1.0  -1.0  -1.0  -1.0     -1.0  -1.0  -1.0  -1.0  -1.0  -1.0
 -1.0  -1.0  -1.0  -1.0  -1.0  -1.0     -1.0  -1.0  -1.0  -1.0  -1.0  -1.0
  ⋮                             ⋮    ⋱          ⋮
 -1.0  -1.0  -1.0  -1.0  -1.0  -1.0     -1.0  -1.0  -1.0  -1.0  -1.0  -1.0
 -1.0  -1.0  -1.0  -1.0  -1.0  -1.0     -1.0  -1.0  -1.0  -1.0  -1.0  -1.0
 -1.0  -1.0  -1.0  -1.0  -1.0  -1.0  …  -1.0  -1.0  -1.0  -1.0  -1.0  -1.0
 -1.0  -1.0  -1.0  -1.0  -1.0  -1.0     -1.0  -1.0  -1.0  -1.0  -1.0  -1.0
 -1.0  -1.0  -1.0  -1.0  -1.0  -1.0     -1.0  -1.0  -1.0  -1.0  -1.0  -1.0
 -1.0  -1.0  -1.0  -1.0  -1.0  -1.0     -1.0  -1.0  -1.0  -1.0  -1.0  -1.0
 -1.0  -1.0  -1.0  -1.0  -1.0  -1.0     -1.0  -1.0  -1.0  -1.0  -1.0  -1.0
 -1.0  -1.0  -1.0  -1.0  -1.0  -1.0  …  -1.0  -1.0  -1.0  -1.0  -1.0  -1.0
 -1.0  -1.0  -1.0  -1.0  -1.0  -1.0     -1.0  -1.0  -1.0  -1.0  -1.0  -1.0
 -1.0  -1.0  -1.0  -1.0  -1.0  -1.0     -1.0  -1.0  -1.0  -1.0  -1.0  -1.0
 -1.0  -1.0  -1.0  -1.0  -1.0  -1.0     -1.0  -1.0  -1.0  -1.0  -1.0  -1.0
 -1.0  -1.0  -1.0  -1.0  -1.0  -1.0     -1.0  -1.0  -1.0  -1.0  -1.0  -1.0
```

In [43]:
```
modify(x) = 0.45 * (x+1) + 0.05
pi_dat_mod = modify.(pi_dat)

maximum(pi_dat_mod), minimum(pi_dat_mod)
```

Out [43]: (0.9500000000000001, 0.05)

In [ ]:
```
using StatsBase
countmap(round.(pi_dat; digits=1))
```

```
# おおよその，値の分布をみてみよう． 1 に近いところが約 30万箇所ある．
```

In [19]:

```
function realtoint(rate,x)
    r = Int( round( rate* x )) + 1
    if r > rate
        r = rate
    end
    return r
end
```

Out [19]: realtoint (generic function with 1 method)

In [40]:

```
function make_pts_from_view(num)

  # initial
  isadd = false
  while !(isadd)
      cdt = Point2(width*rand()+xmin, width*rand()+xmin)
      mi,mj = realtoint(1000, cdt[1]), realtoint(1000, cdt[2])
      isadd = false
      if rand() < pi_dat_mod[mj, mi] # 確率で.
        # if pi_dat[mj,mi] > cri
          global pts = [ cdt ]
          isadd = true
      end
  end

  for i in 2:num

    isadd = false
    while !(isadd)
        cdt = Point2(width*rand()+xmin, width*rand()+xmin)
        mi,mj = realtoint(1000, cdt[1]), realtoint(1000, cdt[2])

       # println(mi, ", ", mj)
#         for j in 1:length(pts)
            # l = norm(pts[j] - cdt)

            if rand() < pi_dat_mod[ mj, mi ]  # 確率で. 重なりは気にしないことに.
                isadd = true
                push!(pts, cdt)
             else
                # if (l < 2*rb) || ( pi_dat[mj,mi] < cri )
                isadd = false
                # break
             end

#        end

#        if isadd
#        end

    end
```

```
  end

  return pts
end
```

Out [40]: make_pts_from_view (generic function with 1 method)

In [44]:

```
numpts = 200
points = make_pts_from_view(numpts)
```

Out [44]:

```
200-element Vector{Point2{Float64}}:
 [0.7611808295825465, 0.9569326166561539]
 [0.7849285328506446, 0.9071719665065973]
 [0.5300769321772214, 0.9430898945487391]
 [0.7867767148996876, 0.8885733979479868]
 [0.6020901019367388, 0.5425329773860785]
 [0.7038785228963176, 0.08755125829342303]
 [0.6868887657401113, 0.8497572761021145]
 [0.3788701432891728, 0.8940334426414009]
 [0.623948035765945, 0.5104751609675175]
 [0.6641113045793525, 0.09970547932348794]
 [0.2637479003196277, 0.9231627816134118]
 [0.03961374272000018, 0.7224108161329479]
 [0.41109229301842376, 0.3119923593317997]
 ⋮
 [0.115405068569703, 0.24232124589679238]
 [0.5465047662813741, 0.9251667643767025]
 [0.6913262589471468, 0.6617430873242817]
 [0.6350489648571764, 0.49398553161896164]
 [0.6858934716603765, 0.5754076166559308]
 [0.5685973634516744, 0.24417365097485044]
 [0.2596587790868613, 0.24726021863386105]
 [0.20781404013201277, 0.1815201936841122]
 [0.5291724000154314, 0.8723631367774483]
 [0.2784397911414833, 0.32536679104337174]
 [0.3712449951331067, 0.5267198600936198]
 [0.5977567835666606, 0.3542785925102224]
```

In [46]:

```
@save "nice-pi-curve-points-200num.jld2" points
```

In [39]:

```
@load "nice-pi-curve-points-150num.jld2" points
```

Out [39]:

```
1-element Vector{Symbol}:
 :points
```

In [47]:

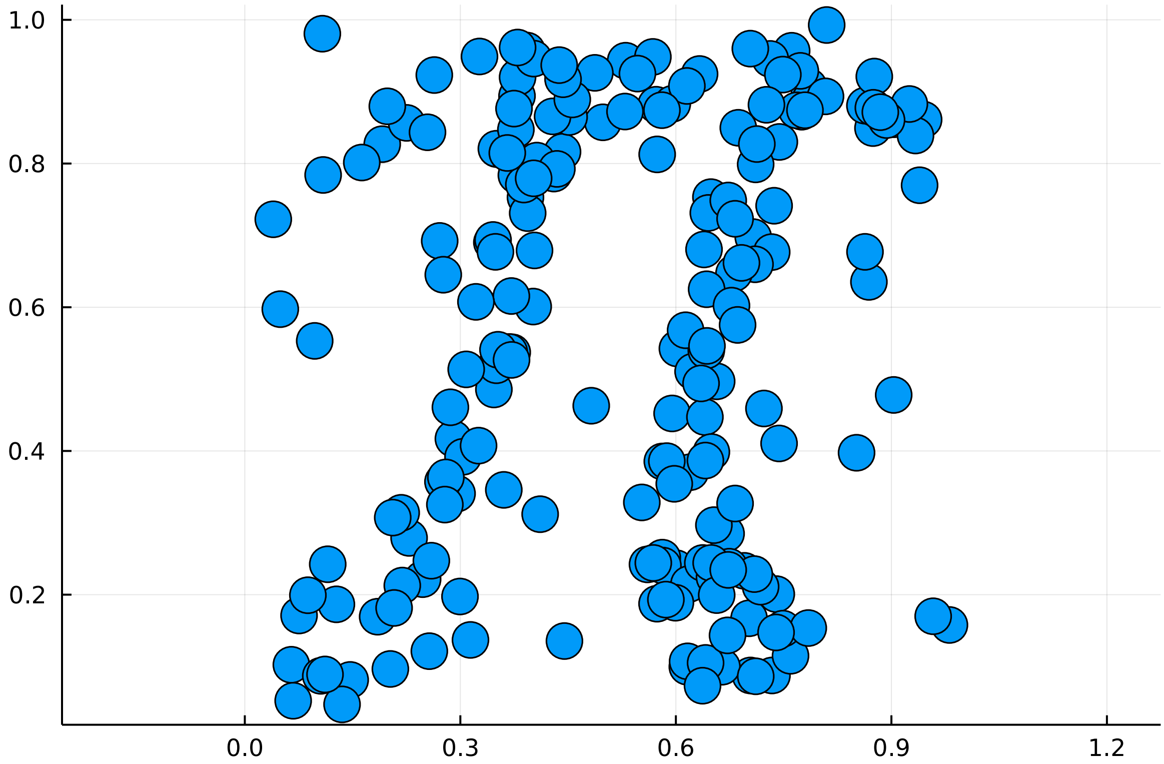
```
numpts = length(points)
```

Out [47]: 200

In [45]:

```
scatter( points, markersize = 10, aspectratio = 1.0, legend = false )
```

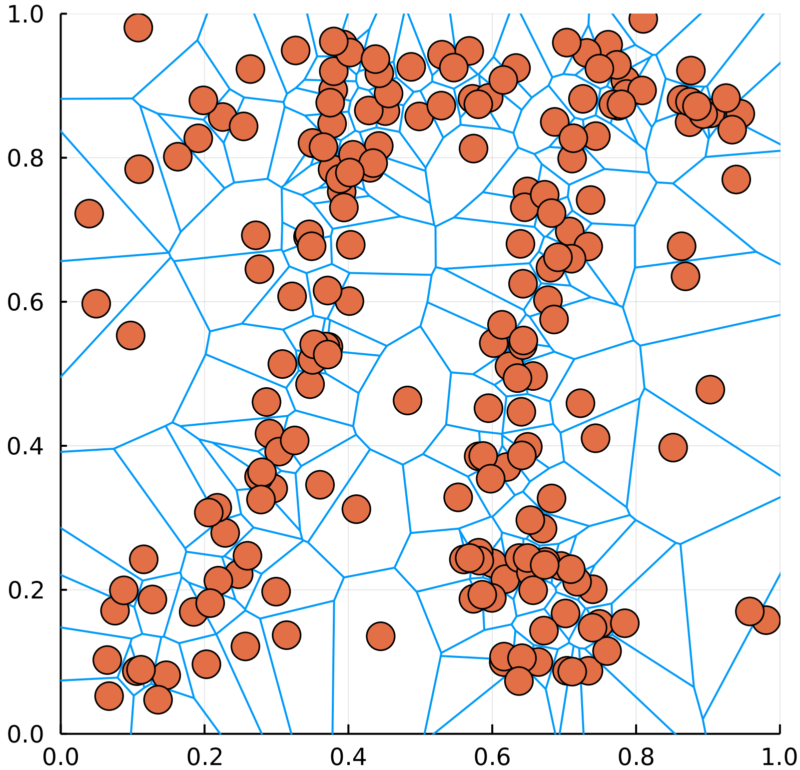
Out [45]:



In [48]:

```
(ox, oy, Connection, mat_l) = make_cx_l(points)
plot(ox, oy, xlims = (0,1.0), ylims = (0,1.0), aspect_ratio = 1.0, legend = false
scatter!(points, markersize = 8, label = "generators", aspectratio = 1.0)
```

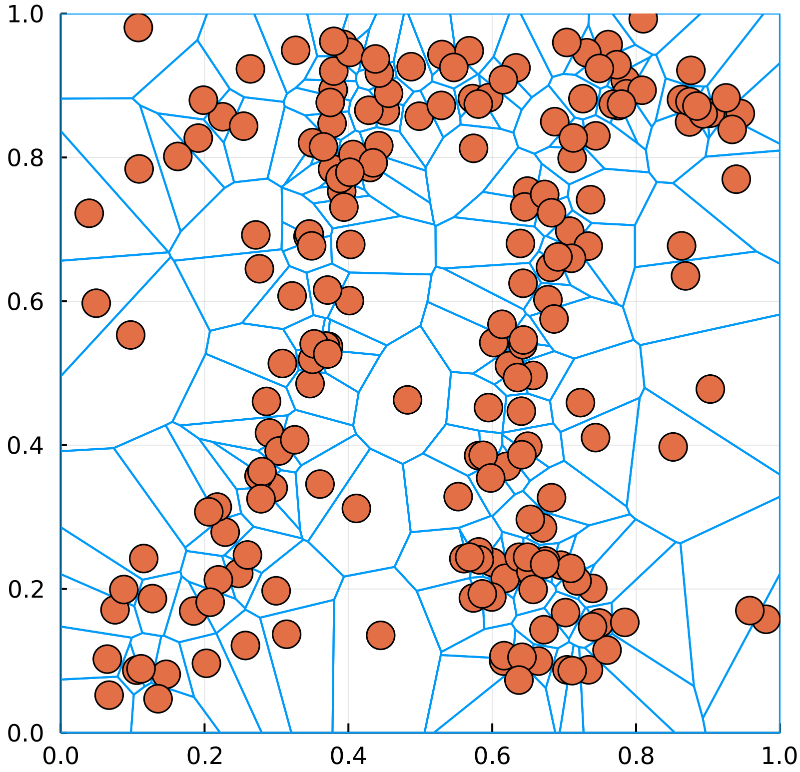
Out [48]:



In [49]:

```
tessc = voronoicells(points, rect);
plot(tessc, aspectratio = 1.0, legend = false)
scatter!(points, markersize = 8, label = "generators", aspectratio = 1.0)
```

Out [49]:



In [50]:

```
function view_v_noanon(pts)
  (ox, oy, Connection, mat_l) = make_cx_l(pts)
  plot(ox, oy, xlims = (0,1.0), ylims = (0,1.0), aspect_ratio = 1.0, legend = fal
  scatter!(pts, markersize = 8, label = "generators", aspectratio = 1.0)
end
```

Out [50]: view_v_noanon (generic function with 1 method)

In [ ]:

```
α = 2

rb = 0.025

Δt = 0.0001
u0 = copy(points)

n_last = 100000
n_skip = div(n_last, 200)

u = u0  # 最初の値はもちろん初期値
u_sq = [ u ] # 初期値を配列に入れておいて…

@showprogress for n in 1:n_last
    u = RK(u)  # Runge-Kutta 法で新しい値をいきなり求める.
    if n % n_skip == 0
        push!(u_sq, u) # その値を配列に追加していく.
    end
end
```

```
[32mProgress:  50%|█████████████████                 |  ETA: 0:22:00[39m
```

In [ ]:

```
view_v_noanon( u_sq[end] )
```

In [53]:
```
dir = "figures-alpha2-ini2"
run(`mkdir $dir`)
```

Out [53]:
```
Process(`[4mmkdir[24m [4mfigures-alpha2-ini2[24m`, ProcessExited(0))
```

In [ ]:
```
@showprogress for n in 1:length(u_sq)
    figure("figures-alpha2-ini2", n)
end
```

In [83]:
```
@save "ch-mor-from-pi-curve-points-150num.jld2" u_sq
```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

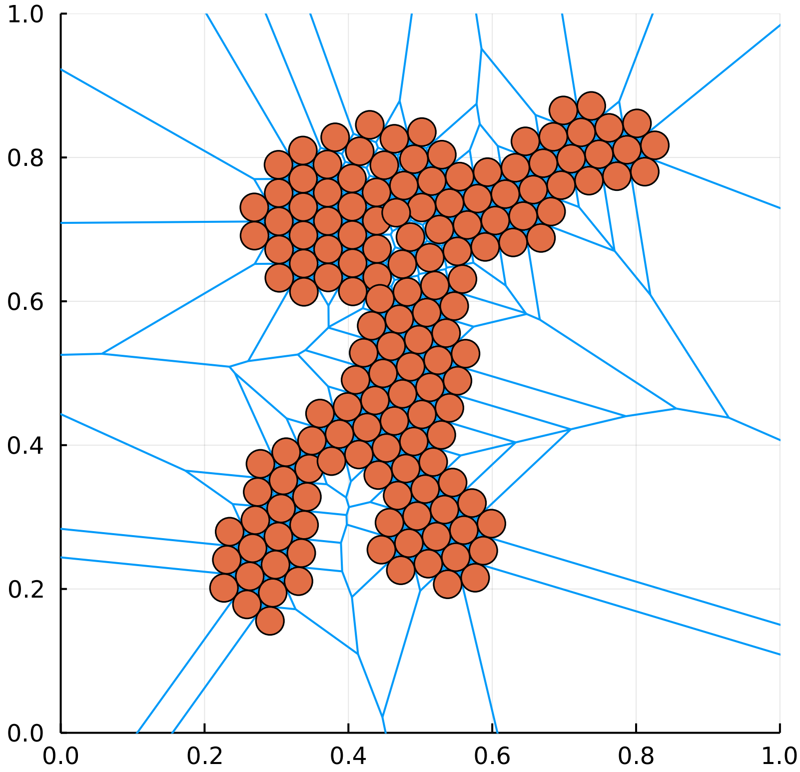
In [ ]:
```

```

In [85]:
```
α = 1

rb = 0.025

Δt = 0.0001
u0 = copy(points)

n_last = 100000
n_skip = div(n_last, 1000)

u = u0  # 最初の値はもちろん初期値
u_sq = [ u ] # 初期値を配列に入れておいて…

@showprogress for n in 1:n_last
    u = RK(u)  # Runge-Kutta 法で新しい値をいきなり求める.
    if n % n_skip == 0
        push!(u_sq, u) # その値を配列に追加していく.
    end
end
```

```
[32mProgress: 100%|█████████████████████████████████| Time: 0:18:36[39m
```

In [86]:
```
view_v_noanon( u_sq[end] )
```

Out [86]:



In [87]:

```
@save "ch-mor-alpha1-from-pi-curve-points-150num.jld2" u_sq
```

In [88]:

```
@showprogress for n in 1:length(u_sq)
    figure("figures-alpha1", n)
end
```

```
[32mProgress: 100%|█████████████████████████████████████████| Time: 0:00:31[39m
```

In [ ]:

In [ ]:

In [ ]:

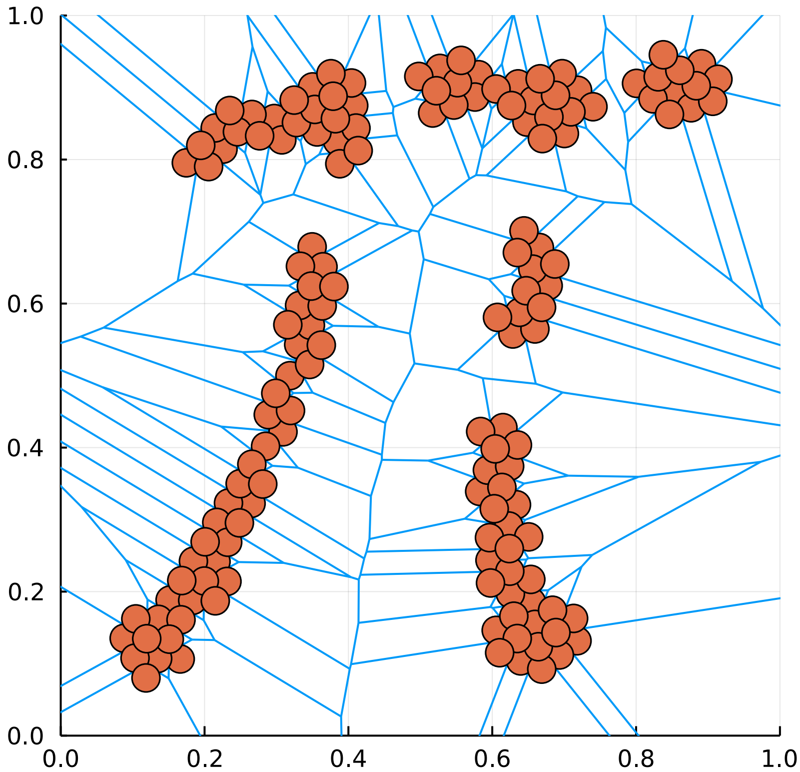
In [ ]:

In [52]:
```
u0 = copy(u_sq[end])

u = u0  # 最初の値はもちろん初期値
# u_sq2 = [ u ] # 初期値を配列に入れておいて…

@showprogress for n in 1:n_last
    u = RK(u)  # Runge-Kutta 法で新しい値をいきなり求める.
    if n % n_skip == 0
        push!(u_sq, u) # その値を配列に追加していく.
    end
end
```

```
[32mProgress: 100%|█████████████████████████████████████████| Time: 0:02:12[39m
```

In [53]:
```
view_v_noanon( u_sq[end] )
```

Out [53]:



In [54]:
```
@save "ch-mor-from-pi-curve-points-150num.jld2" u_sq
```

In [55]:
```
@showprogress for n in 1:length(u_sq)
    figure("figures", n)
end
```

```
[32mProgress: 100%|█████████████████████████████████████████| Time: 0:00:12[39m
```

In [72]:
```
(ox, oy, omatc,omatl) = make_cx_l(u_sq[1]);
```

In [73]:
```
size(omatl)
```

Out [73]: (150, 150)

In [75]:
```
using StatsBase
res = countmap(round.(omatl; digits=1))
```

Out [75]:
```
Dict{Float64, Int64} with 8 entries:
  0.0 => 21942
  0.4 => 16
  0.3 => 18
  0.7 => 4
  0.5 => 8
  0.2 => 46
  0.1 => 464
  0.6 => 2
```

In [ ]:
```
using StatsBase
mat_f = [ mat_l[i,j] > 10*eps() ? 1/(mat_l[i,j])^2 : 0.0 for i in 1:size(mat_l)[1
res = countmap(round.(mat_f; digits=0))
```

In [ ]:
```
?sort
```

In [ ]:
```
sort(collect( keys(res) ))
```

In [ ]:
```
res
```

In [ ]:
```
u0 = copy(u_sq[end])

u = u0  # 最初の値はもちろん初期値
# u_sq3 = [ u ] # 初期値を配列に入れておいて…

@showprogress for n in 1:n_last
    u = RK(u)  # Runge-Kutta 法で新しい値をいきなり求める.
    if n % n_skip == 0
        push!(u_sq, u) # その値を配列に追加していく.
    end
end
```

In [ ]:
```
view_v_noanon( u_sq[end] )
```

In [ ]:
```
u0 = copy(u_sq[end])

u = u0  # 最初の値はもちろん初期値

@showprogress for n in 1:n_last
    u = RK(u)  # Runge-Kutta 法で新しい値をいきなり求める.
    if n % n_skip == 0
        push!(u_sq, u) # その値を配列に追加していく.
    end
end
```

In [ ]:
```
view_v_noanon( u_sq[end] )
```

In [ ]:
```
u0 = copy(u_sq[end])
```

```
u = u0  # 最初の値はもちろん初期値

@showprogress for n in 1:n_last
    u = RK(u)  # Runge-Kutta 法で新しい値をいきなり求める.
    if n % n_skip == 0
        push!(u_sq, u) # その値を配列に追加していく.
    end
end
```

In [ ]:
```
view_v_noanon( u_sq[end] )
```

In [ ]:
```
@save "chv_from_pi_curve_force-is-unlimited.jld2" u_sq
```

少し動画化 of tanh-limit velocity force

In [ ]:
```
@load "chv_from_pi_curve_force-is-limited-by-tanh.jld2" u_sq
```

In [ ]:
```
u_sq
```

In [ ]:
```

```

In [ ]:
```

```