true-particle-dynamics-model-of-Cahn-Hilliard-eq.ipynb

Download this notebook in HTML

In [1]:
```
using VoronoiDelaunay
using VoronoiCells
using GeometryBasics
using LinearAlgebra
using Plots
using ProgressMeter
using JLD2
```

In [2]:
```
import LinearAlgebra: norm
norm( x::Point2D ) = norm( [ x._x, x._y ])

# 距離がとても近い = ほぼ一致する，という判断を下す関数.
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. 上の close関数を使う.
function findpt(pts, pt)
    for i in 1:length(pts)
        if close(pts[i], pt)
            return i
        end
    end
    return 0 # false を返したいが，後で扱いにくいので，見つからない場合は 0 を返す.
end
```

Out [2]:
```
findpt (generic function with 1 method)
```

初期値等

In [3]:
```
m = 0.005 # 粒子の大きさ.

# Cahn-Hilliard eq. のパラメータ.
p = -1.0
q = -0.001
r = 1.0
```

Out [3]:
```
1.0
```

(初期の) 点配置を作成する

In [4]:
```
# 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]

# ball の半径.この ball 内にはせいぜい1点しか入らないようにしたい.
rb = 0.02
```

Out [4]: 0.02

In [5]:
```
function make_pts(num)

    pmin = xmin + 0.15
    pwid = width - 0.3
    randp() = pwid * rand() + pmin

    pts = [ Point2( randp(), randp() ) ]

    for i in 2:num
      isadd = false
      while !(isadd)
          candidate = Point2( randp(), randp() )
          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 [5]: make_pts (generic function with 1 method)

In [6]:
```
points = make_pts(60)
```

Out [6]:
```
60-element Vector{Point2{Float64}}:
 [0.6393563949703913, 0.652354442580869]
 [0.7918059513463728, 0.6812158503584586]
 [0.4926580774738232, 0.7945260289398132]
 [0.7655352844695151, 0.7328820394325856]
 [0.6751600742319119, 0.7806826862192365]
```

```
[0.4225218020636574, 0.529979212857107]
[0.25674546325968894, 0.5133640149588149]
[0.653042818017378, 0.41243852358881194]
[0.47195505172150864, 0.6685803074103629]
[0.8137701676282519, 0.8373687882736839]
[0.7965785368136483, 0.1867135630022065]
[0.6839749479997027, 0.32006011670180895]
[0.2382711049584941, 0.6931227817386684]
⋮
[0.3910743970193288, 0.6655738894227181]
[0.6135597102203856, 0.15286464048193513]
[0.2558745887657953, 0.7438024595429749]
[0.49376239417341217, 0.7463573947398644]
[0.3849657404812351, 0.792363663803519]
[0.5618232499433176, 0.1953045086473449]
[0.3067689421517944, 0.7527887798182265]
[0.6861561944179245, 0.2281652359806733]
[0.1813748187362056, 0.5692652181353474]
[0.21911049024917337, 0.42306726558077346]
[0.3183205531930843, 0.2840126926914137]
[0.767310816401156, 0.45300698608500156]
```

VoronoiDelaunay, VoronoiCells pkg で points に対して様々な Voronoi 情報を計算する.

In [7]:

```
# 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.
# 具体的には, 全体を 4/3 = 1.333... から 5/3 = 1.666... の範囲に収まるように線形変換
# 幅は 5/3 - 4/3 = 1/3 になるので, 元の幅を width とすると, 縮小率は 1/(3 width) と

rate_12sp   = 1.0/(3*width) # これが縮小率.
origin_12sp = 4.0/3         # 12sp の範囲の左端の値.

function limit_to_12sp(x, xmin, xmax)
  return rate_12sp * (x - xmin) + origin_12sp
end

# 逆に戻す.
function expand_from_12sp(x, xmin, xmax)
  return (x - origin_12sp)/rate_12sp + 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 [7]: expand_from_12sp (generic function with 2 methods)

Points をもらって Voronoi の様々な情報を計算する 1stop関数.

In [8]:

```
# それがこれ.
```

```
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.

    veds = voronoiedges(tess)  # Voronoi Edges
    deds = delaunayedges(tess) # Delaunay Edges

    # 一応，お絵かきのために.
    x, y = getplotxy(veds)
    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 deds ]
    # 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] # ただし連結行列で繋がっている場合のみ.
                omatl[i,j] = omatl[j,i] = norm( pts[i] - pts[j] )
            end
        end
    end

    # Voronoi 境界線の長さ. 本来の長さに直している.
    # VoronoiEdges から取り出した edge は、その edge を通じて Voronoi領域が接する格
    # generator として持っている.
    # それらは、getgena(), getgenb() で取り出せる.
    # Voronoi cell の境界面の真の長さ r_{ij}
    omatr = zeros(num, num)
    for edge in veds
      i,j = findpt( pts_12, getgena(edge) ), findpt( pts_12, getgenb(edge) )
      if (i*j > 0) && omatc[i,j]
        # 上で見つけた i or j が Voronoi のテクニカルな仮想点相当の場合(findpt が 0
        # あともちろん，連結行列で繋がっているときのみ.
        omatr[i,j] = omatr[j,i] = norm( geta(edge) - getb(edge) ) / rate_12sp
        # tess は large2small で小さくなっているので、戻す
```

```
        end
    end

    # VoronoiCells の方の機能でもう一度分割することになるが，各 Voronoi Cell の面積
    # しかも，縮小していない状態なので，値を拡大縮小で戻したりする必要がない．
    ov = voronoiarea(voronoicells(pts, rect))

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

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

In [9]:

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

In [10]:

```
omatc
```

Out [10]:

```
60×60 Matrix{Bool}:
 0  0  0  0  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  1  1  0  0  0  0  0  0  0
 0  1  0  0  1  0  0  0  0  1  0  0  0     0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  1  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  1  0  1  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  1  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  1
 0  0  0  0  0  1  0  0  0  0  0  0  0     1  0  0  1  0  0  0  0  0  0  0  0
 0  0  0  1  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  1  0  0  0  0  0  1  0  0  0  0
 0  0  0  0  0  0  0  1  0  0  0  0  0     0  0  0  0  0  0  0  1  0  0  0  1
 0  0  0  0  0  0  0  0  0  0  0  0  0     1  0  1  0  0  0  1  0  0  0  0  0
 ⋮              ⋮              ⋮        ⋱        ⋮              ⋮
 0  0  0  0  0  0  0  0  1  0  0  0  1     0  0  0  1  1  0  1  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  1  0  0     0  0  0  0  0  1  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  1  0  0  0  0  0
 0  0  1  0  0  0  0  0  1  0  0  0  0     1  0  0  0  1  0  0  0  0  0  0  0
 0  0  1  0  0  0  0  0  0  0  0  0  0     1  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  1  0  0  0  0  0  1  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  1     1  0  1  0  1  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  1  1  0  …  0  1  0  0  0  1  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  1  0  0
 0  0  0  0  0  0  1  0  0  0  0  0  0     0  0  0  0  0  0  0  0  1  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  1  0  0  0  1  0     0  0  0  0  0  0  0  0  0  0  0  0
```

In [11]:
```
omatl
```

Out [11]:
```
60×60 Matrix{Float64}:
 0.0  0.0        0.0        0.0        …  0.0        0.0       0.0
 0.0  0.0        0.0        0.0579616     0.0        0.0       0.0
 0.0  0.0        0.0        0.0           0.0        0.0       0.0
 0.0  0.0579616  0.0        0.0           0.0        0.0       0.0
 0.0  0.0        0.0        0.102238      0.0        0.0       0.0
 0.0  0.0        0.0        0.0        …  0.0        0.0       0.0
 0.0  0.0        0.0        0.0           0.0978258  0.0       0.0
 0.0  0.0        0.0        0.0           0.0        0.0       0.121256
 0.0  0.0        0.0        0.0           0.0        0.0       0.0
 0.0  0.0        0.0        0.115083      0.0        0.0       0.0
 0.0  0.0        0.0        0.0        …  0.0        0.0       0.0
 0.0  0.0        0.0        0.0           0.0        0.0       0.156907
 0.0  0.0        0.0        0.0           0.0        0.0       0.0
 ⋮                                     ⋱
 0.0  0.0        0.0        0.0           0.0        0.0       0.0
 0.0  0.0        0.0        0.0           0.0        0.0       0.0
 0.0  0.0        0.0        0.0        …  0.0        0.0       0.0
 0.0  0.0        0.0481813  0.0           0.0        0.0       0.0
 0.0  0.0        0.107714   0.0           0.0        0.0       0.0
 0.0  0.0        0.0        0.0           0.0        0.0       0.0
 0.0  0.0        0.0        0.0           0.0        0.0       0.0
 0.0  0.0        0.0        0.0        …  0.0        0.0       0.0
 0.0  0.0        0.0        0.0           0.150989   0.0       0.0
 0.0  0.0        0.0        0.0           0.0        0.170818  0.0
 0.0  0.0        0.0        0.0           0.170818   0.0       0.0
 0.0  0.0        0.0        0.0           0.0        0.0       0.0
```

In [12]:
```
omatr
```

Out [12]:
```
60×60 Matrix{Float64}:
 0.0  0.0       0.0        0.0        …  0.0        0.0        0.0
 0.0  0.0       0.0        0.066383      0.0        0.0        0.0
 0.0  0.0       0.0        0.0           0.0        0.0        0.0
 0.0  0.066383  0.0        0.0           0.0        0.0        0.0
 0.0  0.0       0.0        0.0795026     0.0        0.0        0.0
 0.0  0.0       0.0        0.0        …  0.0        0.0        0.0
 0.0  0.0       0.0        0.0           0.0222152  0.0        0.0
 0.0  0.0       0.0        0.0           0.0        0.0        0.119029
 0.0  0.0       0.0        0.0           0.0        0.0        0.0
 0.0  0.0       0.0        0.0499894     0.0        0.0        0.0
 0.0  0.0       0.0        0.0        …  0.0        0.0        0.0
 0.0  0.0       0.0        0.0           0.0        0.0        0.0271711
 0.0  0.0       0.0        0.0           0.0        0.0        0.0
 ⋮                                    ⋱
 0.0  0.0       0.0        0.0           0.0        0.0        0.0
 0.0  0.0       0.0        0.0           0.0        0.0        0.0
 0.0  0.0       0.0        0.0        …  0.0        0.0        0.0
 0.0  0.0       0.0798126  0.0           0.0        0.0        0.0
 0.0  0.0       0.734012   0.0           0.0        0.0        0.0
 0.0  0.0       0.0        0.0           0.0        0.0        0.0
 0.0  0.0       0.0        0.0           0.0        0.0        0.0
 0.0  0.0       0.0        0.0        …  0.0        0.0        0.0
 0.0  0.0       0.0        0.0           0.0180474  0.0        0.0
 0.0  0.0       0.0        0.0           0.0        0.0824084  0.0
 0.0  0.0       0.0        0.0           0.0824084  0.0        0.0
 0.0  0.0       0.0        0.0           0.0        0.0        0.0
```

In [13]:
```
ov
```

Out [13]:
```
60-element Vector{Float64}:
 0.004933657961751669
 0.005409549629229002
 0.016520941247671277
 0.007340133224112033
 0.0144478353527281
 0.013605819146154836
 0.00809569021726865
 0.011973526188179442
 0.011543677616035516
 0.052369355493841016
 0.055190573728010714
 0.008609759709967026
 0.005801482050128377
 ⋮
 0.009898085314043546
 0.03492134095424768
 0.004649093246856865
```

```
0.007390886447420045
0.028116450469901588
0.008878597662679392
0.008503935546570949
0.010787725049078978
0.030054945410406098
0.008879375879362909
0.04013229416208567
0.011729981949758696
```

In [14]:

```julia
m ./ ov
```

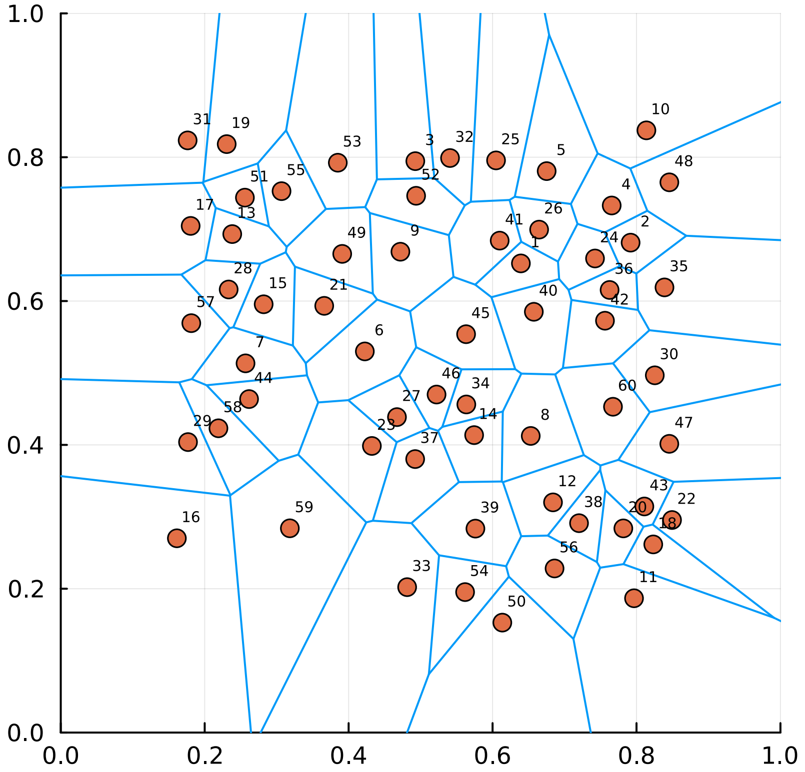
Out [14]:

```
60-element Vector{Float64}:
 1.0134468256135
 0.9242913629970019
 0.3026461946110231
 0.6811865462571179
 0.34607260381437555
 0.3674898178705439
 0.6176125649341998
 0.41758792868688277
 0.4331375291574685
 0.09547568330467679
 0.09059518070315628
 0.5807363002490983
 0.861848740855685
 ⋮
 0.505148202037209
 0.1431789233566594
 1.0754785362458312
 0.6765088376841946
 0.1778318356846806
 0.5631519965159785
 0.5879630639976047
 0.4634897512912605
 0.16636197243828035
 0.5631026400876665
 0.12458794356001876
 0.4262581154357921
```

In [15]:

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

Out [15]:



点の位置から常微分方程式右辺を計算する

In [16]:
```
function rhs(pts, mat_c, mat_l, mat_r, vol )

    num = length(pts)

    u = m ./ vol  # 各 cell での密度.

    # phi = delta G/ delta u
    tilu = 2.0 .* u .- 1.0
    phi = similar(u)
    for i in 1:num
        if abs(tilu[i]^2) < (-p/r)
            phi[i] = p * tilu[i] + r * (tilu[i]^3)
        else
            phi[i] = u[i] > 0.5 ? 100.0 : -100.0  # 本来はここは sign * ∞.
        end
    end

    mat_B = zeros(num, num)
    for i in 1:num
        for j in i+1:num
            if mat_c[i,j]
                mat_B[i,j] = mat_B[j,i] = mat_r[i,j]/mat_l[i,j]
            end
        end
    end
    lapl_u = similar(u)
    one_v = ones(num)
    for i in 1:num
        lapl_u[i] = dot( mat_B[i, :], u - u[i] .* one_v ) / vol[i]
    end
    lapl_u = (2*q) .* lapl_u

    phi = phi + lapl_u

    #
    v = [ [0.0, 0.0] for i in 1:num ]
    for i in 1:num
        for j in 1:num
            if mat_c[i,j]
                v[i] += ( phi[j] * mat_r[i,j] / mat_l[i,j] ).* (pts[j] - pts[i])
            end
        end
    end
    v = (-1/(2*m)) .* v

    return v

end
```

Out [16]: rhs (generic function with 1 method)

In [17]:
```
rhs( points, omatc, omatl, omatr, ov)
```

Out [17]:
```
60-element Vector{Vector{Float64}}:
 [-245.48884372939602, -14.707803184370338]
 [569.616890781495, 443.5582208934571 3]
 [-13.939748696889016, -3.2672621226395866]
 [117.28017730458488, 376.9430792851558]
 [5.213367755854972, -3.5638295783363567]
 [-2.433167932586879, 4.077182359403438]
 [3.083229251105293, 1.7965206203703608]
 [-2.5296973643971854, -3.7200128717937666]
 [1.854980272848292, -0.150067628533173]
 [43.53485389339326, 42.44549924874684]
 [72.57368201910704, -342.045447287739]
 [-915.7098744345566, 732.4539333621323]
 [-254.93733122135995, -744.5723146410245]
 ⋮
 [-2.7000937988941307, -1.7047126471819458]
 [12.893239146297844, -16.50586796108903]
 [4.103542626166358, -3.7972809354477928]
 [1.0965045544112522, -5.711840379282376]
 [4.704033762602654, -7.39879982178128]
 [-0.34147910176961865, 1.8001302409432047]
 [884.0580897847929, 152.4645639132362]
 [-447.3844982815, -727.9160193612598]
 [-0.2528152006543918, 3.190068209999934]
 [4.952071977652846, 4.8247037775965795]
 [-1.5637397217559648, 1.4306023294306347]
 [-5.4614134784516795, 2.683569836257812]
```

時間発展は Runge-Kutta 法で.

In [18]:
```
function rhs(points)
    (ox, oy, omatc, omatl, omatr, ov) = make_cx_l(points)
    return rhs( points, omatc, omatl, omatr, ov)
end
```

Out [18]:
```
rhs (generic function with 2 methods)
```

In [19]:
```
function RK(u)
    r1 = rhs(u)
    r2 = rhs(u + Δt/2 * r1)
    r3 = rhs(u + Δt/2 * r2)
    r4 = rhs(u + Δt * r3)
    return u + Δt * (r1 + 2*r2 + 2*r3 + r4)/6
end
```

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

In [20]:
```
Δt = 0.00000005
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 1:30000
    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: 100%|█████████████████████████████████████████| Time: 0:04:19[39mmmm9m
```

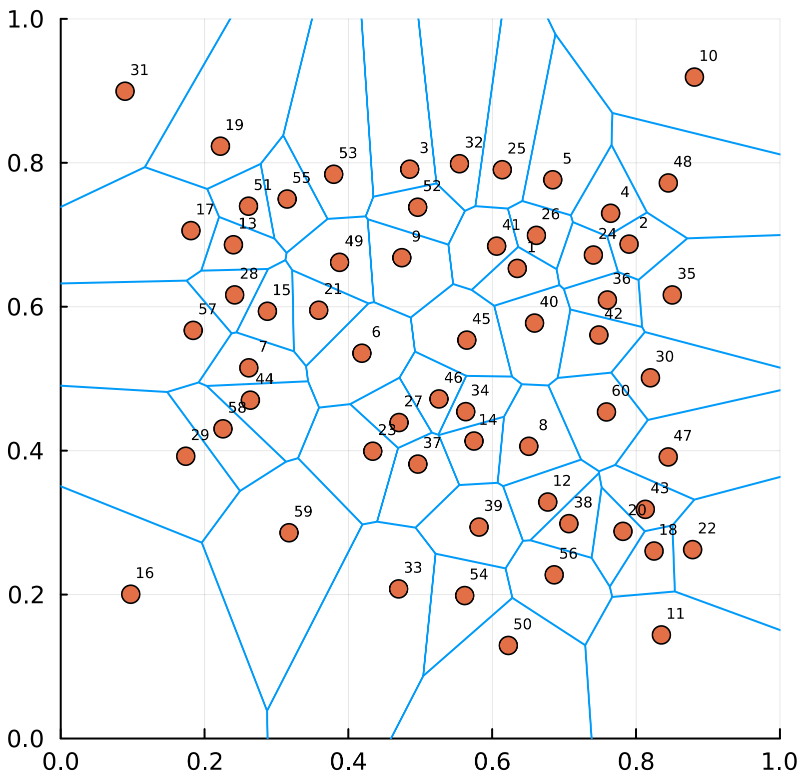
In [21]:
```
u_sq[end]
```

Out [21]: 60-element Vector{Point2{Float64}}:
```
 [0.6347572647300732, 0.653219436909223]
 [0.7901954558905152, 0.686989565794952]
 [0.48524425615188044, 0.7911061203819574]
 [0.7645945609139676, 0.729848598591848]
 [0.6841042455169657, 0.7764077314674231]
 [0.41876108688025954, 0.5353911449488652]
 [0.2615559094112178, 0.5151126655425453]
 [0.6508222139762806, 0.4060643954588918]
 [0.4742848121484637, 0.6681327583484503]
 [0.881088336409271, 0.9190977028053389]
 [0.8351535146907376, 0.14393000909299472]
 [0.6771741476898018, 0.32861447161016094]
 [0.24012112719206583, 0.6860892267445078]
 ⋮
 [0.38772822611053737, 0.6616900643806227]
 [0.6223363168924433, 0.12931299116738135]
 [0.2610918873033611, 0.7397169938474597]
 [0.4963388220726547, 0.7382581075984584]
 [0.37960268673387904, 0.7840626385546914]
 [0.5616709759615586, 0.19861870496310136]
 [0.3148326106126286, 0.7496615773123936]
 [0.6861085274144135, 0.227467242632074]
 [0.18420237332284228, 0.5671386804358902]
 [0.22562431293020593, 0.43033320836991806]
 [0.3174246247374079, 0.2859242018442965]
 [0.7587945114555679, 0.45377345039499334]
```

In [23]:
```
(ox, oy, omatc, omatl, omatr, ov) = make_cx_l(u_sq[end])

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

Out [23]:

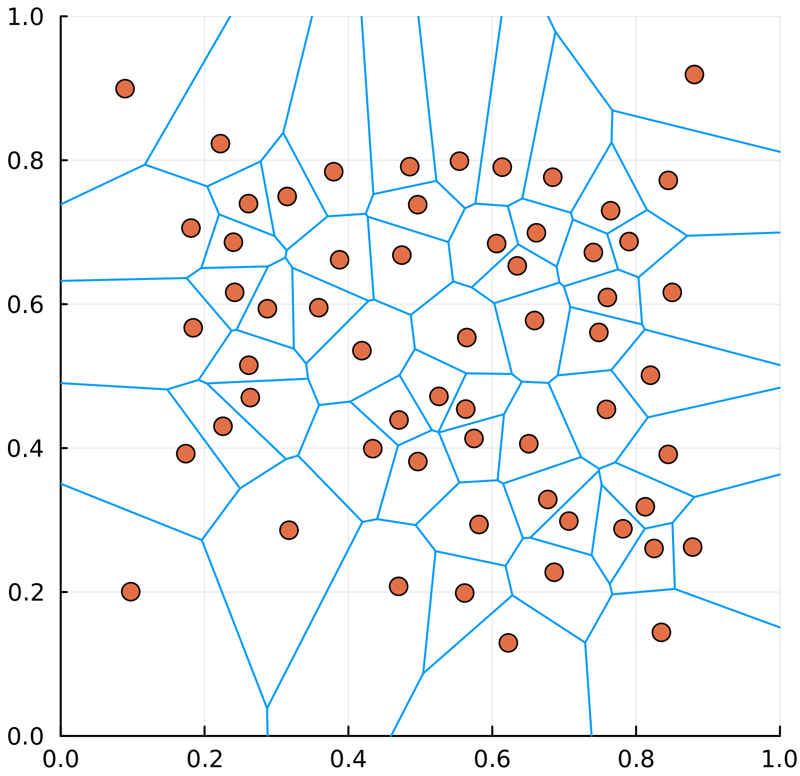


In [24]:
```
function view_v_noanon(pts)
  (ox, oy, omatc, omatl, omatr, ov) = make_cx_l(pts)
  plot(ox, oy, xlims = (0,1.0), ylims = (0,1.0), aspect_ratio = 1.0, legend = fal
  scatter!(pts, markersize = 5, label = "generators", aspectratio = 1.0)
end
```

Out [24]: view_v_noanon (generic function with 1 method)

In [25]:
```
view_v_noanon( u_sq[end] )
```

Out [25]:



In [26]:
```
using Printf # すぐ下の @sprintf を使いたいので.

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

    (ox, oy, omatc, omatl, omatr, ov) = 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 = 5, 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
```

Out [26]: figure (generic function with 1 method)

In [27]:
```
dir = "true-pd-60pts"
run(`cmd /k mkdir $dir`)
```

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

Out [27]: Process(`[4mcmd[24m [4m/k[24m [4mmkdir[24m [4mtrue-pd-60pts[24m`, ProcessExited(0))

In [28]:

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

```
[32mProgress: 100%|█████████████████████████████████████████| Time: 0:00:07[39m
```

Hey there! If you have any feedback for this tool - issues, ideas for improvement, or you want to just tell me about your use case for this, I'd love to know. [E-mail me]() or [tweet at me]().