Les configurations de points et Mathematica

Avant de commencer : dans ce notebook et les notebook à venir, les programmes sont parfois plus longs et on intercalle des commentaires pour mieux en suivre le déroulement. Et nous estimons que le lecteur a une connaissance satisfaisante des méthodes qui sont programmées.

Tout espace géographique est composé de points, de réseaux et de champs. Le tout en interaction. Les points correspondent à des localisations, les arbres dans une forêt, les médecins ou toute autre activité dans une ville, les villes dans un État ou un ensemble d'États, les stations de sports d'hiver dans les Alpes, ... Ces configurations de points sont décrites par de nombreux paramètres qui permettent d'en apprécier la localisation, la densité, l'homogénéité, l'organisation.

Puis, le géographe cherche à connaître si cette configuration est aléatoire dans l'espace ou si un processus stochastique est à l'origine de cette répartition.

Enfin, la plupart du temps, des valeurs sont attribuées à ces points, par exemple un diamètre à chaque arbre, la température journalière dans les stations météorologiques, le nombre d'habitants par ville ou toute autre variable. De nouveaux questionnements surgissent alors. Deux sont essentiels. D'abord, existe-t-il une dépendance entre ces valeurs, donc une auto-corrélation spatiale? Puis, il convient souvent de passer de cette structure de points à un champ, et donc de procéder à une interpolation.

La version 12.2, livrée en décembre 2020, fournit un grand nombre de nouvelles fonctions expérimentales pour étudier les configurations de points.

Décrire des configurations de points géographiques

Comme dans le cas des séries temporelles, la première étape consiste à construire une configuration, puis à visualiser les données.

Créer une configuration de points et la visualiser avec GeoListPlot[]
Après avoir effacé ce qui est en mémoire, nous entrons le nom de l'État qui va être l'objet des traitements. Il suffit de changer ce nom pour obtenir des résultats sur un autre État. La première instruction reg = donne une identité, tandis que la deuxième reg2= donne une fonction. Puis, dans ny est stocké le nombre de villes à retenir, les 75 plus grandes par leur population dans les exercices qui vont suivre. Nous tapons 75 dans la fenêtre qui apparaît sur l'écran. L'instruction suivante donne en résultat l'identité de chacune des 75 villes. Voici une représentation directe des 75 grandes villes françaises récupérées dans le fichier villes. Les deux instructions suivantes donnent les coordonnées des 75 villes, en coordonnées Wolfram, puis en coordonnées mercatore. La fonction **SpatialPointData**[] créé la configuration et associe les positions des points et leur cadre géographique, la France dans cet exemple. Il est alors possible d'afficher toutes les propriétés de cette fonction. Enfin, la fonction **GeoListPlot**[] créé la carte de localisation des 75 villes.

In[187]:=
```
ClearAll["Global`*"]
efface tout
pays = "France";
reg = Entity["Country", pays]
      entité
reg2 = CountryData[pays, {"Polygon", "Mercator"}]
       données de pays      polygone
ny = Input["Choisir le nombre de villes"] // ToExpression;
     entrée                                 en expression
ville = CityData[{All, pays}][[ ;; ny]]
        données de v· tout
coord = Take[Table[QuantityMagnitude[CityData[c, "Coordinates"]],
        prends table magnitude de quantité données de villes
    {c, CityData[{All, pays}]}], ny];
        données de v· tout
coordmercator = Table[First@GeoGridPosition[
                table premier position de grille géographique
     GeoPosition[CityData[ville[[i]]]["Coordinates"]], "Mercator"], {i, 1, ny}];
     position géogra··· données de villes
data = SpatialPointData[ville, reg]
       données de points spatiaux
data["Properties"]
     propriétés
data["Summary"]
GeoListPlot[data]
tracé de cartographie de lieux
```

Out[189]= France

Out[190]= Polygon[ Number of points: 2627 Embedding dimension: 2]

Out[192]= {Paris, Marseille, Lyon, Toulouse, Nice, Nantes, Strasbourg, Montpellier, Bordeaux, Lille, Rennes, Le Havre, Reims, Saint-Étienne, Toulon, Grenoble, Angers, Dijon, Brest, Le Mans, Nîmes, Aix-en-Provence, Limoges, Clermont-Ferrand, Tours, Amiens, Villeurbanne, Metz, Besançon, Perpignan, Orléans, Rouen, Mulhouse, Caen, Boulogne-Billancourt, Nancy, Montreuil, Argenteuil, Roubaix, Saint-Denis, Tourcoing, Avignon, Nanterre, Poitiers, Versailles, Asnières-sur-Seine, Courbevoie, Créteil, Colombes, Pau, Vitry-sur-Seine, Aulnay-sous-Bois, La Rochelle, Champigny-sur-Marne, Rueil-Malmaison, Antibes, Saint-Maur-des-Fossés, Calais, Béziers, Dunkirk, Aubervilliers, Mérignac, Cannes, Bourges, Saint-Nazaire, Colmar, Ajaccio, Valence, Quimper, Drancy, Villeneuve-d'Ascq, Noisy-le-Grand, Évreux, Neuilly-sur-Seine, Levallois-Perret}

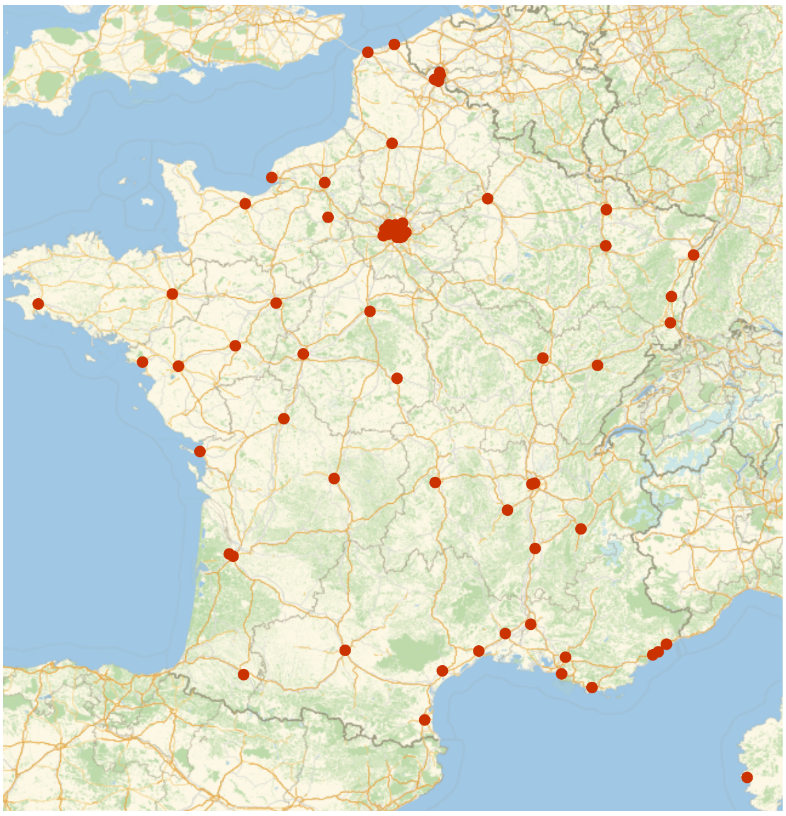
Out[195]= `SpatialPointData[`  Data Type: Geographical
Number of Points: 74
Dimension: 2 `]`

Out[196]=
```
{AnnotationProperties, Annotations, AnnotationsList, Configuration,
 ConfigurationCount, DeveloperProperties, Dimension, FryPlot, MeanPointCount,
 MeanPointDensity, ObservationRegion, PointCount, PointCountList,
 Points, PointsList, Properties, RegionMeasure, Summary, MetaInformation,
 {Configuration, <configuration>}, {Configuration, {<configurations>}}}
```

Out[197]=

Data Type	Geographical
Configuration Count	1
Point Count(s)	74
Dimension	2
Annotations	None

Out[198]=



Vous noterez un bug mineur : en tapant le nombre 75 vous n’obtenez que 74 villes. L’option *Summary* donne bien seulement 74 villes. En outre, quand le nombre de villes est faible, en utilisant la fonction **GeoListPlot**[] le fond de carte est réduit. la Bretagne peut ne pas apparaître. Le fond s’adapte aux coordonnées des villes. Il serait possible d’utiliser la fonction **GeoGraphics**[], comme le montre l’exemple ci-dessous.

De nombreux indicateurs pour analyser la densité d’une configuration de points

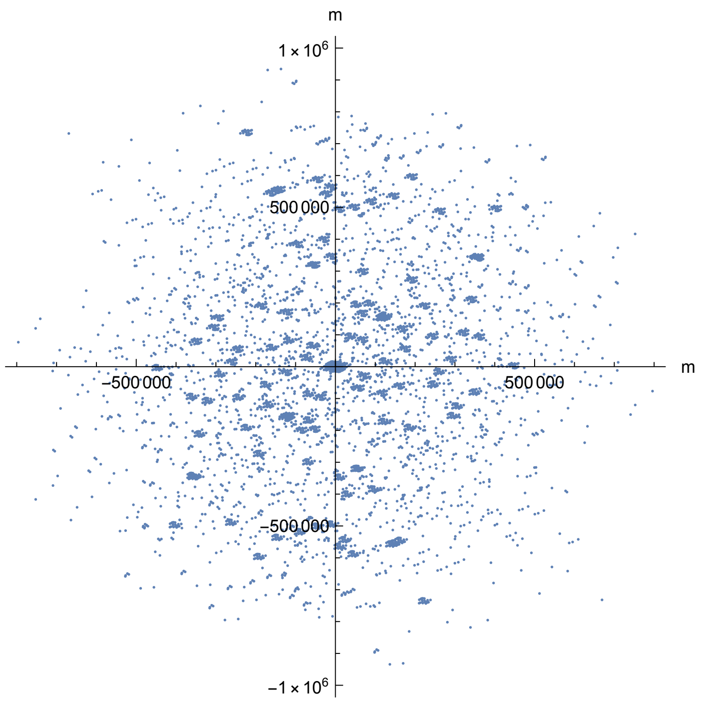
Outre les propriétés du fichier *data*, par exemple le graphique de Fry, pour décrire une configuration de points, le chercheur dispose de nombreuses fonctions. Les premières résument l'ensemble des localisations. Citons les fonctions **SpatialMedian**[] et **CentralFeature**[]. Cette dernière fonction donne le point dont la somme des distances est minimum avec tous les autres points. Ces deux points sont déterminé, puis cartographiés avec les villes.

In[199]:=

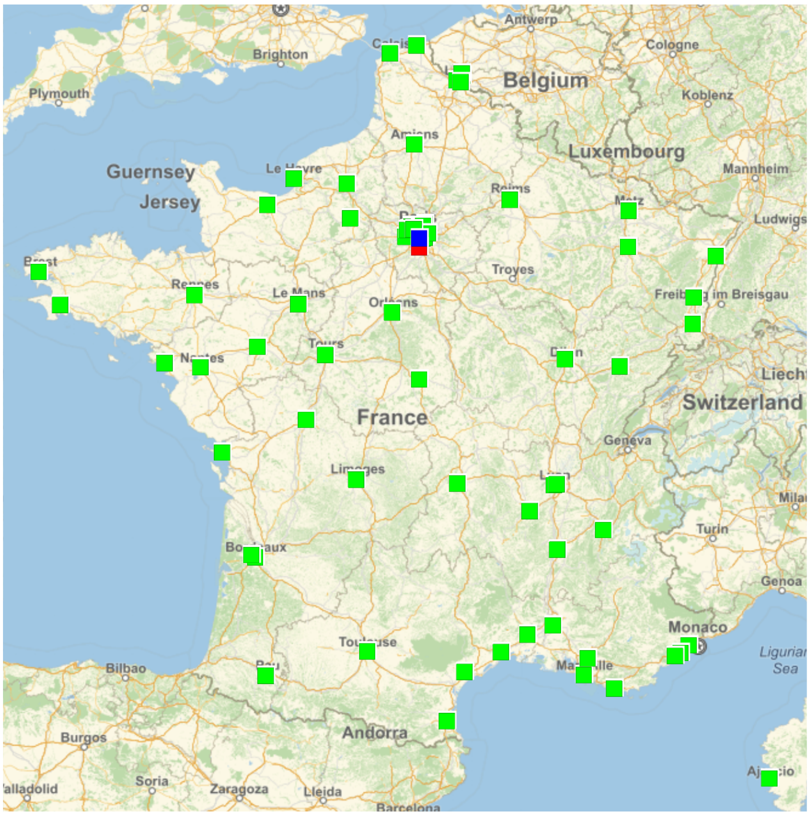
```
Print["Graphique de Fry"]
imprime
data["FryPlot"]
(*Calcul de la médiane spatiale*)
sm = SpatialMedian[data];
     médiane spatiale
(*Calcul du centre par rapport a tous les points*)
cf = CentralFeature[data];
     caractéristique centrale
GeoGraphics[{GeoMarker[ville, Green], GeoMarker[sm, Red], GeoMarker[cf, Blue]}]
carte géographique marqueur géographique vert marqueur géograp·· rouge marqueur géograp·· bleu
```

```
Graphique de Fry
```

Out[200]=



Out[203]=

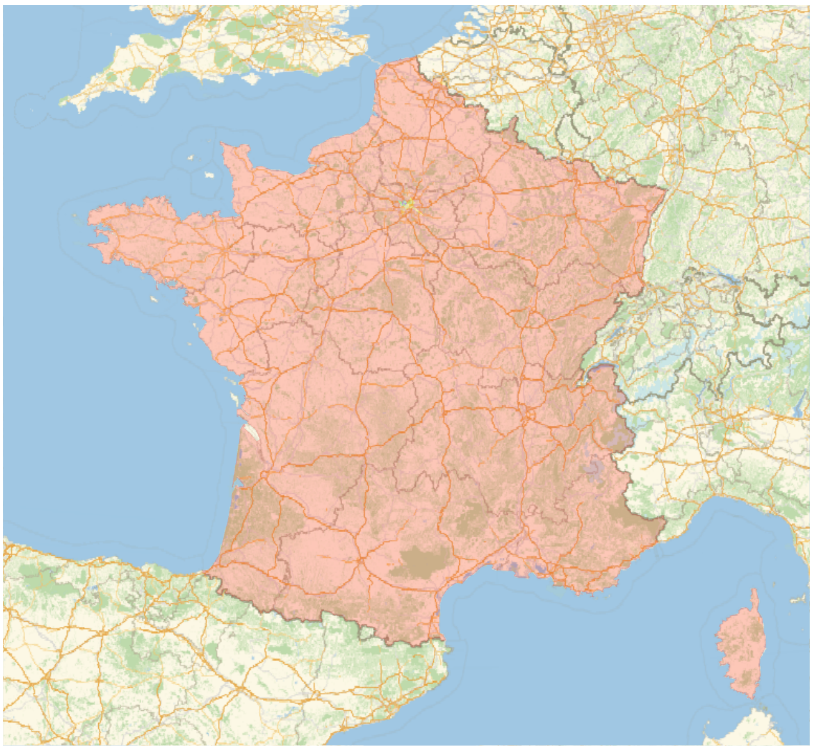


Remarquez que les deux indicateurs de centralité se chevauchent. Leurs valeurs sont très proches.

Mais le géographe s'intéresse aussi aux indicateurs de densité avec les fonctions **MeanPointDensity[], PointDensity[] et HistogramPointDensity[]**. Une première approche consiste à utiliser directement la fonction **PointDensity[]**, puis à cartographier le résultat :

In[204]:=
```
densite1 = PointDensity[data, "Voronoi"];
                 densité de point
Show[densite1["DensityVisualization"]]
montre
```

Out[205]=

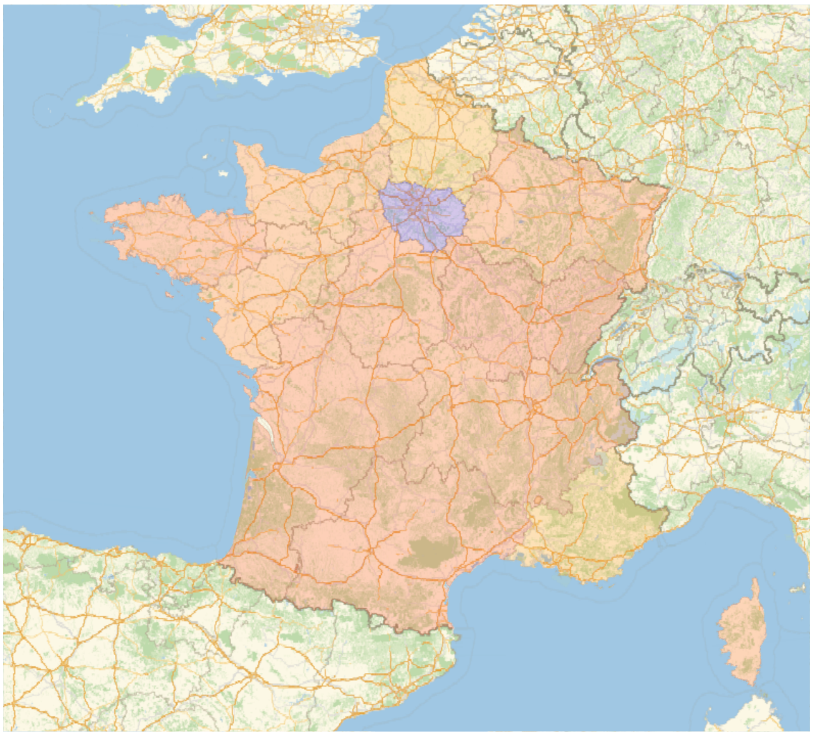


Pour les analyses de densité, il est préférable de représenter l'histogramme des densités de villes par région, ce qui demande une préparation pour obtenir le tracé des régions.

```
Print["Carte régionale des histogrammes de densité"]
imprime
(*divisions regionales de la France*)
divisions = EntityValue[Entity["AdministrativeDivision",
            valeur d'entité   entité
     {EntityProperty["AdministrativeDivision", "ParentRegion"] ->
      propriété d'entité
    Entity["Country", "France"]}], "Entities"];
    entité
pols = Table[div["Polygon"], {div, divisions}];
       table         polygone
(*calcul des histogrammes par region et afficahe du résultat*)
histo = HistogramPointDensity[data, pols];
        densité de points d'histogramme
Show[histo["DensityVisualization"]]
montre
```

Carte régionale des histogrammes de densité

Out[ ]=



Les résultats du calcul des densité peuvent aussi être représenté avec les fonctions **GeoHistogram**[] ou **GeoSmoothHistogram**[].

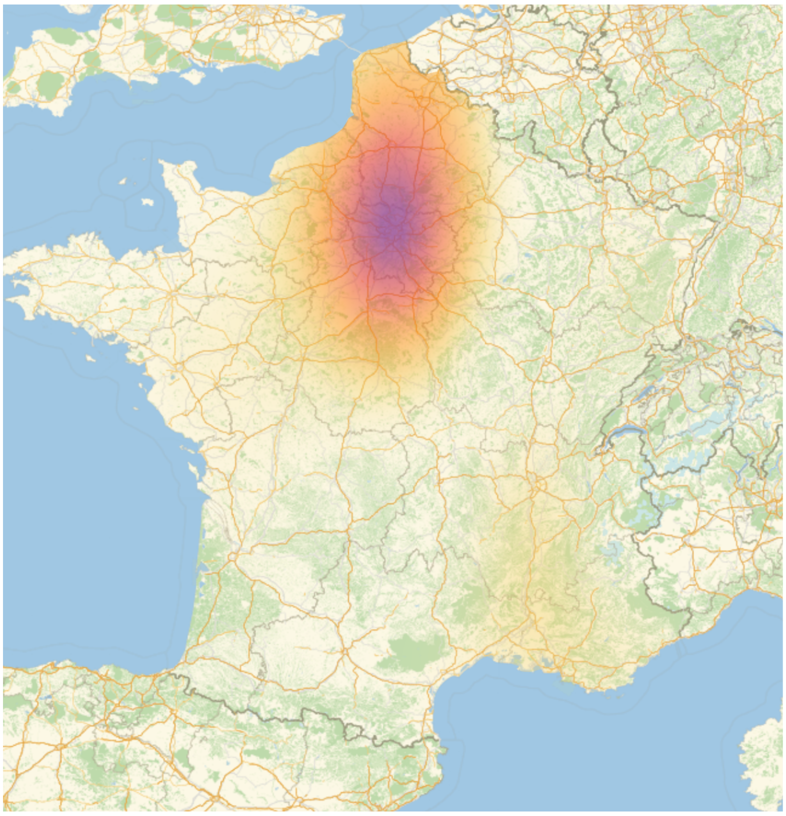
In[206]:=
```
GeoHistogram[data["Points"]]
GeoSmoothHistogram[data["Points"], RegionFunction → reg]
```
histogramme géographique

histogramme lisse géographique

fonction de région

Out[206]=

Out[207]=



La région parisienne est prépondérante. Seul le couloir rhodanien et ses prolongements méditerranéens se perçoivent. Bien d’autres représentations cartographiques s’obtiennent en se servant d’options.

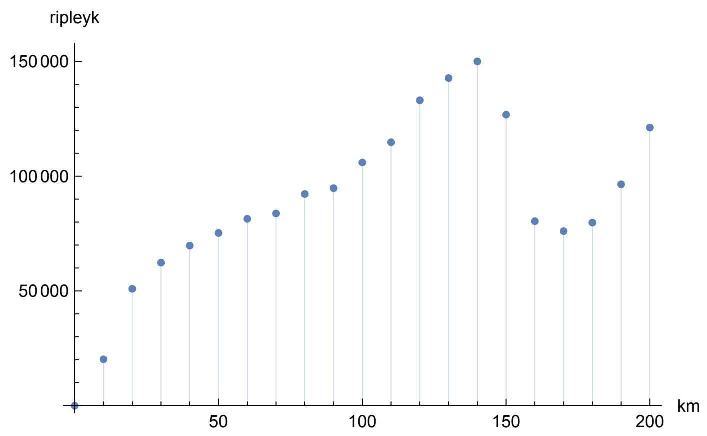
Des fonctions pour analyser l'homogénéité d'une configuration de points

Enfin, il est utile de déterminer pour cet ensemble de points les différentes fonctions qui analysent l'homogénéité spatiale des configurations de ces grandes villes. Pratiquement toutes ces fonctions mettent en jeu les distances entre les points. Ces fonctions permettent de repérer les agrégats de points, et cela à différents échelons spatiaux. Ce sont des approches multi-échelles pour les points. Il serait même très facile de déterminer d'autres indicateurs, par exemple ceux de Diggle. Débutons avec les fonctions **KRipley**[] et **BesagL**[].

In[208]:=
```
Print["Calcul et illustration de la fonction de Ripley"]
imprime
ripley = RipleyK[data, Range[0, 200, 10]];
         K de Ripley   plage
ListPlot[ripley, DataRange → MinMax[Range[0, 200, 10]],
tracé de liste   plage de donn··· minimu··· plage
 AxesLabel → {"km", "ripleyk"}, Filling → Axis]
 étiquette des axes             remplissage axe
Print["Calcul et illustration de la fonction de Besag"]
imprime
bes = BesagL[data];
      L de Besag
ListPlot[
tracé de liste
 Table[{r, bes[r]}, {r, Quantity[0, "km"], bes["MaxRadius"], Quantity[.1, "km"]}],
 table                  quantité                             quantité
 AxesLabel → {"km", "besag"}]
 étiquette des axes
```

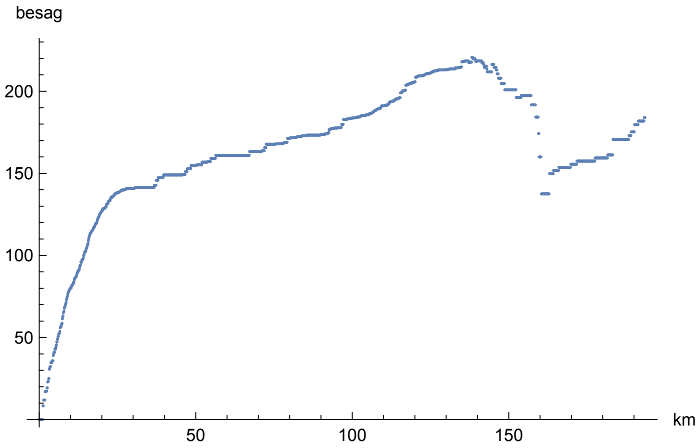
```
Calcul et illustration de la fonction de Ripley
```

Out[210]=




```
Calcul et illustration de la fonction de Besag
```

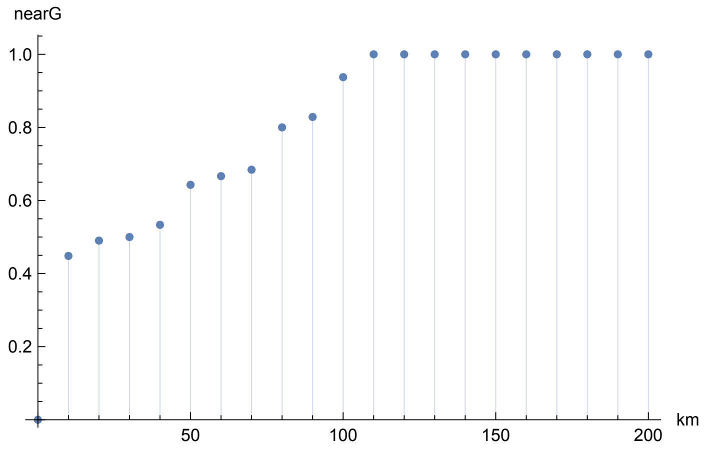
Out[213]=



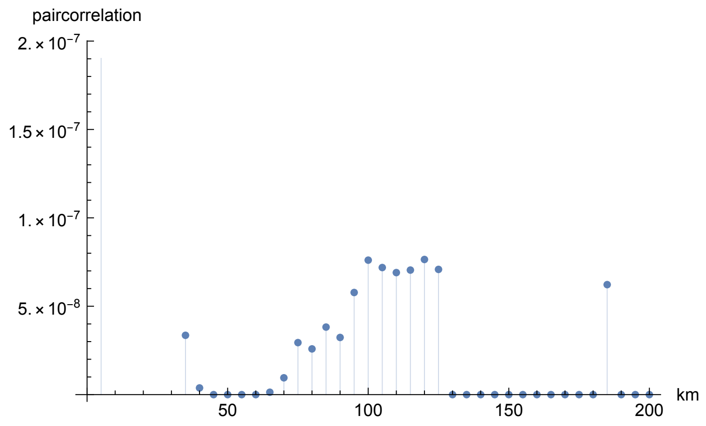
Ces deux traitements montrent que la répartition des villes n’est pas uniforme. Elle n’obéit pas un processus de Poisson. D’autres fonctions, construites sur des principes similaires, se déterminent avec les fonctions **NearestNeighbordG**[] et **PairCorrelationG**[].

```
In[]:= Print["Calcul et illustration de la fonction G du plus proche voisin"]
       ⌊imprime
       nng = NearestNeighborG[data, Range[0, 200, 10]];
             ⌊voisin le plus proche de G  ⌊plage
       ListPlot[nng, DataRange → MinMax[Range[0, 200, 10]],
       ⌊tracé de liste ⌊plage de donn⋯ ⌊minimu⋯ ⌊plage
        AxesLabel → {"km", "nearG"}, Filling → Axis]
        ⌊étiquette des axes          ⌊remplissage ⌊axe
       Print["Calcul et illustration de la fonction de corrélation par aire"]
       ⌊imprime
       paircor = PairCorrelationG[data, Range[0, 200, 5]];
                 ⌊corrélation de paire de G  ⌊plage
       ListPlot[paircor, DataRange → MinMax[Range[0, 200, 5]],
       ⌊tracé de liste   ⌊plage de donn⋯ ⌊minimu⋯ ⌊plage
        Filling → Axis, AxesLabel → {"km", "paircorrelation"}]
        ⌊remplissage ⌊axe  ⌊étiquette des axes
```

```
Calcul et illustration de la fonction G du plus proche voisin
```

Out[]=



```
Calcul et illustration de la fonction de corrélation par aire
```

Out[]=


Avec diverses options, il est possible d’obtenir les valeurs corrigées par divers effets de bord. Pratiquement tous ces résultats indiquent que la configuration des 75 plus grandes villes françaises n’est pas aléatoire. Il est alors nécessaire de s’interroger sur les processus stochastiques qui peuvent engendrer une telle configuration.

Rechercher le processus stochastique correspondant à une configuration de points géographiques

Pour déterminer le processus stochastique, plusieurs fonctions sont utilisables. Mais, toutes ces fonctions sont encore expérimentales et imparfaites. Elles fonctionnent relativement bien quand la fonction **SpatialPointData**[] ne prend pas en compte un polygone représentatif d'un pays, mais retient uniquement comme région, un polygone de Ripley-Rasson calculé automatiquement autour des points. Nous allons donc d'abord constituer une nouvelle configuration avec ces nouvelles limites. Il suffit de ne pas inclure la région *reg* sans la fonction **SpatialPointData**[] :

In[ ]:=
```
data1 = SpatialPointData[ville]
```
données de points spatiaux

Out[ ]=
```
SpatialPointData[
```

Data Type: Geographical
Number of Points: 200
Dimension: 2
```
]
```

La première étape, comme pour les séries temporelles, consiste à tester si la répartition des points, des villes dans ce notebook, est aléatoire.

Tester le caractère aléatoire de la configuration

Le processus de Poisson homogène est l'illustration du caractère uniforme d'une répartition de points dans un espace. Ce test avec la fonction **SpatialRandomessTest**[] compare donc la répartition des points avec une répartition de Poisson.

In[ ]:=
```
test = SpatialRandomnessTest[data1, "TestConclusion"]
```
test de structure spatiale aléatoire

Out[ ]=
```
The null hypothesis that the data exhibits complete spatial randomness
 is rejected at the 5 percent level based on the ModifiedChiSquare test.
```

Un test plus général : le test R

Par ailleurs, il est relativement facile de programmer le test R, qui donne de nouvelles indications sur le type de configuration (aléatoire, régulier ou groupé). Ce test R, égal à 0,001, est le signe d'une configuration concentrée, avec des villes importantes, proches des très grandes métropoles (Paris, Lyon, Lille, Nice).

In[214]:=
```
areas = QuantityMagnitude[CountryData[pays, "Area"]];
        magnitude de quantité  données de pays   aire
coord = Take[Table[QuantityMagnitude[CityData[c, "Coordinates"]],
        prends table magnitude de quantité données de villes
     {c, CityData[{All, pays}]}], ny];
        données de v· tout
n1 = Length[coord];
     longueur
aa = Outer[GeoDistance, coord, coord, 1] // QuantityMagnitude;
     produi·· distance géographique          magnitude de quantité
aa = aa /. 0. → Max[Flatten[aa]];
                ma·· aplatis
aa = aa / 1000;
moydata = Mean[Map[(Apply[Min, #]) &, aa]];
          valeu·· appl··· rempl··· minimum
moytheorique = 1 / 2 * Sqrt[areas / n1];
                       racine carrée
r = moydata / moytheorique;
Print["Indice R = ", r]
imprime
Print["Degré de liberté = ", n1 - 1]
imprime
Print["Si R = 1 la répartition est aléatoire"]
imprime
Print["Si R = 0 la répartition est concentrée"]
imprime
Print["Si R = 2.15 la répartition est régulière"]
imprime
```

```
Indice R = 0.0010095154

Degré de liberté = 74

Si R = 1 la répartition est aléatoire

Si R = 0 la répartition est concentrée

Si R = 2.15 la répartition est régulière
```

Attention, ce programme recalcule les coordonnées des ny (75) villes; Il serait possible de se servir de la fonction data.

Choisir un processus stochastique spatial

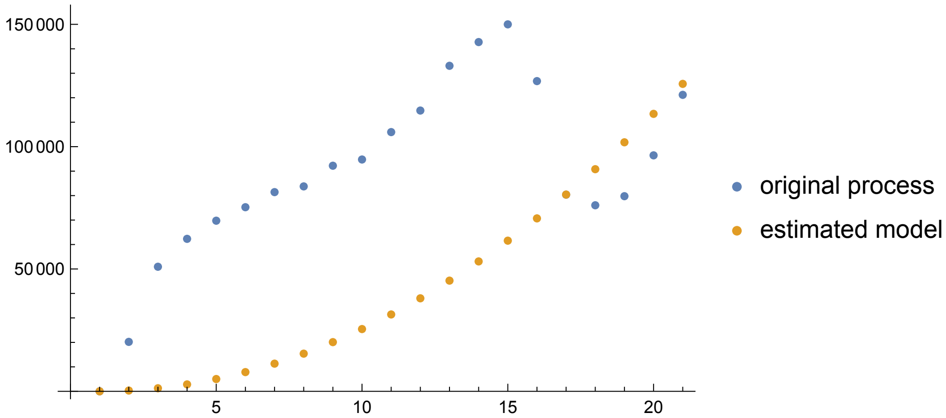
Depuis la dernière version du logiciel (12.2), de très nombreux processus stochastiques adaptés à l'étude de données ponctuelles sont disponibles dans Mathematica. Ils sont classés en trois groupes. Le premier groupe simule des processus d'indépendance (**PoissonPointProcess**[], **InhomogeneousPoissonPointProcess**[], **BinomialPointProcess**[]). Le deuxième groupe comprend les modèles d'interaction entre points (**HardcorePointProcess**[], **StaussPointProcess**[], **GibbsPointProcess**[],...), tandis que la troisième catégorie permet d'ajuster des configurations qui présentent des regroupements de points (**MaternPointProcess**[], **ThomasPointProcess**[],...).

Pour estimer les paramètres de différents processus, une seule instruction est nécessaire, **EstimatedPointProcess**[] ou **FindPointProcessParameters**[] quand le processus est déjà connu. Et, il est possible de vérifier que la configuration des plus grandes villes n'est pas un processus de Poisson, c'est-à-dire une répartition uniforme, avec le petit programme ci-dessous, qui compare les deux fonctions de Ripley pour la configuration des villes et pous un processus de Poisson appliqué aux villes :

```
In[•]:= proc = EstimatedPointProcess[data, PoissonPointProcess[λ, 2]]
               processus ponctuel estimé        processus ponctuel de Poisson
        ListPlot[{RipleyK[data, Range[0, 200, 10]], RipleyK[proc, Range[0, 200, 10]]},
        tracé de liste  K de Ripley     plage             K de Ripley    plage
         PlotLegends → {"original process", "estimated model"}]
         légendes de tracé
```

```
Out[•]= PoissonPointProcess[ 0.000134787 /km² , 2 ]
```

Out[•]=



La figure obtenue indique que la configuration des villes est de type agrégée. Les processus correspondant à des concentrations de points sont plus indiqués. Il est possible d'en calculer les paramètres.

```
est = EstimatedPointProcess[data,
      processus ponctuel estimé
  MaternPointProcess[a, b, c, 2], PointProcessEstimator → "FindClusters"]
  processus ponctuel de Matern    estimateur de processus ponctuel  trouve grappes
```

```
Out[•]= MaternPointProcess[ 5.464338 × 10^-11 per meter² , 2.4666667, 66 356.467, 2 ]
```

Il est alors possible de vérifier la pertinence du modèles avec la fonction PointProcessFitTest[]. Mais, toutes cas fonctions sont encore expérimentales et imparfaites. Elles fonctionnent seulement bien quand la fonction **SpatialPointData**[] ne prend pas en compte un polygone représentatif d'un pays, mais retient uniquement un polygone de Ripley-Rasson.

Analyse des points marqués

Quand une valeur est attribuée à des coordonnées spatiales, par exemple la population aux coordonnées des villes, les points sont dits marqués. Il est possible de créer un nouveau jeu de données avec la fonction SpatialPointData[], en se servant de l'option *annotation*, et donc de rattacher la population d'une ville à ses coordonnées. Les valeurs de ces variables spatialisées ne sont pas, sauf exception, indépendantes. Elles sont corrélées entre elles, donc auto-corrélées. Sans auto-corrélation, il n'y aurait pas de structure spatiale observable. Deux techniques sont privilégiées le corrélogramme et le variogramme. Mais, ces deux outils, très faciles à programmer pour une image, sont plus difficiles à implémenter pour des configurations de points inégalement répartis. On préfère alors calculer l'auto-corrélation de Geary ou de Moran, ou même interpréter les résultats obtenus avec les fonctions de Ripley et de Besag, présentées ci-dessus. Ci-dessous un programme pour calculer les indices de Moran et celui de Geary appliqué aux 120 plus grandes villes d'Italie.

In[256]:=
```
ClearAll["Global`*"]
pays = "Italy";
areas = CountryData[pays, "Area"];
ny = Input["Choisir le nombre de villes"] // ToExpression;
ville = CityData[{All, pays}][[ ;; ny]];
pop = CityData[#, "Population"] & /@ ville // QuantityMagnitude;
coord = Reverse[CityData[#, "Coordinates"]] & /@ ville;
n1 = Length[coord];
gg = NearestNeighborGraph[coord, 3];
wij = Normal[AdjacencyMatrix[gg]];
somWij = Total[Flatten[wij]];
(*Calcule indice de Moran avec graphe de voisinage ordre 3 *)
popm = pop - Mean[pop] // N;
prod = Table[popm[[i]] * popm[[j]], {i, 1, n1}, {j, 1, n1}] // N;
prod = N[wij * prod];
numer = Total[Flatten[prod]] * n1;
prodx = popm ^ 2;
denom = somWij * Total[prodx];
moran = numer / denom // N;
(*Calcule indice de Geary avec graphe de voisinage ordre 3 *)
sompopmean = Total[popm * popm];
denominateur = 2 * somWij * sompopmean // N;
difxixj = Table[pop[[i]] - pop[[j]], {i, 1, n1}, {j, 1, n1}] // N;
difxixj = difxixj ^ 2;
prod = N[wij * difxixj];
numerateur = Total[Flatten[prod]] * (n1 - 1) // N;
geary = numerateur / denominateur // N;
Print["Coefficient d'autocorrelation de Moran = ", moran]
```

```
Print["Coefficient d'autocorrelation de Geary = ", geary]
imprime    coefficient
Print["Attention  l'indice de Moran varie entre
imprime
    -1 et +1, alors que celui de Geary varie entre 0 à 2"]
```

```
Coefficient d'autocorrelation de Moran = -0.087789544

Coefficient d'autocorrelation de Geary = 1.2858442

Attention  l'indice de Moran varie entre
  -1 et +1, alors que celui de Geary varie entre 0 à 2
```

Ce programme affiche une auto-corrélation pratiquement nulle, après avoir déterminé le graphe de la matrice topologique de contiguïté des 120 villes. Il serait possible d'améliorer ce graphe topologique des contiguïté en positionnant bien les différentes villes, donc en réalisant un graphe géographique. Le lecteur peut changer de pays , ou modifier le nombre de villes à retenir dans la fenêtre qui s'affiche.

Des points à l'espace champ

Les méthodes d'interpolation, pour passer de données ponctuelles à un champ, sont très nombreuses. Dans ce notebook nous en présentons trois : les surfaces de tendances, l'interpolation par la méthode des potentiels, et les polygones de Voronoï.

Les surfaces de tendances

Le petit programme ci-dessous calcule la surface de tendance d'ordre 3, de la population des 100 plus grandes villes italiennes, puis positionne cette surface sur la carte d'Italie. Il suffit de remplacer le mot Italie par Spain pour obtenir le même modèle appliqué à la population des 100 plus grandes villes espagnoles. Et rien n'empêche de relancer le programme en réduisant ou augmentant le nombre de villes.

In[317]:=

```
country = "Italy";
ny = Input["Choisir le nombre de villes"] // ToExpression;
coord =
  Take[Table[CityData[c, "Coordinates"], {c, CityData[{All, country}]}], ny];
coordxy = GeoGridPosition[GeoPosition[#, "WGS84"],
      {"UTMZone31", "CentralScaleFactor" → 0.9996,
       "GridOrigin" → {500 000, 0}}][[1]] & /@ coord;
pop = Take[Table[QuantityMagnitude[CityData[c, "Population"]],
     {c, CityData[{All, country}]}], ny];
xyz = Partition[Flatten[Riffle[coordxy, pop]], 3];
Print["Paramètres du modèle d'ordre 3"]
ln = GeneralizedLinearModelFit[xyz,
  {x, y, x * y, x^2, y^2, x * y^2, y * x^2, x^3, y^3}, {x, y}]
Print["tests du modèle d'ordre 3"]
test = ln[{"AIC", "BIC"}];
Grid[{{test_AIC, test[[1]]}, {test_BIC, test[[2]]}}, Frame → All]
fit = ln["BestFit"];
Print["Surface de tendance et valeurs des données"]
minx = Min[xyz[[All, 1]]]; miny = Min[xyz[[All, 2]]];
maxx = Max[xyz[[All, 1]]]; maxy = Max[xyz[[All, 2]]];
surface = Image[ContourPlot[fit, {x, minx, maxx}, {y, miny, maxy}, Contours → 10,
     Frame → False, ClippingStyle → None, ColorFunction → "GrayTones",
     Epilog → {Red, Point[coordxy]}]] // ColorNegate;
GeoGraphics[{GeoStyling[{"GeoImage", Rasterize[surface]}],
  EdgeForm[Thin], Polygon[Entity["Country", country]]}]
```

Paramètres du modèle d'ordre 3

Out[324]= FittedModel[$-8.0071716\times10^{8}+\ll10\gg+8.1289446\times10^{-12}\,x\,y^{2}+4.7071684\times10^{-12}\,y^{3}$]

tests du modèle d'ordre 3

Out[327]=

test_AIC	2827.1673
test_BIC	2853.219

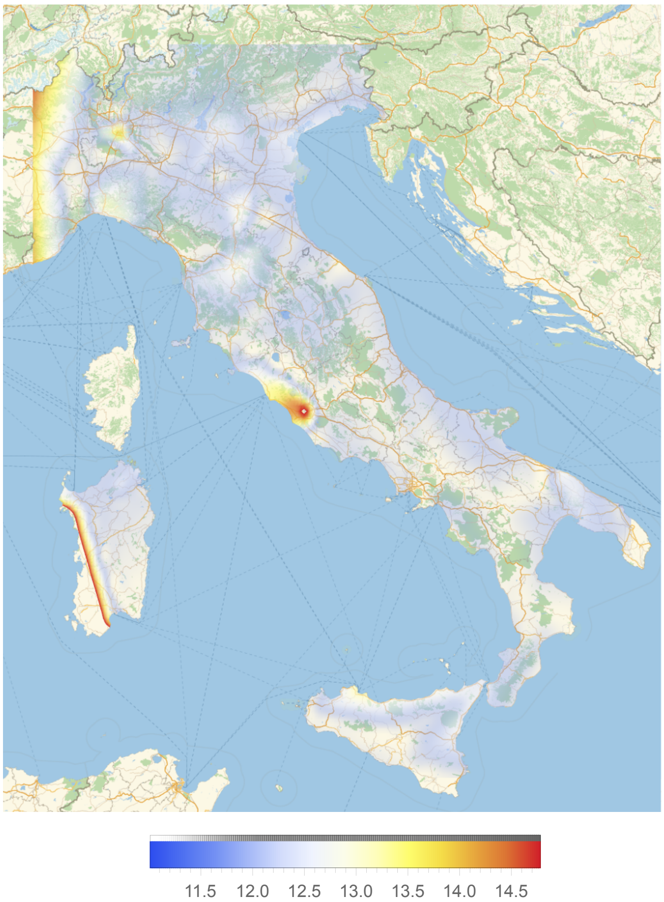
Surface de tendance et valeurs des données

Out[333]=

Les tests AIC et BIC indiquent que ce modèle est peu significatif, sans doute à cause d'erreurs qui grandissent vite sur les bords. Quelques villes littorales ne sont pas affichées sur la carte. Un bug ou votre serviteur? D'autres fonctions, notamment **GeoDensityPlot**[] et **GeoContourPlot**[], permettent de procéder à des interpolations avec des fonctions splines. Le programme ci-dessous calcule puis cartographie les Log des populations des 100 plus grandes villes italiennes.

In[334]:=
```
GeoDensityPlot[coord → Log[pop],
  InterpolationOrder -> 3, ColorFunction -> "TemperatureMap",
  RegionFunction -> Entity["Country", "Italy"],
  PlotLegends → Placed[Automatic, Bottom], GeoZoomLevel → 7]
```
tracé de cartographie de densité | logarithme | ordre d'interpolation | fonction de couleur | fonction de région | entité | légendes de tracé | placé | automatique | bas | niveau de zoom géographique

Out[334]=



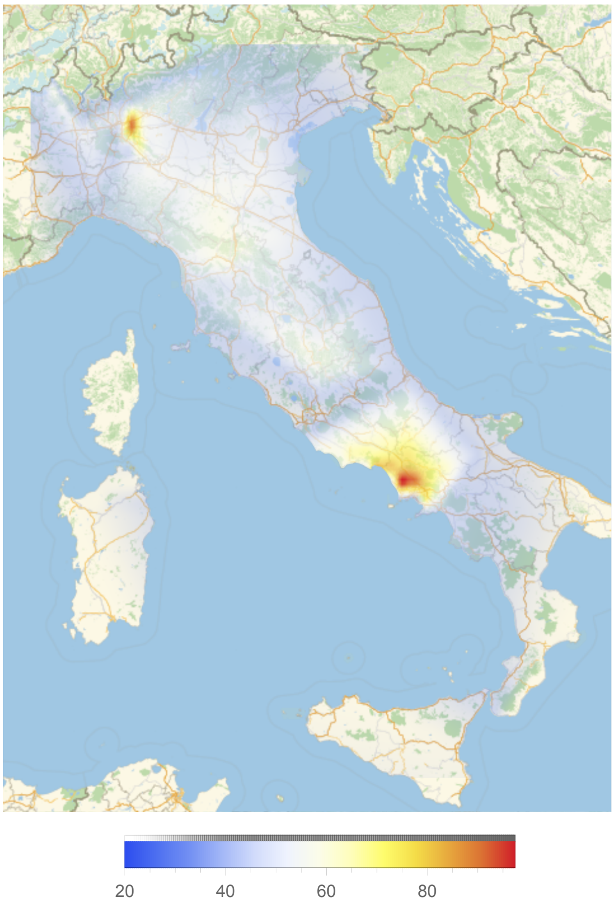
Le lecteur attentif remarquera quelques anomalies sur les marges des cartes, notamment en Sardaigne. Elles peuvent être corrigées.

Les modèles de potentiel

Les modèles de potentiels furent très utilisés au début des années soixante. le programme ci-dessous calcule le potentiel pour la variable population de 50 villes italiennes, puis affiche le résultat sur une carte des densités:

In[335]:=
```
ClearAll["Global`*"]
efface tout
villes = CityData[{All, "Italy"}];
         données de v· tout
ny = Input["Choisir le nombre de villes"] // ToExpression;
     entrée                                 en expression
latlong = Table[Reverse[CityData[villes[[k]], "Coordinates"]], {k, 1, ny}];
          table inverse données de villes
coord = Table[CityData[villes[[k]], "Coordinates"], {k, 1, ny}];
        table données de villes
pop =
  Table[CityData[villes[[k]], "Population"] // QuantityMagnitude, {k, 1, ny}];
  table données de villes                     magnitude de quantité
distances = EuclideanDistance @@@ Tuples[latlong, {2}];
            distance euclidienne      tuples
abs[x_] := If[x == 0, 0.0001, x];
           si
dist = abs /@ Flatten[distances];
              aplatis
distances = Partition[dist, ny];
            partitionne
pot = Table[pop[[i]] / distances[[i, j]], {i, 1, ny}, {j, 1, ny}];
      table
potentiel = Total[pot];
            total
diag = Diagonal[pot];
       diagonale
potc = potentiel - diag;
potrelatif = (potc / Max[potc]) * 100;
                     maximum
GeoDensityPlot[coord → potrelatif,
tracé de cartographie de densité
 InterpolationOrder -> 3, ColorFunction -> "TemperatureMap",
 ordre d'interpolation     fonction de couleur
 RegionFunction -> Entity["Country", "Italy"],
 fonction de région entité
 PlotLegends → Placed[Automatic, Bottom]]
 légendes de tracé placé automatique bas
```

Out[349]=



Remarquez que pour le calcul des distances, les coordonnées doivent être inversées avant d'être incluses dans le fichier latlong.

Les polygones de Voronoï

La détermination des polygones de Voronoï est relativement rapide avec l'instruction **VoronoiMesh**[] appliquée aux coordonnées de 55 villes italiennes. Puis, le programme ci-dessous affiche ces polygones sur la carte d'Italie :

In[350]:=
```
country = "Italy";
ny = Input["Choisir le nombre de villes"] // ToExpression;
      entrée                                              en expression
coords = Take[Table[
              prends table
    Reverse[CityData[c, "Coordinates"]], {c, CityData[{All, country}]}], ny];
    inverse  données de villes                   données de v·  tout
vo = VoronoiMesh[coords];
     maille de Voronoï
Graphics[{Lighter@Brown, CountryData["Italy", "Polygon"],
graphique   plus clair  marron  données de pays            polygone
  PointSize[Medium], Red, Point /@ coords, Thick, White, MeshPrimitives[vo, 1]}]
  taille de point  taille moy··· rouge point          épais   blanc   primitives de maille
```

Out[354]=

Il serait possible de traduire ce graphique en image et de procéder à son traitement. Remarquez que pour obtenir ce résultat, comme dans l'exemple précédent, il faut inverser les coordonnées, avec la fonction **Reverse**[]. Dans les Resource Function, **PowerDiagram**[] généralise les polygones de Voronoï en donnant à chaque point un poids, par exemple la population ou tout autre variable quantitative. Et, il est aussi possible de tracer les polygones de Voronoï autour de lignes et de paraboles avec la Resource Function **ApproximateGeneralizedVoronoiMesh**[]. En clair, tracer les polygones de Voronoï en tenant compte de la localisation des villes et d'axes de transport les reliant.

Conclusion

Bien d'autres méthodes sont utilisées pour analyser les configurations de points, notamment le krigeage. Mais, les plus accessibles sont construites avec les techniques de traitement d'images, notamment la morphologie mathématique. Nous y reviendrons dans un autre notebook.

Ouvrages recommandés :

Noel Cressis, 1991, *Statistics for Spatial Data*, Wiley.
Jean-Marc Zaninetti, 2005, *Statistique spatiale*, Hermès, Lavoisier