

TIPE: Circle Packing : de la distanciation sociale à la théorie des nombres.

Yasser ALGHOUASS

Épreuve de TIPE

Session 2022

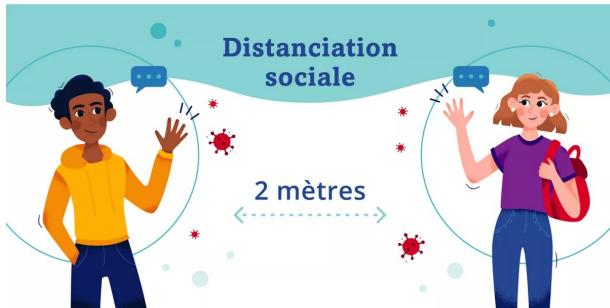
Plan de l'exposé

- 1 Introduction
- 2 Empilement de cercle (Circle packing)
- 3 Cercle d'Apollonius (Apollonian circle packings)
- 4 Application pratique
- 5 Conclusion
- 6 Références bibliographiques
- 7 Annexe

Introduction

Position du problème

- la distanciation sociale



Introduction

Position du problème

- l'utilisation des cercles



Introduction

Position du problème

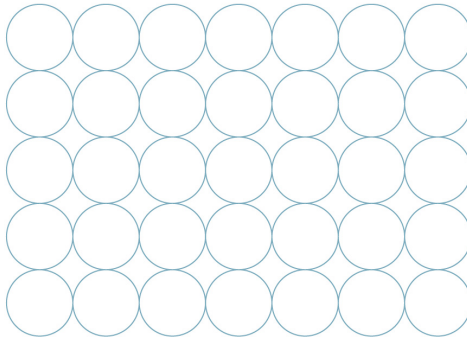
- Comment rouvrir en toute sécurité des bâtiments et des espaces publics dans un contexte de distanciation sociale ?



Introduction

Exemples

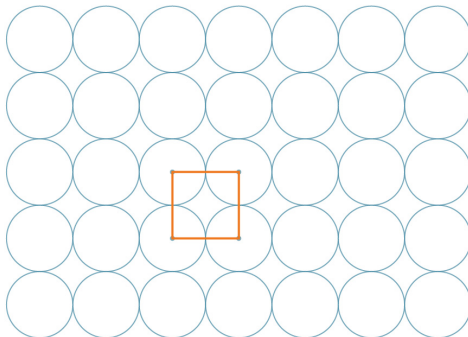
- La caisse de canettes de soda : réseau carré



Introduction

Exemples

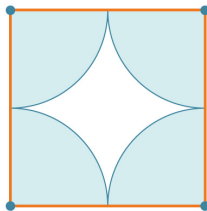
- La caisse de canettes de soda : réseau carré
 - La « densité d'empilement » de l'arrangement :



Introduction

Exemples

- La caisse de canettes de soda : réseau carré
 - La « densité d'empilement » de l'arrangement :

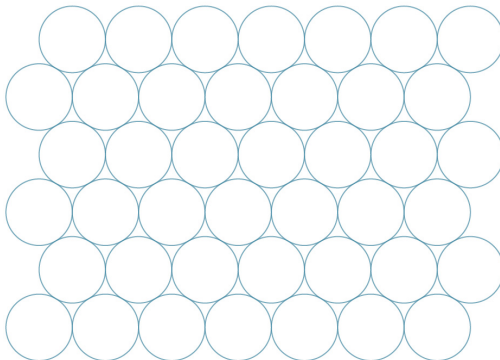


- la densité d'empilement : $\frac{\pi r^2}{(2r)^2} = \frac{\pi}{4} \approx 0.7854$

Introduction

Exemples

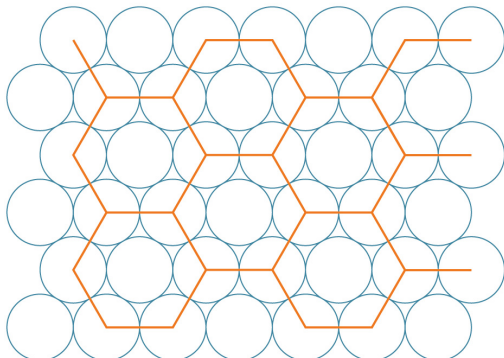
- La caisse de canettes de soda : réseau hexagonal



Introduction

Exemples

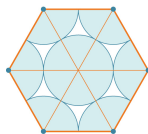
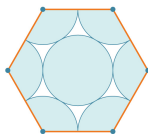
- La caisse de canettes de soda : réseau hexagonal
 - La « densité d'empilement » de l'arrangement :



Introduction

Exemples

- La caisse de canettes de soda : réseau hexagonal
 - La « densité d'empilement » de l'arrangement :



- la densité d'empilement :
 - la surface totale des cercles est $3\pi r^2$
 - la surface de l'hexagone de côté $2r$ est $6 \times \frac{(2r)^2 \sqrt{3}}{4} = 6r^2 \sqrt{3}$
 - la densité est $\frac{3\pi r^2}{6r^2 \sqrt{3}} = \frac{\pi}{2\sqrt{3}} \approx 0.9069$

Introduction

Problématique

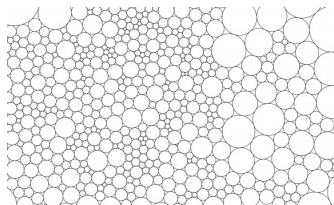
- Comment répartir les gens le plus efficacement possible, autrement dit, comment les répartir de manière à avoir le plus grand nombre de personnes possible dans une certaine pièce de dimension donnée - tout en respectant cette règle de distanciation ?

Empilement de cercle (Circle packing)

Définitions

Empilement de cercle

Un empilement de cercles $\mathcal{E}$ de la sphère de Riemann $\hat{\mathbb{C}}$ est une collection de disques fermés d'intérieurs deux à deux disjoints telle que les composantes connexes du complémentaire de la réunion de ces disques soient bordées par exactement trois arcs de cercle.



Empilement de cercle (Circle packing)

Définitions

Densité d'empilement

La densité d d'un empilement est la proportion de l'espace qui est occupé par les boules : $d = \frac{\text{volume des boules}}{\text{volume de l'espace}}$.

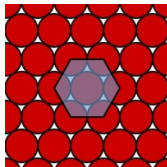
Empilement de cercle (Circle packing)

Théorème

Densité maximale d'un empilement régulier (Lagrange 1773)

La densité maximale d'un empilement régulier est

$d = \frac{\pi}{2\sqrt{3}} \approx 0.9069$, c'est le cas du réseau hexagonal.



- Mais est-ce vraiment l'empilement le plus dense dans le cas général ?

Empilement de cercle (Circle packing)

Théorème

Densité maximale d'un empilement (Thue 1890)

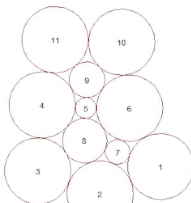
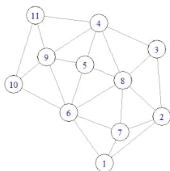
La densité maximale d'un empilement est $d = \frac{\pi}{2\sqrt{3}} \approx 0.9069$, c'est le cas du réseau hexagonal (arrangement hexagonal regulier).

Empilement de cercle (Circle packing)

Définition

Graphe d'incidence

A un empilement $\mathcal{E}$, on associe son graphe d'incidence $G(\mathcal{E})$ dont les sommets sont les disques et les arêtes sont les paires de disques dont les fermetures s'intersectent. Lorsque $\mathcal{E}$ est fini, il s'agit d'une triangulation que l'on peut voir plongée dans la surface.

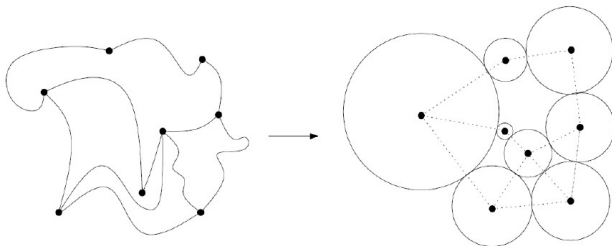


Empilement de cercle (Circle packing)

Théorème

Théorème d'empilement de cercle (Koebe–Andreev–Thurston)

Soit $\mathcal{T}$ une triangulation de la sphère $\mathbb{S}^2$. Il existe un unique empilement de cercles $\mathcal{E}$ de $\hat{\mathbb{C}}$ de graphe d'incidence isomorphe à $\mathcal{T}$, à transformation homographique près.



Cercle d'Apollonius

Définition



Cercle d'Apollonius

Définition

- On agrandit l'image avec un facteur de 252.
- On pose $a(c) = \text{courbure du cercle } c = \frac{1}{\text{rayon}(c)}$



- Il est à noter que la courbure extérieure est par convention

Cercle d'Apollonius

Théorème

Théorème d'Apollonius

Etant donné trois cercles c_1 , c_2 , c_3 mutuellement tangents, il existe exactement deux cercles c et c' tangents à ces trois cercles.

Cercle d'Apollonius

Définition

- En appliquant le théorème d'Apollonius, on place sur la figure les cercles tangents.



Cercle d'Apollonius

Définition

- Le miracle diophantien est que les courbures sont des entiers !



Cercle d'Apollonius

Questions qui sautent aux yeux :

- Quels entiers apparaissent comme des courbures ?
- Y a-t-il une infinité de courbures premières, des nombres premiers jumeaux, c'est-à-dire des paires de cercles tangents avec courbure première ?

Cercle d'Apollonius

Théorème

Théorème de Descartes

Etant donné quatre cercles tangents entre eux, de courbures a_i , $i \in \{1, 2, 3, 4\}$ (avec la convention de signe), alors

$$F(a_1, a_2, a_3, a_4) = 0$$

avec F est la forme quadratique

$$F(a_1, a_2, a_3, a_4) = 2(a_1^2 + a_2^2 + a_3^2 + a_4^2) - (a_1 + a_2 + a_3 + a_4)^2$$

Cercle d'Apollonius

Propriété

Propriété

Etant donné quatre cercles c_1, c_2, c_3, c_4 tangents entre eux, de courbures a_i , $i \in \{1, 2, 3, 4\}$. Si c_4 et c'_4 sont tangents à c_1, c_2, c_3 , alors

$$F(a_1, a_2, a_3, a_4) = 0$$

$$F(a_1, a_2, a_3, a'_4) = 0$$

Donc a_4 et a'_4 sont racines de la même équation quadratique

$$\Rightarrow (1) \begin{cases} a_4 + a'_4 = 2(a_1 + a_2 + a_3) \\ a_4, a'_4 = a_1 + a_2 + a_3 \pm 2\sqrt{\Delta} \\ \Delta = a_1a_2 + a_1a_3 + a_2a_3 \end{cases}$$

Cercle d'Apollonius

Exemple

Exemple

Dans notre exemple P_0 :

$$(a_1, a_2, a_3) = (21, 24, 28) \text{ et } \Delta = 1764 = 42^2$$

Si c_1, c_2, c_3, c_4 ont des courbures entières, alors c'_4 a aussi une courbure entière (ceci à partir de (1))

De la sorte, toute courbure construite est entière !

Groupe d'Apollonius

Définition

(1) en haut implique que lors de la formation d'une nouvelle courbure lors de l'insertion d'un nouveau cercle

$$a'_4 = -a_4 + 2a_1 + 2a_2 + 2a_3$$

On pose :

$$S_4 = \begin{bmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

$$S_2 = \begin{bmatrix} 1 & 2 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 2 & 1 & 0 \\ 0 & 2 & 0 & 1 \end{bmatrix}$$

$$S_3 = \begin{bmatrix} 1 & 0 & 2 & 0 \\ 0 & 1 & 2 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 2 & 1 \end{bmatrix}$$

$$S_1 = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 2 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \end{bmatrix}$$

$$a' = aS_4, \quad a' = (a_1, a_2, a_3, a'_4) \in \mathbb{Z}^4$$

Groupe d'Apollonius

Définition

De même en engendrant $c_1, c_2, c_3, \dots$

$$a' = aS_j, \quad j = 1, 2, 3,$$

On a

$$S_j^2 = I_4, \quad S_j \in GL_4(\mathbb{Z}), \quad j = 1, 2, 3, 4.$$

Définition

Le groupe d'Appolonius A est le sous groupe de $GL_4(\mathbb{Z})$ engendré par S_1, S_2, S_3, S_4 .

C'est le groupe de symétrie du Cercle d'Apollonius.

Cercle d'Apollonius et nombres entiers

Théorèmes

On pose :

$$N_P(x) = |\{C \in P : a(C) \leq x\}|$$

Théorèmes

$$(D.Boyd) : \lim_{x \rightarrow +\infty} \frac{\ln(N_P(x))}{\ln(x)} = \delta \approx 1.30568$$

(A.Kontorovich-H.OH) : Il existe $b = b(P) > 0$ tel que :

$$N_P(x) \underset{x \rightarrow +\infty}{\sim} bx^\delta$$

(J.Bourgain-E.Fuchs) : Il existe $C > 0$ tel que le nombre d'entiers inférieur à x qui sont des courbures vaut au moins Cx .

Cercle d'Apollonius et nombres premiers

Théorème

Théorème

Dans tout Cercle d'Apollonius, il y'a une infinité de nombres premiers jumeaux, en particulier il y'a une infinité de courbures premières.

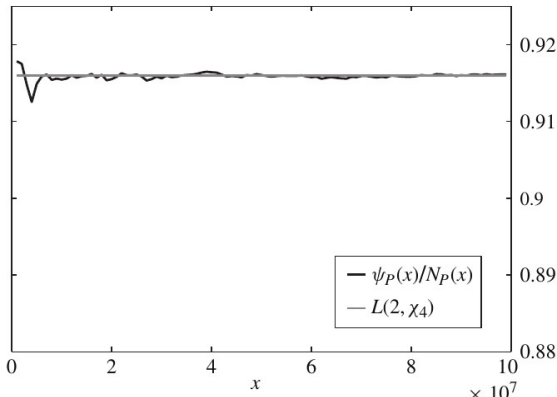
On pose :

$$\Psi_P(x) = \sum_{\substack{c \in P \\ a(c) \text{ premier} \\ a(c) \leq x}} \ln(a(c))$$

Cercle d'Apollonius et nombres premiers

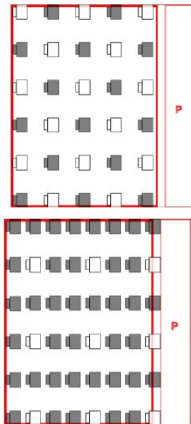
On conjecture que :

$$\frac{\Psi_P(x)}{N_P(x)} \xrightarrow{x \rightarrow +\infty} L(2, \chi_4) \approx 0.9159$$



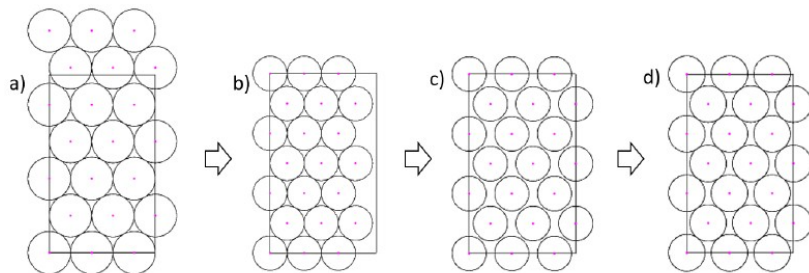
Application pratique

Optimisation de la distanciation sociale dans une salle de classe



Application pratique

Optimisation de la distanciation sociale dans une salle de classe



Conclusion

Après l'étude des possibilité d'empilement de cercles identiques de même dimension, il s'avère que la configuration hexagonale est celle qui permet d'optimiser au maximum l'utilisation d'un espace contraint à une distanciation sociale.

L'empilement de cercles de dimensions différentes, qui est le cas du cercle d'Apollonius, bien qu'il paraît comme problème purement géométrique, il se trouve lié à de nombreux domaines mathématiques, notamment la théorie des nombres...

MERCI POUR VOTRE
ATTENTION!

Références bibliographiques

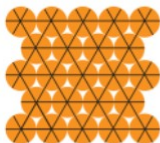
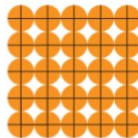
- [1] Patrick Honner : The Math of Social Distancing Is a Lesson in Geometry
- [2] Andres Iglesias and Patricia Suarez : An Optimisation Model for Designing Social Distancing Enhanced Physical Spaces.
- [3] J. C. Bortolete · L. F. Bueno : A support tool for planning classrooms considering social distancing between students .
- [4] Asaf Nachmias : Planar Maps, Random Walks and Circle Packing : Planar Maps, Random Walks and Circle Packing.
- [5] Charles R. Collins , Kenneth Stephenson : A circle packing algorithm.
- [6] ELENA FUCHS : COUNTING PROBLEMS IN APOLLONIAN PACKINGS : BULLETIN (New Series) OF THE AMERICAN MATHEMATICAL SOCIETY Volume 50, Number 2, April 2013, Pages 229–266

Calcul de densité

Empilement sur un réseau carré et réseau hexagonal

Avec la méthode ci-contre, on peut paver le plan par des carrés de côté $2r$, dont les sommets sont aux centres des disques. Chaque carré a pour aire $4r^2$. De cette aire, une proportion de πr^2 est recouverte par des disques, puisque on a quatre quarts de disque de rayon r . Donc, la densité de l'empilement est :

$$d_1 = \frac{\pi r^2}{4r^2} = \frac{\pi}{4} \approx 0,7854.$$



On voit que la densité est indépendante du rayon des disques. Donc, pour simplifier, on supposera par la suite que le rayon est égal à 1. Avec la méthode en haut à droite, on peut paver le plan par des triangles équilatéraux de côté 2, de sommets aux centres des disques. Chaque triangle a pour aire, $\sqrt{3}$. De cette aire, $\pi/2$ est recouverte par des disques, puisque on a trois sixièmes de disque de rayon r . Donc, la densité de l'empilement est :

$$d_2 = \frac{\pi/2}{\sqrt{3}} = \frac{\pi}{2/\sqrt{3}} \approx 0,9069.$$

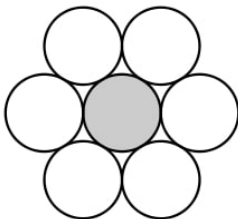
Ce calcul confirme ce que notre œil pressentait, à savoir que l'empilement de droite est plus dense que celui de gauche. Le réseau des centres de droite est appelé *réseau hexagonal*.

Calcul de densité

Densité maximale d'un empilement régulier (Lagrange)

Nombres de contact connus (kissing number) :

En deux dimensions, le nombre de contact est 6. Les boules sont des disques unitaires ; on peut imaginer qu'elles représentent des pièces de monnaie que l'on arrange pour qu'elles touchent toutes une pièce commune.



Calcul de densité

Densité maximale d'un empilement régulier (Lagrange)

Sur un plan, on peut disposer au maximum six cercles de rayon r autour d'un cercle de même rayon. Les centres de trois cercles en contact définissent un triangle équilatéral puisqu'ils sont distants de $2r$ les uns des autres. Chaque angle valant $60^\circ (\frac{\pi}{3})$, on peut mettre ainsi 6 triangles avec un sommet en commun pour former un hexagone régulier.

On peut constater aisément que c'est l'organisation la plus compacte qui soit en rangeant des billes de même volume dans une enceinte de taille appropriée.

La densité surfacique de cet arrangement est :

$$d = \frac{\pi}{2\sqrt{3}} \approx 0.9069$$

Calcul de densité

Densité maximale d'un empilement régulier (Lagrange)

Considérons quatre cercles en contact deux à deux. Les centres de ces cercles forment un losange de côté $2r$. On peut ainsi découper le plan en un pavage de losanges définissant un réseau. Chaque losange comprend deux portions de disque d'angle au centre $\frac{2\pi}{3}$ et deux portions de disque d'angle au centre $\frac{\pi}{3}$. La somme de ces quatre angles au centre est ainsi égale à 2π , donc la somme des aires des quatre portions de disque est égale à la surface d'un disque complet, soit πr^2 .

Le losange lui-même a pour aire $2\sqrt{3} r^2$. Les disques occupent donc une proportion de surface égale à $d = \frac{\pi r^2}{2\sqrt{3} r^2}$.

Calcul de densité

Densité maximale d'un empilement (Thue)

On divise le plan en régions, et on montre que, dans chaque région, la densité est inférieure ou égale à $d = \frac{\pi}{2\sqrt{3}}$.

Agrandissons les disques jusqu'à remplir tout l'espace.



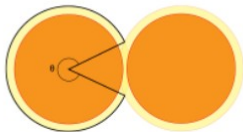
Le grand rayon est le rayon R du cercle circonscrit au triangle : il est égal aux $2/3$ de la hauteur, soit $R = 2\sqrt{3}/3$. On agrandit ainsi tous les disques, que l'on va appeler des grands disques. On peut maintenant diviser notre plan en trois types de régions.

La première région est la portion du plan non couverte par un grand disque. Dans cette région, la densité est bien sûr nulle.

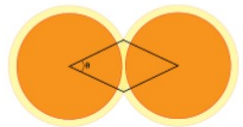
Calcul de densité

Densité maximale d'un empilement (Thue)

Le deuxième type de région est un secteur de grand disque tel qu'encerclé sur la figure :



Le troisième type de région est de la forme :



Calcul de densité

Densité maximale d'un empilement (Thue)

Calcul de la densité dans les régions de deuxième et troisième type

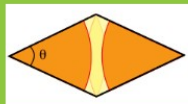
Calculer l'aire d'une région du deuxième type n'est pas difficile. L'aire d'un secteur de disque de rayon r d'ouverture θ est, par une règle de trois, $r^2\theta/2$ (où l'angle θ est en radians). Donc, la densité est :

$$\frac{\theta/2}{R^2\theta/s} = \frac{1}{R^2} = \frac{3}{4} = 0.75,$$

qui est inférieure à $d_2 \approx 0,9069$.

Passons maintenant à la densité dans le troisième type de région. La région couverte par les petits disques est la réunion de deux secteurs de rayon 1 et d'ouverture θ , qui a donc pour aire θ . L'aire de la région est la somme des aires de deux triangles isocèles ayant chacun deux côtés de longueur R = $2\sqrt{3}/3$ séparés par un angle θ . Cette aire vaut donc :

$$2R^2 \sin \frac{\theta}{2} \cos \frac{\theta}{2} = R^2 \sin \theta = \frac{4}{3} \sin \theta.$$



Calcul de densité

Densité maximale d'un empilement (Thue)

La densité dépend de θ , c'est une fonction :

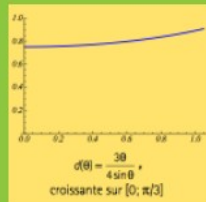
$$d(\theta) = \frac{3\theta}{4 \sin \theta}.$$

Remarquons que $\theta \in [0, \pi/3]$.

On a :

$$d(\pi/3) = \frac{\pi}{2\sqrt{3}}.$$

La conclusion suit si on montre que la fonction est croissante sur l'intervalle $[0, \pi/3]$. Cela peut se montrer rigoureusement, mais nous nous contenterons de le voir sur le graphe de la fonction $d(\theta)$.



Exemple d'empilement

Empilement de cercles dans un carré

Nombre de cercles (n)	Longueur du côté du carré (L)	d_n^1	Densité (n/L ²)	Figure
1	2	∞	0,25	
2	$2 + \sqrt{2}$ $\approx 3,414...$	$\sqrt{2}$ $\approx 1,414...$	0,172...	
3	$2 + \frac{\sqrt{2}}{2} + \frac{\sqrt{6}}{2}$ $\approx 3,931...$	$\sqrt{6} - \sqrt{2}$ $\approx 1,035...$	0,194...	
4	4	1	0,25	
5	$2 + 2\sqrt{2}$ $\approx 4,828...$	$\frac{1}{2}\sqrt{2}$ $\approx 0,707...$	0,215...	
6	$2 + \frac{12}{\sqrt{13}}$ $\approx 5,328...$	$\frac{1}{6}\sqrt{13}$ $\approx 0,601...$	0,211...	
7	$4 + \sqrt{3}$ $\approx 5,732...$	$4 - 2\sqrt{3}$ $\approx 0,536...$	0,213...	

Exemple d'empilement

Empilement de cercles dans un carré

Nombre de cercles unités de nombre n	Rayon du cercle extérieur	Densité	Optimalité	Figure
1	1	1,0000	Trivial	
2	2	0,5000	Trivial	
3	$1 + \frac{2}{\sqrt{3}} \approx 2,154...$	0,6466...	Trivial	
4	$1 + \sqrt{2} \approx 2,414...$	0,6954...	Trivial	
5	$1 + \sqrt{2\left(1 + \frac{1}{\sqrt{6}}\right)} \approx 2,701...$	0,6954...	Trivial Aussi prouvé optimal par Graham (1968) ²	
6	3	0,6667...	Trivial Aussi prouvé optimal par Graham (1968) ²	

Exemple d'empilement

Code informatique pour trouver l'empilement optimale

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from random import randint
4 from random import uniform
5 from matplotlib import animation
6 class Ball:
7     def __init__(self, x, y, radius):
8         self.r = radius
9         self.acceleration = np.array([0, 0])
10        self.velocity = np.array([uniform(0, 1),
11                                   uniform(0, 1)])
12        self.position = np.array([x, y])
13    @property
14    def x(self):
15        return self.position[0]
16    @property
17    def y(self):
18        return self.position[1]
19    def applyForce(self, force):
20        self.acceleration = np.add(self.acceleration, force)
21    def _normalize(self, v):
22        norm = np.linalg.norm(v)
23        if norm == 0:
24            return v
25        return v / norm
26    def update(self):
27        self.velocity = np.add(self.velocity, self.acceleration)
28        self.position = np.add(self.position, self.velocity)
29        self.acceleration *= 0
30
```

Exemple d'empilement

Code informatique pour trouver l'empilement optimale

```
31 class Pack:
32     def __init__(self, radius, list_balls):
33         self.iter = 0
34         self.list_balls = list_balls
35         self.r = radius
36         self.list_separate_forces = [np.array([0, 0])] * len(self.list_balls)
37         self.list_near_balls = [0] * len(self.list_balls)
38         self.wait = True
39     def _normalize(self, v):
40         norm = np.linalg.norm(v)
41         if norm == 0:
42             return v
43         return v / norm
44     def run(self):
45         self.iter += 1
46         for ball in self.list_balls:
47             self.checkBorders(ball)
48             self.checkBallPositions(ball)
49             self.applySeparationForcesToBall(ball)
50             print(ball.position)
51         print("\n")
```

Exemple d'empilement

Code informatique pour trouver l'empilement optimale

```
52 def checkBorders(self, ball):
53     d = np.sqrt(ball.x**2 + ball.y**2)
54     if d >= self.r - ball.r:
55         vr = self._normalize(ball.velocity) * ball.r
56         # P1 is collision point between circle and container
57         P1x = ball.x + vr[0]
58         P1y = ball.y + vr[1]
59         P1 = np.array([P1x, P1y])
60         # Normal vector
61         n_v = -1 * self._normalize(P1)
62         u = np.dot(ball.velocity, n_v) * n_v
63         w = np.subtract(ball.velocity, u)
64         ball.velocity = np.subtract(w, u)
65         ball.update()
66 def checkBallPositions(self, ball):
67     i = self.list_balls.index(ball)
68     # for neighbour in list_neighbours:
69     # ot a full loop; if we had two full loops, we'd compare every
70     # particle to every other particle twice over (and compare each
71     # particle to itself)
72     for neighbour in self.list_balls[i + 1:]:
73         d = self._distanceBalls(ball, neighbour)
74         if d < (ball.r + neighbour.r):
75             return
76     ball.velocity[0] = 0
77     ball.velocity[1] = 0
```

Exemple d'empilement

Code informatique pour trouver l'empilement optimale

```

78 def getSeparationForce(self, c1, c2):
79     steer = np.array([0, 0])
80     d = self._distanceBalls(c1, c2)
81     if d > 0 and d < (c1.r + c2.r):
82         diff = np.subtract(c1.position, c2.position)
83         diff = self._normalize(diff)
84         diff = np.divide(diff, 1 / d**2)
85         steer = np.add(steer, diff)
86     return steer
87
88 def _distanceBalls(self, c1, c2):
89     x1, y1 = c1.x, c1.y
90     x2, y2 = c2.x, c2.y
91     dist = np.sqrt((x2 - x1)**2 + (y2 - y1)**2)
92     return dist
93
94 def applySeparationForcesToBall(self, ball):
95     i = self.list_balls.index(ball)
96     for neighbour in self.list_balls[i + 1:]:
97         j = self.list_balls.index(neighbour)
98         forceij = self.getSeparationForce(ball, neighbour)
99         if np.linalg.norm(forceij) > 0:
100             self.list_separate_forces[i] = np.add(self.list_separate_forces[i], forceij)
101             self.list_separate_forces[j] = np.subtract(self.list_separate_forces[j], forceij)
102             self.list_near_balls[i] += 1
103             self.list_near_balls[j] += 1
104         if np.linalg.norm(self.list_separate_forces[i]) > 0:
105             self.list_separate_forces[i] = np.subtract(self.list_separate_forces[i], ball.velocity)
106         if self.list_near_balls[i] > 0:
107             self.list_separate_forces[i] = np.divide(self.list_separate_forces[i], self.list_near_balls[i])
108         separation = self.list_separate_forces[i]
109         ball.applyForce(separation)
110         ball.update()

```

Exemple d'empilement

Code informatique pour trouver l'empilement optimale

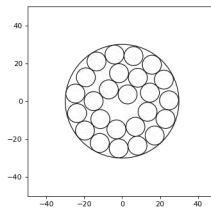
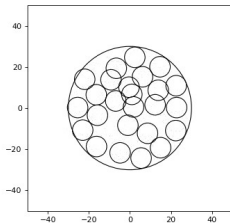
```

109 list_balls = list()
110 for i in range(25):
111     # b = Ball(randint(-15, 15), randint(-15, 15), 5)
112     b = Ball(0, 0, 5)
113     list_balls.append(b)
114 p = Pack(30, list_balls)
115 fig = plt.figure()
116 circle = plt.Circle((0, 0), radius=30, fc='none', ec='k')
117 plt.gca().add_patch(circle)
118 plt.axis('scaled')
119 plt.axes().set_xlim(-50, 50)
120 plt.axes().set_ylim(-50, 50)
121 def draw(i):
122     patches = []
123     p.run()
124     fig.clf()
125     circle = plt.Circle((0, 0), radius=30, fc='none', ec='k')
126     plt.gca().add_patch(circle)
127     plt.axis('scaled')
128     plt.axes().set_xlim(-50, 50)
129     plt.axes().set_ylim(-50, 50)
130     for c in list_balls:
131         ball = plt.Circle((c.x, c.y), radius=c.r, picker=True, fc='none', ec='k')
132         patches.append(plt.gca().add_patch(ball))
133     return patches
134 co = False
135 anim = animation.FuncAnimation(fig, draw,
136                               frames=500, interval=2, blit=True)
137 # plt.show()
138 anim.save('line2.gif', dpi=80, writer='imagemagick')

```

Exemple d'empilement

Code informatique pour trouver l'empilement optimale



Cercle d'Apollonius

Code informatique pour tracer le cercle d'Apollonius

```
1 import math
2 import cmath
3 import random
4 from PIL import Image, ImageDraw
5 lngx = 512
6 lngy = 512
7 image = Image.new('RGB', (lngx, lngy))
8 draw = ImageDraw.Draw(image)
9 maxIt = 2000 # of iterations
10 n = random.randint(2, 7) # of main circles > 1
11 print ('Number of main circles: ' + str(n))
12 eps = 0.0001
13 def appEq(u, v): # approximately equal excluding signs
14     return abs(abs(u) - abs(v)) <= eps
15 # test if 2 circles are approximately equal
16 def appEqCir(a, b):
17     return (abs(a[0] - b[0]) <= eps and appEq(a[1], b[1]))
18 # test if 2 circles are tangent
19 def isTanCir(a, b):
20     return (appEq(a[0] - b[0], abs(a[1]) + abs(b[1])) or appEq(a[0] - b[0], abs(a[1]) - abs(b[1])))
21 # test if 2 circles are intersecting
22 def isIntCir(a, b):
23     dist = abs(a[0] - b[0]) # distance between 2 centers
24     rmin = min(abs(a[1]), abs(b[1])) # min radius
25     rmax = max(abs(a[1]), abs(b[1])) # max radius
26     return (dist <= eps - rmin + rmax and dist >= rmin - rmax + eps)
27 # test if 2 circles are tangent to each other
28 def isTangent(a, b, c):
29     flag = True
30     if appEqCir(a, b) or appEqCir(a, c) or appEqCir(b, c): # all circles must be different
31         flag = False
32     if not isTanCir(a, b):
33         flag = False
34     if not isTanCir(a, c):
35         flag = False
36     if not isTanCir(b, c):
37         flag = False
38     return flag
39 # test if the 4th circle is tangent to previous 3
40 def isTan4(a, b, c, d):
41     return isTangent(a, b, d) and isTangent(a, c, d) and isTangent(b, c, d)
```

Cercle d'Apollonius

Code informatique pour tracer le cercle d'Apollonius

```

42 # input: 3 circles w/ each as a tuple: (complex(x,y),r)
43 # output: 4 tangent circles as a tuple
44 def cif(a, b, c):
45     # curvatures of the circles
46     k1 = 1.0 / a[1]
47     k2 = 1.0 / b[1]
48     k3 = 1.0 / c[1]
49     # curvatures of tangent circles
50     temp0 = 2.0 * math.sqrt(k1 * k2 + k1 * k3 + k2 * k3)
51     temp1 = k1 + k2 + k3
52     k40 = temp1 + temp0
53     k41 = temp1 - temp0
54     # centers of tangent circles
55     temp0 = 2.0 * cmath.sqrt(k1*k2*a[0]*b[0]+k1*k3*a[0]*c[0]+k2*k3*b[0]*c[0])
56     temp1 = k1 * a[0] + k2 * b[0] + k3 * c[0]
57     c40 = (temp1 + temp0) / k40
58     c41 = (temp1 - temp0) / k41
59     c42 = (temp1 - temp0) / k40
60     c43 = (temp1 + temp0) / k41
61     # radius of tangent circles
62     r0 = 1.0 / k40
63     r1 = 1.0 / k41
64     # there are 4 solutions and only 2 are correct (tangent) circles
65     # those are c0 and c1 or c2 and c3
66     c0 = (c40, r0)
67     c1 = (c41, r1)
68     c2 = (c42, r0)
69     c3 = (c43, r1)
70     if isTan4(a, b, c, c0) and isTan4(a, b, c, c1):
71         return (c0, c1)
72     else:
73         return (c2, c3)

```

Cercle d'Apollonius

Code informatique pour tracer le cercle d'Apollonius

```
74 # pixel coordinate calculation
75 def pxy(x, y):
76     kx = int((x - xa) / (xb - xa) * (imgx - 1))
77     ky = int((y - ya) / (yb - ya) * (imgy - 1))
78     return (kx, ky)
79 # generate the main circles: n circles in a circle
80 circles = []
81 a = 2.0 * math.pi / n
82 # radius of the inner circles
83 r = math.hypot(math.cos(a) - math.cos(0), math.sin(a) - math.sin(0)) / 2.0
84 r0 = 1.0 + r # radius of the outer circle
85 r1 = r0 - 2.0 * r # radius of the center circle
86 # origin
87 ox = r0
88 oy = r0
89 rmin = 2.0 * r0 / math.hypot(imgx, imgy) # min radius that can be drawn
90 # drawing area
91 xa = ox - r0
92 xb = ox + r0
93 ya = oy - r0
94 yb = oy + r0
95 circles.append((complex(ox, oy), -r0)) # add the outer circle w/ - curvature
96 for j in range(n): # add the main circles
97     cx = math.cos(a * j) + ox
98     cy = math.sin(a * j) + oy
99     circles.append((complex(cx, cy), r))
100 if r1 > rmin: # if possible add a center circle
101     circles.append((complex(ox, oy), r1))
102 p0 = 0
```

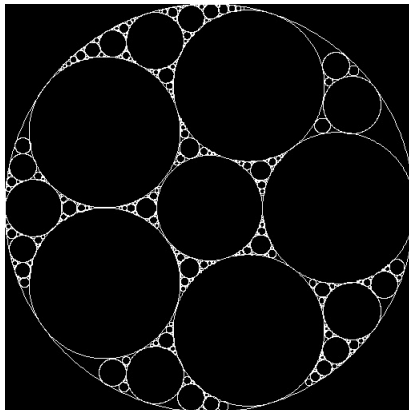
Cercle d'Apollonius

Code informatique pour tracer le cercle d'Apollonius

```
103 for i in range(maxIt):
104     p = int((100 * i / (maxIt - 1)))
105     if p > p0:
106         p0 = p
107         print("%i" % p)
108     # randomly select 3 circles
109     while True:
110         a = random.choice(circles)
111         b = random.choice(circles)
112         c = random.choice(circles)
113         # test if all 3 circles are tangent to each other
114         if isTangent(a, b, c):
115             break
116     # find inversion circles
117     d = ci(a, b, c)
118     for e in d:
119         dx = e[0].real
120         dy = e[0].imag
121         dr = e[1]
122         # ignore the outer circle (- curvature) solutions and too small circles
123         if dx > 0.0 and dy > 0.0 and dr > rmin:
124             # add the new circle if it was not added before and not intersecting
125             flag = True
126             for cir in circles:
127                 # new circle must not be equal/intersecting to any existing circle
128                 if appEqCir(cir, e) or isIntCir(cir, e):
129                     flag = False
130                     break
131             if flag:
132                 circles.append(e)
133     # draw all circles
134     for cir in circles:
135         dx = cir[0].real
136         dy = cir[0].imag
137         dr = abs(cir[1])
138         try:
139             draw.ellipse((pxy(dx - dr, dy - dr), pxy(dx + dr, dy + dr)))
140         except:
141             pass
142     image.save("ApollonianGasket." + str(n) + ".png", "PNG")
```

Cercle d'Apollonius

Code informatique pour tracer le cercle d'Apollonius



Cercle d'Apollonius

Conjecture

$$L(2, \chi_4) = \prod_{p \equiv 1 \pmod{4}} (1 - p^{-2})^{-1} \cdot \prod_{p \equiv 3 \pmod{4}} (1 + p^{-2})^{-1} = 0.9159 \dots,$$
$$\frac{\psi_P(x)}{N_P(x)} \longrightarrow L(2, \chi_4)$$

Application pratique

Algorithme

Algorithm 1: Multi-start procedure for the free-layout algorithm.

Initialization: $n_{\text{trials}} \geq 1$, $S = \emptyset$, $S_{\max} \geq 1$ and $N \geq 1$

1. Compute the solution $(\bar{c}, \bar{d})$ by Algorithm 2
2. **for** $k = 1, \dots, n_{\text{trials}}$ **do**
3. Compute a uniformly random value $Q \in [0, 1]$ and perform the perturbation strategy of Algorithm 3 to obtain $(\hat{c}, \bar{d})$
4. Apply the truncated penalization strategy of Subsection 3.1 with starting point $(\hat{c}, \bar{d})$, obtaining (c^*, d^*)
5. **if** (c^*, d^*) is feasible **then**
 - └ Consider adding (c^*, d^*) to S , possibly removing some other solution to ensure that $|S| \leq S_{\max}$
6. **for** (c^*, d^*) in S **do**
7. Solve problem 3 with Algenca, replacing the non-overlapping constraints by the single constraint 6, with starting point (c^*, d^*) and initial Lagrange multiplier $\lambda^0 = 0$

Application pratique

Algorithme

Algorithm 2: Geometric heuristic for the starting point

Initialization: Let N be the number of seats (points), and b and h be the dimensions of the room (rectangle).

- a. For each pattern p and number of rows f , determine a value q_f^p of elements in the first row:

$$q_f^p = \begin{cases} \lfloor N/f \rfloor, & \text{if } p \text{ is Z-type or I-type} \\ \lfloor (N - \lfloor \frac{f}{2} \rfloor)/f \rfloor + 1, & \text{if } p \text{ is X-type} \\ \lfloor (N - 1 + \lfloor \frac{f}{2} \rfloor)/f \rfloor, & \text{if } p \text{ is D-type} \end{cases}$$

- a. With the value of q_j^0 , calculate the value of the distance d_j^0 using b or h in order to occupy all the space in the dimension of the orientation (vertical or horizontal) of the chosen pattern:

$$d_f^p = \begin{cases} \frac{h}{q^p - 0.5}, & \text{if } p \text{ is horizontal Z-type} \\ \frac{h}{q^p - 0.5}, & \text{if } p \text{ is vertical Z-type} \\ \frac{h}{q^p - 1}, & \text{if } p \text{ is I-type or horizontal X- or D- types} \\ \frac{h}{q^p - 1}, & \text{if } p \text{ is vertical X- or D- types} \end{cases}$$

- a. If all points are internal then

- | Store the distance d_j^p

```

        |
    -else-

```

Resize $\mathcal{A}_f^i$ in order to ensure that all the rows are inside the room and then space the points in the direction of unused space in the dimension of the pattern orientation:

$$d_p^* = \begin{cases} \min \left(\sqrt{\frac{1}{\left(\frac{1}{\gamma^2} - 1\right)^2} + \left(\frac{1}{\gamma^2}\right)^2}, \frac{2\gamma}{\gamma^2} \right), & \text{if } p \text{ is horizontal Z-type} \\ \min \left(\sqrt{\frac{1}{\left(\frac{1}{\gamma^2} - 1\right)^2} + \left(\frac{1}{\gamma^2}\right)^2}, \frac{2\gamma}{\gamma^2} \right), & \text{if } p \text{ is vertical Z-type} \\ \min \left(\sqrt{\frac{1}{\left(\frac{1}{\gamma^2} - 1\right)^2} + \left(\frac{1}{\gamma^2}\right)^2}, \frac{2\gamma}{\gamma^2} \right), & \text{if } p \text{ is horizontal X-type} \\ \min \left(\sqrt{\frac{1}{\left(\frac{1}{\gamma^2} - 1\right)^2} + \left(\frac{1}{\gamma^2}\right)^2}, \frac{2\gamma}{\gamma^2} \right), & \text{if } p \text{ is vertical X-type} \\ b/(I - 1), & \text{if } p \text{ is I-type} \\ 2b/(I - 2)\sqrt{I + 2}, & \text{if } p \text{ is horizontal D-type} \\ 2b/(I - 2)\sqrt{I + 2}, & \text{if } p \text{ is vertical D-type} \end{cases}$$

4. With the values of d_f^p , choose which cases will be used to construct the coordinates. Positioning the first point on one of the rectangle's vertices, increment the coordinates of the other points using the information from p , f , and q_f^p .

Application pratique

Algorithme

Algorithm 3: Perturbation strategy

Input : Pattern p , N , Q , d_f^p , f and $c = (c_1^x, c_1^y, \dots, c_N^x, c_N^y)$ from Algorithm 2

Output : Perturbed $\hat{c}$

1. Select the coordinate to change, following the orientation of pattern p
 2. Enumerate the points, starting from the lower-left one to the end of the first line and following to lines $2, \dots, f$
 3. **for** $i = 1, \dots, N$ **do**
 4. If i is odd and belongs to lines 1 or f , then skip
 5. Compute $\xi \in [0, 1]$ following a uniform distribution
 6. Let $P = d_f^p \xi Q$
 7. **if** i is even and belongs to lines 1 or f **then**
 - Subtract (if $i = f$) or add (if $i = 1$) perturbation $P/2$ to the selected coordinate
 - else**
 - Subtract (if i is odd) or add (if i is even) perturbation P to the selected coordinate
-

Application pratique

Algorithme

1. We add the first two circles (these two form a trivial front).
2. We want to insert the i th circle. We find the circle in the front that is closest to our starting point. Suppose that this circle C_m . If there are more than one circles with the same distance, we pick the one that was inserted earliest.
3. We try to interpolate a circle between C_m and its successor in the front, C_n . Let us call this circle C_i .
4. If C_i intersects a circle of the front:
 - If C_i intersects a circle C_j that precedes C_m , we take out from the front and kill the circles from the successor of C_j up to and including C_m , we let C_j be the new C_m , and we return to step 3.
 - If C_i intersects a circle C_j that follows C_n , we take out from the front and kill the circles from C_n up to and including the predecessor of C_j , we let C_j be the new C_n , and we return to step 3.
5. At this point we have found a circle C_i that does not intersect any circle in the front. We then check whether it breaches the boundaries of our shape.
 - If it does, then the circle C_m that we picked in step 2 cannot be used to add a circle tangent to it inside the boundaries of our shape, so we must go back and try to find a different C_m . We put back in the front any circles that we removed in step 4. We resurrect any circles that we may have killed in step 4, apart from those that we have found in step 2 while trying to find the i th circle. We kill C_m and we return to step 2.
 - If it does not, we insert C_i in the front and we return to step 2 to insert the next circle ($i \leftarrow i + 1$), resurrecting all the circles in the front.