

Nom étudiant: Samuel Ogulluk

Numéro étudiant: 21501479

LU3PY126 FOAD

TP 3 Adsorption de particules sur une surface

Table des matières

1	Simulation de l'adsorption	3	6	Ajout de la surface et de la température à la simulation	9
2	Visualisation de la configuration finale	5	7	Comparaison des résultats obtenus	11
3	Reproduction de l'expérience	6	8	Compacité en fonction de la température	12
4	Comparaison au maximum de compacité	7	10	Augmentation de MAX_TRIES	13
5	Importance de la température	8	11	Augmentation maximale de MAX_TRIES	15

Introduction

Au cours de ce TP, nous nous intéresserons au phénomène de l'adsorption via des simulations numériques selon différents modèles.

Ainsi, nous verrons notamment les point suivants :

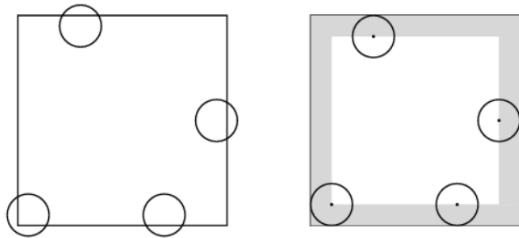
- Simulation de l'adsorption uniforme et sans mouvement.
- Simulation sur grille et avec température
- Etude et interprétation de l'effet de la température
- Étude de MAX_TRIES

Les codes présentés sont en C++ et Python pour la visualisation et sont joint à ce compte-rendu ainsi que disponibles dans le dépôt suivant : Dépôt Github

Question 1 Simulation de l'adsorption

Afin de simuler une adsorption, on vient définir les contraintes suivantes :

- Insertion de particules suivant une loi uniforme
- conditions aux limites (toute particule est strictement incluse dans la zone autorisée)



TP3/tp3.hpp

```
19 extern std::uniform_real_distribution<> dis;
20 inline double coord() {
21     return R + (L - 2.0 * R) * dis(gen);
22 }
```

TP3/q1q3homogene.cpp

```
8 int remplissage_homogene(int max_tries,
    ↪ std::vector<double>& x_out, std::vector<double>&
    ↪ y_out) {
9     x_out.clear();
10    y_out.clear();
11    x_out.reserve(static_cast<int>(L * L / (PI * R *
    ↪ R)));
12    y_out.reserve(static_cast<int>(L * L / (PI * R *
    ↪ R)));
13
14    int echecs = 0;
15    while (echecs < max_tries) {
16        double x_new = coord();
17        double y_new = coord();
18        if (!check_overlap(x_new, y_new, x_out,
    ↪ y_out)) {
19            x_out.push_back(x_new);
20            y_out.push_back(y_new);
21            echecs = 0;
22        } else {
23            echecs++;
24        }
25    }
26    return x_out.size();
27 }
```

TP3/q1q3homogene.cpp

```
44 void exercice_q1_q3() {
45     int M = 1000;
46     double total_particles = 0;
47     std::vector<double> x_final, y_final;
48
49     for (int i = 0; i < M; ++i) {
50         std::vector<double> x_temp, y_temp;
51         remplissage_homogene(MAX_TRIES_DEFAULT,
52                               ↪ x_temp, y_temp);
53         total_particles += x_temp.size();
54         if (i == M - 1) { x_final = x_temp; y_final =
55                               ↪ y_temp; }
56
57         double avg_particles = total_particles / M;
58         double eta = (avg_particles * PI * R * R) / (L *
59                               ↪ L);
60
61         save_results_q1_q3(avg_particles, eta, M, x_final,
62                               ↪ y_final);
63
64         std::cout << "Nombre de cercles: " <<
65                               ↪ x_final.size() << std::endl;
66         std::cout << "pourcentage de surface couverte
67                               ↪ (eta): " << eta << std::endl;
68     }
```

On obtient ainsi :

Nombre de particules	359
η	0.451

TABLE 1 – Résultats obtenus sur un tirage

On observe ainsi :

- Importance des optimisation du compilateur (-O2)
- on est loin de la borne supérieure de $N_{max} = \frac{L^2}{\pi R^2} \approx 800$
- Un seul lancé n'est pas représentatif de l'expérience.

Question 2 Visualisation de la configuration finale

- On vient ajouter un système d'enregistrement de la position de toutes les particules en .res
- Utilisation de Python et Matplotlib (et de la fonction `add_patch`) pour la visualisation

TP3/q1q3homogene.cpp

```
29 void save_results_q1_q3(double avg, double eta, int M,
   ↪ const std::vector<double>& x_final, const
   ↪ std::vector<double>& y_final) {
30     std::ofstream
   ↪ file("resultats/q1_q3_homogene.res");
31     file << "# MAX_TRIES=" << MAX_TRIES_DEFAULT << "
   ↪ M=" << M << std::endl;
32     file << "# Moyenne_particules Eta" << std::endl;
33     file << avg << " " << eta << std::endl;
34     file.close();
35
36     std::ofstream
   ↪ config_file("resultats/q1_q3_config.res");
37     config_file << "# x y R" << std::endl;
38     for (size_t i = 0; i < x_final.size(); ++i) {
39         config_file << x_final[i] << " " << y_final[i]
   ↪ << " " << R << std::endl;
40     }
41     config_file.close();
42 }
```

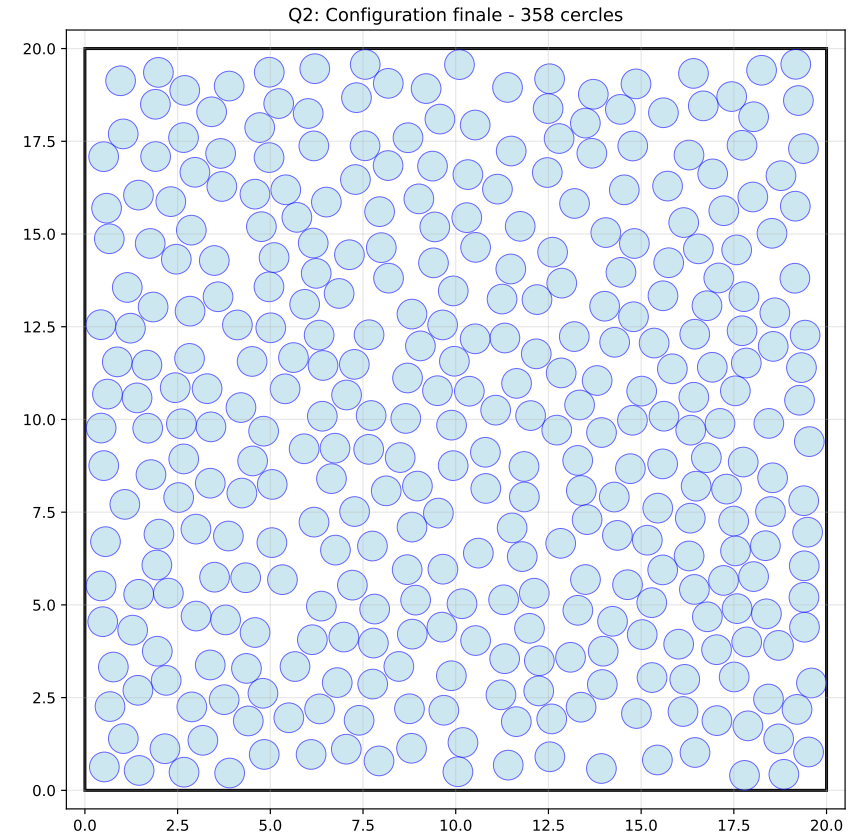


FIGURE 1 – Enter Caption

Question 3 Reproduction de l'expérience

On vient maintenant reproduire l'expérience 1000 fois à l'aide d'une boucle : **TP3/q1q3homogene.cpp**

```
45  int M = 1000;
46  double total_particles = 0;
47  std::vector<double> x_final, y_final;
48
49  for (int i = 0; i < M; ++i) {
50      std::vector<double> x_temp, y_temp;
51      remplissage_homogene(MAX_TRIES_DEFAULT,
52                          ↪ x_temp, y_temp);
53      total_particles += x_temp.size();
54      if (i == M - 1) { x_final = x_temp; y_final =
55                      ↪ y_temp; }
```

Moyenne de particule	372.882
η	0.468577

TABLE 2 – Résultats obtenus

On observe ainsi :

- Importance des optimisation du compilateur (-O2)
- on s'est rapproché de la borne $N_{max} = \frac{L^2}{\pi R^2} \approx 800$

Question 4 Comparaison au maximum de compacité

En moyenne, on est loin d'une compacité idéale (373 contre 795). On peut expliquer ceci par :

- Statistiquement, deux molécules n'ont pas de raison de se coller ou de s'arranger de manière compacte.
- On suppose une molécule collée comme ne bougeant plus et indéformable.
- Les effets de bords (surtout des coins) interviennent pour des bordures non infiniment larges
- chaque atome est supposé indépendant et sans interaction avec les autres
- Le placement aléatoire est inefficace car il crée rapidement des espaces vides trop petits pour accueillir de nouvelles particules.

Question 5 Importance de la température

On se place dans le cas d'un algorithme tel que les cercles préfèrent se placer près des nœuds du réseau, mais peuvent aussi s'adsorber ailleurs avec une probabilité qui dépend de la température T selon le critère de Metropolis : $p = e^{-U/T}$.

$T=0$:

- Les cercles ne peuvent s'adsorber qu'aux nœuds du réseau
- Distribution très ordonnée et régulière
- Les cercles forment un motif quasi-cristallin
- Densité η relativement faible

$T \rightarrow \infty$:

- $p \rightarrow 1$ partout donc répartition uniforme
- Distribution aléatoire uniforme
- Densité $\eta \rightarrow \eta_{max}$

On retrouve le modèle précédent.

Question 6 Ajout de la surface et de la température à la simulation

Pour prendre en compte la température, on ajoute une la fonction **TP3/q6q8temperature.cpp**

```
9 bool check_metropolis(double x, double y, double T) {
10     // Test interaction surface
11     double d = dist_latt(x, y);
12     if (d <= R_SURF) return true;
13     if (T == 0.0) return false;
14     return dis(gen) <= std::exp(-U / T);
15 }
```

- Permet de modéliser la probabilité d'adsorption
- utilise dist_latt qui modélise la distance à un point du réseau

TP3/q6q8temperature.cpp

```
15 }
16
17
18 int remplissage_temperature(double T, int max_tries,
    ↪ std::vector<double>& x_out, std::vector<double>&
    ↪ y_out) {
19     x_out.clear(); y_out.clear();
20     x_out.reserve(2500); y_out.reserve(2500);
21     int echecs = 0;
22     while (echecs < max_tries) {
23         double x_new = coord(); double y_new =
            ↪ coord();
24         if (!check_overlap(x_new, y_new, x_out, y_out)
            ↪ && check_metropolis(x_new, y_new, T)) {
25             x_out.push_back(x_new);
                ↪ y_out.push_back(y_new);
26             echecs = 0;
27         } else { echecs++; }
28     }
29     return x_out.size();
30 }
```

TP3/q6q8temperature.cpp

```

42 void process_temperature(double T, int M, int
    ↪ max_tries, std::ofstream& file, const
    ↪ std::vector<double>& T_save_list) {
43     std::vector<double> x_f, y_f;
44     double total = 0;
45     for (int i = 0; i < M; ++i) {
46         std::vector<double> x_t, y_t;
47         remplissage_temperature(T, max_tries, x_t,
            ↪ y_t);
48         total += x_t.size();
49         if (i == M - 1) { x_f = x_t; y_f = y_t; }
50     }
51     double eta = (total / M * PI * R * R) / (L * L);
52     file << std::fixed << std::setprecision(1) << T <<
        ↪ " " << std::setprecision(6) << eta <<
        ↪ std::endl;
53     for (double T_s : T_save_list) {
54         if (std::abs(T - T_s) < 0.01) {
            ↪ save_temp_config(T, x_f, y_f); break; }
55     }
56 }

```

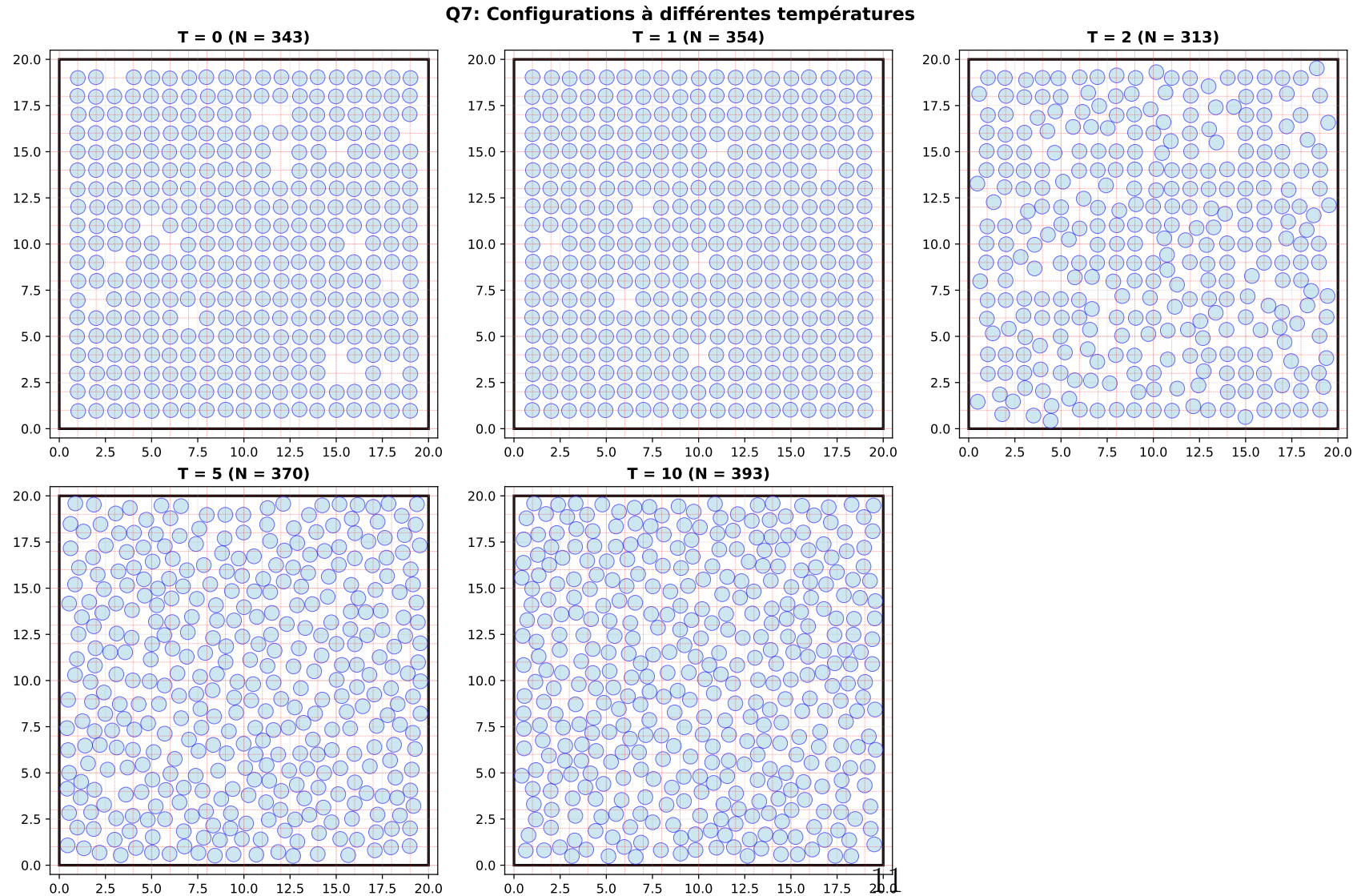
TP3/q6q8temperature.cpp

```

58 void exercice_q6_q8() {
59     int M = 100, max_tries = 10000;
60     std::ofstream
        ↪ file("resultats/q6_q8_temperature.res");
61     file << "# M=" << M << " max_tries=" << max_tries
        ↪ << "\n# Temperature Eta" << std::endl;
62     std::vector<double> T_to_save = {0.0, 1.0, 2.0,
        ↪ 5.0, 10.0};
63     for (double T = 0.0; T <= 10.0; T += 0.5) {
64         process_temperature(T, M, max_tries, file,
            ↪ T_to_save);
65     }
66     file.close();
67 }

```

Question 7 Comparaison des résultats obtenus



On observe :

- L'entropie croît avec la température
- $T=0 \rightarrow$ crystal
- η est corrélé négativement avec la température
- MAX_TRIES = 10000 ici

Question 8 Compacité en fonction de la température

A partir des résultats obtenus et à l'aide d'une boucle de $M=100$ pour chaque température, on trace $\eta = f(T)$
On observe :

- globalement η est corrélé positivement avec la température
- À basse température, les particules sont contraintes de s'aligner sur les sites du réseau cristallin, ce qui permet un empilement régulier.
- Minimum absolu à $T=2 \rightarrow$ quelques molécules empêchent l'accès à des noeuds du réseau $\rightarrow \min(\eta)$ on est entre le désordre et l'ordre

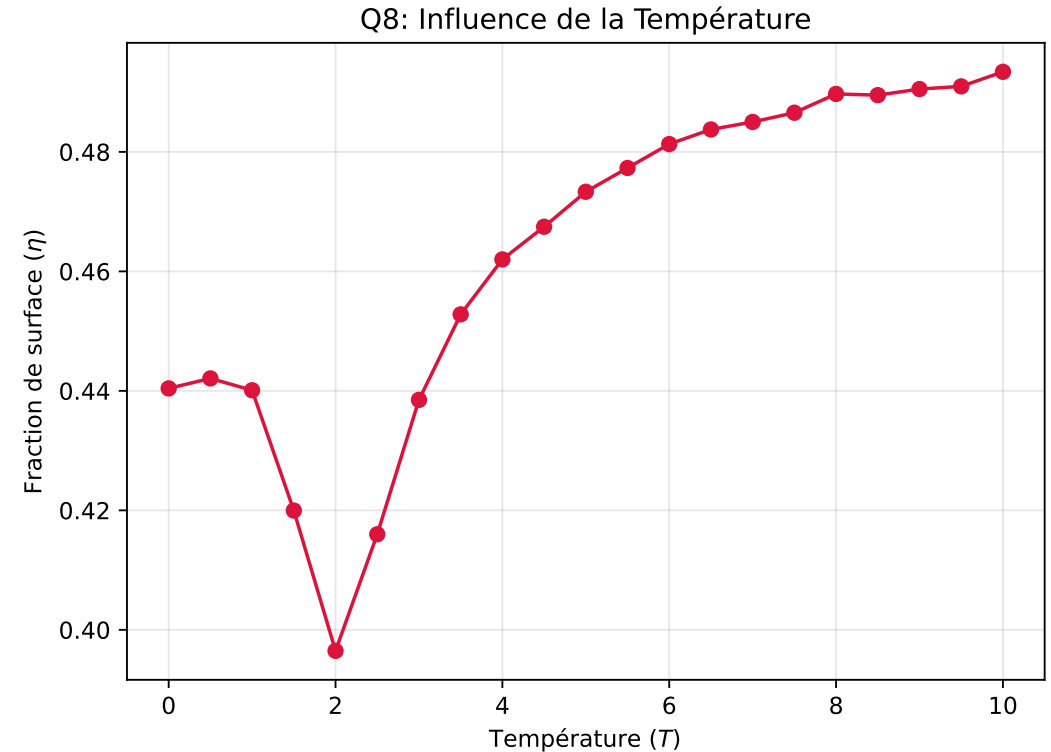


FIGURE 2 – $\eta = f(T)$

Question 10 Augmentation de MAX_TRIES

On s'intéresse maintenant au nombre d'itérations MAX_TRIES.

- L'augmentation du nombre d'échecs tolérés permet au programme de trouver les derniers interstices.
- il est exponentiellement plus difficile de placer les dernières particules en approchant de la limite asymptotique.

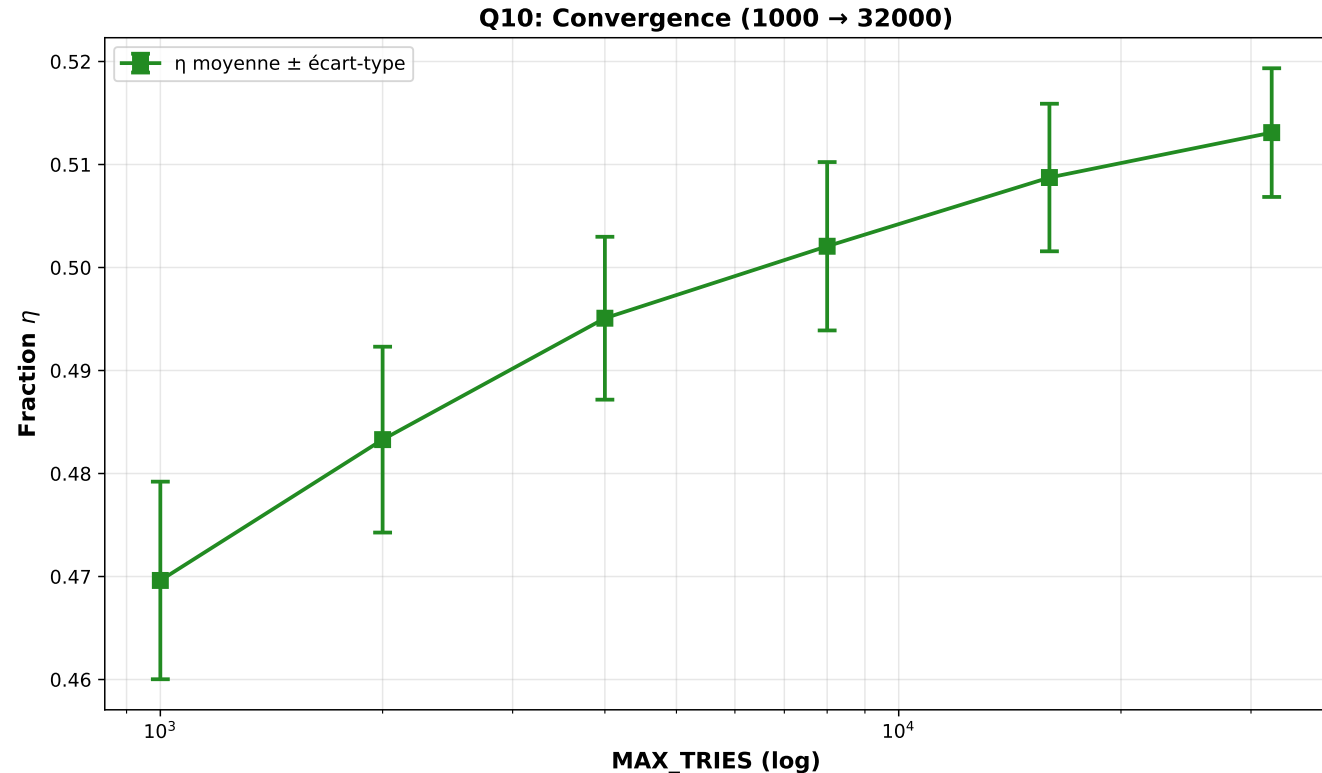


FIGURE 3 – $\eta = f(\text{MAX_TRIES})$

TP3/q10q11maxtries.cpp

```

9 int remplissage_convergence(int max_tries) {
10     std::vector<double> x_vec, y_vec;
11     x_vec.reserve(static_cast<int>(L * L / (PI * R *
12         ↪ R)));
13     y_vec.reserve(static_cast<int>(L * L / (PI * R *
14         ↪ R)));
15     int echecs = 0;
16     while (echecs < max_tries) {
17         double x_n = coord(), y_n = coord();
18         if (!check_overlap(x_n, y_n, x_vec, y_vec)) {
19             x_vec.push_back(x_n);
20             ↪ y_vec.push_back(y_n);
21             echecs = 0;
22         } else { echecs++; }
23     }
24     return x_vec.size();
25 }
26 void run_and_save_stats(int mt, int M, std::ofstream&
27     ↪ file) {
28     std::vector<double> eta_vals;
29     double sum = 0, var_sum = 0;

```

TP3/q10q11maxtries.cpp

```

29     for (int i = 0; i < M; ++i) {
30         double eta = (remplissage_convergence(mt) * PI
31             ↪ * R * R) / (L * L);
32         eta_vals.push_back(eta);
33         sum += eta;
34     }
35     double avg = sum / M;
36     for (double e : eta_vals) var_sum += (e - avg) *
37         ↪ (e - avg);
38     file << mt << " " << avg << " " <<
39         ↪ std::sqrt(var_sum / M) << std::endl;
40 }
41 void exercice_q10_q11() {
42     int M = 100;
43     std::ofstream
44         ↪ file("resultats/q10_q11_maxtries.res");
45     file << "# M=" << M << "\n# MAX_TRIES Eta_moyenne
46         ↪ Ecart_type" << std::endl;
47     for (int mt = 1000; mt <= 32000; mt *= 2)
48         ↪ run_and_save_stats(mt, M, file);
49     for (int mt = 64000; mt <= 512000; mt *= 2)
50         ↪ run_and_save_stats(mt, M, file);
51     file.close();
52 }

```

Question 11 Augmentation maximale de MAX_TRIES

On peut augmenter encore davantage MAX_TRIES On observe alors

- Converge vers η_{max} théorique.
- Convergence lente et logarithmique
- Pour MAX_TRIES élevée, - O2 devient indispensable

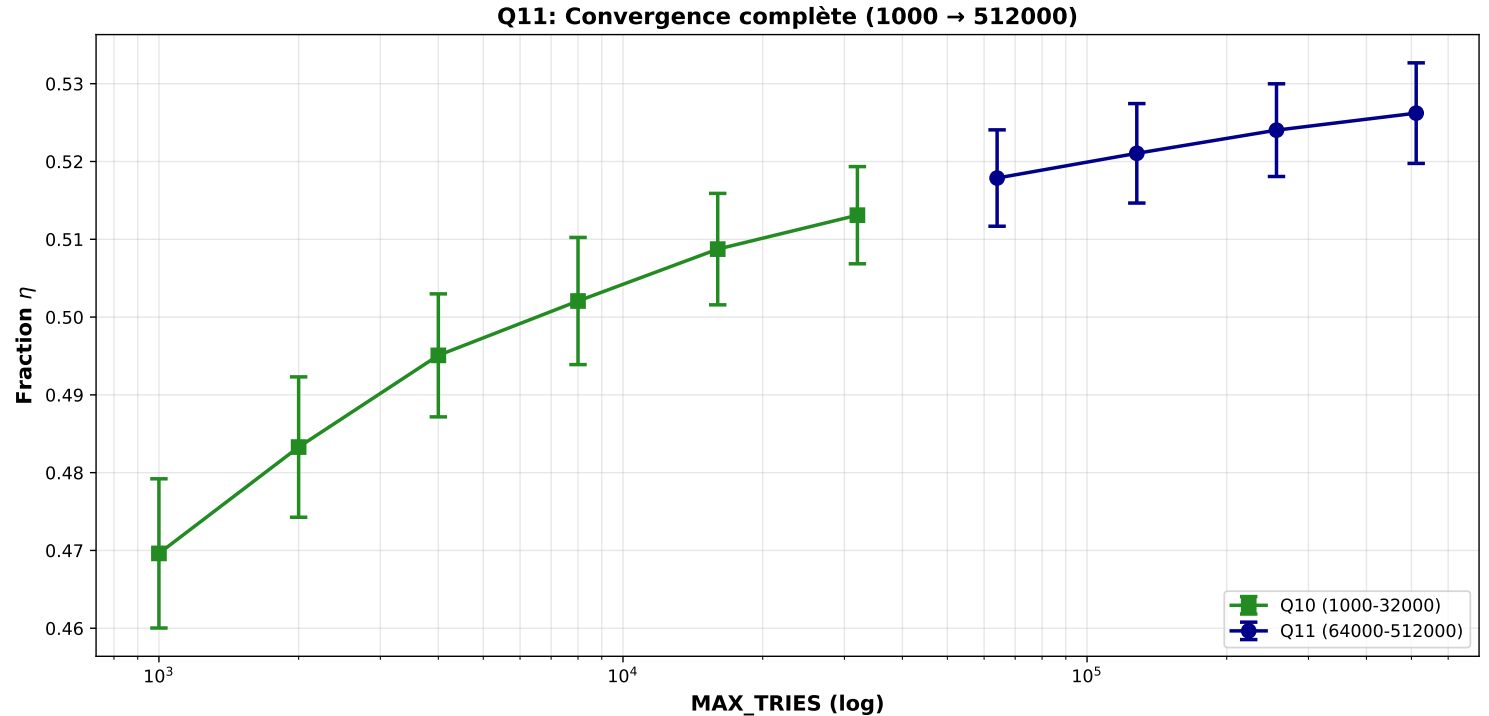


FIGURE 4 – $\eta = f(\text{MAX_TRIES étendu})$

Annexe

TP3/q1q3homogene.cpp

```
29 void save_results_q1_q3(double avg, double eta, int M, const std::vector<double>& x_final, const
   ↪ std::vector<double>& y_final) {
30     std::ofstream file("resultats/q1_q3_homogene.res");
31     file << "# MAX_TRIES=" << MAX_TRIES_DEFAULT << " M=" << M << std::endl;
32     file << "# Moyenne_particules_Eta" << std::endl;
33     file << avg << " " << eta << std::endl;
34     file.close();
35
36     std::ofstream config_file("resultats/q1_q3_config.res");
37     config_file << "# x y R" << std::endl;
38     for (size_t i = 0; i < x_final.size(); ++i) {
39         config_file << x_final[i] << " " << y_final[i] << " " << R << std::endl;
40     }
41     config_file.close();
42 }
```

TP3/q6q8temperature.cpp

```
32 void save_temp_config(double T, const std::vector<double>& x, const std::vector<double>& y) {
33     std::string filename = "resultats/q6_q8_config_T" + std::to_string(static_cast<int>(T)) + ".res";
34     std::ofstream config_file(filename);
35     config_file << "# x y R (T = " << T << ")" << std::endl;
36     for (size_t i = 0; i < x.size(); ++i) {
37         config_file << x[i] << " " << y[i] << " " << R << std::endl;
38     }
39     config_file.close();
40 }
```


TP3/visualisation.py

```

22 def q6():
23     """Q6: Influence de la température"""
24     try:
25         data =
            ↪ np.loadtxt(os.path.join(RES_DIR,
            ↪ "q6_q8_temperature.res"))
26         T, eta = data[:, 0], data[:, 1]
27
28         plt.figure()
29         plt.plot(T, eta, 'o-',
            ↪ color='crimson', markersize=6)
30         plt.xlabel('Température ($T$)')
31         plt.ylabel(r'Fraction de surface
            ↪ ($\eta$)')
32         plt.title('Q8: Influence de la
            ↪ Température')
33         plt.grid(True, alpha=0.3)
34         plt.tight_layout()
35         plt.savefig(os.path.join(FIG_DIR,
            ↪ "Q8_Temperature.pdf"), dpi=150,
            ↪ bbox_inches='tight')
36         plt.close()
37     except Exception as e:
38         print(f"Erreur Q6: {e}")

```

TP3/visualisation.py

```

80 def q10():
81     """Q10: Convergence MAX_TRIES (1000-32000)"""
82     try:
83         data = np.loadtxt(os.path.join(RES_DIR,
            ↪ "q10_q11_maxtries.res"))
84         max_tries, eta_mean = data[:, 0], data[:, 1]
85         eta_std = data[:, 2] if data.shape[1] >= 3 else
            ↪ np.zeros_like(eta_mean)
86         mask = max_tries <= 32000
87
88         fig, ax = plt.subplots(figsize=(10, 6))
89         ax.errorbar(max_tries[mask], eta_mean[mask],
            ↪ yerr=eta_std[mask],
90                     fmt='s-', color='green', markersize=8,
            ↪ linewidth=2,
91                     capsize=5, capthick=2, label=r'$\eta$ moyenne
            ↪ $\pm$ écart-type')
92         ax.set_xscale('log')
93         ax.set_xlabel('MAX_TRIES (log)', fontsize=12,
            ↪ fontweight='bold')
94         ax.set_ylabel(r'Fraction $\eta$', fontsize=12,
            ↪ fontweight='bold')
95         ax.set_title('Q10: Convergence (1000 → 32000)',
            ↪ fontsize=13, fontweight='bold')
96         ax.grid(True, alpha=0.3, which='both')
97         ax.legend()
98         plt.tight_layout()
99         plt.savefig(os.path.join(FIG_DIR, "Q10_Convergence.pdf"),
            ↪ dpi=150, bbox_inches='tight')
100        plt.close()

```

TP3/visualisation.py

```

40 def q7():
41     """Q7: Configurations à différentes
    ↪ températures"""
42     try:
43         temperatures = [0, 1, 2, 5, 10]
44         fig, axes = plt.subplots(2, 3, figsize=(15,
    ↪ 10))
45         axes = axes.flatten()
46
47         for idx, T in enumerate(temperatures):
48             ax = axes[idx]
49             try:
50                 data =
    ↪ np.loadtxt(os.path.join(RES_DIR,
    ↪ f"q6_q8_config_T{T}.res"))
51             if data.ndim == 1:
52                 data = data.reshape(1, -1)
53             x, y, R_data = data[:, 0], data[:, 1],
    ↪ data[:, 2]
54
55             ax.add_patch(Rectangle((0, 0), L, L,
    ↪ fill=False, edgecolor='black',
    ↪ linewidth=2))
56             for i in range(len(x)):
57                 ax.add_patch(Circle((x[i], y[i]),
    ↪ R_data[i], fill=True,
    ↪ alpha=0.6,
58
    ↪ edgecolor='blue',
    ↪ facecolor='lightblue',
    ↪ linewidth=0.5))

```

TP3/visualisation.py

```

59         for i in range(int(L) + 1):
60             ax.axhline(i, color='red',
    ↪ linewidth=0.3, alpha=0.3,
    ↪ linestyle='--')
61             ax.axvline(i, color='red',
    ↪ linewidth=0.3, alpha=0.3,
    ↪ linestyle='--')
62
63             ax.set_xlim(-0.5, L + 0.5)
64             ax.set_ylim(-0.5, L + 0.5)
65             ax.set_aspect('equal')
66             ax.set_title(f'T = {T} (N =
    ↪ {len(x)})', fontweight='bold')
        ax.grid(True, alpha=0.2)
    except:
        ax.text(0.5, 0.5, f'T =
    ↪ {T}\nmanquant', ha='center',
    ↪ va='center',
    ↪ transform=ax.transAxes)
67
68     fig.delaxes(axes[5])
69     plt.suptitle('Q7: Configurations à différentes
    ↪ températures', fontsize=14,
    ↪ fontweight='bold')
70     plt.tight_layout()
71     plt.savefig(os.path.join(FIG_DIR,
72         ↪ "Q7_Configurations_Temperature.pdf"),
73         ↪ dpi=150, bbox_inches='tight')
74     plt.close()
75     except Exception as e:
76         print(f"Erreur Q7: {e}")

```

TP3/visualisation.py

```
104 def q11():
105     """Q11: Convergence MAX_TRIES complète (jusqu'à 512000)"""
106     try:
107         data = np.loadtxt(os.path.join(RES_DIR, "q10_q11_maxtries.res"))
108         max_tries, eta_mean = data[:, 0], data[:, 1]
109         eta_std = data[:, 2] if data.shape[1] >= 3 else np.zeros_like(eta_mean)
110         mask_q10, mask_q11 = max_tries <= 32000, max_tries > 32000
111
112         fig, ax = plt.subplots(figsize=(12, 6))
113         ax.errorbar(max_tries[mask_q10], eta_mean[mask_q10], yerr=eta_std[mask_q10],
114                    fmt='s-', color='green', markersize=8, linewidth=2,
115                    capsize=5, capthick=2, label='Q10 (1000-32000)')
116         if np.any(mask_q11):
117             ax.errorbar(max_tries[mask_q11], eta_mean[mask_q11], yerr=eta_std[mask_q11],
118                        fmt='o-', color='darkblue', markersize=8, linewidth=2,
119                        capsize=5, capthick=2, label='Q11 (64000-512000)')
120         ax.set_xscale('log')
121         ax.set_xlabel('MAX_TRIES (log)', fontsize=12, fontweight='bold')
122         ax.set_ylabel(r'Fraction $\eta$', fontsize=12, fontweight='bold')
123         ax.set_title('Q11: Convergence complète (1000 → 512000)', fontsize=13, fontweight='bold')
124         ax.grid(True, alpha=0.3, which='both')
125         ax.legend(loc='lower right')
126         plt.tight_layout()
127         plt.savefig(os.path.join(FIG_DIR, "Q11_Convergence_Full.pdf"), dpi=150, bbox_inches='tight')
128         plt.close()
129     except Exception as e:
130         print(f"Erreur Q11: {e}")
```