Internship report M2 – 2023

A probabilistic approach to diffusion-mediated surface phenomena

Yilin YE

SUPERVISOR: Denis GREBENKOV - *Laboratoire de Physique de la Matière Condensée (UMR 7643), CNRS—École Polytechnique, IP Paris, 91128 Palaiseau, France*

Abstract
We concentrate on the diffusion inside a complicated environment, as the mimic of bioactive surface reactions. The 2D Koch boundary is studied for properties of the harmonic measure by numerical simulations based on “Geometry-adopted fast random walk” algorithm. Beyond the first passage problem, we model the reactive surface with “boundary local time”, for a random encounter times towards a successful reaction. For a particle near the border, the “reflection” approach is verified well for correct probability distributions.

Keywords
Diffusion, Complex geometry, Harmonic measure, Boundary local time, Reactivity

Introduction

The importance of diffusion in chemical reactions had been first recognized by von Smoluchowski in 1918 [1] and then diffusion-controlled reactions or, more generally, diffusion-mediated surface phenomena, became a broad field of intensive research, with examples ranging from oxygen capture by lung alveolar surface to heterogeneous catalysis, gene regulation, membrane permeation, and filtration processes (Fig. 1). The involved species (atoms, ions, molecules, and even whole organisms such as bacteria) have first to encounter each other, or to find a specific target or a substrate, to initiate a reaction. Most former works focused on the role of this first-passage step and its dependence on the geometric structure of the environment, on the size and shape of the target, and on the type of diffusive process. In turn, the role of surface reactions, occurring after the first arrival of a particle onto the target remained underestimated. In practice, one successful reaction event is generally preceded by a series of failed reaction attempts on the target, and thus involves multiple diffusive excursions in the bulk, whose statistical description is still missing. Herein, a novel probabilistic approach based on the concept of boundary local time has been recently proposed to investigate the intricate dynamics of diffusing particles near a reactive target [2]. This general paradigm allows us to describe sophisticated surface reactions controlled by random encounters between particles and the specific target, such as passivation of catalysts or activation processes in microbiology such as signaling in neurons, synaptic plasticity, cell differentiation and division. The disentanglement of the geometric structure of the medium from its surface reactivity opens far-reaching perspectives for modeling, optimization, and control of diffusion-mediated surface phenomena in nature and industry.

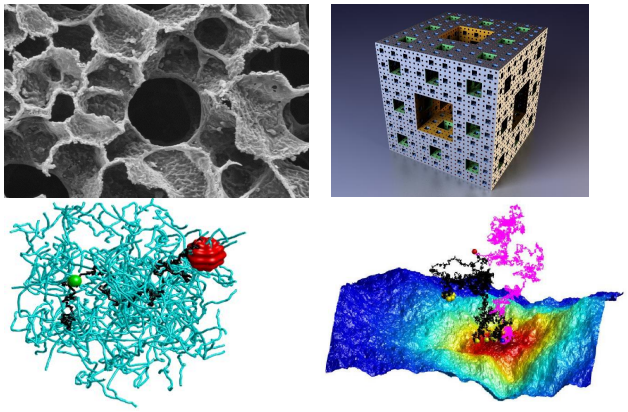

Figure 1. Top left: a cut of the pulmonary acinus exhibits a complex microstructure that determines the oxygen capture by blood and thus controls human respiration (credits to E. Weibel). Top right: the Menger sponge, a geometric model of a multiscale porous medium. Bottom left: a random trajectory of a bioactive molecule (in green) towards a protein (in red) through a network of actin filaments (in light blue). Bottom right: a particle diffusing near a reactive surface exhibits numerous encounters with that surface.

For the whole project, we aim at elaborating an original approach in these complementary directions below:
(1) The ordinary diffusion in the bulk will be replaced by a

more general stochastic processes such as continuous-time random walks, in order to make the description of bulk motion more realistic and closer to applications such as intracellular transport, which exhibits various anomalous features [3, 4].
(2) Known examples of the studied environments are limited to simple shapes such as a sphere, for which exact solutions are available; while more complicated confining media should therefore be explored by numerical methods in order to reveal the impact of their geometric structure onto diffusion-controlled reactions. This step should also open a possibility to address the optimization problem of constructing efficient structures under specific surface reaction constraints. This fundamental problem can find applications in pharmaceutical industry, like programmable drug release.
(3) While the original approach laid the mathematical foundation for dealing with new surface reaction mechanisms, their potential has not been explored. For instance, the successful reaction can be followed by the unbinding event to implement reversible bimolecular reactions and adsorption/desorption phenomena in porous media. Using the probabilistic construction of binding and unbinding events, one can investigate anomalous relaxation to an equilibrium or even non-equilibrium settings. The new approach also allows one to describe saturation, passivation or activation of the target via the so-called encounter-dependent reactivity.
(4) In many intracellular processes, the target has to accumulate a prescribed number of signaling particles (e.g., proteins or ions) to trigger a specific biological event such as apoptosis or signal transmission in neurons [5]. The probabilistic description of surface reactions allows one to investigate the collective effect of multiple independently diffusing particles that plays a crucial role for ensuring robust decision-making at the cellular level, but can be dramatically altered by diseases.

As for this internship, we focus on (1) and (2), developing theoretical and numerical aspects of diffusion-mediated surface phenomena by means of the original recently developed mathematical framework [2]. This interdisciplinary project applies mathematical, physical and numerical tools to uncover and model sophisticated microscopic mechanisms of surface reactions in chemical and biological systems, with eventual far-reaching applications in heterogeneous catalysis, neurosciences, medicine, and pharmaceutical industry.

To sum up, we mainly develop an efficient numerical method based on previous work, to incorporate the random walk or diffusion of particle into a complex boundary environment, including the first arrival and multi-arrival problem. Also, the concept of the boundary local time related to reactivity plays a paramount role. As the core strategy, the encounter-based approach is rather modified for special cases, apart from the the validation on simple geometry.

Theoretical Analysis

Diffusion-mediated surface phenomena

As a stochastic process, the diffusion of a particle could be described by its random walk. Inside an homogenous environment, this motion could be simply modeled by Langevin equation [8] with a constant friction coefficient ζ in eq. 1:

$$m\dot{v} = -\zeta v + \delta F \tag{1}$$

where m is the particle mass, v is its velocity, and δF refers to the random force of environment, which is related to thermal temperature. In general, we regard δF as a Gaussian white noise, and $\langle \delta F^2 \rangle \propto k_B T/m$. Additionally, for the overdamped case, we ignore the inertia and then express the displacement r as the integration of random force such that

$$\frac{\mathrm{d}r}{\mathrm{d}t} = \frac{\delta F}{\zeta} \qquad r = \int_0^t \frac{\delta F(\tau)}{\zeta}\mathrm{d}\tau \tag{2}$$

By Einstein's relation and Kubo's relation, a constant ζ can furnish a constant diffusion coefficient D in this case.

$$D = \frac{k_B T}{\zeta} = \lim_{t\to\infty} \frac{\langle \Delta r^2 \rangle}{2t} = \int_0^\infty \langle v(0)v(t)\rangle \,\mathrm{d}t \tag{3}$$

where Δr is the displacement in 1D, and t is the diffusion time. For higher dimension d, we have $D = \lim_{t\to\infty} \frac{\langle \Delta r^2 \rangle}{2dt}$.

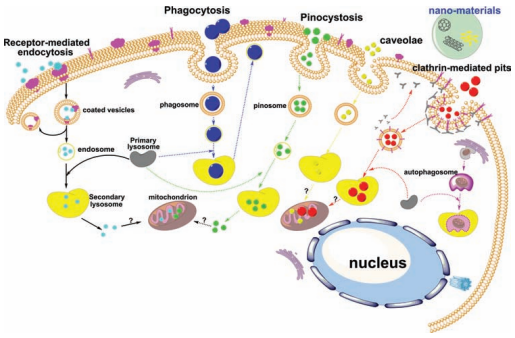

Figure 2. Known pathways for intracellular uptake of nanoparticles. [9]

In order to simulate the diffusion process inside a cell, we suppose that the internal environment keeps homogenous, and ignore the possible movement of cell membranes like exocytosis (Fig. 2). Indeed, near a soft surface, we should take its deformation into account [10, 11], for the lift force on particle's motion. For the sake of simplification, we regard all activities inside cell as four parts [12]: bulk diffusion, bulk reaction, surface diffusion, and surface reaction (Fig. 3), neglecting all interactions within cell organelles. As a complex domain, local information could be caught only if the particle approaches the membrane surface, which is similar to fractal surface. Therefore, we'd like to exploit the fractal boundary as the mimics of cell membranes, since it is rather simple to generate due to symmetry. In these circumstances, biological active molecules are regarded as simple point-like particles, while cell membranes are modeled by fractal borders, i.e. Koch snowflakes.

We consider a general approach to generate Koch snowflake with an arbitrary angle $\alpha \in (0, \pi)$ (Fig. 4). Suppose that the

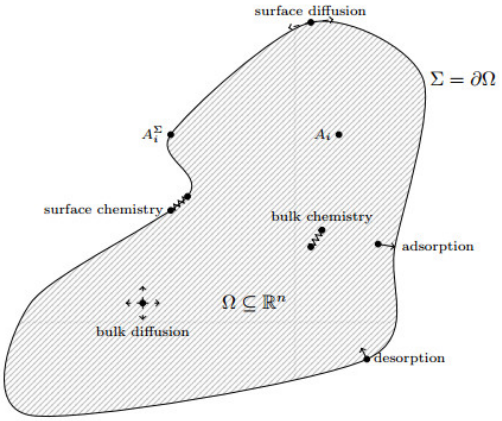

Figure 3. Physical and chemical mechanisms in a bulk-surface reaction diffusion system. [12]

length of original segment is L, we could obviously obtain equations below:

$$d^2 = l^2 + l^2 - 2l^2\cos\alpha \qquad d + 2l = L \tag{4}$$

and thus:

$$l = \frac{L}{\sqrt{2(1-\cos\alpha)}+2} \qquad d = \frac{\sqrt{2(1-\cos\alpha)}L}{\sqrt{2(1-\cos\alpha)}+2} \tag{5}$$

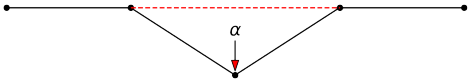

Figure 4. Portion of Koch snowflake with arbitrary angle. The length of four black lines is l, while the length of red line is d.

With these lengths known, we could generate Koch snowflake by these parameters: initial polygon shape, angle α, direction (concave or convex), and level or generation g. We do not care the size or the initial segment length L at the beginning. Then we nominate a snowflake such that "6_π/3+4" refers to a fractal configuration generated on hexagon, with angle $\alpha = \pi/3$, concave ($-$ for convex), up to the fourth level. Here are some example in Fig. 5.

In fact, particles could be inside or outside of the cell. Then considering the direction of our complex boundary such as convex or concave, there would be 4 possible cases for diffusion near a fractal domain (Fig. 6). During this internship, we only consider the first case where the particle is located inside the complex boundary, which is the classical Koch snowflake based on equilateral triangle with angle $\alpha = \pi/3$, like 3_π/3 − 4 or 3_π/3 − 2.

Reactivity and boundary local time

Generally, there would be many different acceptors situated on the cell membranes as the targets for surface diffusion

Figure 5. Examples of Koch snowflake. Top left: 4_π/4 − 3. Top right: 4_π/4 + 3. Bottom left: 5_π/5 − 3. Bottom right: 5_π/5 + 3.

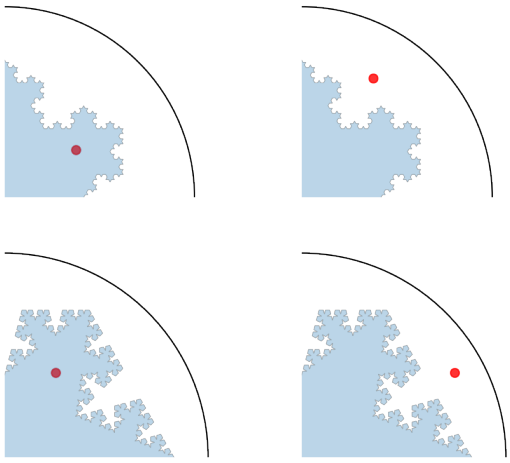

Figure 6. Possible cases for diffusion of a particle (in red) near Koch snowflake boundary (in blue) and circle domain (in black). Top left: Particle inside a convex boundary. Top right: Particle outside a convex boundary, but inside a large circle one. Bottom left: Particle inside a concave boundary. Bottom right: Particle outside a concave boundary, but inside a large circle one.

or surface reactions. However, particles near such a surface could not always find out the correct goal at their first arrivals, while a longer trajectory including several arrivals on other targets would be frequently observed in fact. Thus we need a variable to describe this procedure, for instance, how much time a particle spent in seeking an exact target near the border, or how many times a particle contacts the border. For each particle, it would enter the boundary layer from different positions, and thus leading to distinct time for reaching an exact target. Qualitatively, highly reactive surfaces would be teemed with all kinds of targets, which requires few or just only one contact for related reactions; while rather passivate surface owns limited targets for much longer time or more contacts to find the correct one.

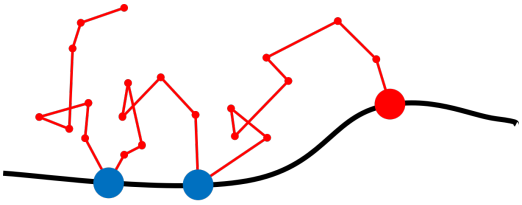

Figure 7. A possible Brownian motion trajectory of a particle (in red) near surface towards the correct target (in red), with two unsuccessful contacts with other targets (in blue) on the surface.

Following the paradigm [2], we consider the stop moment with the help of "boundary local time". Consider the Langevin equation for a particle with external force $\mathbf{F}$ near a boundary layer with thickness ε:

$$\mathrm{d}\mathbf{X}_t = \frac{D}{k_\mathrm{B}T}\mathbf{F}(\mathbf{X}_t)\mathrm{d}t + \sqrt{2D}\mathrm{d}\mathbf{W}_t \tag{6}$$

where $\mathbf{W}_t$ is a standard Brownian motion. In fact, $\mathbf{F}(\mathbf{x})$ should be non-zero only near the frontier, making the particle to move away from the boundary back to the interior, such as

$$\frac{\mathbf{F}(\mathbf{x})}{k_\mathrm{B}T} = \frac{1}{a}\mathbf{n}(\mathbf{x})\mathbb{I}_{\partial\Omega_\varepsilon}(\mathbf{x}) \tag{7}$$

where $\mathbf{n}(\mathbf{x})$ is the normal vector to the boundary $\partial\Omega$. We then introduce

$$\ell_t^\varepsilon = \frac{D}{\varepsilon}\int_0^t \mathrm{d}t'\mathbb{I}_{\partial\Omega_\varepsilon}(\mathbf{X}_{t'}) \tag{8}$$

and Langevin equation turns to

$$\mathrm{d}\mathbf{X}_t = \mathbf{n}(\mathbf{X}_t)\mathrm{d}\ell_t^\varepsilon + \sqrt{2D}\mathrm{d}\mathbf{W}_t \tag{9}$$

So $\frac{\varepsilon}{D}\ell_t^\varepsilon$ refers to the residence time of $\mathbf{X}_t$ inside the boundary layer $\partial\Omega_a$ up to time t. For thin layer limit, we denote the boundary local time ℓ_t as:

$$\ell_t = \lim_{\varepsilon\to 0}\ell_t^\varepsilon \tag{10}$$

Besides, the boundary local time ℓ_t can also be related to the number $\mathscr{N}_t^\varepsilon$ of crossings of the boundary layer $\partial\Omega_\varepsilon$.

$$\ell_t = \lim_{\varepsilon\to 0}\varepsilon\mathscr{N}_t^\varepsilon \tag{11}$$

On the partially reactive region $\partial\Omega_R$, we employ the Robin boundary condition

$$-D\partial_n c(\mathbf{x},t) = \kappa_0 c(\mathbf{x},t) \tag{12}$$

with κ_0 is the reactivity of the region. If the particle attempts to react independently on each encounter with probability $p \simeq \varepsilon\kappa_0/D \ll 1$, so the probability of no surface reaction up to the n-th encounter reads:

$$\mathbb{P} = 1 - \sum_{k=1}^{n} p(1-p)^{k-1} = (1-p)^n \simeq e^{-pn} \simeq e^{-q\ell} \tag{13}$$

where $q = \kappa_0/D$ refers to reactivity, and $\ell = n\varepsilon$.

Furthermore, due to the complex essence of our fractal border, we could hardly perform integration like eq. 8, or count number like eq. 11. Hence we use another rather local approach to calculate the boundary local time when one particle enters the boundary layer. Based on eq. 3, we have

$$\tau = \frac{\delta^2}{4D} \qquad t_{k+1} = t_k + \tau \qquad \ell_{t_{k+1}} = \ell_{t_k} + \sqrt{\frac{\pi}{2}D\tau} \tag{14}$$

where D is the diffusion coefficient, τ refers to diffusion time, and δ is the displacement. Still, we follow eq. 13 for

$$\Psi(\ell) = \mathbb{P}\left\{\ell < \hat{\ell}\right\} = \int_\ell^\infty \mathrm{d}\ell'\psi(\ell') = e^{-q\ell} \tag{15}$$

and then the distribution of the threshold $\hat{\ell}$ as $\psi(\hat{\ell}) = qe^{-q\hat{\ell}}$.

Numerical Simulations

A convex Koch snowflake with angle $\pi/3$ has been chosen as the complex environment. We consider particles at the center of this fractal domain, and simulate their diffusions towards boundary segments. Internal media is regarded homogenous without external potentials or forces, and particles can not leave the boundary. Indeed, we could hardly figure out the analytical solutions for final hitting probability distribution due to diffusion inside such a complex domain, and thus numerical simulations are strongly needed. To be exact, we consider two cases below:

- Continue the diffusion until the particle is attached on the border, such that the distance is less than a given value. See the subsection First arrival problem.
- Continue the diffusion until the particle has passed enough time near the frontier, such that the boundary local time ℓ_t is larger than a given random threshold $\hat{\ell}$. See the subsection Multi-arrival problem.

First arrival problem

The harmonic measure of a subset of the boundary of a bounded domain in n-dimensional Euclidean space $\mathbb{R}^n, n \geq 2$, is the probability that a Brownian motion starting inside a domain hits that subset of the boundary. In the complex plane, harmonic measure can be used to estimate the modulus of an analytic function inside a domain $\mathscr{D}$ given bounds on the modulus on the boundary of the domain.

Apart from few particular cases, the harmonic measure could not be derived analytically. Therefore, we seek it by numerical method, namely the distribution of p_k as the hitting probability on the k-th segment after particles' first arrival. Herein, the boundary is the convex Koch snowflake in the fourth generation with $\alpha = \pi/3$.

Markov-chain Monte Carlo (MCMC) method

In this overdamped case, the 2D Brownian motion could be expressed by two independent uniformly distributed random displacement variables (See Eq. 2), such that $\Delta x_i = \text{ran}(-\delta, +\delta)$ and $\Delta y_i = \text{ran}(-\delta, +\delta)$ with a constant δ. Indeed, this method would furnish varying radius for each diffusion step (Fig. 8).

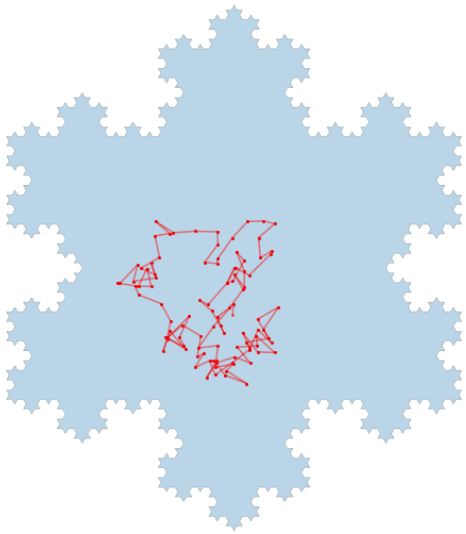

Figure 8. Brownian motion trajectory of a particle (in red) inside Koch snowflake domain (in blue) by MCMC method before reaching the boundary.

However, there would be some problem in practice. If the particle gets close to the fractal border in a given step, it would be highly probable that the particle jumps out of the boundary directly. Even though we could adjust Metropolis algorithm to draw back the outlier, the fractal boundary would make the corresponding verification much complex than a smooth surface. Moreover, for the sake of more precise result, a rather little value of δ would lead to much longer time for simulations.

To avoid this problem, we could modify the MCMC method such that $\Delta x_i = r\cos\theta_i$ and $\Delta y_i = r\sin\theta_i$, with only one random variable uniformly distributed as $\theta_i = \text{ran}(0, 2\pi)$, and a constant radius r (Fig. 9). Once we select r less than the segment length, we still meet the same problem such that a precise result is quite time-consuming.

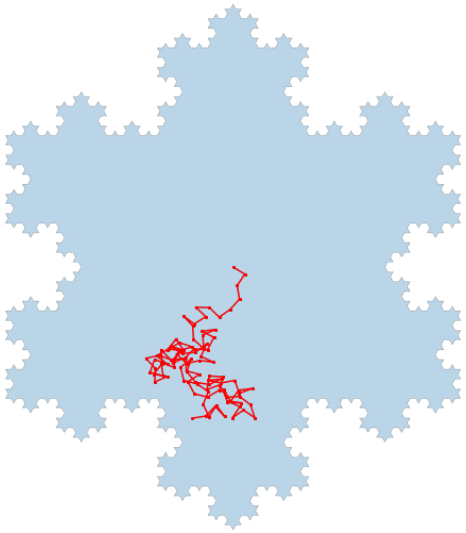

Figure 9. Brownian motion trajectory of a particle (in red) inside Koch snowflake domain (in blue) by modified MCMC method before reaching the boundary.

Geometry-adopted fast random walk

Based on the discussion above, we really need a method efficient enough, which could furnish a proper diffusion radius for precise results without particles leaving the domain. In fact, the core problem is just the diffusion near the fractal boundary, while the bulk diffusion near center would meet no limitation. Therefore, to accelerate our numerical simulations by diminishing the real time, or numerical steps spent in the middle due to little value of r, we are inclined to vary this radius according to the nearby surroundings, i.e. smaller value of r while the particle approaches the border.

By "Geometry-adopted fast random walk" algorithm [13], we step further based on modified MCMC toward a varying radius r_i, such that $\Delta x_i = r_i\cos\theta_i$ and $\Delta y_i = r_i\sin\theta_i$ as $\theta_i = \text{ran}(0, 2\pi)$. We calculate r_i for each step according to one particle's provisional location, and then jump it to one position located on this circle uniformly (Fig. 10). Except the first jump from center (in red), we always take the minimal distance between particle and frontier as the maximal radius, shown in cyan, green, purple, and black in order. Once the radius is less than a given value, say $\varepsilon = 10^{-3}L$ for L is segment length, we think that the particle is attached, and then stop the relevant simulation.

This approach could be valid since Brownian motion is continuous on time. Based on eq. 3, given the time t, we

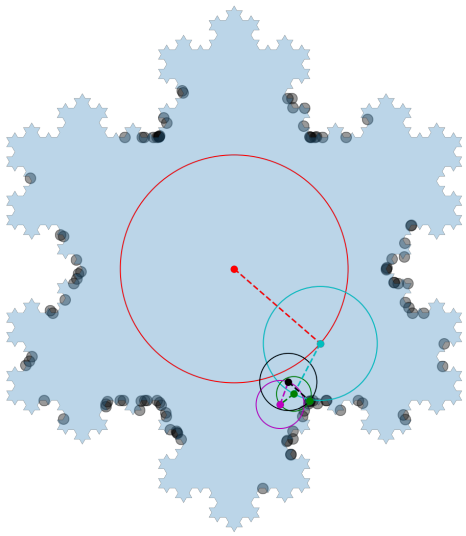

Figure 10. Brownian motion trajectory of a particle (in red) inside Koch snowflake domain (in blue, $\alpha = \pi/3, g = 4$) by GAFRW algorithm. Final distribution after the first arrival on the boundary (in grey, 100 particles shown).

obtain an exact displacement square $\langle \Delta r^2 \rangle$. For many-particle approximation, we ignore the average and take $r^2 \simeq 4Dt$. By rotational symmetry, there should be a uniformly distribution on this circle after a given diffusion time t. Simply, we replace one Brownian motion trajectory inside a circle by a random jump from center to a uniformly distributed point on the boundary. Repeat this process until the particle arrives on the boundary, we accelerate simulations with varying radius.

For this GAFRW algorithm, the most difficult part is to compute the distance between a given particle and the whole complex frontier. It is not necessary to search among all boundary segments. Briefly speaking, we record two nearest points in the level $g-1$, and only search segments inside this region in the level g for a proper radius r_i. For more details, see Appendix.

Hitting probability distribution

With GAFRW algorithm in hand, we could easily realize the diffusion inside a complex frontier, especially the fractal one. Then, supposing no interaction among particles, we do simulations for $N = 10^6$ particles to obtain a good hitting probability distribution result. As we see (Fig. 11), there would be higher probability for segments closer to center. Also, due to the domain's symmetry, we see periodic peaks. In Appendix, there are two additional figures for harmonic measure for simpler frontier, i.e. $3\_\pi/3-3$ and $3\_\pi/3-2$.

Multi-arrival problem

In this subsection, we are still interested in the harmonic measure, but with not only one contact. So we should continue numerical simulations even after the first arrival. We do not know when one particle would find out the right target in reality, and there could be several possible choices to stop our numerical simulations. For example:

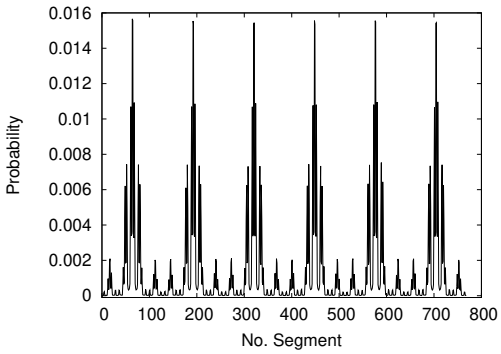

Figure 11. Hitting probability distribution as the function of segment index in fractal boundary. $g = 4, N = 10^6$

- Stop the simulation after a given time T.
- Stop the simulation after a random time threshold δ, such that $\mathbb{P}(\delta > t) = e^{-pt}$, where t is time passed for diffusion.
- Stop the simulation after the boundary local time ℓ_t exceeds a random threshold $\hat{\ell}$

The third one is chosen in practice with the same probability shown in eq. 15: $\mathbb{P}\{\ell < \hat{\ell}\} = e^{-q\ell}$. Thus local time threshold is distributed by $\psi(\hat{\ell}) = qe^{-q\hat{\ell}}$, where q is reactivity; and then $\hat{\ell} = -\ln[\text{ran}(0,1)]/q$.

Encounter-based reflection

Still, we insist GAFRW algorithm, such as $\vec{x}_{k+1} = \vec{x}_k + \rho(\cos\theta_k, \sin\theta_k)$ in Fig. 12, where ρ refers to radius. However, after entering the boundary layer $\partial\Omega_\varepsilon$, we would not stop simulation but consider the boundary local time computed in eq. 14. Exactly, since particles could not leave the domain, and it is almost impossible for one particle to attach on the border (i.e. with zero distance), we employ the "reflection" method to treat this case, with following steps:

- Take a ρ proportional to the layer thickness ε.
- Execute GAFRW algorithm with this given ρ.
- Once the new position is out of the domain, we **reflect** this position with boundary $\partial\Omega$ for another location inside the domain.

Only after the first reflection, namely the first arrival, we start to accumulate the boundary local time ℓ_t, until $\ell_t \geq \hat{\ell}$.

Verification on circle

Before the fractal surface, we should verify the reflection method. Here, we take a simple domain, a circle with radius $R = 1$, and we put particles at $(r_0, 0)$ initially. For a circle

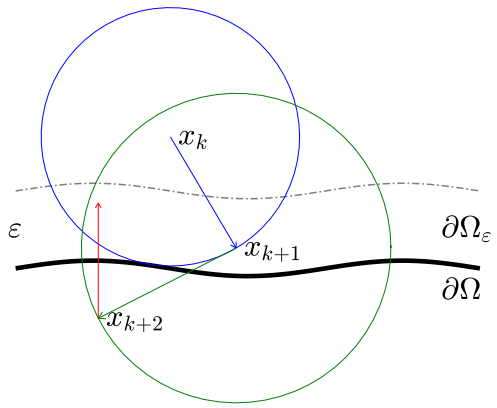

Figure 12. Illustration of several step of the walk-on-spheres algorithm [14], with real boundary $\partial\Omega$ and virtual layer $\partial\Omega_\varepsilon$. At x_k, we take one circle tangential to surface; while at x_{k+1}, we take $\rho \propto \varepsilon$. Once the particle leaves $\partial\Omega$ (at x_{k+2} on green circle), we exploit reflection backwards (in red arrow).

domain, we approximate it by regular N-gon, and its analytical solution of harmonic measure reads:

$$p_k = \frac{1}{NR} + \sum_{j=1}^{\infty} \frac{2qr_0^j}{\pi j(j+q)} \sin\left(\frac{\pi j}{N}\right) \cos\left(\frac{2\pi j}{N}(k-\frac{1}{2})\right) pkexp \tag{16}$$

where q is reactivity mentioned previously, and $\psi(\hat{\ell}) = qe^{-q\hat{\ell}}$.

A priori, we need a reasonable ratio for $\rho \propto \varepsilon$. Below we test several choices (Fig. 13), demonstrating that we should take $\rho = 2\varepsilon$. Once we take other ratios, there would be errors to some extent.

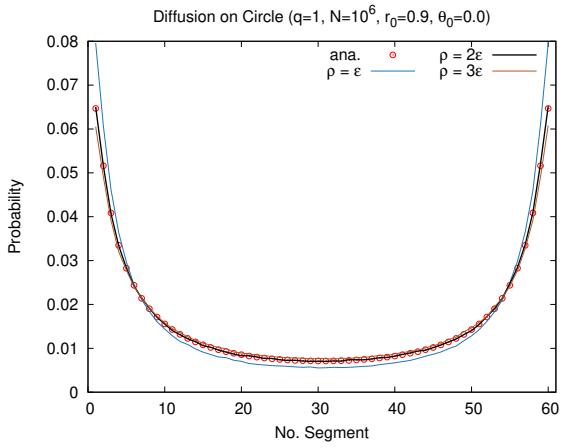

Figure 13. Comparison of harmonic measure on a circle domain (expressed by regular 60-gon) between the analytical expectation (in red points) and numerical one (in color lines). Only in $\rho = 2\varepsilon$ we see perfect correspondence.

After our verification that ρ should be equal to 2ε, we step further towards different cases. In Fig. 14, a typical trajectory is shown with initial position at $(r_0 = 0.9, \theta = 0)$ and reactivity $q = 1$.

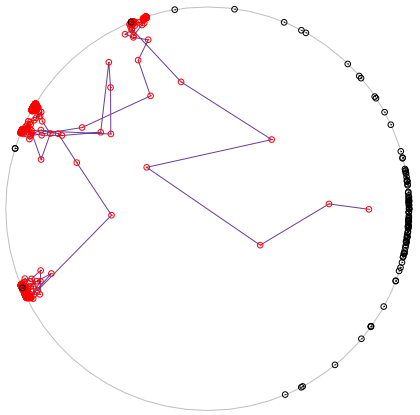

Figure 14. Trajectory (in violet, red points for a series positions) inside a circle domain $r = 1$ with reflection method with initial position $r_0 = 0.9, \theta = 0$. Final distribution after several arrivals ($q = 1$) on the surface (in black, 100 particles shown).

Also we test this method for different cases, with constant reactivity q but varying initial positions (Fig. 15), or with the same initial position but varying reactivity (Fig. 16).

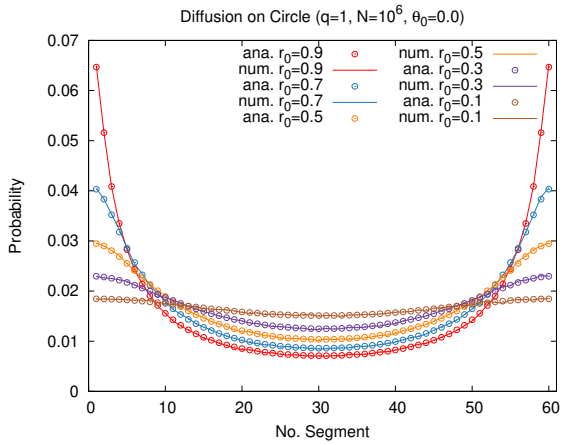

Figure 15. Harmonic measure in a circle domain with radius $R = 1$ under different initial position and a constant reactivity $q = 1$. Analytical expectations by eq. **??** are shown in dots, while numerical results are drawn in lines.

Practice on fractal domain

Then we practice the reflection method on complex border. Still, we take the Koch snowflake like $3\_\pi/3-2$. With a rather large domain, such that the length of equilateral triangular $L = 10^3$, we exam that once the boundary layer thickness ε is much less that border segment length L_g in g-th level, i.e.

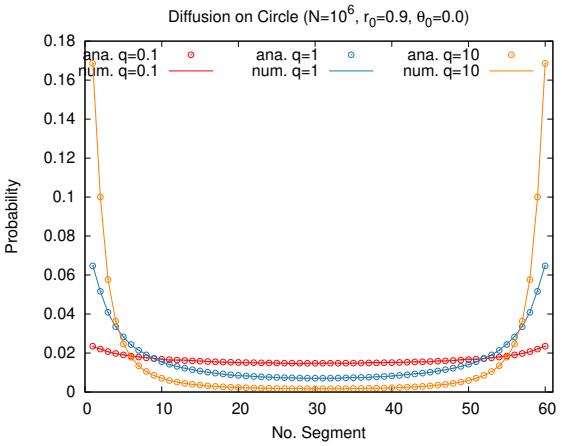

Figure 16. Harmonic measure in a circle domain with radius $R = 1$ under a constant initial position ($r_0 = 0.9$) and different reactivities. Analytical expectations by eq. **??** are shown in dots, while numerical results are drawn in lines.

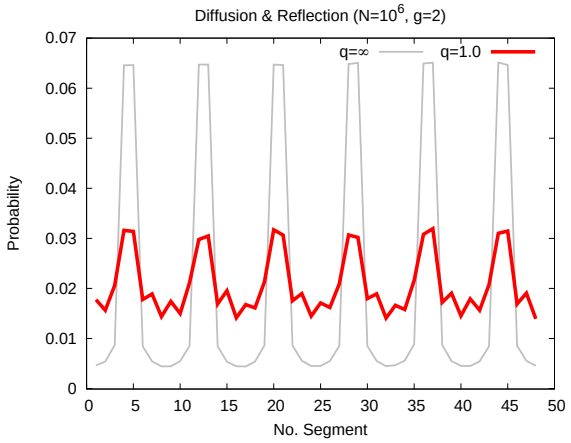

Figure 18. Comparison of harmonic measure inside the fractal domain $3\_\pi/3-2$ with $L = 2$. The first-arrival result is shown in gray, while the other is derived by reactivity $q = 1$.

$\varepsilon \ll L_g = L/3^g$, there would be no significant difference for harmonic measure. (Fig. 17)

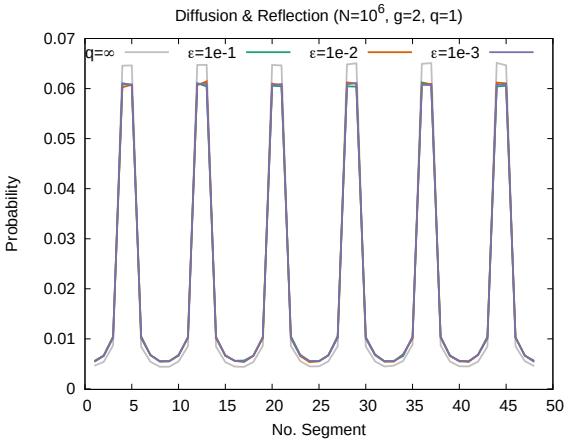

Figure 17. Comparison of harmonic measure inside the fractal domain $3\_\pi/3-2$ with $L = 10^3$. The first-arrival result is shown in gray, while others cases are derived by reactivity $q = 1$.

Since $q = \kappa_0/D$, we should take qL into account together as a dimensionless parameter. Note that the domain size would affect harmonic measure under a constant q. Invariant result means the equal qL. For little qL, namely larger domain size or larger boundary local time threshold, there would be highly probable that the particle would be trapped in a corner. Therefore, the most difficult part is to define well the reflection rule near two types of corner: one for $\pi/3$, the other for $4\pi/3$. In practice, *pending to smooth description and well written.*

Conclusion and Perspectives

During this short internship from the beginning of April to July, a rather complete work has been achieved at PMC, on the "A probabilistic approach to diffusion-mediated surface phenomena". Through various numerical simulations by "geometry-adapted fast random walk" technique, we have figured out the harmonic measure inside a complex environment. Based on a Koch snowflake fractal boundary, we consider the diffusion with the first arrival condition, and then introduce the boundary local time for more arrivals, as a mimic of "target-finding" process, which play a significant role for biochemical reactions. As the function of several parameters, probability distributions have also been computed clearly for both two cases. Further research would be continued, focusing on different boundary conditions and reversible chemical reactions, and even surfaces in 3D.

While in the preliminary stage, we focus on developing theoretical and numerical aspects of the aforementioned phenomena, the ultimate goal consists in discovering applications of this framework in chemistry and biophysics. In this light, tremendous progress in optical microscopy over the last two decades opens unprecedented opportunities to follow the random trajectory of a single molecule and therefore to access experimentally the statistics of its encounters and interactions with the surface [6, 7]. The increasing number of single-particle experimental data can thus help to reveal limitations of conventional approaches and to propose more accurate models of surface reactions. This ambitious goal requires elaborated statistical tools to characterize binding and unbinding events on the surface and to infer their statistics from a limited number of available noisy trajectories. While the development and applications of such tools to experimental data is a long-term project, the progress in understanding sophisticated surface reactions achieved later will prepare the

theoretical ground and synthetic datasets of single-particle trajectories for future research.

Personally, I have practiced well my coding skill in such a short time range. Numerical simulations are driven by *Fortran* and *Python*, while figures are drawn by *Python* and *gnuplot*. To step further, the subsequent research will perhaps be concentrated on a boundary in 3D rather than a 2D one, or even that with an irregular deformation. As for the soft surface, there exist numerous examples in the nature of such biological membranes.

Acknowledgments

Teemed with appreciation and gratitude, I would like to thank my supervisor Denis GREBENKOV at first, for his careful pedagogy, precise advice, and patient instruction during my whole internship. The prompt discussion could always give me effective feedback, not only the profile about the whole project, but also technical details in codes. Also, I thank Adrien CHAIGNEAU, the first-year PhD in our group, for his great insight on mathematical details, inspiring discussions, as well as his work on harmonic measure for necessary comparisons.

From April to present, I have been quite immersed in PMC, especially the warm group lunch. Furthermore, thanks to Anne-Marie DUJARDIN of PMC, École Polytechnique; as well as Médina MAHREZ of ENS for their generous help on administrative issues. In addition, we acknowledge the financial supports from CNRS.

References

[1] M. von Smoluchowski, *Ann. Phys.* **1915**, *48*, 1103–1112, **Über Brownsche Molekularbewegung unter Einwirkung äusserer Kräfte und deren Zusammenhang mit der verallgemeinerten Diffusionsgleichung**.

[2] D. S. Grebenkov, *Phys. Rev. Lett.* **2020**, *125(7)*, 078102, **Paradigm shift in diffusion-mediated surface phenomena**.

[3] Y. Lanoiselée, N. Moutal, and D. S. Grebenkov, *Nat. Commun.* **2018**, *9*, 4398, **Diffusion-limited reactions in dynamic heterogeneous media**.

[4] P. Witzel, M. Götz, Y. Lanoiselée, T. Franosch, D. S. Grebenkov, and D. Heinrich, *Biophys. J.* **2019**, *117*, 203-213, **heterogeneities shape passive intracellular transport**.

[5] M. Reva, D. A. DiGregorio, and D. S. Grebenkov, *Sci. Rep.* **2021**, *11*, 5377, **A first-passage approach to diffusion- influenced reversible binding: insights into nanoscale signaling at the presynapse**.

[6] D. Wang, H. Wu, and D. K. Schwartz, *Phys. Rev. Lett.* **2017**, *119*, 268001, **three-dimensional tracking of interfacial hopping diffusion**.

[7] T. Sungkaworn, M.-L. Jobin, K. Burnecki, A. Weron, M. J. Lohse, and D. Calebiro, *Nature*. **2017**, *550*, 543-547, **Single- molecule imaging reveals receptor-G protein interactions at cell surface hot spots**.

[8] P. Langevin, *Compt. Rendus* **1908**, *146*, 530-533, **Sur la théorie du mouvement brownien**.

[9] F. Zhao, Y. Zhao, Y. Liu, X. Chang, C. Chen, and Y. Zhao, *Small*. **2011**, *7(10)*, 1322-1337, **Cellular uptake, intracellular trafficking, and cytotoxicity of nanomaterials**.

[10] T. Salez, L. Mahadevan, *J. Fluid Mech.* **2015**, *779*, 181-196, **Elastohydrodynamics of a sliding, spinning and sedimenting cylinder near a soft wall**.

[11] V. Bertin, Y. Amarouchene, E. Raphael, and T. Salez, *J. Fluid Mech.* **2022**, *933*, A23, **Soft-lubrication interactions between a rigid sphere and an elastic wall**.

[12] B. Augner, and D. Bothe, *arXiv:1911.13030*. **2019**, **The fast-sorption–fast-surface-reaction limit of a heterogeneous catalysis model**.

[13] D. S. Grebenkov, A. A. Lebedev, M. Filoche, and B. Sapoval, *Phys. Rev. E*. **2005**, *71(5)*, 056121, **Multifractal properties of the harmonic measure on Koch boundaries in two and three dimensions**.

[14] Y. Zhou, W. Cai, and E. Hsu, *Commun. Math. Sci.* **2017**, *15(1)*, 237-259, **Computation of the local time of reflecting brownian motion and the probabilistic representation of the Neumann problem**.

Appendix

First arrival problem

Hitting probability distribution

Below are two figures about harmonic measure on the second or third generation of Koch snowflake.

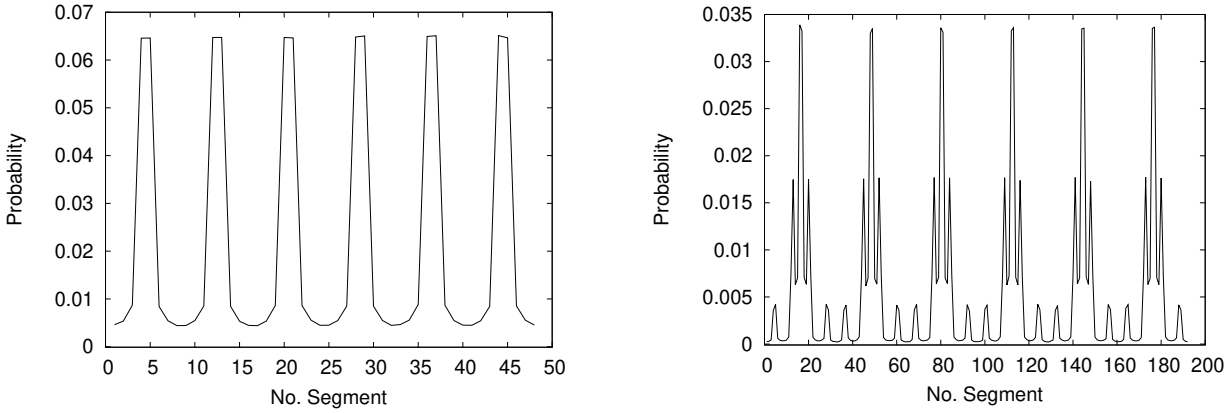

Figure 19. Left: Hitting probability distribution for $3\_\pi/3-2$. Right: Hitting probability distribution for $3\_\pi/3-3$.

To realized GAFRW algorithm, we exploit several methods in practice. The core strategy is finding the maximum distance for diffusion in each step, such that the particle would not quit the domain. Therefore, it is necessary to search enough boundary points and figure out the radius for each step. Below are typical ideas:

- **Alg 1**. Whole space searching (1) Calculate distances between the particle and all boundary segments and all boundary points at each step. (2) Determine the radius as the minimum one of all distances for uniform diffusion by GAFRW. (3) Repeat this process until the particle is attached on the boundary, with only one or several arrivals.
  This method could assure that we select the **correct** radius value for GAFRW; however, it is not efficient enough since the searching time is proportional to 4^g for the first arrival problem, and even much longer for other cases. Thus we only used this algorithm for verification.

- **Alg 3**. Partial space searching (1) Determine the searching range in g according to endpoints in $g-1$. (2) Calculate distances between the particle and all boundary segments and all boundary points inside a given range in generation g determined previously. (3) Determine the radius as the minimum one of all distances obtained above for uniform diffusion by GAFRW. (4) If the particle quits the known range after the diffusion, update endpoints in each level. (5) If the two nearest points form the unit segment in g, we explore search inside this region for next generation $g+1$. (6) Repeat the whole process until the maximal generation $g_{\max}$. (7) Stop the simulation if the particle is attached on the boundary, or the local time exceeds the threshold.
  This method is much more **efficient**, with searching time almost linear to g. Thus we used this algorithm for almost all simulations, both the first arrival and multi arrival problem.

- **Alg 6**. Generation-wise diffusion. (1) Search all boundary points in the first generation and find the diffusion radius. (2) Repeat the previous step until the particle is attached on the boundary. (3) Consider the next generation of boundary for following diffusions. (4) Particle in $g+1$ could not quit the old visual boundary in g. (5) Stop the simulation if the particle is attached on the boundary in generation $g_{\max}$.
  This method is efficient; however, it would **not** furnish the correct results as Alg 1. (See Fig. 20) Indeed, there would be little probability for one particle quit the visual boundary and towards other segments.

Indeed, there is no analytical solution for fractal boundary like Koch snowflake, such that we sum up all probabilities on segments in sub-leading generation $g+1$ as the value of the segment in leading generation g. See Fig. 20, we did numerical simulations for the first arrival case inside the Koch snowflake domain with $g=2$ with all threes algorithms mentioned above. Alg 6 (in light blue) could not reach the peak as Alg 1 (in black) and Alg 3 (in red). Therefore, we realized following simulations mainly based on “Alg 3” for efficiency.

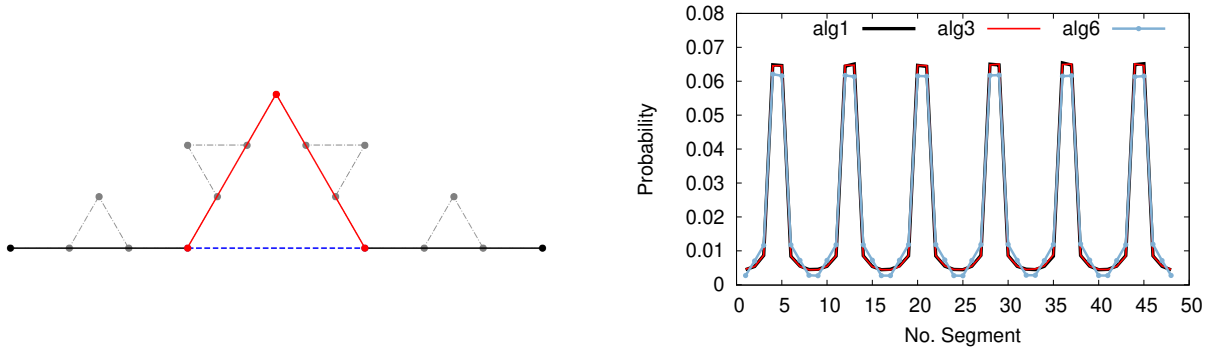

Figure 20. Left: Part of fractal boundary. The hitting probability on blue dashed segment is not equal to sum of them on two red ones. Right: Comparison of hitting probability distribution from different algorithms.

Multi-arrival problem

Random threshold

For the continuous distribution $\pi(x)$, we have the cumulative distribution as: $\Pi(x) = \Pi(x - \mathrm{d}x) + \pi(x)\mathrm{d}x = \int_{-\infty}^{x} \pi(x)\mathrm{d}x$. Working with normalized $\pi(x)$, the possible value of Π, which we call Υ, is a uniform distribution on (0,1). Let Π^{-1} be the inverse function of Π, then the random number $x = \Pi^{-1}(\Upsilon)$ is distributed as $\pi(x)$. The tricky step is usually to find Π^{-1}.

In our case, the local time threshold is a function of reactivity q, with the distribution as $\psi(\ell) = qe^{-q\ell}$, then we compute the cumulative distribution

$$\Pi(\ell) = \int_0^{\ell} \psi(x)\mathrm{d}x = 1 - e^{-q\ell} = \Upsilon = \mathrm{ran}(0,1) \tag{A1}$$

Then $\ell = \Pi^{-1}(\Upsilon) = -\ln(1-\Upsilon)/q$. Since both Υ and $1-\Upsilon$ are ran(0,1), we have the random threshold as

$$\hat{\ell} = -\frac{1}{q}\ln[\mathrm{ran}(0,1)] \tag{A2}$$

Analytical probability distribution on circle

Inside a circle domain with radius R and centre at origin, we consider a particle with initial position (r_0, θ_0) and its diffusion. Define the probability density ω_q such that a particle is attached on the circle (R, θ) finally.

$$\omega_q(\theta|r_0,\theta_0) = \frac{1}{2\pi R} \times \left\{ 1 + 2\sum_{j=1}^{\infty} \left(\frac{r_0}{R}\right)^j \frac{\cos[j(\theta-\theta_0)]}{1+\frac{j}{qR}} \right\} \tag{A3}$$

There is no segment for continuous θ, so we divide the circle into N arcs in numerical practice, such that $\theta \in [k-1,k] \times \frac{2\pi}{N}$, with $k = 1,2,3,...,N$. The hitting probability p_k on k-th segment is thus expressed as the integration of density ω_q:

$$p_k = \int_{\frac{2\pi}{N}(k-1)}^{\frac{2\pi}{N}k} \omega_q(\theta|r_0,\theta_0)d\theta = \frac{1}{NR} + \sum_{j=1}^{\infty} \frac{2q}{\pi j(j+qR)} \left(\frac{r_0}{R}\right)^j \sin\left(\frac{\pi j}{N}\right) \cos\left[j\left(\frac{2\pi}{N}(k-\frac{1}{2}) - \theta_0\right)\right] \tag{A4}$$

Codes

In this subsection, we list all *Fortran* codes used during the internship.

- Listing 1: "test_circle.f90". Code to validate "Reflection" method on a circle domain.
- Listing 2: "para.h". Common variables used by "PSD_rfl.f90".
- Listing 3: "PSD_rfl.f90". Code with "Reflection" method for the fractal domain. The first arrival problem could be realized by setting $q = \infty$.

Listing 1. test_circle.f90

```
!!!!! Diffuion inside circle for validation.
!!!!! Yilin YE @ 06/2023

Program main
    use MATHS
    implicit none
    integer :: i,j,k, date(8)
    integer :: num_points, num_particle, traj_max, final_max, inside, touch, n_iter
    real*8 :: angle_ini, tta, taille, radius_min, u, eps, ell, ell_hat, time_diff
    real*8 :: rho_ratio, q_rect, diff_coef, d, delta, rint, xini, yini, theta0, r0
    real*8,allocatable :: position(:,:), coord_x(:), coord_y(:), account(:), pbb_ana(:)
    real*8,external :: span,pente !! Function to calculate distance between two points.
    real*8 :: temps_debut, temps_fin, temps_provisoire
    character*128 :: rue_total

    call cpu_time(temps_debut)
    continue !! formats
        62 format(10X,'X coord',10X,'Y coord',10X,'Radius',10X,'Time',10X,'Local Time')
        63 format(12X,'X coord',12X,'Y coord',12X,'Time',12X,'Local Time',12X,'Threshold')
        64 format('#',3X,'Segment Index',3X,'Hitting Probability')
        71 format(2(3X,ES16.6))
        72 format(2(3X,f16.8),3(2X,f16.8))
        73 format(5(4X,f16.8))
        74 format(2X,I6,10X,ES16.8)
        91 format('# Polygon',I10,', Circle Radius',f8.2,', Thickness ',ES8.2)
        92 format('# No. Particle',I9,', Rho ratio',f6.2,', Reactivity ',ES8.2,', Diff coef ',ES8.2)
        94 format('Circle Test @ YYE. Version 1.1.2.. Simulation on ',I4,2('-',I2),'   ',I2,'h',I2)
        95 format('Done for particle',I9,', Time used (s)',ES11.3)
        99 format('It spent',f10.4,' seconds for the whole program.')
    continue

    num_points = 61 !! Define boundary parameters
    num_particle = 1000000; traj_max = 5; final_max = 100
    allocate(position(num_particle,2)); allocate(coord_x(num_points))
    allocate(coord_y(num_points)); allocate(account(num_points-1))
    coord_x = 0.0d0; coord_y = 0.0d0; account = 0.0d0

    rho_ratio = 1.5d0; diff_coef = 1.0d0 !! Determine diffusion/reflection parameters
    taille = 1.0d0; eps = 1.0d-3; q_rect = 1.0d0

    rue_total = "cc-" !! Generate data files.
    call date_and_time(values=date)
    open(unit=31,file=TRIM(rue_total)//"coord.txt"); write(31,94) date(1), date(2), date(3), date(5), date(6)
    open(unit=32,file=TRIM(rue_total)//"particle_traj.txt"); write(32,94) date(1), date(2), date(3), date(5), date(6)
        write(32,62)
    open(unit=33,file=TRIM(rue_total)//"particle_final.txt"); write(33,94) date(1), date(2), date(3), date(5), date(6)
        write(33,63); write(33,*)
    open(unit=34,file=TRIM(rue_total)//"pbnum.txt"); write(34,94) date(1), date(2), date(3), date(5), date(6)
        write(34,91) num_points-1, taille, eps
    write(34,92) num_particle, rho_ratio, q_rect, diff_coef; write(34,*); write(34,64); write(34,*)
    open(unit=36,file=TRIM(rue_total)//"donnees.txt")

    angle_ini = twopi/(num_points - 1) !! Generate boundary points
    do i = 1, num_points
        j = i - 1; tta = angle_ini * j
        coord_x(i) = taille * cos(tta); coord_y(i) = taille * sin(tta)
        write(31,71) coord_x(i), coord_y(i)
    end do

    xini = 0.9d0; yini = 0.0d0 !! Compute analytical probability.
    theta0 = pente(xini,yini); r0 = span(0.0d0,0.0d0,xini,yini)
    open(unit=35,file=TRIM(rue_total)//"pbana.txt")
        write(35,94) date(1), date(2), date(3), date(5), date(6); write(35,92) num_particle, rho_ratio, q_rect, diff_coef
        write(35,*) "# r0, 0 =", r0, theta0; write(35,*); write(34,*) "# r0, 0 =", r0, theta0; write(34,*)
    allocate(pbb_ana(num_points-1)); n_iter = 50
    do k = 1, num_points-1
        pbb_ana(k) = 1.0d0 / (taille * (num_points-1))
        do j = 1, n_iter
            pbb_ana(k) = pbb_ana(k) + (2.0d0 * q_rect)/(pi*j*(j+q_rect*taille)) * (r0/taille)**j * &
            & sin(pi*j/(num_points-1)) * cos(j*(twopi/(num_points-1)*(k-0.5d0)-theta0))
        end do
        write(35,74) k, pbb_ana(k)
    end do
    deallocate(pbb_ana); close(35)

    do i = 1, num_particle !! Diffusion for all particles
                !! Initialization
        call random_number(u); ell_hat = -log(u) / q_rect !! PDF needed. psi(l) = q*exp(-q*l);
        ell = 0.0d0; time_diff = 0.0d0; inside = 0; touch = 0
        position(i,1) = xini; position(i,2) = yini
```

```
        if (i <= traj_max) then
            write(32,*); write(32,*) "#    ",i
            write(32,72) position(i,1), position(i,2), radius_min, time_diff, ell
        end if

        do while (1 > 0) !! Non-stop diffusion
            radius_min = taille - span(0.0d0,0.0d0,position(i,1),position(i,2))
            if (radius_min > eps) then !! Not enter diffusion layer
                call diffusion(position(i,1), position(i,2), radius_min)
                if (radius_min > taille) then
                    write(32,*) "Leave the circle !!"
                    goto 1101
                end if
            else
                inside = 1; radius_min = eps * rho_ratio
                call diffusion(position(i,1), position(i,2), radius_min)

                d = span(0.0d0,0.0d0,position(i,1),position(i,2))
                if (d < taille - eps) then !! Out of diffusion layer
                    inside = 2
                    !touch = 0
                else if (d > taille) then !! Need reflection!
                    touch = touch + 1
                    if (position(i,1) == 0.0d0) then
                        if (position(i,2) > 0.0d0) then
                            position(i,2) = +2.0d0 * taille - position(i,2)
                        else
                            position(i,2) = -2.0d0 * taille - position(i,2)
                        end if
                    else !!                                    x
                        tta = pente(position(i,1), position(i,2))
                        rint = 2.0d0 * taille - d !span(0.0d0,0.0d0,position(i,1), position(i,2))
                        position(i,1) = rint * cos(tta); position(i,2) = rint * sin(tta)
                    end if
                end if

                if (touch > 0) then
                    delta = radius_min**2 / (4.0d0 * diff_coef); time_diff = time_diff + delta
                    ell = ell + sqrt(pi * diff_coef * delta / 2.0d0)
                    if (ell >= ell_hat) then
                        if (i <= traj_max) write(32,72) position(i,1), position(i,2), radius_min, time_diff, ell
                        if (i == 1) write(36,72) position(i,1), position(i,2), radius_min, time_diff, ell
                        goto 1101
                    end if
                end if

                if (inside == 2) then !! If leave diffusion layer, then clear "touch" and reset "inside"
                    touch = 0; inside = 0
                end if
            end if
            if (i <= traj_max) write(32,72) position(i,1), position(i,2), radius_min, time_diff, ell
            if (i == 1) write(36,72) position(i,1), position(i,2), radius_min, time_diff, ell
        end do
        1101 continue

        tta = pente(position(i,1),position(i,2)); u = (num_points-1) * tta / twopi
        k = ceiling(u) !!                    0                        ++
        account(k) = account(k) + 1

        if (i <= traj_max) write(32,*) "Local Time Threshold =", ell_hat
        if (i <= final_max) write(33,73) position(i,1), position(i,2), time_diff, ell, ell_hat
        if (MOD(i,ceiling(num_particle/5.0d0)).eq.0) then
            call cpu_time(temps_provisoire); write(*,95) i,temps_provisoire-temps_debut
        end if
    end do

    do i = 1, num_points - 1 !!! Normalize account
        account(i) = account(i) / num_particle
        write(34,74) i, account(i)
    end do
    deallocate(position); deallocate(coord_x); deallocate(coord_y); deallocate(account)
    close(31); close(32); close(33); close(34); close(36)
    call cpu_time(temps_fin)
    write(*,99) temps_fin-temps_debut !! Write the total time consumed to the screen.
end Program main

MODULE MATHS
    implicit none
    real(kind=8),parameter :: pi = 4.0d0*atan(1.0d0), twopi = 2.0d0*pi
END MODULE MATHS

real*8 function span(x1,y1,x2,y2)
    implicit none
```


```
    real*8 :: x1,y1,x2,y2
    span = sqrt((x1-x2)**2 + (y1-y2)**2)
    return
end function span

subroutine diffusion (x0,y0,rayon)
    use MATHS
    implicit none
    real*8,intent(in) :: rayon
    real*8 :: x0,y0, u, random_angle, move_x, move_y
    call random_number(u); random_angle = u * twopi
    move_x = rayon * cos(random_angle); move_y = rayon * sin(random_angle)
    x0 = x0 + move_x; y0 = y0 + move_y
end subroutine

real*8 function pente(x0,y0)
    use MATHS
    implicit none
    real*8 :: x0, y0, theta
    if (x0 == 0.0d0) then
        if (y0 > 0.0d0) then
            theta = pi / 2.0d0
        else
            theta = pi / 2.0d0 * 3.0d0
        end if
    else
        theta = atan(y0/x0)
        if (theta < 0.0d0) then
            if (x0 < 0.0d0) then
                theta = theta + pi
            else
                theta = theta + twopi
            end if
        else if (theta == 0.0d0) then
            if (x0 < 0.0d0) theta = pi
        else
            if (x0 < 0.0d0) theta = theta + pi
        end if
    end if
    pente = theta
    return
end function
```

Listing 2. para.h

```
!!!!! We define parameters used for equations

integer :: polygon_shape, Generation_max, num_particle, alpha_angle_ratio, direct, num_points
common/paraint/ polygon_shape, Generation_max, num_particle, alpha_angle_ratio, direct, num_points
real*8 :: Length, seg_l, thickness, rho_ratio, q_rect, diff_coef
common/parareal/ Length, seg_l, thickness, rho_ratio, q_rect, diff_coef
```

Listing 3. PSD_rfl.f90

```
!!!!! Diffusion near arbitrary Koch
!!!!! Reflection after first arrval
!!!!! Yilin YE @ 05/2023


Program main
    use MATHS
    implicit none
    include "para.h"
    integer :: i,j,k,l,jj ! Loop variables
    integer :: outpoint(2), intpoint(3), out_gap, int_gap, rec_old1, rec_old2
    integer :: qmax, g_act, g_test, case, infvl, supvl, search_length, k1, k2
    integer :: cross, inside, touch !! Variables for Reflection
    integer :: traj_max
    real*8 :: angle_ini, tta, new_x, new_y, radius_min, radius, u, l1, l2, ell, ell_hat, time_diff
    real*8 :: avg_seuil, avg_local, avg_temps
    real*8,allocatable :: position(:,:), coord_x(:), coord_y(:), account(:), zeta(:)
    integer,allocatable :: rec_index(:,:), search_index(:) !! Arrays to record indices
    integer,external:: noloop !! Function to return reasonnable segment index
    real*8,external :: span !! Function to calculate distance between two points.

    integer :: date(8)
    real*8 :: temps_debut, temps_fin, temps_provisoire!, time_begin, time_end !! Define variables to record time consumed
    character*128 :: rue_total !! Define path name for outputs
```

```
call cpu_time(temps_debut)
continue
    62 format(10X,'X coord',10X,'Y coord',10X,'Generation',10X,'Radius',10X,'Time',10X,'Local Time')
    63 format(12X,'X coord',12X,'Y coord',12X,'Time',12X,'Local Time',12X,'Threshold')
    64 format('#',3X,'Segment Index',3X,'Hitting Probability')
    71 format(2(3X,ES20.10))
    72 format(2(3X,f16.8),2X,I6,3(2X,f16.8))
    73 format(5(4X,f16.8))
    74 format(2X,I6,10X,ES10.4)
    !76 format(8(2X,I6),f14.5) !! format for file(36)=debug.txt
    91 format('# Polygon',I4,', Gmax',I4,', Length',f8.2,', Thickness ',ES8.2)
    92 format('# No. Particle',I9,', Rho ratio',f6.2,', Reactivity ',ES8.2,', Diff coef ',ES8.2)
    94 format('PSD @ YYE. Version 2.1.0.. Simulation on ',I4,2('-',I2),'   ',I2,'h',I2)
    95 format("Done for particle",I9,", Time used (s)",ES11.3)
    99 format('It spent',f10.4,'seconds for the whole program.')
continue



call obtaindata
call date_and_time(values=date)
rue_total = "rfl-"
open(unit=30,file=TRIM(rue_total)//"record.txt")
    write(30,94) date(1), date(2), date(3), date(5), date(6)
open(unit=31,file=TRIM(rue_total)//"coord.txt")
    write(31,94) date(1), date(2), date(3), date(5), date(6)
open(unit=32,file=TRIM(rue_total)//"particle_traj.txt")
    write(32,94) date(1), date(2), date(3), date(5), date(6)
write(32,62); write(32,*)
open(unit=33,file=TRIM(rue_total)//"particle_final.txt")
    write(33,94) date(1), date(2), date(3), date(5), date(6)
    write(33,63); write(33,*)
open(unit=34,file=TRIM(rue_total)//"pbb.txt")
    write(34,94) date(1), date(2), date(3), date(5), date(6)
    write(34,91) polygon_shape, Generation_max, Length, thickness
    write(34,92) num_particle, rho_ratio, q_rect, diff_coef; write(34,*)
    write(34,64); write(34,*)
!open(unit=35,file=TRIM(rue_total)//"zeta.txt"); write(35,94) date(1), date(2), date(3), date(5), date(6)
open(unit=36,file=TRIM(rue_total)//"debug.txt")
    write(36,94) date(1), date(2), date(3), date(5), date(6); !write(36,*)
open(unit=37,file=TRIM(rue_total)//"temps.txt")
    write(37,94) date(1), date(2), date(3), date(5), date(6);
    write(37,91) polygon_shape, Generation_max, Length, thickness
    write(37,92) num_particle, rho_ratio, q_rect, diff_coef; write(37,*)
num_points = polygon_shape * 4**Generation_max + 1 !! + 1 to form a circle
allocate(position(num_particle,2)); allocate(coord_x(num_points)); allocate(coord_y(num_points))
allocate(account(num_points-1)) !! account = count @ Python
coord_x = 0.0d0; coord_y = 0.0d0; account = 0.0d0
traj_max = 10
avg_seuil = 0.0d0; avg_local = 0.0d0; avg_temps = 0.0d0



    !!! Generate initial polygon coordinates
angle_ini = twopi / polygon_shape; tta = pi * (1.0d0 - 2.0d0/polygon_shape)
    !!!!! Note, all integers in Python should be converted as real values for +-*/, else mod taken...
new_x = - Length/2.0d0; new_y = new_x * tan(tta/2.0d0)
do i = 0, polygon_shape - 1
    j = i * 4**Generation_max + 1
    coord_x(j) = new_x; coord_y(j) = new_y !coordinate.append((new_x,new_y))
    new_x = new_x + Length * cos(i * angle_ini)
    new_y = new_y + Length * sin(i * angle_ini)
end do
coord_x(num_points) = coord_x(1); coord_y(num_points) = coord_y(1) ! To form a circle.
    !!! Insert fractal points inside
do i = 0, Generation_max - 1
    k = 4**i * polygon_shape
    out_gap = 4**(Generation_max - i); int_gap = out_gap / 4
    do j = 0, k-1
        l = out_gap * j
        outpoint(1) = l + 1
        outpoint(2) = outpoint(1) + out_gap
        intpoint(1) = outpoint(1) + int_gap
        intpoint(2) = intpoint(1) + int_gap
        intpoint(3) = intpoint(2) + int_gap
        call koch_line(coord_x(outpoint(1)), coord_y(outpoint(1)), coord_x(outpoint(2)), coord_y(outpoint(2)), &
                  &pi/alpha_angle_ratio, direct, coord_x(intpoint(1)), coord_y(intpoint(1)), &
                  &coord_x(intpoint(2)), coord_y(intpoint(2)), coord_x(intpoint(3)), coord_y(intpoint(3)))
    end do
end do
!u = sqrt( 2 * (1 - cos(pi/alpha_angle_ratio)) ) + 2
!seg_l = Length / u**Generation_max !! 22/05/23 added. Give the minimal segment length.
```

```
!goto 1102
    !!! Output fractal points
do i=1,num_points
    write(31,71) coord_x(i), coord_y(i)
end do
!1102 continue


allocate(rec_index(Generation_max + 2, 2))
out_gap = 4**Generation_max * polygon_shape !! 28/04/23 Added. (g=0) -> 3, (g=1) -> 12, (g=2) -> 48
!!! Diffusion for each particle
do i = 1, num_particle
    1100 continue

        !! Initialization
    ell = 0.0d0; call random_number(u)
    ell_hat = -log(u) / q_rect !! PDF needed. psi(l) = q*exp(-q*l);
    time_diff = 0.0d0

    position(i,:) = 0.0d0; g_act = 1
    if (i <= traj_max) then
        write(32,*) "#",i !! Furnish particle index in traj.
        write(32,72) position(i,1), position(i,2), g_act, 0.0d0, time_diff, ell
    end if
    !write(36,*) "Position Initialized!"; !write(36,*) "***** i =", i, "*****"
    !write(36,*) "Local Time Threshold", ell_hat; !write(36,*)

    rec_index(1,1) = 1; rec_index(1,2) = num_points
    do j = 2, Generation_max + 2
        rec_index(j,1) = rec_index(1,1) ! = 1
        rec_index(j,2) = rec_index(1,2) ! = num_points
    end do


        !### [1.1] **Initialize `g_act` ** (for start-up)
    g_act = 2 !! g_act = {1, 2, 3, ..., Gmax+2} ~ g = {0, 0, 1, ..., Gmax}
    radius_min = Length / 4.0d0 !! Arbitrary diffusion at first step but still inside the boundary.
    call diffusion(position(i,1), position(i,2), radius_min)
    time_diff = time_diff + radius_min**2 / (4.0d0 * diff_coef)
    if (i <= traj_max) write(32,72) position(i,1), position(i,2), g_act, radius_min, time_diff, ell


        !! Iteration for GAFRW
    do while (g_act < Generation_max + 3) !!          (1.gt.0)

        int_gap = 4**(Generation_max - g_act + 2)
        !write(36,*) "[1.2] Determine g & interval."
        !write(36,*) "g_act / int_gap =", g_act, int_gap


        continue
                        !### [2] **Find 2 nearest points** (not for start-up)
            !!
            if (g_act == 2) then
                k1 = rec_index(g_act-1,1) ! = 1
                k2 = rec_index(g_act-1,2) - int_gap ! = num_points - int_gap
                infvl = k1 + int_gap; supvl = k2 - int_gap
                search_length = polygon_shape - 2
            else
                infvl = rec_index(g_act-1,1)
                supvl = rec_index(g_act-1,2)
                call cherche(infvl, supvl, int_gap, k1, k2, search_length) !! subroutine cherche(niveau, connu1, connu2, gap, nou
            end if
            l1 = span(coord_x(k1), coord_y(k1), position(i,1), position(i,2))
            l2 = span(coord_x(k2), coord_y(k2), position(i,1), position(i,2))


            allocate(search_index(search_length))
            do j = 1, search_length
                search_index(j) = noloop(infvl + (j-1) * int_gap)
            end do

            !write(36,*) "[2] Determine search space"
            do j = 1, search_length !!              do j = infvl, supvl, int_gap?
                k = search_index(j)
                u = span(coord_x(k), coord_y(k), position(i,1), position(i,2))
                !write(36,*) "search index / distance =", k, u
                call ordre(k1, k2, l1, l2, k, u) !! subroutine ordre(k1,k2,l1,l2, k,l)
            end do
            rec_index(g_act,1)= k1; rec_index(g_act,2)= k2
            deallocate(search_index)
        continue
```

```
            !### [3] **Determine diffusion radius** (able for start-up)
search_length = noloop(rec_index(g_act,2) - rec_index(g_act,1)) / int_gap + 1
!write(36,*) "[3] Determine diffusion radius"
!write(36,*) "endpoint index", rec_index(g_act,1), rec_index(g_act,2), " search length =", search_length
allocate(search_index(search_length))
do j = 1, search_length
    search_index(j) = noloop(rec_index(g_act,1) + (j-1) * int_gap)
end do
!write(36,*) "Search index", search_index(:)
radius_min = Length!/4**(g_act-1)
k = search_index(1) !! 04/05/2023
do jj = 1, search_length - 1
    j = search_index(jj)
    l = search_index(jj+1) !! 04/05/2023                                    search_length
    call distance(coord_x(j),coord_y(j), coord_x(1),coord_y(1), position(i,1), position(i,2), case, radius)
    if (case.ne.-1) then
        if (radius_min > radius) then
            radius_min = radius
            k = j
        end if
    else
        deallocate(search_index) !! 25/05/23 added. Avoid < Attempting to allocate already allocated variable 'search_ind
        goto 1100
    end if
end do
deallocate(search_index)



            !### [4] **Diffusion** with given `radius_min`.
!write(36,*) "[4] Diffusion uniform"
!write(36,*) "R_min =", radius_min!, "(last) radius =", radius
!write(36,*) "g before", g_act
if (radius_min == Length) goto 1100
if (case.ne.+2) then
!if (radius_min > 0.0d0) then
    call diffusion(position(i,1), position(i,2), radius_min)
    time_diff = time_diff + radius_min**2 / (4.0d0 * diff_coef)
    if (i <= traj_max) write(32,72) position(i,1), position(i,2), g_act, radius_min, time_diff, ell !! Record trajs of ea
    !write(36,*) "Position afterwards.", position(i,1), position(i,2)
else !! radius_min == 0.0d0
    if (g_act == Generation_max+2) then
        cross = 0; inside = 1; touch = 0

        do while (inside > 0) !! Repete the process once the particle enter the layer.
            if (inside == 2) exit
            if (inside == 3) then
                k = noloop(k+1)
                inside = 1
            end if
            call reflection(coord_x(k),coord_y(k), coord_x(noloop(k+1)),coord_y(noloop(k+1)), &
            &position(i,1),position(i,2), coord_x(noloop(k-1)),coord_y(noloop(k-1)), &
            &coord_x(noloop(k+2)),coord_y(noloop(k+2)), cross, inside, touch, radius_min, time_diff, ell, ell_hat)
            if (abs(cross) == 1) then !! Perhaps hitting segments change.
                k = noloop(k + cross)
                cross = 0
            end if
            !write(36,*) "Call Reflection + 1", i
            !write(36,*) "Position afterwards.", position(i,1), position(i,2)
            !write(36,*) "cross, inside, touch = ", cross, inside, touch
            !write(36,*) "Time =", time_diff, "Local Time =", ell
            if (i <= traj_max) write(32,72) position(i,1), position(i,2), g_act, radius_min, time_diff, ell
        end do

        if ((ell >= ell_hat).or.(inside == 2)) then
            if (i <= traj_max) then
                write(32,*) "Final Time Used =", time_diff
                write(32,*) "Final Local Time =", ell
                write(32,*) "Local Time Threshold =", ell_hat
                write(32,*)
            end if
            if (i <= 20) write(33,73) position(i,1), position(i,2), time_diff, ell, ell_hat !! Print final positions, dif
            account(k) = account(k) + 1
            !write(36,*) "***** Particle Attached! *****"; !write(36,*) "i = ",i; !write(36,*); !write(36,*); !write(36,*
            !write(37,'(ES14.6)') time_diff
            avg_temps = avg_temps + time_diff
            avg_local = avg_local + ell
            avg_seuil = avg_seuil + ell_hat
            goto 1101 !! Jump the out while loop.
        end if
```

```
            else
                g_act = g_act + 1
            end if
        end if
        !write(36,*) "g after ", g_act



        !### [5] **Determine if go out** (not for start up)
        !write(36,*) "[5] Verify correct region."
        !do g_test = 2, g_act !! Search all passed level g. Make sure the particle not out of region at all level.
        g_test = 2
        do while (g_test <= g_act)
            int_gap = 4**(Generation_max - g_test + 2) !

            if (g_test.eq.2) then
                k1 = rec_index(g_test-1,1) ! = 1
                k2 = rec_index(g_test-1,2) - int_gap ! = num_points - int_gap
                infv1 = k1 + int_gap; supv1 = k2 - int_gap
                search_length = polygon_shape - 2
            else
                infv1 = rec_index(g_test-1,1) !! Define search inf
                supv1 = rec_index(g_test-1,2) !! Define search sup
                call cherche(infv1, supv1, int_gap, k1, k2, search_length) !! subroutine cherche(niveau, connu1, connu2, gap, nou
            end if
            l1 = span(coord_x(k1), coord_y(k1), position(i,1), position(i,2))
            l2 = span(coord_x(k2), coord_y(k2), position(i,1), position(i,2))



            allocate(search_index(search_length))
            do j = 1, search_length
                search_index(j) = noloop(infv1 + (j-1) * int_gap)
            end do
            rec_old1 = rec_index(g_test,1); rec_old2 = rec_index(g_test,2) !

            !write(36,*) "g_test / int_gap =", g_test, int_gap
            !write(36,*) "Endpoint index", rec_index(g_test-1,:)
            !write(36,*) "k1/k2 & l1/l2", k1, k2, l1, l2
            !write(36,*) "search length = ", search_length
            do j = 1, search_length
                k = search_index(j)
                u = span(coord_x(k), coord_y(k), position(i,1), position(i,2))
                !write(36,*) "search_index / distance", k, u
                call ordre(k1,k2, l1,l2, k,u) !! subroutine ordre(k1,k2,l1,l2, k,l)
            end do
            rec_index(g_test,1) = k1; rec_index(g_test,2) = k2
            deallocate(search_index)
            !write(36,*) "k1/k2 & l1/l2", k1, k2, l1, l2



            if ((noloop(rec_old1).ne.noloop(rec_index(g_test,1))).or.(noloop(rec_old2).ne.noloop(rec_index(g_test,2)))) then
                !write(36,*) "Break @ g_test =", g_test
                g_act = g_test !!                                   gg_test
                !if (g_act > Generation_max+2) g_act = Generation_max
            end if

            g_test = g_test + 1
        end do


        if (noloop(rec_index(g_act,2) - rec_index(g_act,1)) == int_gap) then
            if (g_act < Generation_max + 2) g_act = g_act + 1
        end if


        !write(36,*) "Done once for while loop."
        !write(36,*) "i =",i,"g_act =",g_act
        !write(36,*) "#rec(1)", rec_index(:,1)
        !write(36,*) "#rec(2)", rec_index(:,2)
        !write(36,*); !write(36,*)


    end do

    1101 continue
    if ((MOD(i,ceiling(num_particle/5.0d0))==0).or.(i==num_particle)) then
        call cpu_time(temps_provisoire)
        write(*,95) i,temps_provisoire-temps_debut
        write(30,95) i,temps_provisoire-temps_debut
    end if
end do
```

```
        !!! Normalize account
    do i = 1, num_points - 1
        account(i) = account(i) / num_particle
        write(34,74) i, account(i)
    end do

    write(30,*); write(30,*)
    avg_seuil = avg_seuil / num_particle; write(30,*) "Average Threshold  ", avg_seuil
    avg_local = avg_local / num_particle; write(30,*) "Average Local Time ", avg_local
    avg_temps = avg_temps / num_particle; write(30,*) "Average Diff Time  ", avg_temps
    write(30,*); write(30,*); write(30,*)



    goto 1104
        qmax = 10
        allocate(zeta(qmax))
        do i = 1, qmax
            u = 0.0d0
            do k = 1, num_points - 1
                u = u + account(k)**i
            end do
            zeta(i) = u
            write(35,*) i,zeta(i)
        end do
        deallocate(zeta)
    1104 continue



    deallocate(position); deallocate(coord_x); deallocate(coord_y); deallocate(account)
    deallocate(rec_index)
    close(31); close(32); close(33); close(34); !close(35);
    close(36); close(37)


    call cpu_time(temps_fin)
    write(*,99) temps_fin-temps_debut !! Write the total time consumed to the screen.
    write(30,99) temps_fin-temps_debut
    close(30)
        ! close files?
end Program main




MODULE MATHS
    implicit none
    real(kind=8),parameter :: pi = 4.0d0*atan(1.0d0), twopi = 2.0d0*pi
    real(kind=8),parameter :: sqrt2 = sqrt(2.0d0), sqrt3 = sqrt(3.0d0), septds3 = 7.0d0/sqrt3, cinqds3 = 5.0d0/sqrt3
    CONTAINS
    function normaldist(mean,std,n) result(r)
        implicit none
        real(kind=8),intent(in) :: mean,std
        integer,intent(in) :: n
        real(kind=8) :: r(n,2)
        real(kind=8),dimension(n,2) :: zeta
        !call random_seed()
        call random_number(zeta)
        r(:,1) = dsqrt(-2.0d0*log(zeta(:,1)))*cos(twopi*zeta(:,2))
        r(:,2) = dsqrt(-2.0d0*log(zeta(:,1)))*sin(twopi*zeta(:,2))
        r = mean + std * r
    end function normaldist
END MODULE MATHS




subroutine obtaindata
    implicit none
        include "para.h"
    integer :: status1
    character*128 :: msg,ruein

    ruein="input_rf1.txt"
    open(unit=201,file=TRIM(ruein),form='formatted',status='old',action='read',iostat=status1,iomsg=msg)

    read(201,*) polygon_shape !! 3 = triangular; 4 = square; 5 = star; etc.
```


```
    read(201,*) Generation_max !! The maximum value of generation / recursion
    read(201,*) num_particle !! Number of particles
    read(201,*) Length !! Edge length for initial polygon
    read(201,*) alpha_angle_ratio !! Fractal angle
    read(201,*) direct
    read(201,*) thickness
    read(201,*) rho_ratio
    read(201,*) q_rect
    read(201,*) diff_coef

    close(201)
end subroutine
subroutine koch_line(x1,y1,x2,y2,alpha,direction, b1,b2,c1,c2,d1,d2)
    use MATHS
    implicit none
    real*8, intent(in) :: x1,y1,x2,y2,alpha ! coordinates for start/end points & fractal angle
    integer, intent(in) :: direction !     +1 = concave,     -1 = convex
    real*8, intent(out) :: b1,b2,c1,c2,d1,d2 ! coordinates for inserted points
    real*8 :: deltax,deltay,l,coef,segm,beta,theta,changex,changey,degree

    deltax = x2 - x1; deltay = y2 - y1
    l = sqrt((deltax)**2 + (deltay)**2) ! the length of the line
    coef = sqrt( 2 * (1 - cos(alpha)) ) + 2; segm = l / (coef); beta = (pi - alpha)/2
    if (x1.eq.x2) then
        if (y1.lt.y2) theta = + pi / 2
        if (y1.gt.y2) theta = - pi / 2
    else
        theta = atan(deltay/deltax)
    end if
    if (x1.gt.x2) theta = theta + pi

    changex = deltax / coef; changey = deltay / coef
    b1 = x1 + changex; b2 = y1 + changey ! second point: one third in each direction from the first point
    degree = theta + beta * direction
    c1 = b1 + segm * cos(degree); c2 = b2 + segm * sin(degree) ! third point: rotation for multiple of 60 degrees
    d1 = x2 - changex; d2 = y2 - changey ! fourth point: two thirds in each direction from the first point
end subroutine
subroutine distance(x1, y1, x2, y2, x0, y0, dehors, dmin)
    use MATHS
    implicit none
        include "para.h"
    real*8, intent(in) :: x1, y1, x2, y2, x0, y0 !! Given 2 points: (x1,y1) & (x2,y2); Particle: (x0,y0)
    integer,intent(out):: dehors
    real*8, intent(out):: dmin
    real*8 :: deltax, deltay, dist_xy, eps, theta, st, ct, xdist, ydist, dxnew, dynew, d1, d2
    real*8,external :: span

    deltax = x2 - x1; deltay = y2 - y1
    !dist_xy = sqrt(deltax**2 + deltay**2)
    dist_xy = span(x1,y1, x2,y2)
    !eps = dist_xy * thickness !eps = dist_xy / 10**3, 19/05/23 modified
    eps = thickness !! 20/05/23 modified. Independent variable, not a ratio.

    if (x1.eq.x2) then
        if (y1.le.y2) then
            theta = + pi / 2.0d0
        else
            theta = - pi / 2.0d0
        end if
    else
        theta = atan(deltay/deltax)
    end if
    if (x1.gt.x2) theta = theta + pi
    st = sin(theta); ct = cos(theta)

    xdist = x0 - x1; ydist = y0 - y1
    dxnew = + ct * xdist + st * ydist; dynew = - st * xdist + ct * ydist

    if (dynew.lt.0) then !# To avoid errors while g>2.
        dmin = Length; dehors = -1
    else
        if ((dxnew.ge.0).and.(dxnew.le.dist_xy)) then
            dmin = dynew; dehors = +1
            if (dmin.lt.eps) then
                !dmin = 0.0d0
                dehors = +2
            end if
        else
            dehors = 0
            d1 = sqrt(xdist**2 + ydist**2)
            d2 = sqrt((x0-x2)**2 + (y0-y2)**2)
            if (d1.le.d2) then
                dmin = d1
```

```
            else
                dmin = d2
            end if
        end if
    end if
end subroutine
real*8 function span(x1, y1, x2, y2)
    implicit none
    real*8 :: x1, y1, x2, y2
    span = sqrt((x1-x2)**2 + (y1-y2)**2)
    return
end function span



subroutine cherche(connu1, connu2, gap, nouveau1, nouveau2, combien)
    !! Here is a procedure to extend the search space
    implicit none !! "niveau" = g_act/g_test; "gap" = distance of two point indices
    integer,intent(in) :: connu1, connu2, gap !! "connu1/2" refer to obvious search range, namely #rec(g_act-1,:)
    integer,intent(out):: nouveau1, nouveau2, combien !! "nouveau1/2" = two new point indices after extension.
    integer,external :: noloop

    combien = noloop(connu2 - connu1) / gap + 1
    nouveau1 = noloop(connu1 - gap)
    nouveau2 = noloop(connu2 + gap)
end subroutine
integer function noloop(index)
    implicit none
    include "para.h"
    integer :: index

    if (index >= num_points) then
        noloop = mod(index, num_points-1)
    else
        if (index <= 0) index = index + num_points - 1
        noloop = index
    end if
    return
end function noloop



subroutine ordre(k1,k2,l1,l2, k,l)
    !! Here is a procedure to find 2 nearest points. Added 03/05/2023
    implicit none
    include "para.h"
    integer,intent(in) :: k
    integer :: k1, k2, kint
    real*8,intent(in) :: l
    real*8 :: l1, l2, u, lint

    !!
    if (l1 == l2) then
        if (l < l1) then
            call random_number(u)
            if (u < 0.5d0) then
                k1 = k; l1 = l
            else
                k2 = k; l2 = l
            end if
        end if
    else
        if (l1 > l2) then !!
            kint = k1; k1 = k2; k2 = kint
            lint = l1; l1 = l2; l2 = lint
        end if

        if (l < l1) then
            k2 = k1; k1 = k
            l2 = l1; l1 = l
        else
            if (l < l2) then
                k2 = k; l2 = l
                goto 3101
            end if

            if (l == l2) then !!
                call random_number(u)
                if (u < 0.5d0) then
                    k2 = k; l2 = l
                end if
                goto 3101
```

```
            end if

            3101 continue
        end if
    end if

    !!
    if (((k1 > k2).and.((k1 - k2) < num_points/2)).or.&
    & ((k1 < k2).and.((k2 - k1) > num_points/2))) then
        lint = l1; l1 = l2; l2 = lint
        kint = k1; k1 = k2; k2 = kint
    end if
end subroutine



subroutine diffusion(x0, y0, rayon)
    !! Here is a procedure to realize uniform diffusion on the circle of radius = "rayon". Added 03/05/23.
    use MATHS
    implicit none
    real*8,intent(in) :: rayon
    real*8 :: x0, y0
    real*8 :: u, random_angle, move_x, move_y

    call random_number(u); random_angle = u * twopi
    move_x = rayon * cos(random_angle); move_y = rayon * sin(random_angle)
    x0 = x0 + move_x; y0 = y0 + move_y
end subroutine



subroutine reflection(x1,y1, x2,y2, x0,y0, x3,y3, x4,y4, croise, interieure, touche, rayon, time, local_time, threshold)
    !! Here is a procedure to realize reflection when particle inside boundary layer. Added 19/05/23.
    use MATHS
    implicit none
    include "para.h"
    real*8, intent(in) :: x1,y1, x2,y2, x3,y3, x4,y4 !! Boundary points with order 3-1-2-4
    real*8,intent(out) :: rayon
    real*8 :: x0, y0, time, local_time !! Particle coordinates, inside 1-2. time t; local_time ell_t
    real*8, intent(in) :: threshold
    integer :: croise, interieure, touche !! encounter: how many times reflection on the boundary; interieure: inside (=1) layer or n
    integer :: case1, case2, case_new, angle1, angle2 !! Left/Right boundary point environments. Condition after diffusion.
    real*8 :: d1, d2, radius_min, d14, d23, limite !! Left/Right boundary point distance.
    real*8 :: deltax, deltay, dist_xy, theta, st, ct, xdist, ydist, dxnew, dynew, u !! Variables for Rotations.
    real*8 :: eps, rho, delta !! epaisseur, rayon, temps.
    real*8,external :: span
    !!! 1. Rotation; 2. Determine case & radius; 3. Diffusion; 4. Reflection; 5. Determine inside or not; 6. Anti-rotation


    !!! [01] Pre-Rotation
    deltax = x2 - x1; deltay = y2 - y1
    dist_xy = sqrt(deltax**2 + deltay**2) !! Do not understand. But this expression run faster.
    !dist_xy = span(x1,y1,x2,y2) !dist_xy = seg_l


    !!! [0] Determine abuse or obtuse angle
    d23 = span(x2,y2,x3,y3)
    if (d23 > sqrt2 * dist_xy) then
        angle1 = 2 !! 4  /3 = 240
    else
        angle1 = 1 !!   /3 = 60
    end if
    d14 = span(x1,y1,x4,y4)
    if (d14 > sqrt2 * dist_xy) then
        angle2 = 2 !! 4  /3 = 240
    else
        angle2 = 1 !!   /3 = 60
    end if


    !!! [1] Rotation
    if (x1 == x2) then
        if (y1 <= y2) then
            theta = + pi / 2.0d0
        else
            theta = - pi / 2.0d0
        end if
    else
        theta = atan(deltay/deltax)
    end if
    if (x1 > x2) theta = theta + pi
    st = sin(theta); ct = cos(theta); xdist = x0 - x1; ydist = y0 - y1
    dxnew = + ct * xdist + st * ydist; dynew = - st * xdist + ct * ydist
```

```
!!! [2] Determine case & radius
eps = thickness; rho = rho_ratio * eps; limite = 1.0d-6
!case1 = 6; case2 = 6 !! Help to determine whether we call "distance", and then which point is closer.
if (abs(croise) > 1) then !! Inside crossing region
    call random_number(u)
    if (croise > 0) then !! Close to xy2
        interieure = 3
        croise = -croise
        goto 3201
        if (croise == 2) then !! 4  /3 close to xy2
            d2 = span(x2,y2, x0,y0)
            if (d2 < eps * limite) then
                radius_min = eps * 0.2d0
                u = pi / 3.0d0 * (4.0d0 * u - 1.0d0)
                if (u < pi / 6.0d0) then
                    croise = +1
                else if (u > pi / 2.0d0) then
                    croise = 0
                end if
                dxnew = radius_min * cos(u) + dist_xy
                dynew = radius_min * sin(u) !!            xy2
                goto 3203 !! Complete Diffusion, Then only consider the local time increasement.
            else
                radius_min = d2 !! Can Apply Normal Diffusion
                !u = u * twopi
            end if
        else !! (croise == 3),   /3 close to xy2
            call distance(x1,y1, x2,y2, x0,y0, case1, d1)
            call distance(x2,y2, x4,y4, x0,y0, case2, d2)
            if ((d1 < eps * limite).and.(d2 < eps * limite)) then
                radius_min = eps * sqrt3
                if (u < 0.5d0) then
                    croise = +1
                else
                    croise = 0
                end if
                u = pi / 3.0d0 * (u + 2.0d0)
                dxnew = radius_min * cos(u) + dist_xy
                dynew = radius_min * sin(u) !!            xy2
                goto 3203 !! Complete Diffusion, Then only consider the local time increasement.
            else !! Reflection allowed, 05/06/23 added
                radius_min = max(d1,d2) !! Can Apply Normal Diffusion
                !u = u * twopi
            end if
        end if
    else !! Close to xy1
        if (croise == -2) then !! 4  /3 close to xy1
            d1 = span(x1,y1, x0,y0)
            if (d1 < eps * limite) then !! Almost converge to point xy1
                radius_min = eps * 0.2d0
                u = pi / 3.0d0 * u * 4.0d0
                if (u < pi / 2.0d0) then
                    croise = 0
                else if (u > pi / 6.0d0 * 5.0d0) then
                    croise = -1
                end if
                dxnew = radius_min * cos(u)
                dynew = radius_min * sin(u)
                goto 3203 !! Complete Diffusion, Then only consider the local time increasement.
            else
                radius_min = d1  !! Can Apply Normal Diffusion
                !u = u * twopi
            end if
        else !! (croise == -3),   /3 close to xy1
            !call distance(x3,y3, x1,y1, x0,y0, case1, d1)
            d1 = abs(sqrt3 * dxnew - dynew) / 2.0d0
            !call distance(x1,y1, x2,y2, x0,y0, case2, d2)
            d2 = dynew
            if ((d1 < eps * limite).and.(d2 < eps * limite)) then
                radius_min = eps * sqrt3
                if (u < 0.5d0) then
                    croise = 0
                else
                    croise = -1
                end if
                u = pi / 3.0d0 * u
                dxnew = radius_min * cos(u)
                dynew = radius_min * sin(u)
                goto 3203 !! Complete Diffusion, Then only consider the local time increasement.
            else !! Reflection allowed, 05/06/23 added
                radius_min = max(d1,d2) !! Can Apply Normal Diffusion
```

```
                !u = u * twopi
            end if
        end if
    end if
    goto 3202 !! Once inside region "croise = 2 / 3", determine radius directly and so jump normal case.
end if


!! Normal case inside 1/2 segment / interval.
if (dxnew < septds3 * eps) then !! Close 31 segment.
    if (angle1 == 2) then !! 4 /3 = 240
        d1 = span(x1,y1,x0,y0)
    else !! /3 = 60
        !call distance(x3,y3, x1,y1, x0,y0, case1, d1)
        d1 = abs(sqrt3 * dxnew - dynew) / 2.0d0
        !case1 = 0
    end if
    radius_min = min(rho, d1); croise = -1
else if (dxnew > dist_xy - septds3 * eps) then !! Close to 24 segment
    if (angle2 == 2) then !! 4 /3 = 240
        d2 = span(x2,y2,x0,y0)
    else !! /3 = 60
        !call distance(x2,y2, x4,y4, x0,y0, case2, d2)
        d2 = abs(sqrt3 * dxnew + dynew - dist_xy) / 2.0d0
        !case2 = 0
    end if
    radius_min = min(rho, d2); croise = +1
else
    radius_min = rho
end if
3202 continue
rayon = radius_min


!!! [3] Diffusion
call diffusion(dxnew, dynew, radius_min)
!!! [4] Reflection ?
if (dynew < 0.0d0) then
    touche = touche + 1
    dynew = - dynew
end if


3203 continue !!
!!! [6] Anti-Rotation
xdist = ct * dxnew - st * dynew; ydist = st * dxnew + ct * dynew
x0 = x1 + xdist; y0 = y1 + ydist
!!! [7] Add Local Time
if (touche > 0) then !! Only the reflection case can increase local time.
    delta = radius_min**2 / (4.0d0 * diff_coef)
    time = time + delta
    local_time = local_time + sqrt(pi * diff_coef * delta / 2.0d0)
    if (local_time >= threshold) then
        interieure = 2 !! 25/05/23 added. Stop "Reflection" even inside the diffusion layer.
        goto 3201
    end if
end if


!!! [5] Inside ?
if (croise == 0) then !! Assure that even after the diffusion, the particle doesn't enter other boundary. septds3!
    if (dynew > eps) interieure = 0
else
    !if (case1 > case2) then !! Particle closer to right boundary point
    if (croise > 0) then !! Particle closer to right boundary point
        call distance(x2,y2, x4,y4, x0,y0, case_new, d2)
        select case (case_new)
            case(0) !! angle = 120
                if (dynew > eps) then !!
                    interieure = 0
                    goto 3201
                end if
                if (dxnew > dist_xy) then
                    if (d2 > eps) then !!
                        interieure = 0
                    else !!
                        croise = +2
                    end if
                else !!       12
                    croise = 0 !! 31/05/23 added.
                end if
            case(1) !! angle = 60, but not inside layer
                if (dynew > eps) then !!
```

```
                    interieure = 0
                else !!              12
                    croise = 0 !! 31/05/23 added.
                end if
            case(2) !! angle = 60, but inside layer
                !if (d2 < dynew) then
                if (dynew > eps) then !!                          1224
                    croise = +1 !! Attach to right segment
                else !!                  124                /3
                    !croise = 0
                    croise = +3
                end if
        end select
    else !! Particle closer to left boundary point
        call distance(x3,y3, x1,y1, x0,y0, case_new, d1)
        select case (case_new)
            case(0) !! angle = 120
                if (dynew > eps) then !!
                    interieure = 0
                    goto 3201
                end if
                if (dxnew < 0) then
                    if (d1 > eps) then !!
                        interieure = 0
                    else !!
                        croise = -2
                    end if
                else !!       12
                    croise = 0 !! 31/05/23 added.
                end if
            case(1) !! angle = 60, but not inside layer
                if (dynew > eps) then  !!
                    interieure = 0
                else !!              12
                    croise = 0 !! 31/05/23 added.
                end if
            case(2) !! angle = 60, but inside layer
                !if (d1 < dynew) then
                if (dynew > eps) then !!                          1231
                    croise = -1 !! Attach to left segment
                else !!                  312                /3
                    !croise = 0
                    croise = -3 !! 31/05/23 added.
                end if
        end select
    end if
end if
3201 continue

end subroutine
```