# Efficient algorithm for the periodic Lorentz gas in 2 and 3 dimensions

Atahualpa S. Kraemer*
*Institut für Theoretische Physik II - Soft Matter Heinrich-Heine-Universität Düsseldorf Building 25.32 Room O2.56 Universitätsstrasse 1 D-40225 Düsseldorf, Germany*

Nikolay Kryukov† and David P. Sanders‡
*Departamento de Física, Facultad de Ciencias, Universidad Nacional Autónoma de México, Ciudad Universitaria, México D.F. 04510, Mexico*

(Dated: June 23, 2015)

We present an efficient algorithm to calculate collisions for a periodic Lorentz gas in 2 and 3 dimensions, using continued fractions. The method works very well in the 2-dimensional case, obtaining the exact disc with which a particle will collide at each step, instead of using periodic boundary conditions. However, the 3D case obtains these coordinates only with a probability $p_{al}$ which depends on the size of the obstacles as $p_{al} = 1-(16-\sqrt{128}-\frac{4}{3}\pi)r^3 \sim 1-0.5r^3$. This 3D algorithm is generalizable to higher dimensions; however, the efficiency of the method becomes worse.

## INTRODUCTION

Lorentz gases are probably the simplest systems that present deterministic chaos [**?** ], and probably one of the most popular models in statistical mechanics and nonlinear dynamics which is known as Sinai billiard when the scatterers are periodic [**?** ]. This model consists of an array of scatterers (usually spheres), placed in the vertices of a grid, and particles which move freely until they encounter obstacles where they experience inelastic collisions.

This kind of systems can have different configuration of the scatterers, as for example, random lattices [**? ? ?** ] or quasiperiodic systems [**? ?** ]. However, because of the simplicity, the periodic case has been widely studied (see e.g. [**? ? ? ?** ]), but there are still many open questions [**? ? ? ?** ]. Many of the results obtained theoretically for these gases are in the limit where obstacles are very small, i.e. the Boltzmann–Grads limit [**? ? ? ? ? ? ? ? ?** ].

Usually, in this case, simulations are made in a single cell, with periodic boundary conditions (or reflecting in the borders) with an obstacle in the middle of the cell[**? ?** ]. However, this implies that the program must check each time when the particle crosses a border, whether it will collide with the obstacle in the cell, or it will move to the next cell. That implies solving at least two linear equations and one quadratic, then checking if the solution of the quadratic equation is real and finally taking the maximum value among the three (or two in case of the complex solution to the quadratic equation) results. If the obstacle is big enough, it is quite probable that the particle will collide each time it crosses the boundary. However, in the Boltzmann-Grads limit, this situation does not happen very often, and then the program loses much time checking when particles cross the boundary and if there is a collision with the obstacle even if there is not.

A suitable situation would be a formula to find the coordinates of the next obstacle giving the position and velocity of the particle. As we will see in section  this is indeed the first solution of the Diophantine inequality:

$$|\alpha q + b - p| \leq \frac{v^2}{v_x} r \tag{1}$$

where $r$ is the radius of the obstacles, $q$ and $p$ are integer variables, and $\alpha$ and $b$ are two real numbers related to the position $\vec{x}$ and velocity $\vec{v}$ of the particle. But this inequality is closely related to the best rational approximant to an irrational number for the case $b = 0$ and the solutions can be obtained by the continued fraction algorithm.

Apparently the algorithm for this, using continued fractions, was developed before [**?** ]. However, it was never published by their authors. Nevertheless, the continued fractions were used on several occasions to develop the theory about the free path distribution of the periodic Lorentz gas in the Boltzmann-Grads limit [**? ? ? ? ? ? ? ? ?** ], and relatively recently, Golse has developed an algorithm to coding particles trajectories by using the continued fraction algorithm, and the 3-length theorem [**?** ]. However, this algorithm a priory works only if the particle leaves the surface of a disk. This restriction makes the algorithm not useful in other geometries as quasiperiodic arrays or two incommensurate overlapping arrays of square lattices and different shapes of obstacles, but those system may produce a number of surprising effects []. On the other hand, because the way the algorithm is made (using the 3-length theorem and continued fraction algorithm), it is not possible to use it in higher dimensions, the fact that was noted by Golse, arguing that the one involving higher dimensions "is a notoriously more difficult problem".

## 2D ALGORITHM

The goal of the 2D algorithm is to calculate efficiently where the first collision of a particle will occur inside a square lattice. Without loss of generality, we will use the lattice formed by the integer coordinates in the 2D plane. Then, we want to calculate the minimal time $t > 0$, such that $|\vec{c} - (\vec{x}_0 + \vec{v}_0 t)| = r$, where $r$ is the radius of the obstacle, $\vec{x}_0$

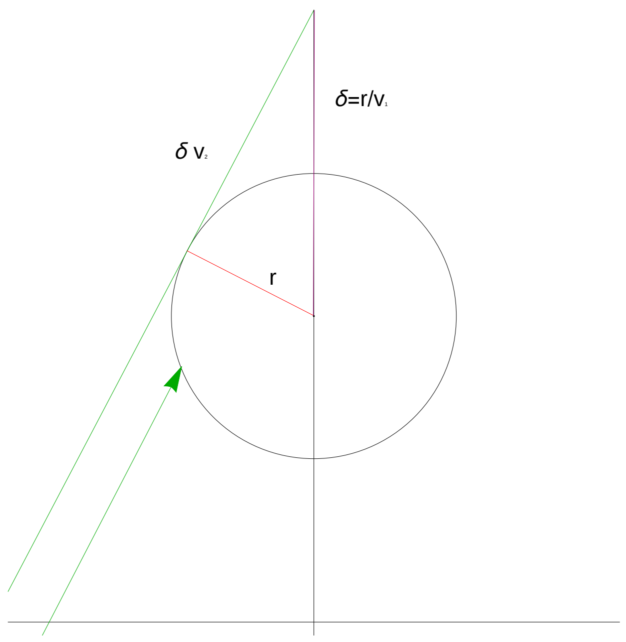

FIG. 1. Relation between the intersection of a line and a circle with integer coordinates and the intersection of the line with the x-axis

and $\vec{v}_0$ are the initial position and velocity of the particles, and $\vec{c} = (q, p)$ are the integer coordinates. In this case, the collision takes place at the point $\vec{x}_0 + \vec{v}_0 t$, with the center of the obstacle at $\vec{c}$.

As the first improvement over the classical algorithm for the 2D case note that instead of finding the intersection between a line (corresponding to the trajectory of the particle, $y = \alpha x + b$) and a circle (corresponding to the obstacle $(x-q)^2 + (y-p)^2 = r^2$), we can solve the intersection between two lines: $y = \alpha x + b$ and $x = q$. As it is shown in figure 1, if $|\alpha q + b - p| = |b| < \delta = r/v_1$, then a collision will take place. Of course, this is true only if $v_1 > 0$ and $v_2 > 0$, but after each collision, we can always rotate or reflect the system (because of the symmetry) such that these conditions are satisfied. Even better, because of the periodic boundary conditions, we can always put $1 > b > 0$, then we only need to solve $b < \delta$, this already gives us a very powerful tool to calculate with which obstacle the particle will collide.

Indeed, we do not need apply at every step periodic boundary conditions, we only need to check

$$|\{\alpha k'_n\} + b - 1| < \delta, \tag{2}$$

where $\{\cdot\}$ denote the fractional part, and $k_n = k_{n-1} + 1$, where $k_1$ is the $x$-coordinate of the closest obstacle to the particle at $t = 0$. Then, the first $k_n$ that satisfied this inequality, will be $q$. To calculate $p$, we can note that always $p = \lfloor \alpha q + b \rfloor$ or $p = \lfloor \alpha q + b \rfloor + 1$

Now, to simplify even more the algorithm, consider the integer coordinates $(k'_n, h'_n)$ such that

$$|\alpha k'_n - h'_n + b| < \delta, \tag{3}$$

and for any pair of numbers $(i, j)$ such that $i < k'_n$, then $|\alpha i - j + b| > \delta$, $q = k'_n$, and $p = h'_n$.

But $|\alpha k'_i - h'_i + b|$ are the distances between the integer coordinates $(k'_i, h'_i)$ and the point $(k'_i, \alpha k'_i + b)$, then, we would like a succession such that

$$|\alpha k'_i - h'_i + b| < |\alpha k'_{i-1} - h'_{i-1} + b| \tag{4}$$

for every $i > 1 \in \mathbb{N}$. Also, the first pair of integer coordinates $k'_0$ and $h'_0$ should be $(0,0)$ or $(0,1)$, minimizing $|\alpha k'_0 - h'_0 + b|$, that is:

$$|\alpha k'_1 - h'_1 + b| < f(b) = \begin{cases} b & \text{if } b < 1/2 \\ 1-b & \text{if } b > 1/2 \end{cases} \tag{5}$$

Note that $h'_n = \lfloor \alpha k'_n + b \rfloor = \begin{cases} \lfloor \alpha k'_n \rfloor & \text{if } b + \alpha k'_n - \lfloor \alpha k'_n \rfloor < 1 \\ \lfloor \alpha k'_n \rfloor + 1 & \text{if } b + \alpha k'_n - \lfloor \alpha k'_n \rfloor > 1 \end{cases}$ if $b < 1/2$ or $h'_n = \lfloor \alpha k'_n + b \rfloor + 1 = \begin{cases} \lfloor \alpha k'_n \rfloor + 1 & \text{if } b + \alpha k'_n - \lfloor \alpha k'_n \rfloor < 1 \\ \lfloor \alpha k'_n \rfloor + 2 & \text{if } b + \alpha k'_n - \lfloor \alpha k'_n \rfloor > 1 \end{cases}$ if $b > 1/2$, then, substituting the four cases form here in the two cases of equation 5, we obtain that indeed $h'_1 = \lfloor \alpha k'_1 \rfloor + 1$, and then, iterating equation 4 we obtain

$$h'_n = \lfloor \alpha k'_n \rfloor + 1. \tag{6}$$

Mixing the inequality 3 and equation 6, we obtain again equation 2.

Thus, we have reduced the solution from two linear equations and one quadratic to one linear equation. Even more, now we do not check in every periodic cell, because if $\alpha > 1$, for every $k_n$ we advance $\alpha$ cells. And we don't need to apply periodic boundary conditions until we reach the obstacle.

## The Diophantine inequality: $|\alpha p - q| \leq \varepsilon$

Now, a better algorithm should find a way to find the set of $k'_i$, such that inequality 4 keeps and there is not any integer $q$ such that there exists an $i$ such that $k'_i < q < k'_{i-1}$ and $|\{\alpha k'_i\} + b - 1| < |\{\alpha q\} + b - 1| < |\{\alpha k'_{i-1}\} + b - 1|$.

In order to do this, we can use the continued fraction algorithm to obtain solutions to the inequality $|\alpha q - p| \leq \varepsilon$. This algorithm already gives the succession of $(k_n, h_n)$ such that $|\alpha k_i - h_i| < |\alpha k_{i-1} - h_{i-1}|$ if $k_{i-1} < k_i$. So, if we turn our inequality 5 into this other inequality, we will find our algorithm just by using the continued fraction algorithm. Indeed, using equation 6 and the inequality 5, we obtain $|\{\alpha k'_1\} - 1| < \begin{cases} 2b & \text{if } b < 1/2 \\ 2(1-b) & \text{if } b > 1/2 \end{cases}$, which is almost the continued fraction inequality, except that $p$ is always equal to $\lfloor \alpha q \rfloor + 1$.

$$|\alpha q - p| < \begin{cases} 2b & \text{if } b < 1/2 \\ 2(1-b) & \text{if } b > 1/2 \end{cases} \tag{7}$$

Then, we can apply the continued fraction algorithm to obtain $p$ and $q$ of inequality 7. If $\lfloor \alpha q \rfloor + 1$, then, we have found $(k'_1, h'_1)$ otherwise not, but we know that $h'_1 \geq p$, and $k'_1 \geq q$ then we can just use $(q,p)$ even if they do not satisfy inequality 4, then calculate the succession $b_i$ as $b_0 = b$, $b_i = \{\alpha k'_i + b\}$. If $b_n < \delta$ the algorithm stops, and the collision will take place with the obstacle centered at the coordinates $(k'_n, h'_n)$.

## Explicit algorithm

Now we have all the necessary tools to implement the complete algorithm step by step. The Julia language is used here.

Call `frac(`$\alpha, \varepsilon$`)` the function that obtains the first integers $(q,p)$, such that $|\alpha q - p| < \varepsilon$ by using the continued fraction algorithm (see appendix).

```
function frac(x, epsilon)
   h1, h2 = 1, 0
    k1, k2 = 0, 1
    b = x
   while abs(k1*x - h1) > epsilon
       a = ifloor(b)
       h1, h2 = a*h1 + h2, h1
        k1, k2 = a*k1 + k2, k1
       b = 1/(b - a)
    end
   return k1, h1
end
```

Now, define the function `efficient_algorithm(m, b,` $\varepsilon$`)` as follows:

```
function efficient_algorithm(m, b, epsilon)
    kn = 0
    while b > epsilon && 1 - b > epsilon
        if b < 0.5
            (q, p) = frac(m, 2b)
        else
            (q, p) = frac(m, 2*(1 - b))
        end
        b = mod(m*q + b, 1)
        kn += q
    end
    q = kn
    p = ifloor(m*q) + 1
    return (q, p)
end
```

This function applies the continued fraction algorithm along with the inequality 7 until the distance in the y-direction between the line and the integer coordinates $(q,p)$ is less than $\varepsilon$.

Then, the algorithm to localize the first obstacle, with which a particle with velocity $\vec{v}$ and position $\vec{x}$ will collide, is as follows (in pseudocode):

```
if (v is in quadrant I) then
   (q,p)=efficient_algorithm(m, b, delta)
   p=int(m*q)+1
else if (v is in quadrant II) then
   m=-m
   % reflect the system with
   %respect to y-axis
   (q,p)=efficient_algorithm(m, b, delta)
   p=int(m*q)+1
   m=-m
   % reflect the system with
   %respect to y-axis
   q=-q
   % reflect the system with
   %respect to y-axis
else if (v is in quadrant III) then
   b=1-b
   % reflect the system with
   %respect to the origin
   (q,p)=efficient_algorithm(m, b, delta)
   b=1-b
   % reflect the system with
   %respect to the origin
   p=-(int(m*q)+1)
   % reflect the system with
   %respect to the origin
   q=-q
   % reflect the system with
   %respect to the origin
else if (v is in quadrant IV) then
   b=1-b
   % reflect the system with
   %respect to the x-axis
   m=-m
   % reflect the system with
   %respect to the x-axis
   (q,p)=efficient_algorithm(m, b, delta)
   b=1-b
   % reflect the system with
   %respect to the x-axis
   m=-m
   % reflect the system with
   %respect to the x-axis
   p=-(int(m*q)+1)
   % reflect the system with
   %respect to the x-axis
endif
```

Using the technique mentioned above, which, for the initial position $(0,b), 0 < b < 1$ and for any initial velocity returns the integer coordinates of the obstacle of the first collision, we make the function `collisions` that calculates the trajectory of the particle in the 2D plane. In order to calculate the exact collision point and the velocity after the collision, we need to use the classical collision function `collide`.

```
function collide(q, p, x, y, vx, vy, r)
```

```
    r0 = [x, y]
    v0 = [vx, vy]
    v0 /= norm(v0)
    R = [q, p]
    crossz(x, y) = x[1]*y[2] - x[2]*y[1]
    discr = norm(v0)^2*r^2 - (crossz(v0, r0-R))^2
    t1 = (-dot(v0, r0-R) - sqrt(discr))/norm(v0)^2
    N0 = r0 + v0*t1 - R
    N = N0/norm(N0)
    v1 = v0 - 2*dot(v0, N)*N
    r1 = r0 + v0*t1
    return r1[1], r1[2], v1[1], v1[2]
end
```

At every step of the trajectory, we create an array of the integer corners of the square where the particle is, and then, given the line of motion of the particle, we determine whether it will leave the square or experience a collision in the same square; also, we make sure that the corner which is behind the particle (i.e., is at the direction opposite to the motion) does not get counted as a collision. There is a small problem at the first step: in order for the function to work, we put a dummy point into the array of the coordinates of the collisions (usually a point very far from the origin), and at the first step it may be counted as a collision, so we delete it. If the particle exits the square, we determine the side through which it exits: if it is one of the two vertical sides, we simply apply the efficient algorithm and obtain the place of the next collision; if it is one of the two horizontal sides, we rotate the coordinates, apply the efficient algorithm, and rotate the coordinates back, thus obtaining the place of the next collision. If the particle does not exit the square, we use the classical collision function to determine the new point of collision and the new velocity.

The functions `frac`, `efficient_algorithm` and `first_collision`, which comprise the implementation of the efficient algorithm, and the function `collisions`, which calculates the trajectory, can be found in the `EfficientLorentz` module (LINK TO MODULE).

## 3D CASE

In the previous section we developed an algorithm to efficiently calculate the motion of particles in a 2-dimensional periodic Lorentz gas. However, the same ideas cannot be applied directly to a 3-dimensional case. In this section we will show how to use this method in higher dimensions, losing some efficiency.

*3D algorithm* If we project a 3D cubic lattice onto the $xy$, $xz$ or $yz$ plane, we will obtain a periodic square lattice. Also, if we project a 3D line onto those planes, we will obtain a 2D line parametrized by a linear function of the time. Then we can apply the 2D algorithm onto the 3 projections, and take the maximum time $t$ among the 3 obtained times. We will call $s$ the distance covered by the particle in the maximum time $t$. If the obstacle coordinates are the same in the 3 planes, we

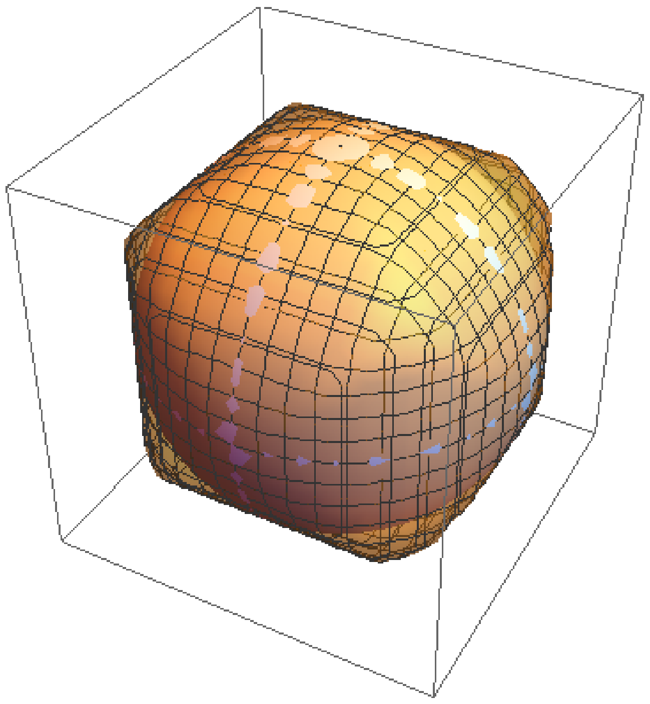

FIG. 2. A sphere of radius $r$ embedded into the intersection of 3 orthogonal cylinders of the same radius. The volume where there is no sphere, but there is the intersection of the 3 cylinders, is the place where trajectories of particles can be considered as having collisions, when really they do not use the proposed algorithm in the 3D case.

will check if the particle collides with this obstacle; if not, we move the particle by the distance $s$, then we advance to the next cell, we apply periodic boundary conditions, and we use again the 2D continued fraction algorithm in the 3 projected planes.

Using the 2D continued fraction algorithm in each of the 3 planes $xy$, $xz$ and $yz$ is equivalent to calculating a collision with a cylinder orthogonal to those planes. Then, taking only the solution where the particle collides in the 3 planes is equivalent to calculating the coordinates of the center of the intersection of the three orthogonal cylinders with the same radius. As Figure 2 shows, even in the case when the 3 planes obtain the same coordinate of the obstacle, with this distance ($s$), the 3D algorithm will not always find the next collision because some trajectories will collide with the intersection of the three orthogonal cylinders but will not have a collision with the sphere. However, we can assure that the first collision will take place in a time longer or equal than the time $t$. Then, we can check if the collision takes place. If it happens, we calculate the collision by calculating the intersection between the sphere with radius $\delta$ and coordinates $\vec{h}_n$ where $\vec{h}_n$ are the 3 integers obtained with the 3D algorithm, and the line that describes the trajectory of the particle. If there is no collision, we move up to the next cell after the calculated distance, apply periodic boundary conditions, and apply again the 3D algorithm.

Also, as Figure 2 suggests, the difference $p_{al}$ between the

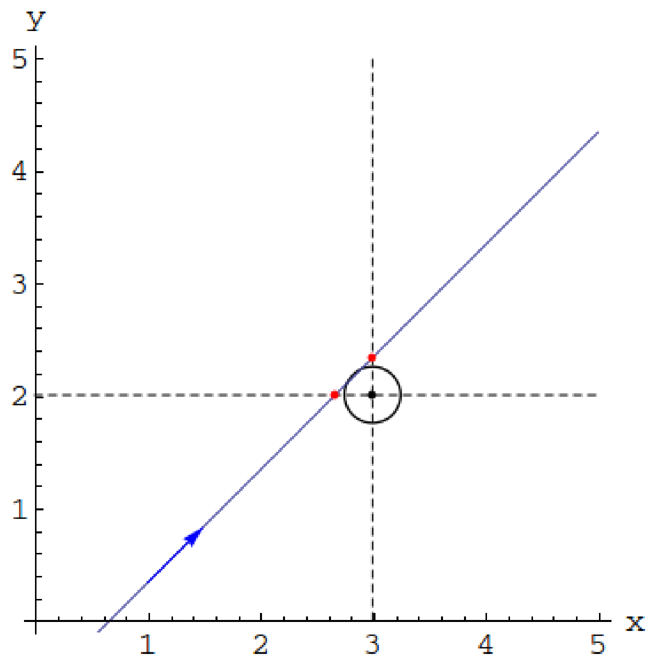

FIG. 3. Times to the integer coordinates of the circle in the case of the velocity angle close to $\pi/4$.

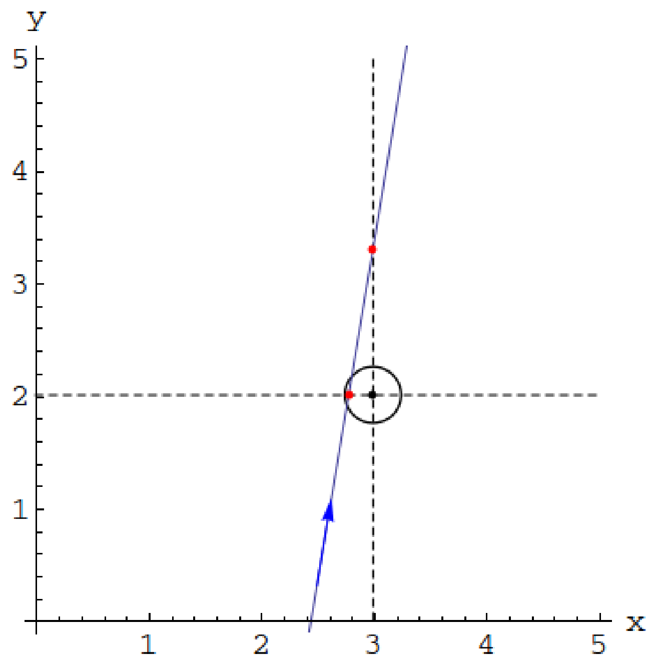

FIG. 4. Times to the integer coordinates of the circle in the case of the velocity angle close to $\pi/2$.

real obstacle and the intersection of the cylinders is not large. This difference measures directly the probability that the 3D algorithm will find the next collision. In other words, the probability $p_{al}$ that the 3D algorithm will find the next collision is:

$$p_{al} = 1 - (16 - \sqrt{128} - \frac{4}{3}\pi)r^3 \sim 1 - 0.5r^3, \qquad (8)$$

which is very small for small obstacles (for example, obstacles of radius $r = 0.01$ will produce a probability $p_{al} \sim 0.999995$ that the algorithm will find the next collision).

It turns out that out of the three planes to calculate the collision by projections, only two are enough. We estimate the time to the first collision in the xy-plane and the yz-plane, using the corresponding projections of the coordinates and velocities on these planes. If the times are not approximately equal (we take the maximum absolute difference to be 0.4, given the normalized velocity), it means that there is no real collision in 3D space. We select the lesser of the two times and advance to the next 2D obstacle determined by the efficient algorithm, and so on until the times are approximately equal. At that moment we obtain the 3D "candidate" for the valid collision. Then, using the classical 3D function `collide3d`, which calculates the coordinates of the collision, given the initial conditions of the particle, the coordinates of the center of the obstacle and its radius (and which outputs `false` when there is no collision) we determine if the collision is real. If it is, we record the obstacle, the coordinates of the collision and the new velocity and go to the next step. If it is not, we keep going with the same velocity and repeat this step.

The function measuring the approximate time to the circle (2D) may be expressed in a straightforward formula dividing the distance between the two points and the speed, which involves square roots. To make it slightly more efficient, we developed another method which avoids square roots and employs only the four arithmetic operations and elementary functions such as sign, absolute value and maximum/minimum. The idea is as follows. We measure the times of the displacement to the integer x- and y-coordinates (or the corresponding two coordinates on the plane; here we denote them $x$ and $y$ without the loss of generality) of the center of the circle. From Figure 3 we can see that if the angle is about $\pi/4$ (or a multiple of it, not including 0), then these times are not very different from each other.

However, when the angle is close to $\pi/2$ (or a multiple of it, including 0), then the two times can be significantly different (see Figure 4). Moreover, whether the greater or the lesser time of the two is closer to the approximate time to the circle depends on whether the velocity vector has a greater angle $\varphi = \text{arctg}(v_y/v_x)$ than the vector from the initial point to the center of the circle. If it is greater (as in Figure 3 or 4), then, in the case presented in Figure 4 the minimum of the two times corresponds to the approximate value. If the velocity angle is less than that of the vector to the center (for example, if the trajectory in Figure 4 had the same slope but lesser y-intercept, i.e., would go "under" the circle), then the maximum of the two times would occur near the circle and thus correspond to the approximate time. The condition on the two angles can be found measuring the sign of the z-component of the cross product of the velocity and the vector to the center.

The first condition (whether the angle is close to a multiple of $\pi/2$ and which one – basically, in which of the 8 octants of the plane separated by the multiples of $\pi/4$ the angle is) can be found simply by evaluating the sign of the product $v_x v_y$ and whose absolute value (of $v_x$ and $v_y$) is greater. By employing the two conditions, the approximate time is found as the maximum or the minimum of the two times, depending on the conditions.

The function `time_to_circle` is given in the function `collisions3d_time`, which implements the technique described here and is presented in the `EfficientLorentz` module (LINK TO MODULE). We also use the classical functions `collide3d`, to calculate the place where the particle will collide if the obstacle has center $x_2$ and radius $r$, and the particle has velocity $v$ and initial position $x_1$, and `v_new`, to calculate the velocity after the collision at the point $x_1$, where $x_2$ is the center of the sphere.

```
function collide3d(x1, x2, v, r)
    b = dot(x1 - x2, v)/norm(v)^2
    c = norm(x1 - x2)^2 - r^2
    if b^2 - c < 0
        return false
    end
    t = -b - sqrt(b^2 - c)
    x = v*t + x1
    return x
end

function v_new(x1, x2, v)
    n = x1 - x2
    n /= norm(n)
    vn = dot(n, v)*n
    v -= 2vn
    v /= norm(v)
    return v
end
```

## NUMERICAL MEASUREMENTS

We measured the average time of the execution of the function finding the first collision starting from the initial point around the origin, depending on the radius of the obstacle, for both the classical and efficient algorithms.

Figures 5 and 6 show the results of this simulation for 2D and 3D versions. As we can see, the new algorithm is significantly more efficient for $r < 0.01$.

## CONCLUSIONS

We have presented an efficient algorithm to study periodic Lorentz gases in 2 and 3 dimensions when the obstacles are in the Boltzmann-Grad limit, i.e. when the obstacles are very small.

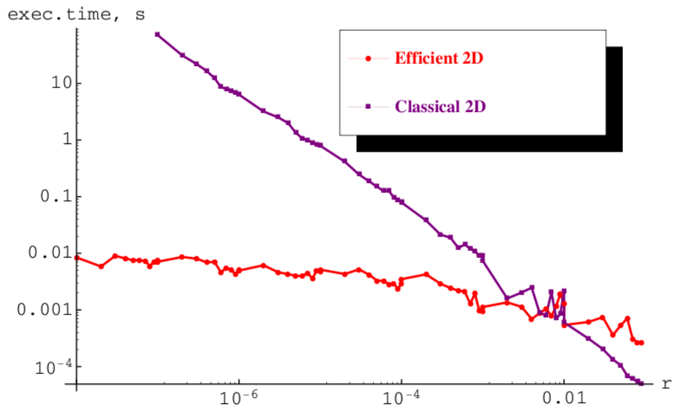

FIG. 5. Average execution time of finding the first collision in 2D Lorentz gas, for the classical and efficient allgorithms.

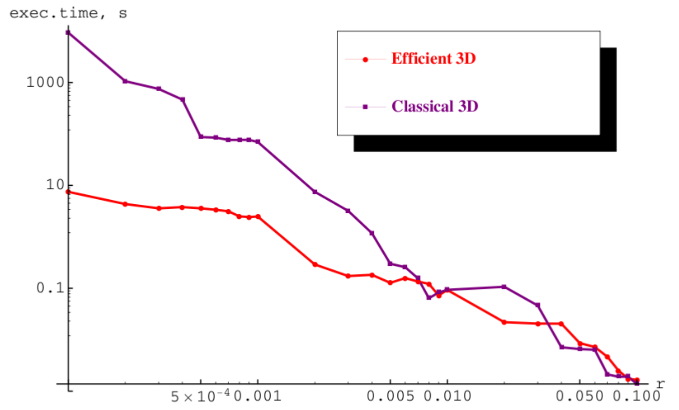

FIG. 6. Average execution time of finding the first collision in 2D Lorentz gas, for the classical and efficient allgorithms.

## ACKNOWLEDGMENTS

We thank ... ASK received support from the ERC Advanced Grant INTERCOCOS (Grant No. 267499). and DPS received financial support from CONACYT Grant CB-101246 and DGAPA-UNAM PAPIIT Grants IN116212 and IN117214.

## APPENDIX: APPROXIMATION OF IRRATIONAL NUMBERS BY RATIONAL

In this section we will summarize some of the principal results about continued fractions that we will use in the algorithm. The demonstrations can be found in many books on number theory (see for example [**?** ]). The geometrical interpretation was also suggested before by many other authors (see, for example, [**?** ]).

Let us define a continued fraction as follows: A continued fraction is an expression obtained through an iterative process of representing a number $\alpha$ as the sum of its integer part $a_0$ and the reciprocal of another number $\alpha_1 = \alpha - a_0$, then writing $\alpha_1$ as the sum of its integer part $a_1$ and the reciprocal of $\alpha_2 = \alpha_1 - a_1$, and so on.

$$\alpha = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \ldots}}}$$

This expression produces a succession of integer numbers $\lfloor \alpha \rfloor = a_0, \lfloor \alpha_1 \rfloor = a_1, \lfloor \alpha_2 \rfloor = a_2, \ldots$. Then, we define inductively two successions of integers $\{h_n\}$ and $\{k_n\}$ in the following way:

$$h_{-2} = 0, \quad h_{-1} = 1, \quad h_i = a_i h_{i-1} + h_{i-2} \tag{9}$$
$$k_{-2} = 1, \quad k_{-1} = 0, \quad k_i = a_i k_{i-1} + k_{i-2} \tag{10}$$

With this succession we can approximate any irrational number $\alpha$ using the Hurwitz theorem: *For every irrational number $\alpha$ all the relative prime integers $h_n$, $k_n$ of the successions defined in equations 9 and 10 keep the formula:*

$$|\alpha - \frac{h_n}{k_n}| \leq \frac{1}{{k_n}^2} \tag{11}$$