Fractional dimension in Rubinian gravitation from scratch, for C++ programmers

S. Halayka*

Wednesday 21st May, 2025 14:35

Abstract

In this paper, we investigate the fractional dimension of the gravitational field in the simulation of the non-relativistic gravitation of disk-like zero-pressure density systems. Solutions for Newtonian (e.g. Keplerian) and Galactic flat rotation curve (e.g. Rubinian) gravitation are given in terms of the gravitational field lines from the holographic principle. Full C++ code is provided.

1 Introduction

Up to here, we have posited that there is dark matter, in order to explain the Galactic flat rotation curve. In this paper we show that the gravitational field can be made to be anisotropic, which strengthens the effect of gravitation.

At first, we define entropy in terms of the n-bit integers, and then in terms of the black holes. In the end, they're pretty much identical – a black hole is mathematics manifested in its purest form.

Then we discuss the brute force numerical solution of the equations for the Newtonian orbit and the Galactic flat rotation curve orbit.

Next, we use a heuristic, analytical approach to the solve the equations for the Galactic flat rotation curve orbit.

Finally, we visualize the difference between the Newtonian orbit and the flat rotation curve orbit, in terms of the relativistic Doppler effect.

In this paper, we consider that the pressure density is near zero at the Galactic scale (e.g. at interstellar distances, there is no pressure acting between the stars), which simplifies the equations, and allows for anistropic gravitation in the first place.

2 Entropy

Entropy is a measure of how much information is required to encode a set of microscopic states (e.g. microstates). Where there are W distinct microstates, the Shannon binary entropy n is:

$$n = \frac{-\sum_{i=1}^{W} p_i \log p_i}{\log 2}, \tag{1}$$

*sjhalayka@gmail.com

where p_i is the probability of the ith microstate, and all probabilities add up to 1:

$$\sum_{i=1}^{W} p_i = 1. \tag{2}$$

For equiprobable microstates, like in the case of the 8-bit integers for instance, where $p_i = 1/W$, the equation for n greatly simplifies:

$$n = \frac{\log W}{\log 2}. \tag{3}$$

The number of equiprobable, distinct 8-bit integers is $2^8 = 256$, and so $n = 8$ and $W = 256$. This is to say that if you understand the concept of the 8-bit integers, then you also understand binary entropy in the case of equiprobable microstates. For the 16-bit integers, $n = 16$ and $W = 65536$, and so on and so forth.

At low density (e.g. everyday densities on the Earth), this binary entropy is proportional to mass. The Boltzmann binary entropy is:

$$n = \frac{k \log W}{\log 2}. \tag{4}$$

At the highest density (e.g. black hole densities), this binary entropy is proportional to mass squared. The Bekenstein-Hawking binary entropy is related to the black hole event horizon area A:

$$n = \frac{Akc^3}{4G\hbar \log 2}. \tag{5}$$

The following is a code for calculating the entropy of a string of characters. Note that the calculation allows for non-equiprobabilities, and so it's a bit more general.

```
#include <iostream>
#include <map>
#include <string>
#include <sstream>
#include <cmath>
using namespace std;

int main(void)
{
    // Non-equiprobable
    string input_string = "Hello world";

    // Equiprobable
    //string input_string = "abcdefgh";

    size_t length = input_string.length();

    ostringstream output_stream;

    double entropy = 0.0;

    if(length > 0)
    {
        map<char, size_t> input_map;
```

```
        for(size_t i = 0; i < length; i++)
            input_map[input_string[i]]++;

        for(map<char, size_t>::const_iterator ci = input_map.begin();
                        ci != input_map.end();
                        ci++)
        {
            double probability = ci->second / static_cast<double>(length);

            output_stream <<
                        "Character: \'" << ci->first <<
                        "\', Count: " << ci->second <<
                        ", Probability: " << probability <<
                        endl;

            entropy += probability * log(probability);
        }

        entropy = -entropy / log(2.0);
    }

    cout << output_stream.str() << endl;
    cout << "Entropy: " << entropy << endl;

    return 0;
}
```

3 Brute force: field line intersection density gradient

It was Gerard 't Hooft's idea to quantize the black hole's event horizon area into gravitational field line emitters (e.g. dividing the entropy by $\log 2$) in his pioneering work on the holographic principle. From here on we take n to be the gravitational field line count.

In this paper we numerically solve for the Newtonian and flat (e.g. constant velocity) gradients (e.g. derivatives) using an axis-aligned bounding box by obtaining the lengths of the intersecting field line segments emitted by the gravitating body. See Fig.1.

Here,

$$D = (2, 3] \tag{6}$$

stands for dimension, and

$$d = 3 - D = [0, 1) \tag{7}$$

stands for disk-like.

Regarding the holographic principle, the Schwarzschild black hole event horizon radius is:

$$r_s = \sqrt{\frac{A}{4\pi}} = \sqrt{\frac{nG\hbar \log 2}{kc^3\pi}}, \tag{8}$$

and the mass is:

$$M = \frac{c^2 r_s}{2G} = \sqrt{\frac{nc\hbar \log 2}{4Gk\pi}}. \tag{9}$$

Where R is some far distance from the centre of the gravitating body (e.g, $R \gg r_s$), β is the get intersecting line length density function, and ϵ is some small value (e.g 10^{-5}), the gradient is:

$$\gamma = \frac{\beta(R+\epsilon) - \beta(R)}{\epsilon}. \tag{10}$$

The following equation for the gradient strength was found to be true through trial and error:

$$g = -\gamma\pi \approx \frac{n}{2R^D}. \tag{11}$$

Thus, the Newtonian acceleration a_{Newton}, where $D = 3$, is:

$$a_{Newton} = \frac{v_{Newton}^2}{R} = \sqrt{\frac{nGc\hbar \log 2}{4k\pi R^4}} = \frac{GM}{R^2}, \tag{12}$$

and the acceleration a_{flat} for a flat rotation curve, where $D < 3$, and v_{flat} is known via observation of the relativistic Doppler effect, is:

$$a_{flat} = \frac{v_{flat}^2}{R} = \frac{nc\hbar \log 2}{4k\pi R^{(D-1)} M}. \tag{13}$$

The ratio of the two accelerations is:

$$\frac{a_{flat}}{a_{Newton}} = \frac{v_{flat}^2}{v_{Newton}^2} = R^d, \tag{14}$$

and the fractional dimension of the gravitational field at R is:

$$D = 3 - \frac{\log R^d}{\log R}. \tag{15}$$

It should be noted that the inherent strength of the gravitational interaction does not increase with the reduction of dimension in Rubinian gravity. This is to be explored in a future tutorial on relativistic gravitation, where the gravitational field is non-linear, and so the gravitational field itself also counts as mass-energy (e.g. a source of gravitational lensing in Bullet Cluster (1E 0657-56), for example). For instance, one candidate equation pair is simply:

$$\frac{a_{flat}}{a_{Newton}} = R^d c^d, \tag{16}$$

$$D = \frac{3\log R - \log \frac{R^d c^d}{c^3}}{\log R + \log c}. \tag{17}$$

The whole code for this section can be obtained at:
`https://github.com/sjhalayka/ellipsoid_emitter`

The following code snippets show how the whole code calculates the gravitational field lines based on a disk-like emitter.

```
bool intersect_AABB(
        const vector_3 min_location, const vector_3 max_location,
        const vector_3 ray_origin, const vector_3 ray_dir,
        double &tmin, double &tmax)
{
```

Figure 1: This figure shows an axis-aligned bounding box and an isotropic emitter, looking from slightly above. An example field line (red) and intersecting line segment (green) are given. The bounding box is filled with these green intersecting line segments. It is the gradient of the density of these line segments that forms the gravitational acceleration.

```
        tmin = (min_location.x - ray_origin.x) / ray_dir.x;
        tmax = (max_location.x - ray_origin.x) / ray_dir.x;

        if (tmin > tmax) swap(tmin, tmax);

        double tymin = (min_location.y - ray_origin.y) / ray_dir.y;
        double tymax = (max_location.y - ray_origin.y) / ray_dir.y;

        if (tymin > tymax) swap(tymin, tymax);
        if ((tmin > tymax) || (tymin > tmax)) return false;
        if (tymin > tmin) tmin = tymin;
        if (tymax < tmax) tmax = tymax;

        double tzmin = (min_location.z - ray_origin.z) / ray_dir.z;
        double tzmax = (max_location.z - ray_origin.z) / ray_dir.z;

        if (tzmin > tzmax) swap(tzmin, tzmax);
        if ((tmin > tzmax) || (tzmin > tmax)) return false;
        if (tzmin > tmin) tmin = tzmin;
        if (tzmax < tmax) tmax = tzmax;

        return true;
}


vector_4 RayEllipsoid(vector_3 ro, vector_3 rd, vector_3 r)
{
        vector_3 r2 = r * r;
        double a = rd.dot(rd / r2);
        double b = ro.dot(rd / r2);
        double c = ro.dot(ro / r2);
        double h = b * b - a * (c - 1.0);

        if (h < 0.0)
                return vector_4(-1, 0, 0, 0);

        double t = (-b - sqrt(h)) / a;

        vector_3 pos = ro + rd * t;

        return vector_4(t, pos.x, pos.y, pos.z);
}

vector_3 EllipsoidNormal(vector_3 pos, vector_3 ra)
{
        vector_3 normal = (pos / (ra * ra));
        normal.normalize();

        return -normal;
}

...

for (size_t i = 0; i < n; i++)
{
        // Something much smaller than unit vectors
```

```
        vector_3 oscillator =
                RandomUnitVector() * 0.01;

        const vector_4 rv =
                RayEllipsoid(
                        vector_3(0, 0, 0),
                        oscillator,
                        vector_3(1.0 - disk_like, 1.0, 1.0 - disk_like));

        normals[i] =
                EllipsoidNormal(
                        vector_3(rv.y, rv.z, rv.w),
                        vector_3(1.0 - disk_like, 1.0, 1.0 - disk_like));
}

...

for (size_t i = 0; i < n; i++)
{
        vector_3 ray_origin = vector_3(0, 0, 0);
        vector_3 ray_dir = normals[i];

        double tmin = 0, tmax = 0;

        if (intersect_AABB(
                min_location,
                max_location,
                ray_origin,
                ray_dir,
                tmin, tmax))
        {
                // If pointing in the wrong direction
                if (tmin < 0 || tmax < 0)
                        continue;

                vector_3 ray_hit_start = ray_origin + ray_dir * tmin;
                vector_3 ray_hit_end = ray_origin + ray_dir * tmax;

                double intersecting_line_segment_length =
                        (ray_hit_end - ray_hit_start).length();

                density0 += intersecting_line_segment_length;
        }
}

density0 /=
        (max_location.x - min_location.x) *
        (max_location.y - min_location.y) *
        (max_location.z - min_location.z);

...
```

See Figs 2, 3, and 4 for visualization of the gravitational field where $D = 3$, $D = 2.1$, and $D = 2.001$.

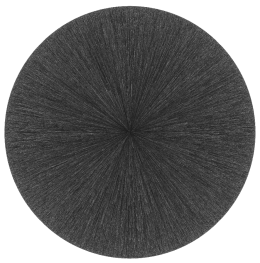

Figure 2: Where $D = 3$, as viewed from the side. The field lines are isotropic, spherical.

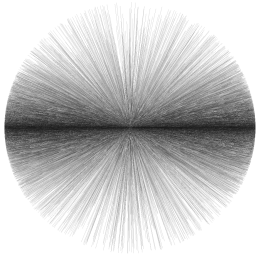

Figure 3: Where $D = 2.1$, as viewed from the side. The field lines are increasingly anisotropic.

Figure 4: Where $D = 2.001$, as viewed from the side. The field lines are anisotropic, disk-like.

4 Heuristic: field line intersection density gradient

Below is an analytical code that calculates the dimension D for the Galactic orbit of the Solar System at a speed of $v = 220000$ metres per second. The output dimension is $D = 2.98352$. That is, the Galactic gravitational field shape at R is slightly flattened, as observed.

```
#include <cmath>
#include <iostream>
using namespace std;

const double G = 6.67430e-11;
const double c = 299792458;
const double c2 = c * c;
const double c3 = c * c * c;
const double pi = 4.0 * atan(1.0);
const double h = 6.62607015e-34;
const double hbar = h / (2.0 * pi);
const double k = 1.380649e-23;

int main(void)
{
        double M = 1e41;

        double r_s = 2 * G * M / c2;
        double A_s = 4 * pi * r_s * r_s;
        double n = A_s * k * c3 / (4 * G * hbar * log(2.0));

        double R = 3e20;

        double a_Newton =
                sqrt(
                (n * G * c * hbar * log(2.0)) /
                (4 * k * pi * R * R * R * R));

        double v_Newton = sqrt(a_Newton * R);
        double v_flat = 220000;

        double D = 3.0 -
                log(v_flat*v_flat / (v_Newton*v_Newton)) /
                log(R);

        double a_flat_ =
                n * c * hbar * log(2.0) /
                (4 * k * pi * pow(R, D - 1.0) * M);

        double a_flat = pow(220000, 2.0) / R;

        cout << a_flat_ << " " << a_flat << endl;

        return 0;
}
```

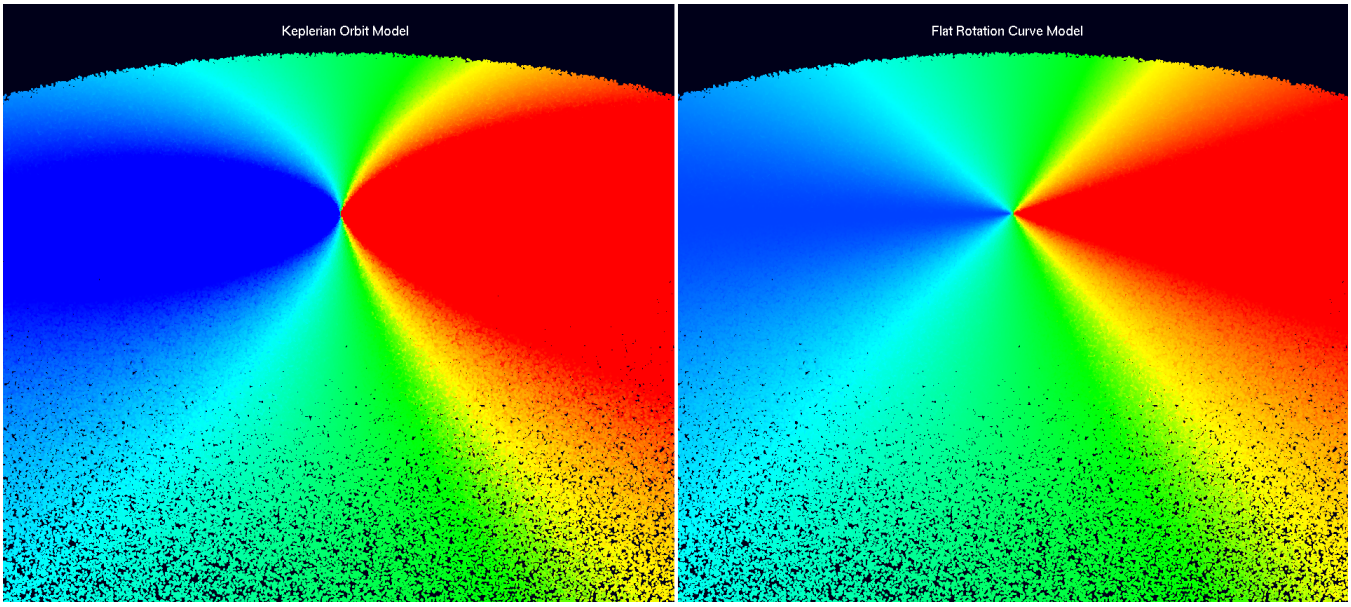

Figure 5: Visualization of the relativistic Doppler effect for 10^6 stars. The Newtonian orbit is on the left, and the flat rotation curve orbit is on the right. The redshift of the wavelength indicates stars moving away from the camera, and blueshift of the wavelength indicates stars moving toward the camera. Thus, the stars in the galaxy are orbiting counterclockwise. On the left, the wavelength is dependent on angle and distance from the galactic centre. On the right, the wavelength is dependent only on angle, which means that there is a constant orbit speed that is independent of the distance from the galactic centre. This is exactly what Vera Rubin discovered.

5 Application: visualizing the relativistic Doppler effect

For the sake of reference, attached here is a visualization that highlights the difference between Newtonian orbit and flat rotation curve orbit, in terms of the relativistic Doppler effect.

The whole code for this section can be found at:
`https://github.com/sjhalayka/relativistic_doppler_effect`

6 Review

How many bits n does it take to encode W equiprobable microstates? The answer is: $n = \frac{\log W}{\log 2}$. For instance, consider a fair $W = 2^n$-sided die, where all sides are equiprobable, and so $p_i = 1/W$.

Both the brute force and heuristic methods for solving the equations for the Newtonian orbit and the Galactic flat rotation curve orbit have been given.

The dimension of the gravitational field has been shown to be fractional.

The relativistic Doppler effect is used to calculate the orbit speed of stars.

References

[1] Einstein. On the electrodynamics of moving bodies. (1905)

[2] Misner et al. Gravitation. (1970)

[3] 't Hooft. Dimensional reduction in quantum gravity. (1993)

[4] Susskind. The World as a Hologram. (1994)

[5] Mandelbrot. How Long Is the Coast of Britain? Statistical Self-Similarity and Fractional Dimension. (1967)

[6] Rubin. A century of galaxy spectroscopy. (1995)