compass_numbers

May 31, 2023

1 A friendly introduction to complex numbers at work : Time atlas and compass-numbers

The idea is to make a friendly re-introduction (meaning I assume you already know the core basics) to the idea of complex numbers with a playfull example and the analogy of compass is used instead of using the word complex every time.

The usual explanation starts with:

$$i^2 = -1$$

- Rectangular representation (horizontal,vertical)

$$z = x + iy$$

- Polar representation using Euler

$$z = re^{i\theta}$$

$$r = \sqrt{x^2 + y^2}$$

$$\theta = \arctan(\frac{y}{x})$$

- So unpacking

$$z = re^{i\theta} = \sqrt{x^2 + y^2}e^{i \arctan(\frac{y}{x})}$$

- Or introduced as a 2D number and how it can be used to operate with them as if they where vectors

$$z = (x, y)$$

- For this text I'll asume you already know the basics.
- Explaining this and doing the usual introduction on how they are 2D numbers and can be added as vectors, and can also be scaled and shifted.

2 Imagine a large beam compass we use to place symbols in a 2D plane where we define r and θ

Just for notation simplicity, let's define a 'phase-generating-number' as the simple "compass" number e^i, and let's call it Θ.

The compass can be set to an angle taken between $-\pi$ and π we'll call $\$\ \$$ can be used as the argument in a (eigen)function noted as Θ^θ.

This sets the angle for our beam compass destination (imagine that at rest the pencil is aligned with the horizontal axis, and from there counter-clock-wise we make a displacement of θ radians)

$$\Theta = e^i = \cos(1) + i\sin(1)$$

$$\Theta^\theta = e^{i\theta} = \cos(\theta) + i\sin(\theta)$$

2.0.1 The radius

We will introduce also the idea of length from our center of coordinates (where our beam compass center is placed)

We will call this r for the radius of the circle we can plot to set the compass to this length:

$$r \in \mathbb{R}$$

or more generally r is also in $\mathbb{C}$, meaning I can use it to shift the phase of the Θ^θ

For historic reasons, related to how where this type of number was discovered/invented, this is called formally as a Complex Number.

$$r\Theta^\theta \in \mathbb{C}$$

The i used as an exponent here for notation simplicity is indicating that the argument of the $exp()$ function has as an argument $i\theta = \sqrt{-1}\theta$.

$$\theta_k = \omega_k k$$

In Python, i is indicated using $j = 1j$.

Now, bear with me…

- Let's use $\omega_k = \frac{2\pi}{T_k}$ as a measure of how many symbols T_k we want to place distributed in an arc of 2π radians to represent as a set of compass numbers the integer sequence denoted as k. (mention introductory relationship roots of unity,modular arithmetic and periodicity)
- The use of ω here might be strange, because we are not precisely referencing angular velocity, but keep in mind this is just a notation trick that will make more sense later on.
- For this experiment let's consider for simplicity the example of hourly data of a set of random climate conditions over many years.

So being discrete-time we always have a main index n that will be called $h \in \mathbb{N}_0$ for this example

Let h be the hour, d the day, m the month, and y the year, we have as an initial condition set for this map

- The total amount of years we are going to analyze (7 in this example from 2023 to 2029) so we imagine a mod 7 clock going from 0 to 6 where we wish the hourly data to be placed for each year

$$T_y = N$$

$$\omega_y = \frac{2\pi}{T_y}$$

- The number of months in a year

$$T_m = 12$$

$$\omega_m = \frac{2\pi}{T_m}$$

- We are going to use $T_d = 31$. The total number of days in a given month, as the maximum number of days in the current month will be a number from 28 to 31. This can be taken from a function that takes the year and the month and returns the total number of days in that month. Choosing 31 as a fixed T_d makes it easier to see the months that have 30 or 28/29 because some slots of the mod 31 wheel will not have any hour going around them.

$$T_d = monthrange(2023, 12) = 31$$

- or for a month 2 in a leap year

$$T_d = monthrange(2024, 2) = 29$$

$$\omega_d = \frac{2\pi}{T_d}$$

- The total number of hours in a given day

$$T_h = 24$$

$$\omega_h = \frac{2\pi}{T_h}$$

- The total number of minutes in an hour

$$T_{min} = 60$$

$$\omega_{min} = \frac{2\pi}{T_{min}}$$

- The total number of seconds in a minute

$$T_{sec} = 60$$

$$\omega_{sec} = \frac{2\pi}{T_{sec}}$$

- Etc, we could go to the femtosecond or to the Planck time scale, it doesn't matter now

2.1 Example set of compass numbers to plot the example hourly data of N years as an atlas

- Compass numbers for the year in hourly example data

$$\hat{y} = r_y \Theta^{\omega_y y}$$

$$\hat{m} = r_m \Theta^{\omega_m m}$$

$$\hat{d} = r_d \Theta^{\omega_d d}$$

$$\hat{h} = r_h \Theta^{\omega_h h}$$

2.2 $\hat{A}$ We get the atlas by adding the compass numbers

We take into our advantage that adding complex numbers is as easy as adding vectors, so we concatenate the different cogs of our clocks accordingly using the different radius r in our representation.

$$\hat{A} = \hat{y} + \hat{m} + \hat{d} + \hat{h}$$

2.2.1 Unpacking the atlas to a more scary-looking expression helps us circle back and remember we are just adding vectors in the tip of our set of finite-beam-compasses:

We can unpack this to its core, but keep in mind we are just adding vectors, meaning where to place the pencil thinking 'orthogonally' the horizontal and the vertical axis. These projections of the vector to the axis are called formally the 'Real' and 'Imaginary' parts of the complex number, but again, we are looking for friendlier expressions…

$$\hat{A} = r_y \Theta^{\omega_y y} + r_m \Theta^{\omega_m m} + r_d \Theta^{\omega_d d} + r_h \Theta^{\omega_h h}$$

$$\hat{A} = r_y e^{i\frac{2\pi}{T_y}y} + r_m e^{i\frac{2\pi}{T_m}m} + r_d e^{i\frac{2\pi}{T_d}d} + r_h e^{i\frac{2\pi}{T_h}h}$$

$$\hat{A} = r_y \left(\cos\left(\frac{2\pi}{T_y}y\right) + i\sin\left(\frac{2\pi}{T_y}y\right)\right) + r_m \left(\cos\left(\frac{2\pi}{T_m}m\right) + i\sin\left(\frac{2\pi}{T_m}m\right)\right) + r_d \left(\cos\left(\frac{2\pi}{T_d}d\right) + i\sin\left(\frac{2\pi}{T_d}d\right)\right) + r_h \left(\c$$

3 A more general expression example (x_t, y_t, z_t) being any 3D sequence over the discrete base time t

$$\hat{A}_t = \hat{x} + \hat{y} + \hat{z} + \hat{t}$$

$$\hat{A}_t = r_x \Theta^{\omega_x x} + r_y \Theta^{\omega_y y} + r_z \Theta^{\omega_z z} + r_t \Theta^{\omega_t t}$$

$$\hat{A}_t = r_x e^{i\frac{2\pi}{T_x}x} + r_y e^{i\frac{2\pi}{T_y}y} + r_z e^{i\frac{2\pi}{T_z}z} + r_t e^{i\frac{2\pi}{T_t}t}$$

4 What the $exp()$ function is really doing behind the scenes is a important story

- Exponential functions can be represented by infinite series (in practice we use a finite ammount of terms and tricks to improve precission)

$$\exp(\theta) = e^{\theta} = \sum_{n=0}^{\infty} \frac{\theta^n}{n!} = 1 + \frac{\theta}{1!} + \frac{\theta^2}{2!} + \frac{\theta^3}{3!} + \cdots$$

$$\Theta^{\theta} = e^{i\theta} = \cos(\theta) + i\sin(\theta)$$

- If you unpack this summation, you will find that it has a way to associate even exponents as real numbers and odd exponents will have the i attached to them. This is the key to understand what we actually do numerically

$$e^{i\theta} = \sum_{n=0}^{\infty} \frac{(i\theta)^n}{n!}$$

- We might remember that cosine is the even one and sine the odd one to see that:

$$\cos(\theta) = \sum_{n=0}^{\infty} (-1)^n \frac{\theta^{2n}}{(2n)!} = 1 - \frac{\theta^2}{2!} + \frac{\theta^4}{4!} - \frac{\theta^6}{6!} + \cdots$$

$$\sin(\theta) = \sum_{n=0}^{\infty} (-1)^n \frac{\theta^{2n+1}}{(2n+1)!} = \theta - \frac{\theta^3}{3!} + \frac{\theta^5}{5!} - \frac{\theta^7}{7!} + \cdots$$

- So a full unpacking of what we are doing starts with understanding the dynamics of this sum

$$e^{i\theta} = \sum_{n=0}^{\infty} \left[(-1)^n \frac{\theta^{2n}}{(2n)!} \right] + i \sum_{n=0}^{\infty} \left[(-1)^n \frac{\theta^{2n+1}}{(2n+1)!} \right]$$

5 Now we make some code to explain the idea better

We are going to use pandas and numpy, and take advantage of the datetime format to simplify the creation of dummy data to place in the year atlas experiment and initialize the integer sequences y, m, d, h , being h in this example the mod 24 hour index .

The code can be improved dramatically, but the idea is to make it simple to follow just to place the cogs in motion and understand the process.

[123]:
```
import pandas as pd
import numpy as np
import calendar
import re
import matplotlib.pyplot as plt
from matplotlib import cm
import random
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import image
import matplotlib.animation as animation
from matplotlib.animation import FuncAnimation, FFMpegWriter
import warnings
warnings.filterwarnings("ignore")
```

5.1 We create some sample data to give color to the idea

[124]:
```
def create_random_climate_data(start,end,freq):
    dt_index = pd.date_range(start=start, end=end, freq=freq)

    # Create a dataframe with random temperature, humidity, and pressure data

    data = {'temperature': np.random.uniform(low=10, high=30,␣
 ↪size=len(dt_index)),
        'humidity': np.random.uniform(low=20, high=80, size=len(dt_index)),
        'pressure': np.random.uniform(low=980, high=1020, size=len(dt_index))}

    climate_data = pd.DataFrame(data=data, index=dt_index)

    return climate_data
```

5.1.1 We extract components from the dataframe as arrays

[125]:
```
def extract_climate_data(df):
    # Extract the datetime components as numpy arrays
    year = df.index.year.values
    month = df.index.month.values
    day = df.index.day.values
    hour = df.index.hour.values
    minute = df.index.minute.values
    second = df.index.second.values

    # Extract the data columns as numpy arrays
    temperature = df['temperature'].values
    humidity = df['humidity'].values
    pressure = df['pressure'].values

    return year, month, day, hour, minute, second, temperature, humidity,␣
 ↪pressure
```

5.1.2 We create a 7 years hourly data example -

- For now we will just color the hour of the day using mod 24 hsv colors

To make the code cleaner hour will be the mod 24 hours given by the datetime format, and h will be the main integer index of the complete hourly dataframe .Note the connection between modular-arithmetic and the way this compass numbers behave:

[126]:
```
start = pd.Timestamp('2023-01-01 00:00:00')
end = pd.Timestamp('2029-12-31 23:00:00')
freq ='H'
df  = create_random_climate_data(start,end,freq)
year, month, day, hour, minute, second, temperature, humidity, pressure =␣
 ↪extract_climate_data(df)
h = np.arange(len(df.index))
h.size
```

[126]:
```
61368
```

5.1.3 Days in each month

Thirty days have September, April, June, and November, all the rest have thirty-one.

February has twenty-eight, but leap year coming one in four, February then has one day more.

Leap Leap years are calculated based on the rules established by the Gregorian calendar, which is the calendar currently used by most of the world.

To calculate whether a given year is a leap year or not, you can follow these rules:

If the year is divisible by 4, it is a leap year, except for: If the year is also divisible by 100, it is not a leap year, except for: If the year is also divisible by 400, it is a leap year. Using these rules, you can determine whether a particular year is a leap year or not. For example:

2000 was a leap year because it is divisible by 4 and 400. 1900 was not a leap year because it is divisible by 4 and 100, but not by 400. 2024 will be a leap year because it is divisible by 4.

[127]:
```
def max_days_in_month(year, month):
    if month == 2 and calendar.isleap(year):
        return 29
    return calendar.monthrange(year, month)[1]
```

Cleaning up

[128]:
```
h    = h
h_24 = hour
d    = day
m    = month
y    = year

T_h =   24
T_d =   31 # to be changed inside the loop in the function maxdays 28,29,30 or␣
 ↪31
T_m =   12
T_y =   7

r_y =   1
r_m =   1
r_d =   1
r_h =   1


# Initializing a dummy atlas for preallocation
y_hat = r_y*np.exp((2*np.pi*1j/T_y)*y)
m_hat = r_m*np.exp((2*np.pi*1j/T_m)*m)
d_hat = r_d*np.exp((2*np.pi*1j/T_d)*d)
h_hat = r_h*np.exp((2*np.pi*1j/T_h)*h)
atlas = y_hat + m_hat +d_hat+h_hat
```

5.1.4 We take a look of how the empty structure is looking

[129]:
```
plt.rcParams.update({
        "lines.color": "black",
        "patch.edgecolor": "white",
        "text.color": "white",
        "axes.facecolor": "black",
        "axes.edgecolor": "black",
        "axes.labelcolor": "black",
```

```
        "xtick.color": "white",
        "ytick.color": "white",
        "grid.color": "gray",
        "figure.facecolor": "black",
        "figure.edgecolor": "black",
        "savefig.facecolor": "black",
        "savefig.edgecolor": "black"})
```

[130]:
```
def init_atlas(sx,sy):

    plt.figure(figsize=(sx,sy))



def see_atlas(atlas,symbol_size,symbol_shape):
    #plt.scatter(0,0,s=10,marker='*')
    plt.scatter(atlas.real,atlas.imag,s=symbol_size,marker=symbol_shape)


def color_atlas(atlas,symbol_size,symbol_shape,symbol_color,cmap,T_color):

    get_color =  cm.get_cmap(cmap, T_color)
    plt.scatter(atlas.real,atlas.
 ↪imag,s=symbol_size,marker=symbol_shape,color=get_color(symbol_color%T_color))

def␣
 ↪atlas_view(symbol,T_symbol,atlas,symbol_size,symbol_shape,symbol_color,cmap,T_color,char_fl
 ↪


    get_color =  cm.get_cmap(cmap, T_color)
    get_char_color = cm.get_cmap(cmap_char, T_char)
    plt.scatter(atlas.real,atlas.
 ↪imag,s=symbol_size,marker=symbol_shape,color=get_color(symbol_color%T_color))
    if char_flag:
        plt.text(atlas.real,atlas.
 ↪imag,str(symbol%T_symbol),size=char_size,color=get_char_color(char_color%T_symbol))
```

6 How counting in base two gives the 8 basic symmetries of the number $a + bi$ by shifting a and b individually 180 and doing an axis permutation

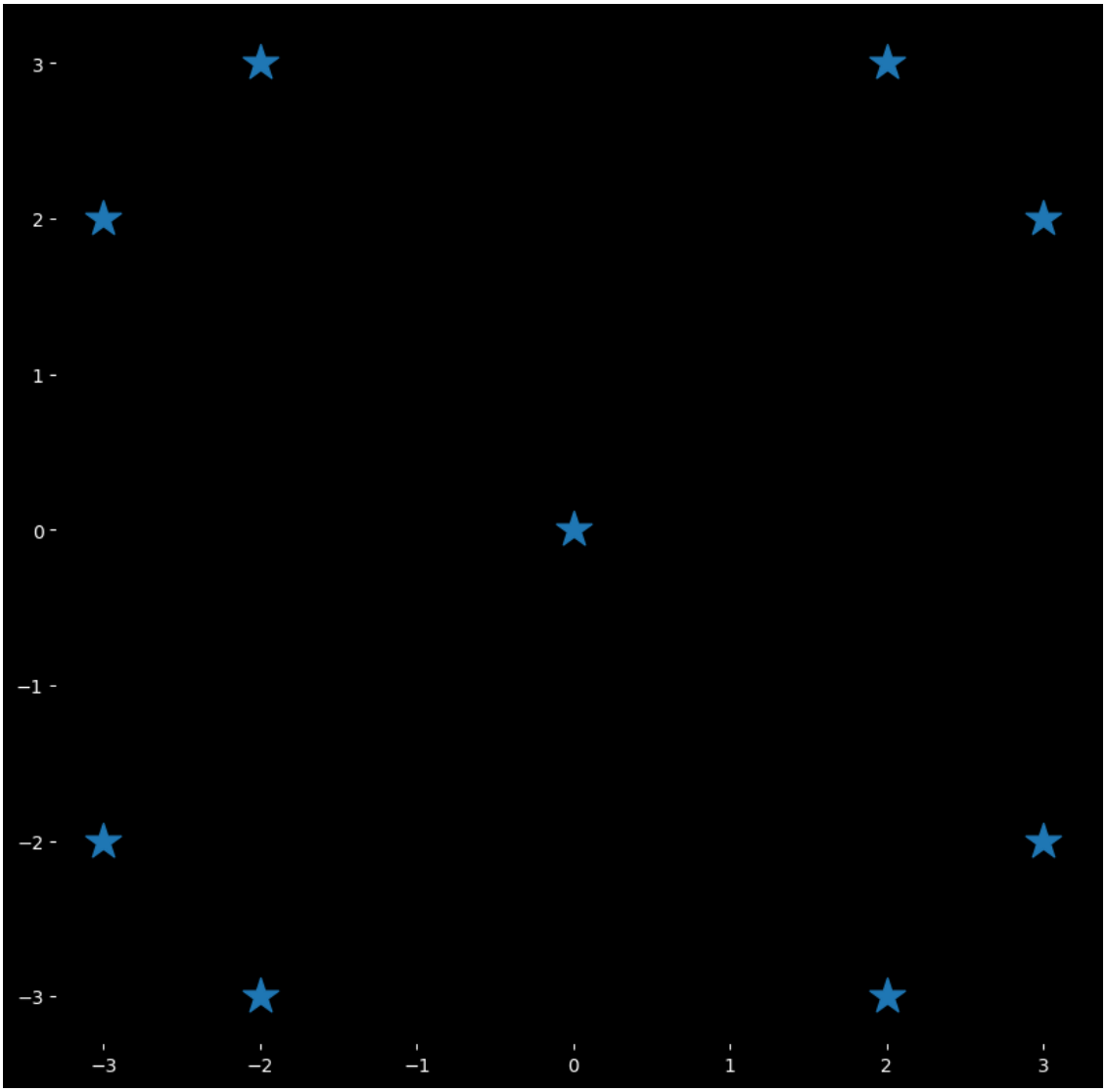
Binary	Decimal	MSb	LSb Symmetry
b2 b1 b0	0 to 7	b2 Permute	b1 b0
———	———	———	———-
000	0	0	a + i*b

Binary	Decimal	MSb	LSb Symmetry
001	1	0	a - i*b
010	2	0	-a + i*b
011	3	0	-a - i*b
100	4	1	b + i*a
101	5	1	b - i*a
110	6	1	-b + i*a
111	7	1	-b - i*a

6.0.1 We can use this new function for a simple array of 8 Symmetric Compass Numbers

Remember that i is noted as j in python .

[131]:
```
init_atlas(10,10)
see_atlas(np.array([0, 2+3j,2-3j,-2+3j,-2-3j,3+2j,3-2j,-3+2j,-3-2j]),400,'*')
```



6.1 The sum of this 8 compass numbers is zero

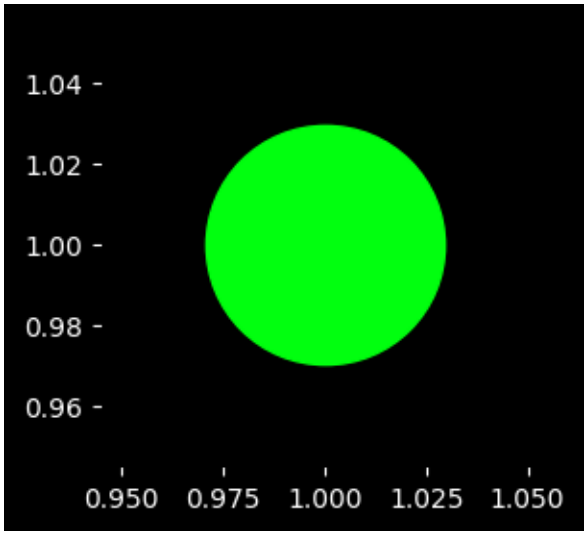
[132]:
```
np.array([0, 2+3j,2-3j,-2+3j,-2-3j,3+2j,3-2j,-3+2j,-3-2j]).sum()
```

[132]:
```
0j
```

We can use the function to plot a simple compass number

[133]:
```
a=1
b=1
size=8000
cmap='hsv'
T_color=T_h
```

```
init_atlas(3,3)
marker='o'
dot_position=a+1j*b
color=8
color_atlas(dot_position,size,marker,color,cmap,T_color)
```



7 First plot of the RAW atlas, without any filtering in the angle and the radius movements of our Compass

Imagine you are not from earth but you where born in an earth like planet with a similar development,but your world has an accepted calendar where months are always 33 days and your days have only 8 hours , and your year has let's say 16 months.(I just chose numbers at random)

Imagine that in your mailbox you receive a list of 61368 2D numbers (you understand them because they where written in binary, and in two columns) and you also recieve a little gold compass in a quartz box.

The idea is to imagine for a second you receive a two signals that tell you each movement of your compass where you plot a color indicating in this example the hour with a symbol (maybe a color or the symbol itself, or maybe a deviation from time what would be add an other weighted complex number representing the symbol to the time representation)

7.1 Imagine you want to make sense of this noise : This mess has all $r_k = 1$ and the 24 colors represent the integer hour mod 24 of the day in hsv.

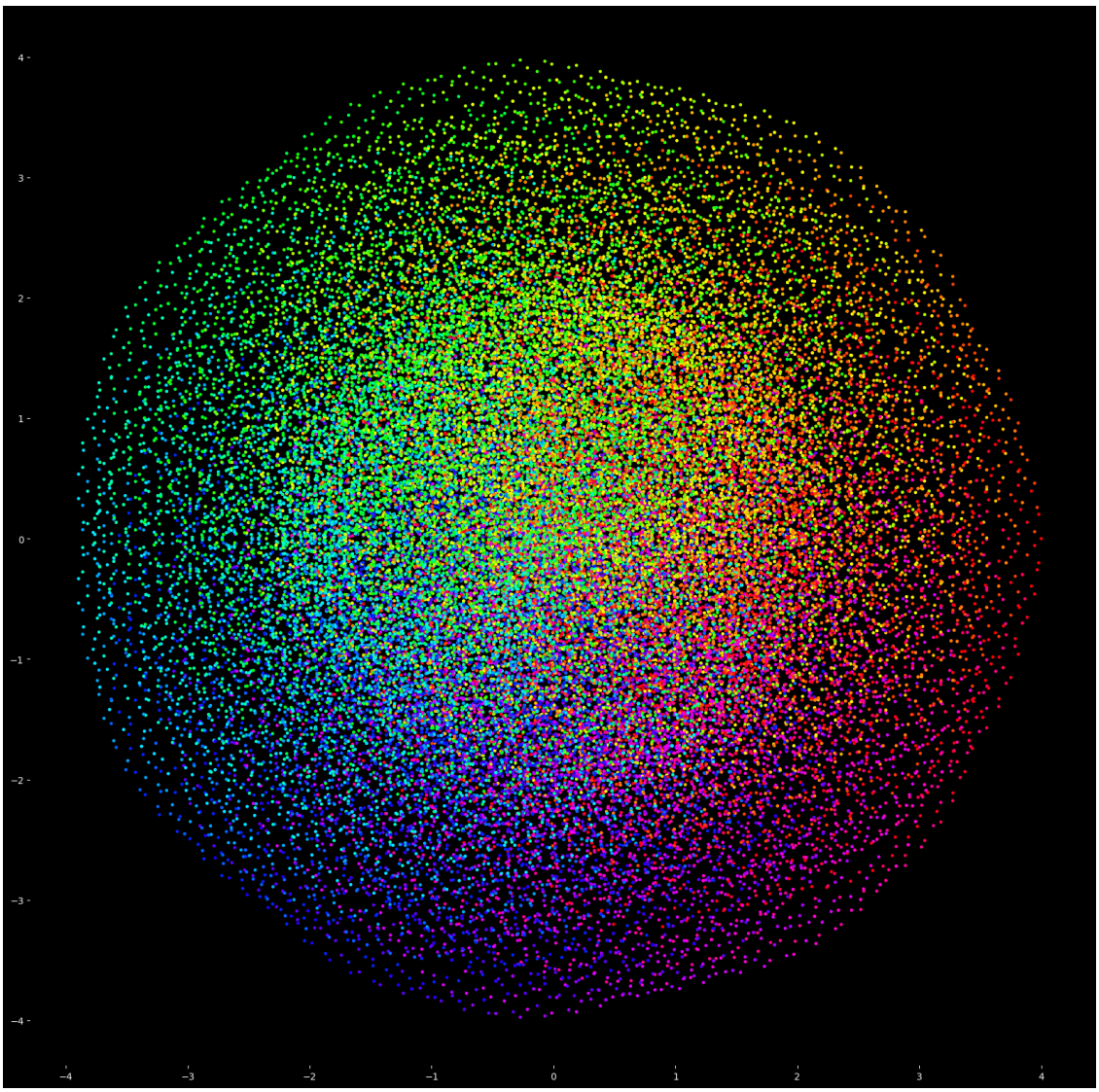
We make

$$T_{color} = T_h$$

And

$$S_{color} = hour$$

[134]:
```
size=1
cmap='hsv'
T_color=T_h
marker='o'
symbol_color=hour

init_atlas(20,20)
color_atlas(atlas,size*6,marker,symbol_color,cmap,T_color)
image_path = './img/2aliased_year_atlas_first_plot.png'
#plt.savefig(image_path)
```



8 The idea of filter

In this example (on previous plot) we can see a noisy representation of a hourly calendar from 2023 to 2029. The color represents the hour of the day in a wheel of 24 slots , counterclockwise from 0 to 23.

But we cannot see any pattern if we don't see it develop step by step (imagine you just got that image along with the coordinates and the compass) .

8.1 The noisy hourly data index ($r_k = 1$) vs the weighted(filtered) data index

We have the noisy set we imagine we receive from another planet :

$$y_h = \Theta^{\omega_y y_h}$$

$$m_h = \Theta^{\omega_m m_h}$$

$$d_h = \Theta^{\omega_d d_h}$$

$$h_h = \Theta^{\omega_h h}$$

$$\hat{A}_h = y_h + m_h + d_h + h_h$$

And the organized set we need to make sense of the data:

$$\hat{y}_h = r_y \Theta^{\omega_y y_h}$$

$$\hat{m}_h = r_m \Theta^{\omega_m m_h}$$

$$\hat{d}_h = r_d \Theta^{\omega_d d_h}$$

$$\hat{h}_h = r_h \Theta^{\omega_h h}$$

$$\hat{F}_h = \hat{y}_h + \hat{m}_h + \hat{d}_h + \hat{h}_h$$

So the filter we can call $z_h \in \mathbb{C}$ would be something like this if we knew the solution to the puzzle:

$$z_h = \frac{r_y \Theta^{\omega_y y_h} + r_m \Theta^{\omega_m m_h} + r_d \Theta^{\omega_d d_h} + r_h \Theta^{\omega_h h}}{\Theta^{\omega_y y_h} + \Theta^{\omega_m m_h} + \Theta^{\omega_d d_h} + \Theta^{\omega_h h}}$$

But we could have started with a generic filter we test on our data and adjust acording to the patterns we obtain :

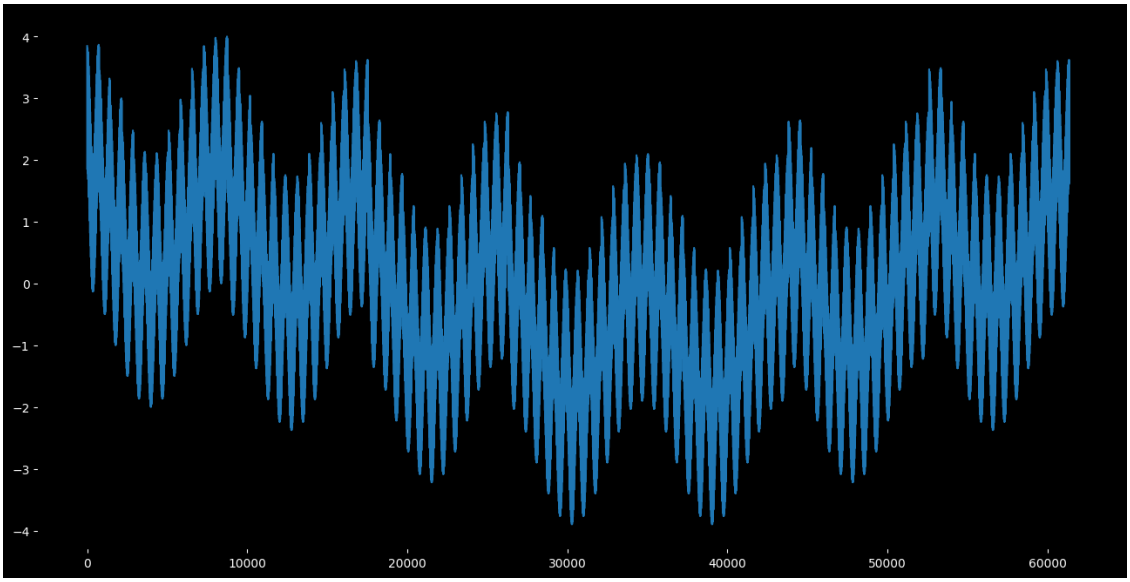
$$z_h = \frac{r_u \Theta^{\omega_u u_h} + r_v \Theta^{\omega_v v_h} + r_w \Theta^{\omega_w w_h} + r_s \Theta^{\omega_s s}}{\Theta^{\omega_y y_h} + \Theta^{\omega_m m_h} + \Theta^{\omega_d d_h} + \Theta^{\omega_h h}}$$

$$z_h = \frac{r_u \Theta^{\omega_u u_h} + r_v \Theta^{\omega_v v_h} + r_w \Theta^{\omega_w w_h} + r_s \Theta^{\omega_s s}}{\hat{A}_{h-1}}$$

8.1.1 The real part (horizontal)

[135]:
```
init_atlas(16,8)
plt.plot(h,atlas.real)
```

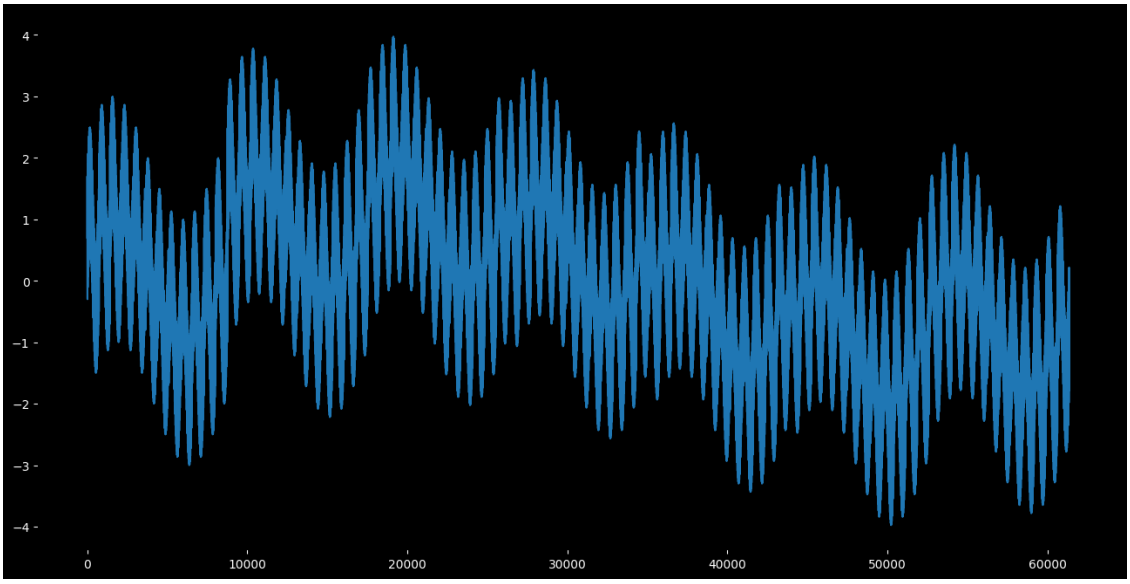
[135]: [<matplotlib.lines.Line2D at 0x7f0dd9078d00>]



8.1.2 The imaginary part (vertical)

[136]:
```
init_atlas(16,8)
plt.plot(h,atlas.imag)
```

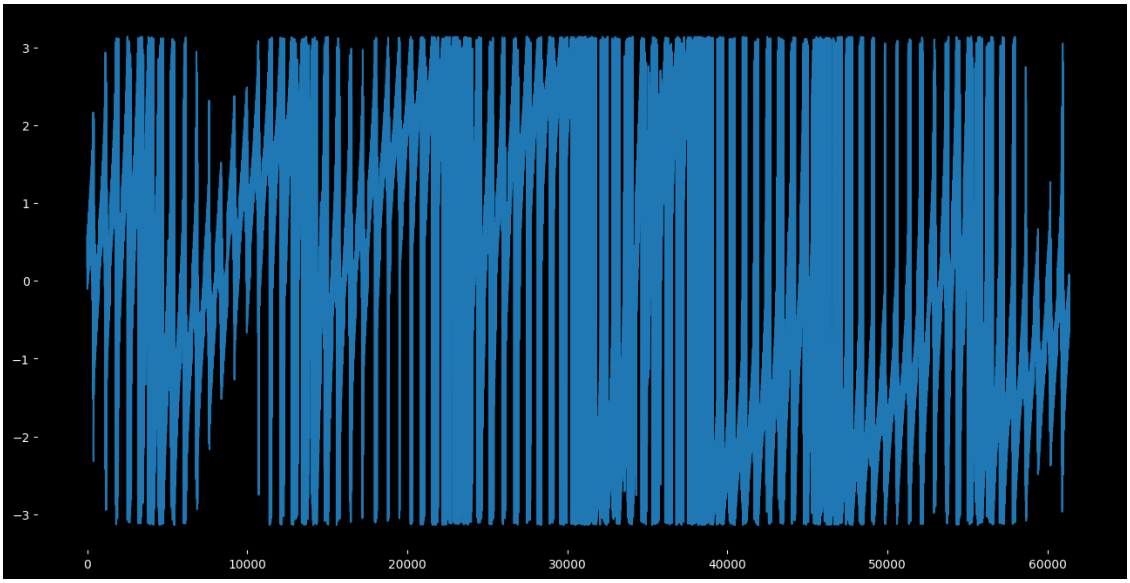
[136]: [<matplotlib.lines.Line2D at 0x7f0dd8713e20>]



8.1.3 The pre-filter angle of the $\hat{A}$ atlas over time

[137]:
```
init_atlas(16,8)
plt.plot(h,np.angle(atlas))
```

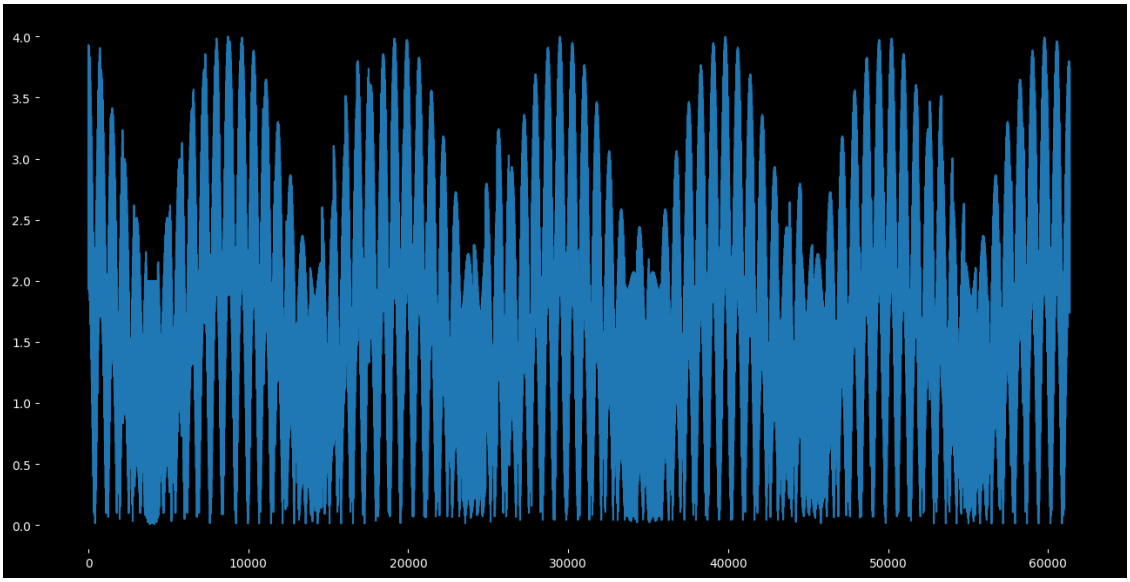
[137]: [<matplotlib.lines.Line2D at 0x7f0dd86ccb80>]



8.1.4 The pre-filter radius of the compass over time

[138]:
```
init_atlas(16,8)
plt.plot(h,np.abs(atlas))
```

[138]:
```
[<matplotlib.lines.Line2D at 0x7f0dda134ca0>]
```



8.1.5 Shifting the time color mapping

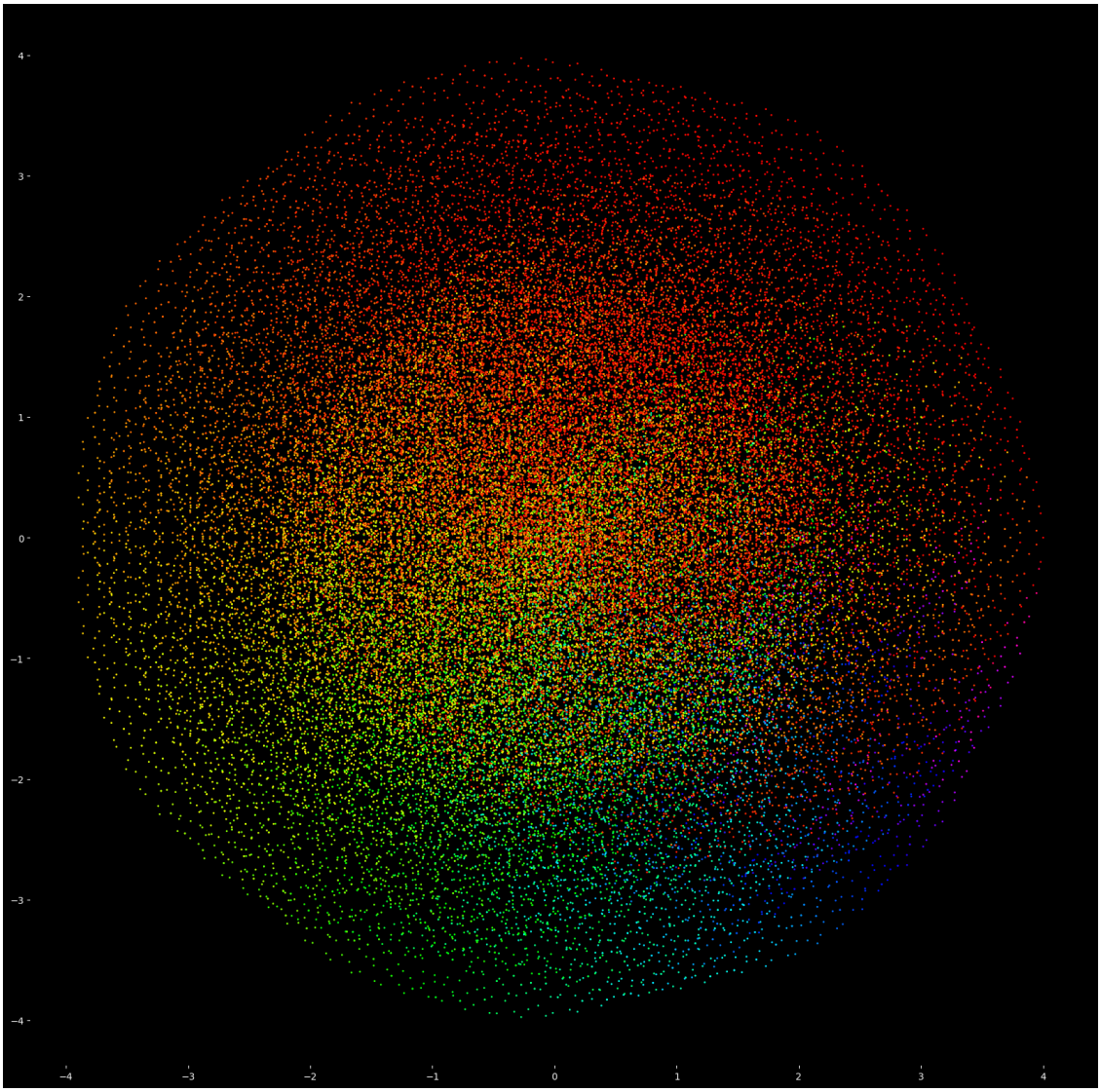
We make

$$T_{color} = T_h T_d T_m$$

And

$$S_{color} = hour * day * month$$

[139]:
```
size=1
cmap='hsv'
T_color=T_h*T_d*T_m
marker='o'
symbol_color=hour*day*month

init_atlas(20,20)
color_atlas(atlas,size,marker,symbol_color,cmap,T_color)
image_path = './img/shifted_color_aliased_year_atlas_first_plot.png'
#plt.savefig(image_path)
```

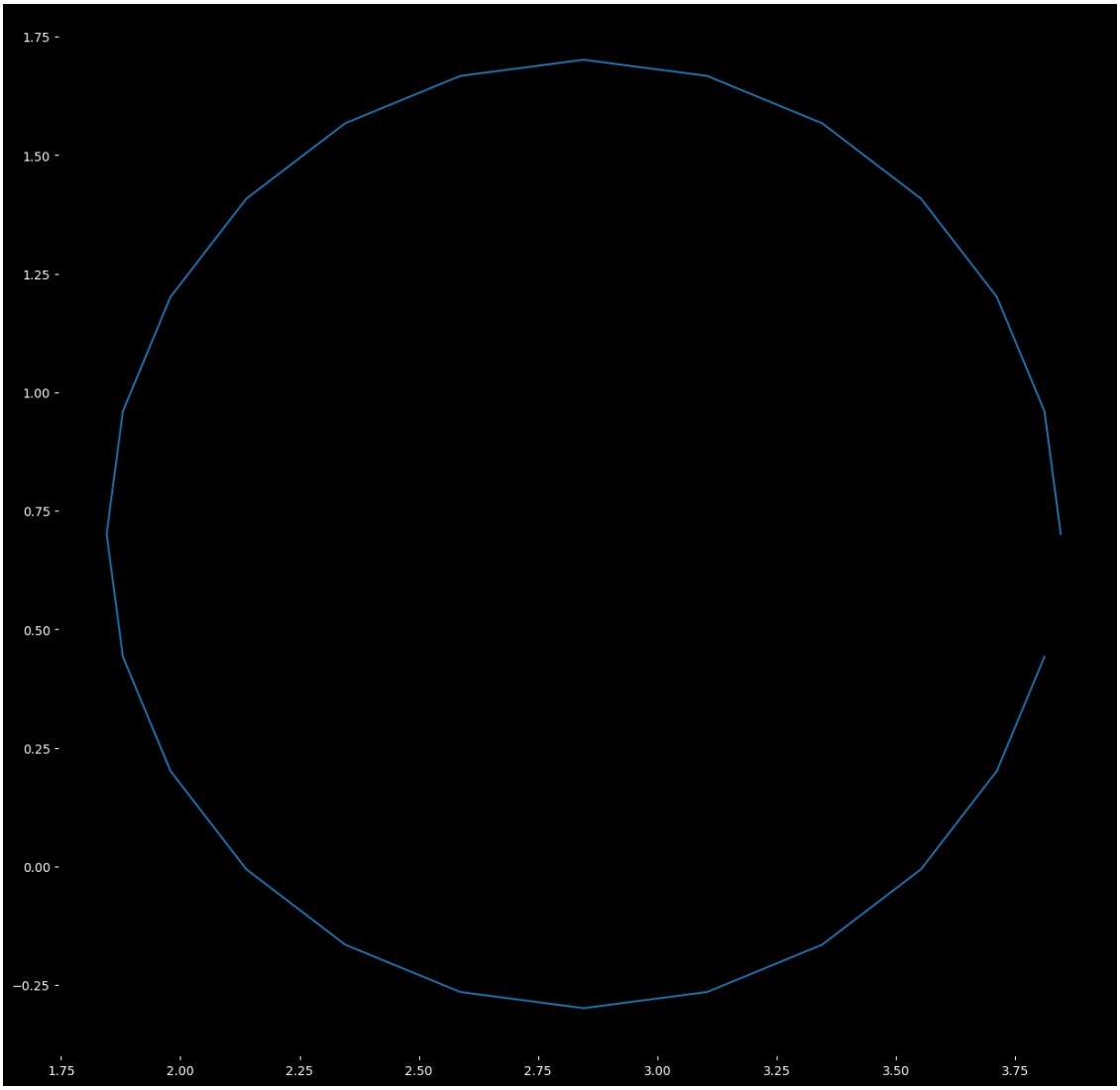


9 Here all radius are 1, we get this plots

9.1 1 day mesh graph - 24 symbols - main structure for hourly data

[140]:
```
plt.figure(figsize=(15,15))
plt.plot(atlas.real[0:24*1*1],atlas.imag[0:24*1*1])
```

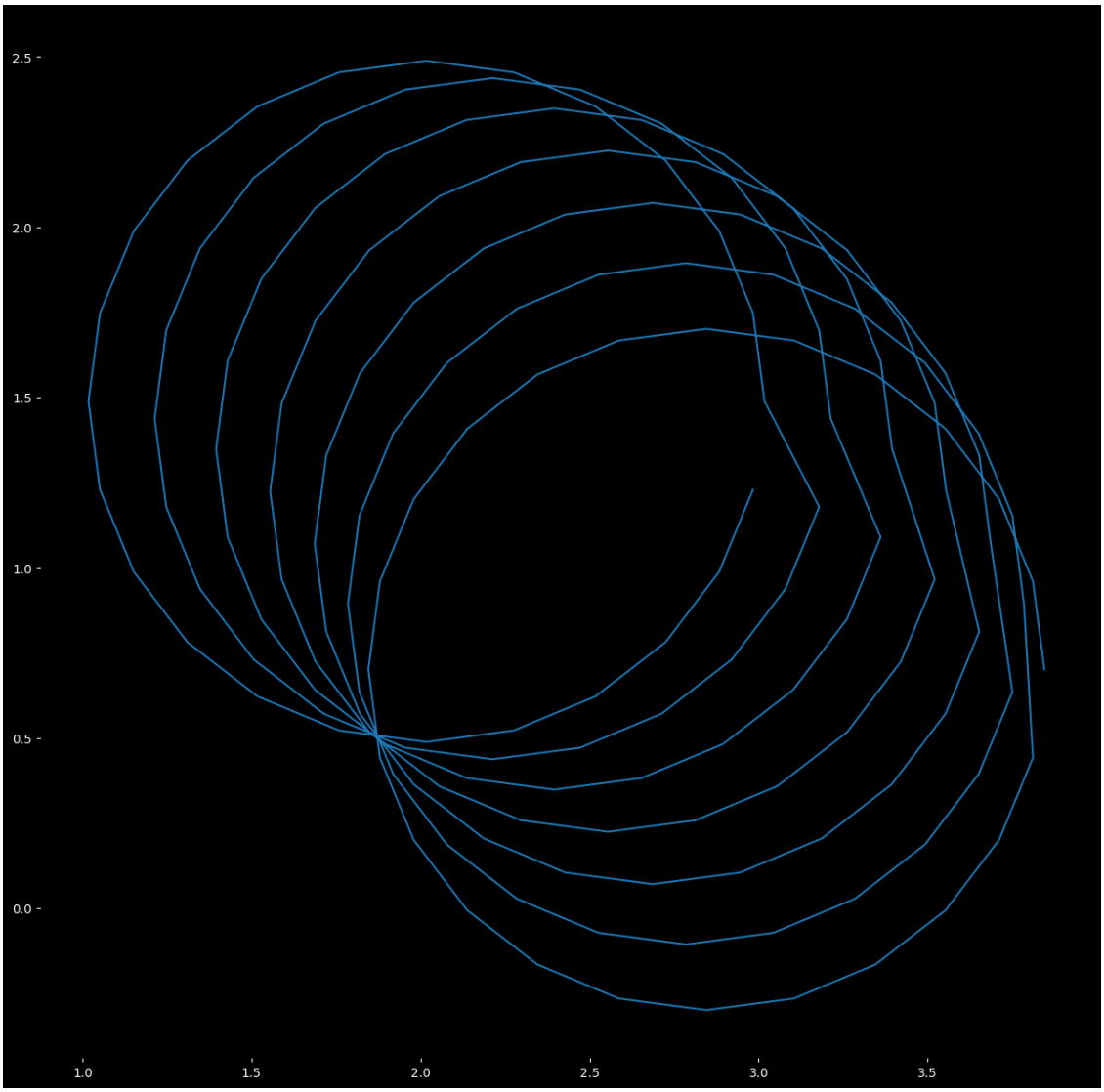
[140]: [<matplotlib.lines.Line2D at 0x7f0dd880ef50>]



9.2 7 day mesh graph

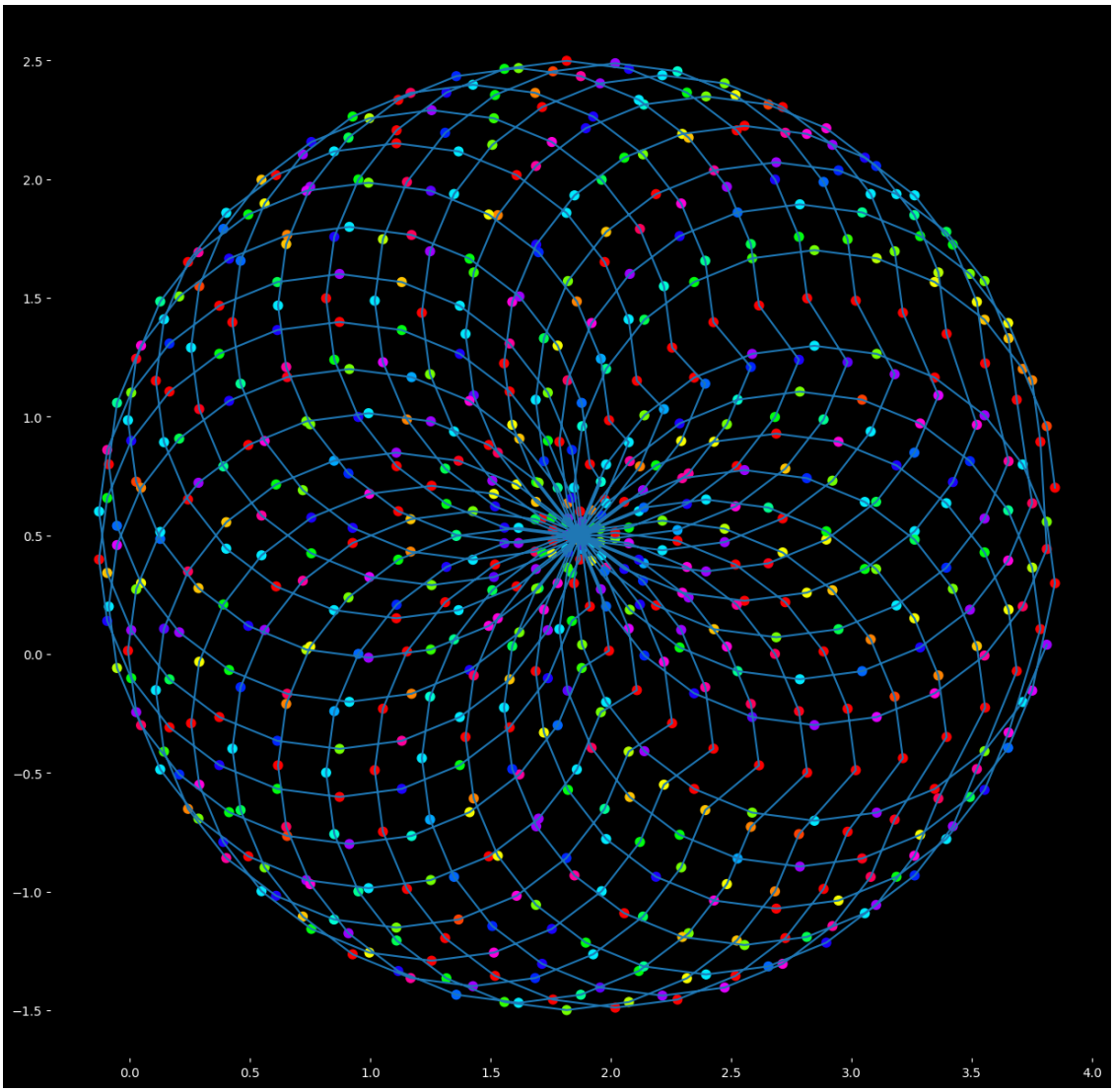
[141]:
```
plt.figure(figsize=(15,15))
plt.plot(atlas.real[0:24*1*7],atlas.imag[0:24*1*7])
```

[141]:
```
[<matplotlib.lines.Line2D at 0x7f0dd872fac0>]
```



9.3 1 month mesh graph

[142]:
```
plt.figure(figsize=(15,15))
plt.plot(atlas.real[0:24*30*1],atlas.imag[0:24*30*1])
color_atlas(atlas[0:24*30*1],size*50,marker,symbol_color[0:24*30*1],cmap,T_h)
```



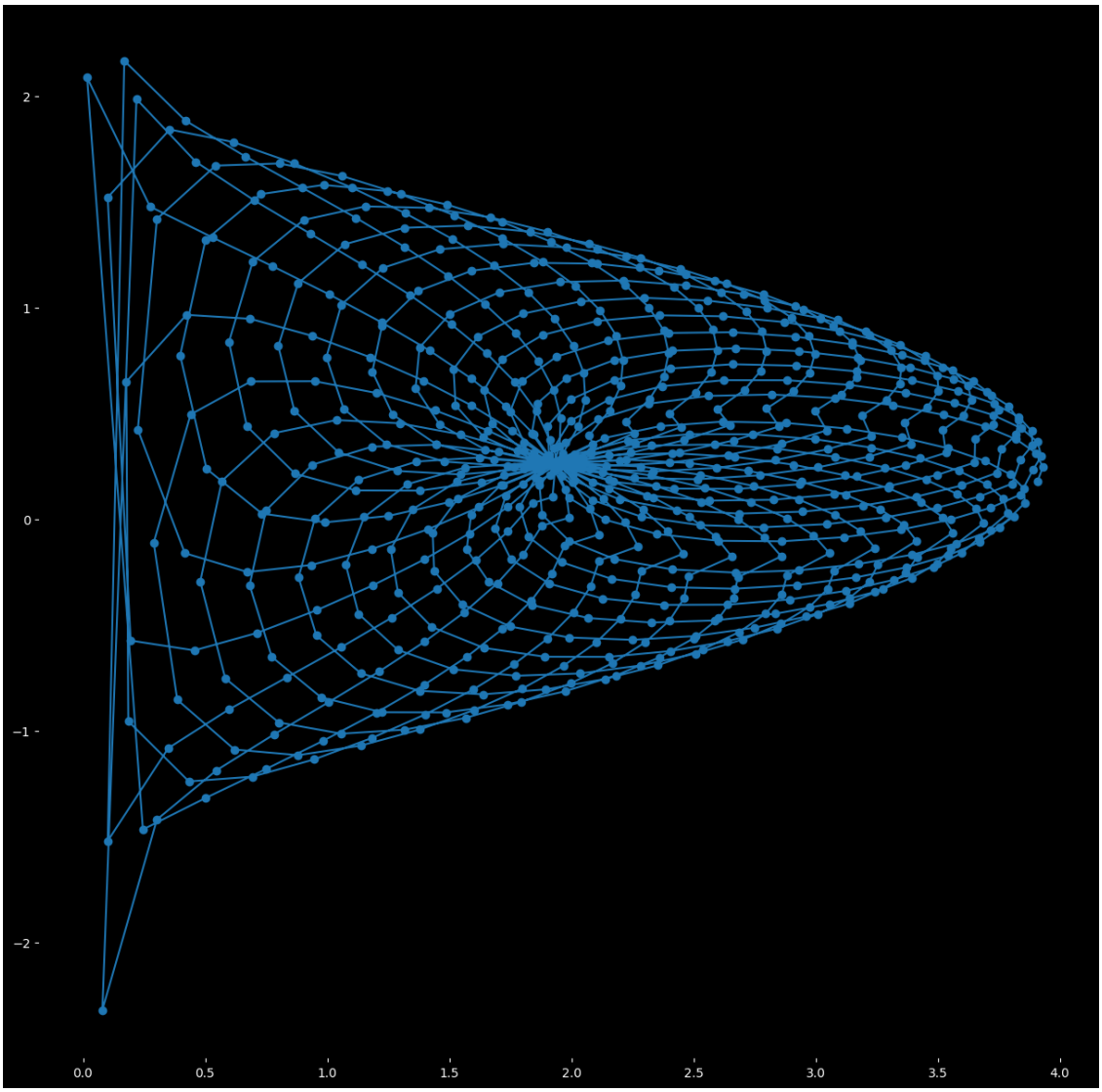
9.3.1 Is this normal? What would happen if we interpreted the 2D numbers to be the real and imaginary part instead of the radius and the angle ?

In the experiment we chose to imagine that we recieve a column of two numbers, and we chose to use them one to set the radius and the other to set the angle of the compass, so r and θ . Now this numbers in one of our experiments we use them as $z = r + i\theta$

$$B_h = r_h + i\theta_h$$

[143]:
```
bad_atlas=np.abs(atlas)+1j*np.angle(atlas)
plt.figure(figsize=(15,15))
plt.plot(bad_atlas.real[0:24*30*1],bad_atlas.imag[0:24*30*1])
plt.scatter(bad_atlas.real[0:24*30*1],bad_atlas.imag[0:24*30*1])
```

[143]: <matplotlib.collections.PathCollection at 0x7f0dd9024c70>



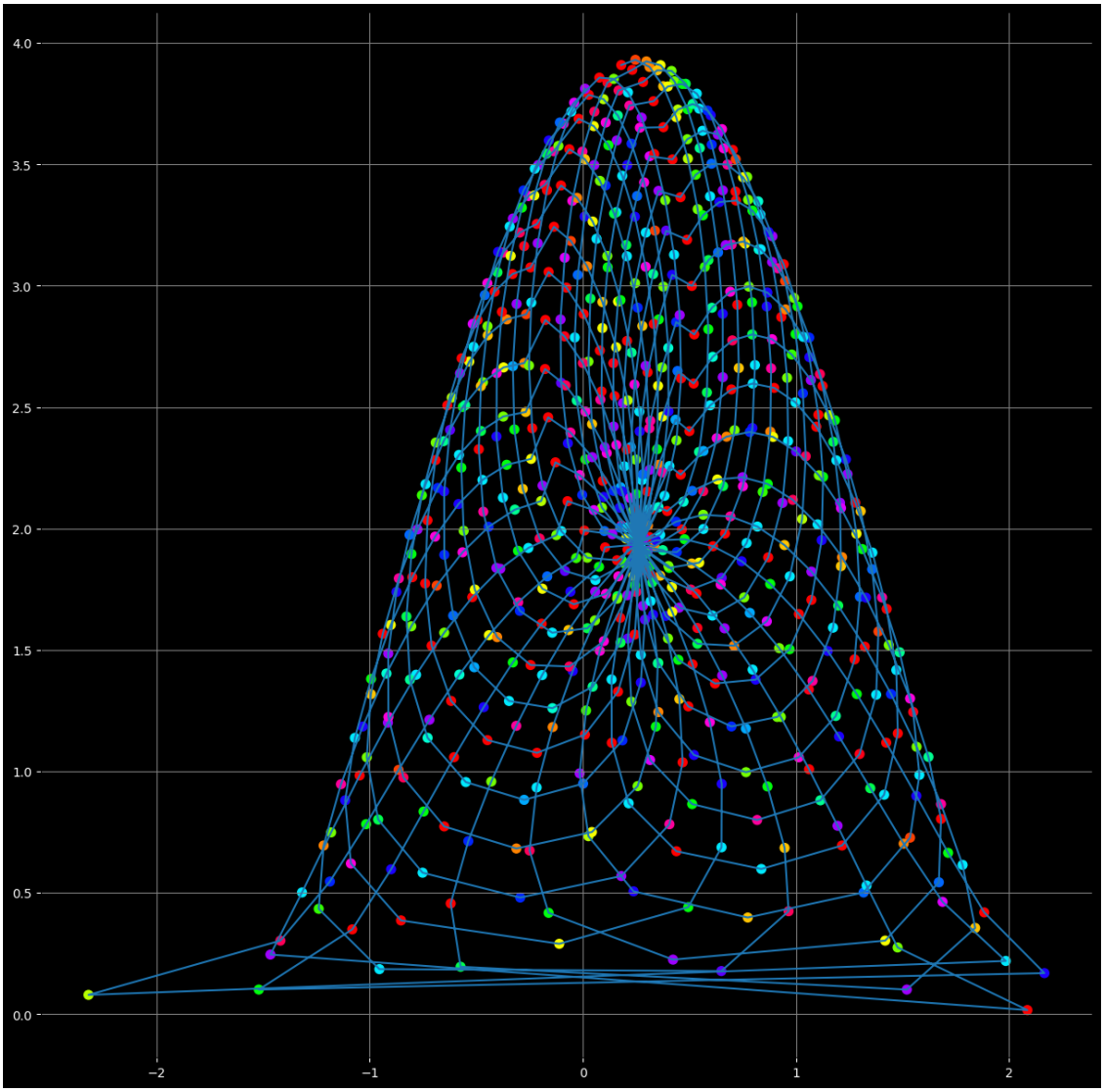
9.3.2 It's normal that is asking to be shifted?

Here 1 month hourly data again:

$$B_h = \theta_h + ir_h$$

[144]:
```
bad_atlas=np.abs(atlas)+1j*np.angle(atlas)
plt.figure(figsize=(15,15))
#plt.scatter(bad_atlas.imag[0:24*30*1],bad_atlas.real[0:24*30*1])
plt.plot(bad_atlas.imag[0:24*30*1],bad_atlas.real[0:24*30*1])
```

```
color_atlas(bad_atlas.imag[0:24*30*1]+1j*bad_atlas.real[0:
↪24*30*1],size*50,marker,symbol_color[0:24*30*1],cmap,T_h)
plt.grid()
```

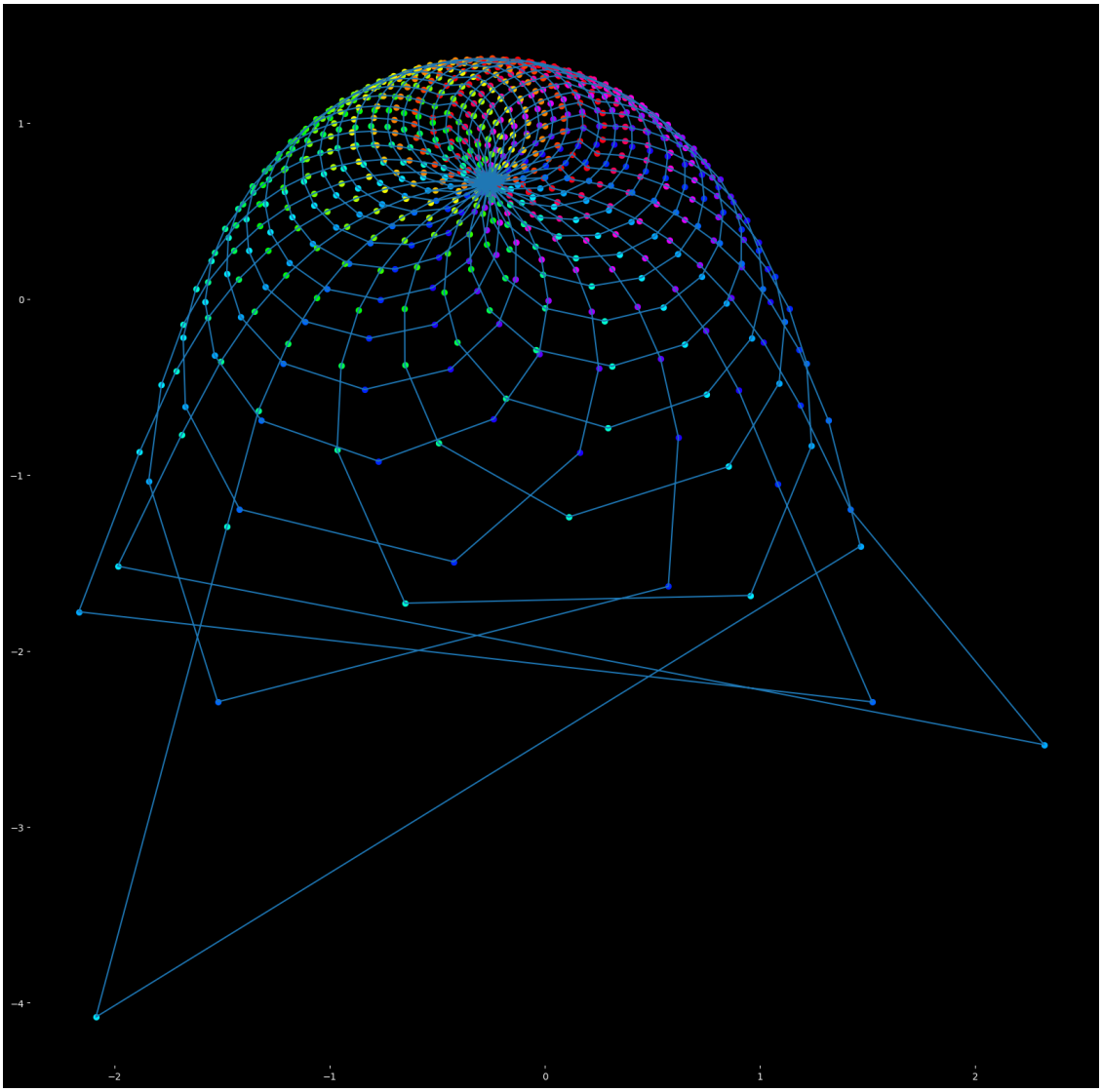


9.3.3 Using the $ilog(\hat{A}_h)$ we get the same idea

$$\log(re^{i\theta}) = log(r) + i\theta$$

[145]:
```
#bad_atlas_f2=np.abs(atlas_filter_2)+1j*np.angle(atlas_filter_2)
bad_atlas_f3=1j*np.log(atlas)

init_atlas(20,20)
plt.plot(bad_atlas_f3.real[0:24*31*1],bad_atlas_f3[0:24*31*1].imag)
```

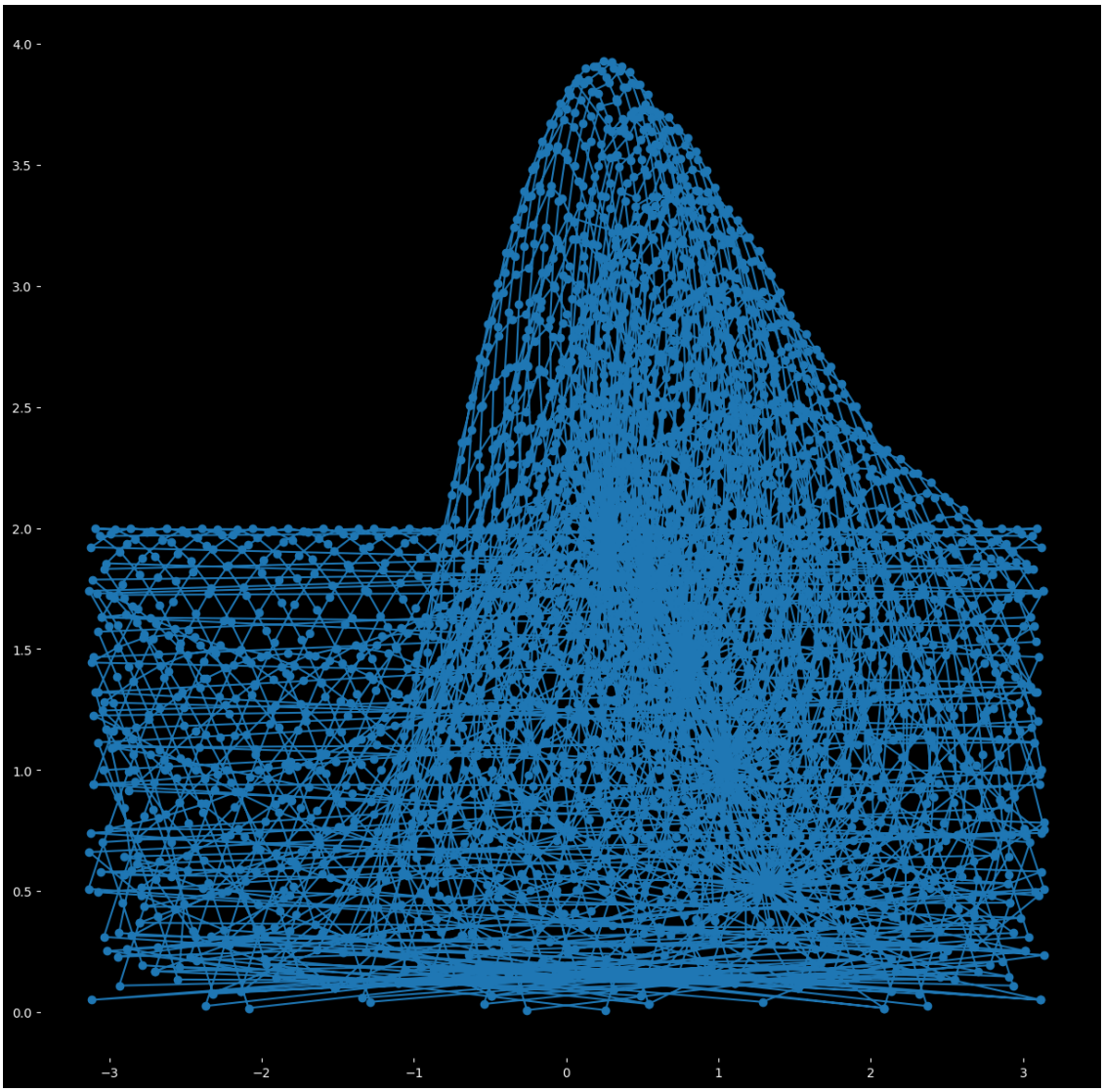
```
for n in h[0:24*31*1]:
    color_atlas(bad_atlas_f3.real[n]+1j*bad_atlas_f3.
 ↪imag[n],size*33,marker,hour[n],cmap,T_h)

    #plt.text(bad_atlas_f3.real[n],bad_atlas_f3.
 ↪imag[n],str(h[n]%T_h),size=6,color='gray')

    #plt.text(bad_atlas_f2.imag[n]-h_hat_f2.real[n],bad_atlas_f2.
 ↪real[n]-h_hat_f2.imag[n],str(d[h]),size=18,color='black')
    #plt.text(atlas_filter_2.real[n]-h_hat_f2.real[n],atlas_filter_2.
 ↪imag[n]-h_hat_f2.imag[n],str(d[n]),size=13,color='gray')
```



9.3.4 The same for 6 months

[146]:
```
plt.figure(figsize=(15,15))
plt.plot(bad_atlas.imag[0:24*30*6],bad_atlas.real[0:24*30*6])
plt.scatter(bad_atlas.imag[0:24*30*6],bad_atlas.real[0:24*30*6])
```

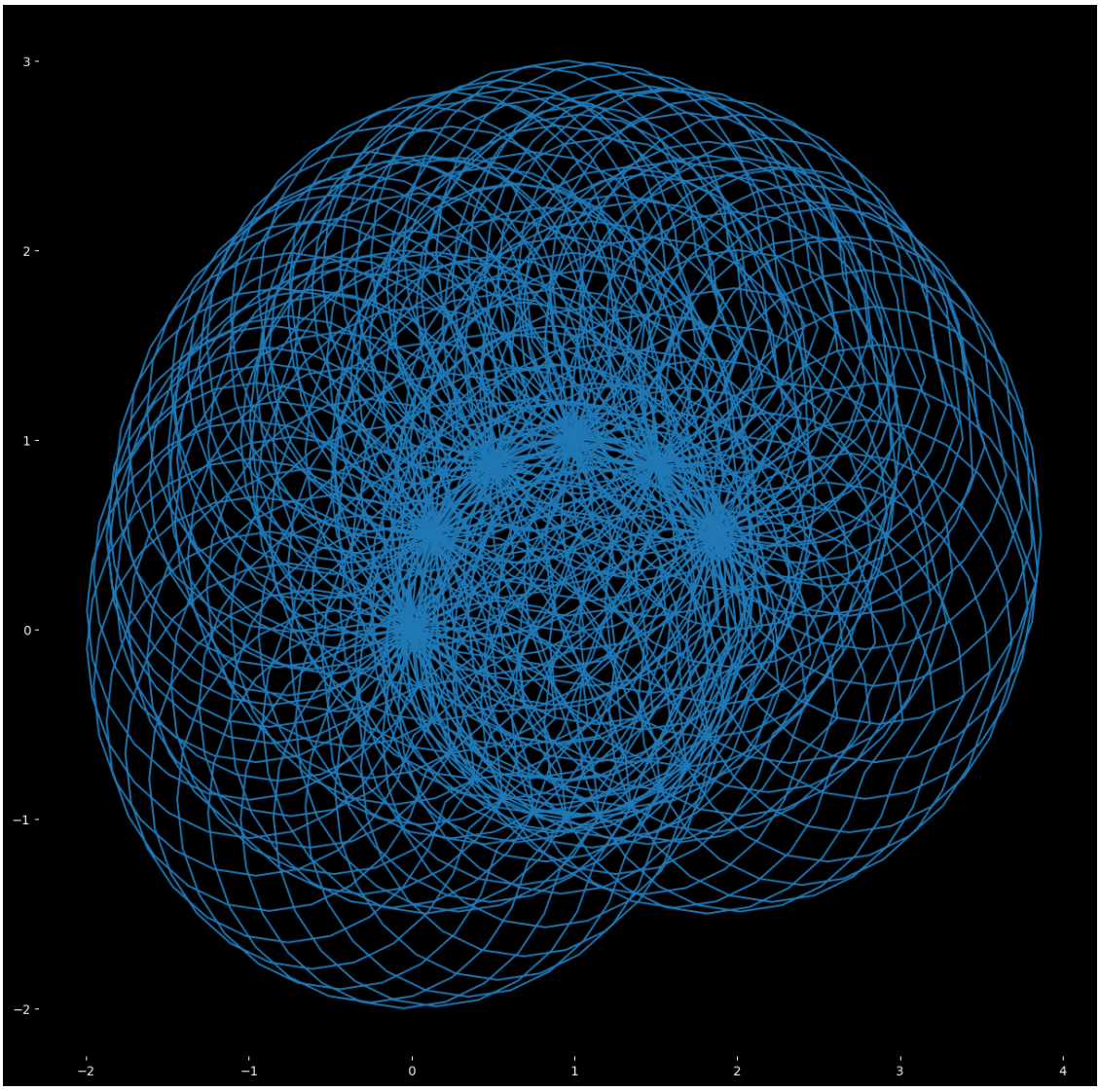
[146]: <matplotlib.collections.PathCollection at 0x7f0dd82859c0>



9.3.5 6 months mesh graph - Can you see the 6 months being the crowded intersections

[147]:
```
plt.figure(figsize=(15,15))
plt.plot(atlas.real[0:24*30*6],atlas.imag[0:24*30*6])
```

[147]:
```
[<matplotlib.lines.Line2D at 0x7f0dd83050f0>]
```



9.4 Filter 0 for seven years hourly: Lets shift the radius of the Compass Numbers to assign weights according to the size of the time scale

- 7 years hourly data

This is not the optimal visualization, but to get an initial feeling we could set the radius of the hourly data compass acording to the

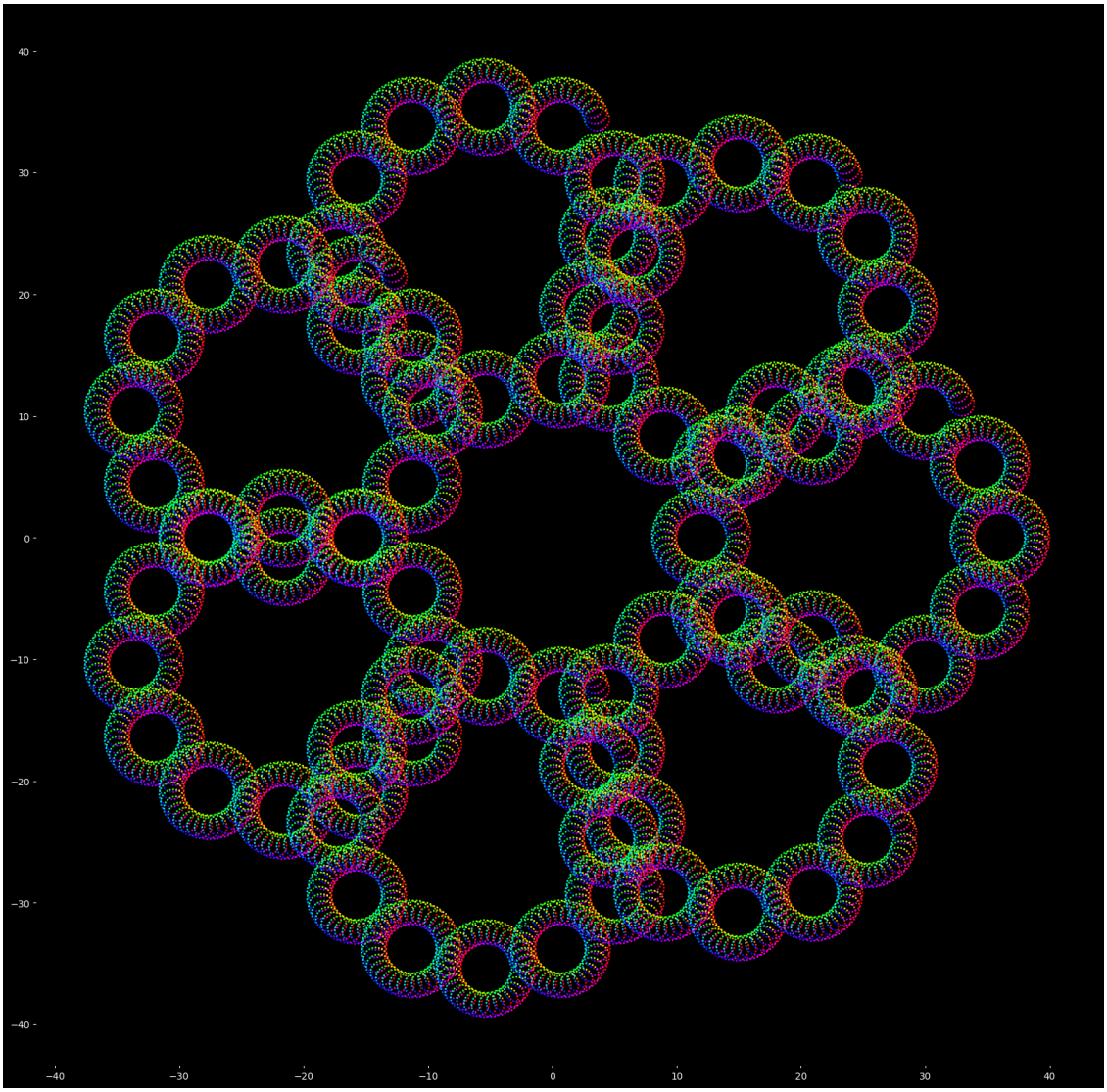
$$r_y = 24$$

$$r_m = 12$$

$$r_d = 3$$

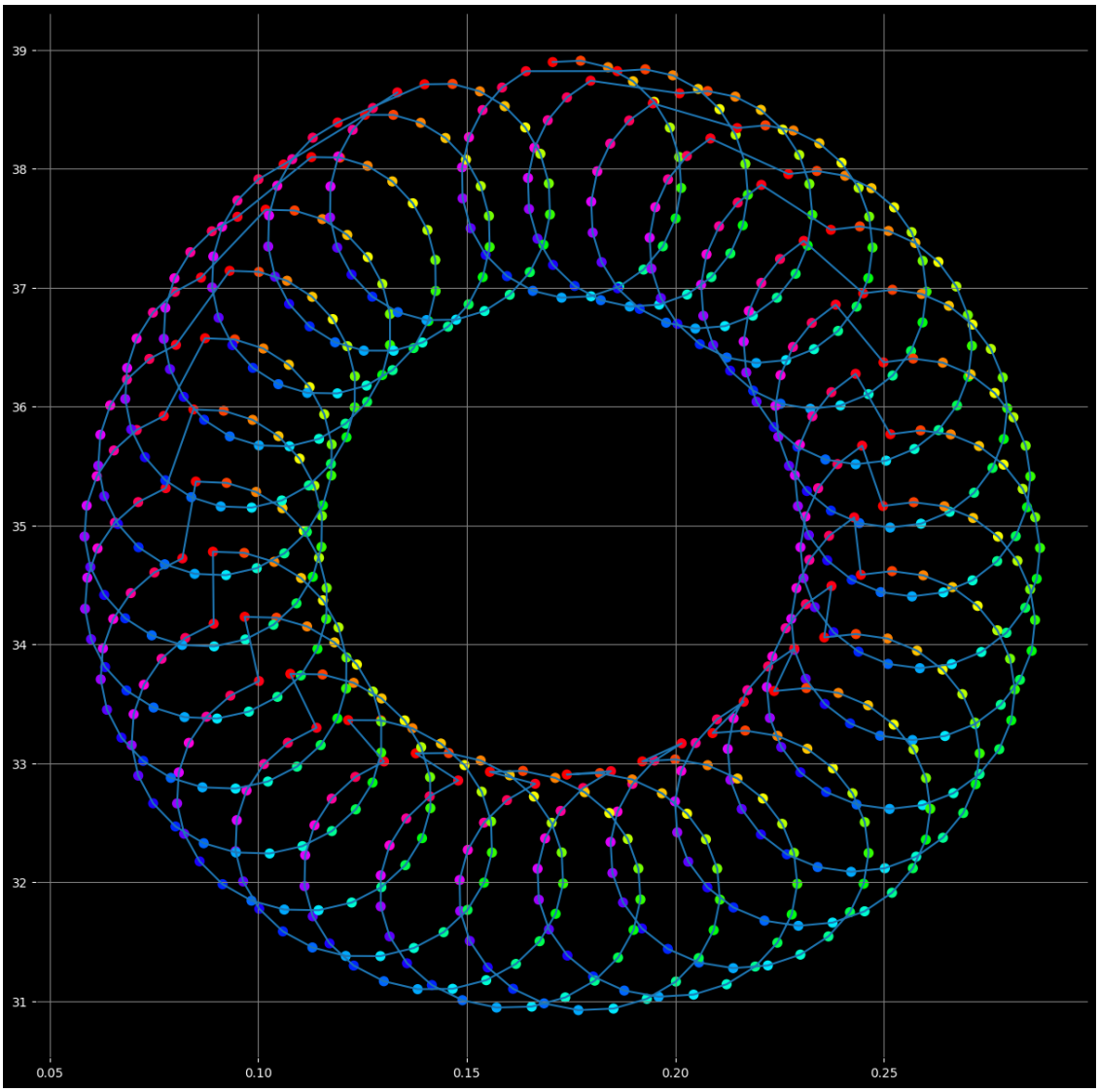
$$r_h = 1$$

[148]:
```
r_y=24
r_m=12
r_d=3
r_h=1
T_color=T_h
symbol_color=hour

# Initializing a dummy atlas for preallocation
y_hat_f1 = r_y*np.exp((2*np.pi*1j/T_y)*y)
m_hat_f1 = r_m*np.exp((2*np.pi*1j/T_m)*m)
d_hat_f1 = r_d*np.exp((2*np.pi*1j/T_d)*d)
h_hat_f1 = r_h*np.exp((2*np.pi*1j/T_h)*h)
atlas_filter_1 = y_hat_f1 + m_hat_f1 +d_hat_f1+h_hat_f1

init_atlas(20,20)
#plt.plot(atlas.real,atlas.imag)
color_atlas(atlas_filter_1,size,marker,symbol_color,cmap,T_color)

image_path = './img/filter_0.png'
plt.savefig(image_path)
```



B_h again, now not normal but orbital

[149]:
```
bad_atlas_f1=np.abs(atlas_filter_1)+1j*np.angle(atlas_filter_1)
plt.figure(figsize=(15,15))
#plt.scatter(bad_atlas.imag[0:24*30*1],bad_atlas.real[0:24*30*1])
plt.plot(bad_atlas_f1.imag[0:24*30*1],bad_atlas_f1.real[0:24*30*1])
color_atlas(bad_atlas_f1.imag[0:24*30*1]+1j*bad_atlas_f1.real[0:
  ↪24*30*1],size*50,marker,symbol_color[0:24*30*1],cmap,T_h)
plt.grid()
```



[150]:
```
2023%7
```

[150]: 0

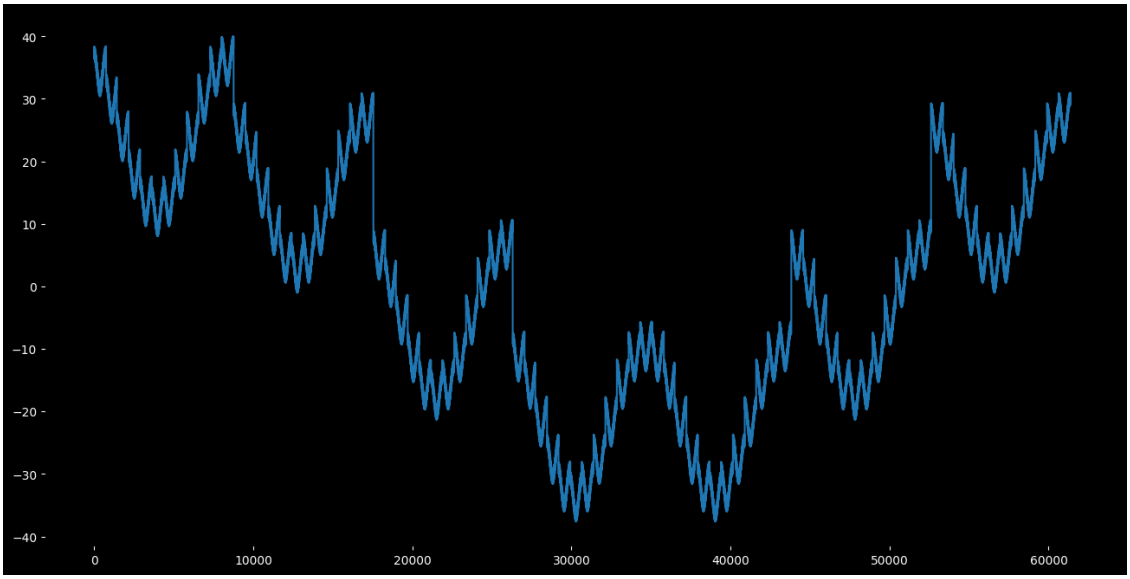
[151]:
```
17*17*7
```

[151]: 2023

9.4.1 Real and Imaginary parts after the filter

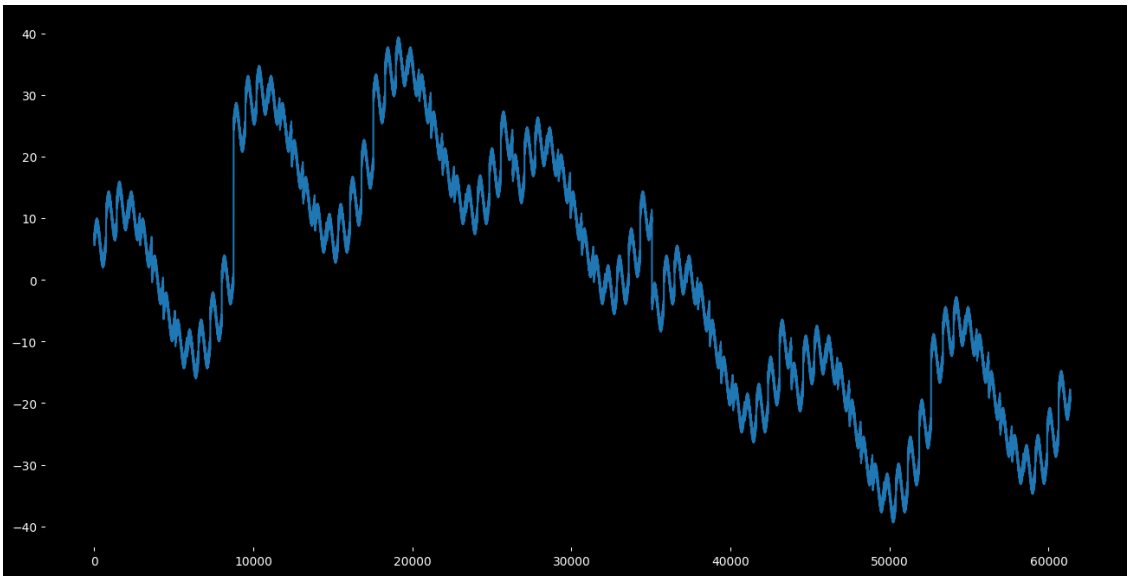
[152]:
```
init_atlas(16,8)
plt.plot(h,atlas_filter_1.real)
```

[152]: [<matplotlib.lines.Line2D at 0x7f0ddc0d3220>]



[153]:
```
init_atlas(16,8)
plt.plot(h,atlas_filter_1.imag)
```

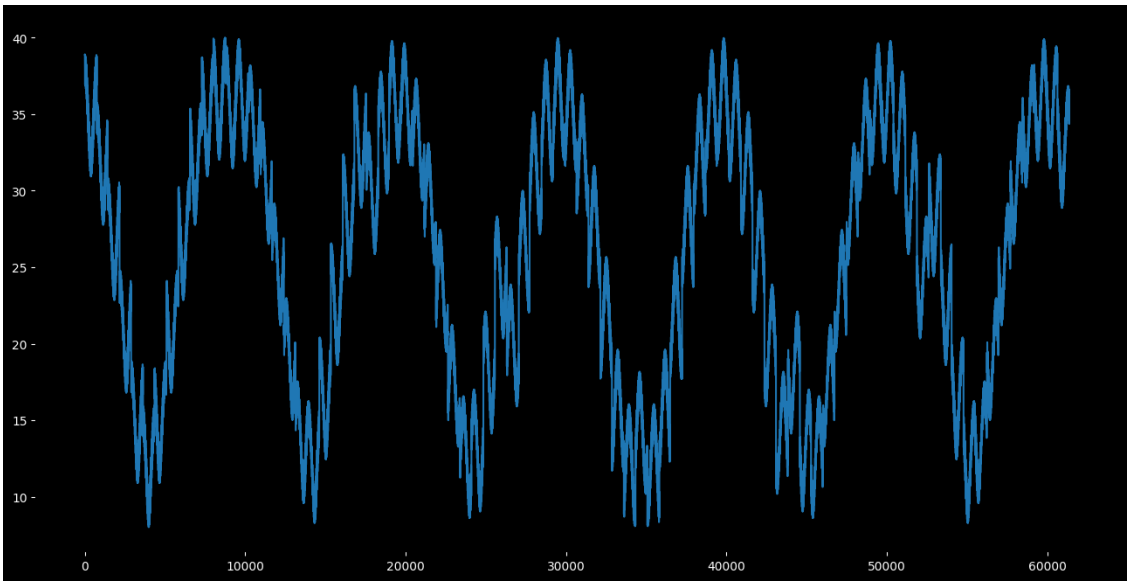
[153]: [<matplotlib.lines.Line2D at 0x7f0dd96252d0>]



9.4.2 Radius of the compass over time

[154]:
```
init_atlas(16,8)
plt.plot(h,np.abs(atlas_filter_1))
```

[154]: [<matplotlib.lines.Line2D at 0x7f0dd8ad6d70>]



9.5 Filter 2: Let's modify the radius of our compass to make sense of previous plot -we can try with factors of 2^{-x}

In previous examples r_y, r_m ,r_d , r_h where set to 1 in the first example and to another combination in the second. In the second plot we can now distinguish with some 'aliasing' the 24 hours in the day, going around the number of days in a month (careful here), and each month going around the 12 months in the year .

They where as individual clocks spining independently added together as vectos.

$$\hat{y} = r_y \Theta^{\omega_y y}$$

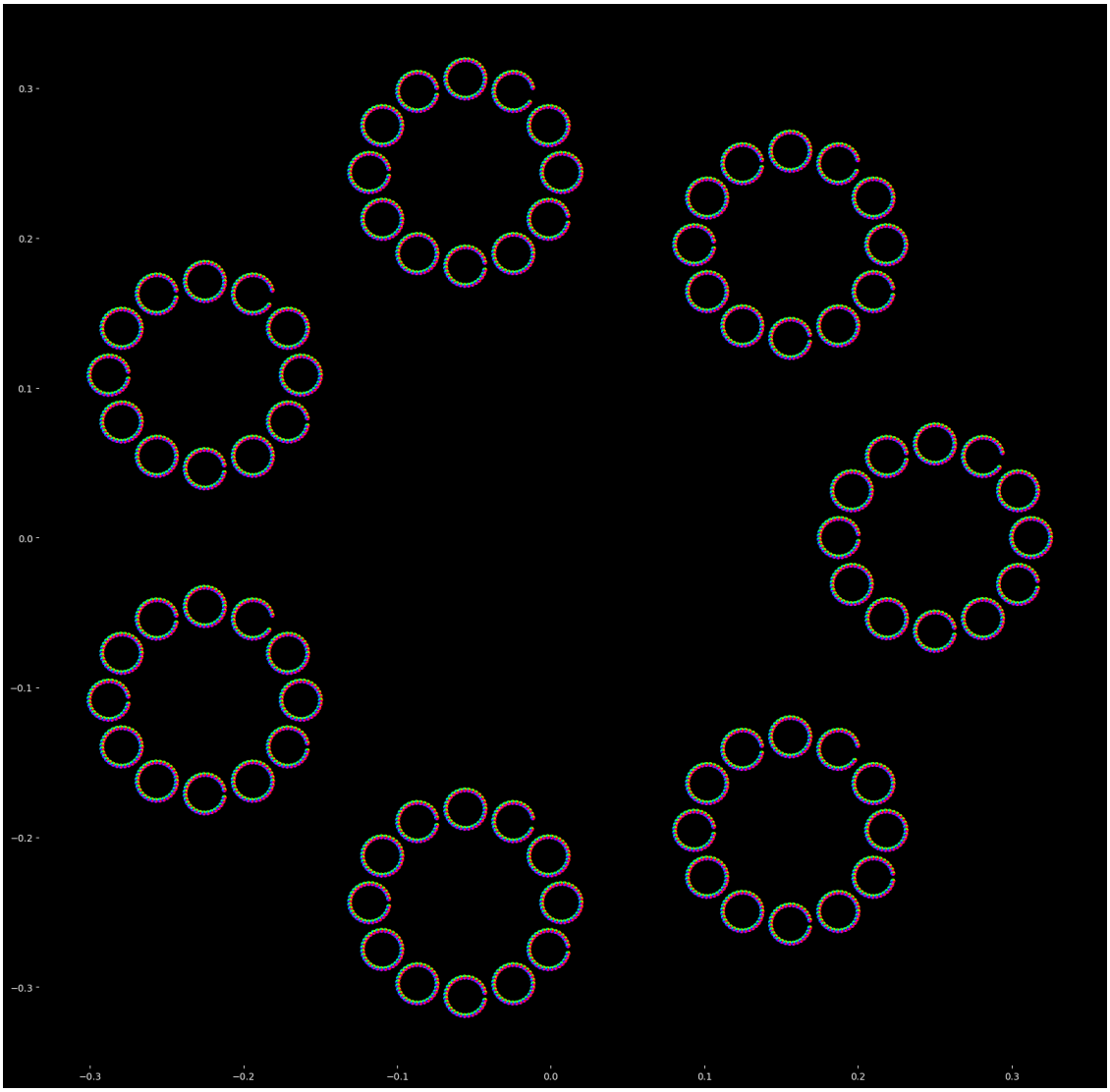
$$\hat{m} = r_m \Theta^{\omega_m m}$$

$$\hat{d} = r_d \Theta^{\omega_d d}$$

$$\hat{h} = r_h \Theta^{\omega_h h}$$

Now we will change this to assign them according to the scale of time they represent, we can try with factors of 2^{-x}

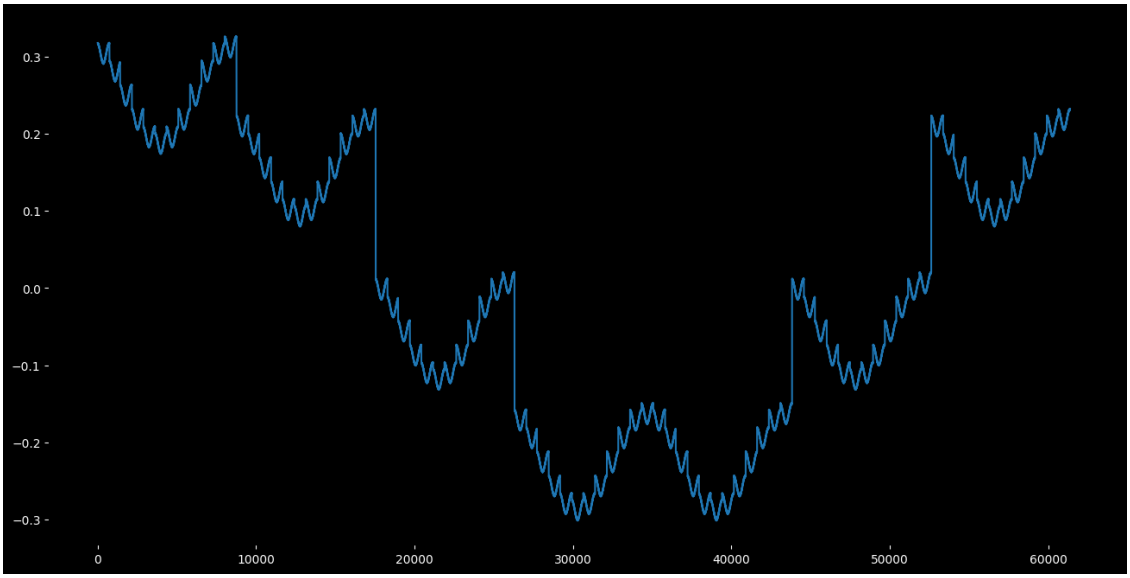
[155]:
```
T_h=24
T_d=31 # to be changed inside the loop in the function maxdays 28,29,30 or 31
T_m=12
T_y=7
r_y=1/2**2
r_m=1/2**4
r_d=1/2**(2*np.pi)
r_h=1/2**10


y_hat_f2 = r_y*np.exp((2*np.pi*1j/T_y)*y)
m_hat_f2 = r_m*np.exp((2*np.pi*1j/T_m)*m)
d_hat_f2 = r_d*np.exp((2*np.pi*1j/T_d)*d)
h_hat_f2 = r_h*np.exp((2*np.pi*1j/T_h)*h)
atlas_filter_2 = y_hat_f2 + m_hat_f2 +d_hat_f2+h_hat_f2

init_atlas(20,20)
color_atlas(atlas_filter_2,size,marker,symbol_color,cmap,T_color)

image_path = './img/filter_2.png'
plt.savefig(image_path)
```



9.5.1 Real and Imaginary parts after the second filter

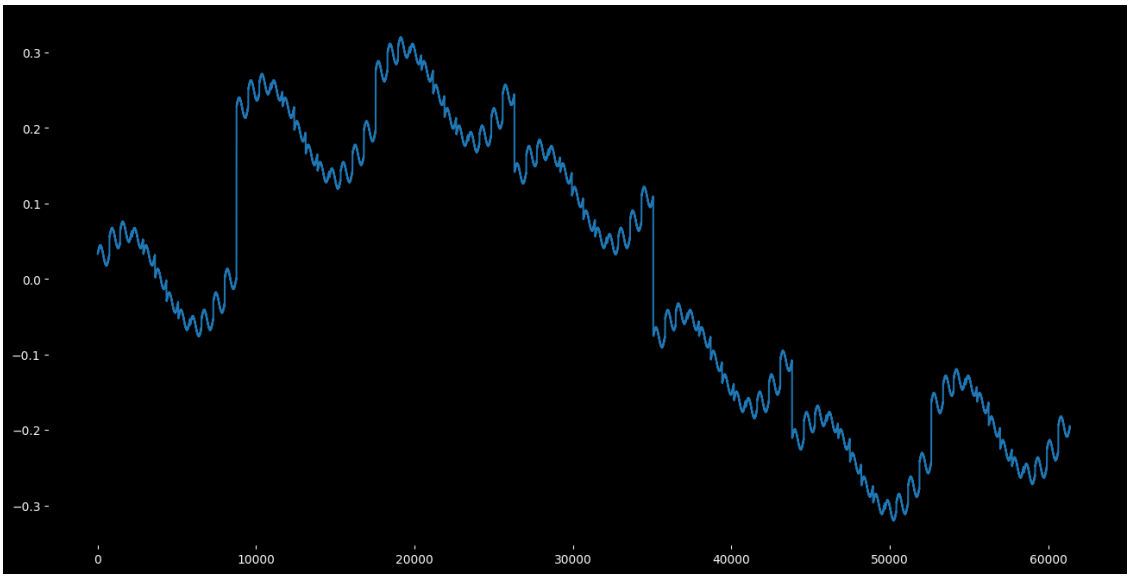
[156]:
```
init_atlas(16,8)
plt.plot(h,atlas_filter_2.real)
```

[156]: [<matplotlib.lines.Line2D at 0x7f0dd96468c0>]



[157]:
```
init_atlas(16,8)
plt.plot(h,atlas_filter_2.imag)
```

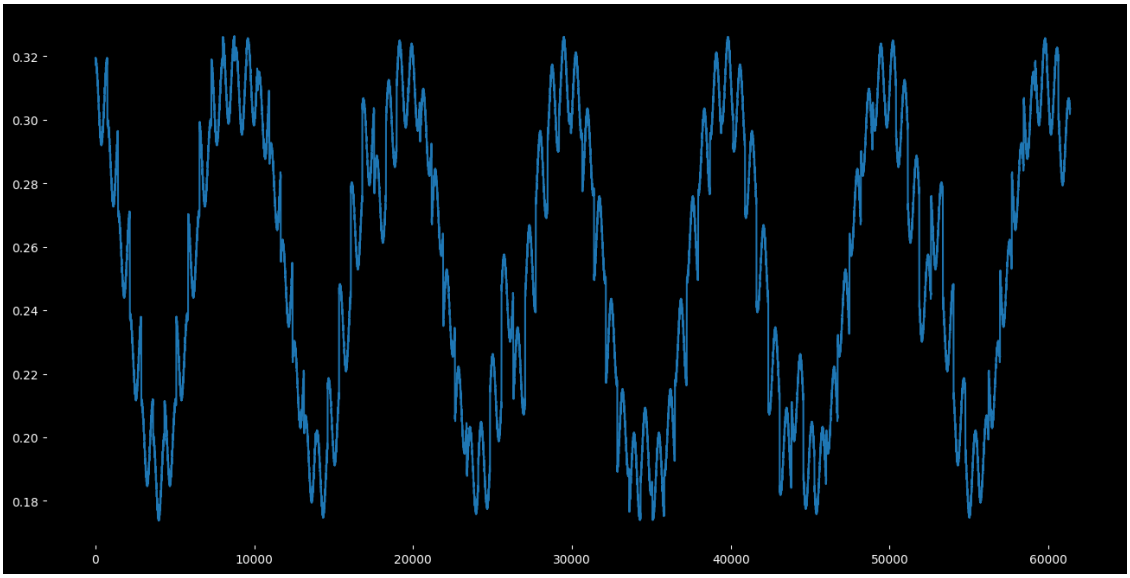
[157]: [<matplotlib.lines.Line2D at 0x7f0dde1ac250>]



9.5.2 The 'filtered' radius of the compass over time

[158]:
```
init_atlas(16,8)
plt.plot(h,np.abs(atlas_filter_2))
```

[158]: [<matplotlib.lines.Line2D at 0x7f0ddc0e6440>]

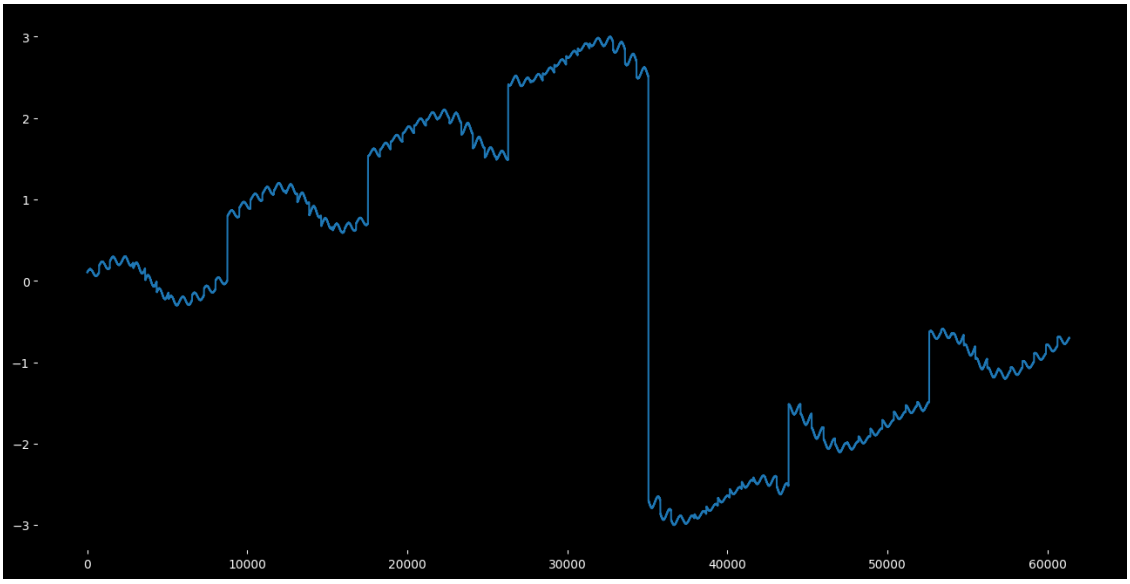


9.5.3 The 'filtered' angle of the compass over time

- The angles in this representation are from $-\pi$ to π , this is why you see the drop to negative values in the following plot

[159]:
```
init_atlas(16,8)
plt.plot(h,np.angle(atlas_filter_2))
```

[159]: [<matplotlib.lines.Line2D at 0x7f0dd9572b90>]

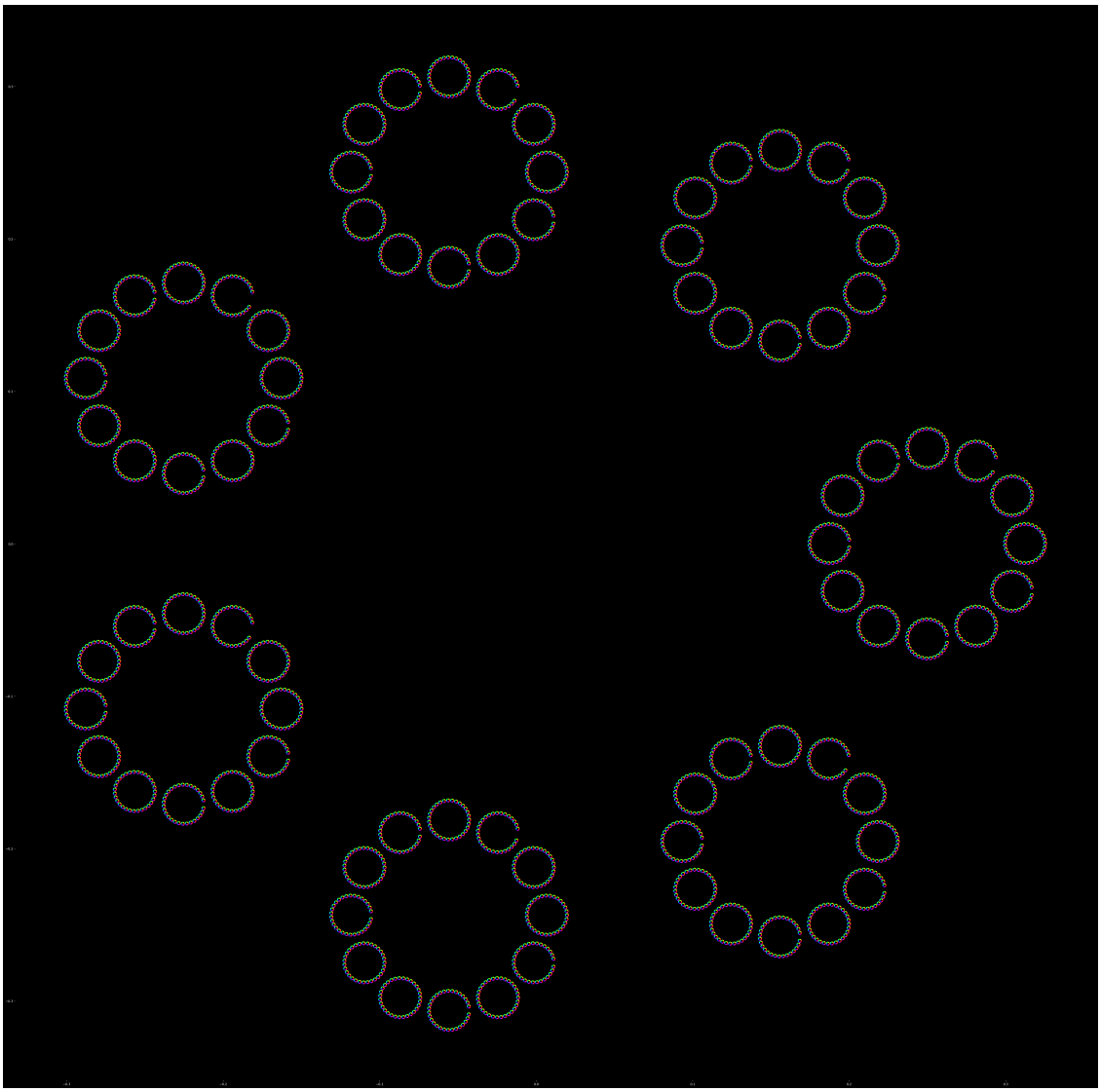


9.5.4 Hourly periodic mod 24 data for years 2023 to 2029

[160]:
```
year%7
```

[160]:
```
array([0, 0, 0, …, 6, 6, 6])
```

[161]:
```
init_atlas(60,60)
color_atlas(atlas_filter_2,size*4,marker,hour,cmap,T_color)
image_path = './img/3sparse_year_atlas_first_plot.png'
#plt.plot(atlas.real,atlas.imag)
plt.savefig(image_path)
```

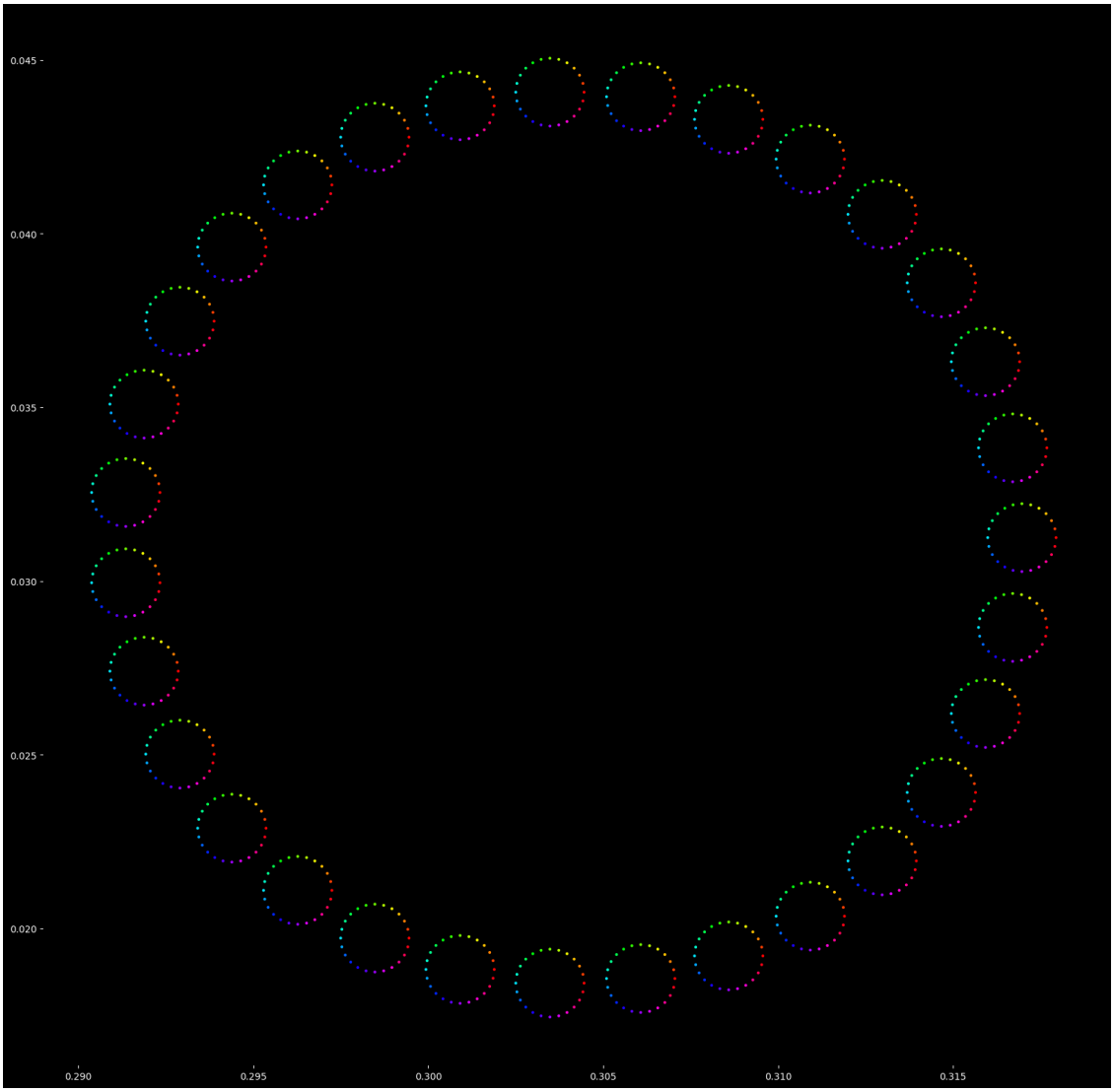


[162]:
```
z_h=atlas_filter_2/atlas
```

10 Let's see just one month

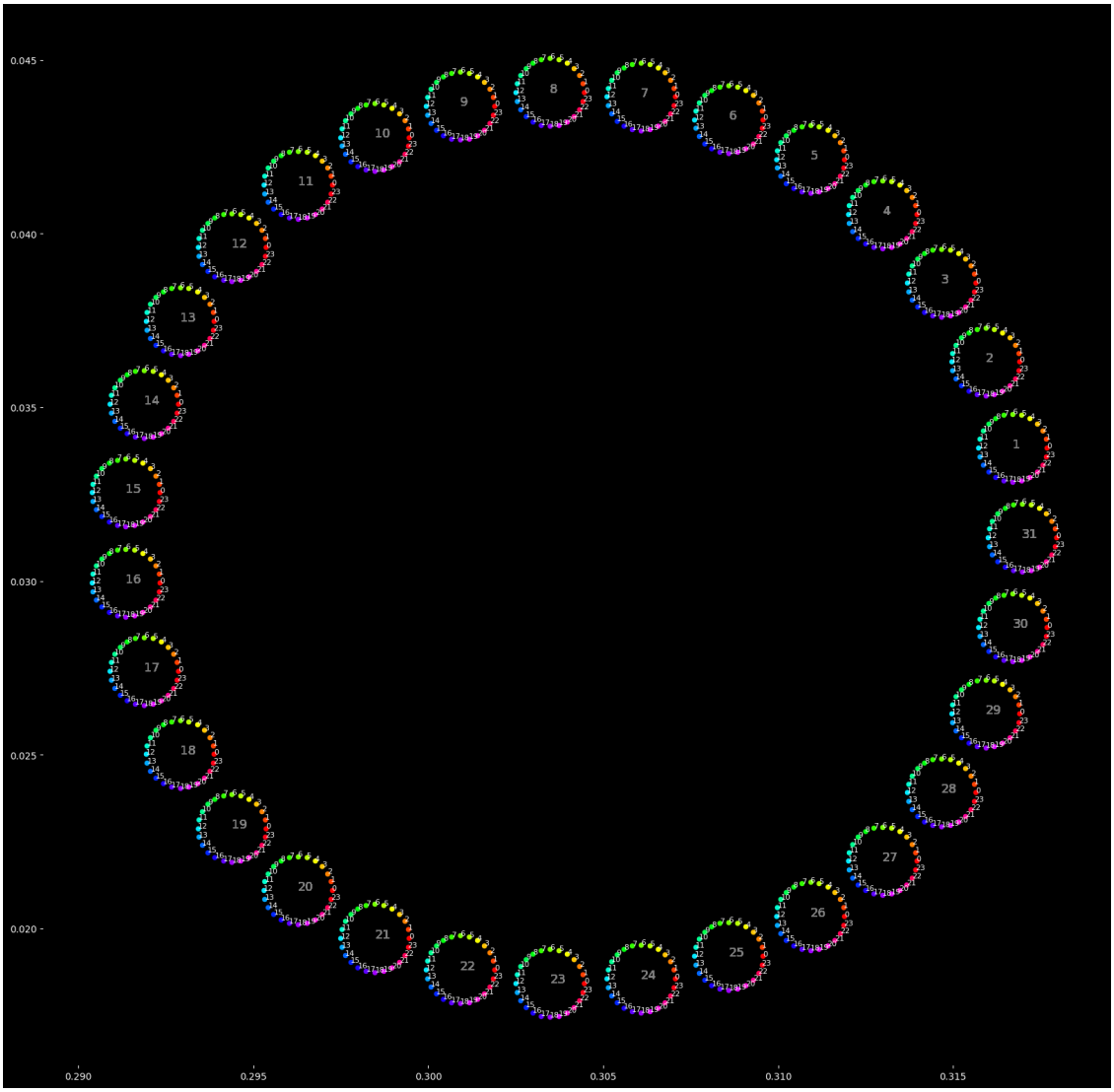
- Note the color here just represents the hour of the day mod 24 , but it could easily be a climate periodic signal or the relative shift in price of an asset of interest, etc.

[163]:
```
init_atlas(20,20)
color_atlas(atlas_filter_2[0:24*31],size*4,marker,hour[0:24*31],cmap,T_color)
image_path = './img/sparse_month_atlas_first_plot.png'
plt.savefig(image_path)
```



10.1 31 days in january numbered hours

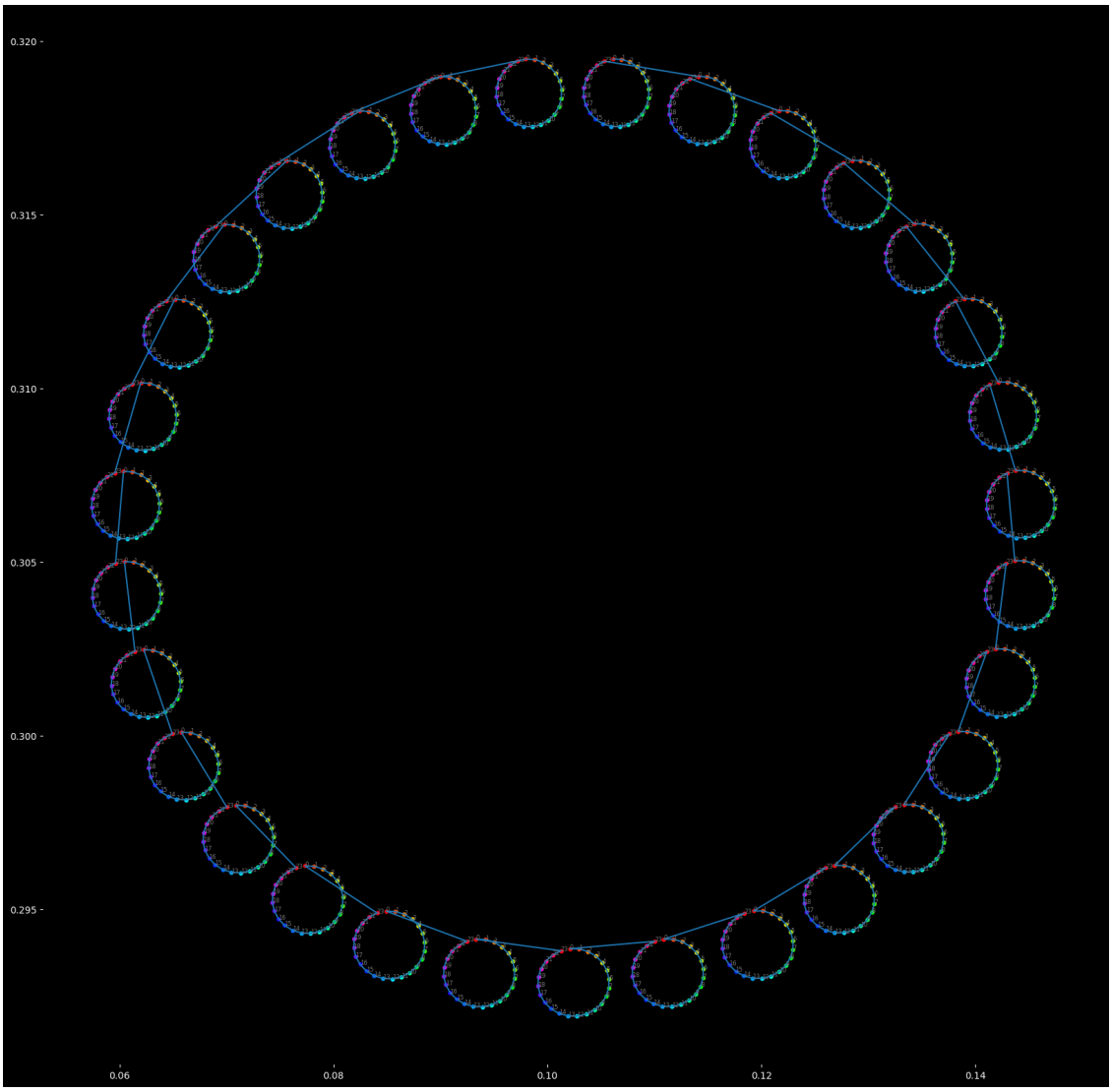
[164]:
```
init_atlas(20,20)
for n in h[0:24*31]:
    color_atlas(atlas_filter_2[n],size*20,marker,hour[n],cmap,T_h)
    plt.text(atlas_filter_2.real[n],atlas_filter_2.
 ↪imag[n],str(h[n]%T_h),size=8,color='white')
    plt.text(atlas_filter_2.real[n]-h_hat_f2.real[n],atlas_filter_2.
 ↪imag[n]-h_hat_f2.imag[n],str(d[n]),size=13,color='gray')
```



10.1.1 Bad atlas mixup of the filtered data???,Now the normal is the completely filtered orbital

[165]:
```
bad_atlas_f2=np.abs(atlas_filter_2)+1j*np.angle(atlas_filter_2)
init_atlas(20,20)
plt.plot(bad_atlas_f2.imag[0:24*31*1],bad_atlas_f2[0:24*31*1].real)
for n in h[0:24*31*1]:
    color_atlas(bad_atlas_f2.imag[n]+1j*bad_atlas_f2.
 ↪real[n],size*13,marker,hour[n],cmap,T_h)

    plt.text(bad_atlas_f2.imag[n],bad_atlas_f2.
 ↪real[n],str(h[n]%T_h),size=6,color='gray')
```

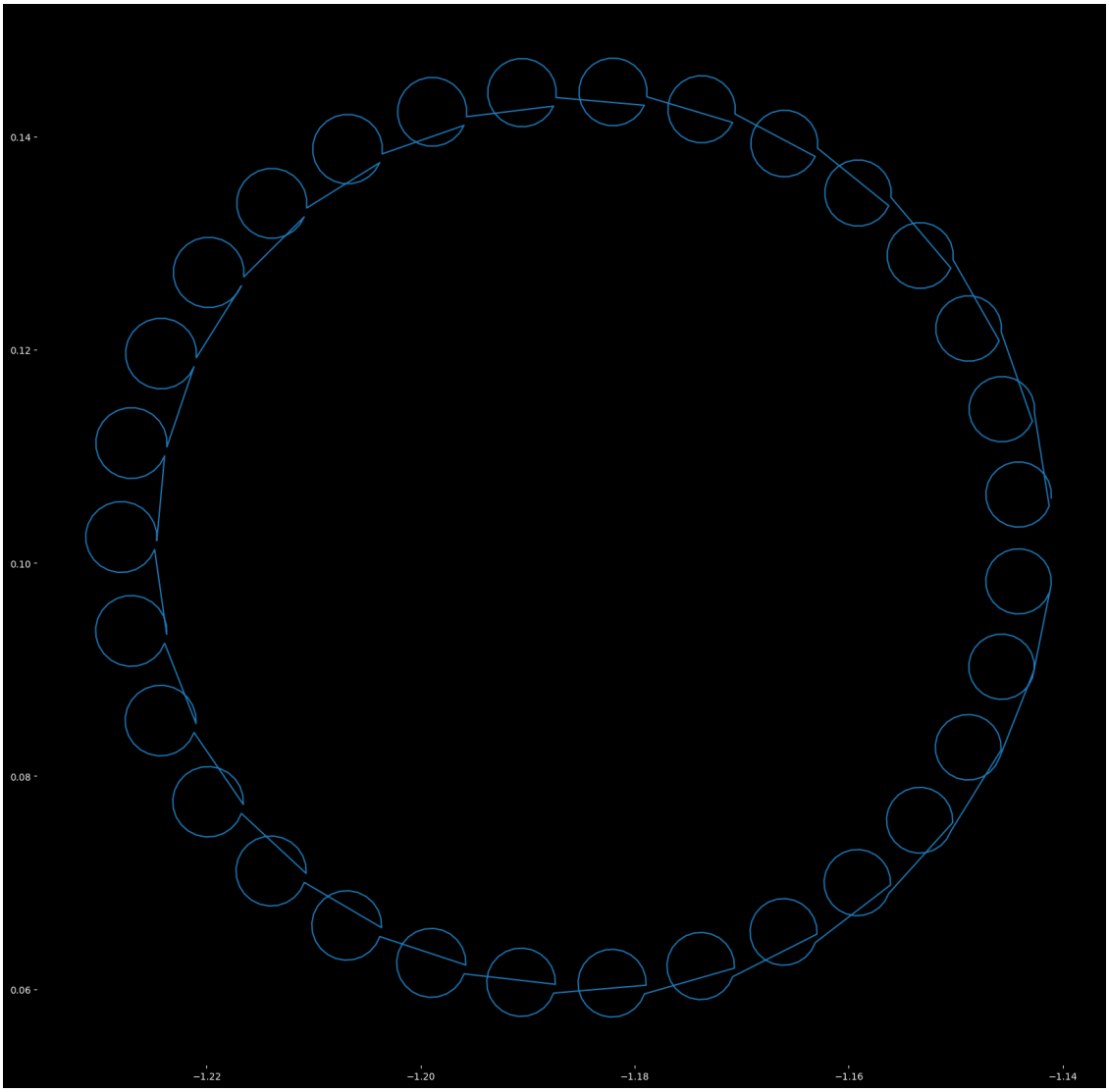
```
    #plt.text(bad_atlas_f2.imag[n]-h_hat_f2.real[n],bad_atlas_f2.
↪real[n]-h_hat_f2.imag[n],str(d[h]),size=18,color='black')
    #plt.text(atlas_filter_2.real[n]-h_hat_f2.real[n],atlas_filter_2.
↪imag[n]-h_hat_f2.imag[n],str(d[n]),size=13,color='gray')
```



[166]:
```
#bad_atlas_f2=np.abs(atlas_filter_2)+1j*np.angle(atlas_filter_2)
bad_atlas_f2=np.log(atlas_filter_2)

init_atlas(20,20)
plt.plot(bad_atlas_f2.real[0:24*31*1],bad_atlas_f2[0:24*31*1].imag)
# for n in h[0:24*31*1]:
```

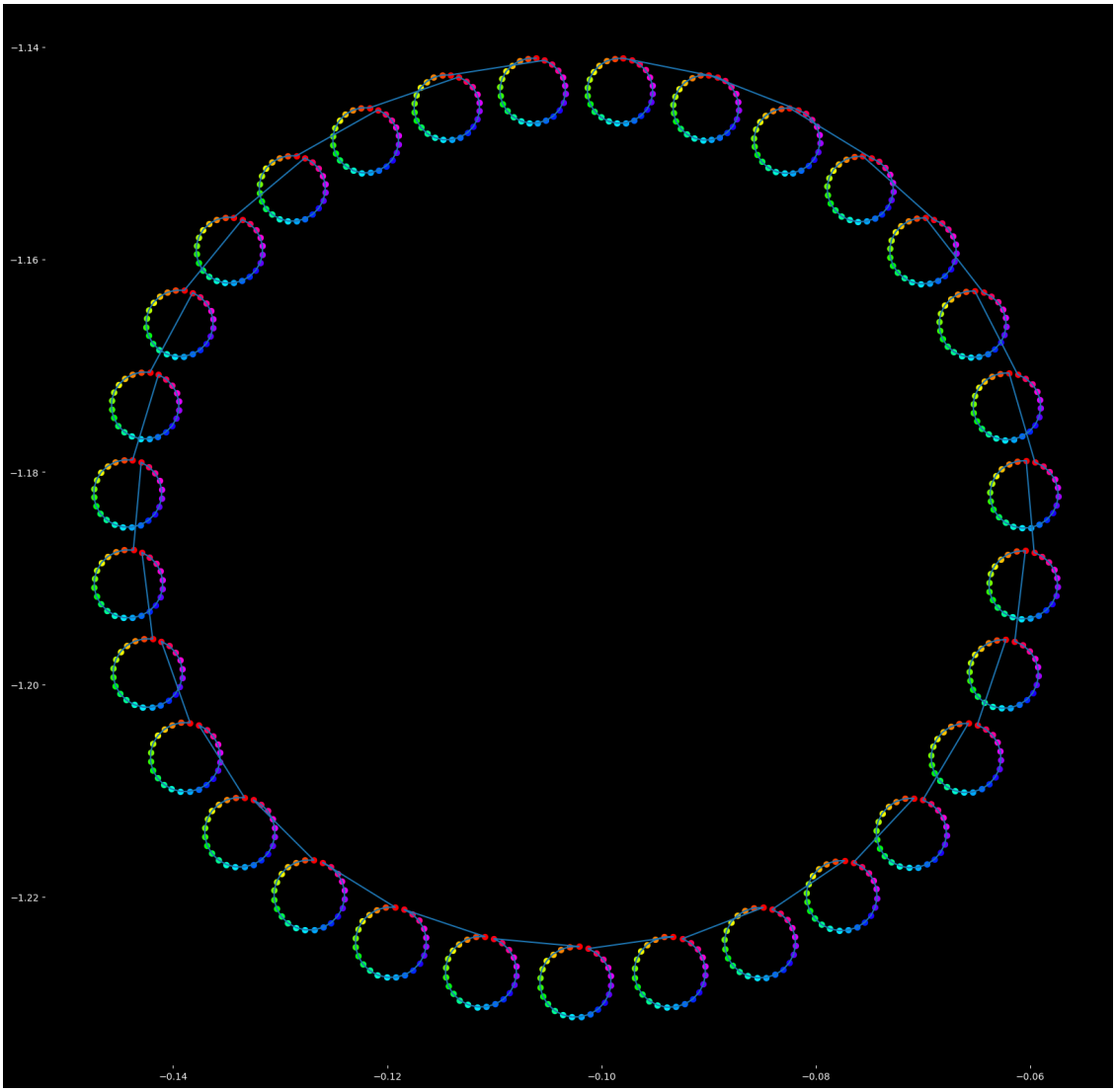
```
#     color_atlas(bad_atlas_f2.imag[n]+1j*bad_atlas_f2.
 ↪real[n],size*13,marker,hour[n],cmap,T_h)

#     plt.text(bad_atlas_f2.imag[n],bad_atlas_f2.
 ↪real[n],str(h[n]%T_h),size=6,color='gray')

#     #plt.text(bad_atlas_f2.imag[n]-h_hat_f2.real[n],bad_atlas_f2.
 ↪real[n]-h_hat_f2.imag[n],str(d[h]),size=18,color='black')
#     #plt.text(atlas_filter_2.real[n]-h_hat_f2.real[n],atlas_filter_2.
 ↪imag[n]-h_hat_f2.imag[n],str(d[n]),size=13,color='gray')
```

[166]: [<matplotlib.lines.Line2D at 0x7f0dd0f2a380>]



10.1.2 Using $log(z)$ again

[167]:
```
#bad_atlas_f2=np.abs(atlas_filter_2)+1j*np.angle(atlas_filter_2)
bad_atlas_f4=1j*np.log(atlas_filter_2)

init_atlas(20,20)
plt.plot(bad_atlas_f4.real[0:24*31*1],bad_atlas_f4[0:24*31*1].imag)
for n in h[0:24*31*1]:
    color_atlas(bad_atlas_f4.real[n]+1j*bad_atlas_f4.
 ↪imag[n],size*33,marker,hour[n],cmap,T_h)

    #plt.text(bad_atlas_f3.real[n],bad_atlas_f3.
 ↪imag[n],str(h[n]%T_h),size=6,color='gray')

    #plt.text(bad_atlas_f2.imag[n]-h_hat_f2.real[n],bad_atlas_f2.
 ↪real[n]-h_hat_f2.imag[n],str(d[h]),size=18,color='black')
    #plt.text(atlas_filter_2.real[n]-h_hat_f2.real[n],atlas_filter_2.
 ↪imag[n]-h_hat_f2.imag[n],str(d[n]),size=13,color='gray')
```

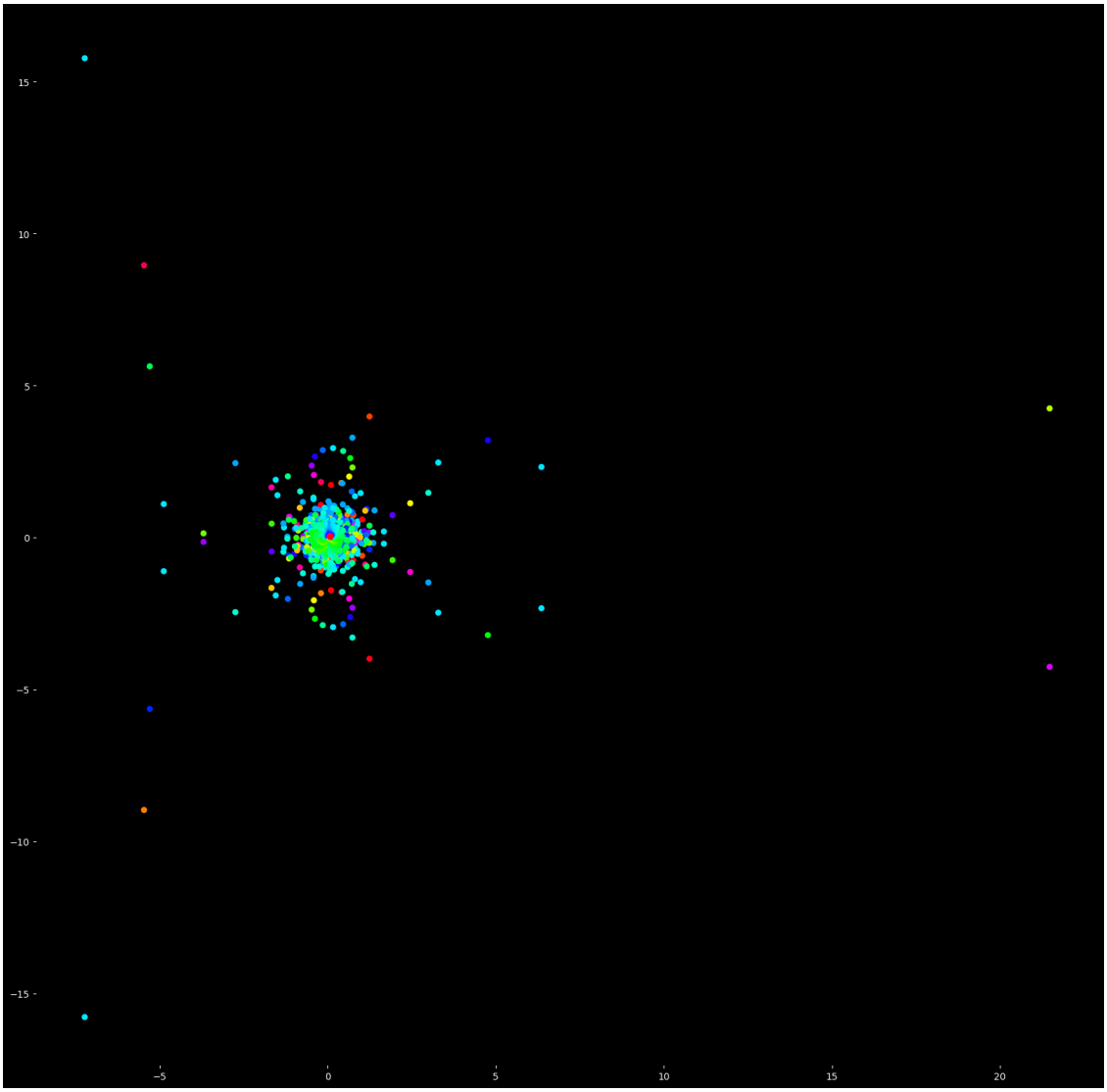


11 The filter we'd create if we knew the solution to the puzzle

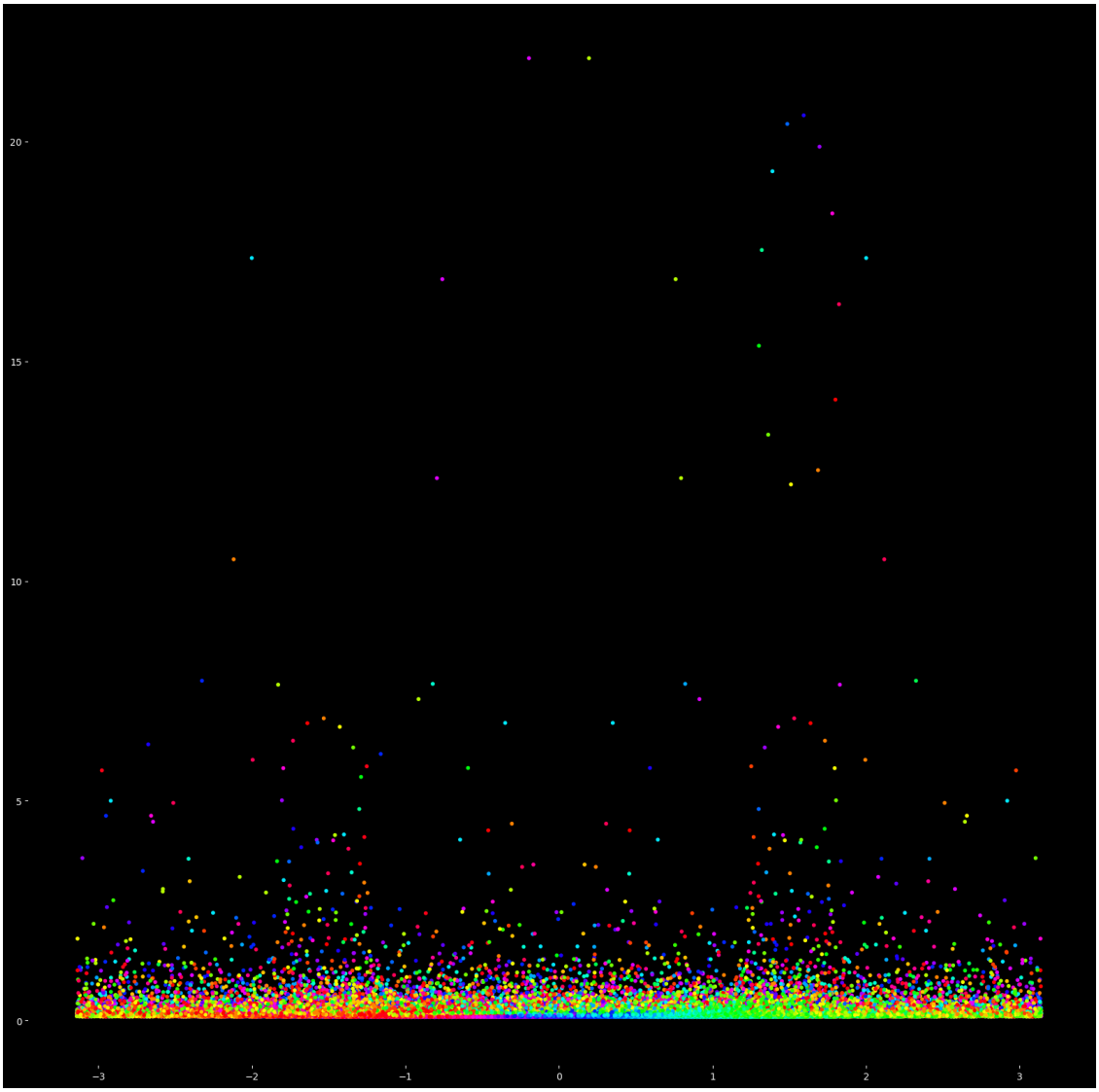
[168]:
```
z_h=atlas_filter_2/atlas
```

[ ]:
```

```

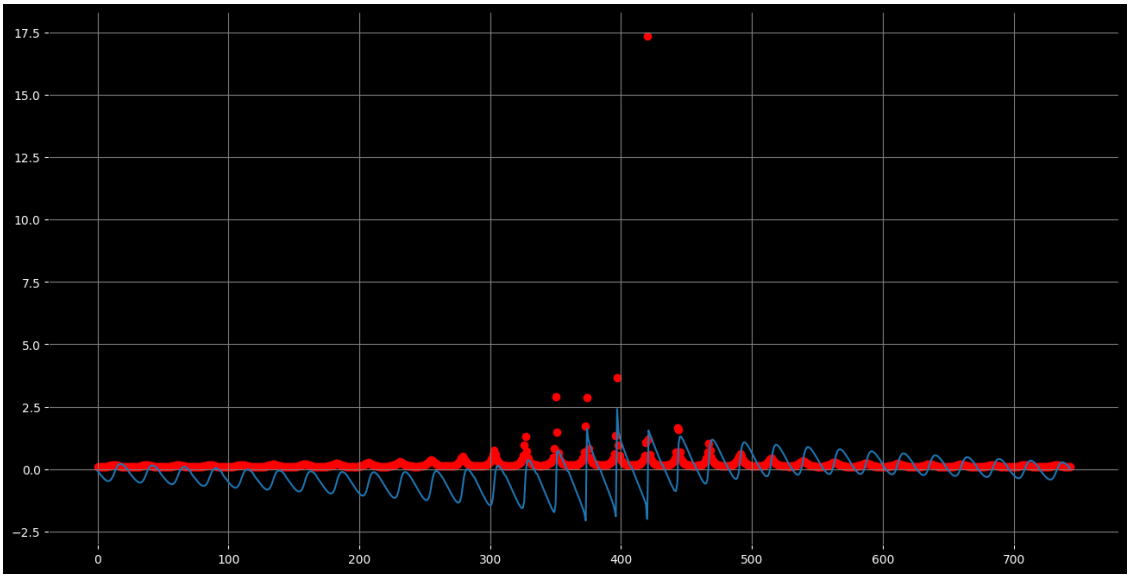
[169]:
```
init_atlas(20,20)
color_atlas(z_h[0:24*30*12],size*30,marker,hour[0:24*30*12],cmap,T_h)
```



[170]:
```
init_atlas(20,20)
color_atlas(np.angle(z_h[0:24*30*12*6])+1j*np.abs(z_h[0:
↪24*30*12*6]),size*10,marker,hour[0:24*30*12*6],cmap,T_h)
#plt.plot(np.angle(z_h[0:24*30*12*6]),np.abs(z_h[0:24*30*12*6]))
```



[171]:
```
init_atlas(16,8)

#plt.plot(h,np.abs(generic_filter))
plt.scatter(h[0:24*31]%(24*31),np.abs(z_h[0:24*31]), c='red')
plt.plot(h[0:24*31]%(24*31),np.angle(z_h[0:24*31]))
plt.grid()
```



11.1 If the filter doesn't work (generic one)

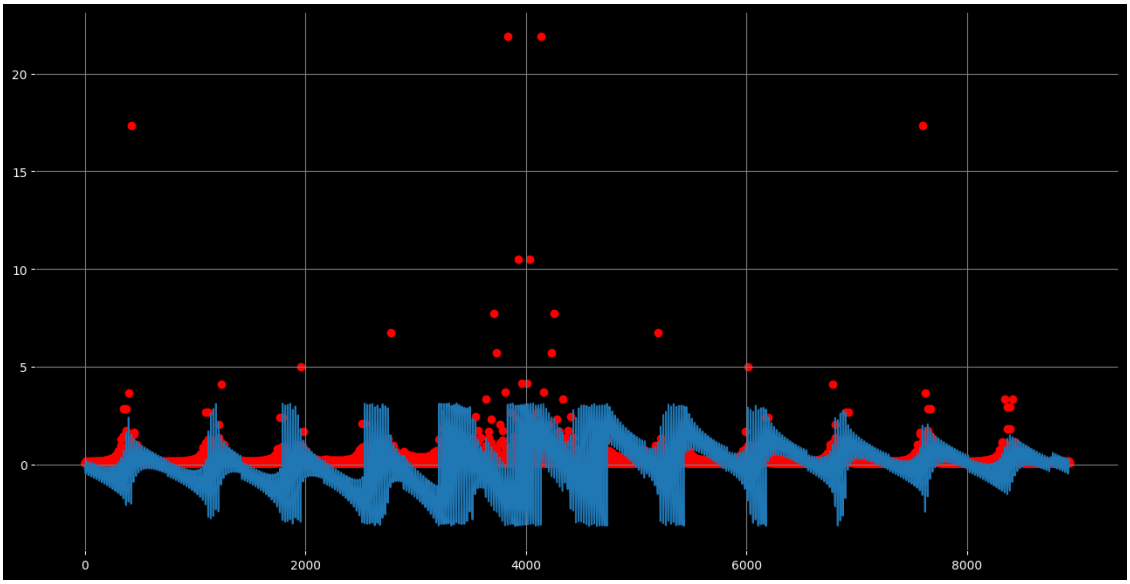
Here we test with a generic random filter and we see that there are no cycles in the filter against the data

[172]:
```
T_s=1
T_w=2 # to be changed inside the loop in the function maxdays 28,29,30 or 31
T_v=3
T_u=4
r_u=1/4
r_v=1/2
r_w=3/4
r_s=1
# u=y
# v=m
# w=d
# s=h
u=d
v=h
w=y
s=m



u_hat = r_u*np.exp((2*np.pi*1j/T_u)*u)
v_hat = r_v*np.exp((2*np.pi*1j/T_v)*v)
w_hat = r_w*np.exp((2*np.pi*1j/T_w)*w)
s_hat = r_s*np.exp((2*np.pi*1j/T_s)*s)
generic_atlas=u_hat+v_hat+w_hat+s_hat
```

```
generic_filter=generic_atlas/atlas
init_atlas(20,20)
color_atlas(generic_filter,size*30,marker,symbol_color,cmap,T_color)

image_path = './img/filter_3.png'
plt.savefig(image_path)
```

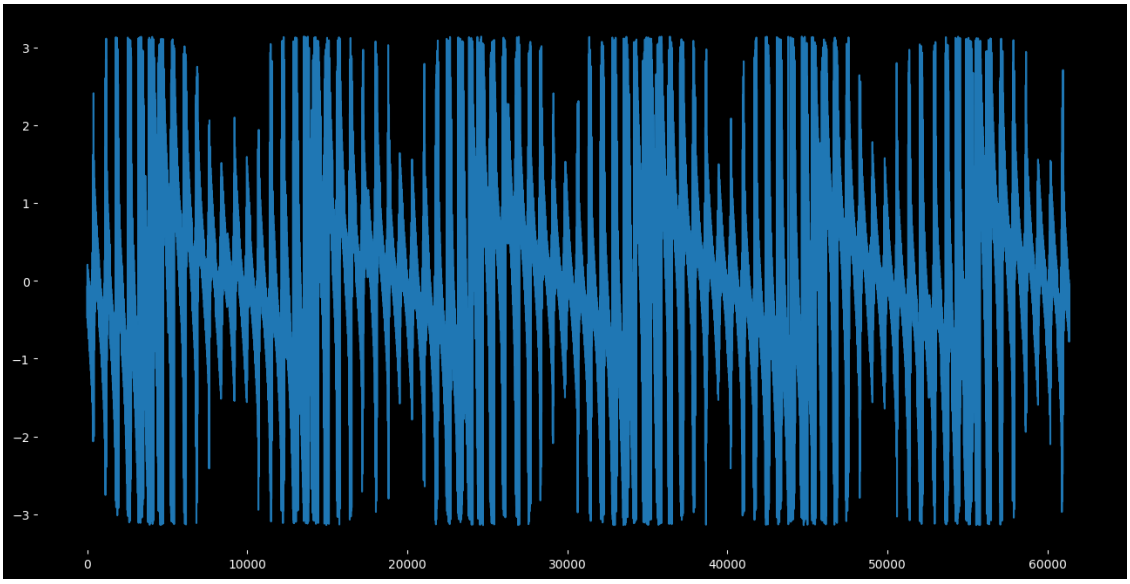
[ ]:

[ ]:
```
init_atlas(16,8)

#plt.plot(h,np.abs(generic_filter))
plt.scatter(h[0:24*31*12*1]%(60000),np.abs(z_h[0:24*31*12*1]), c='red')
plt.plot(h[0:24*31*12*1]%(60000),np.angle(z_h[0:24*31*12*1]))
plt.grid()
```



[ ]:
```
init_atlas(16,8)

#plt.plot(h,np.abs(generic_filter))
plt.plot(h,np.angle(z_h))
```

[ ]:
```
[<matplotlib.lines.Line2D at 0x7f0dd88ef9d0>]
```



[ ]:
```
# start = pd.Timestamp('2023-01-01 00:00:00')
# end = pd.Timestamp('2023-12-31 23:00:00')
# freq ='H'
# df  = create_random_climate_data(start,end,freq)
# year, month, day, hour, minute, second, temperature, humidity, pressure =␣
 ↪extract_climate_data(df)
# h = np.arange(len(df.index))
# h.size
```

[ ]:
```
#! jupyter nbconvert --no-input --to pdf year_atlas_example.ipynb
```

[ ]:
```
! jupyter nbconvert  --to pdf -o white_with_code_v0.pdf  compass_numbers.ipynb
```