clean_code

May 31, 2023

1 1 year atlas close look (let's look at 2022)

[89]:
```
import pandas as pd
import numpy as np
import calendar
import random
import matplotlib.pyplot as plt
from matplotlib.path import Path
from matplotlib import cm
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")
```

[131]:
```
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
from matplotlib.text import TextPath
import numpy as np

def get_atlas_video_text(atlas_in, symbol, fps, colormap, output_path,␣
 ↪variable_size, fixed_size, padding, sx, sy):
    markers, norm, colors, norm_seq, sizes = get_colors(symbol, colormap,␣
 ↪variable_size, fixed_size)

    fig, ax = plt.subplots(figsize=(sx, sy))
    scatter = ax.scatter([], [], color=[], s=[], edgecolors='black', alpha=0.5)
    texts = []

    frames = len(atlas_in)
    x_values = []
    y_values = []
    sizes_values = []
    colors_values = []
    markers_values = []
```

```python
    def update(frame):
        nonlocal texts

        # Accumulate coordinates, sizes, colors, and markers from all previous frames
        x_values.extend([atlas_in[frame].real])
        y_values.extend([atlas_in[frame].imag])
        sizes_values.extend([sizes[frame]])
        colors_values.extend([colors[frame]])
        markers_values.extend([symbol[frame]])

        # Update the scatter plot with accumulated coordinates, sizes, colors, and markers
        scatter.set_offsets(np.column_stack((x_values, y_values)))
        scatter.set_sizes(sizes_values)
        scatter.set_color(colors_values)
        scatter.set_edgecolors('black')
        scatter.set_alpha(0.5)

        # Determine marker based on the value of the symbol for all frames
        marker_paths = []
        for marker in markers_values:
            if marker >= 0:
                marker_path = Path.unit_circle()
            else:
                marker_path = Path.unit_regular_polygon(4)
            marker_paths.append(marker_path)

        scatter.set_paths(marker_paths)

        # Clear previous text annotations
        for text in texts:
            text.remove()
        texts = []

        # Create text annotations at symbol coordinates with the same color as the symbol
        for x, y, marker, color in zip(x_values, y_values, markers_values, colors_values):
            if marker <= -1:
                text = ax.text(x, y, str(symbol), fontsize=8, color=color, ha='center', va='center')
                texts.append(text)

        # Adjust the axis limits based on the data
        ax.set_xlim(np.min(x_values) - padding, np.max(x_values) + padding)
        ax.set_ylim(np.min(y_values) - padding, np.max(y_values) + padding)
```

```
    animation = FuncAnimation(fig, update, frames=frames, blit=False)

    # Specify the writer (FFMpegWriter) and the output filename
    writer = FFMpegWriter(fps=fps)  # Adjust the frames per second (fps) as␣
 ↪needed
    animation.save(output_path, writer=writer)
```

[141]:

```
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
from matplotlib.text import TextPath
import numpy as np

def get_atlas_video_text(atlas_in, symbol, fps, colormap, output_path,␣
 ↪variable_size, fixed_size, padding, sx, sy):
    markers, norm, colors, norm_seq, sizes = get_colors(symbol, colormap,␣
 ↪variable_size, fixed_size)

    fig, ax = plt.subplots(figsize=(sx, sy))
    scatter = ax.scatter([], [], color=[], s=[], edgecolors='black', alpha=0.5)
    texts = []

    frames = len(atlas_in)
    x_values = []
    y_values = []
    sizes_values = []
    colors_values = []
    markers_values = []

    def update(frame):
        nonlocal texts

        # Accumulate coordinates, sizes, colors, and markers from all previous␣
 ↪frames
        x_values.extend([atlas_in[frame].real])
        y_values.extend([atlas_in[frame].imag])
        sizes_values.extend([sizes[frame]])
        colors_values.extend([colors[frame]])
        markers_values.extend([symbol[frame]])

        # Update the scatter plot with accumulated coordinates, sizes, colors,␣
 ↪and markers
        scatter.set_offsets(np.column_stack((x_values, y_values)))
        scatter.set_sizes(sizes_values)
        scatter.set_color(colors_values)
        scatter.set_edgecolors('black')
        scatter.set_alpha(0.5)
```

```
        # Determine marker based on the value of the symbol for all frames
        marker_paths = []
        for marker in markers_values:
            if marker >= 0:
                marker_path = Path.unit_circle()
            else:
                marker_path = Path.unit_regular_polygon(4)
            marker_paths.append(marker_path)

        scatter.set_paths(marker_paths)

        # Clear previous text annotations
        for text in texts:
            text.remove()
        texts = []

        # Create text annotations at symbol coordinates with the same color as␣
↪the symbol
        for x, y, marker, color,n in zip(x_values, y_values, markers_values,␣
↪colors_values,np.arange(atlas_in.size)):
            if marker >= 0:
                text = ax.text(x, y, str(symbol[n]), fontsize=7, color=color,␣
↪ha='center', va='center')
                texts.append(text)

        # Adjust the axis limits based on the data
        ax.set_xlim(np.min(x_values) - padding, np.max(x_values) + padding)
        ax.set_ylim(np.min(y_values) - padding, np.max(y_values) + padding)

    animation = FuncAnimation(fig, update, frames=frames, blit=False)

    # Specify the writer (FFMpegWriter) and the output filename
    writer = FFMpegWriter(fps=fps)  # Adjust the frames per second (fps) as␣
↪needed
    animation.save(output_path, writer=writer)
```

[91]:
```
# primes_list_path='./data/primes.csv'
# primes_list=pd.read_csv(primes_list_path)
def isPrime(n):
    # Check if n is less than 2
    if n < 2:
        return False

    # Check if n is divisible by any integer between 2 and the square root of n
    for i in range(2, int(n ** 0.5) + 1):
        if n % i == 0:
```

```
            return False

    # If none of the above conditions are met, n is prime
    return True
```

[92]:
```
isPrime(3)
```

[92]: True

[93]:
```
def␣
↪get_eigen_atlas_2D(input_symbol,sym_src,start_index,batch_size,M,To,k_o,r_o,t_o,frac,delta_
↪
#Variables preallocation
    symbol=np.zeros(batch_size,dtype=int)
    is_n_prime=np.zeros(batch_size,dtype=int)
    is_symbol_prime=np.zeros(batch_size,dtype=int)
    is_n_fibo=np.zeros(batch_size,dtype=int)
    is_symbol_fibo=np.zeros(batch_size,dtype=int)
    r_time_v   =np.zeros(batch_size,dtype=float)
    zeta_symbol     =np.zeros(batch_size,dtype=complex)
    z_log_scat =np.zeros(batch_size,dtype=complex)
    psi_v     =np.zeros(batch_size,dtype=complex)
    cplx_index =np.zeros(batch_size,dtype=complex)
    psi_v_z=np.zeros(batch_size,dtype=complex)
    z_time_norm_phase= np.zeros(batch_size,dtype=complex)
    z_carrier_information=np.zeros(batch_size,dtype=float)
    psi_v_information=np.zeros(batch_size,dtype=float)
    z_time_phase = np.zeros(batch_size,dtype=complex)
    z_time_norm  = np.zeros(batch_size,dtype=complex)
    z_time=np.zeros(batch_size,dtype=complex)
    z_carrier_alpha=np.zeros(batch_size,dtype=float)
    z_carrier_beta=np.zeros(batch_size,dtype=float)
    z_carrier=np.zeros(batch_size,dtype=complex)

    index       = np.arange(start_index,batch_size,dtype=int)
    nmod = index%To
    smod = input_symbol%M
    wzero_time = 2 * np.pi * (1/To)
    wzero_symbol=2 * np.pi * (1/M)

    for n in index:

        # Index bending
        if nmod[n]==0:
            r_time_v[n]=r_time_v[n-1]+delta_0
        else:
            r_time_v[n]=r_time_v[n-1]
```

```
    cplx_index[n]=r_time_v[n]*np.exp(1j * wzero_time * n )


    z_time_phase[n] = ((n+1)/r_o) * np.exp(1j * ((r_o)/r_phase) * (n+1) )
    z_time_norm[n]  = (n+1) * r_norm
    z_time_norm_phase[n]  = (n+1) * r_norm**(((r_o)/r_phase) * (n+1))
    z_time[n]=z_time_norm[n]/z_time_phase[n]
    z_carrier_alpha[n]=(r_o+t_o-(frac/r_o)*n)*z_time[n].real
    z_carrier_beta[n]=(r_o+t_o-(frac*root_norm)*n)*z_time[n].imag
    z_carrier[n]=z_carrier_alpha[n]+1j*z_carrier_beta[n]

    # Symbol bending

    if sym_src=='nmod':
        symbol[n]       = nmod[n] # symbols follow events index
    elif sym_src=='input':
        symbol[n]      = input_symbol[n] # input symbol (curated␣
↪representation of discretized reality )
    elif sym_src=='rand':
        symbol[n]      = random.randint(0, M) # symbols mod M from RNG
    elif sym_src=='smod':
        symbol[n]       = smod[n] # symbols mod M



    zeta_symbol[n]    = np.exp(1j * wzero_symbol * symbol[n])


    #Some possible marriages between time and symbols
    psi_v[n] =cplx_index[n] + f_psi*zeta_symbol[n]
    psi_v_z[n] = z_carrier[n] + f_psi*zeta_symbol[n]
    z_log_scat[n]= f_log_scat*np.log2(n+2)*zeta_symbol[n]
    z_carrier_information=np.log2(np.abs(z_carrier[n]))


    # Index moments
    if isPrime(n)==1:
        is_n_prime[n]=1
    else:
        is_n_prime[n]=0



    # Symbol moments
    if isPrime(symbol[n])==1:
        is_symbol_prime[n]=1
```

```
        else:
            is_symbol_prime[n]=0




    df = {'index':index,'nmod':nmod,'symbol':symbol,'smod':smod,'cplx_index':
 ↪cplx_index,'z_time_phase':z_time_phase,'z_time_norm':z_time_norm,'z_time':
 ↪z_time,'z_carrier':z_carrier,'zeta_symbol':zeta_symbol,'psi_v':
 ↪psi_v,'psi_v_z':psi_v_z,'z_log_scat':z_log_scat,'z_carrier_information':
 ↪z_carrier_information,'z_time_norm_phase':
 ↪z_time_norm_phase,'is_symbol_prime':is_symbol_prime,'is_n_prime':
 ↪is_n_prime,'is_symbol_fibo':is_symbol_fibo,'is_n_fibo':is_n_fibo}
    df =pd.DataFrame(df)
    df = df.set_index('index')
    return df
```

[94]:
```
def␣
 ↪plot_eigen_atlas(eigen_atlas,sx,sy,start,end,allow_scatter,allow_text,allow_anim_png,mode,s
 ↪
    x=eigen_atlas.z_carrier.values.real
    y=eigen_atlas.z_carrier.values.imag
    symbol=eigen_atlas.symbol.values
    smod=eigen_atlas.smod.values
    isprime=eigen_atlas.is_n_prime.values
    ff=3
    aa=0.1
    if mode=='3D':
        z=np.log(np.abs(x+1j*y))
        plt.style.use('dark_background')
        plt.rcParams.update({
        "lines.color": "black",
        "patch.edgecolor": "black",
        "text.color": "black",
        "axes.facecolor": "black",
        "axes.edgecolor": "black",
        "axes.labelcolor": "black",
        "xtick.color": "black",
        "ytick.color": "black",
        "grid.color": "black",
        "figure.facecolor": "black",
        "figure.edgecolor": "black",
        "savefig.facecolor": "black",
        "savefig.edgecolor": "black"})
        fig = plt.figure(figsize=(sx,sy))
        ax = fig.add_subplot(111, projection='3d')
```

```
        ax.grid(False)
        ax.w_xaxis.pane.fill = False
        ax.w_yaxis.pane.fill = False
        ax.w_zaxis.pane.fill = False

    elif mode=='2D':
        a=1111



    for n in np.arange(start,end):


            if allow_scatter:

                if isprime[n]:

                    ax.
↪scatter(x[n],y[n],-z[n],color=symbol_color(symbol[n-d_0]),s=sz_2*ff*3,marker=L_0,alpha=0.
↪8)
                    ax.
↪scatter(x[n],-y[n],-z[n],color=symbol_color(symbol[n-d_0]),s=sz_4*ff,marker=L_3,alpha=0.
↪2)
                    ax.
↪scatter(-x[n],y[n],-z[n],color=symbol_color(symbol[n-d_0]),s=sz_4*ff,marker=L_3,alpha=0.
↪2)
                    ax.
↪scatter(-x[n],-y[n],-z[n],color=symbol_color(symbol[n-d_0]),s=sz_4*ff,marker=L_3,alpha=0.
↪2)
                    ax.
↪scatter(y[n],x[n],-z[n],color=symbol_color(symbol[n-d_0]),s=sz_4*ff,marker=L_1,alpha=0.
↪3)
                    ax.
↪scatter(y[n],-x[n],-z[n],color=symbol_color(symbol[n-d_0]),s=sz_4*ff,marker=L_3,alpha=0.
↪3)
                    ax.
↪scatter(-y[n],x[n],-z[n],color=symbol_color(symbol[n-d_0]),s=sz_4*ff,marker=L_3,alpha=0.
↪3)
                    ax.
↪scatter(-y[n],-x[n],-z[n],color=symbol_color(symbol[n-d_0]),s=sz_4*ff,marker=L_3,alpha=0.
↪3)

                    ax.text(x[n]+aa,y[n]+aa,-z[n]+aa,str(n),␣
↪color=symbol_color(symbol[n-0]), size=text_size[n])
```

```
                    ax.
↪scatter(x[n],y[n],z[n],color=symbol_color(symbol[n-d_0]),s=sz_4*ff*6,marker=L_0,alpha=0.
↪8)
                    ax.
↪scatter(x[n],-y[n],z[n],color=symbol_color(symbol[n-d_0]),s=sz_4*ff,marker=L_3,alpha=0.
↪2)
                    ax.
↪scatter(-x[n],y[n],z[n],color=symbol_color(symbol[n-d_0]),s=sz_4*ff,marker=L_3,alpha=0.
↪2)
                    ax.
↪scatter(-x[n],-y[n],z[n],color=symbol_color(symbol[n-d_0]),s=sz_4*ff,marker=L_3,alpha=0.
↪2)
                    ax.
↪scatter(y[n],x[n],z[n],color=symbol_color(symbol[n-d_0]),s=sz_4*ff,marker=L_1,alpha=0.
↪3)
                    ax.
↪scatter(y[n],-x[n],z[n],color=symbol_color(symbol[n-d_0]),s=sz_4*ff,marker=L_3,alpha=0.
↪3)
                    ax.
↪scatter(-y[n],x[n],z[n],color=symbol_color(symbol[n-d_0]),s=sz_4*ff,marker=L_3,alpha=0.
↪3)
                    ax.
↪scatter(-y[n],-x[n],z[n],color=symbol_color(symbol[n-d_0]),s=sz_4*ff,marker=L_3,alpha=0.
↪3)
                    if allow_text:
                        ax.text(x[n]+aa,y[n]+aa,z[n]+aa,str(n), color='white',␣
↪size=text_size[n])

                    #
                    # ax.
↪scatter(x[n],z[n],y[n],color='aqua',s=sz_1*ff*2,marker=L_0)
                    # ax.
↪scatter(x[n],z[n],-y[n],color='brown',s=sz_2*ff,marker=L_1,alpha=0.2)
                    # ax.
↪scatter(x[n],-z[n],y[n],color='aqua',s=sz_3*ff,marker=L_2,alpha=0.2)
                    # ax.
↪scatter(x[n],-z[n],-y[n],color='brown',s=sz_4*ff,marker=L_3,alpha=0.2)
                    # ax.
↪scatter(-x[n],z[n],y[n],color='aqua',s=sz_4*ff,marker=L_3,alpha=0.2)
                    # ax.
↪scatter(-x[n],z[n],-y[n],color='brown',s=sz_4*ff,marker=L_3,alpha=0.2)
                    # ax.
↪scatter(-x[n],-z[n],y[n],color='aqua',s=sz_4*ff,marker=L_3,alpha=0.2)
                    # ax.
↪scatter(-x[n],-z[n],-y[n],color='brown',s=sz_4*ff,marker=L_3,alpha=0.2)
                    #
```

```
                    # #
                    # ax.
↪scatter(y[n],z[n],y[n],color='aqua',s=sz_1*ff*2,marker=L_0)
                    # ax.
↪scatter(y[n],z[n],-y[n],color='brown',s=sz_2*ff,marker=L_1,alpha=0.3)
                    # ax.
↪scatter(y[n],-z[n],y[n],color='aqua',s=sz_3*ff,marker=L_2,alpha=0.2)
                    # ax.
↪scatter(y[n],-z[n],-y[n],color='brown',s=sz_4*ff,marker=L_3,alpha=0.3)
                    # ax.
↪scatter(-y[n],z[n],y[n],color='aqua',s=sz_4*ff,marker=L_3,alpha=0.3)
                    # ax.
↪scatter(-y[n],z[n],-y[n],color='brown',s=sz_4*ff,marker=L_3,alpha=0.3)
                    # ax.
↪scatter(-y[n],-z[n],y[n],color='aqua',s=sz_4*ff,marker=L_3,alpha=0.3)
                    # ax.
↪scatter(-y[n],-z[n],-y[n],color='brown',s=sz_4*ff,marker=L_3,alpha=0.3)
                    # #
                    # ax.
↪scatter(z[n],x[n],y[n],color='aqua',s=sz_1*ff*2,marker=L_0)
                    # ax.
↪scatter(z[n],x[n],-y[n],color='brown',s=sz_2*ff,marker=L_1,alpha=0.3)
                    # ax.
↪scatter(z[n],-x[n],y[n],color='aqua',s=sz_3*ff,marker=L_2,alpha=0.2)
                    # ax.
↪scatter(z[n],-x[n],-y[n],color='brown',s=sz_4*ff,marker=L_3,alpha=0.3)
                    # ax.
↪scatter(-z[n],x[n],y[n],color='aqua',s=sz_4*ff,marker=L_3,alpha=0.3)
                    # ax.
↪scatter(-z[n],x[n],-y[n],color='brown',s=sz_4*ff,marker=L_3,alpha=0.3)
                    # ax.
↪scatter(-z[n],-x[n],y[n],color='aqua',s=sz_4*ff,marker=L_3,alpha=0.3)
                    # ax.
↪scatter(-z[n],-x[n],-y[n],color='brown',s=sz_4*ff,marker=L_3,alpha=0.3)
                    # #
                    # ax.
↪scatter(z[n],y[n],x[n],color='aqua',s=sz_1*ff*2,marker=L_0)
                    # ax.
↪scatter(z[n],y[n],-x[n],color='brown',s=sz_2*ff,marker=L_1,alpha=0.3)
                    # ax.
↪scatter(z[n],-y[n],x[n],color='aqua',s=sz_3*ff,marker=L_2,alpha=0.2)
                    # ax.
↪scatter(z[n],-y[n],-x[n],color='brown',s=sz_4*ff,marker=L_3,alpha=0.3)
```

```
                    # ax.
↪scatter(-z[n],y[n],x[n],color='aqua',s=sz_4*ff,marker=L_3,alpha=0.3)
                    # ax.
↪scatter(-z[n],y[n],-x[n],color='brown',s=sz_4*ff,marker=L_3,alpha=0.3)
                    # ax.
↪scatter(-z[n],-y[n],x[n],color='aqua',s=sz_4*ff,marker=L_3,alpha=0.3)
                    # ax.
↪scatter(-z[n],-y[n],-x[n],color='brown',s=sz_4*ff,marker=L_3,alpha=0.3)
                    #

                else:
                    ax.
↪scatter(x[n],y[n],z[n],color=symbol_color(symbol[n-d_0]),s=sz_1,marker=L_1)
                    ax.
↪scatter(x[n],-y[n],z[n],color='gray',s=sz_2,marker=L_1,alpha=0.3)
                    ax.
↪scatter(-x[n],y[n],z[n],color='gray',s=sz_3,marker=L_2,alpha=0.2)
                    ax.
↪scatter(-x[n],-y[n],z[n],color='gray',s=sz_4,marker=L_3,alpha=0.3)
                    ax.
↪scatter(y[n],x[n],z[n],color='gray',s=sz_2,marker=L_1,alpha=0.3)
                    ax.
↪scatter(y[n],-x[n],z[n],color='gray',s=sz_3,marker=L_2,alpha=0.2)
                    ax.
↪scatter(-y[n],x[n],z[n],color='gray',s=sz_4,marker=L_3,alpha=0.3)
                    ax.
↪scatter(-y[n],-x[n],z[n],color='gray',s=sz_2,marker=L_1,alpha=0.3)

                    #
                    ax.
↪scatter(x[n],y[n],-z[n],color=symbol_color(symbol[n-d_0]),s=sz_1,marker=L_1)
                    ax.
↪scatter(x[n],-y[n],-z[n],color='gray',s=sz_2,marker=L_1,alpha=0.3)
                    ax.
↪scatter(-x[n],y[n],-z[n],color='gray',s=sz_3,marker=L_2,alpha=0.2)
                    ax.
↪scatter(-x[n],-y[n],-z[n],color='gray',s=sz_4,marker=L_3,alpha=0.3)
                    ax.
↪scatter(y[n],x[n],-z[n],color='gray',s=sz_2,marker=L_1,alpha=0.3)
                    ax.
↪scatter(y[n],-x[n],-z[n],color='gray',s=sz_3,marker=L_2,alpha=0.2)
                    ax.
↪scatter(-y[n],x[n],-z[n],color='gray',s=sz_4,marker=L_3,alpha=0.3)
                    ax.
↪scatter(-y[n],-x[n],-z[n],color='gray',s=sz_2,marker=L_1,alpha=0.3)
```

```
                    #
                    # ax.
↪scatter(x[n],z[n],y[n],color='gray',s=sz_1*2,marker=L_0)
                    # ax.
↪scatter(x[n],z[n],-y[n],color='gray',s=sz_2,marker=L_1,alpha=0.2)
                    # ax.
↪scatter(x[n],-z[n],y[n],color='gray',s=sz_3,marker=L_2,alpha=0.2)
                    # ax.
↪scatter(x[n],-z[n],-y[n],color='gray',s=sz_4,marker=L_3,alpha=0.2)
                    # ax.
↪scatter(-x[n],z[n],y[n],color='gray',s=sz_4,marker=L_3,alpha=0.2)
                    # ax.
↪scatter(-x[n],z[n],-y[n],color='gray',s=sz_4,marker=L_3,alpha=0.2)
                    # ax.
↪scatter(-x[n],-z[n],y[n],color='gray',s=sz_4,marker=L_3,alpha=0.2)
                    # ax.
↪scatter(-x[n],-z[n],-y[n],color='gray',s=sz_4,marker=L_3,alpha=0.2)
                    # #
                    # ax.
↪scatter(y[n],x[n],z[n],color='gray',s=sz_1*2,marker=L_0)
                    # ax.
↪scatter(y[n],x[n],-z[n],color='gray',s=sz_2,marker=L_1,alpha=0.2)
                    # ax.
↪scatter(y[n],-x[n],z[n],color='gray',s=sz_3,marker=L_2,alpha=0.2)
                    # ax.
↪scatter(y[n],-x[n],-z[n],color='gray',s=sz_4,marker=L_3,alpha=0.2)
                    # ax.
↪scatter(-y[n],x[n],z[n],color='gray',s=sz_4,marker=L_3,alpha=0.2)
                    # ax.
↪scatter(-y[n],x[n],-z[n],color='gray',s=sz_4,marker=L_3,alpha=0.2)
                    # ax.
↪scatter(-y[n],-x[n],z[n],color='gray',s=sz_4,marker=L_3,alpha=0.2)
                    # ax.
↪scatter(-y[n],-x[n],-z[n],color='gray',s=sz_4,marker=L_3,alpha=0.2)
                    # #
                    # ax.
↪scatter(y[n],z[n],y[n],color='gray',s=sz_1*2,marker=L_0)
                    # ax.
↪scatter(y[n],z[n],-y[n],color='gray',s=sz_2,marker=L_1,alpha=0.2)
                    # ax.
↪scatter(y[n],-z[n],y[n],color='gray',s=sz_3,marker=L_2,alpha=0.2)
                    # ax.
↪scatter(y[n],-z[n],-y[n],color='gray',s=sz_4,marker=L_3,alpha=0.2)
                    # ax.
↪scatter(-y[n],z[n],y[n],color='gray',s=sz_4,marker=L_3,alpha=0.2)
```

```
                    # ax.
↪scatter(-y[n],z[n],-y[n],color='gray',s=sz_4,marker=L_3,alpha=0.2)
                    # ax.
↪scatter(-y[n],-z[n],y[n],color='gray',s=sz_4,marker=L_3,alpha=0.2)
                    # ax.
↪scatter(-y[n],-z[n],-y[n],color='gray',s=sz_4,marker=L_3,alpha=0.2)
                    # #
                    # ax.
↪scatter(z[n],x[n],y[n],color='gray',s=sz_1*2,marker=L_0)
                    # ax.
↪scatter(z[n],x[n],-y[n],color='gray',s=sz_2,marker=L_1,alpha=0.2)
                    # ax.
↪scatter(z[n],-x[n],y[n],color='gray',s=sz_3,marker=L_2,alpha=0.2)
                    # ax.
↪scatter(z[n],-x[n],-y[n],color='gray',s=sz_4,marker=L_3,alpha=0.2)
                    # ax.
↪scatter(-z[n],x[n],y[n],color='gray',s=sz_4,marker=L_3,alpha=0.2)
                    # ax.
↪scatter(-z[n],x[n],-y[n],color='gray',s=sz_4,marker=L_3,alpha=0.2)
                    # ax.
↪scatter(-z[n],-x[n],y[n],color='gray',s=sz_4,marker=L_3,alpha=0.2)
                    # ax.
↪scatter(-z[n],-x[n],-y[n],color='gray',s=sz_4,marker=L_3,alpha=0.2)
                    # #
                    # ax.
↪scatter(z[n],y[n],x[n],color='gray',s=sz_1*2,marker=L_0)
                    # ax.
↪scatter(z[n],y[n],-x[n],color='gray',s=sz_2,marker=L_1,alpha=0.2)
                    # ax.
↪scatter(z[n],-y[n],x[n],color='gray',s=sz_3,marker=L_2,alpha=0.2)
                    # ax.
↪scatter(z[n],-y[n],-x[n],color='gray',s=sz_4,marker=L_3,alpha=0.2)
                    # ax.
↪scatter(-z[n],y[n],x[n],color='gray',s=sz_4,marker=L_3,alpha=0.2)
                    # ax.
↪scatter(-z[n],y[n],-x[n],color='gray',s=sz_4,marker=L_3,alpha=0.2)
                    # ax.
↪scatter(-z[n],-y[n],x[n],color='gray',s=sz_4,marker=L_3,alpha=0.2)
                    # ax.
↪scatter(-z[n],-y[n],-x[n],color='gray',s=sz_4,marker=L_3,alpha=0.2)



                if allow_anim_png:
                  plt.savefig('./img/eigen_atlas/geometric_numbers_%d.png' % n)
```

```
[95]: def generate_primes(n):
          primes = []
          sieve = [True] * (n+1)
          p = 2
          while p * p <= n:
              if sieve[p]:
                  for i in range(p * p, n+1, p):
                      sieve[i] = False
              p += 1
          for p in range(2, n+1):
              if sieve[p]:
                  primes.append(p)
          return primes
```

```
[96]: 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"})
```

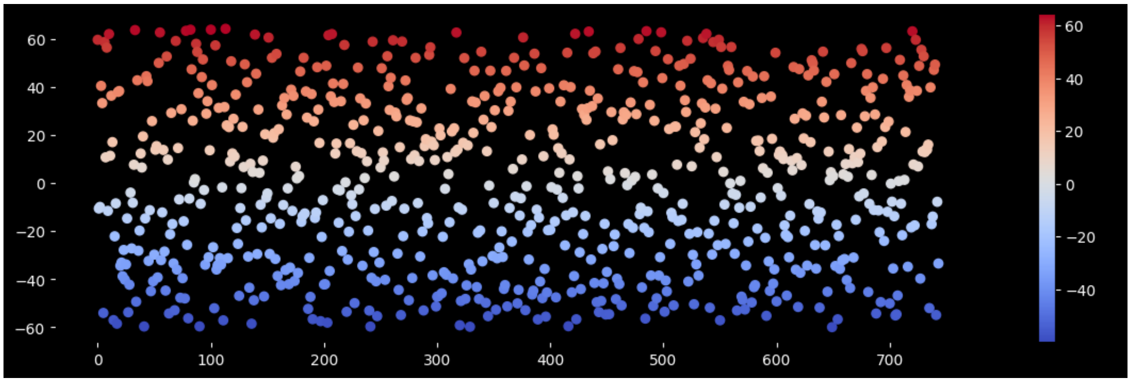
```
[97]: # Example use
      start = pd.Timestamp('2023-01-01 00:00:00')
      end = pd.Timestamp('2023-01-31 23:00:00')
      freq ='H'
      df   = create_random_climate_data(start,end,freq)
      year, month, day, hour, minute, second, temperature, humidity, pressure =␣
       ↪extract_climate_data(df)

      # Main index
      n = np.arange(len(df.index))
      h    = n
      h_24 = hour
      d    = day
      m    = month
```

```
y     = year

# Structural index
year_clk     =   compass(r=(2)**(0),T=7,s=year)
month_clk    =   compass(r=(2)**(-2.3),T=12,s=month)
day_clk      =   compass(r=(2)**(-4.5),T=31,s=day)
hour_clk     =   compass(r=(2)**(-8),T=24,s=n)

#temp_symbol =   compass(r=1,T=)



clk_0   =   hour_clk
clk_1   =   day_clk
clk_2   =   month_clk
clk_3   =   year_clk



atlas = clk_0+clk_1+clk_2+clk_3
```

2 Adapt the data to the time structure

Here are no general rules. You can use your compas however you like. As we are testing with climate data, and with possitive and negative numbers we have to adapt our data to our clocks structure.

[98]:
```
MAX=60
MIN=-60
K=3
n=n
seq_size=n.size
# symbol = np.arange(-60,61,0.5)
# symbol = np.random.randint(-60,61,300)
#symbol = np.random.rand(int(seq_size))*MAX + np.random.rand(seq_size)*MIN
symbol    = np.random.uniform(low=-60, high=65, size=seq_size)
cmap = plt.cm.coolwarm
norm = plt.Normalize(min(symbol), max(symbol))
colors = [cmap(norm(value)) for value in symbol]
init_atlas(14,4)

plt.scatter(np.arange(len(symbol)), symbol, color=colors)
#plt.plot(np.arange(len(symbol)), symbol)
cbar = plt.colorbar(plt.cm.ScalarMappable(norm=norm, cmap=cmap))
cbar.set_label('Color for Positive and Negative Values')

plt.show()
```



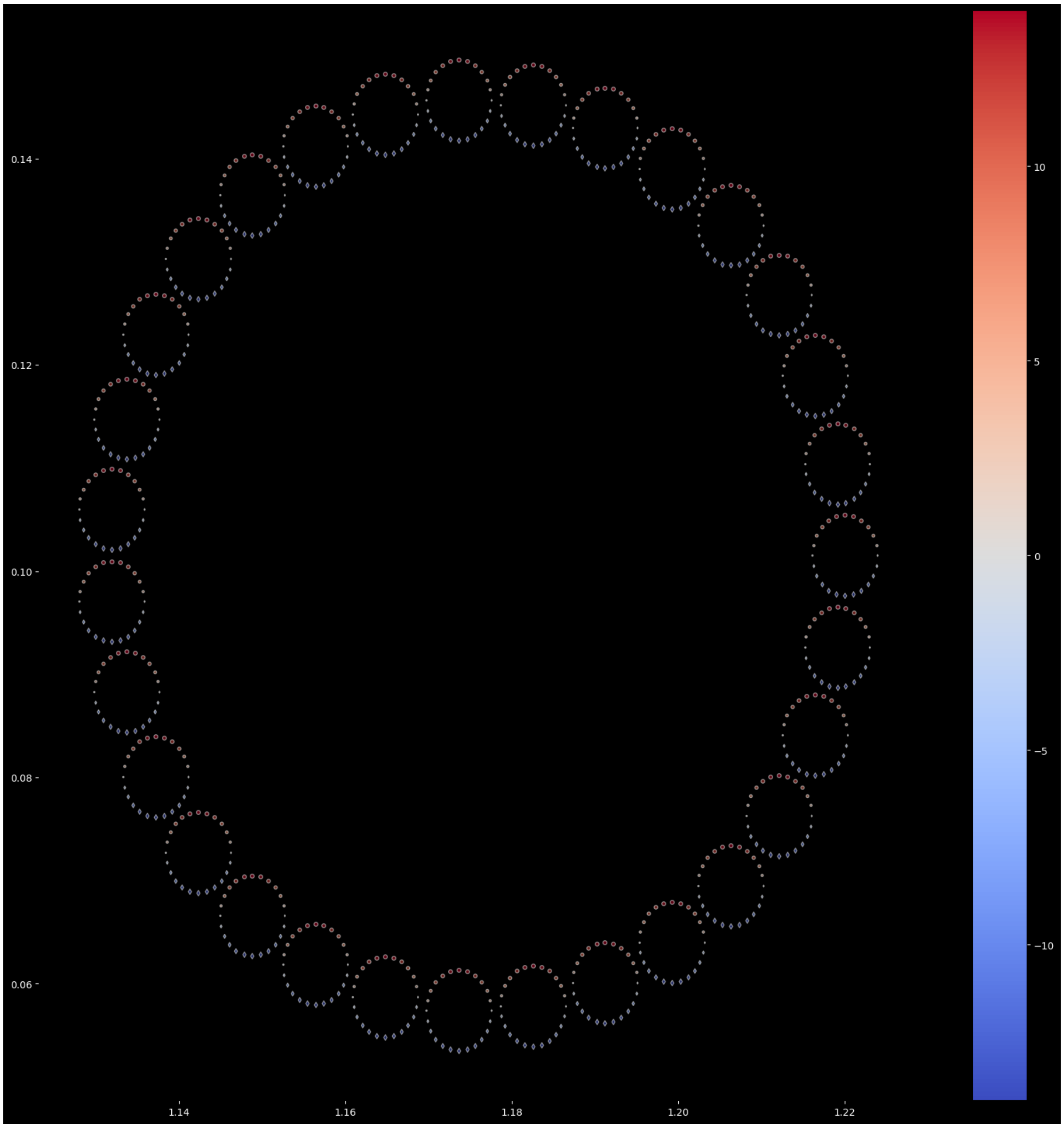
[99]:
```
atlas.size
```

[99]: 744

3 The sine function in 1 month atlas

$$s = r_s * \sin(\frac{2\pi}{24}n)$$

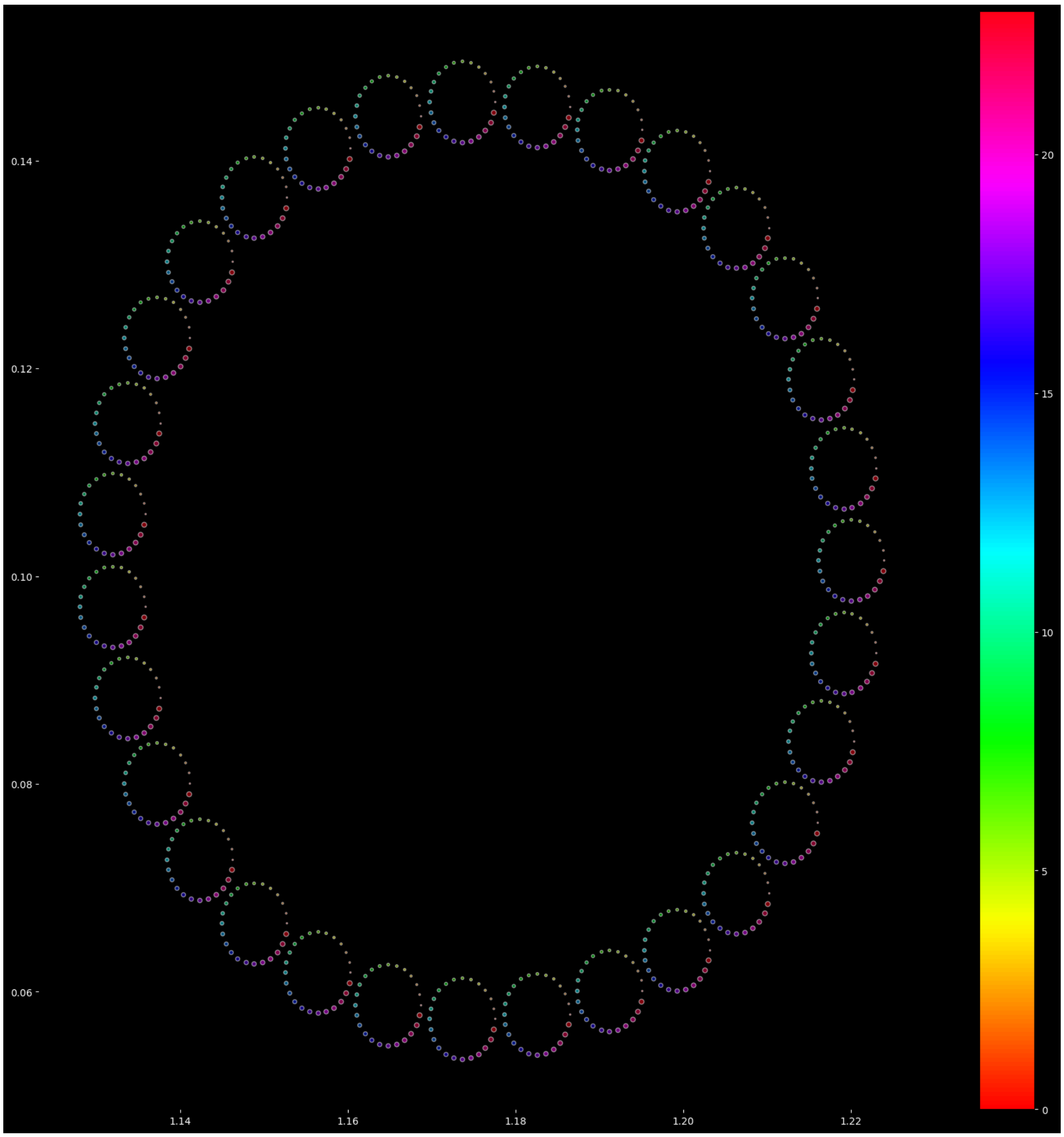
[100]:
```
symbol    =14*np.sin(2*np.pi/24*np.arange(len(atlas)))
cmap=plt.cm.coolwarm
plot_atlas(atlas=atlas,symbol=symbol,cmap=cmap,alpha=0.
 ↪5,sx=20,sy=20,legend_flag=1,variable_sizes=1,base_size=1)
```



3.1 Hour number 31 days

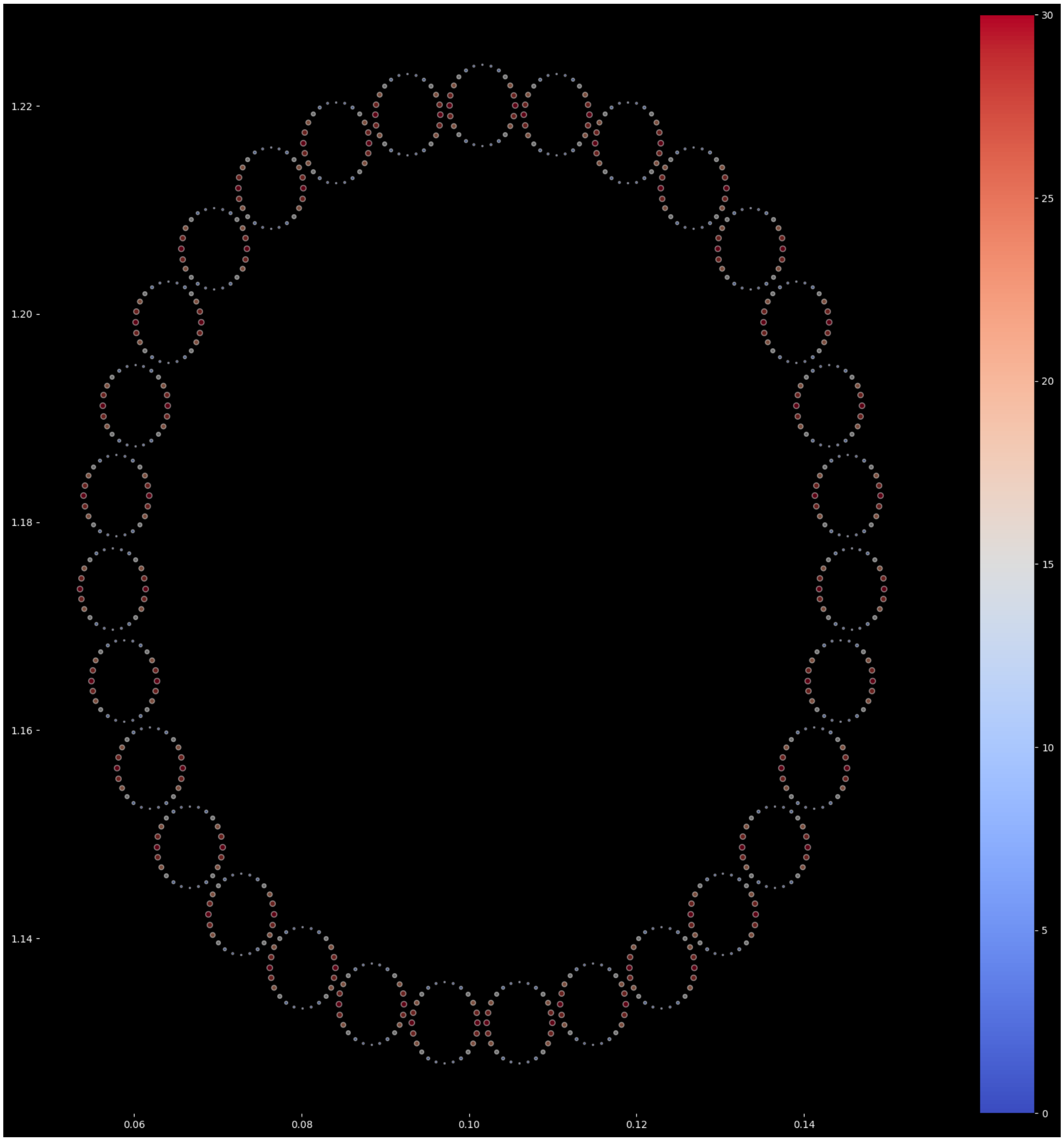
[101]:
```
# Structural index
year_clk    =   compass(r=(2)**(0),T=7,s=year)
month_clk   =   compass(r=(2)**(-2.3),T=12,s=month)
day_clk     =   compass(r=(2)**(-4.5),T=31,s=day)
hour_clk    =   compass(r=(2)**(-8),T=24,s=n)

#temp_symbol =   compass(r=1,T=)

smaller_clock = hour_clk
```

```
clk_0   =   smaller_clock
clk_1   =   day_clk
clk_2   =   month_clk
clk_3   =   year_clk


atlas = clk_0+clk_1+clk_2+clk_3
plot_atlas(atlas=atlas,symbol=hour,cmap=plt.cm.hsv,alpha=0.
 ↪5,sx=20,sy=20,legend_flag=1,variable_sizes=1,base_size=1)
```

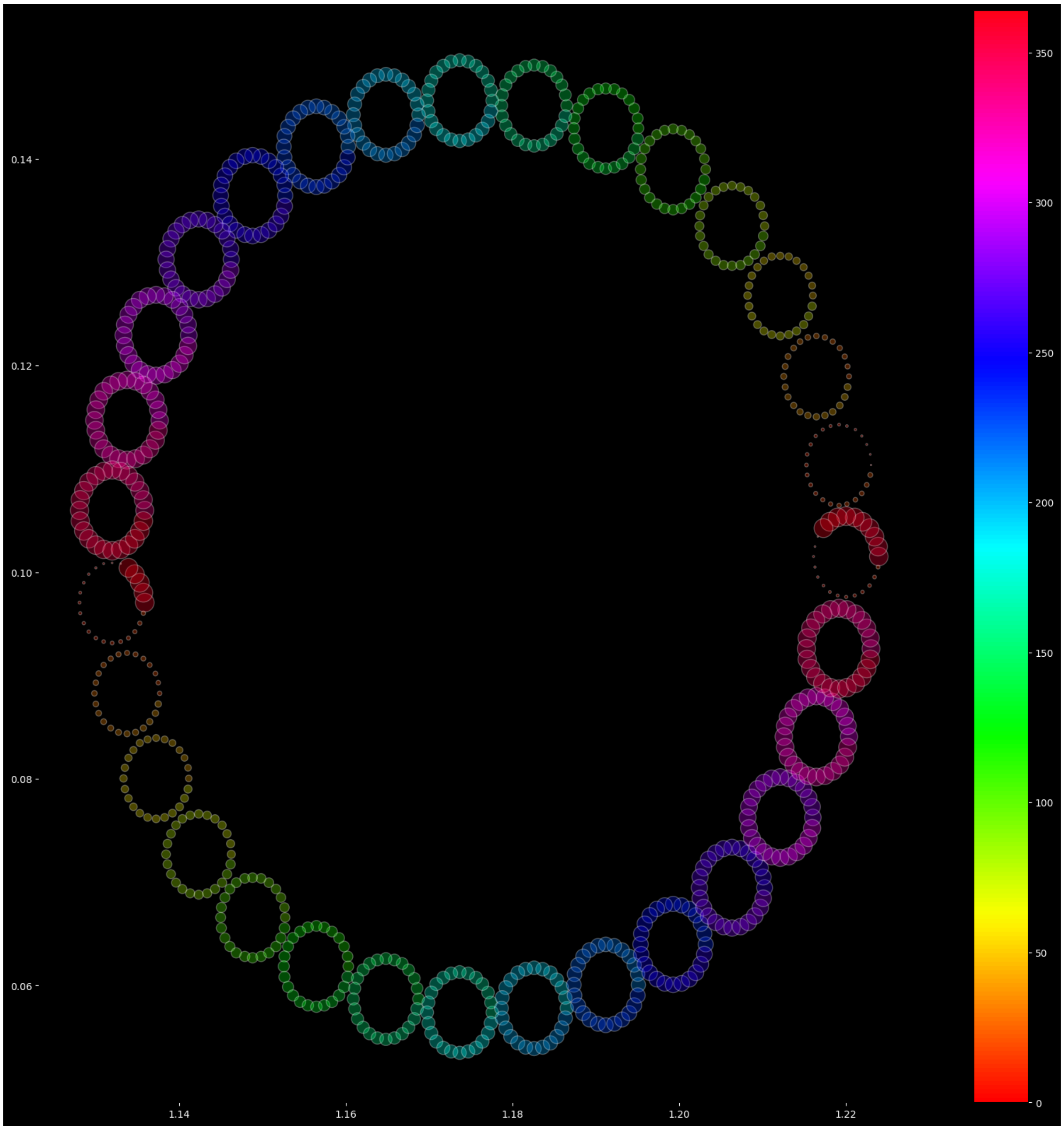


3.2 Other variations

[102]:
```
plot_atlas(atlas=atlas.imag+1j*atlas.real,symbol=30*np.sin(2*np.pi*hour/
↪24)**2,cmap=plt.cm.coolwarm,alpha=0.
↪5,sx=20,sy=20,legend_flag=1,variable_sizes=1,base_size=1)
```

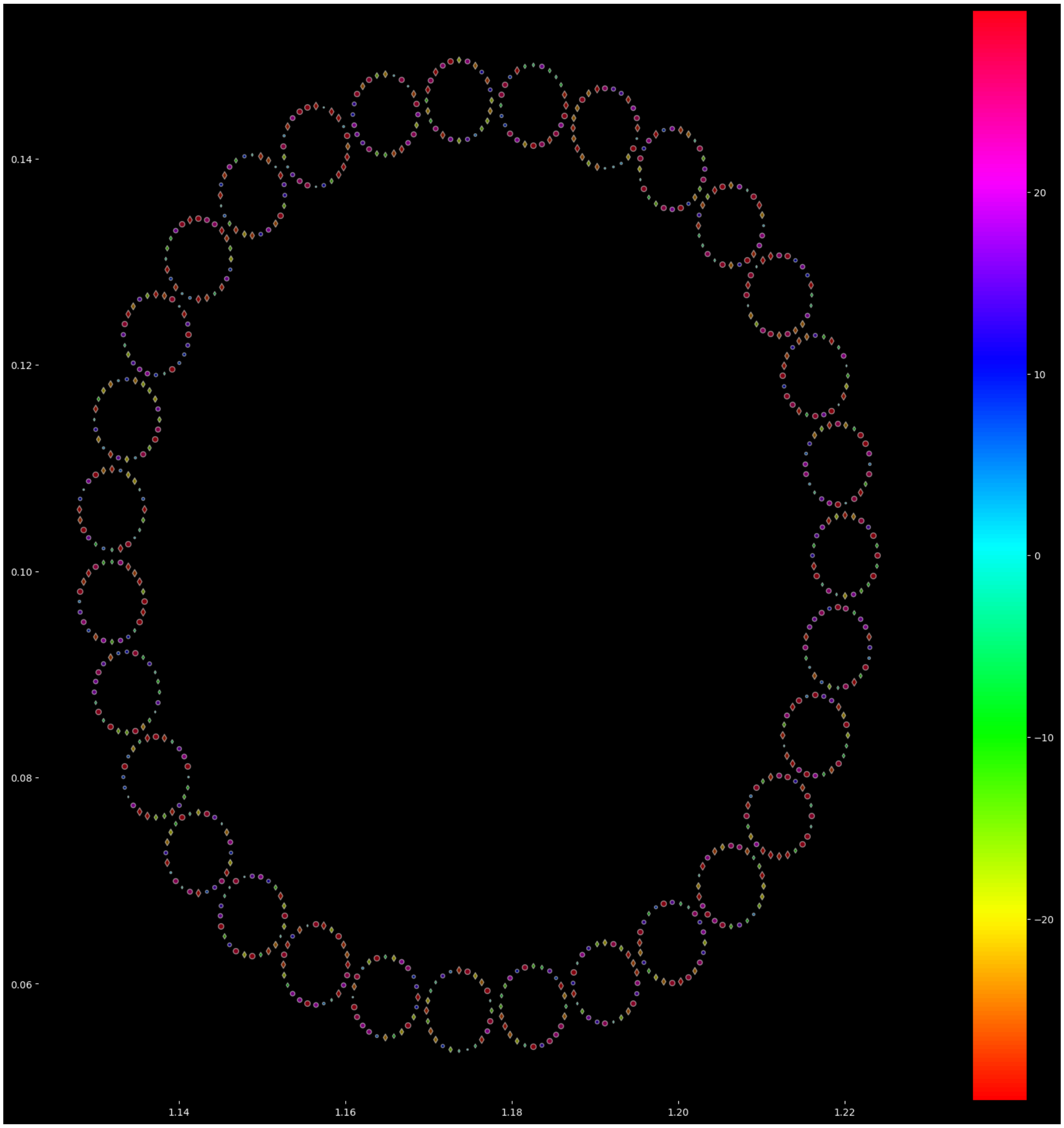


[103]:
```
plot_atlas(atlas=atlas,symbol=n%365,cmap=plt.cm.hsv,alpha=0.
↪3,sx=20,sy=20,legend_flag=1,variable_sizes=1,base_size=1)
```



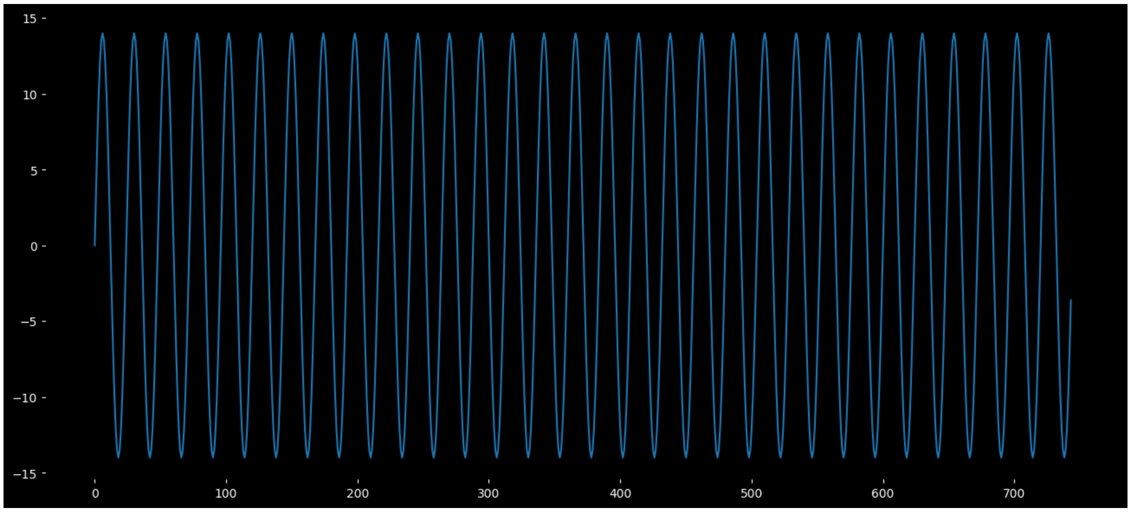
3.3 Random symbol

[104]:
```
plot_atlas(atlas=atlas,symbol=30*np.sin(2*np.pi*temperature/60),cmap=plt.cm.
    ↪hsv,alpha=0.5,sx=20,sy=20,legend_flag=1,variable_sizes=1,base_size=1)
```



[105]:
```
init_atlas(16,7)
plt.plot(symbol)
```

[105]: [<matplotlib.lines.Line2D at 0x7fa3de61d6f0>]

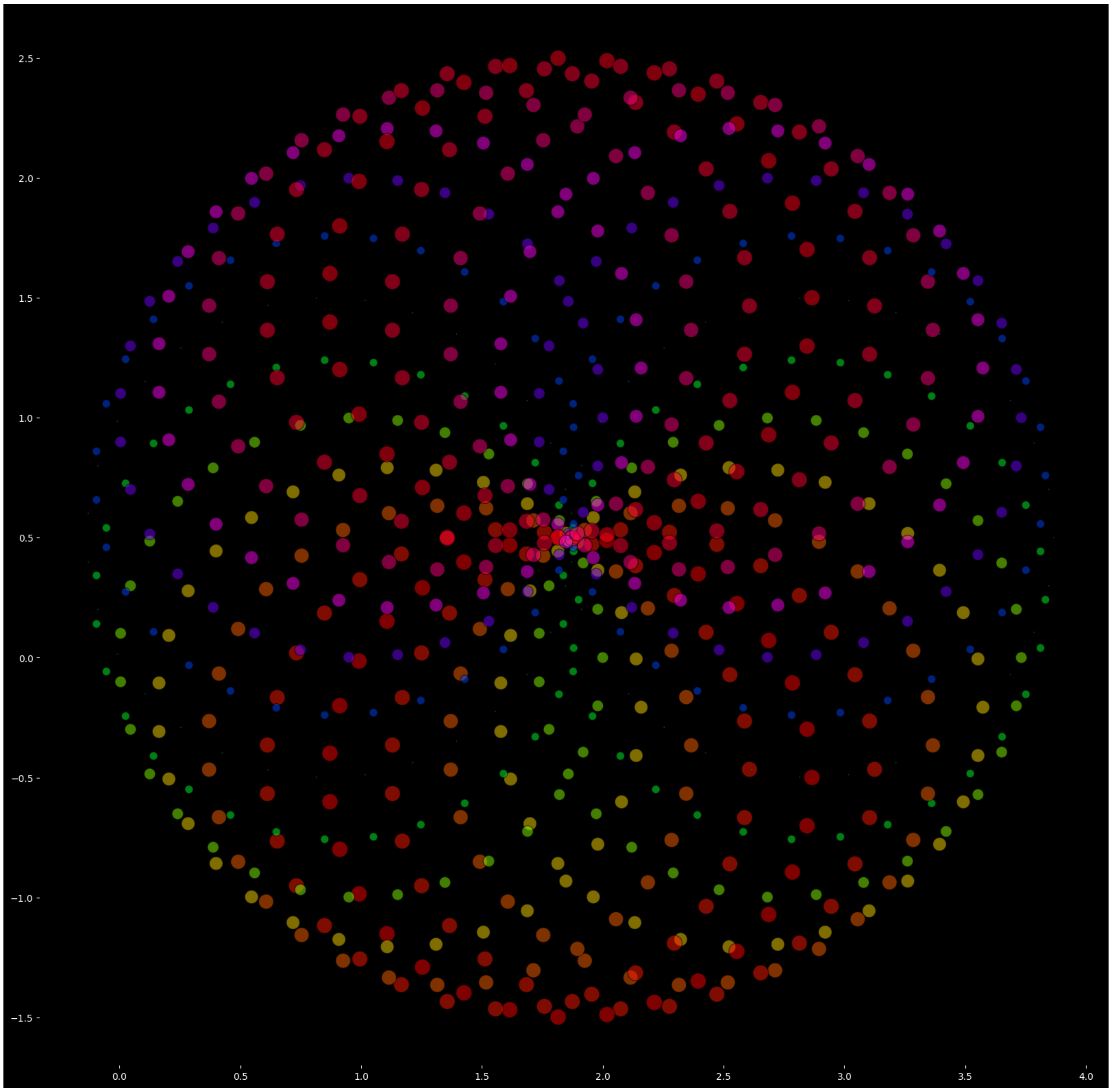


[106]:
```
T_0=24
T_1=30
T_2=12
T_3=1
n=np.arange(T_0*T_1*T_2*T_3)
s_0=n
s_1=n%T_1
s_2=n%T_2
s_3=n%T_3
```

3.3.1 Unfiltered data

[107]:
```
# Atlas
input_symbol=hour
clk_0   =   compass(r=1,T=24,s=hour)
clk_1   =   compass(r=1,T=31,s=day)
clk_2   =   compass(r=1,T=12,s=month)
clk_3   =   compass(r=1,T=7,s=year)


atlas_in = clk_0+clk_1+clk_2+clk_3

markers,norm,colors,norm_seq,sizes=get_colors(symbol,plt.cm.hsv,1,20)

init_atlas(20,20)

plt.scatter(atlas_in.real,atlas_in.imag, color=colors, s=sizes, marker='o',␣
  ↪facecolor=colors, edgecolors='black', alpha=0.5)
```

[107]:
```
<matplotlib.collections.PathCollection at 0x7fa3e41d4be0>
```



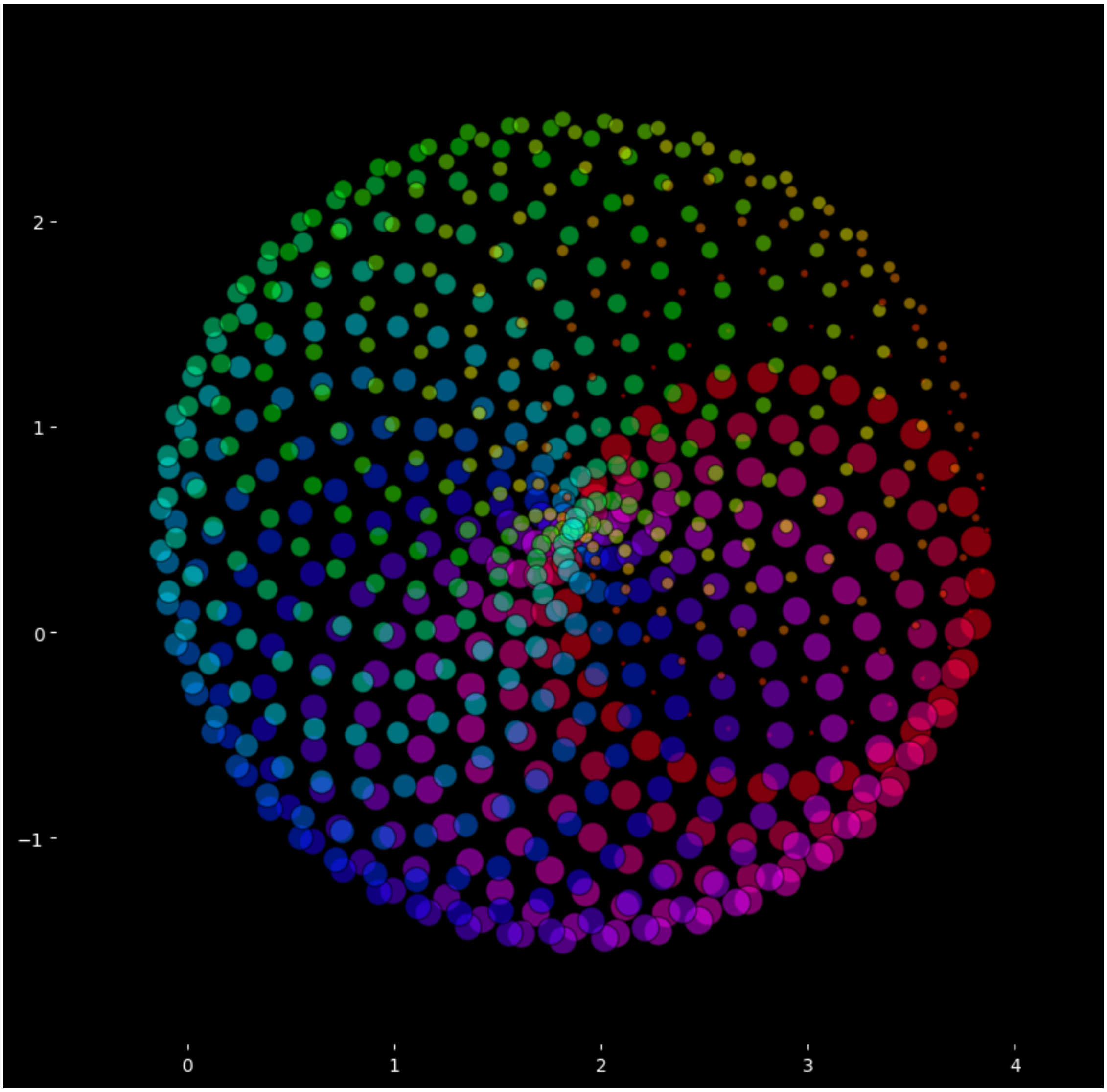
3.4 Animations with NO memory

[108]:
```
# # Atlas
# input_symbol = hour
# clk_0 = compass(r=1, T=24, s=hour)
# clk_1 = compass(r=1, T=31, s=day)
# clk_2 = compass(r=1, T=12, s=month)
# clk_3 = compass(r=1, T=7, s=year)

# atlas_pre_filter = clk_0 + clk_1 + clk_2 + clk_3

# markers, norm, colors, norm_seq, sizes = get_colors(symbol, plt.cm.hsv, 1, 20)
```

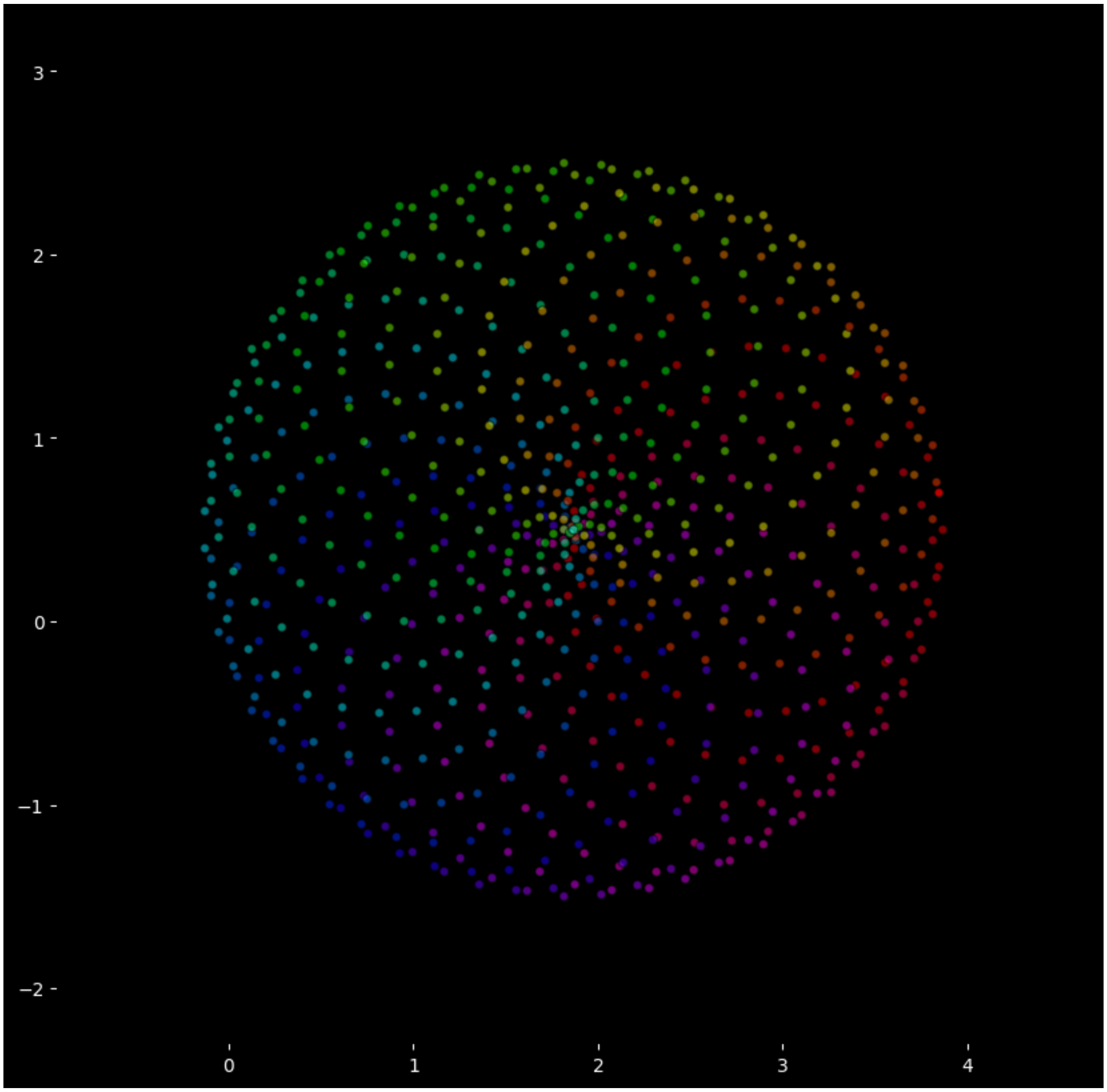
```
# init_atlas(20, 20)

# fig = plt.figure()
# scatter = plt.scatter(atlas_pre_filter.real, atlas_pre_filter.imag,␣
↪color=colors, s=sizes, marker='o',
#                        facecolor=colors, edgecolors='black', alpha=0.5)

# def update(frame):
#     # Update the scatter plot with each element of the atlas_pre_filter array
#     scatter.set_offsets([atlas_pre_filter[frame].real,␣
↪atlas_pre_filter[frame].imag])

# animation = FuncAnimation(fig, update, frames=len(atlas_pre_filter),␣
↪blit=False)

# # Specify the writer (FFMpegWriter) and the output filename
# writer = FFMpegWriter(fps=10)  # Adjust the frames per second (fps) as needed
# animation.save('animation.mp4', writer=writer)
```

3.5 Animations WITH Memory

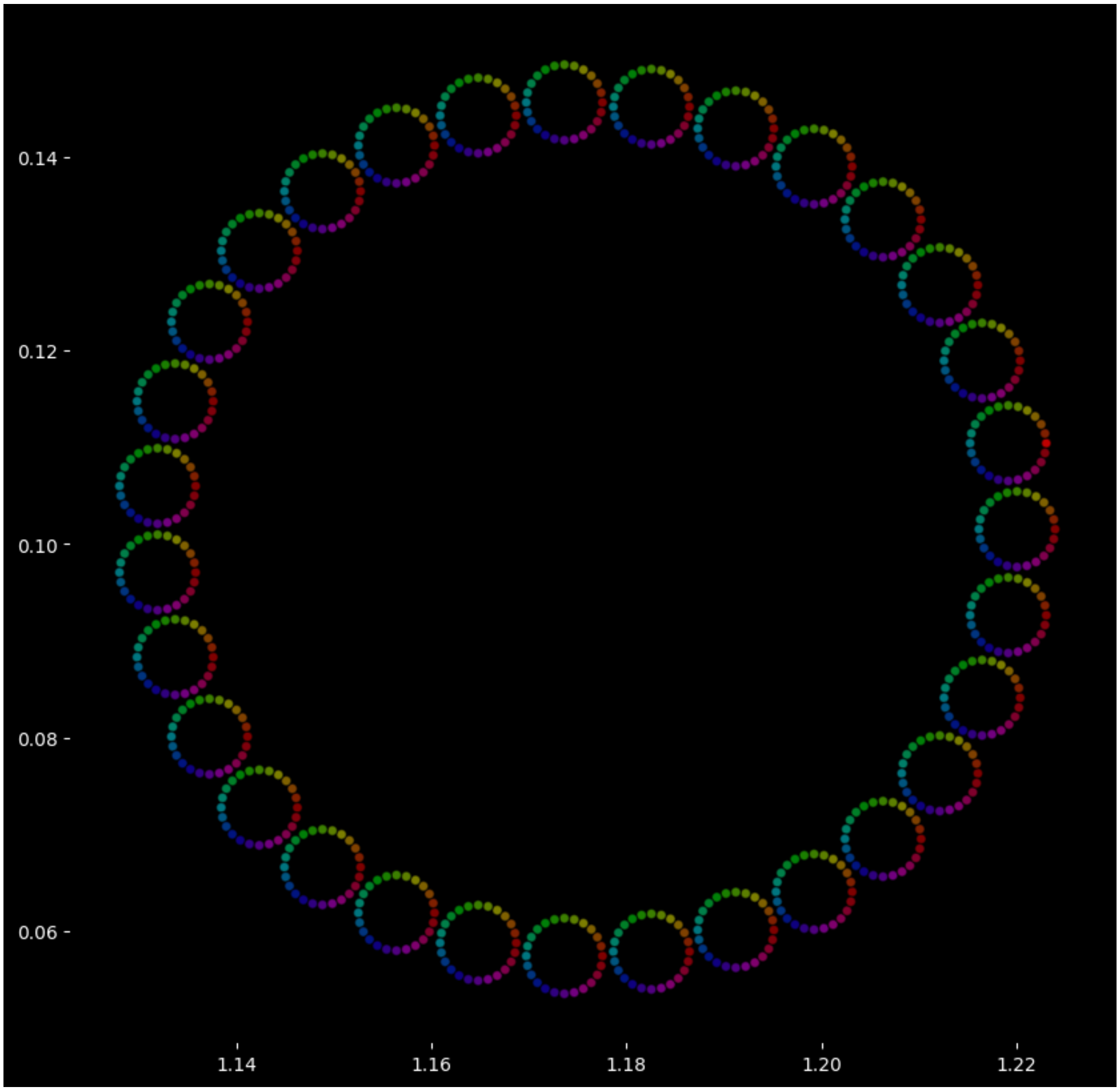
[109]:
```
symbol=hour
#symbol=hour
atlas_in = clk_0 + clk_1 + clk_2 + clk_3
fps=10
colormap=plt.cm.hsv
output_path='2_high_res_hourly_aliased_variable_size.mp4'
variable_size=1
fixed_size=3
padding=0.5
sx=10
sy=sx
get_atlas_video(atlas_in,symbol,fps,colormap,output_path,variable_size,fixed_size,padding,sx,s
```



[110]:
```
symbol=hour
#symbol=hour
atlas_in = clk_0 + clk_1 + clk_2 + clk_3
fps=10
colormap=plt.cm.hsv
output_path='High_reshourly_aliased_fixed_size.mp4'
variable_size=0
fixed_size=6
padding=0.8
get_atlas_video(atlas_in,symbol,fps,colormap,output_path,variable_size,fixed_size,padding,sx,s
```



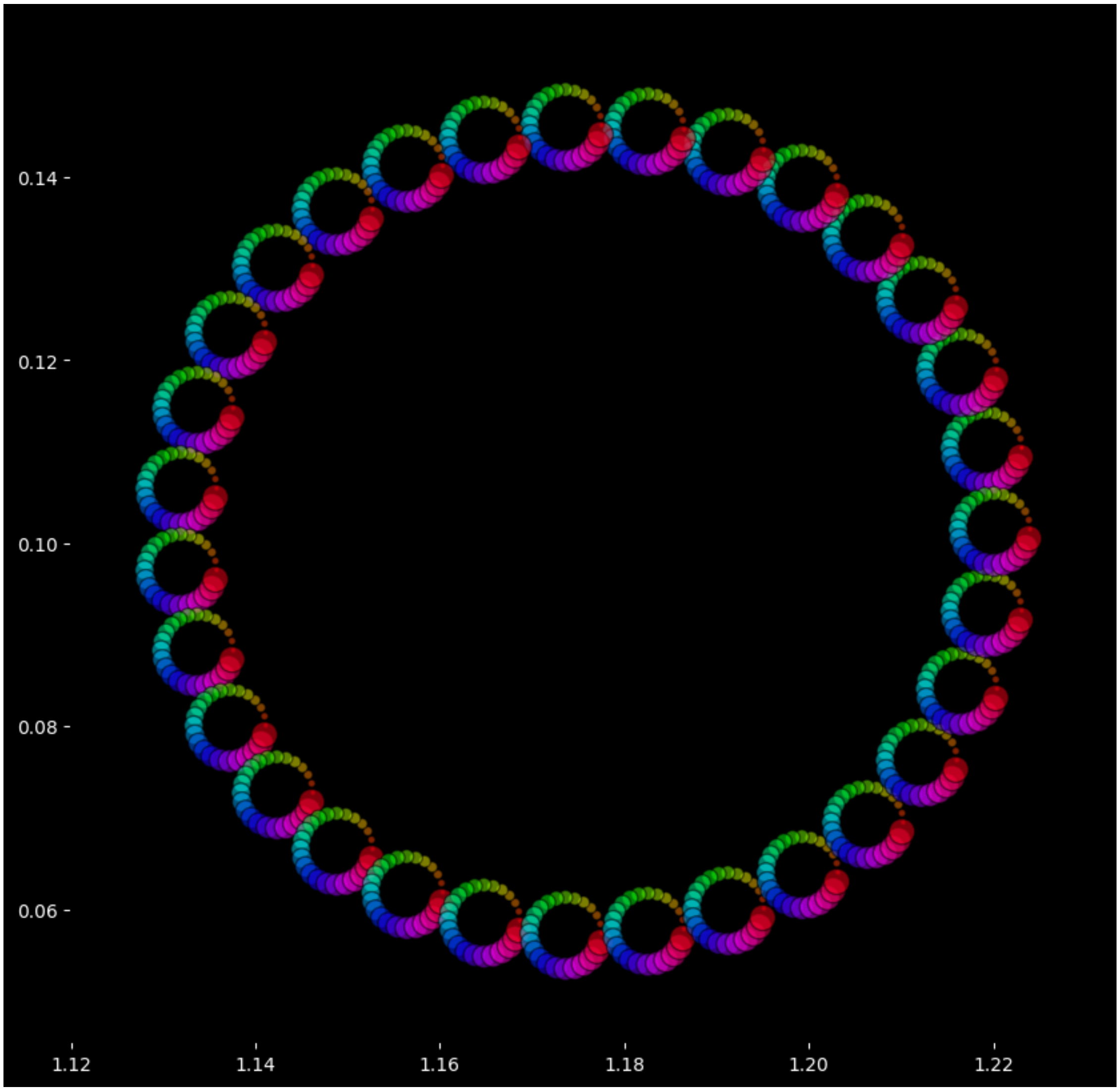
[111]:
```
symbol=hour
#symbol=hour
#atlas_in = clk_0 + clk_1 + clk_2 + clk_3
fps=10
colormap=plt.cm.hsv
output_path='Highreshourly_filtered_fixed_size.mp4'
variable_size=0
fixed_size=7
padding=0.005
start=0
end=atlas.size
get_atlas_video(atlas[start:end],symbol[start:
↪end],fps,colormap,output_path,variable_size,fixed_size,padding,sx,sy)
```



[112]:

```
symbol=hour
#symbol=hour
#atlas_in = clk_0 + clk_1 + clk_2 + clk_3
fps=10
colormap=plt.cm.hsv
output_path='High_res_hourly_filtered_variable_size.mp4'
variable_size=1#flag 1 for variable size or 0 for fixed size
fixed_size=2
padding=0.008
start=0
end=atlas.size
sx=10
sy=10
```

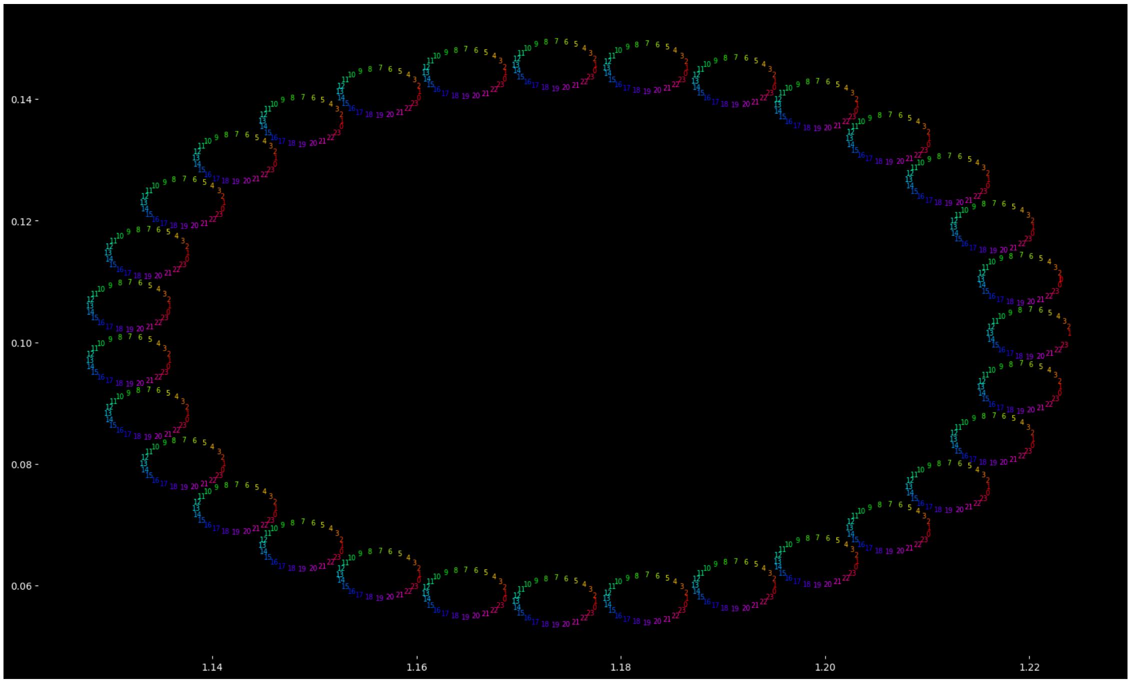
```
get_atlas_video(atlas[start:end],symbol[start:
 ↪end],fps,colormap,output_path,variable_size,fixed_size,padding,sx,sy)
```



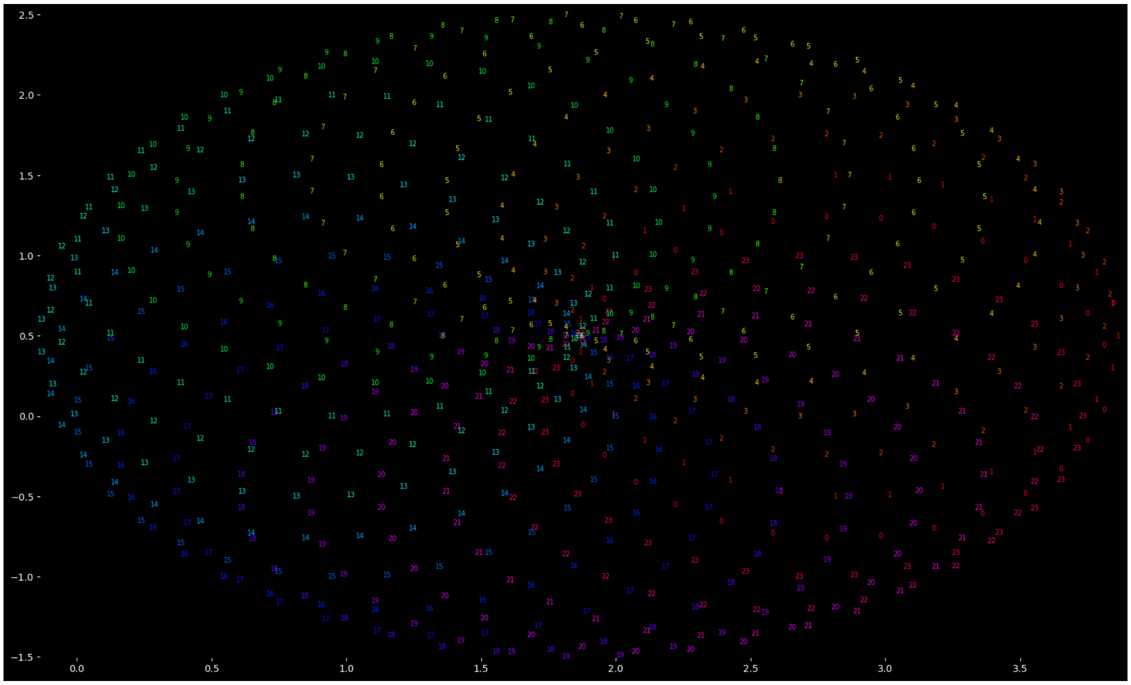
3.6 Adding text to the animations

[144]:
```
symbol=hour
#symbol=hour
#atlas_in = clk_0 + clk_1 + clk_2 + clk_3
fps=10
colormap=plt.cm.hsv
output_path='1month_hourly_size.mp4'
variable_size=0
fixed_size=0.1
```

```
padding=0.005
start=0
end=atlas.size
get_atlas_video_text(atlas[start:end],symbol[start:
  ↪end],fps,colormap,output_path,variable_size,fixed_size,padding,sx,sy)
```



[145]:
```
symbol=hour
#symbol=hour
atlas_in = clk_0 + clk_1 + clk_2 + clk_3
fps=10
colormap=plt.cm.hsv
output_path='symbol_size.mp4'
variable_size=0
fixed_size=0.1
padding=0.005
start=0
end=atlas.size
get_atlas_video_text(atlas_in[start:end],symbol[start:
  ↪end],fps,colormap,output_path,variable_size,fixed_size,padding,sx,sy)
```



3.7 Custom structure - a 360 days year simplification

If we didn't use the datetime structure from python, we could imagine we design our own calendar going from the most simple one to understand of one year of 360 days (to be 1)

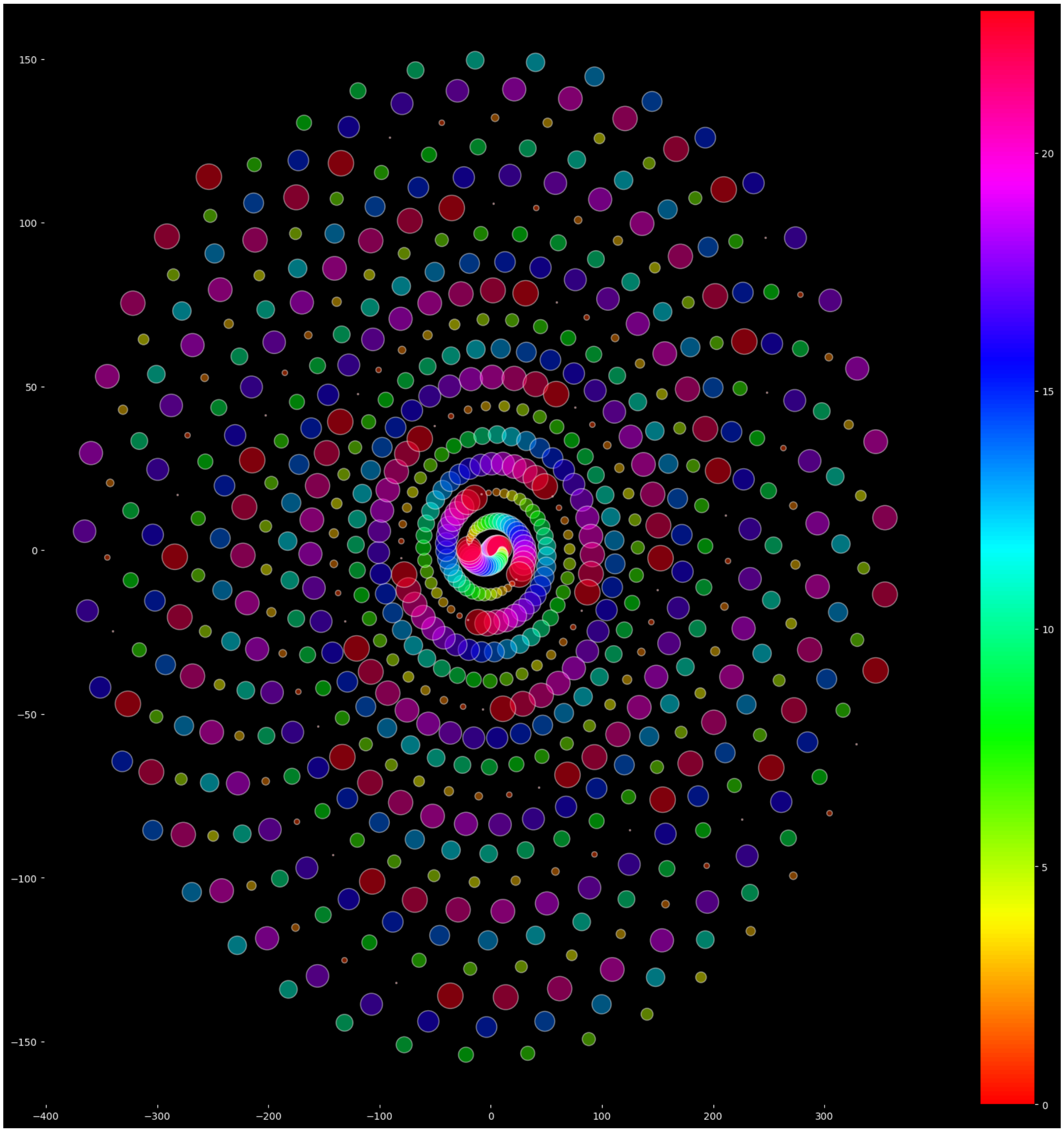
3.8 Eigen Atlas for Hourly Data

[113]:
```
# Inputs for basic eigen_atlas plot
sessions     = 3
To=39
M=To
sym_src='input' # 'nmod' , 'input' or 'rand'
start_index=0







stretch_space=4 # 4 is one "natural" factor
stretch_time=3 # 3 is one 'natural' factor
phase=5
norm=3
frac=1/2
```

```
#Initial conditions

k_o         = 1/stretch_time
k_1 =1/(stretch_space*To)
r_o=To*k_1
t_o=To*k_o

delta_0 = 1/To

f_psi=1/10
f_log_scat = 1/10

#root impedances - initial conditions
root_phase=np.sqrt(phase)
root_norm=np.sqrt(norm)
r_phase=frac*(1+root_phase)
r_norm=frac*(1+1j*root_norm)

# Input symbols
```

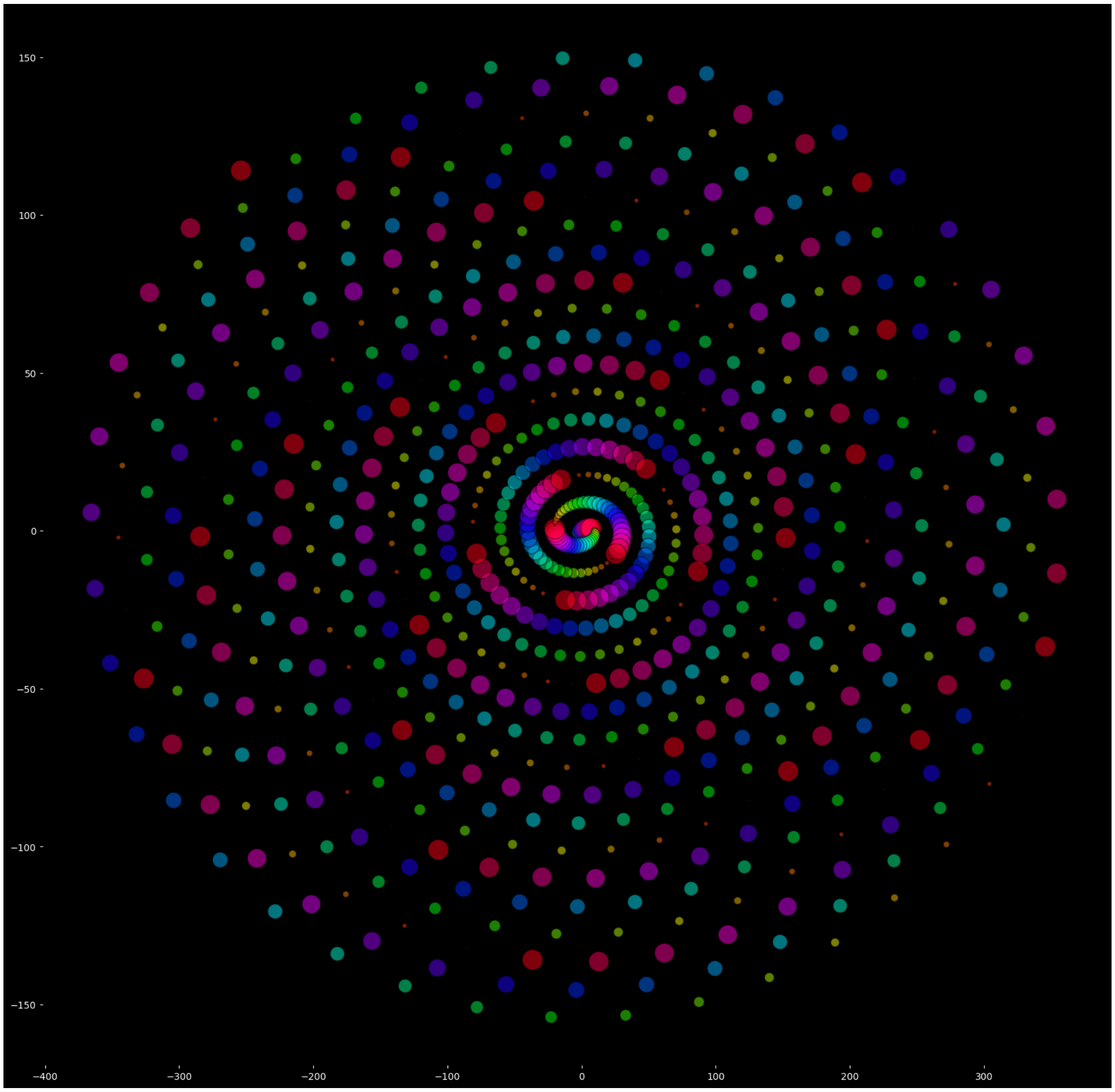
[114]:
```
# Symbol follow the index
input_symbol=hour
batch_size  = input_symbol.size
sym_src='input' # 'nmod' , 'input' or 'rand'
eigen_atlas=get_eigen_atlas_2D(input_symbol,sym_src,start_index,batch_size,M,To,k_o,r_o,t_o,fr
```

[115]:
```
z_carrier=eigen_atlas.z_carrier.values
plot_atlas(atlas=z_carrier[0:1000],symbol=input_symbol[0:1000],cmap=plt.cm.
 ↪hsv,alpha=0.5,sx=20,sy=20,legend_flag=1,variable_sizes=1,base_size=29)
```



[116]:
```
markers,norm,colors,norm_seq,sizes=get_colors(input_symbol,plt.cm.hsv,1,20)

init_atlas(20,20)

plt.scatter(z_carrier.real, z_carrier.imag, color=colors, s=sizes, marker='o',␣
  ↪facecolor=colors, edgecolors='black', alpha=0.5)
```

[116]: <matplotlib.collections.PathCollection at 0x7fa3dea03a90>



[117]:
```
# markers, norm, colors, norm_seq, sizes = get_colors(input_symbol, plt.cm.hsv,␣
↪1, 20)

# init_atlas(20, 20)

# plt.scatter(z_carrier.real, z_carrier.imag, c=norm_seq, cmap=plt.cm.hsv,␣
↪s=sizes, marker=markers[value : for value in n], edgecolors='black', alpha=0.
↪5)
# plt.colorbar()
```

[118]:
```
# Start and end index
start=0
end=batch_size
```

```
#Plot settings (flags)
allow_plot = 1
allow_scatter = 1
allow_text = 1
allow_anim_png = 0
mode ='3D' # '2D' or '3D'

#Plot constants
base_symbol_size=25
base_text_size=16
dots_per_inch=300
L_0="*"
L_1="o"
L_2="d"
L_3="o"
L_4="d"
L_5="o"
L_6="d"
L_7="o"
d_0=0
d_1=1
d_2=2
d_3=3
d_4=4
d_5=5
d_6=6
d_7=7
jump=0
sz_1=base_symbol_size
sz_2=sz_1+jump
sz_3=sz_2+jump
sz_4=sz_3+jump
sz_5=sz_4+jump
sz_6=sz_5+jump
sz_7=sz_6+jump
sz_8=sz_7+jump
colormap_time='hsv'
colormap_symbol='hsv'
sx=20
sy=12

alpha_non_prime=0.22
# Plot colors
time_color = cm.get_cmap(colormap_time, To)
symbol_color  =cm.get_cmap(colormap_symbol, M)
```

```
symbol_size =base_symbol_size*np.ones(batch_size,dtype=int)
text_size =base_text_size*np.ones(batch_size,dtype=int)
allow_anim_png=0
#plot_eigen_atlas(eigen_atlas,sx,sy,start,end,allow_scatter,allow_text,allow_anim_png,mode,sym
```

[ ]:

```
! jupyter nbconvert  --to pdf -o clean_v0.pdf  clean_code.ipynb
```