Análise de dados espaciais com animações do matplotlib

Tópicos Avançados de Bases de Dados

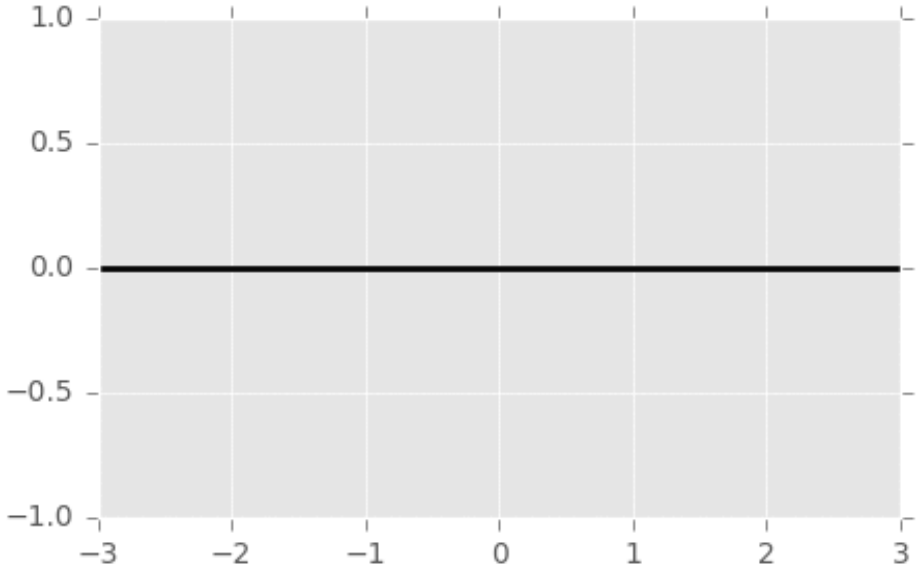
Animação de dados

- A exploração de informação é uma tarefa importante, sendo essencial o desenvolvimento de boas ferramentas de visualização;

- Em muitos casos, a visualização torna-se complexa quando a informação tem mais que duas dimensões, como acontece nos dados que temos usado (longitude, latitude, e dimensão temporal);

- Em muito exemplos, a possibilidade de animarmos os dados permite facilitar e melhorar as tarefas de análise.

Animações com o matplotlib (linhas)



Animação passo a passo

Passo 1: importar os módulos necessários

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

Passo 2: definir a área de desenho

```
fig, ax = plt.subplots(figsize=(5, 3))
ax.set(xlim=(-3, 3), ylim=(-1, 1))
```

A primeira linha define a figura e os seus eixos. A segunda linha fixa os limites para os eixos.
Fixar os limites evita o redimensionamento dinâmico da figura que tornaria a animação confusa e pouco legível.

Passo 3: criar os dados que vão ser desenhados

```
x = np.linspace(-3, 3, 91)
t = np.linspace(1, 25, 30)
X2, T2 = np.meshgrid(x, t)

sinT2 = np.sin(2*np.pi*T2/T2.max())
F = 0.9*sinT2*np.sinc(X2*(1 + sinT2))
```

F é um array 2D que tem dados arbitrários para serem desenhados. Neste caso não são arbitrários, sendo definidos de forma a produzirem um gráfico cíclico perfeito.

Animação passo a passo

Passo 4: desenhar a primeira linha

```
line = ax.plot(x, F[0, :], color='k', lw=2)[0]
```

Esta linha de código define uma primeira linha com os atributos pretendidos (cor e grossura da linha). Reparem no [0] no fim. Isto é necessário porque o comando plot retorna uma lista de linhas. Aqui pretendemos apenas desenhar uma única linha, daí que indexemos apenas a primeira (i.e., zeroth) na lista de linhas.

Passo 5: criar uma função para actualizar a linha

```
def animate(i):
    line.set_ydata(F[i, :])
```

Esta função recebe um argumento. O único comando desta função altera as coordenadas y da linha. Veremos mais tarde comandos para alterar outras coisas.

Animação passo a passo

Passo seis: chamar a FuncAnimation e exibir

```
anim = FuncAnimation(
    fig, animate, interval=100, frames=len(t)-1)

plt.draw()
plt.show()
```

A FuncAnimation recebe dois argumentos: o objecto da figura ‘fig’ e a função de animação ‘animate’. O argumento de intervalo é opcional The interval argument is opcional e define o intervalo entre frames em mili-segundos. O argumento de frames é apenas necessário se exportarmos a animação. As duas últimas linhas são ou não necessárias dependendo The frames argument is needed only if the animation is to be exported.

Passo opcional: exportar a animação

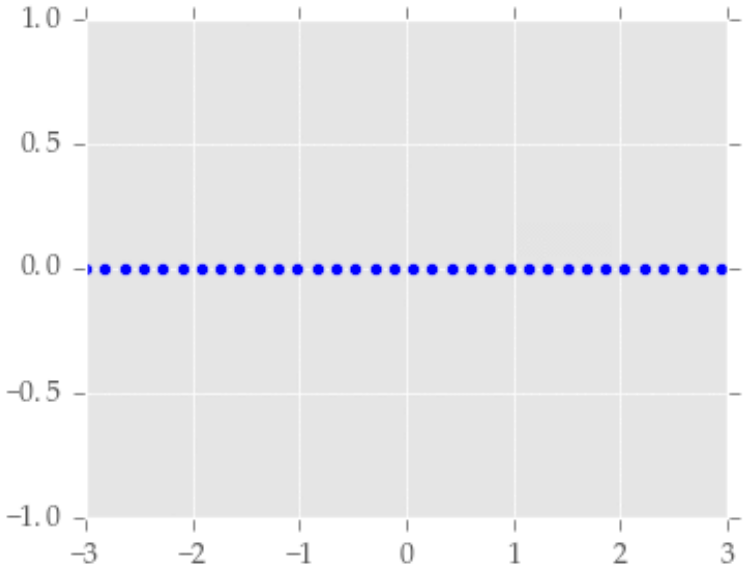
Para exportar a animação, por exemplo para um video em formato mpeg, basta executar o comando:

```
anim.save('filename.mp4')
```

Ou para exportar para um gif animado:

```
anim.save('filename.gif', writer='imagemagick')
```

Animação scatter



Animação scatter

Scatter

```
ax.set(xlim=(-3, 3), ylim=(-1, 1))
scat = ax.scatter(x[::3], F[0, ::3])

def animate(i):
    y_i = F[i, ::3]
    scat.set_offsets(np.c_[x[::3], y_i])
```

Notem que o set_offsets deve ser passado como um array N×2. Aqui usamos np.c_[] para esse fim. O uso de [::3] reduz a densidade dos pontos scatter.

Código scatter completo

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

def animate(i):
    y_i = F[i]
    scat.set_offsets(np.c_[x, y_i])


fig, ax = plt.subplots(figsize=(5, 3))
ax.set(xlim=(-3, 3), ylim=(-1, 1))

x = np.linspace(-3, 3, 9)
t = np.linspace(1, 25, 30)
X2, T2 = np.meshgrid(x, t)

sinT2 = np.sin(2*np.pi*T2/T2.max())
F = 0.9*sinT2*np.sinc(X2*(1 + sinT2))

scat = ax.scatter(x, F[0])

anim = FuncAnimation(
    fig, animate, interval=100, frames=len(t)-1)

plt.draw()
plt.show()
```

Animação de um trajecto de um táxi

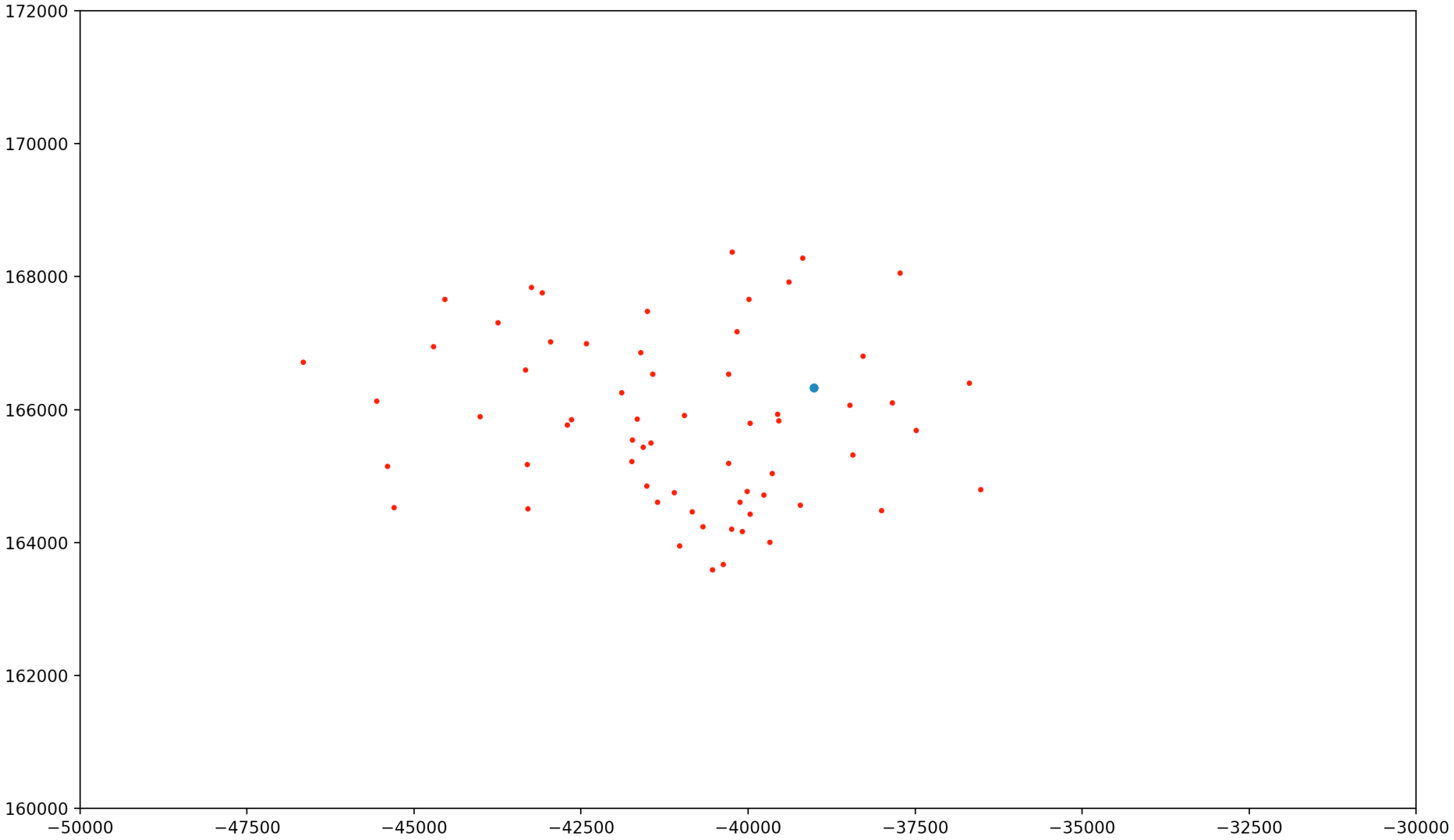
```
import numpy as np
import matplotlib.pyplot as plt
import psycopg2
import math
from matplotlib.animation import FuncAnimation

def animate(i):
    scat.set_offsets([x[i],y[i]])

def linestring_to_points(line_string):
    xs, ys = [],[]
    points = line_string[11:-1].split(',')
    for point in points:
        (x,y) = point.split()
        xs.append(float(x))
        ys.append(float(y))
    return xs,ys

scale=1/60000
conn = psycopg2.connect("dbname=michelferreira user=michelferreira")
cursor_psql = conn.cursor()
```

(continua)…

Animação de um trajecto de um táxi

```
(continuação)…

xs_min, xs_max, ys_min, ys_max = -50000, -30000, 160000, 172000
width_in_inches = (xs_max-xs_min)/0.0254*1.1
height_in_inches = (ys_max-ys_min)/0.0254*1.1

fig, ax = plt.subplots(figsize=(width_in_inches*scale, height_in_inches*scale))
ax.set(xlim=(xs_min, xs_max), ylim=(ys_min, ys_max))

sql = """select st_astext(proj_track) from tracks where taxi='20000333' order by ts"""
cursor_psql.execute(sql)
results = cursor_psql.fetchall()

x, y = [], []
for row in results:
    xs, ys = linestring_to_points(row[0])
    x = x + xs
    y = y + ys

scat = ax.scatter(x[0],y[0],s=10)
anim = FuncAnimation(
    fig, animate, interval=1, frames=len(y)-1)

plt.draw()
plt.show()
conn.close()
```

Animação de trajectos



Listagem de métodos set_... para as funções de animação

```
>>> scat = ax.scatter(x,y,s=10)
>>> [x for x in dir(scat) if 'set_' in x]
['_offset_position', '_set_edgecolor', '_set_facecolor', '_set_gc_clip', 'get_offset_position', 'get_offset_transform',
'set_aa', 'set_agg_filter', 'set_alpha', 'set_animated', 'set_antialiased', 'set_antialiaseds', 'set_array',
'set_capstyle', 'set_clim', 'set_clip_box', 'set_clip_on', 'set_clip_path', 'set_cmap', 'set_color', 'set_contains',
'set_dashes', 'set_ec', 'set_edgecolor', 'set_edgecolors', 'set_facecolor', 'set_facecolors', 'set_fc', 'set_figure',
'set_gid', 'set_hatch', 'set_in_layout', 'set_joinstyle', 'set_label', 'set_linestyle', 'set_linestyles',
'set_linewidth', 'set_linewidths', 'set_ls', 'set_lw', 'set_norm', 'set_offset_position', 'set_offsets',
'set_path_effects', 'set_paths', 'set_picker', 'set_pickradius', 'set_rasterized', 'set_sizes', 'set_sketch_params',
'set_snap', 'set_transform', 'set_url', 'set_urls', 'set_visible', 'set_zorder']
```

```
def animate(i):
    if i%2 == 0:
        scat.set_facecolor('red')
    else:
        scat.set_facecolor('green')
    scat.set_offsets([x[i],y[i]])
```

Exercícios com animação de dados

1. Alterar a animação dos trajectos de um determinado táxi, usando uma cor diferente em função do estado associado ao trajecto (BUSY - vermelho, FREE - verde, PICKUP - amarelo, PAUSE - cinzento). Mostrar ainda a hora e a distância percorrida;

2. Mostrar a animação simultânea dos trajectos de dois determinados táxis (mantendo a sincronização de tempo nos trajectos de ambos);

3. Mostrar a animação simultânea de todos os trajectos de todos os táxis que atravessam uma freguesia do concelho do Porto.

Trabalho de avaliação

- A análise da mobilidade das pessoas tem estado em grande debate como forma de permitir um combate mais eficaz à propagação da doença COVID-19. Na China, foi possível recorrer a dados de localização recolhidos e transmitidos por smartphones para analisar a mobilidade das pessoas e dos contactos que existiram. Dessa forma foi possível desenhar modelos probabilísticos de infecção que direccionaram os testes para pessoas com maior probabilidade de estarem infectadas.

- Sendo que o vírus da COVID-19 não se move autonomamente, é a mobilidade das pessoas que o propaga. Analisar essa mobilidade é portanto muito importante para analisar e conter a propagação do vírus.

- As bases de dados espaciais e ferramentas de visualização como o módulo matplotlib do python permitem efectuar a análise e visualização de tal mobilidade.

- O que se pretende com o este trabalho de avaliação é uma análise de propagação epidemiológica baseada nos trajectos da tabela tracks e a visualização recorrendo às animações do matplotlib de tal propagação.

- Pretende-se a representação do mapa de Portugal continental e a animação scatter da mobilidade de todos os táxis no dataset da tabela tracks. Os pontos móveis são representados em duas cores (verde e vermelho), representando o facto de não estar contaminado ou de estar contaminado, respectivamente. Inicialmente são definidos dois táxis como pontos iniciais de infecção, um aleatoriamente seleccionado dos primeiros 10 táxis a surgirem no concelho de Lisboa e outro dos primeiros 10 táxis a surgirem no concelho do Porto.

- A propagação da contaminação segue uma probabilidade e um critério de distância: por exemplo, sempre que um táxi esteja a menos de 50 metros de um táxi contaminado, a probalidade de ficar infectado é de 10% a cada minuto de tal proximidade.

- A simulação da propagação da contaminação vai pintando os pontos contaminados, ao mesmo tempo que desenha um gráfico da evolução do número de contaminações, que deverá seguir uma curva exponencial, que todos conhecemos bem por estes dias.

- Existem várias possibilidades para a visualização da animação, sendo aqui que é dada larga liberdade para apresentarem a simulação e análise de mobilidade da forma que pretenderem. Por exemplo, usar níveis de escala variáveis para a exibição do mapa; representar polígonos convexos que envolvam todos os infectados, etc.