1-MAP_performance_surface

May 22, 2024

[1]:
```python
import os
import itertools
import pathos
import functools
from functools import partial

import numpy as np
from scipy import stats as st
from sklearn.feature_selection import mutual_info_regression as MI

import plotly.graph_objects as go
from matplotlib import pyplot as plt
import seaborn as sns
sns.set()

import sys
sys.path.append('/home/leonardo/git/bayesian_bss')
from src import MAPGradientEstimator, InstantaneousMixtureModel

np.random.seed(2000)
```

[2]:
```python
# Number of sources and observations
NSOURCES=2
NOBS=10000

# Define initial conditions for optimizations
initial_B = np.random.normal(
    0,1,
    (NSOURCES, NSOURCES, 1)
)

# Learning rate
learning_rate=1E-3

# Optimization configs
thresh = 0.0005
max_it=100000
```

1 ENSAIO - CURVAS DE DESEMPENHO MAP

Objetivos:

- Verificar o efeito da especificação da distribuição a priori bayesiana (boa ou ruim) sobre ponto de convergência de estimação de matriz de separação;

- Verificar o efeito da especificação da distribuição das fontes (boa ou ruim) sobre ponto de convergência de estimação de matriz de separação;

Para tal:

- Constroem-se gráficos da vizinhança estimada do ponto ótimo (verdadeiro), seguindo exercício feito em Cardoso (1998)

- Realizam-se otimizações com o objetivo de determinar a matriz de separação, simulando um cenário de separação de fontes. Nas otimizações, a condição inicial é colocada próxima à verdadeira, para que, a partir de uma convergência facilitada, possa ser determinado o ponto global de máximo nos vários cenários testados.

2 Função a ser otimizada (probabilidade a posteriori)

De Djafari, 2000

$$\log P(B|y) = T \log |det(B)| + \sum_t \sum_i \log p_i(y_i(t)) + \log P(B) + cte$$

3 Equação de otimização (derivada)

De Taylor, 2008

$$\Delta B \propto (B^{-1})^T + \frac{1}{T} \sum_T g(Bx)x^T + \frac{1}{T} h(B)$$

4 1. PERFECT MODEL

4.1 1.1. Initialize Sources

[3]:
```
def source_cumulative(x):
    return 1/(1+np.exp(-x))

def source_pdf(x):
    return np.exp(-x)/np.square(1+np.exp(-x))

def source_generator(
    nsources,
    nobs,
    mu
):
```

```
    return np.random.logistic(
        loc=mu,
        scale=1,
        size=(nsources, nobs)
    )
```

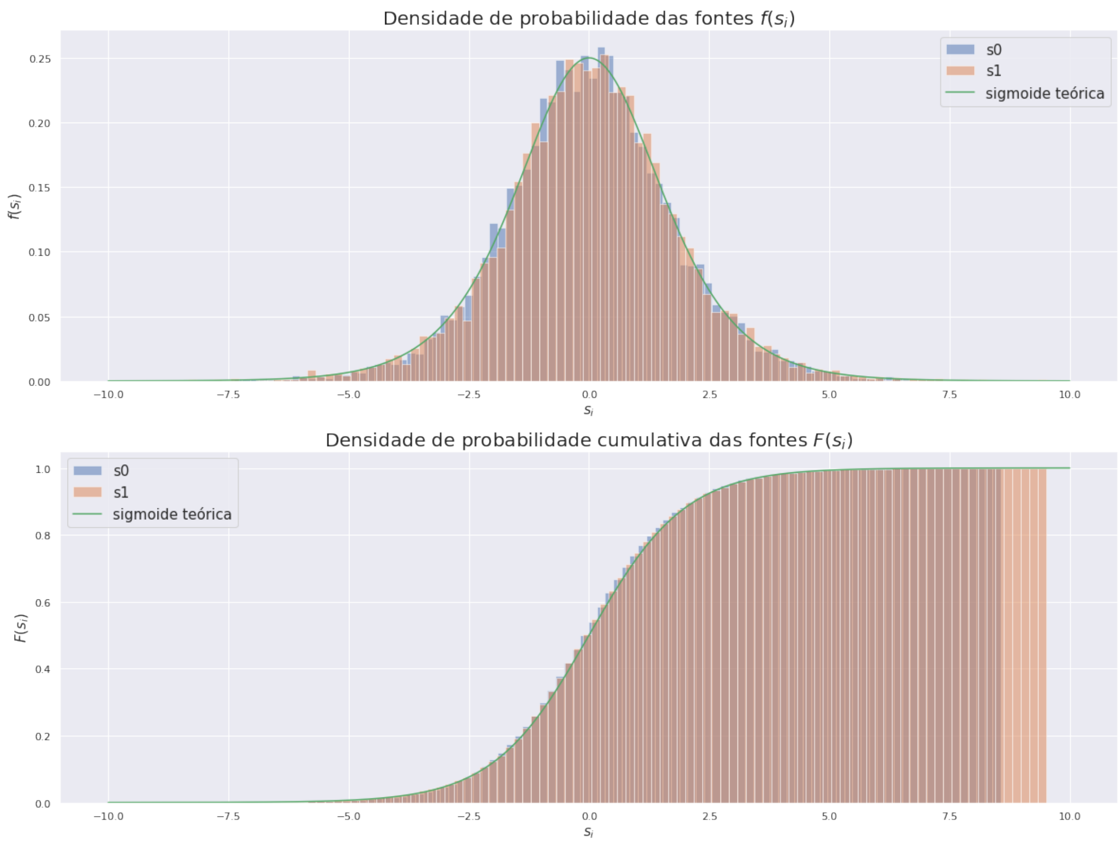
[4]:
```
MU=0.0

s = source_generator(
    nsources=NSOURCES,
    nobs=NOBS,
    mu=MU
)

print('-'*100)
print('NÚMERO DE FONTES: {}'.format(NSOURCES))
print('TAMANHO DOS SINAIS DAS FONTES: {}'.format(NOBS))
for i in range(s.shape[0]):
    print(
        'CURTOSE s{}: {}'.format(
            i,
            st.kurtosis(a=s[i,:])
        )
    )

print('-'*100)
```

```
--------------------------------------------------------------------------------
--------------------
NÚMERO DE FONTES: 2
TAMANHO DOS SINAIS DAS FONTES: 10000
CURTOSE s0: 0.9273716006264192
CURTOSE s1: 1.185617783891403
--------------------------------------------------------------------------------
--------------------
```

[5]:
```
fig, (ax1, ax2) = plt.subplots(
    nrows=2, ncols=1,
    figsize=(20,15)
)


for i in range(1, NSOURCES+1):
    ax1.hist(
        x=s[i-1,:],
        bins=100,
        density=True,
```

```
        label='s{}'.format(i-1),
        alpha=0.5
    )
    ax2.hist(
        x=s[i-1,:],
        bins=100,
        cumulative=True,
        density=True,
        label='s{}'.format(i-1),
        alpha=0.5
    )

ax1.plot(
    np.linspace(-10,10,1000),
    [source_pdf(x) for x in np.linspace(-10,10,1000)],
    label='sigmoide teórica'
)



ax2.plot(
    np.linspace(-10,10,1000),
    [source_cumulative(x) for x in np.linspace(-10,10,1000)],
    label='sigmoide teórica'
)

ax1.set_xlabel(
    '$s_{i}$',
    fontsize=15
)
ax1.set_ylabel(
    '$f(s_{i})$',
    fontsize=15
)
ax1.set_title(
    'Densidade de probabilidade das fontes $f(s_{i})$',
    fontsize=20
)

ax2.set_xlabel(
    '$s_{i}$',
    fontsize=15
)
ax2.set_ylabel(
    '$F(s_{i})$',
    fontsize=15
)
ax2.set_title(
```

```
    'Densidade de probabilidade cumulativa das fontes $F(s_{i})$',
    fontsize=20
)

l1=ax1.legend(fontsize=15)
l2=ax2.legend(fontsize=15)
```



4.2 1.2. Mix sources and generate observations

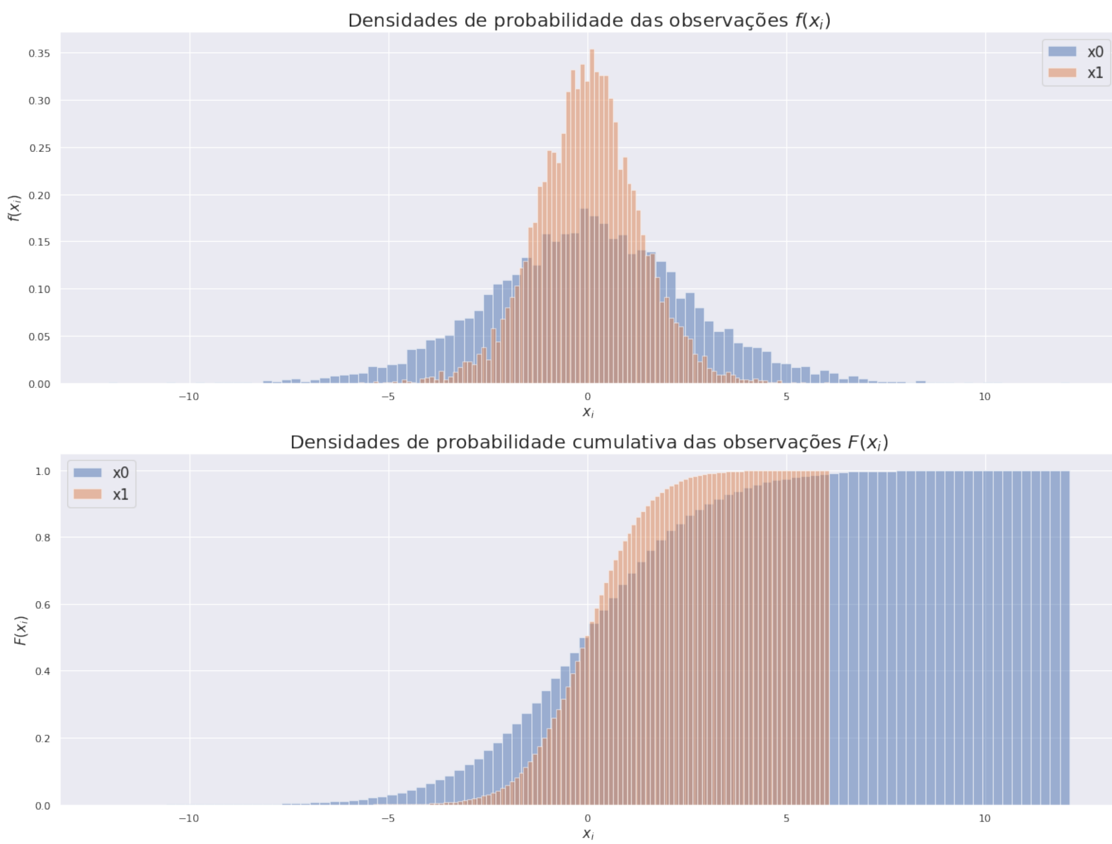
[6]:
```
# Mixing matrix
A = np.array([
    [1, 1],
    [-0.5, 0.5]
])
print('MIXING MATRIX A:')
print(A)
```

```
MIXING MATRIX A:
[[ 1.   1. ]
 [-0.5  0.5]]
```

[7]:
```
initial_B = np.linalg.inv(
    A
) + np.random.normal(
    0,0.1,
    A.shape
)
initial_B = initial_B.reshape(
    (NSOURCES, NSOURCES, 1)
)
```

[8]:
```
# Observed sources
x = A@s

fig, (ax1, ax2) = plt.subplots(
    nrows=2, ncols=1,
    figsize=(20,15)
)

for i in range(1, NSOURCES+1):
    ax1.hist(
        x=x[i-1,:],
        bins=100,
        density=True,
        label='x{}'.format(i-1),
        alpha=0.5
    )
    ax2.hist(
        x=x[i-1,:],
        bins=100,
        cumulative=True,
        density=True,
        label='x{}'.format(i-1),
        alpha=0.5
    )

ax1.set_xlabel(
    '$x_{i}$',
    fontsize=15
)
ax1.set_ylabel(
    '$f(x_{i})$',
    fontsize=15
)
ax1.set_title(
    'Densidades de probabilidade das observações $f(x_{i})$',
    fontsize=20
)
```

```
ax2.set_xlabel(
    '$x_{i}$',
    fontsize=15
)
ax2.set_ylabel(
    '$F(x_{i})$',
    fontsize=15
)
ax2.set_title(
    'Densidades de probabilidade cumulativa das observações $F(x_{i})$',
    fontsize=20
)

l1=ax1.legend(fontsize=15)
l2=ax2.legend(fontsize=15)
```



4.3 1.3. Performance Curves

[9]:
```
def get_posteriori_neighborhood(
    A,
    x,
    T,
    u_vec,
    v_vec,
    n_points,
    source_pdf_fn,
    prior_pdf_fn,
    njobs=1
):

    def __get_posteriori(
        A,
        x,
        T,
        source_pdf_fn,
        prior_pdf_fn,
        idx_info
    ):
        i, u = idx_info[0]
        j, v = idx_info[-1]

        A_shifted = A + u*symmetric_basis + v*skew_symmetric_basis

        B = np.linalg.inv(A_shifted)

        likelihood = np.log(np.abs(np.linalg.det(B)))

        likelihood_iterator = [
            (t,i) for t,i in itertools.product(
                range(x.shape[-1]),
                range(x.shape[0]))
        ]

        s_est = B@x

        likelihood += (1/T)*np.sum([
            np.log(source_pdf_fn(s_est[i,t])) for t,i in likelihood_iterator
        ])

        prior = np.log(prior_pdf_fn(B))/T

        return {
            'i': i,
            'j': j,
```

```
        'log_posteriori': likelihood + prior
    }


symmetric_basis = np.array([
    [0, 1],
    [1, 0]
])

skew_symmetric_basis = np.array([
    [0, -1],
    [1, 0]
])

iterator = itertools.product(
    enumerate(u_vec),
    enumerate(v_vec)
)
# it = [x for x in iterator]

exec_fn = partial(
    __get_posteriori,
    A,
    x,
    T,
    source_pdf_fn,
    prior_pdf_fn
)

z = np.empty(
    shape=(
        u_vec.shape[0],
        v_vec.shape[0]
    )
)
with pathos.multiprocessing.ProcessingPool(njobs) as p:
    results = p.map(exec_fn, iterator)

for r in results:
    i=r['i']
    j=r['j']
    z[i, j]=r['log_posteriori']

return z
```

4.3.1 1.3.1. Likelihood

Prior:

$p(B) = cte$

[10]:
```
def source_pdf(x):
    return np.exp(-x)/np.square(1+np.exp(-x))

def prior_pdf(B):
    return 1
```

[11]:
```
%%time

u_lims = (-0.05, 0.05)
v_lims = (-0.05, 0.05)
n_points = 251
central_point = [0, 0]

# Get step
u_step = (u_lims[-1] - u_lims[0])/(n_points-1)
v_step = (v_lims[-1] - v_lims[0])/(n_points-1)

# Step multipliers
step_multipliers = range(-(n_points-1)//2, (n_points-1)//2)

# Get u_vec and v_vec
u_vec = np.array([central_point[0] + s*u_step for s in step_multipliers])
v_vec = np.array([central_point[-1] + s*v_step for s in step_multipliers])

z = get_posteriori_neighborhood(
    A=A,
    x=x,
    T=NOBS,
    u_vec=u_vec,
    v_vec=v_vec,
    n_points=n_points,
    source_pdf_fn=source_pdf,
    prior_pdf_fn=prior_pdf,
    njobs=os.cpu_count()-1
)
```

```
CPU times: user 1.77 s, sys: 114 ms, total: 1.88 s
Wall time: 18min 20s
```

```
/tmp/ipykernel_45077/2236618269.py:42: RuntimeWarning: divide by zero
encountered in log
  prior = np.log(prior_pdf_fn(B))/T
/tmp/ipykernel_45077/2236618269.py:42: RuntimeWarning: divide by zero
```

```
encountered in log
  prior = np.log(prior_pdf_fn(B))/T
/tmp/ipykernel_45077/2236618269.py:42: RuntimeWarning: divide by zero
encountered in log
  prior = np.log(prior_pdf_fn(B))/T
/tmp/ipykernel_45077/2236618269.py:42: RuntimeWarning: divide by zero
encountered in log
  prior = np.log(prior_pdf_fn(B))/T
/tmp/ipykernel_45077/2236618269.py:42: RuntimeWarning: divide by zero
encountered in log
  prior = np.log(prior_pdf_fn(B))/T
/tmp/ipykernel_45077/2236618269.py:42: RuntimeWarning: divide by zero
encountered in log
  prior = np.log(prior_pdf_fn(B))/T
/tmp/ipykernel_45077/2236618269.py:42: RuntimeWarning: divide by zero
encountered in log
  prior = np.log(prior_pdf_fn(B))/T
/tmp/ipykernel_45077/2236618269.py:42: RuntimeWarning: divide by zero
encountered in log
  prior = np.log(prior_pdf_fn(B))/T
/tmp/ipykernel_45077/2236618269.py:42: RuntimeWarning: divide by zero
encountered in log
  prior = np.log(prior_pdf_fn(B))/T
/tmp/ipykernel_45077/2236618269.py:42: RuntimeWarning: divide by zero
encountered in log
  prior = np.log(prior_pdf_fn(B))/T
/tmp/ipykernel_45077/2236618269.py:39: RuntimeWarning: divide by zero
encountered in log
  np.log(source_pdf_fn(s_est[i,t])) for t,i in likelihood_iterator
/tmp/ipykernel_45077/3229642154.py:2: RuntimeWarning: overflow encountered in
square
  return np.exp(-x)/np.square(1+np.exp(-x))
/tmp/ipykernel_45077/2236618269.py:39: RuntimeWarning: divide by zero
encountered in log
  np.log(source_pdf_fn(s_est[i,t])) for t,i in likelihood_iterator
/tmp/ipykernel_45077/3229642154.py:2: RuntimeWarning: overflow encountered in
exp
  return np.exp(-x)/np.square(1+np.exp(-x))
/tmp/ipykernel_45077/3229642154.py:2: RuntimeWarning: invalid value encountered
in double_scalars
  return np.exp(-x)/np.square(1+np.exp(-x))
/tmp/ipykernel_45077/3229642154.py:2: RuntimeWarning: overflow encountered in
square
  return np.exp(-x)/np.square(1+np.exp(-x))
/tmp/ipykernel_45077/2236618269.py:39: RuntimeWarning: divide by zero
encountered in log
  np.log(source_pdf_fn(s_est[i,t])) for t,i in likelihood_iterator
/tmp/ipykernel_45077/3229642154.py:2: RuntimeWarning: overflow encountered in
```

```
exp
  return np.exp(-x)/np.square(1+np.exp(-x))
/tmp/ipykernel_45077/3229642154.py:2: RuntimeWarning: invalid value encountered
in double_scalars
  return np.exp(-x)/np.square(1+np.exp(-x))
/tmp/ipykernel_45077/3229642154.py:2: RuntimeWarning: overflow encountered in
square
  return np.exp(-x)/np.square(1+np.exp(-x))
/tmp/ipykernel_45077/2236618269.py:39: RuntimeWarning: divide by zero
encountered in log
  np.log(source_pdf_fn(s_est[i,t])) for t,i in likelihood_iterator
/tmp/ipykernel_45077/3229642154.py:2: RuntimeWarning: overflow encountered in
exp
  return np.exp(-x)/np.square(1+np.exp(-x))
/tmp/ipykernel_45077/3229642154.py:2: RuntimeWarning: invalid value encountered
in double_scalars
  return np.exp(-x)/np.square(1+np.exp(-x))
/tmp/ipykernel_45077/3229642154.py:2: RuntimeWarning: overflow encountered in
square
  return np.exp(-x)/np.square(1+np.exp(-x))
/tmp/ipykernel_45077/2236618269.py:39: RuntimeWarning: divide by zero
encountered in log
  np.log(source_pdf_fn(s_est[i,t])) for t,i in likelihood_iterator
/tmp/ipykernel_45077/3229642154.py:2: RuntimeWarning: overflow encountered in
exp
  return np.exp(-x)/np.square(1+np.exp(-x))
/tmp/ipykernel_45077/3229642154.py:2: RuntimeWarning: invalid value encountered
in double_scalars
  return np.exp(-x)/np.square(1+np.exp(-x))
/tmp/ipykernel_45077/3229642154.py:2: RuntimeWarning: overflow encountered in
square
  return np.exp(-x)/np.square(1+np.exp(-x))
/tmp/ipykernel_45077/3229642154.py:2: RuntimeWarning: overflow encountered in
exp
  return np.exp(-x)/np.square(1+np.exp(-x))
/tmp/ipykernel_45077/3229642154.py:2: RuntimeWarning: invalid value encountered
in double_scalars
  return np.exp(-x)/np.square(1+np.exp(-x))
/tmp/ipykernel_45077/3229642154.py:2: RuntimeWarning: overflow encountered in
square
  return np.exp(-x)/np.square(1+np.exp(-x))
/tmp/ipykernel_45077/2236618269.py:39: RuntimeWarning: divide by zero
encountered in log
  np.log(source_pdf_fn(s_est[i,t])) for t,i in likelihood_iterator
/tmp/ipykernel_45077/3229642154.py:2: RuntimeWarning: overflow encountered in
exp
  return np.exp(-x)/np.square(1+np.exp(-x))
/tmp/ipykernel_45077/3229642154.py:2: RuntimeWarning: invalid value encountered
```

```
in double_scalars
  return np.exp(-x)/np.square(1+np.exp(-x))
/tmp/ipykernel_45077/3229642154.py:2: RuntimeWarning: overflow encountered in
square
  return np.exp(-x)/np.square(1+np.exp(-x))
/tmp/ipykernel_45077/2236618269.py:39: RuntimeWarning: divide by zero
encountered in log
  np.log(source_pdf_fn(s_est[i,t])) for t,i in likelihood_iterator
/tmp/ipykernel_45077/3229642154.py:2: RuntimeWarning: overflow encountered in
exp
  return np.exp(-x)/np.square(1+np.exp(-x))
/tmp/ipykernel_45077/3229642154.py:2: RuntimeWarning: invalid value encountered
in double_scalars
  return np.exp(-x)/np.square(1+np.exp(-x))
/tmp/ipykernel_45077/3229642154.py:2: RuntimeWarning: overflow encountered in
square
  return np.exp(-x)/np.square(1+np.exp(-x))
/tmp/ipykernel_45077/2236618269.py:39: RuntimeWarning: divide by zero
encountered in log
  np.log(source_pdf_fn(s_est[i,t])) for t,i in likelihood_iterator
/tmp/ipykernel_45077/3229642154.py:2: RuntimeWarning: overflow encountered in
exp
  return np.exp(-x)/np.square(1+np.exp(-x))
/tmp/ipykernel_45077/3229642154.py:2: RuntimeWarning: invalid value encountered
in double_scalars
  return np.exp(-x)/np.square(1+np.exp(-x))
/tmp/ipykernel_45077/3229642154.py:2: RuntimeWarning: overflow encountered in
square
  return np.exp(-x)/np.square(1+np.exp(-x))
/tmp/ipykernel_45077/2236618269.py:39: RuntimeWarning: divide by zero
encountered in log
  np.log(source_pdf_fn(s_est[i,t])) for t,i in likelihood_iterator
/tmp/ipykernel_45077/3229642154.py:2: RuntimeWarning: overflow encountered in
exp
  return np.exp(-x)/np.square(1+np.exp(-x))
/tmp/ipykernel_45077/3229642154.py:2: RuntimeWarning: invalid value encountered
in double_scalars
  return np.exp(-x)/np.square(1+np.exp(-x))
/tmp/ipykernel_45077/3229642154.py:2: RuntimeWarning: overflow encountered in
square
  return np.exp(-x)/np.square(1+np.exp(-x))
/tmp/ipykernel_45077/2236618269.py:39: RuntimeWarning: divide by zero
encountered in log
  np.log(source_pdf_fn(s_est[i,t])) for t,i in likelihood_iterator
/tmp/ipykernel_45077/3229642154.py:2: RuntimeWarning: overflow encountered in
exp
  return np.exp(-x)/np.square(1+np.exp(-x))
/tmp/ipykernel_45077/3229642154.py:2: RuntimeWarning: invalid value encountered
```

```
in double_scalars
  return np.exp(-x)/np.square(1+np.exp(-x))
/tmp/ipykernel_45077/3229642154.py:2: RuntimeWarning: overflow encountered in
square
  return np.exp(-x)/np.square(1+np.exp(-x))
/tmp/ipykernel_45077/2236618269.py:39: RuntimeWarning: divide by zero
encountered in log
  np.log(source_pdf_fn(s_est[i,t])) for t,i in likelihood_iterator
/tmp/ipykernel_45077/3229642154.py:2: RuntimeWarning: overflow encountered in
exp
  return np.exp(-x)/np.square(1+np.exp(-x))
/tmp/ipykernel_45077/3229642154.py:2: RuntimeWarning: invalid value encountered
in double_scalars
  return np.exp(-x)/np.square(1+np.exp(-x))
/tmp/ipykernel_45077/2236618269.py:42: RuntimeWarning: divide by zero
encountered in log
  prior = np.log(prior_pdf_fn(B))/T
```

[12]:

```
# Ponto de máximo
max_post_point = np.unravel_index(
    z.argmax(),
    z.shape
)

#
max_post = z[max_post_point]

print('-'*100)
print('Ponto de máximo: u={}, v={}'.format(u_vec[max_post_point[0]],␣
 ↪v_vec[max_post_point[1]]))
print('-'*100)
```

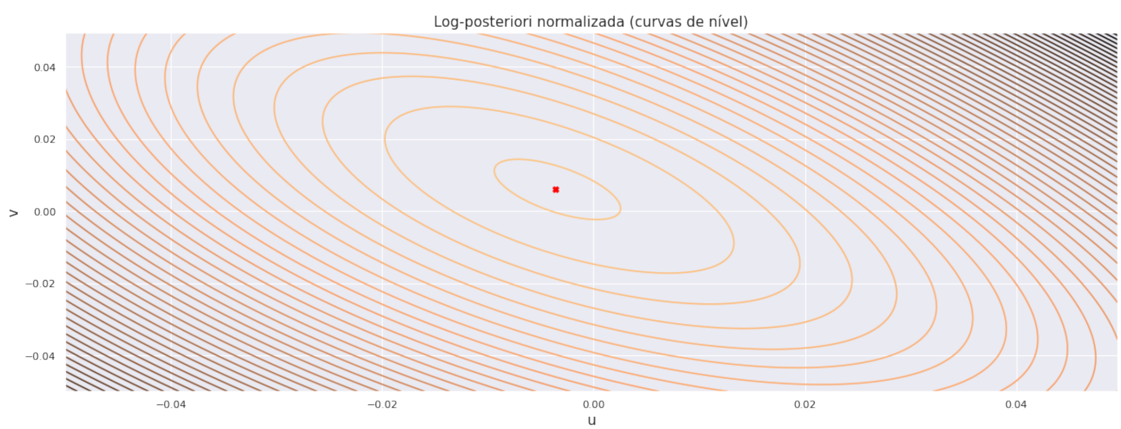
```
--------------------------------------------------------------------------------
--------------------
Ponto de máximo: u=-0.0036000000000000003, v=0.006
--------------------------------------------------------------------------------
--------------------
```

[13]:

```
# Plot de curvas de nível
fig = plt.figure(figsize=(20,7))

U, V = np.meshgrid(u_vec, v_vec)

plt.contour(
    U,
    V,
    z.T,
```

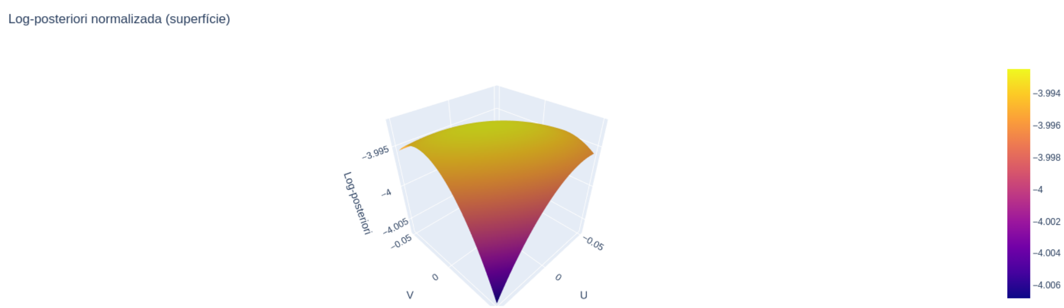
```
    levels=50,
    cmap='copper'
)

plt.scatter(
    u_vec[max_post_point[0]],
    v_vec[max_post_point[-1]],
    marker='X',
    color='red'
)

plt.xlabel(
    'u',
    fontsize=15
)
plt.ylabel(
    'v',
    fontsize=15
)

plt.title(
    'Log-posteriori normalizada (curvas de nível)',
    fontsize=15
)
```

[13]: Text(0.5, 1.0, 'Log-posteriori normalizada (curvas de nível)')



[14]:
```
# Plot interativo
fig = go.Figure(
    go.Surface(
        x=u_vec,
        y=v_vec,
```

```
        z=z
    )
)

fig.update_layout(
    title='Log-posteriori normalizada (superfície)',
    autosize=False,
    width=500, height=500,
    scene=dict(
        xaxis_title='U',
        yaxis_title='V',
        zaxis_title='Log-posteriori',
    )
)

fig.show()
```



[15]:
```
def source_pdf(x):
    """
        This method takes SCALAR value x and return SCALAR source pdf evaluated␣
 ↪at x.
    """
    return np.exp(-x)/np.square(1+np.exp(-x))


def source_pdf_derivative(x):
    """
        This method takes SCALAR value x and return SCALAR source pdf␣
 ↪derivative evaluated at x.
    """
    return (2*np.exp(-2*x)*(1+np.exp(-x)) - np.exp(-x)*np.square(1+np.exp(-x)))/
 ↪np.power([1+np.exp(-x)],[4])
```

```
def prior_pdf(X):
    """
        This method takes VECTOR value X and return SCALAR prior pdf evaluated␣
 ↪at X.
    """
    return 1

def prior_pdf_derivative(X):
    """
        This method takes VECTOR value x and return VECTOR prior pdf derivative␣
 ↪evaluated at X.
    """
    return np.zeros(
        shape=X.shape
    )
```

4.4 3.1. Standard Gradient

[16]:
```
# Initialize estimator
estimator = MAPGradientEstimator(
    learning_rate=learning_rate,
    thresh=thresh,
    max_it=max_it,
    source_pdf=source_pdf,
    source_pdf_derivative=source_pdf_derivative,
    prior_pdf=prior_pdf,
    prior_pdf_derivative=prior_pdf_derivative,
    is_natural_gradient=False
)
```

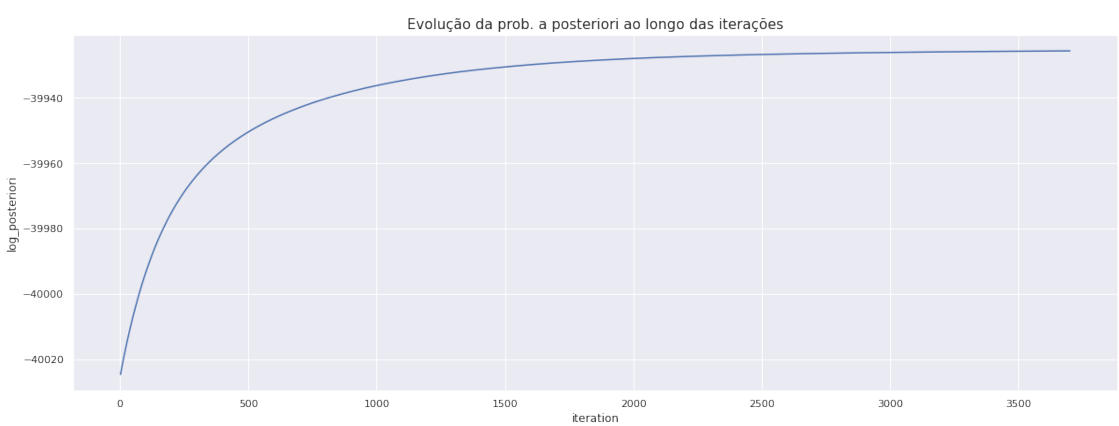
[17]:
```
%%time

# Run optimization
estimator.fit(
    s=s,
    x=x,
    initial_condition=initial_B,
    n_initializations=1,
    n_jobs=os.cpu_count()-1
)

# Parse results
B=estimator.fit_results[0]['unmixing_matrix']
logs=estimator.fit_results[0]['logs']

print('-'*100)
```

```
print('Total iterations: {}'.format(logs.shape[0]))
print('Final log-posteriori: {}'.format(logs.iloc[-1]['log_posteriori']))
print('-'*100)
```

```
/home/leonardo/git/bayesian_bss/src/estimator.py:144: FutureWarning:

elementwise comparison failed; returning scalar instead, but in the future will
perform elementwise comparison


--------------------------------------------------------------------------------
--------------------
Total iterations: 3700
Final log-posteriori: -39925.65231215294
--------------------------------------------------------------------------------
--------------------
CPU times: user 23min 31s, sys: 3min 21s, total: 26min 53s
Wall time: 20min 24s
```

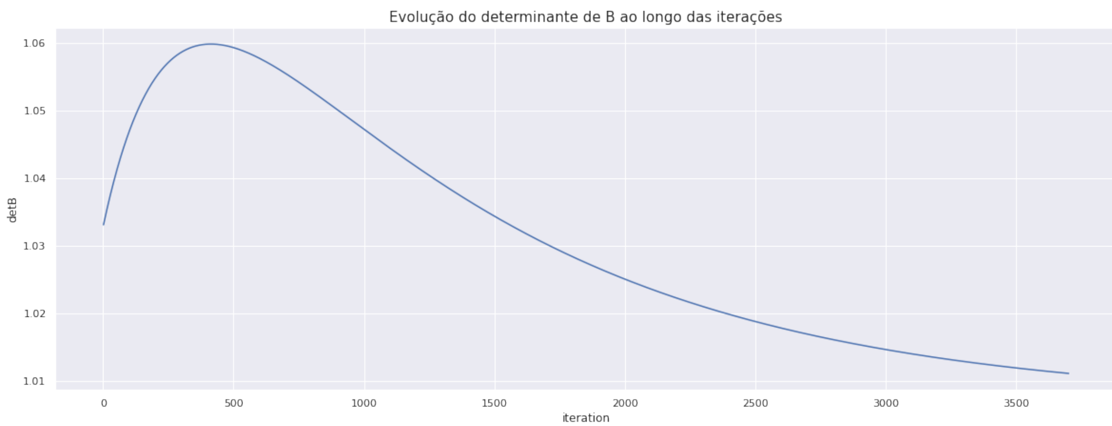
[18]:
```
fig = plt.figure(figsize=(20,7))
sns.lineplot(
    data=logs,
    x='iteration',
    y='log_posteriori'
)
plt.title(
    'Evolução da prob. a posteriori ao longo das iterações',
    fontsize=15
)
```

[18]:
```
Text(0.5, 1.0, 'Evolução da prob. a posteriori ao longo das iterações')
```



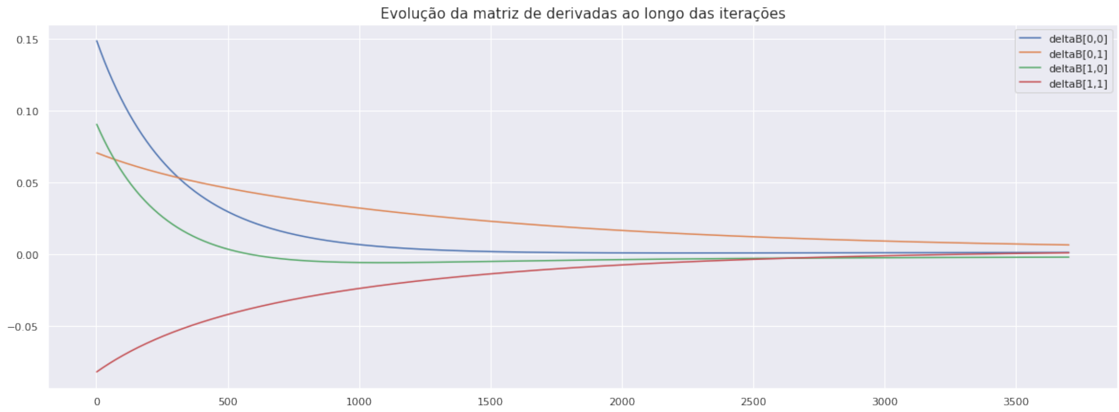
[19]:
```python
fig = plt.figure(figsize=(20,7))
sns.lineplot(
    data=logs,
    x='iteration',
    y='detB'
)
plt.title(
    'Evolução do determinante de B ao longo das iterações',
    fontsize=15
)
```

[19]: Text(0.5, 1.0, 'Evolução do determinante de B ao longo das iterações')



[20]:
```python
fig = plt.figure(figsize=(20,7))
for i, j in np.ndindex(B.shape):
    plt.plot(
        logs.iteration.values,
        [row['gradient'][i,j] for n, row in logs.iterrows()],
        label='deltaB[{},{}]'.format(i,j)
    )

plt.title(
    'Evolução da matriz de derivadas ao longo das iterações',
    fontsize=15
)
plt.legend()
```

[20]: <matplotlib.legend.Legend at 0x7f60e187c970>



[21]:
```
B
```

[21]:
```
array([[ 0.48601819, -1.0538953 ],
       [ 0.52403165,  0.94410418]])
```

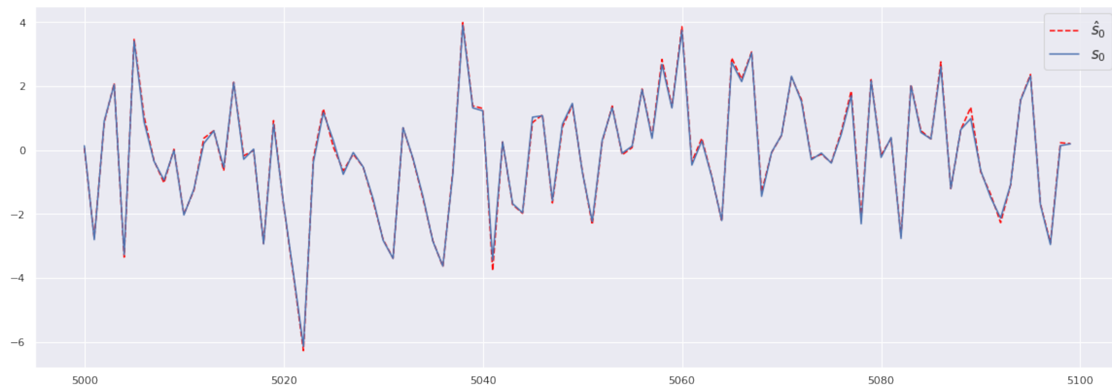
[22]:
```
np.linalg.inv(A)
```

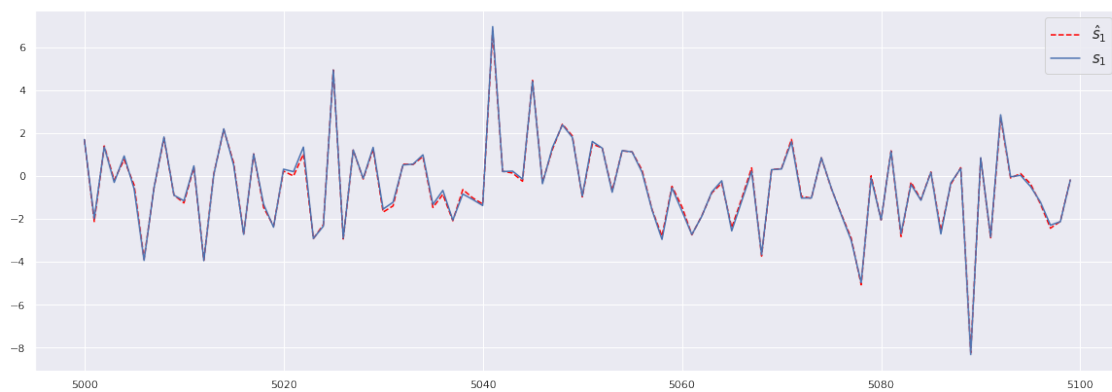
[22]:
```
array([[ 0.5, -1. ],
       [ 0.5,  1. ]])
```

[23]:
```
fig, (ax1, ax2) = plt.subplots(
    nrows=2, ncols=1,
    figsize=(20,15)
)
NPOINTS=100
PLOT_START=NOBS//2
PLOT_END=PLOT_START+NPOINTS

t=range(NOBS)
s_est = B@x

#Ax1
ax1.plot(
    t[PLOT_START:PLOT_END],
    s_est[0,PLOT_START:PLOT_END],
    label='$\hat{s}_{0}$',
    color='red',
    linestyle='--'
)
ax1.plot(
    t[PLOT_START:PLOT_END],
    s[0,PLOT_START:PLOT_END],
    label='$s_{0}$'
```

```
)
ax1.legend(fontsize=15)

#Ax2
ax2.plot(
    t[PLOT_START:PLOT_END],
    s_est[1,PLOT_START:PLOT_END],
    label='$\hat{s}_{1}$',
    color='red',
    linestyle='--'
)
ax2.plot(
    t[PLOT_START:PLOT_END],
    s[1,PLOT_START:PLOT_END],
    label='$s_{1}$'
)
ax2.legend(fontsize=15)
```

[23]: <matplotlib.legend.Legend at 0x7f60e13864c0>





[24]:
```python
# Error norm
print('Norma do erro de estimação: {}'.format(
np.linalg.norm(
    np.subtract(s,s_est)
)/np.size(s)))
```

```
Norma do erro de estimação: 0.0006090251577063119
```

4.4.1 1.3.2. MAP - determinant equal to one

Prior:

$p(B) \propto \exp\left[-\frac{1}{2\sigma^2}(det(B)-1)^2\right]$

[25]:
```python
def source_pdf(x):
    return np.exp(-x)/np.square(1+np.exp(-x))

def prior_pdf(B):
    sig=0.1
    desired_det=1
    return (1/np.sqrt(2*np.pi*np.square(sig)))*np.exp(-np.square(np.linalg.
↪det(B)-desired_det)/(2*np.square(sig)))
```

[26]:
```python
%%time

u_lims = (-0.05, 0.05)
v_lims = (-0.05, 0.05)
n_points = 251
central_point = [0, 0]

# Get step
u_step = (u_lims[-1] - u_lims[0])/(n_points-1)
v_step = (v_lims[-1] - v_lims[0])/(n_points-1)

# Step multipliers
step_multipliers = range(-(n_points-1)//2, (n_points-1)//2)

# Get u_vec and v_vec
u_vec = np.array([central_point[0] + s*u_step for s in step_multipliers])
v_vec = np.array([central_point[-1] + s*v_step for s in step_multipliers])

z = get_posteriori_neighborhood(
    A=A,
    x=x,
    T=NOBS,
    u_vec=u_vec,
    v_vec=v_vec,
    n_points=n_points,
```

```
    source_pdf_fn=source_pdf,
    prior_pdf_fn=prior_pdf,
    njobs=os.cpu_count()-1
)
```

```
CPU times: user 1.61 s, sys: 140 ms, total: 1.75 s
Wall time: 19min 46s
```

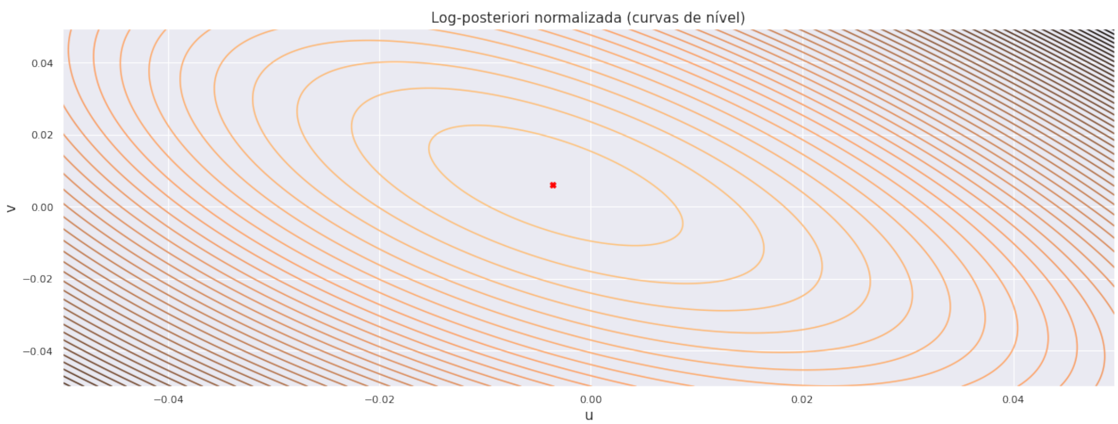
[27]:
```
# Ponto de máximo
max_post_point = np.unravel_index(
    z.argmax(),
    z.shape
)

#
max_post = z[max_post_point]

print('-'*100)
print('Ponto de máximo: u={}, v={}'.format(u_vec[max_post_point[0]],␣
 ↪v_vec[max_post_point[1]]))
print('-'*100)
```

```
--------------------------------------------------------------------------------
--------------------
Ponto de máximo: u=-0.0036000000000000003, v=0.006
--------------------------------------------------------------------------------
--------------------
```

[28]:
```
# Plot de curvas de nível
fig = plt.figure(figsize=(20,7))

U, V = np.meshgrid(u_vec, v_vec)

plt.contour(
    U,
    V,
    z.T,
    levels=50,
    cmap='copper'
)

plt.scatter(
    u_vec[max_post_point[0]],
    v_vec[max_post_point[-1]],
    marker='X',
    color='red'
)
```

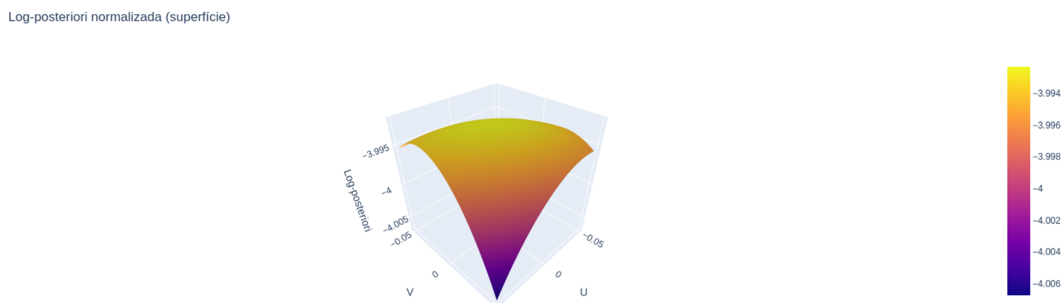
```python
plt.xlabel(
    'u',
    fontsize=15
)
plt.ylabel(
    'v',
    fontsize=15
)

plt.title(
    'Log-posteriori normalizada (curvas de nível)',
    fontsize=15
)
```

[28]: Text(0.5, 1.0, 'Log-posteriori normalizada (curvas de nível)')



[29]:
```python
# Plot interativo
fig = go.Figure(
    go.Surface(
        x=u_vec,
        y=v_vec,
        z=z
    )
)

fig.update_layout(
    title='Log-posteriori normalizada (superfície)',
    autosize=False,
    width=500, height=500,
    scene=dict(
        xaxis_title='U',
```

```
        yaxis_title='V',
        zaxis_title='Log-posteriori',
    )
)

fig.show()
```



[30]:
```
def source_pdf(x):
    """
        This method takes SCALAR value x and return SCALAR source pdf evaluated at x.
    """
    return np.exp(-x)/np.square(1+np.exp(-x))


def source_pdf_derivative(x):
    """
        This method takes SCALAR value x and return SCALAR source pdf derivative evaluated at x.
    """
    return (2*np.exp(-2*x)*(1+np.exp(-x)) - np.exp(-x)*np.square(1+np.exp(-x)))/np.power([1+np.exp(-x)],[4])

def prior_pdf(X):
    sig=0.1
    desired_det=1
    return (1/np.sqrt(2*np.pi*np.square(sig)))*np.exp(-np.square(np.linalg.det(X)-desired_det)/(2*np.square(sig)))

def prior_pdf_derivative(X):
    sig=0.1
    return -1*prior_pdf(X)*(np.linalg.det(X)-1)/np.square(sig)
```

4.5 3.1. Standard Gradient

[31]:
```
# Initialize estimator
estimator = MAPGradientEstimator(
    learning_rate=learning_rate,
    thresh=thresh,
    max_it=max_it,
    source_pdf=source_pdf,
    source_pdf_derivative=source_pdf_derivative,
    prior_pdf=prior_pdf,
    prior_pdf_derivative=prior_pdf_derivative,
    is_natural_gradient=False
)
```

[32]:
```
%%time

# Run optimization
estimator.fit(
    s=s,
    x=x,
    initial_condition=initial_B,
    n_initializations=1,
    n_jobs=os.cpu_count()-1
)

# Parse results
B=estimator.fit_results[0]['unmixing_matrix']
logs=estimator.fit_results[0]['logs']

print('-'*100)
print('Total iterations: {}'.format(logs.shape[0]))
print('Final log-posteriori: {}'.format(logs.iloc[-1]['log_posteriori']))
print('-'*100)
```

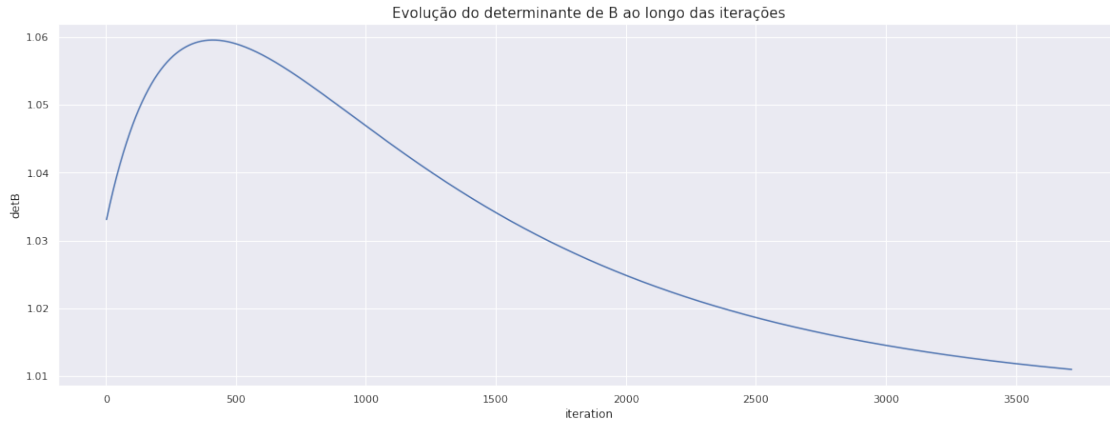
```
/home/leonardo/git/bayesian_bss/src/estimator.py:144: FutureWarning:

elementwise comparison failed; returning scalar instead, but in the future will
perform elementwise comparison


--------------------------------------------------------------------------------
--------------------
Total iterations: 3714
Final log-posteriori: -39924.34020971472
--------------------------------------------------------------------------------
--------------------
CPU times: user 26min 28s, sys: 3min 18s, total: 29min 46s
Wall time: 23min 20s
```

[33]:
```
fig = plt.figure(figsize=(20,7))
sns.lineplot(
    data=logs,
    x='iteration',
    y='log_posteriori'
)
plt.title(
    'Evolução da prob. a posteriori ao longo das iterações',
    fontsize=15
)
```

[33]: Text(0.5, 1.0, 'Evolução da prob. a posteriori ao longo das iterações')

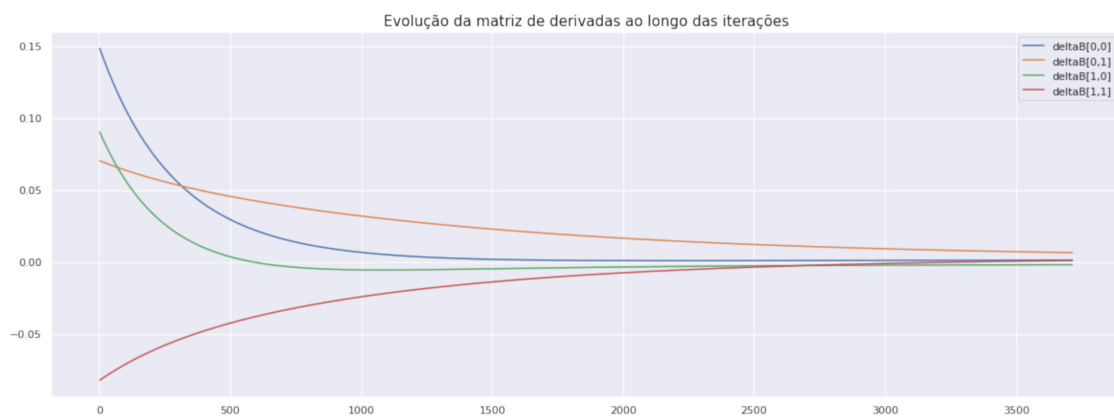
[34]:
```
fig = plt.figure(figsize=(20,7))
sns.lineplot(
    data=logs,
    x='iteration',
    y='detB'
)
plt.title(
    'Evolução do determinante de B ao longo das iterações',
    fontsize=15
)
```

[34]: Text(0.5, 1.0, 'Evolução do determinante de B ao longo das iterações')



[35]:
```
fig = plt.figure(figsize=(20,7))
for i, j in np.ndindex(B.shape):
    plt.plot(
        logs.iteration.values,
        [row['gradient'][i,j] for n, row in logs.iterrows()],
        label='deltaB[{},{}]'.format(i,j)
    )

plt.title(
    'Evolução da matriz de derivadas ao longo das iterações',
    fontsize=15
)
plt.legend()
```

[35]: <matplotlib.legend.Legend at 0x7f60e1069e20>



[36]:
```
B
```

[36]:
```
array([[ 0.4856303 , -1.05457859],
       [ 0.5243053 ,  0.94322969]])
```

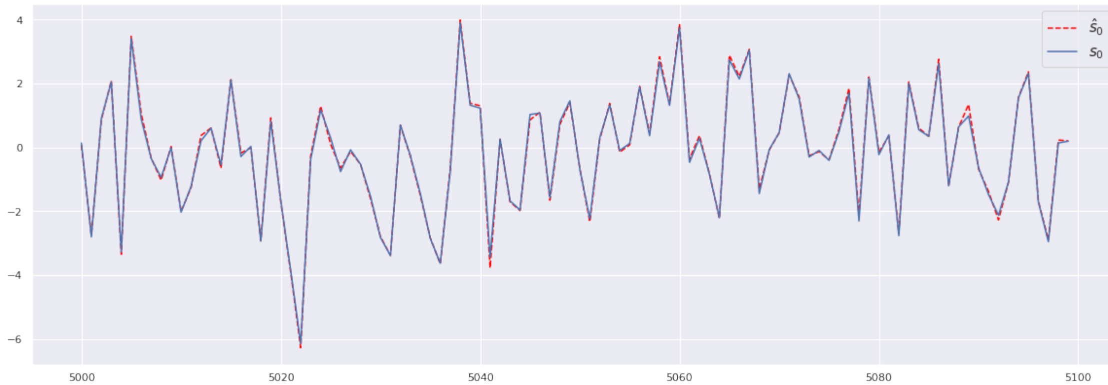
[37]:
```
np.linalg.inv(A)
```

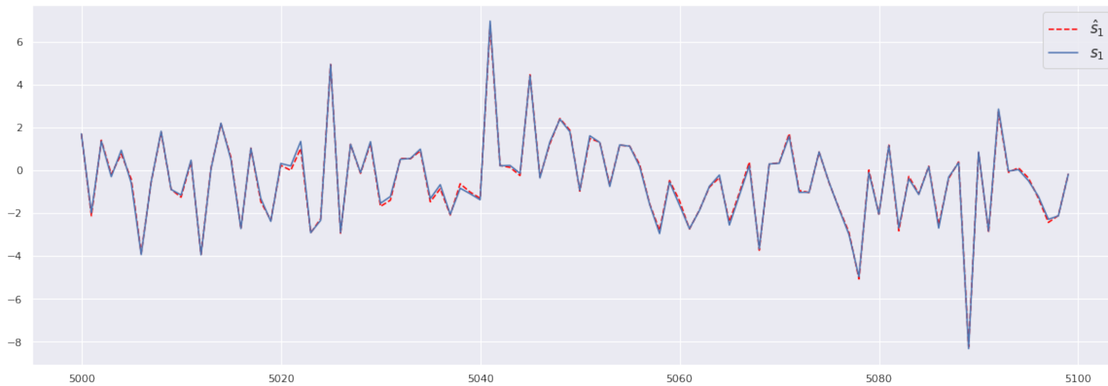
[37]:
```
array([[ 0.5, -1. ],
       [ 0.5,  1. ]])
```

[38]:
```
fig, (ax1, ax2) = plt.subplots(
    nrows=2, ncols=1,
    figsize=(20,15)
)
NPOINTS=100
PLOT_START=NOBS//2
PLOT_END=PLOT_START+NPOINTS

t=range(NOBS)
s_est = B@x

#Ax1
ax1.plot(
    t[PLOT_START:PLOT_END],
    s_est[0,PLOT_START:PLOT_END],
    label='$\hat{s}_{0}$',
    color='red',
    linestyle='--'
)
ax1.plot(
    t[PLOT_START:PLOT_END],
    s[0,PLOT_START:PLOT_END],
    label='$s_{0}$'
)
ax1.legend(fontsize=15)

#Ax2
ax2.plot(
    t[PLOT_START:PLOT_END],
    s_est[1,PLOT_START:PLOT_END],
    label='$\hat{s}_{1}$',
    color='red',
    linestyle='--'
)
ax2.plot(
    t[PLOT_START:PLOT_END],
    s[1,PLOT_START:PLOT_END],
    label='$s_{1}$'
)
```

```
ax2.legend(fontsize=15)
```

[38]: <matplotlib.legend.Legend at 0x7f60e16a8cd0>





[39]:
```
# SCATTER PLOT verdadeiro vs estimado

# Error norm
print('Norma do erro de estimação: {}'.format(
np.linalg.norm(
    np.subtract(s,s_est)
)/np.size(s)))
```

```
Norma do erro de estimação: 0.0006179860287488588
```

4.6 1.3.3. MAP - near-identity transformation

Prior:

$p(B) \propto \exp\left[-\frac{1}{2\sigma^2}||B-I||^2\right]$

[40]:
```
def source_pdf(x):
    return np.exp(-x)/np.square(1+np.exp(-x))
```

```python
def prior_pdf(X):
    sig=0.1
    return np.exp(
        (-1/2/np.square(sig))*np.linalg.norm(X-np.eye(X.shape[0]))
    )
```

[41]:
```python
%%time

u_lims = (-0.05, 0.05)
v_lims = (-0.05, 0.05)
n_points = 251
central_point = [0, 0]

# Get step
u_step = (u_lims[-1] - u_lims[0])/(n_points-1)
v_step = (v_lims[-1] - v_lims[0])/(n_points-1)

# Step multipliers
step_multipliers = range(-(n_points-1)//2, (n_points-1)//2)

# Get u_vec and v_vec
u_vec = np.array([central_point[0] + s*u_step for s in step_multipliers])
v_vec = np.array([central_point[-1] + s*v_step for s in step_multipliers])

z = get_posteriori_neighborhood(
    A=A,
    x=x,
    T=NOBS,
    u_vec=u_vec,
    v_vec=v_vec,
    n_points=n_points,
    source_pdf_fn=source_pdf,
    prior_pdf_fn=prior_pdf,
    njobs=os.cpu_count()-1
)
```

```
CPU times: user 1.81 s, sys: 176 ms, total: 1.99 s
Wall time: 20min 47s
```

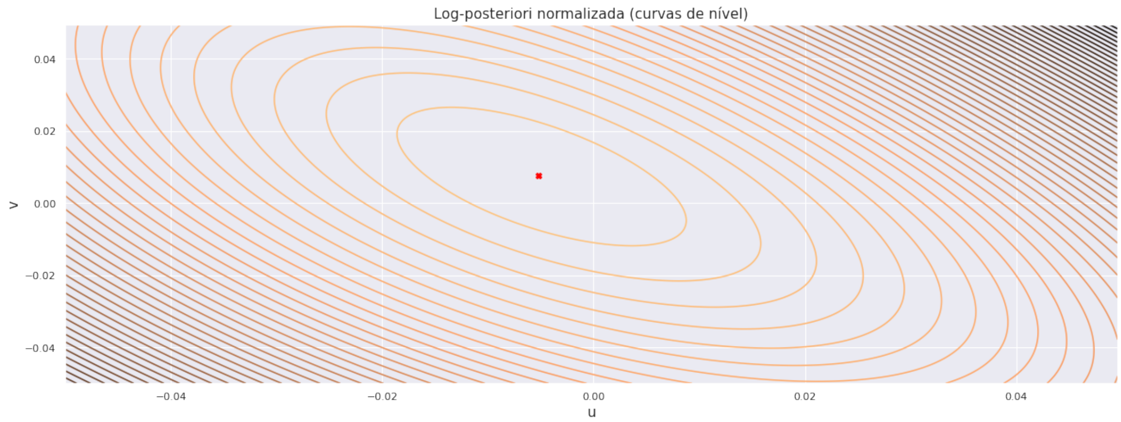
[42]:
```python
# Ponto de máximo
max_post_point = np.unravel_index(
    z.argmax(),
    z.shape
)

max_post = z[max_post_point]
```

```
print('-'*100)
print('Ponto de máximo: u={}, v={}'.format(u_vec[max_post_point[0]],␣
 ↪v_vec[max_post_point[1]]))
print('-'*100)
```

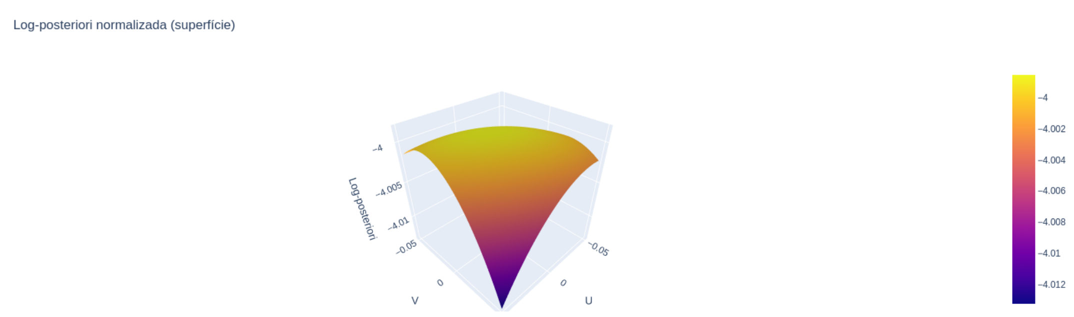
```
--------------------------------------------------------------------------------
--------------------
Ponto de máximo: u=-0.005200000000000001, v=0.0076
--------------------------------------------------------------------------------
--------------------
```

[43]:
```
# Plot de curvas de nível
fig = plt.figure(figsize=(20,7))

U, V = np.meshgrid(u_vec, v_vec)

plt.contour(
    U,
    V,
    z.T,
    levels=50,
    cmap='copper'
)

plt.scatter(
    u_vec[max_post_point[0]],
    v_vec[max_post_point[-1]],
    marker='X',
    color='red'
)

plt.xlabel(
    'u',
    fontsize=15
)
plt.ylabel(
    'v',
    fontsize=15
)

plt.title(
    'Log-posteriori normalizada (curvas de nível)',
    fontsize=15
)
```

[43]:
```
Text(0.5, 1.0, 'Log-posteriori normalizada (curvas de nível)')
```



[44]:
```
# Plot interativo
fig = go.Figure(
    go.Surface(
        x=u_vec,
        y=v_vec,
        z=z
    )
)

fig.update_layout(
    title='Log-posteriori normalizada (superfície)',
    autosize=False,
    width=500, height=500,
    scene=dict(
        xaxis_title='U',
        yaxis_title='V',
        zaxis_title='Log-posteriori',
    )
)

fig.show()
```



[45]:
```
def source_pdf(x):
    """
        This method takes SCALAR value x and return SCALAR source pdf evaluated␣
 ↪at x.
    """
    return np.exp(-x)/np.square(1+np.exp(-x))


def source_pdf_derivative(x):
    """
        This method takes SCALAR value x and return SCALAR source pdf␣
 ↪derivative evaluated at x.
    """
    return (2*np.exp(-2*x)*(1+np.exp(-x)) - np.exp(-x)*np.square(1+np.exp(-x)))/
 ↪np.power([1+np.exp(-x)],[4])

def prior_pdf(X):
    sig=0.1
    return np.exp(
        (-1/2/np.square(sig))*np.linalg.norm(X-np.eye(X.shape[0]))
    )

def prior_pdf_derivative(X):
    sig=0.1
    der_X=np.empty(X.shape)
    I=np.eye(X.shape[0])
    for i, j in np.ndindex(X.shape):
        der_X[i, j] = prior_pdf(X)*(-1/np.square(sig))*(X[i,j]-I[i,j])
    return der_X
```

4.7 3.1. Standard Gradient

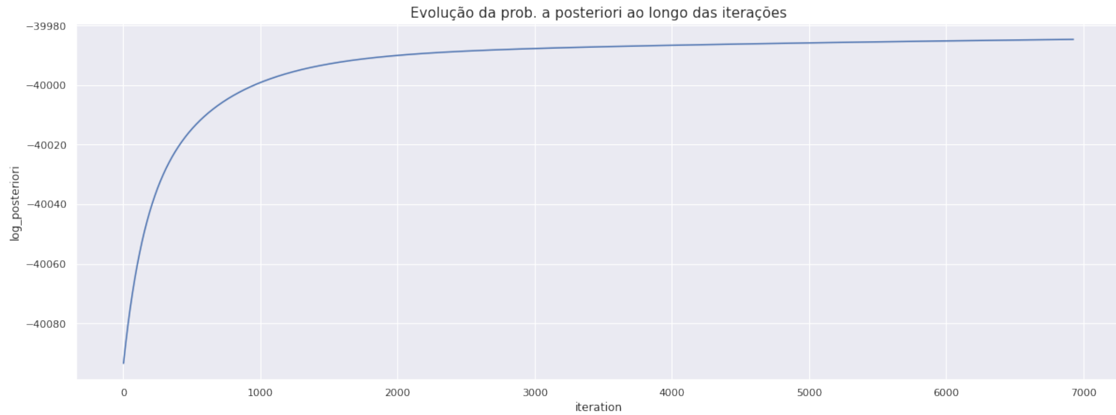
[46]:
```
# Initialize estimator
estimator = MAPGradientEstimator(
    learning_rate=learning_rate,
    thresh=thresh,
    max_it=max_it,
    source_pdf=source_pdf,
    source_pdf_derivative=source_pdf_derivative,
    prior_pdf=prior_pdf,
    prior_pdf_derivative=prior_pdf_derivative,
    is_natural_gradient=False
)
```

[47]:
```
%%time

# Run optimization
estimator.fit(
    s=s,
    x=x,
    initial_condition=initial_B,
    n_initializations=1,
    n_jobs=os.cpu_count()-1
)

# Parse results
B=estimator.fit_results[0]['unmixing_matrix']
logs=estimator.fit_results[0]['logs']

print('-'*100)
print('Total iterations: {}'.format(logs.shape[0]))
print('Final log-posteriori: {}'.format(logs.iloc[-1]['log_posteriori']))
print('-'*100)
```

```
/home/leonardo/git/bayesian_bss/src/estimator.py:144: FutureWarning:

elementwise comparison failed; returning scalar instead, but in the future will
perform elementwise comparison


--------------------------------------------------------------------------------
--------------------
Total iterations: 6926
Final log-posteriori: -39984.698086625445
--------------------------------------------------------------------------------
--------------------
CPU times: user 45min 34s, sys: 6min 14s, total: 51min 49s
Wall time: 39min 46s
```

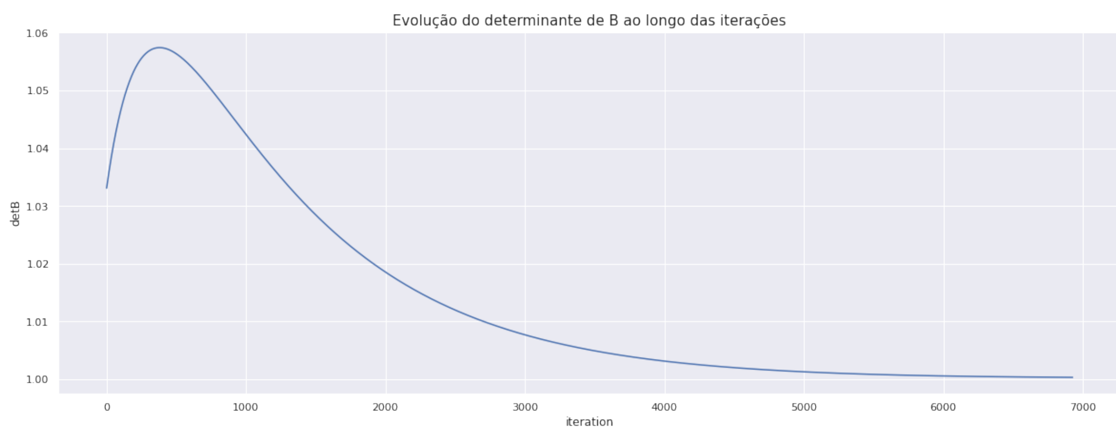
[48]:
```
fig = plt.figure(figsize=(20,7))
sns.lineplot(
    data=logs,
    x='iteration',
    y='log_posteriori'
)
plt.title(
    'Evolução da prob. a posteriori ao longo das iterações',
    fontsize=15
)
```

[48]:
```
Text(0.5, 1.0, 'Evolução da prob. a posteriori ao longo das iterações')
```



[49]:
```
fig = plt.figure(figsize=(20,7))
sns.lineplot(
    data=logs,
    x='iteration',
    y='detB'
)
plt.title(
    'Evolução do determinante de B ao longo das iterações',
    fontsize=15
)
```

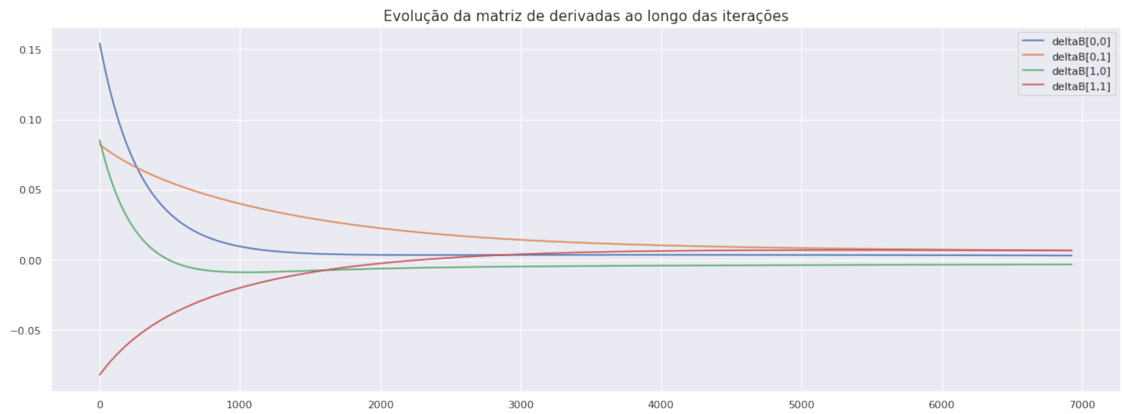
[49]: Text(0.5, 1.0, 'Evolução do determinante de B ao longo das iterações')



[50]:
```
fig = plt.figure(figsize=(20,7))
for i, j in np.ndindex(B.shape):
    plt.plot(
        logs.iteration.values,
```

```
        [row['gradient'][i,j] for n, row in logs.iterrows()],
        label='deltaB[{},{}]'.format(i,j)
    )

plt.title(
    'Evolução da matriz de derivadas ao longo das iterações',
    fontsize=15
)
plt.legend()
```

[50]: <matplotlib.legend.Legend at 0x7f60b6d705b0>



[51]:
```
B
```

[51]:
```
array([[ 0.50756324, -1.00186544],
       [ 0.50136257,  0.98118674]])
```

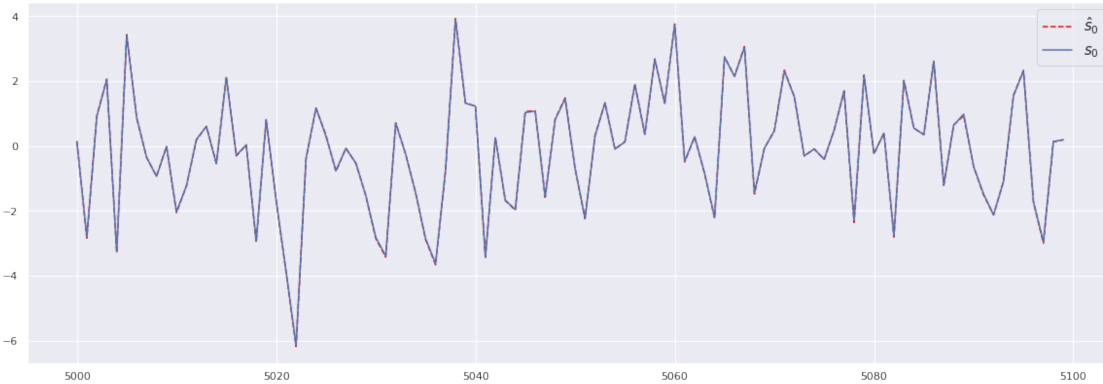
[52]:
```
np.linalg.inv(A)
```

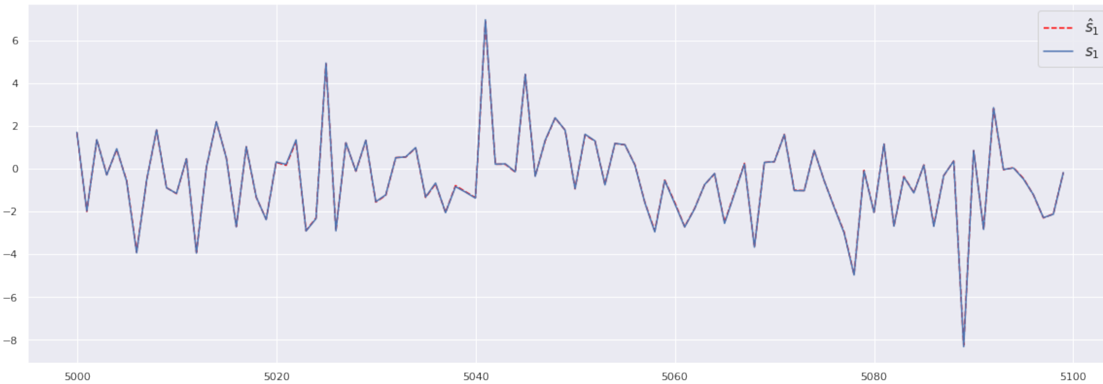
[52]:
```
array([[ 0.5, -1. ],
       [ 0.5,  1. ]])
```

[53]:
```
fig, (ax1, ax2) = plt.subplots(
    nrows=2, ncols=1,
    figsize=(20,15)
)
NPOINTS=100
PLOT_START=NOBS//2
PLOT_END=PLOT_START+NPOINTS

t=range(NOBS)
s_est = B@x
```

```python
#Ax1
ax1.plot(
    t[PLOT_START:PLOT_END],
    s_est[0,PLOT_START:PLOT_END],
    label='$\hat{s}_{0}$',
    color='red',
    linestyle='--'
)
ax1.plot(
    t[PLOT_START:PLOT_END],
    s[0,PLOT_START:PLOT_END],
    label='$s_{0}$'
)
ax1.legend(fontsize=15)

#Ax2
ax2.plot(
    t[PLOT_START:PLOT_END],
    s_est[1,PLOT_START:PLOT_END],
    label='$\hat{s}_{1}$',
    color='red',
    linestyle='--'
)
ax2.plot(
    t[PLOT_START:PLOT_END],
    s[1,PLOT_START:PLOT_END],
    label='$s_{1}$'
)
ax2.legend(fontsize=15)
```

```
[53]: <matplotlib.legend.Legend at 0x7f60e1033ca0>
```





[54]:
```
# Error norm
print('Norma do erro de estimação: {}'.format(
np.linalg.norm(
    np.subtract(s,s_est)
)/np.size(s)))
```

```
Norma do erro de estimação: 0.00015524201893414157
```

5 2. SLIGHTLY MISSPECIFIED MODEL

5.1 2.1. Initialize Sources

[55]:
```
def source_cumulative(x):
    return 1/(1+np.exp(-x))

def source_pdf(x):
    return np.exp(-x)/np.square(1+np.exp(-x))

def source_generator(
    nsources,
    nobs,
```

```
    mu,
    scale
):
    return np.random.logistic(
        loc=mu,
        scale=scale,
        size=(nsources, nobs)
    )
```

[56]:

```
NSOURCES=2
NOBS=7500
MU=0.1
SCALE=1.1

s = source_generator(
    nsources=NSOURCES,
    nobs=NOBS,
    mu=MU,
    scale=SCALE
)

print('-'*100)
print('NÚMERO DE FONTES: {}'.format(NSOURCES))
print('TAMANHO DOS SINAIS DAS FONTES: {}'.format(NOBS))
for i in range(s.shape[0]):
    print(
        'CURTOSE s{}: {}'.format(
            i,
            st.kurtosis(a=s[i,:])
        )
    )

print('-'*100)
```

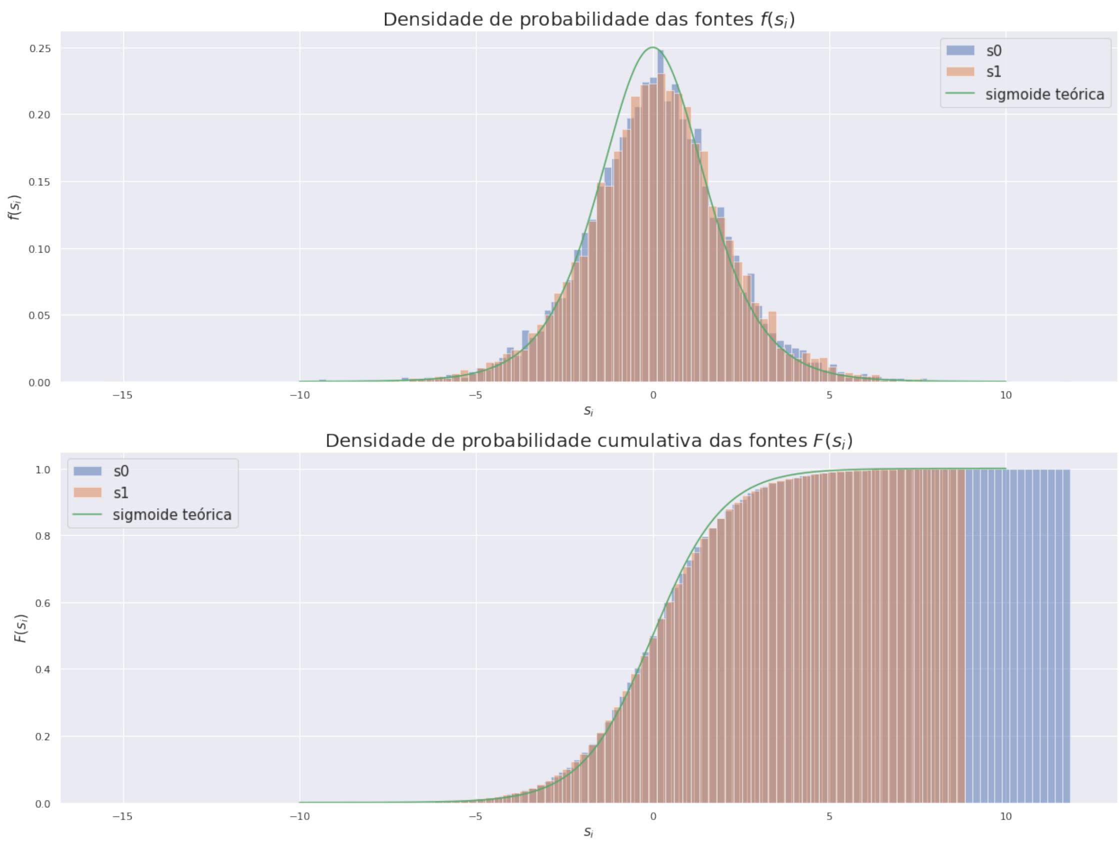
```
--------------------------------------------------------------------------------
--------------------
NÚMERO DE FONTES: 2
TAMANHO DOS SINAIS DAS FONTES: 7500
CURTOSE s0: 1.2718964379009572
CURTOSE s1: 1.2297013264888292
--------------------------------------------------------------------------------
--------------------
```

[57]:

```
fig, (ax1, ax2) = plt.subplots(
    nrows=2, ncols=1,
    figsize=(20,15)
)
```

```
for i in range(1, NSOURCES+1):
    ax1.hist(
        x=s[i-1,:],
        bins=100,
        density=True,
        label='s{}'.format(i-1),
        alpha=0.5
    )
    ax2.hist(
        x=s[i-1,:],
        bins=100,
        cumulative=True,
        density=True,
        label='s{}'.format(i-1),
        alpha=0.5
    )

ax1.plot(
    np.linspace(-10,10,1000),
    [source_pdf(x) for x in np.linspace(-10,10,1000)],
    label='sigmoide teórica'
)


ax2.plot(
    np.linspace(-10,10,1000),
    [source_cumulative(x) for x in np.linspace(-10,10,1000)],
    label='sigmoide teórica'
)

ax1.set_xlabel(
    '$s_{i}$',
    fontsize=15
)
ax1.set_ylabel(
    '$f(s_{i})$',
    fontsize=15
)
ax1.set_title(
    'Densidade de probabilidade das fontes $f(s_{i})$',
    fontsize=20
)

ax2.set_xlabel(
    '$s_{i}$',
```

```
    fontsize=15
)
ax2.set_ylabel(
    '$F(s_{i})$',
    fontsize=15
)
ax2.set_title(
    'Densidade de probabilidade cumulativa das fontes $F(s_{i})$',
    fontsize=20
)

l1=ax1.legend(fontsize=15)
l2=ax2.legend(fontsize=15)
```



5.2 2.2. Mix sources and generate observations

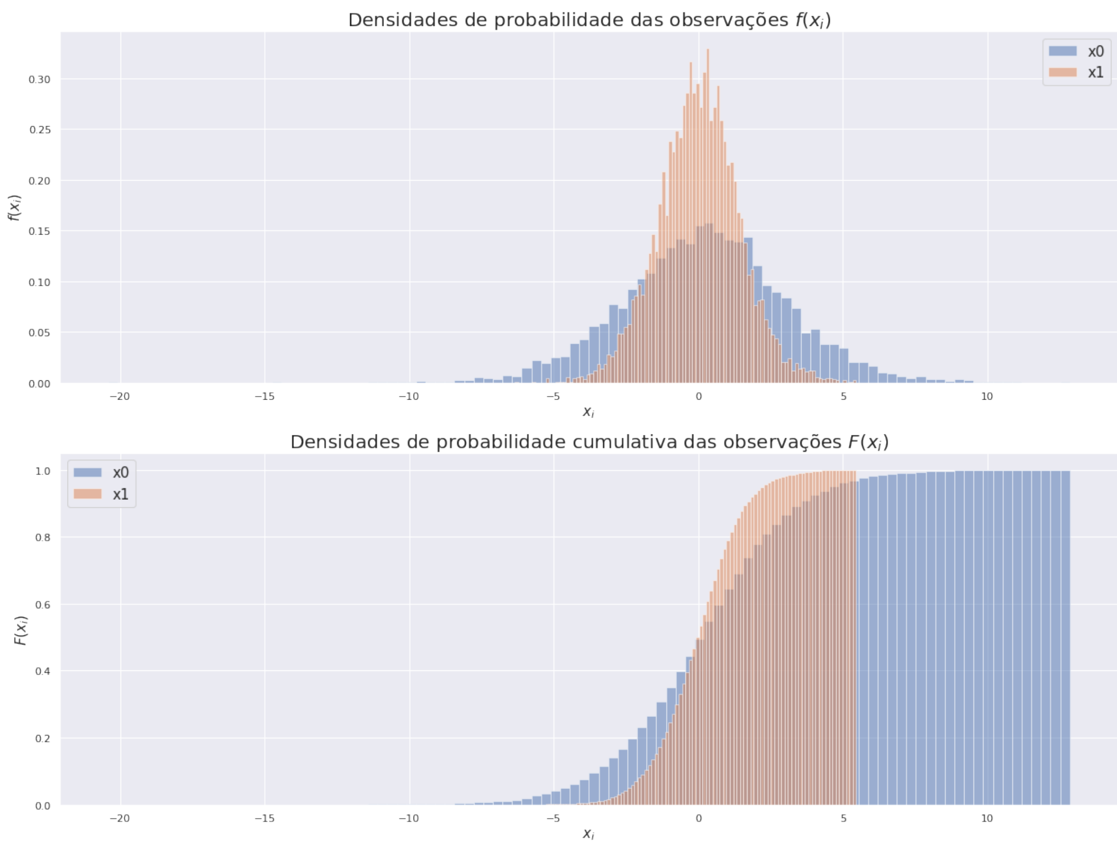
[58]:
```
# Mixing matrix
A = np.array([
    [1, 1],
    [-0.5, 0.5]
```

```
])
print('MIXING MATRIX A:')
print(A)
```

```
MIXING MATRIX A:
[[ 1.   1. ]
 [-0.5  0.5]]
```

[59]:
```
# Observed sources
x = A@s

fig, (ax1, ax2) = plt.subplots(
    nrows=2, ncols=1,
    figsize=(20,15)
)

for i in range(1, NSOURCES+1):
    ax1.hist(
        x=x[i-1,:],
        bins=100,
        density=True,
        label='x{}'.format(i-1),
        alpha=0.5
    )
    ax2.hist(
        x=x[i-1,:],
        bins=100,
        cumulative=True,
        density=True,
        label='x{}'.format(i-1),
        alpha=0.5
    )

ax1.set_xlabel(
    '$x_{i}$',
    fontsize=15
)
ax1.set_ylabel(
    '$f(x_{i})$',
    fontsize=15
)
ax1.set_title(
    'Densidades de probabilidade das observações $f(x_{i})$',
    fontsize=20
)

ax2.set_xlabel(
```

```
    '$x_{i}$',
    fontsize=15
)
ax2.set_ylabel(
    '$F(x_{i})$',
    fontsize=15
)
ax2.set_title(
    'Densidades de probabilidade cumulativa das observações $F(x_{i})$',
    fontsize=20
)

l1=ax1.legend(fontsize=15)
l2=ax2.legend(fontsize=15)
```



5.3 2.3. Performance Curves

[60]:
```
def get_posteriori_neighborhood(
    A,
    x,
```

```
    T,
    u_vec,
    v_vec,
    n_points,
    source_pdf_fn,
    prior_pdf_fn,
    njobs=1
):

    def __get_posteriori(
        A,
        x,
        T,
        source_pdf_fn,
        prior_pdf_fn,
        idx_info
    ):
        i, u = idx_info[0]
        j, v = idx_info[-1]

        A_shifted = A + u*symmetric_basis + v*skew_symmetric_basis

        B = np.linalg.inv(A_shifted)

        likelihood = np.log(np.abs(np.linalg.det(B)))

        likelihood_iterator = [
            (t,i) for t,i in itertools.product(
                range(x.shape[-1]),
                range(x.shape[0]))
        ]

        s_est = B@x

        likelihood += (1/T)*np.sum([
            np.log(source_pdf_fn(s_est[i,t])) for t,i in likelihood_iterator
        ])

        prior = np.log(prior_pdf_fn(B))/T

        return {
            'i': i,
            'j': j,
            'log_posteriori': likelihood + prior
        }
```

```
    symmetric_basis = np.array([
        [0, 1],
        [1, 0]
    ])

    skew_symmetric_basis = np.array([
        [0, -1],
        [1, 0]
    ])

    iterator = itertools.product(
        enumerate(u_vec),
        enumerate(v_vec)
    )
    # it = [x for x in iterator]

    exec_fn = partial(
        __get_posteriori,
        A,
        x,
        T,
        source_pdf_fn,
        prior_pdf_fn
    )

    z = np.empty(
        shape=(
            u_vec.shape[0],
            v_vec.shape[0]
        )
    )
    with pathos.multiprocessing.ProcessingPool(njobs) as p:
        results = p.map(exec_fn, iterator)

    for r in results:
        i=r['i']
        j=r['j']
        z[i, j]=r['log_posteriori']

    return z
```

5.3.1 2.3.1. Likelihood

[61]:
```
def source_pdf(x):
    return np.exp(-x)/np.square(1+np.exp(-x))

def prior_pdf(B):
```

```
    return 1
```

[62]:

```
%%time

u_lims = (-0.3, 0.3)
v_lims = (-0.3, 0.3)
n_points = 200

u_vec = np.linspace(u_lims[0], u_lims[-1], n_points)
v_vec = np.linspace(v_lims[0], v_lims[-1], n_points)

z = get_posteriori_neighborhood(
    A=A,
    x=x,
    T=NOBS,
    u_vec=u_vec,
    v_vec=v_vec,
    n_points=n_points,
    source_pdf_fn=source_pdf,
    prior_pdf_fn=prior_pdf,
    njobs=os.cpu_count()-1
)
```

```
CPU times: user 1.25 s, sys: 46.1 ms, total: 1.29 s
Wall time: 9min 37s
```

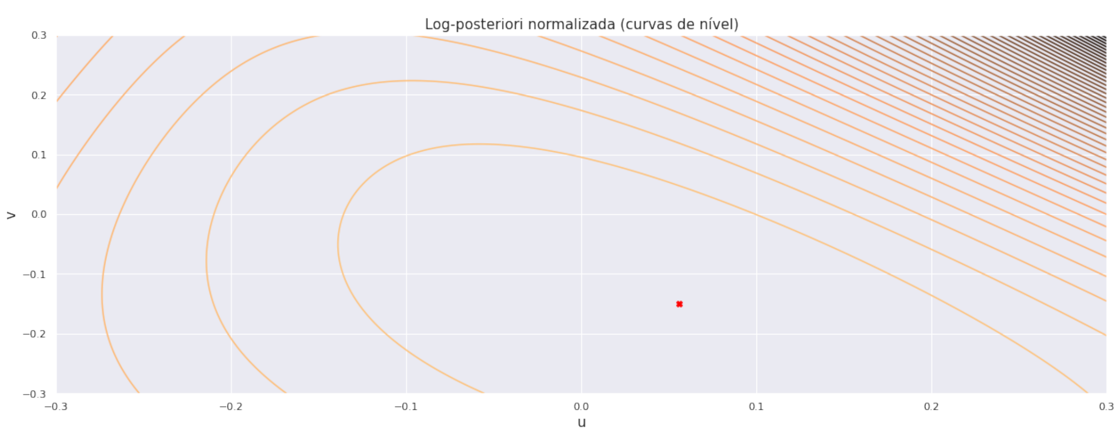
[63]:

```
# Ponto de máximo
max_post_point = np.unravel_index(
    z.argmax(),
    z.shape
)

#
max_post = z[max_post_point]

print('-'*100)
print('Ponto de máximo: u={}, v={}'.format(u_vec[max_post_point[0]],␣
 ↪v_vec[max_post_point[1]]))
print('-'*100)
```

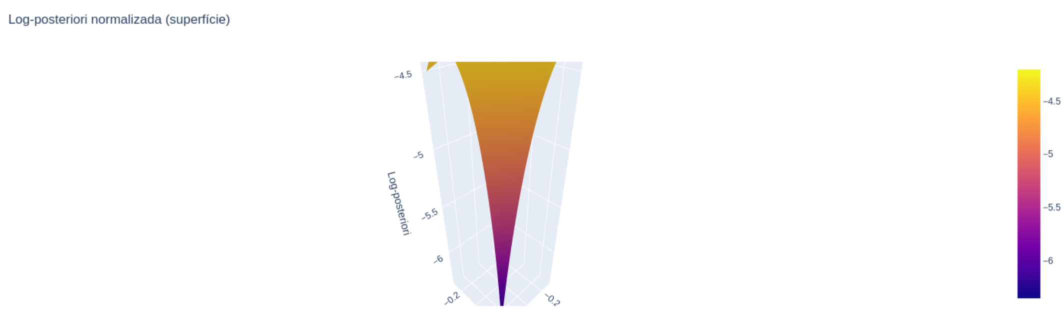
```
--------------------------------------------------------------------------------
--------------------
Ponto de máximo: u=0.055778894472361784, v=-0.1492462311557789
--------------------------------------------------------------------------------
--------------------
```

[64]:
```
# Plot de curvas de nível
fig = plt.figure(figsize=(20,7))

U, V = np.meshgrid(u_vec, v_vec)

plt.contour(
    U,
    V,
    z.T,
    levels=50,
    cmap='copper'
)

plt.scatter(
    u_vec[max_post_point[0]],
    v_vec[max_post_point[-1]],
    marker='X',
    color='red'
)

plt.xlabel(
    'u',
    fontsize=15
)
plt.ylabel(
    'v',
    fontsize=15
)

plt.title(
    'Log-posteriori normalizada (curvas de nível)',
    fontsize=15
)
```

[64]: Text(0.5, 1.0, 'Log-posteriori normalizada (curvas de nível)')



[65]:
```python
# Plot interativo
fig = go.Figure(
    go.Surface(
        x=u_vec,
        y=v_vec,
        z=z
    )
)

fig.update_layout(
    title='Log-posteriori normalizada (superfície)',
    autosize=False,
    width=500, height=500,
    scene=dict(
        xaxis_title='U',
        yaxis_title='V',
        zaxis_title='Log-posteriori',
    )
)

fig.show()
```



[66]:
```python
def source_pdf(x):
    """
        This method takes SCALAR value x and return SCALAR source pdf evaluated at x.
    """
    return np.exp(-x)/np.square(1+np.exp(-x))


def source_pdf_derivative(x):
```

```
    """
        This method takes SCALAR value x and return SCALAR source pdf␣
↪derivative evaluated at x.
    """
    return (2*np.exp(-2*x)*(1+np.exp(-x)) - np.exp(-x)*np.square(1+np.exp(-x)))/
↪np.power([1+np.exp(-x)],[4])


def prior_pdf(X):
    """
        This method takes VECTOR value X and return SCALAR prior pdf evaluated␣
↪at X.
    """
    return 1

def prior_pdf_derivative(X):
    """
        This method takes VECTOR value x and return VECTOR prior pdf derivative␣
↪evaluated at X.
    """
    return np.zeros(
        shape=X.shape
    )
```

5.4 3.1. Standard Gradient

[67]:
```
# Initialize estimator
estimator = MAPGradientEstimator(
    learning_rate=learning_rate,
    thresh=thresh,
    max_it=max_it,
    source_pdf=source_pdf,
    source_pdf_derivative=source_pdf_derivative,
    prior_pdf=prior_pdf,
    prior_pdf_derivative=prior_pdf_derivative,
    is_natural_gradient=False
)
```

[68]:
```
%%time

# Run optimization
estimator.fit(
    s=s,
    x=x,
    initial_condition=initial_B,
    n_initializations=1,
```

```python
    n_jobs=os.cpu_count()-1
)

# Parse results
B=estimator.fit_results[0]['unmixing_matrix']
logs=estimator.fit_results[0]['logs']

print('-'*100)
print('Total iterations: {}'.format(logs.shape[0]))
print('Final log-posteriori: {}'.format(logs.iloc[-1]['log_posteriori']))
print('-'*100)
```

```
/home/leonardo/git/bayesian_bss/src/estimator.py:144: FutureWarning:

elementwise comparison failed; returning scalar instead, but in the future will
perform elementwise comparison


--------------------------------------------------------------------------------
--------------------
Total iterations: 3621
Final log-posteriori: -31508.75193057853
--------------------------------------------------------------------------------
--------------------
CPU times: user 20min 44s, sys: 3min 14s, total: 23min 58s
Wall time: 17min 42s
```

[69]:

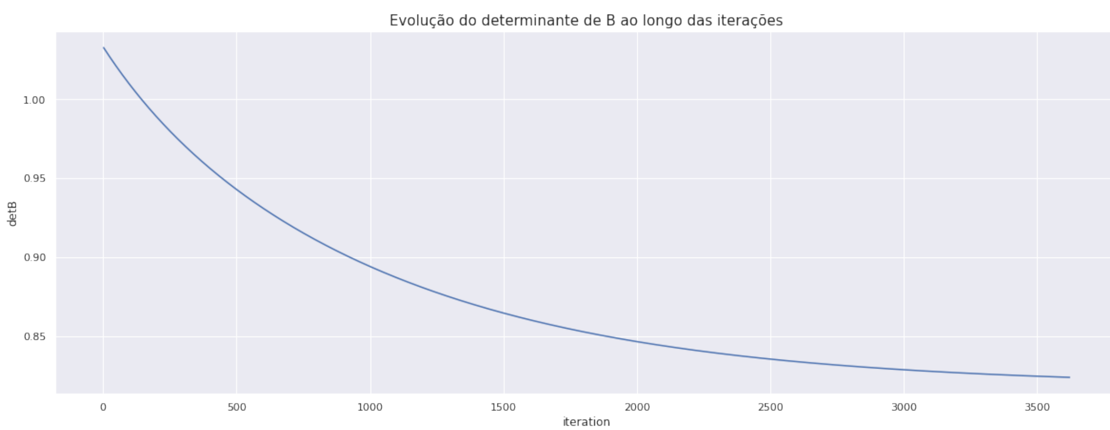
```python
fig = plt.figure(figsize=(20,7))
sns.lineplot(
    data=logs,
    x='iteration',
    y='log_posteriori'
)
plt.title(
    'Evolução da prob. a posteriori ao longo das iterações',
    fontsize=15
)
```

[69]:

```
Text(0.5, 1.0, 'Evolução da prob. a posteriori ao longo das iterações')
```

[70]:
```
fig = plt.figure(figsize=(20,7))
sns.lineplot(
    data=logs,
    x='iteration',
    y='detB'
)
plt.title(
    'Evolução do determinante de B ao longo das iterações',
    fontsize=15
)
```

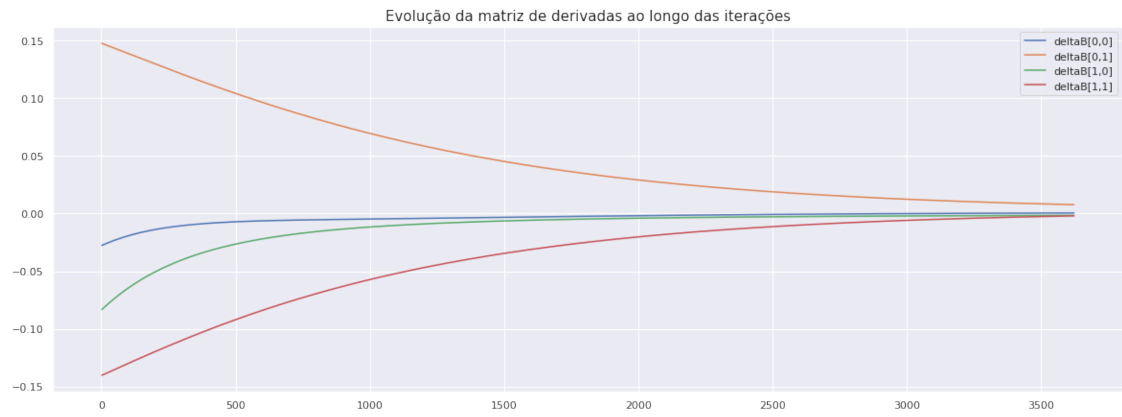
[70]: Text(0.5, 1.0, 'Evolução do determinante de B ao longo das iterações')



[71]:
```
fig = plt.figure(figsize=(20,7))
for i, j in np.ndindex(B.shape):
    plt.plot(
        logs.iteration.values,
```

```
        [row['gradient'][i,j] for n, row in logs.iterrows()],
        label='deltaB[{},{}]'.format(i,j)
    )

plt.title(
    'Evolução da matriz de derivadas ao longo das iterações',
    fontsize=15
)
plt.legend()
```

[71]: <matplotlib.legend.Legend at 0x7f60900b19a0>



[72]:
```
B
```

[72]:
```
array([[ 0.42398145, -0.9650854 ],
       [ 0.47453864,  0.86270222]])
```

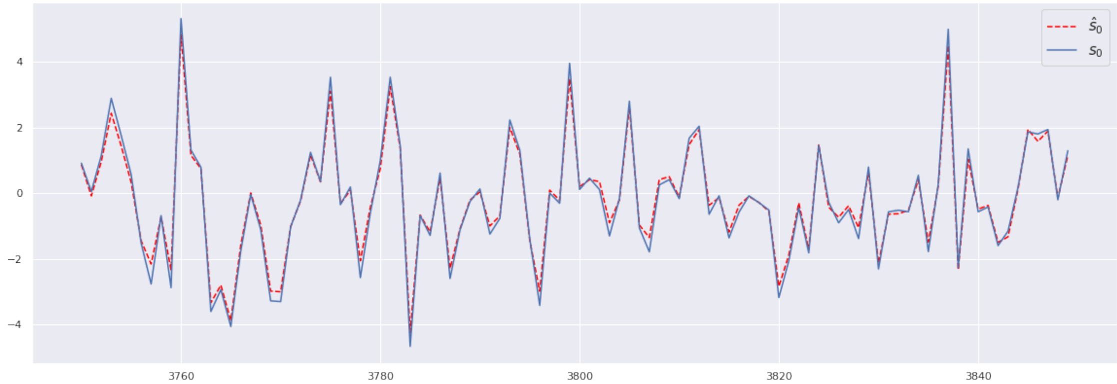
[73]:
```
np.linalg.inv(A)
```

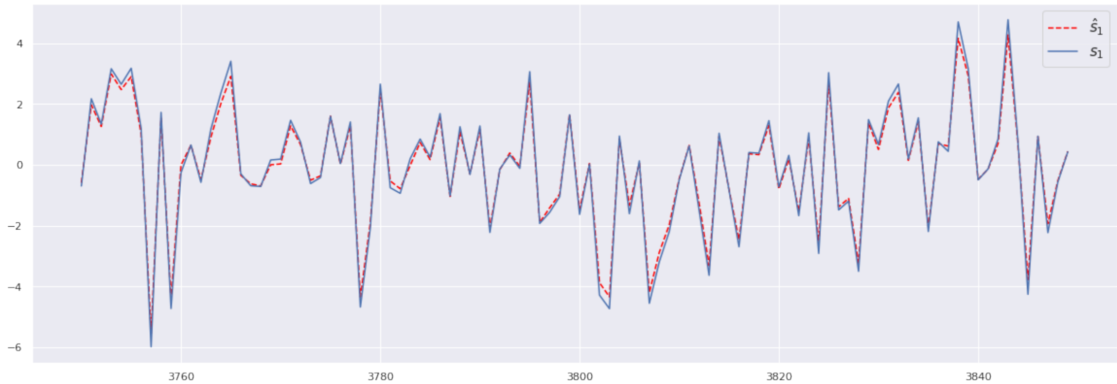
[73]:
```
array([[ 0.5, -1. ],
       [ 0.5,  1. ]])
```

[74]:
```
fig, (ax1, ax2) = plt.subplots(
    nrows=2, ncols=1,
    figsize=(20,15)
)
NPOINTS=100
PLOT_START=NOBS//2
PLOT_END=PLOT_START+NPOINTS

t=range(NOBS)
s_est = B@x
```

```
#Ax1
ax1.plot(
    t[PLOT_START:PLOT_END],
    s_est[0,PLOT_START:PLOT_END],
    label='$\hat{s}_{0}$',
    color='red',
    linestyle='--'
)
ax1.plot(
    t[PLOT_START:PLOT_END],
    s[0,PLOT_START:PLOT_END],
    label='$s_{0}$'
)
ax1.legend(fontsize=15)

#Ax2
ax2.plot(
    t[PLOT_START:PLOT_END],
    s_est[1,PLOT_START:PLOT_END],
    label='$\hat{s}_{1}$',
    color='red',
    linestyle='--'
)
ax2.plot(
    t[PLOT_START:PLOT_END],
    s[1,PLOT_START:PLOT_END],
    label='$s_{1}$'
)
ax2.legend(fontsize=15)
```

```
[74]: <matplotlib.legend.Legend at 0x7f60902e6820>
```





[75]:
```
# Error norm
print('Norma do erro de estimação: {}'.format(
np.linalg.norm(
    np.subtract(s,s_est)
)/np.size(s)))
```

```
Norma do erro de estimação: 0.0017527322277779728
```

5.4.1 2.3.2. MAP - determinant equal to one

[76]:
```
def source_pdf(x):
    return np.exp(-x)/np.square(1+np.exp(-x))

def prior_pdf(B):
    sig=0.1
    desired_det=1
    return (1/np.sqrt(2*np.pi*np.square(sig)))*np.exp(-np.square(np.linalg.
 ↪det(B)-desired_det)/(2*np.square(sig)))
```

[77]:
```
%%time
```

```
u_lims = (-0.3, 0.3)
v_lims = (-0.3, 0.3)
n_points = 200

u_vec = np.linspace(u_lims[0], u_lims[-1], n_points)
v_vec = np.linspace(v_lims[0], v_lims[-1], n_points)

z = get_posteriori_neighborhood(
    A=A,
    x=x,
    T=NOBS,
    u_vec=u_vec,
    v_vec=v_vec,
    n_points=n_points,
    source_pdf_fn=source_pdf,
    prior_pdf_fn=prior_pdf,
    njobs=os.cpu_count()-1
)
```

```
CPU times: user 1.45 s, sys: 335 ms, total: 1.78 s
Wall time: 12min 23s
```

[78]:

```
# Ponto de máximo
max_post_point = np.unravel_index(
    z.argmax(),
    z.shape
)

#
max_post = z[max_post_point]

print('-'*100)
print('Ponto de máximo: u={}, v={}'.format(u_vec[max_post_point[0]],
 ↪v_vec[max_post_point[1]]))
print('-'*100)
```

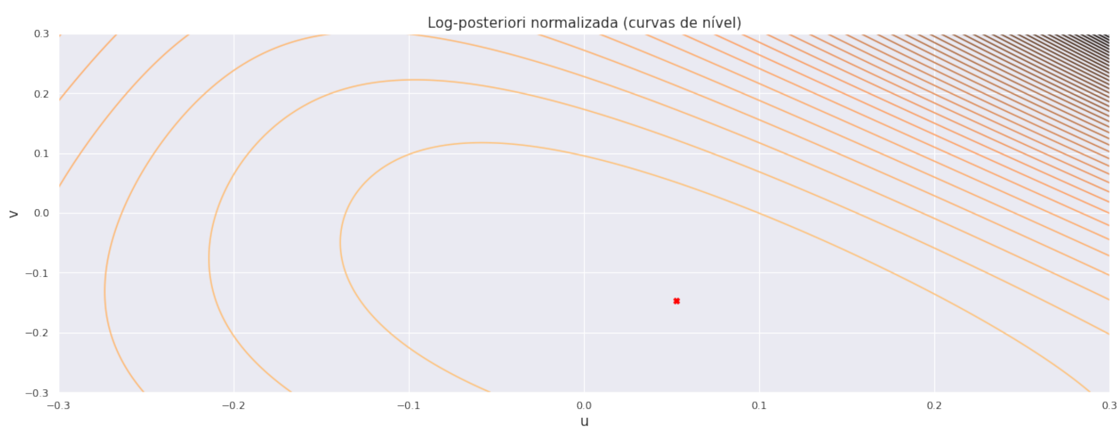
```
--------------------------------------------------------------------------------
--------------------
Ponto de máximo: u=0.052763819095477390, v=-0.14623115577889448
--------------------------------------------------------------------------------
--------------------
```

[79]:

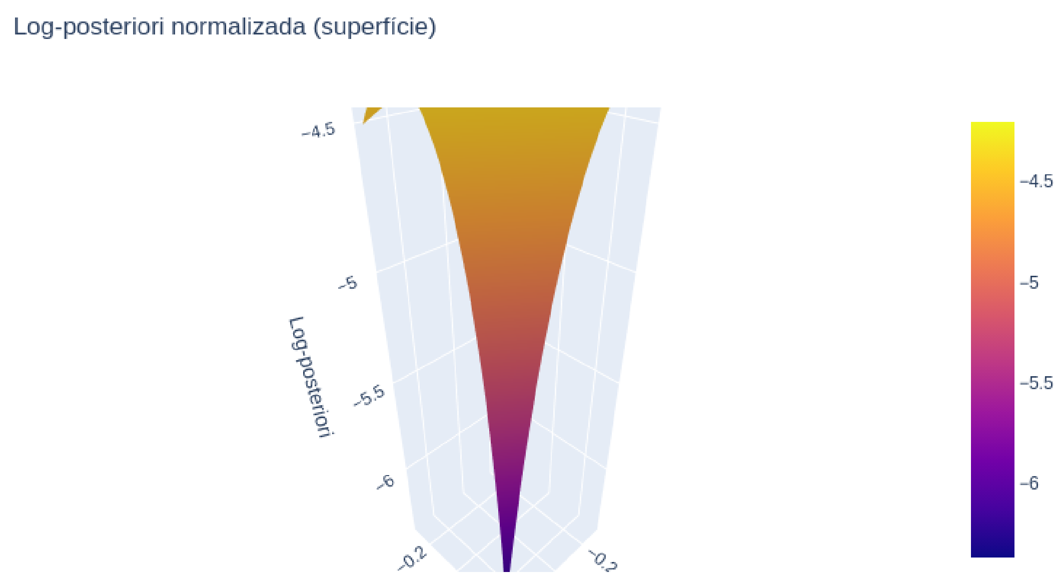
```
# Plot de curvas de nível
fig = plt.figure(figsize=(20,7))

U, V = np.meshgrid(u_vec, v_vec)
```

```
plt.contour(
    U,
    V,
    z.T,
    levels=50,
    cmap='copper'
)

plt.scatter(
    u_vec[max_post_point[0]],
    v_vec[max_post_point[-1]],
    marker='X',
    color='red'
)

plt.xlabel(
    'u',
    fontsize=15
)
plt.ylabel(
    'v',
    fontsize=15
)

plt.title(
    'Log-posteriori normalizada (curvas de nível)',
    fontsize=15
)
```

[79]: Text(0.5, 1.0, 'Log-posteriori normalizada (curvas de nível)')



[80]:
```python
# Plot interativo
fig = go.Figure(
    go.Surface(
        x=u_vec,
        y=v_vec,
        z=z
    )
)

fig.update_layout(
    title='Log-posteriori normalizada (superfície)',
    autosize=False,
    width=500, height=500,
    scene=dict(
        xaxis_title='U',
        yaxis_title='V',
        zaxis_title='Log-posteriori',
    )
)

fig.show()
```



[81]:
```python
def source_pdf(x):
    """
```

```
        This method takes SCALAR value x and return SCALAR source pdf evaluated␣
↪at x.
    """
    return np.exp(-x)/np.square(1+np.exp(-x))


def source_pdf_derivative(x):
    """
        This method takes SCALAR value x and return SCALAR source pdf␣
↪derivative evaluated at x.
    """
    return (2*np.exp(-2*x)*(1+np.exp(-x)) - np.exp(-x)*np.square(1+np.exp(-x)))/
↪np.power([1+np.exp(-x)],[4])

def prior_pdf(X):
    sig=0.1
    desired_det=1
    return (1/np.sqrt(2*np.pi*np.square(sig)))*np.exp(-np.square(np.linalg.
↪det(X)-desired_det)/(2*np.square(sig)))

def prior_pdf_derivative(X):
    sig=0.1
    return -1*prior_pdf(X)*(np.linalg.det(X)-1)/np.square(sig)
```

5.5 3.1. Standard Gradient

[82]:
```
# Initialize estimator
estimator = MAPGradientEstimator(
    learning_rate=learning_rate,
    thresh=thresh,
    max_it=max_it,
    source_pdf=source_pdf,
    source_pdf_derivative=source_pdf_derivative,
    prior_pdf=prior_pdf,
    prior_pdf_derivative=prior_pdf_derivative,
    is_natural_gradient=False
)
```

[83]:
```
%%time

# Run optimization
estimator.fit(
    s=s,
    x=x,
    initial_condition=initial_B,
    n_initializations=1,
```

```
    n_jobs=os.cpu_count()-1
)

# Parse results
B=estimator.fit_results[0]['unmixing_matrix']
logs=estimator.fit_results[0]['logs']

print('-'*100)
print('Total iterations: {}'.format(logs.shape[0]))
print('Final log-posteriori: {}'.format(logs.iloc[-1]['log_posteriori']))
print('-'*100)
```

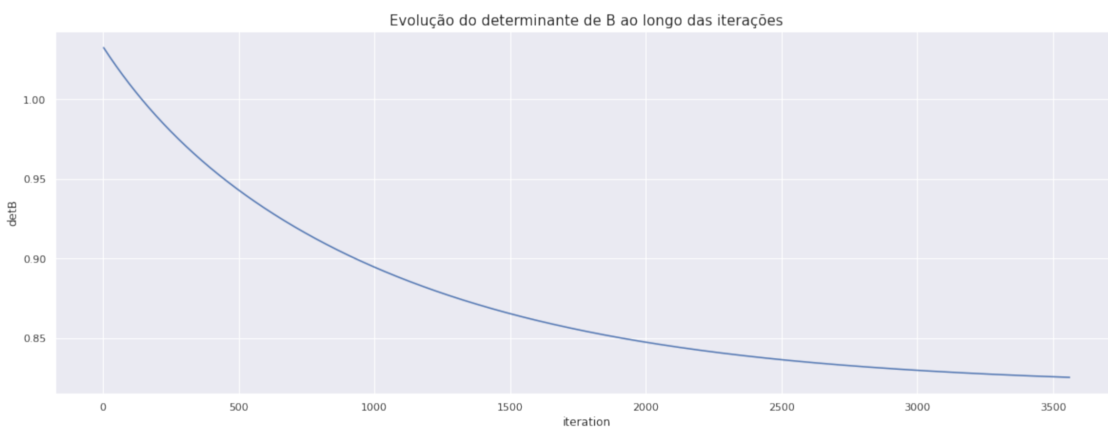
```
/home/leonardo/git/bayesian_bss/src/estimator.py:144: FutureWarning:

elementwise comparison failed; returning scalar instead, but in the future will
perform elementwise comparison


--------------------------------------------------------------------------------
--------------------
Total iterations: 3561
Final log-posteriori: -31508.759143928924
--------------------------------------------------------------------------------
--------------------
CPU times: user 21min 41s, sys: 3min 8s, total: 24min 49s
Wall time: 18min 41s
```

[84]:
```
fig = plt.figure(figsize=(20,7))
sns.lineplot(
    data=logs,
    x='iteration',
    y='log_posteriori'
)
plt.title(
    'Evolução da prob. a posteriori ao longo das iterações',
    fontsize=15
)
```

[84]:
```
Text(0.5, 1.0, 'Evolução da prob. a posteriori ao longo das iterações')
```

[85]:
```
fig = plt.figure(figsize=(20,7))
sns.lineplot(
    data=logs,
    x='iteration',
    y='detB'
)
plt.title(
    'Evolução do determinante de B ao longo das iterações',
    fontsize=15
)
```

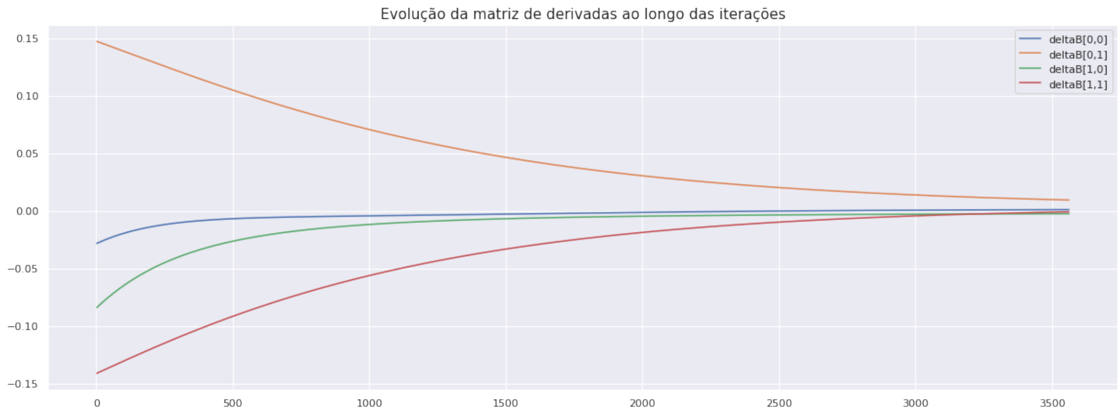
[85]: `Text(0.5, 1.0, 'Evolução do determinante de B ao longo das iterações')`



[86]:
```
fig = plt.figure(figsize=(20,7))
for i, j in np.ndindex(B.shape):
    plt.plot(
        logs.iteration.values,
```

```
        [row['gradient'][i,j] for n, row in logs.iterrows()],
        label='deltaB[{},{}]'.format(i,j)
    )

plt.title(
    'Evolução da matriz de derivadas ao longo das iterações',
    fontsize=15
)
plt.legend()
```

[86]: <matplotlib.legend.Legend at 0x7f60e1969fa0>



[87]:
```
B
```

[87]:
```
array([[ 0.42621741, -0.96127562],
       [ 0.47367341,  0.86770686]])
```

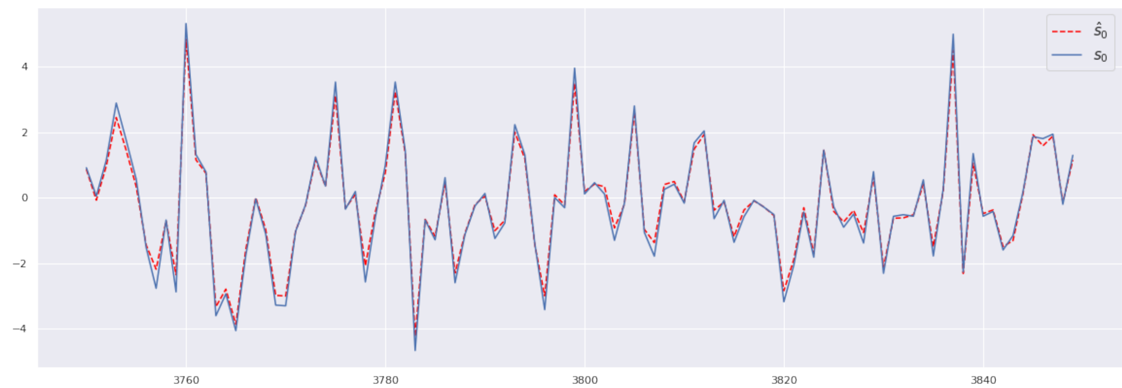
[88]:
```
np.linalg.inv(A)
```

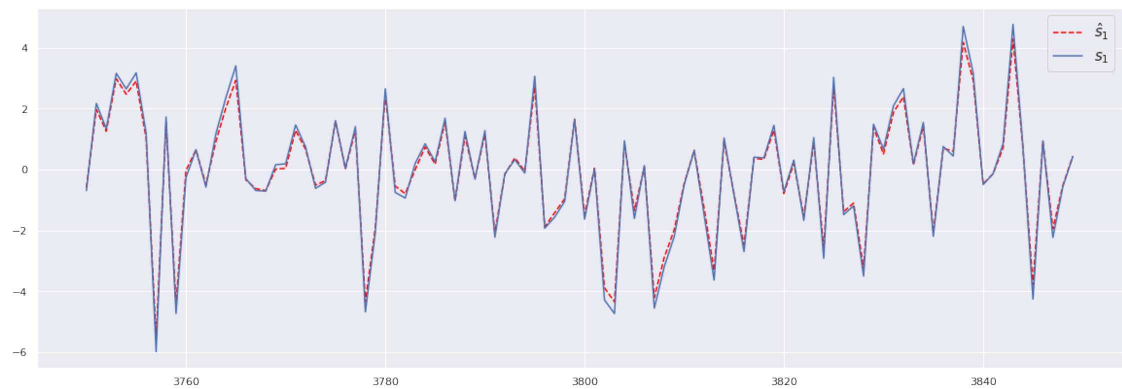
[88]:
```
array([[ 0.5, -1. ],
       [ 0.5,  1. ]])
```

[89]:
```
fig, (ax1, ax2) = plt.subplots(
    nrows=2, ncols=1,
    figsize=(20,15)
)
NPOINTS=100
PLOT_START=NOBS//2
PLOT_END=PLOT_START+NPOINTS

t=range(NOBS)
s_est = B@x
```

```python
#Ax1
ax1.plot(
    t[PLOT_START:PLOT_END],
    s_est[0,PLOT_START:PLOT_END],
    label='$\hat{s}_{0}$',
    color='red',
    linestyle='--'
)
ax1.plot(
    t[PLOT_START:PLOT_END],
    s[0,PLOT_START:PLOT_END],
    label='$s_{0}$'
)
ax1.legend(fontsize=15)

#Ax2
ax2.plot(
    t[PLOT_START:PLOT_END],
    s_est[1,PLOT_START:PLOT_END],
    label='$\hat{s}_{1}$',
    color='red',
    linestyle='--'
)
ax2.plot(
    t[PLOT_START:PLOT_END],
    s[1,PLOT_START:PLOT_END],
    label='$s_{1}$'
)
ax2.legend(fontsize=15)
```

[89]: <matplotlib.legend.Legend at 0x7f60b68b55e0>





[90]:
```
# Error norm
print('Norma do erro de estimação: {}'.format(
np.linalg.norm(
    np.subtract(s,s_est)
)/np.size(s)))
```

```
Norma do erro de estimação: 0.0017096813766631503
```

5.6 2.3.3. MAP - near-identity transformation

[91]:
```
def source_pdf(x):
    return 0.5*np.exp(-np.abs(x))
    # return np.exp(-x)/np.square(1+np.exp(-x))

def prior_pdf(B):
    sig=0.1
    return np.exp(
        (-1/2/np.square(sig))*np.square(np.linalg.norm(B-np.eye(B.shape[0])))
    )
```

[92]:

```
%%time

u_lims = (-0.5, 0.5)
v_lims = (-0.5, 0.5)
n_points = 200

u_vec = np.linspace(u_lims[0], u_lims[-1], n_points)
v_vec = np.linspace(v_lims[0], v_lims[-1], n_points)

z = get_posteriori_neighborhood(
    A=A,
    x=x,
    T=NOBS,
    u_vec=u_vec,
    v_vec=v_vec,
    n_points=n_points,
    source_pdf_fn=source_pdf,
    prior_pdf_fn=prior_pdf,
    njobs=os.cpu_count()-1
)
```

```
---------------------------------------------------------------------------
RemoteTraceback                           Traceback (most recent call last)
RemoteTraceback:
"""
Traceback (most recent call last):
  File "/home/leonardo/py3env/lib/python3.8/site-packages/multiprocess/pool.py",
 ↪line 125, in worker
    result = (True, func(*args, **kwds))
  File "/home/leonardo/py3env/lib/python3.8/site-packages/multiprocess/pool.py",
 ↪line 48, in mapstar
    return list(map(*args))
  File "/home/leonardo/py3env/lib/python3.8/site-packages/pathos/helpers/
 ↪mp_helper.py", line 15, in <lambda>
    func = lambda args: f(*args)
  File "/tmp/ipykernel_45077/2236618269.py", line 26, in __get_posteriori
    B = np.linalg.inv(A_shifted)
  File "<__array_function__ internals>", line 180, in inv
  File "/home/leonardo/py3env/lib/python3.8/site-packages/numpy/linalg/linalg.
 ↪py", line 552, in inv
    ainv = _umath_linalg.inv(a, signature=signature, extobj=extobj)
  File "/home/leonardo/py3env/lib/python3.8/site-packages/numpy/linalg/linalg.
 ↪py", line 89, in _raise_linalgerror_singular
    raise LinAlgError("Singular matrix")
numpy.linalg.LinAlgError: Singular matrix
"""
```

```
The above exception was the direct cause of the following exception:

LinAlgError                               Traceback (most recent call last)
<timed exec> in <module>

/tmp/ipykernel_45077/2236618269.py in get_posteriori_neighborhood(A, x, T,␣
↪u_vec, v_vec, n_points, source_pdf_fn, prior_pdf_fn, njobs)
     81     )
     82     with pathos.multiprocessing.ProcessingPool(njobs) as p:
---> 83         results = p.map(exec_fn, iterator)
     84 
     85     for r in results:

~/py3env/lib/python3.8/site-packages/pathos/multiprocessing.py in map(self, f,␣
↪*args, **kwds)
    137         AbstractWorkerPool._AbstractWorkerPool__map(self, f, *args,␣
↪**kwds)
    138         _pool = self._serve()
--> 139         return _pool.map(star(f), zip(*args)) # chunksize
    140     map.__doc__ = AbstractWorkerPool.map.__doc__
    141     def imap(self, f, *args, **kwds):

~/py3env/lib/python3.8/site-packages/multiprocess/pool.py in map(self, func,␣
↪iterable, chunksize)
    362         in a list that is returned.
    363         '''
--> 364         return self._map_async(func, iterable, mapstar, chunksize).get(
    365 
    366     def starmap(self, func, iterable, chunksize=None):

~/py3env/lib/python3.8/site-packages/multiprocess/pool.py in get(self, timeout)
    769             return self._value
    770         else:
--> 771             raise self._value
    772 
    773     def _set(self, i, obj):

LinAlgError: Singular matrix
```

[93]:
```
# Ponto de máximo
max_post_point = np.unravel_index(
    z.argmax(),
    z.shape
)

#
max_post = z[max_post_point]
```

```
print('-'*100)
print('Ponto de máximo: u={}, v={}'.format(u_vec[max_post_point[0]],␣
↪v_vec[max_post_point[1]]))
print('-'*100)
```

```
--------------------------------------------------------------------------------
--------------------
Ponto de máximo: u=0.0879396984924623, v=-0.24371859296482412
--------------------------------------------------------------------------------
--------------------
```

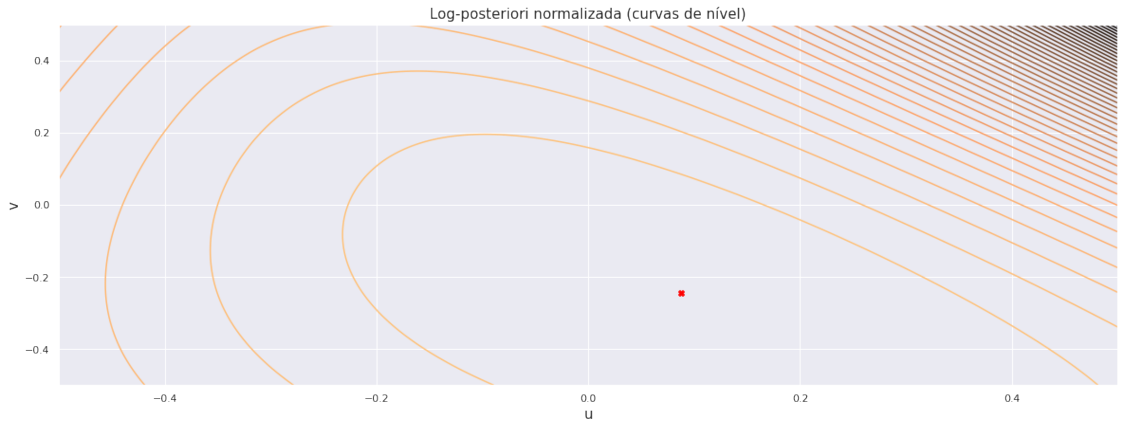
[94]:
```
max_post_point
```

[94]: (117, 51)

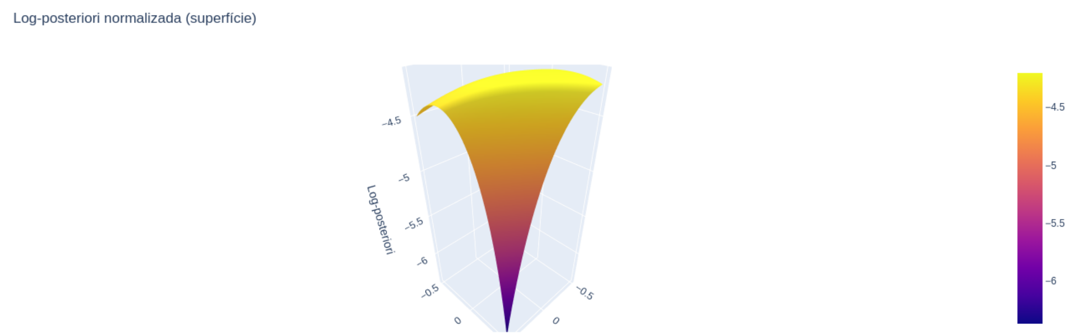
[95]:
```
# Plot de curvas de nível
fig = plt.figure(figsize=(20,7))

U, V = np.meshgrid(u_vec, v_vec)

plt.contour(
    U,
    V,
    z.T,
    levels=50,
    cmap='copper'
)

plt.scatter(
    u_vec[max_post_point[0]],
    v_vec[max_post_point[-1]],
    marker='X',
    color='red'
)

plt.xlabel(
    'u',
    fontsize=15
)
plt.ylabel(
    'v',
    fontsize=15
)

plt.title(
    'Log-posteriori normalizada (curvas de nível)',
```

```
    fontsize=15
)
```

[95]: Text(0.5, 1.0, 'Log-posteriori normalizada (curvas de nível)')



[96]:
```
# Plot interativo
fig = go.Figure(
    go.Surface(
        x=u_vec,
        y=v_vec,
        z=z
    )
)

fig.update_layout(
    title='Log-posteriori normalizada (superfície)',
    autosize=False,
    width=500, height=500,
    scene=dict(
        xaxis_title='U',
        yaxis_title='V',
        zaxis_title='Log-posteriori',
    )
)

fig.show()
```



[97]:
```python
def source_pdf(x):
    """
        This method takes SCALAR value x and return SCALAR source pdf evaluated␣
    ↪at x.
    """
    return np.exp(-x)/np.square(1+np.exp(-x))


def source_pdf_derivative(x):
    """
        This method takes SCALAR value x and return SCALAR source pdf␣
    ↪derivative evaluated at x.
    """
    return (2*np.exp(-2*x)*(1+np.exp(-x)) - np.exp(-x)*np.square(1+np.exp(-x)))/
    ↪np.power([1+np.exp(-x)],[4])

def prior_pdf(X):
    sig=0.1
    return np.exp(
        (-1/2/np.square(sig))*np.linalg.norm(X-np.eye(X.shape[0]))
    )

def prior_pdf_derivative(X):
    sig=0.1
    der_X=np.empty(X.shape)
    I=np.eye(X.shape[0])
    for i, j in np.ndindex(X.shape):
        der_X[i, j] = prior_pdf(X)*(-1/np.square(sig))*(X[i,j]-I[i,j])
    return der_X
```

5.7 3.1. Standard Gradient

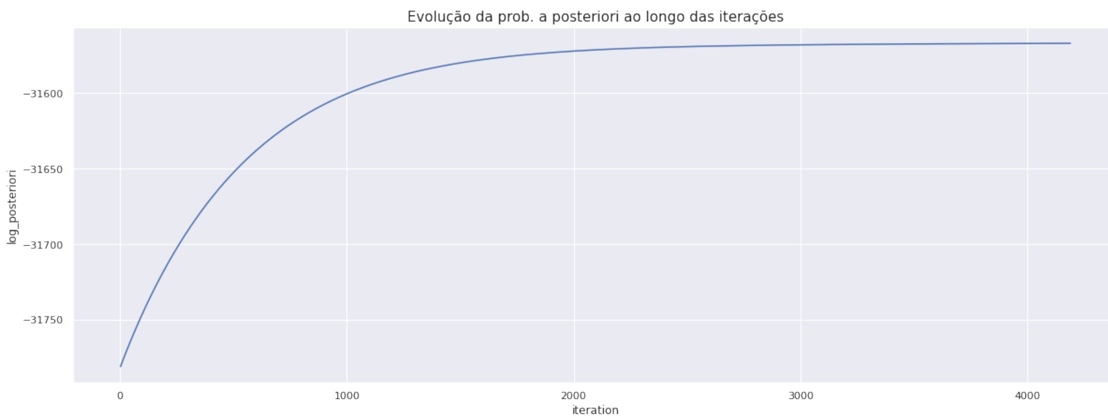
[98]:
```
# Initialize estimator
estimator = MAPGradientEstimator(
    learning_rate=learning_rate,
    thresh=thresh,
    max_it=max_it,
    source_pdf=source_pdf,
    source_pdf_derivative=source_pdf_derivative,
    prior_pdf=prior_pdf,
    prior_pdf_derivative=prior_pdf_derivative,
    is_natural_gradient=False
)
```

[99]:
```
%%time

# Run optimization
estimator.fit(
    s=s,
    x=x,
    initial_condition=initial_B,
    n_initializations=1,
    n_jobs=os.cpu_count()-1
)

# Parse results
B=estimator.fit_results[0]['unmixing_matrix']
logs=estimator.fit_results[0]['logs']

print('-'*100)
print('Total iterations: {}'.format(logs.shape[0]))
print('Final log-posteriori: {}'.format(logs.iloc[-1]['log_posteriori']))
print('-'*100)
```

```
/home/leonardo/git/bayesian_bss/src/estimator.py:144: FutureWarning:

elementwise comparison failed; returning scalar instead, but in the future will
perform elementwise comparison


--------------------------------------------------------------------------------
--------------------
Total iterations: 4190
Final log-posteriori: -31567.15416747123
--------------------------------------------------------------------------------
--------------------
CPU times: user 21min 4s, sys: 3min 46s, total: 24min 50s
Wall time: 17min 33s
```

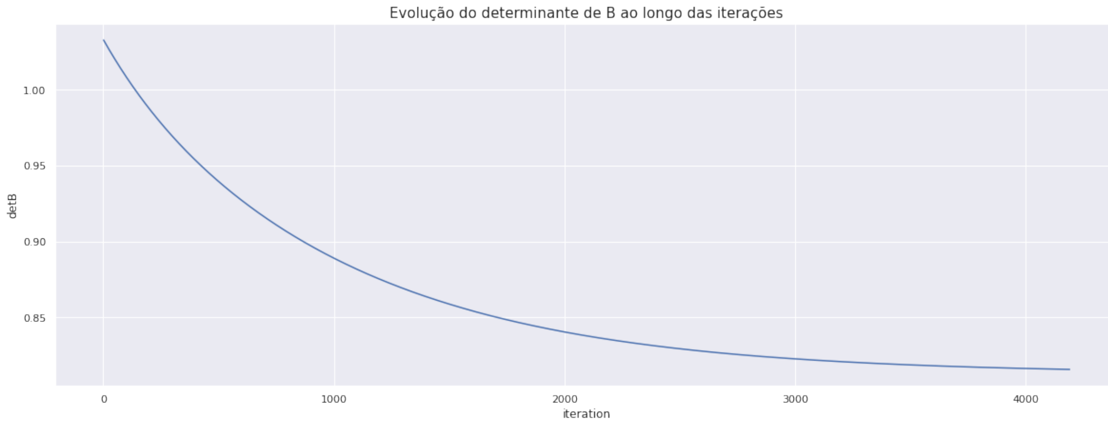
[100]:
```python
fig = plt.figure(figsize=(20,7))
sns.lineplot(
    data=logs,
    x='iteration',
    y='log_posteriori'
)
plt.title(
    'Evolução da prob. a posteriori ao longo das iterações',
    fontsize=15
)
```

[100]: Text(0.5, 1.0, 'Evolução da prob. a posteriori ao longo das iterações')



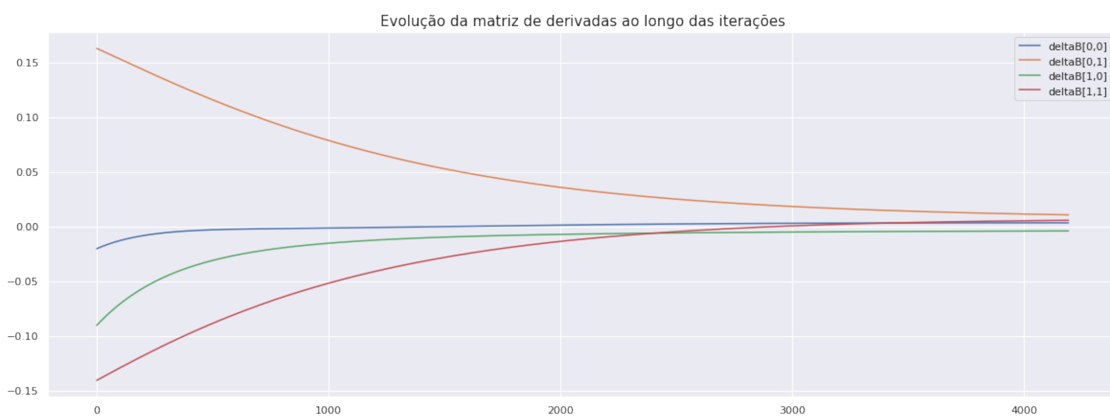
[101]:
```python
fig = plt.figure(figsize=(20,7))
sns.lineplot(
    data=logs,
    x='iteration',
    y='detB'
)
plt.title(
    'Evolução do determinante de B ao longo das iterações',
    fontsize=15
)
```

[101]: Text(0.5, 1.0, 'Evolução do determinante de B ao longo das iterações')



[102]:
```
fig = plt.figure(figsize=(20,7))
for i, j in np.ndindex(B.shape):
    plt.plot(
        logs.iteration.values,
        [row['gradient'][i,j] for n, row in logs.iterrows()],
        label='deltaB[{},{}]'.format(i,j)
    )

plt.title(
    'Evolução da matriz de derivadas ao longo das iterações',
    fontsize=15
)
plt.legend()
```

[102]: <matplotlib.legend.Legend at 0x7f60b22a2f70>



[103]:
```
B
```

[103]:
```
array([[ 0.43895287, -0.92856308],
       [ 0.4596433 ,  0.88590361]])
```

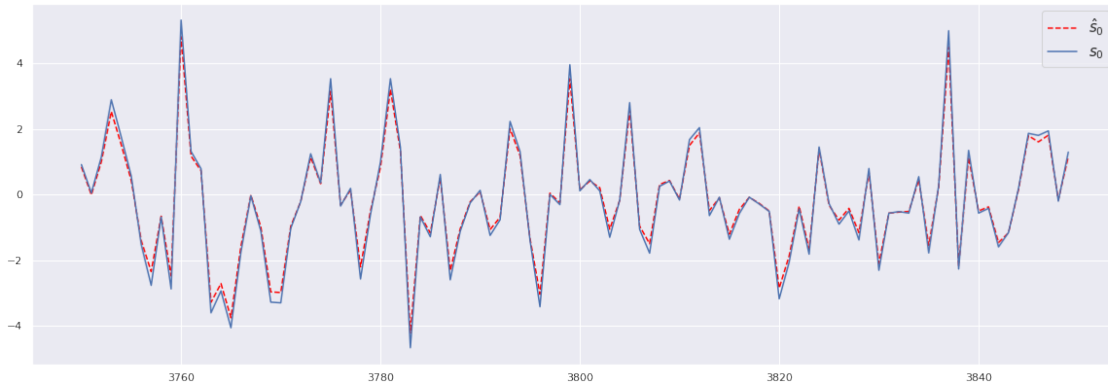
[104]:
```python
np.linalg.inv(A)
```

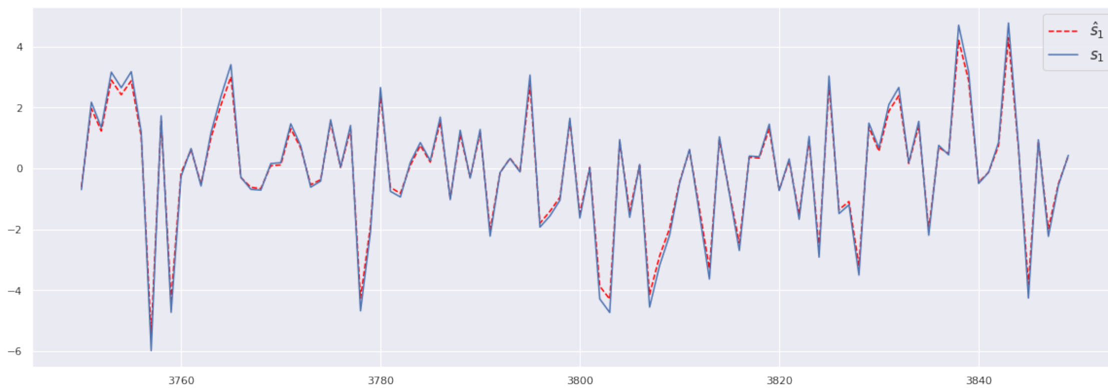
[104]:
```
array([[ 0.5, -1. ],
       [ 0.5,  1. ]])
```

[105]:
```python
fig, (ax1, ax2) = plt.subplots(
    nrows=2, ncols=1,
    figsize=(20,15)
)
NPOINTS=100
PLOT_START=NOBS//2
PLOT_END=PLOT_START+NPOINTS

t=range(NOBS)
s_est = B@x

#Ax1
ax1.plot(
    t[PLOT_START:PLOT_END],
    s_est[0,PLOT_START:PLOT_END],
    label='$\hat{s}_{0}$',
    color='red',
    linestyle='--'
)
ax1.plot(
    t[PLOT_START:PLOT_END],
    s[0,PLOT_START:PLOT_END],
    label='$s_{0}$'
)
ax1.legend(fontsize=15)

#Ax2
ax2.plot(
    t[PLOT_START:PLOT_END],
    s_est[1,PLOT_START:PLOT_END],
    label='$\hat{s}_{1}$',
    color='red',
    linestyle='--'
)
ax2.plot(
    t[PLOT_START:PLOT_END],
    s[1,PLOT_START:PLOT_END],
    label='$s_{1}$'
)
```

```
ax2.legend(fontsize=15)
```

[105]: <matplotlib.legend.Legend at 0x7f60afe13f10>





[106]:
```
# Error norm
print('Norma do erro de estimação: {}'.format(
np.linalg.norm(
    np.subtract(s,s_est)
)/np.size(s)))
```

```
Norma do erro de estimação: 0.0016288898959477957
```

6 3. MORE MISSPECIFIED MODEL

6.1 3.1. Initialize Sources

[107]:
```
def source_cumulative(x):
    return 1/(1+np.exp(-x))

def source_pdf(x):
    return np.exp(-x)/np.square(1+np.exp(-x))
```

```python
def source_generator(
    nsources,
    nobs,
    mu,
    scale
):
    return np.random.logistic(
        loc=mu,
        scale=scale,
        size=(nsources, nobs)
    )
```

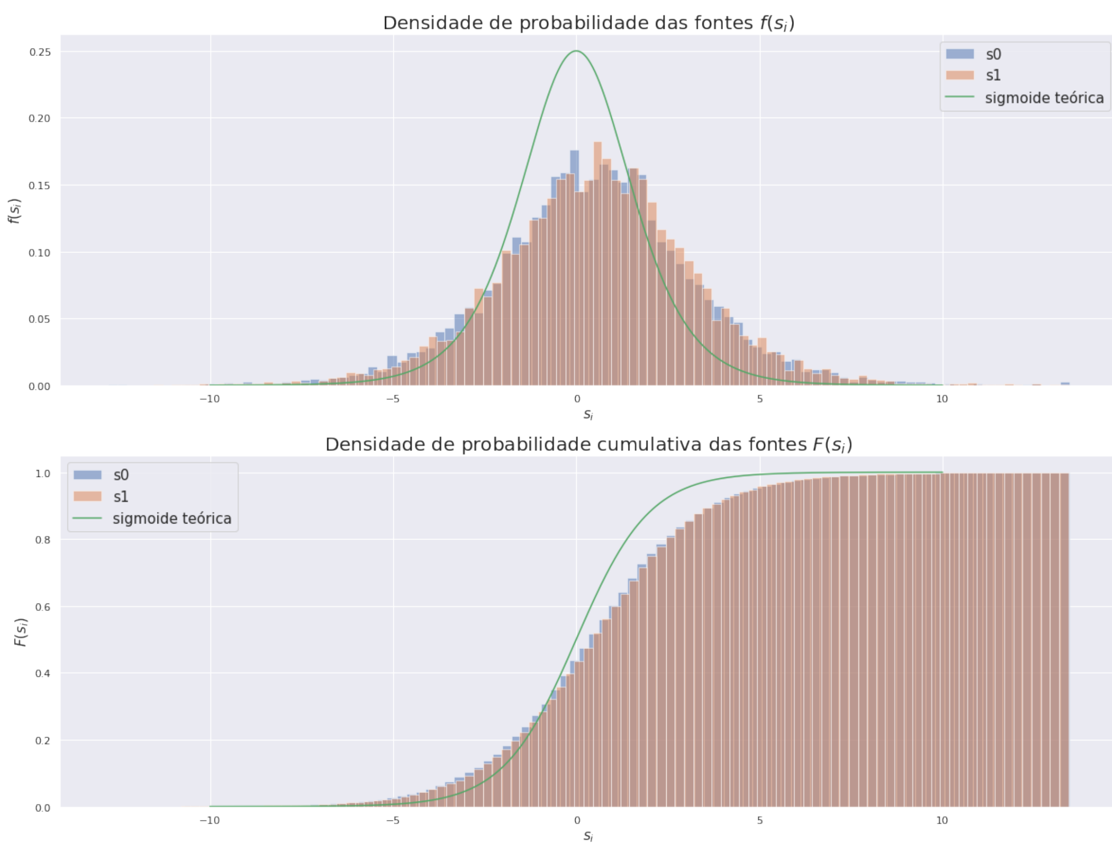
[108]:
```python
NSOURCES=2
NOBS=7500
MU=0.5
SCALE=1.5

s = source_generator(
    nsources=NSOURCES,
    nobs=NOBS,
    mu=MU,
    scale=SCALE
)

print('-'*100)
print('NÚMERO DE FONTES: {}'.format(NSOURCES))
print('TAMANHO DOS SINAIS DAS FONTES: {}'.format(NOBS))
for i in range(s.shape[0]):
    print(
        'CURTOSE s{}: {}'.format(
            i,
            st.kurtosis(a=s[i,:])
        )
    )

print('-'*100)
```

```
--------------------------------------------------------------------------------
--------------------
NÚMERO DE FONTES: 2
TAMANHO DOS SINAIS DAS FONTES: 7500
CURTOSE s0: 1.2352238713024697
CURTOSE s1: 1.1583051619319455
--------------------------------------------------------------------------------
--------------------
```

[109]:

```
fig, (ax1, ax2) = plt.subplots(
    nrows=2, ncols=1,
    figsize=(20,15)
)


for i in range(1, NSOURCES+1):
    ax1.hist(
        x=s[i-1,:],
        bins=100,
        density=True,
        label='s{}'.format(i-1),
        alpha=0.5
    )
    ax2.hist(
        x=s[i-1,:],
        bins=100,
        cumulative=True,
        density=True,
        label='s{}'.format(i-1),
        alpha=0.5
    )

ax1.plot(
    np.linspace(-10,10,1000),
    [source_pdf(x) for x in np.linspace(-10,10,1000)],
    label='sigmoide teórica'
)


ax2.plot(
    np.linspace(-10,10,1000),
    [source_cumulative(x) for x in np.linspace(-10,10,1000)],
    label='sigmoide teórica'
)

ax1.set_xlabel(
    '$s_{i}$',
    fontsize=15
)
ax1.set_ylabel(
    '$f(s_{i})$',
    fontsize=15
)
ax1.set_title(
    'Densidade de probabilidade das fontes $f(s_{i})$',
    fontsize=20
```

```
)

ax2.set_xlabel(
    '$s_{i}$',
    fontsize=15
)
ax2.set_ylabel(
    '$F(s_{i})$',
    fontsize=15
)
ax2.set_title(
    'Densidade de probabilidade cumulativa das fontes $F(s_{i})$',
    fontsize=20
)

l1=ax1.legend(fontsize=15)
l2=ax2.legend(fontsize=15)
```



6.2 2.2. Mix sources and generate observations

[110]:

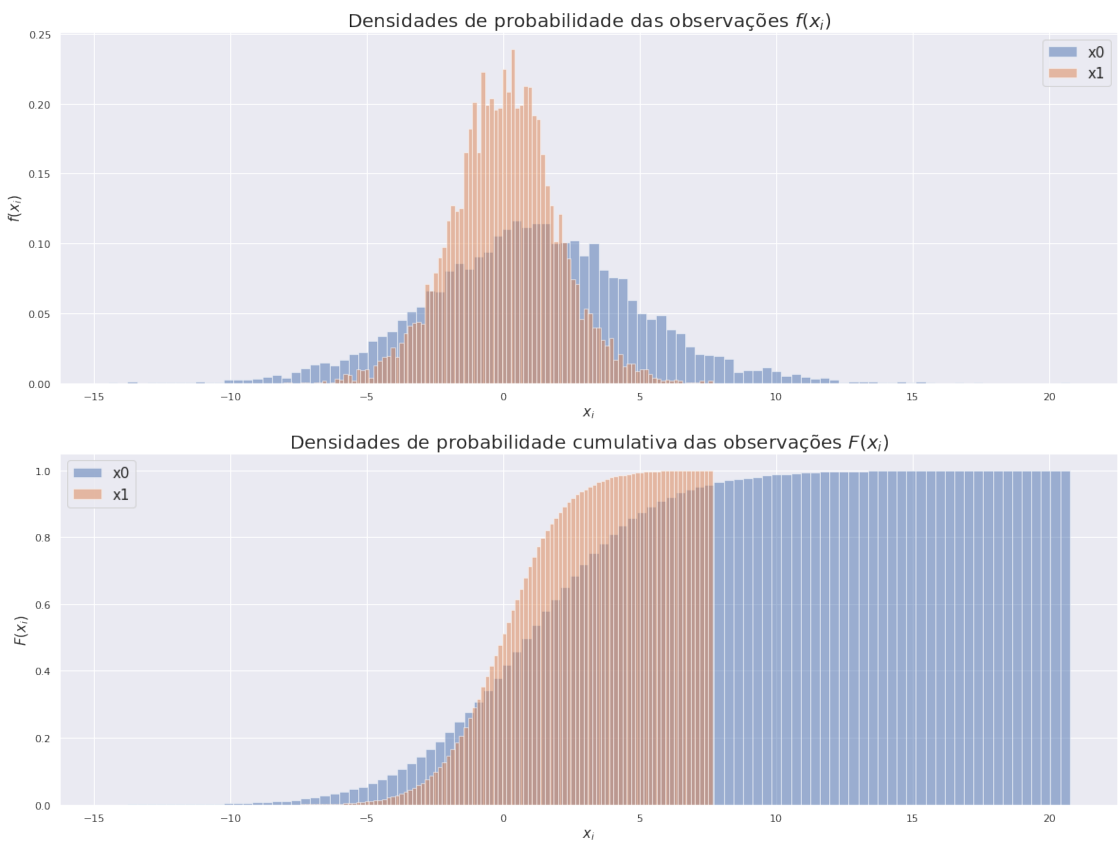
```python
# Mixing matrix
A = np.array([
    [1, 1],
    [-0.5, 0.5]
])
print('MIXING MATRIX A:')
print(A)
```

```
MIXING MATRIX A:
[[ 1.   1. ]
 [-0.5  0.5]]
```

[111]:

```python
# Observed sources
x = A@s

fig, (ax1, ax2) = plt.subplots(
    nrows=2, ncols=1,
    figsize=(20,15)
)

for i in range(1, NSOURCES+1):
    ax1.hist(
        x=x[i-1,:],
        bins=100,
        density=True,
        label='x{}'.format(i-1),
        alpha=0.5
    )
    ax2.hist(
        x=x[i-1,:],
        bins=100,
        cumulative=True,
        density=True,
        label='x{}'.format(i-1),
        alpha=0.5
    )

ax1.set_xlabel(
    '$x_{i}$',
    fontsize=15
)
ax1.set_ylabel(
    '$f(x_{i})$',
    fontsize=15
)
```

```
ax1.set_title(
    'Densidades de probabilidade das observações $f(x_{i})$',
    fontsize=20
)

ax2.set_xlabel(
    '$x_{i}$',
    fontsize=15
)
ax2.set_ylabel(
    '$F(x_{i})$',
    fontsize=15
)
ax2.set_title(
    'Densidades de probabilidade cumulativa das observações $F(x_{i})$',
    fontsize=20
)

l1=ax1.legend(fontsize=15)
l2=ax2.legend(fontsize=15)
```



6.3 2.3. Performance Curves

[112]:
```python
def get_posteriori_neighborhood(
    A,
    x,
    T,
    u_vec,
    v_vec,
    n_points,
    source_pdf_fn,
    prior_pdf_fn,
    njobs=1
):

    def __get_posteriori(
        A,
        x,
        T,
        source_pdf_fn,
        prior_pdf_fn,
        idx_info
    ):
        i, u = idx_info[0]
        j, v = idx_info[-1]

        A_shifted = A + u*symmetric_basis + v*skew_symmetric_basis

        B = np.linalg.inv(A_shifted)

        likelihood = np.log(np.abs(np.linalg.det(B)))

        likelihood_iterator = [
            (t,i) for t,i in itertools.product(
                range(x.shape[-1]),
                range(x.shape[0]))
        ]

        s_est = B@x

        likelihood += (1/T)*np.sum([
            np.log(source_pdf_fn(s_est[i,t])) for t,i in likelihood_iterator
        ])

        prior = np.log(prior_pdf_fn(B))/T

        return {
            'i': i,
            'j': j,
```

```
            'log_posteriori': likelihood + prior
        }


    symmetric_basis = np.array([
        [0, 1],
        [1, 0]
    ])

    skew_symmetric_basis = np.array([
        [0, -1],
        [1, 0]
    ])

    iterator = itertools.product(
        enumerate(u_vec),
        enumerate(v_vec)
    )
    # it = [x for x in iterator]

    exec_fn = partial(
        __get_posteriori,
        A,
        x,
        T,
        source_pdf_fn,
        prior_pdf_fn
    )

    z = np.empty(
        shape=(
            u_vec.shape[0],
            v_vec.shape[0]
        )
    )
    with pathos.multiprocessing.ProcessingPool(njobs) as p:
        results = p.map(exec_fn, iterator)

    for r in results:
        i=r['i']
        j=r['j']
        z[i, j]=r['log_posteriori']

    return z
```

6.3.1 2.3.1. Likelihood

[113]:
```
def source_pdf(x):
    return np.exp(-x)/np.square(1+np.exp(-x))

def prior_pdf(B):
    return 1
```

[114]:
```
%%time

u_lims = (-1, 1)
v_lims = (-1, 1)
n_points = 200

u_vec = np.linspace(u_lims[0], u_lims[-1], n_points)
v_vec = np.linspace(v_lims[0], v_lims[-1], n_points)

z = get_posteriori_neighborhood(
    A=A,
    x=x,
    T=NOBS,
    u_vec=u_vec,
    v_vec=v_vec,
    n_points=n_points,
    source_pdf_fn=source_pdf,
    prior_pdf_fn=prior_pdf,
    njobs=os.cpu_count()-1
)
```

```
---------------------------------------------------------------------------
RemoteTraceback                           Traceback (most recent call last)
RemoteTraceback:
"""
Traceback (most recent call last):
  File "/home/leonardo/py3env/lib/python3.8/site-packages/multiprocess/pool.py",
 ↪line 125, in worker
    result = (True, func(*args, **kwds))
  File "/home/leonardo/py3env/lib/python3.8/site-packages/multiprocess/pool.py",
 ↪line 48, in mapstar
    return list(map(*args))
  File "/home/leonardo/py3env/lib/python3.8/site-packages/pathos/helpers/
 ↪mp_helper.py", line 15, in <lambda>
    func = lambda args: f(*args)
  File "/tmp/ipykernel_45077/2236618269.py", line 26, in __get_posteriori
    B = np.linalg.inv(A_shifted)
  File "<__array_function__ internals>", line 180, in inv
  File "/home/leonardo/py3env/lib/python3.8/site-packages/numpy/linalg/linalg.
 ↪py", line 552, in inv
```

```
    ainv = _umath_linalg.inv(a, signature=signature, extobj=extobj)
  File "/home/leonardo/py3env/lib/python3.8/site-packages/numpy/linalg/linalg.
↪py", line 89, in _raise_linalgerror_singular
    raise LinAlgError("Singular matrix")
numpy.linalg.LinAlgError: Singular matrix
"""

The above exception was the direct cause of the following exception:

LinAlgError                               Traceback (most recent call last)
<timed exec> in <module>

/tmp/ipykernel_45077/2236618269.py in get_posteriori_neighborhood(A, x, T,␣
↪u_vec, v_vec, n_points, source_pdf_fn, prior_pdf_fn, njobs)
     81     )
     82     with pathos.multiprocessing.ProcessingPool(njobs) as p:
---> 83         results = p.map(exec_fn, iterator)
     84
     85     for r in results:

~/py3env/lib/python3.8/site-packages/pathos/multiprocessing.py in map(self, f,␣
↪*args, **kwds)
    137         AbstractWorkerPool._AbstractWorkerPool__map(self, f, *args,␣
↪**kwds)
    138         _pool = self._serve()
--> 139         return _pool.map(star(f), zip(*args)) # chunksize
    140     map.__doc__ = AbstractWorkerPool.map.__doc__
    141     def imap(self, f, *args, **kwds):

~/py3env/lib/python3.8/site-packages/multiprocess/pool.py in map(self, func,␣
↪iterable, chunksize)
    362         in a list that is returned.
    363         '''
--> 364         return self._map_async(func, iterable, mapstar, chunksize).get(
    365
    366     def starmap(self, func, iterable, chunksize=None):

~/py3env/lib/python3.8/site-packages/multiprocess/pool.py in get(self, timeout)
    769             return self._value
    770         else:
--> 771             raise self._value
    772
    773     def _set(self, i, obj):

LinAlgError: Singular matrix
```

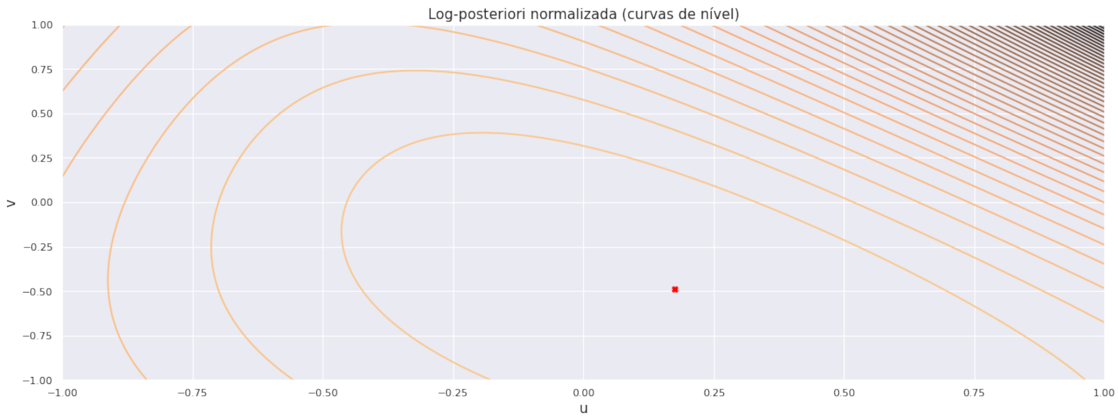
[115]:
```
# Ponto de máximo
max_post_point = np.unravel_index(
    z.argmax(),
    z.shape
)

#
max_post = z[max_post_point]

print('-'*100)
print('Ponto de máximo: u={}, v={}'.format(u_vec[max_post_point[0]],
 ↪v_vec[max_post_point[1]]))
print('-'*100)
```

```
--------------------------------------------------------------------------------
--------------------
Ponto de máximo: u=0.1758793969849246, v=-0.48743718592964824
--------------------------------------------------------------------------------
--------------------
```

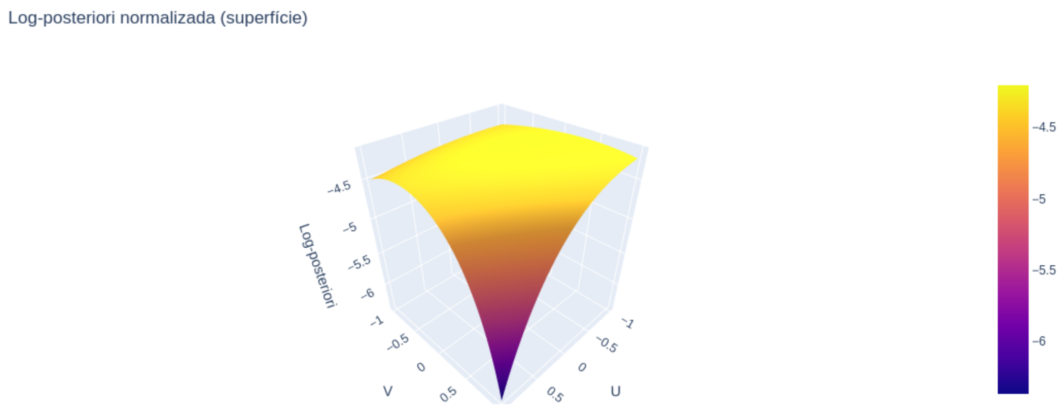
[116]:
```
# Plot de curvas de nível
fig = plt.figure(figsize=(20,7))

U, V = np.meshgrid(u_vec, v_vec)

plt.contour(
    U,
    V,
    z.T,
    levels=50,
    cmap='copper'
)

plt.scatter(
    u_vec[max_post_point[0]],
    v_vec[max_post_point[-1]],
    marker='X',
    color='red'
)

plt.xlabel(
    'u',
    fontsize=15
)
plt.ylabel(
    'v',
    fontsize=15
```

```
)

plt.title(
    'Log-posteriori normalizada (curvas de nível)',
    fontsize=15
)
```

[116]: Text(0.5, 1.0, 'Log-posteriori normalizada (curvas de nível)')



[117]:
```
# Plot interativo
fig = go.Figure(
    go.Surface(
        x=u_vec,
        y=v_vec,
        z=z
    )
)

fig.update_layout(
    title='Log-posteriori normalizada (superfície)',
    autosize=False,
    width=500, height=500,
    scene=dict(
        xaxis_title='U',
        yaxis_title='V',
        zaxis_title='Log-posteriori',
    )
)

fig.show()
```



[118]:
```python
def source_pdf(x):
    """
        This method takes SCALAR value x and return SCALAR source pdf evaluated at x.
    """
    return np.exp(-x)/np.square(1+np.exp(-x))


def source_pdf_derivative(x):
    """
        This method takes SCALAR value x and return SCALAR source pdf derivative evaluated at x.
    """
    return (2*np.exp(-2*x)*(1+np.exp(-x)) - np.exp(-x)*np.square(1+np.exp(-x)))/np.power([1+np.exp(-x)],[4])


def prior_pdf(X):
    """
        This method takes VECTOR value X and return SCALAR prior pdf evaluated at X.
    """
    return 1

def prior_pdf_derivative(X):
    """
        This method takes VECTOR value x and return VECTOR prior pdf derivative evaluated at X.
    """
```

```
    return np.zeros(
        shape=X.shape
    )
```

6.4 3.1. Standard Gradient

[119]:
```
# Initialize estimator
estimator = MAPGradientEstimator(
    learning_rate=learning_rate,
    thresh=thresh,
    max_it=max_it,
    source_pdf=source_pdf,
    source_pdf_derivative=source_pdf_derivative,
    prior_pdf=prior_pdf,
    prior_pdf_derivative=prior_pdf_derivative,
    is_natural_gradient=False
)
```

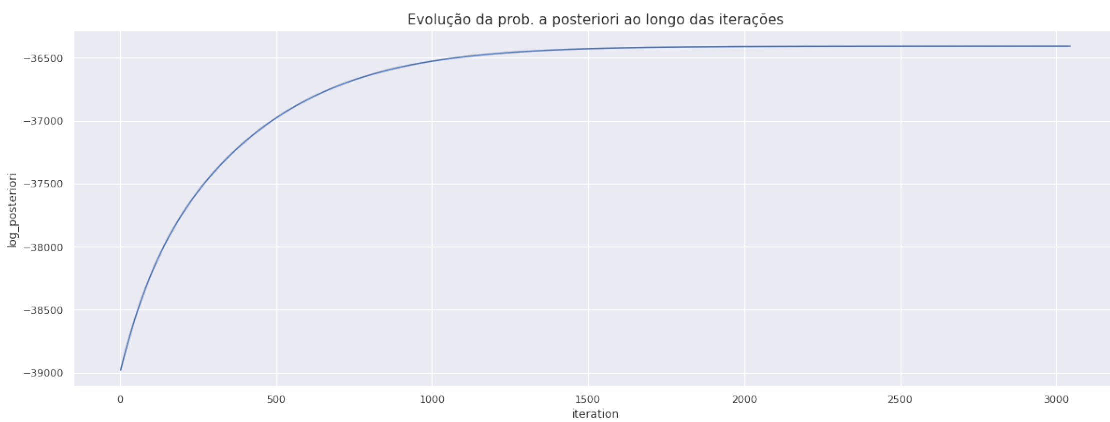
[120]:
```
%%time

# Run optimization
estimator.fit(
    s=s,
    x=x,
    initial_condition=initial_B,
    n_initializations=1,
    n_jobs=os.cpu_count()-1
)

# Parse results
B=estimator.fit_results[0]['unmixing_matrix']
logs=estimator.fit_results[0]['logs']

print('-'*100)
print('Total iterations: {}'.format(logs.shape[0]))
print('Final log-posteriori: {}'.format(logs.iloc[-1]['log_posteriori']))
print('-'*100)
```

```
/home/leonardo/git/bayesian_bss/src/estimator.py:144: FutureWarning:

elementwise comparison failed; returning scalar instead, but in the future will
perform elementwise comparison


--------------------------------------------------------------------------------
--------------------
Total iterations: 3044
Final log-posteriori: -36408.56339734319
```

```
-------------------------------------------------------------------------------
--------------------
CPU times: user 15min 11s, sys: 2min 43s, total: 17min 55s
Wall time: 12min 36s
```

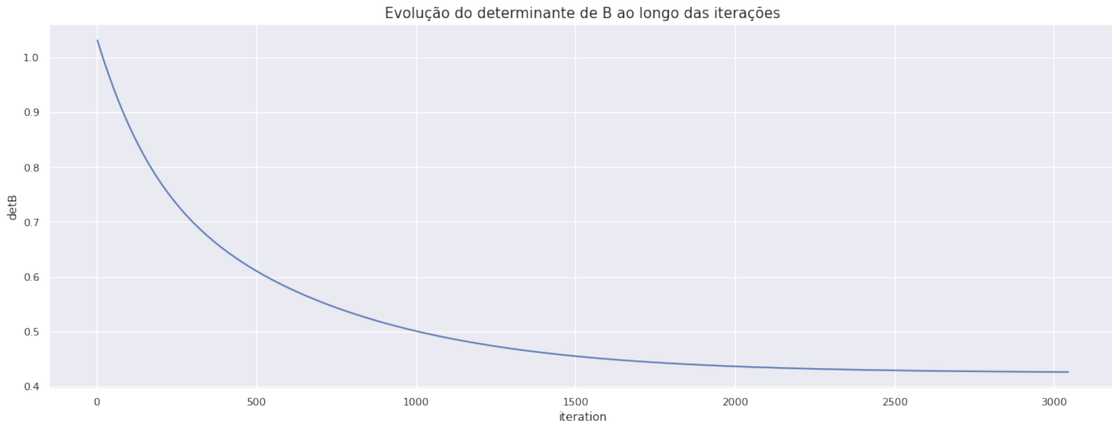
[121]:
```python
fig = plt.figure(figsize=(20,7))
sns.lineplot(
    data=logs,
    x='iteration',
    y='log_posteriori'
)
plt.title(
    'Evolução da prob. a posteriori ao longo das iterações',
    fontsize=15
)
```

[121]:
```
Text(0.5, 1.0, 'Evolução da prob. a posteriori ao longo das iterações')
```



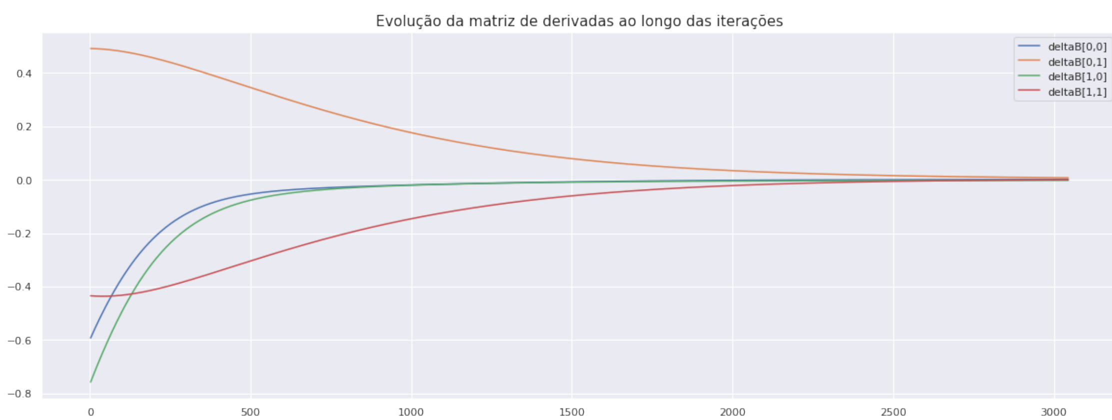
[122]:
```python
fig = plt.figure(figsize=(20,7))
sns.lineplot(
    data=logs,
    x='iteration',
    y='detB'
)
plt.title(
    'Evolução do determinante de B ao longo das iterações',
    fontsize=15
)
```

[122]:
```
Text(0.5, 1.0, 'Evolução do determinante de B ao longo das iterações')
```



[123]:
```
fig = plt.figure(figsize=(20,7))
for i, j in np.ndindex(B.shape):
    plt.plot(
        logs.iteration.values,
        [row['gradient'][i,j] for n, row in logs.iterrows()],
        label='deltaB[{},{}]'.format(i,j)
    )

plt.title(
    'Evolução da matriz de derivadas ao longo das iterações',
    fontsize=15
)
plt.legend()
```

[123]: <matplotlib.legend.Legend at 0x7f60afbd68b0>



[124]:
```
B
```

[124]: array([[0.3052715 , -0.69353207],
       [0.33772114, 0.62930106]])

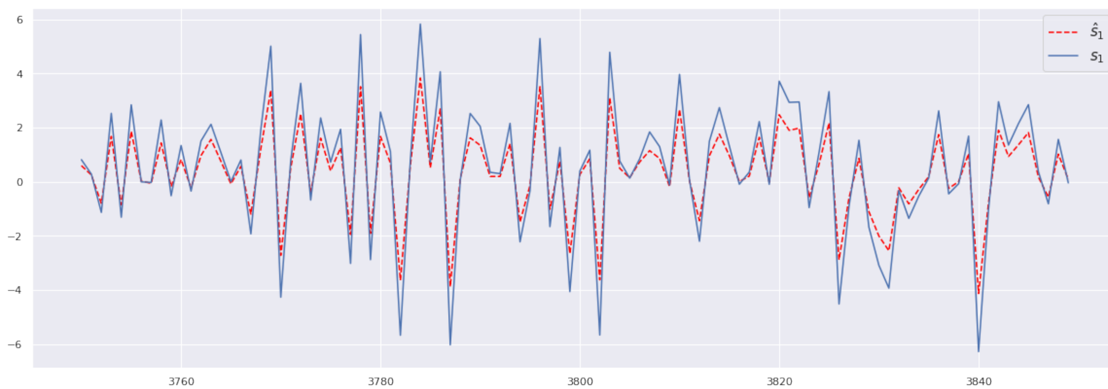
[125]:
```
np.linalg.inv(A)
```

[125]: array([[0.5, -1.],
       [0.5, 1.]])

[126]:
```
fig, (ax1, ax2) = plt.subplots(
    nrows=2, ncols=1,
    figsize=(20,15)
)
NPOINTS=100
PLOT_START=NOBS//2
PLOT_END=PLOT_START+NPOINTS

t=range(NOBS)
s_est = B@x

#Ax1
ax1.plot(
    t[PLOT_START:PLOT_END],
    s_est[0,PLOT_START:PLOT_END],
    label='$\hat{s}_{0}$',
    color='red',
    linestyle='--'
)
ax1.plot(
    t[PLOT_START:PLOT_END],
    s[0,PLOT_START:PLOT_END],
    label='$s_{0}$'
)
ax1.legend(fontsize=15)

#Ax2
ax2.plot(
    t[PLOT_START:PLOT_END],
    s_est[1,PLOT_START:PLOT_END],
    label='$\hat{s}_{1}$',
    color='red',
    linestyle='--'
)
ax2.plot(
    t[PLOT_START:PLOT_END],
    s[1,PLOT_START:PLOT_END],
    label='$s_{1}$'
)
```

```
ax2.legend(fontsize=15)
```

[126]: <matplotlib.legend.Legend at 0x7f60afb01160>



[127]:
```
# Error norm
print('Norma do erro de estimação: {}'.format(
np.linalg.norm(
    np.subtract(s,s_est)
)/np.size(s)))
```

```
Norma do erro de estimação: 0.007931396230619118
```

6.4.1 2.3.2. MAP - determinant equal to one

[128]:
```
def source_pdf(x):
    return np.exp(-x)/np.square(1+np.exp(-x))

def prior_pdf(B):
    sig=0.1
    desired_det=1
```

```
    return (1/np.sqrt(2*np.pi*np.square(sig)))*np.exp(-np.square(np.linalg.
↪det(B)-desired_det)/(2*np.square(sig)))
```

[129]:
```
%%time

u_lims = (-1, 1)
v_lims = (-1, 1)
n_points = 200

u_vec = np.linspace(u_lims[0], u_lims[-1], n_points)
v_vec = np.linspace(v_lims[0], v_lims[-1], n_points)

z = get_posteriori_neighborhood(
    A=A,
    x=x,
    T=NOBS,
    u_vec=u_vec,
    v_vec=v_vec,
    n_points=n_points,
    source_pdf_fn=source_pdf,
    prior_pdf_fn=prior_pdf,
    njobs=os.cpu_count()-1
)
```

```
---------------------------------------------------------------------------
RemoteTraceback                           Traceback (most recent call last)
RemoteTraceback:
"""
Traceback (most recent call last):
  File "/home/leonardo/py3env/lib/python3.8/site-packages/multiprocess/pool.py"
↪line 125, in worker
    result = (True, func(*args, **kwds))
  File "/home/leonardo/py3env/lib/python3.8/site-packages/multiprocess/pool.py"
↪line 48, in mapstar
    return list(map(*args))
  File "/home/leonardo/py3env/lib/python3.8/site-packages/pathos/helpers/
↪mp_helper.py", line 15, in <lambda>
    func = lambda args: f(*args)
  File "/tmp/ipykernel_45077/2236618269.py", line 26, in __get_posteriori
    B = np.linalg.inv(A_shifted)
  File "<__array_function__ internals>", line 180, in inv
  File "/home/leonardo/py3env/lib/python3.8/site-packages/numpy/linalg/linalg.
↪py", line 552, in inv
    ainv = _umath_linalg.inv(a, signature=signature, extobj=extobj)
  File "/home/leonardo/py3env/lib/python3.8/site-packages/numpy/linalg/linalg.
↪py", line 89, in _raise_linalgerror_singular
    raise LinAlgError("Singular matrix")
```

```
numpy.linalg.LinAlgError: Singular matrix
"""

The above exception was the direct cause of the following exception:

LinAlgError                               Traceback (most recent call last)
<timed exec> in <module>

/tmp/ipykernel_45077/2236618269.py in get_posteriori_neighborhood(A, x, T,␣
 ↪u_vec, v_vec, n_points, source_pdf_fn, prior_pdf_fn, njobs)
     81     )
     82     with pathos.multiprocessing.ProcessingPool(njobs) as p:
---> 83         results = p.map(exec_fn, iterator)
     84 
     85     for r in results:

~/py3env/lib/python3.8/site-packages/pathos/multiprocessing.py in map(self, f,␣
 ↪*args, **kwds)
    137         AbstractWorkerPool._AbstractWorkerPool__map(self, f, *args,␣
 ↪**kwds)
    138         _pool = self._serve()
--> 139         return _pool.map(star(f), zip(*args)) # chunksize
    140     map.__doc__ = AbstractWorkerPool.map.__doc__
    141     def imap(self, f, *args, **kwds):

~/py3env/lib/python3.8/site-packages/multiprocess/pool.py in map(self, func,␣
 ↪iterable, chunksize)
    362         in a list that is returned.
    363         '''
--> 364         return self._map_async(func, iterable, mapstar, chunksize).get(
    365 
    366     def starmap(self, func, iterable, chunksize=None):

~/py3env/lib/python3.8/site-packages/multiprocess/pool.py in get(self, timeout)
    769             return self._value
    770         else:
--> 771             raise self._value
    772 
    773     def _set(self, i, obj):

LinAlgError: Singular matrix
```

[130]:
```
# Ponto de máximo
max_post_point = np.unravel_index(
    z.argmax(),
    z.shape
)
```

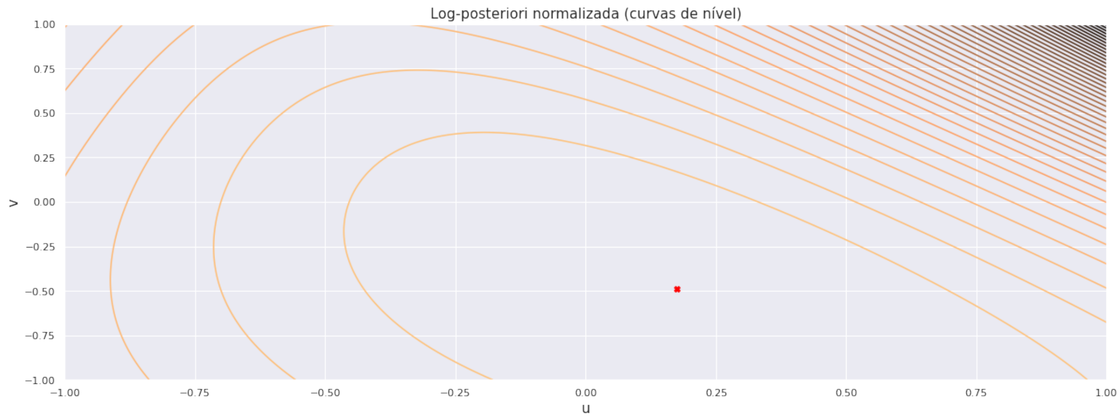
```
#
max_post = z[max_post_point]

print('-'*100)
print('Ponto de máximo: u={}, v={}'.format(u_vec[max_post_point[0]],
 ↪v_vec[max_post_point[1]]))
print('-'*100)
```

```
--------------------------------------------------------------------------------
--------------------
Ponto de máximo: u=0.1758793969849246, v=-0.48743718592964824
--------------------------------------------------------------------------------
--------------------
```

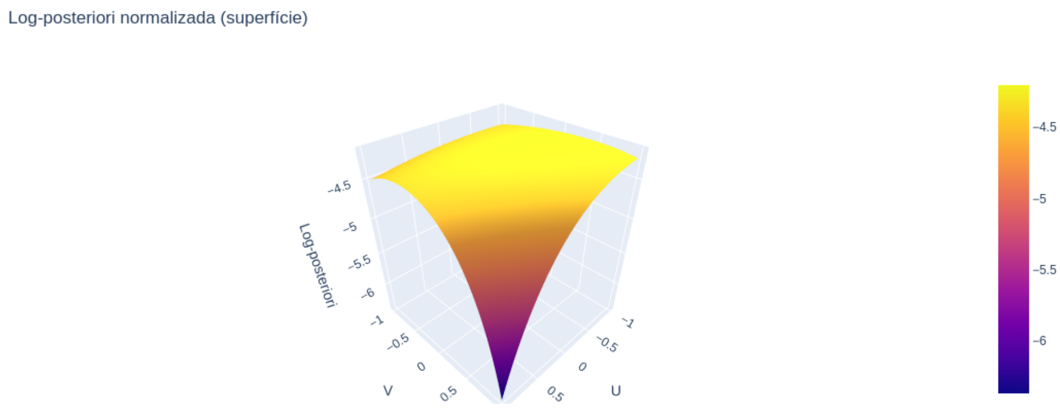
[131]:
```
# Plot de curvas de nível
fig = plt.figure(figsize=(20,7))

U, V = np.meshgrid(u_vec, v_vec)

plt.contour(
    U,
    V,
    z.T,
    levels=50,
    cmap='copper'
)

plt.scatter(
    u_vec[max_post_point[0]],
    v_vec[max_post_point[-1]],
    marker='X',
    color='red'
)

plt.xlabel(
    'u',
    fontsize=15
)
plt.ylabel(
    'v',
    fontsize=15
)

plt.title(
    'Log-posteriori normalizada (curvas de nível)',
    fontsize=15
```

```
)
```

[131]: Text(0.5, 1.0, 'Log-posteriori normalizada (curvas de nível)')



[132]:
```
# Plot interativo
fig = go.Figure(
    go.Surface(
        x=u_vec,
        y=v_vec,
        z=z
    )
)

fig.update_layout(
    title='Log-posteriori normalizada (superfície)',
    autosize=False,
    width=500, height=500,
    scene=dict(
        xaxis_title='U',
        yaxis_title='V',
        zaxis_title='Log-posteriori',
    )
)

fig.show()
```



[133]:
```python
def source_pdf(x):
    """
        This method takes SCALAR value x and return SCALAR source pdf evaluated at x.
    """
    return np.exp(-x)/np.square(1+np.exp(-x))


def source_pdf_derivative(x):
    """
        This method takes SCALAR value x and return SCALAR source pdf derivative evaluated at x.
    """
    return (2*np.exp(-2*x)*(1+np.exp(-x)) - np.exp(-x)*np.square(1+np.exp(-x)))/np.power([1+np.exp(-x)],[4])

def prior_pdf(X):
    sig=0.1
    desired_det=1
    return (1/np.sqrt(2*np.pi*np.square(sig)))*np.exp(-np.square(np.linalg.det(X)-desired_det)/(2*np.square(sig)))

def prior_pdf_derivative(X):
    sig=0.1
    return -1*prior_pdf(X)*(np.linalg.det(X)-1)/np.square(sig)
```

6.5 3.1. Standard Gradient

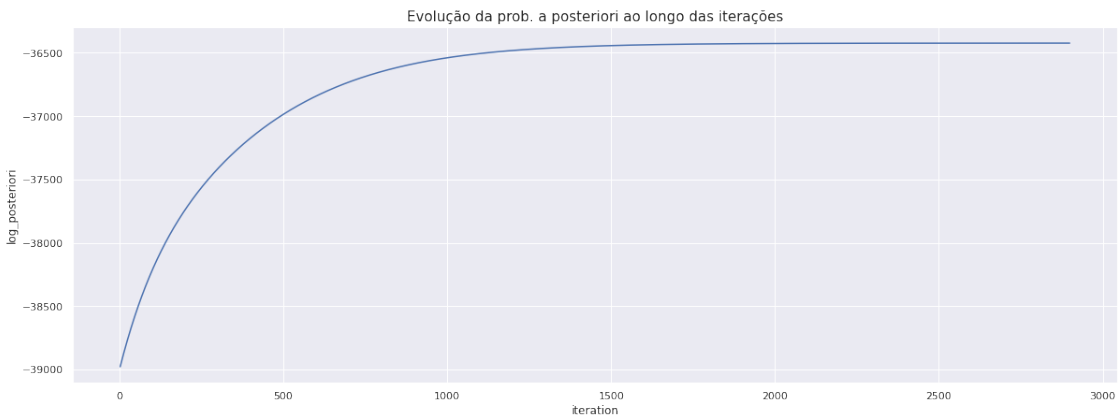
[134]:
```
# Initialize estimator
estimator = MAPGradientEstimator(
    learning_rate=learning_rate,
    thresh=thresh,
    max_it=max_it,
    source_pdf=source_pdf,
    source_pdf_derivative=source_pdf_derivative,
    prior_pdf=prior_pdf,
    prior_pdf_derivative=prior_pdf_derivative,
    is_natural_gradient=False
)
```

[135]:
```
%%time

# Run optimization
estimator.fit(
    s=s,
    x=x,
    initial_condition=initial_B,
    n_initializations=1,
    n_jobs=os.cpu_count()-1
)

# Parse results
B=estimator.fit_results[0]['unmixing_matrix']
logs=estimator.fit_results[0]['logs']

print('-'*100)
print('Total iterations: {}'.format(logs.shape[0]))
print('Final log-posteriori: {}'.format(logs.iloc[-1]['log_posteriori']))
print('-'*100)
```

```
/home/leonardo/git/bayesian_bss/src/estimator.py:144: FutureWarning:

elementwise comparison failed; returning scalar instead, but in the future will
perform elementwise comparison


--------------------------------------------------------------------------------
--------------------
Total iterations: 2898
Final log-posteriori: -36423.03636011298
--------------------------------------------------------------------------------
--------------------
CPU times: user 14min 24s, sys: 2min 36s, total: 17min
Wall time: 11min 57s
```

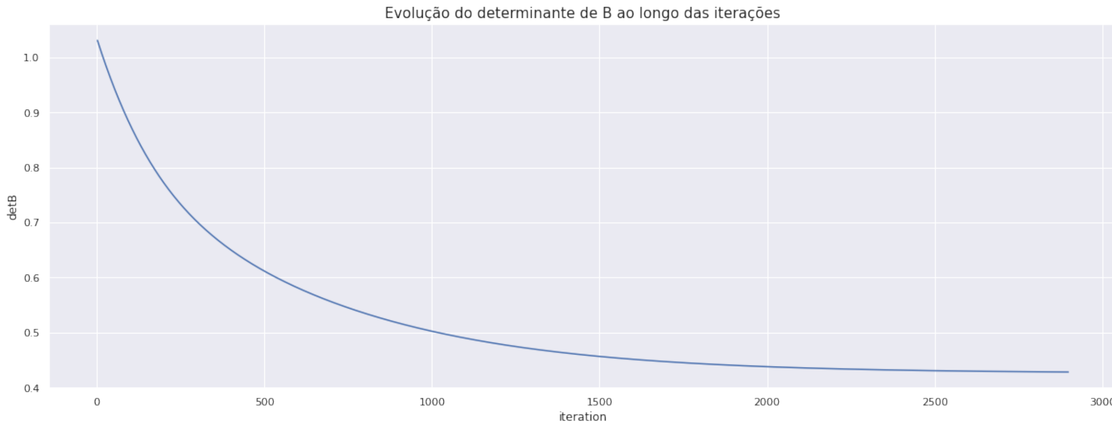
[136]:
```
fig = plt.figure(figsize=(20,7))
sns.lineplot(
    data=logs,
    x='iteration',
    y='log_posteriori'
)
plt.title(
    'Evolução da prob. a posteriori ao longo das iterações',
    fontsize=15
)
```

[136]: Text(0.5, 1.0, 'Evolução da prob. a posteriori ao longo das iterações')



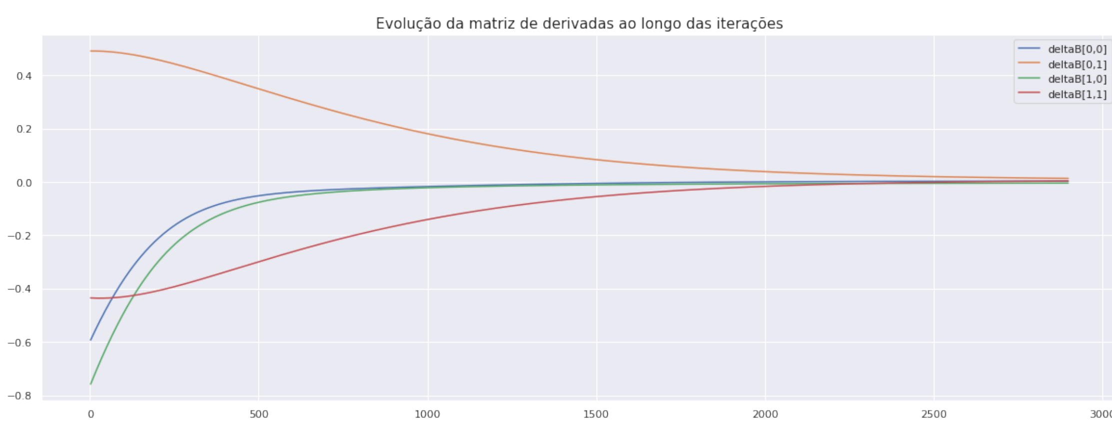
[137]:
```
fig = plt.figure(figsize=(20,7))
sns.lineplot(
    data=logs,
    x='iteration',
    y='detB'
)
plt.title(
    'Evolução do determinante de B ao longo das iterações',
    fontsize=15
)
```

[137]: Text(0.5, 1.0, 'Evolução do determinante de B ao longo das iterações')




```
[138]: fig = plt.figure(figsize=(20,7))
for i, j in np.ndindex(B.shape):
    plt.plot(
        logs.iteration.values,
        [row['gradient'][i,j] for n, row in logs.iterrows()],
        label='deltaB[{},{}]'.format(i,j)
    )

plt.title(
    'Evolução da matriz de derivadas ao longo das iterações',
    fontsize=15
)
plt.legend()
```

```
[138]: <matplotlib.legend.Legend at 0x7f60b1fb3d00>
```




```
[139]: B
```

[139]: array([[0.31114747, -0.68278962],
       [0.33431782, 0.64267774]])

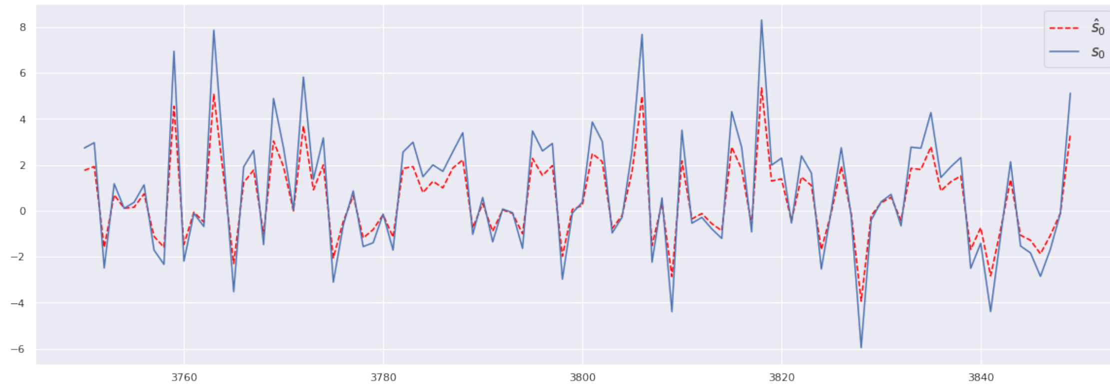
[140]:
```
np.linalg.inv(A)
```

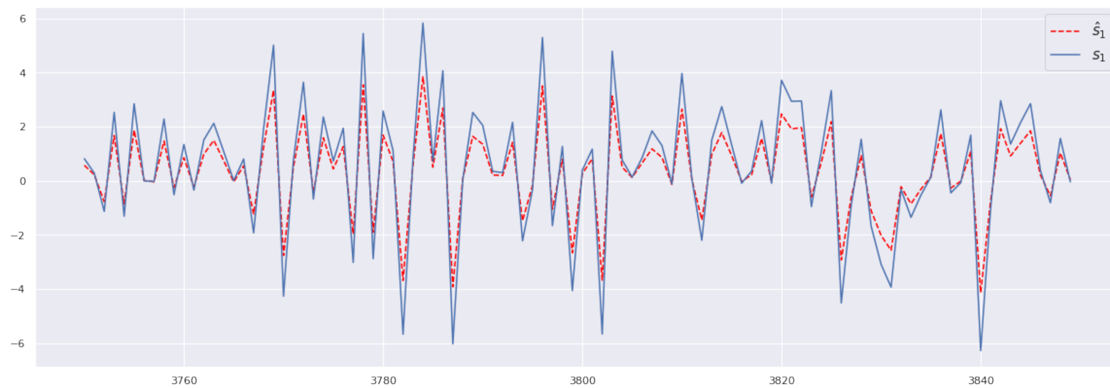
[140]: array([[0.5, -1.],
       [0.5, 1.]])

[141]:
```
fig, (ax1, ax2) = plt.subplots(
    nrows=2, ncols=1,
    figsize=(20,15)
)
NPOINTS=100
PLOT_START=NOBS//2
PLOT_END=PLOT_START+NPOINTS

t=range(NOBS)
s_est = B@x

#Ax1
ax1.plot(
    t[PLOT_START:PLOT_END],
    s_est[0,PLOT_START:PLOT_END],
    label='$\hat{s}_{0}$',
    color='red',
    linestyle='--'
)
ax1.plot(
    t[PLOT_START:PLOT_END],
    s[0,PLOT_START:PLOT_END],
    label='$s_{0}$'
)
ax1.legend(fontsize=15)

#Ax2
ax2.plot(
    t[PLOT_START:PLOT_END],
    s_est[1,PLOT_START:PLOT_END],
    label='$\hat{s}_{1}$',
    color='red',
    linestyle='--'
)
ax2.plot(
    t[PLOT_START:PLOT_END],
    s[1,PLOT_START:PLOT_END],
    label='$s_{1}$'
)
```

```
ax2.legend(fontsize=15)
```

[141]: <matplotlib.legend.Legend at 0x7f60b25f7c70>





[142]:
```
# Error norm
print('Norma do erro de estimação: {}'.format(
np.linalg.norm(
    np.subtract(s,s_est)
)/np.size(s)))
```

```
Norma do erro de estimação: 0.007869652901626422
```

6.6 2.3.3. MAP - near-identity transformation

[143]:
```
def source_pdf(x):
    return np.exp(-x)/np.square(1+np.exp(-x))

def prior_pdf(B):
    sig=0.1
    return np.exp(
        (-1/2/np.square(sig))*np.square(np.linalg.norm(B-np.eye(B.shape[0])))
```

```
    )
```

[144]:

```
%%time

u_lims = (-1, 1)
v_lims = (-1, 1)
n_points = 200

u_vec = np.linspace(u_lims[0], u_lims[-1], n_points)
v_vec = np.linspace(v_lims[0], v_lims[-1], n_points)

z = get_posteriori_neighborhood(
    A=A,
    x=x,
    T=NOBS,
    u_vec=u_vec,
    v_vec=v_vec,
    n_points=n_points,
    source_pdf_fn=source_pdf,
    prior_pdf_fn=prior_pdf,
    njobs=os.cpu_count()-1
)
```

```
---------------------------------------------------------------------------
RemoteTraceback                           Traceback (most recent call last)
RemoteTraceback:
"""
Traceback (most recent call last):
  File "/home/leonardo/py3env/lib/python3.8/site-packages/multiprocess/pool.py",
 ↪line 125, in worker
    result = (True, func(*args, **kwds))
  File "/home/leonardo/py3env/lib/python3.8/site-packages/multiprocess/pool.py",
 ↪line 48, in mapstar
    return list(map(*args))
  File "/home/leonardo/py3env/lib/python3.8/site-packages/pathos/helpers/
 ↪mp_helper.py", line 15, in <lambda>
    func = lambda args: f(*args)
  File "/tmp/ipykernel_45077/2236618269.py", line 26, in __get_posteriori
    B = np.linalg.inv(A_shifted)
  File "<__array_function__ internals>", line 180, in inv
  File "/home/leonardo/py3env/lib/python3.8/site-packages/numpy/linalg/linalg.
 ↪py", line 552, in inv
    ainv = _umath_linalg.inv(a, signature=signature, extobj=extobj)
  File "/home/leonardo/py3env/lib/python3.8/site-packages/numpy/linalg/linalg.
 ↪py", line 89, in _raise_linalgerror_singular
    raise LinAlgError("Singular matrix")
numpy.linalg.LinAlgError: Singular matrix
```

```
"""

The above exception was the direct cause of the following exception:

LinAlgError                               Traceback (most recent call last)
<timed exec> in <module>

/tmp/ipykernel_45077/2236618269.py in get_posteriori_neighborhood(A, x, T,␣
↪u_vec, v_vec, n_points, source_pdf_fn, prior_pdf_fn, njobs)
     81     )
     82     with pathos.multiprocessing.ProcessingPool(njobs) as p:
---> 83         results = p.map(exec_fn, iterator)
     84 
     85     for r in results:

~/py3env/lib/python3.8/site-packages/pathos/multiprocessing.py in map(self, f,␣
↪*args, **kwds)
    137         AbstractWorkerPool._AbstractWorkerPool__map(self, f, *args,␣
↪**kwds)
    138         _pool = self._serve()
--> 139         return _pool.map(star(f), zip(*args)) # chunksize
    140     map.__doc__ = AbstractWorkerPool.map.__doc__
    141     def imap(self, f, *args, **kwds):

~/py3env/lib/python3.8/site-packages/multiprocess/pool.py in map(self, func,␣
↪iterable, chunksize)
    362         in a list that is returned.
    363         '''
--> 364         return self._map_async(func, iterable, mapstar, chunksize).get(
    365 
    366     def starmap(self, func, iterable, chunksize=None):

~/py3env/lib/python3.8/site-packages/multiprocess/pool.py in get(self, timeout)
    769             return self._value
    770         else:
--> 771             raise self._value
    772 
    773     def _set(self, i, obj):

LinAlgError: Singular matrix
```

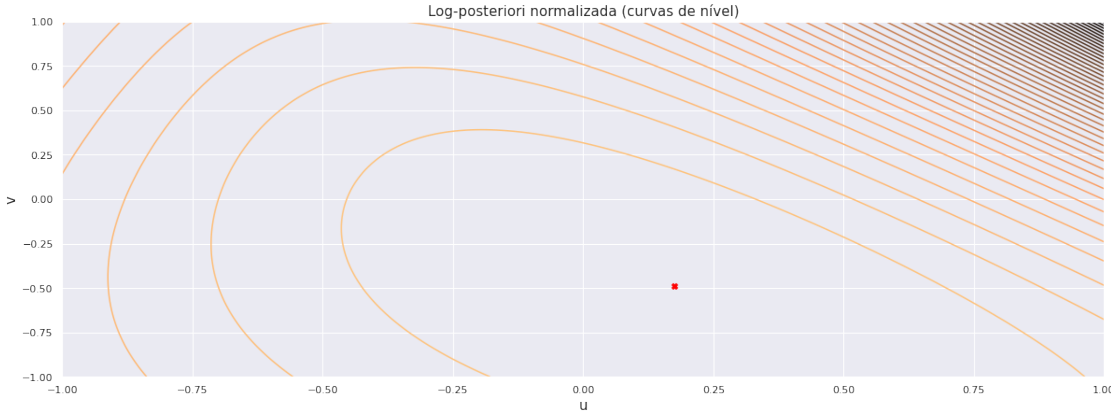
[145]:
```
# Ponto de máximo
max_post_point = np.unravel_index(
    z.argmax(),
    z.shape
)
```

```
#
max_post = z[max_post_point]

print('-'*100)
print('Ponto de máximo: u={}, v={}'.format(u_vec[max_post_point[0]],␣
 ↪v_vec[max_post_point[1]]))
print('-'*100)
```

```
--------------------------------------------------------------------------------
--------------------
Ponto de máximo: u=0.1758793969849246, v=-0.48743718592964824
--------------------------------------------------------------------------------
--------------------
```

[146]:
```
# Plot de curvas de nível
fig = plt.figure(figsize=(20,7))

U, V = np.meshgrid(u_vec, v_vec)

plt.contour(
    U,
    V,
    z.T,
    levels=50,
    cmap='copper'
)

plt.scatter(
    u_vec[max_post_point[0]],
    v_vec[max_post_point[-1]],
    marker='X',
    color='red'
)

plt.xlabel(
    'u',
    fontsize=15
)
plt.ylabel(
    'v',
    fontsize=15
)

plt.title(
    'Log-posteriori normalizada (curvas de nível)',
    fontsize=15
)
```

[146]: Text(0.5, 1.0, 'Log-posteriori normalizada (curvas de nível)')



[147]:
```
# Plot interativo
fig = go.Figure(
    go.Surface(
        x=u_vec,
        y=v_vec,
        z=z
    )
)

fig.update_layout(
    title='Log-posteriori normalizada (superfície)',
    autosize=False,
    width=500, height=500,
    scene=dict(
        xaxis_title='U',
        yaxis_title='V',
        zaxis_title='Log-posteriori',
    )
)

fig.show()
```

[148]:
```
def source_pdf(x):
    """
        This method takes SCALAR value x and return SCALAR source pdf evaluated␣
 ↪at x.
    """
    return np.exp(-x)/np.square(1+np.exp(-x))
```

```
def source_pdf_derivative(x):
    """
        This method takes SCALAR value x and return SCALAR source pdf␣
 ↪derivative evaluated at x.
    """
    return (2*np.exp(-2*x)*(1+np.exp(-x)) - np.exp(-x)*np.square(1+np.exp(-x)))/
 ↪np.power([1+np.exp(-x)],[4])

def prior_pdf(X):
    sig=0.1
    return np.exp(
        (-1/2/np.square(sig))*np.linalg.norm(X-np.eye(X.shape[0]))
    )

def prior_pdf_derivative(X):
    sig=0.1
    der_X=np.empty(X.shape)
    I=np.eye(X.shape[0])
    for i, j in np.ndindex(X.shape):
        der_X[i, j] = prior_pdf(X)*(-1/np.square(sig))*(X[i,j]-I[i,j])
    return der_X
```

6.7 3.1. Standard Gradient

[149]:
```
# Initialize estimator
estimator = MAPGradientEstimator(
    learning_rate=learning_rate,
    thresh=thresh,
    max_it=max_it,
    source_pdf=source_pdf,
    source_pdf_derivative=source_pdf_derivative,
    prior_pdf=prior_pdf,
    prior_pdf_derivative=prior_pdf_derivative,
    is_natural_gradient=False
)
```

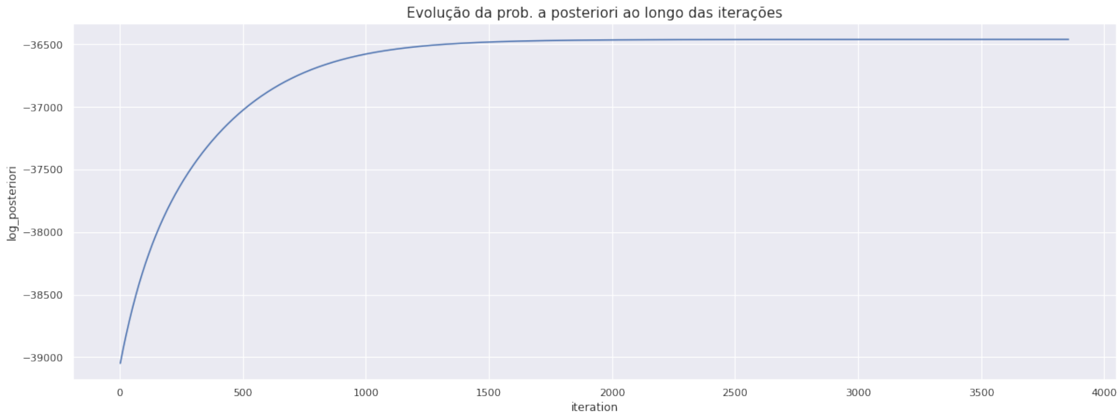
[150]:
```
%%time

# Run optimization
estimator.fit(
    s=s,
    x=x,
    initial_condition=initial_B,
    n_initializations=1,
    n_jobs=os.cpu_count()-1
)
```

```python
# Parse results
B=estimator.fit_results[0]['unmixing_matrix']
logs=estimator.fit_results[0]['logs']

print('-'*100)
print('Total iterations: {}'.format(logs.shape[0]))
print('Final log-posteriori: {}'.format(logs.iloc[-1]['log_posteriori']))
print('-'*100)
```

```
/home/leonardo/git/bayesian_bss/src/estimator.py:144: FutureWarning:

elementwise comparison failed; returning scalar instead, but in the future will
perform elementwise comparison


--------------------------------------------------------------------------------
--------------------
Total iterations: 3854
Final log-posteriori: -36461.013596780285
--------------------------------------------------------------------------------
--------------------
CPU times: user 19min 1s, sys: 3min 29s, total: 22min 30s
Wall time: 15min 46s
```

[151]:
```python
fig = plt.figure(figsize=(20,7))
sns.lineplot(
    data=logs,
    x='iteration',
    y='log_posteriori'
)
plt.title(
    'Evolução da prob. a posteriori ao longo das iterações',
    fontsize=15
)
```

[151]:
```
Text(0.5, 1.0, 'Evolução da prob. a posteriori ao longo das iterações')
```



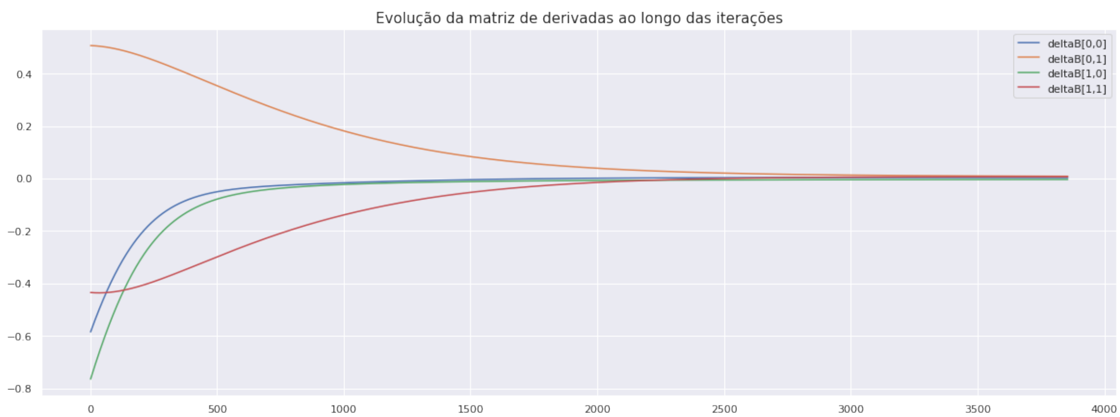
[152]:
```
fig = plt.figure(figsize=(20,7))
sns.lineplot(
    data=logs,
    x='iteration',
    y='detB'
)
plt.title(
    'Evolução do determinante de B ao longo das iterações',
    fontsize=15
)
```

[152]: Text(0.5, 1.0, 'Evolução do determinante de B ao longo das iterações')

[153]:
```
fig = plt.figure(figsize=(20,7))
for i, j in np.ndindex(B.shape):
    plt.plot(
        logs.iteration.values,
```

```
        [row['gradient'][i,j] for n, row in logs.iterrows()],
        label='deltaB[{},{}]'.format(i,j)
    )

plt.title(
    'Evolução da matriz de derivadas ao longo das iterações',
    fontsize=15
)
plt.legend()
```

[153]: <matplotlib.legend.Legend at 0x7f60b68035e0>



[154]:
```
B
```

[154]:
```
array([[ 0.31773882, -0.66644726],
       [ 0.32609563,  0.65252477]])
```

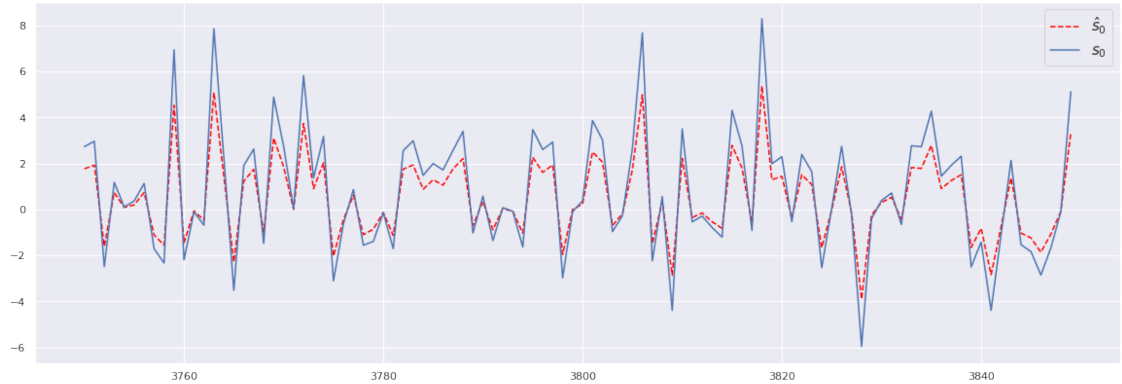
[155]:
```
np.linalg.inv(A)
```

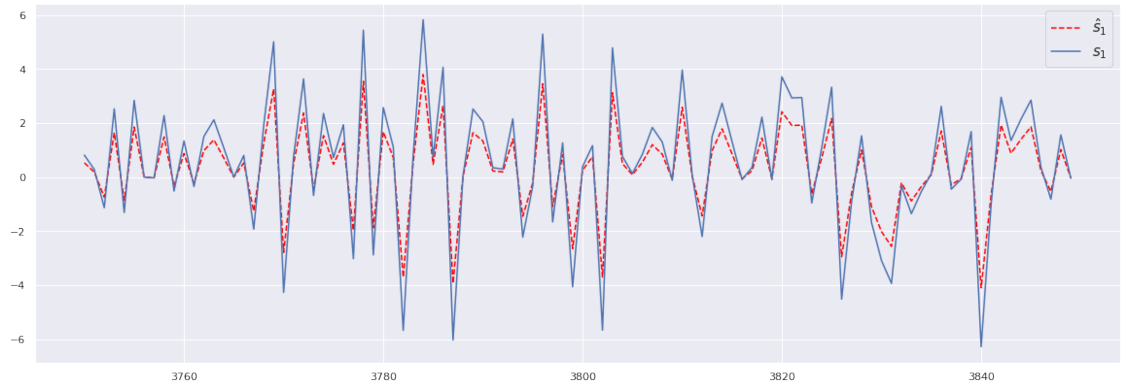
[155]:
```
array([[ 0.5, -1. ],
       [ 0.5,  1. ]])
```

[156]:
```
fig, (ax1, ax2) = plt.subplots(
    nrows=2, ncols=1,
    figsize=(20,15)
)
NPOINTS=100
PLOT_START=NOBS//2
PLOT_END=PLOT_START+NPOINTS

t=range(NOBS)
s_est = B@x
```

```python
#Ax1
ax1.plot(
    t[PLOT_START:PLOT_END],
    s_est[0,PLOT_START:PLOT_END],
    label='$\hat{s}_{0}$',
    color='red',
    linestyle='--'
)
ax1.plot(
    t[PLOT_START:PLOT_END],
    s[0,PLOT_START:PLOT_END],
    label='$s_{0}$'
)
ax1.legend(fontsize=15)

#Ax2
ax2.plot(
    t[PLOT_START:PLOT_END],
    s_est[1,PLOT_START:PLOT_END],
    label='$\hat{s}_{1}$',
    color='red',
    linestyle='--'
)
ax2.plot(
    t[PLOT_START:PLOT_END],
    s[1,PLOT_START:PLOT_END],
    label='$s_{1}$'
)
ax2.legend(fontsize=15)
```

```
[156]: <matplotlib.legend.Legend at 0x7f60ad744c40>
```





[157]:
```python
# Error norm
print('Norma do erro de estimação: {}'.format(
np.linalg.norm(
    np.subtract(s,s_est)
)/np.size(s)))
```

```
Norma do erro de estimação: 0.007910452699809693
```

7 4. LARGELY MISSPECIFIED MODEL - SUPERGAUSSIAN

7.1 3.1. Initialize Sources

[158]:
```python
def source_cumulative(x):
    return 1/(1+np.exp(-x))

def source_pdf(x):
    return np.exp(-x)/np.square(1+np.exp(-x))

def source_generator(
    nsources,
    nobs,
```

```
    mu,
    scale
):
    return np.random.laplace(
        loc=mu,
        scale=scale,
        size=(nsources, nobs)
    )
```

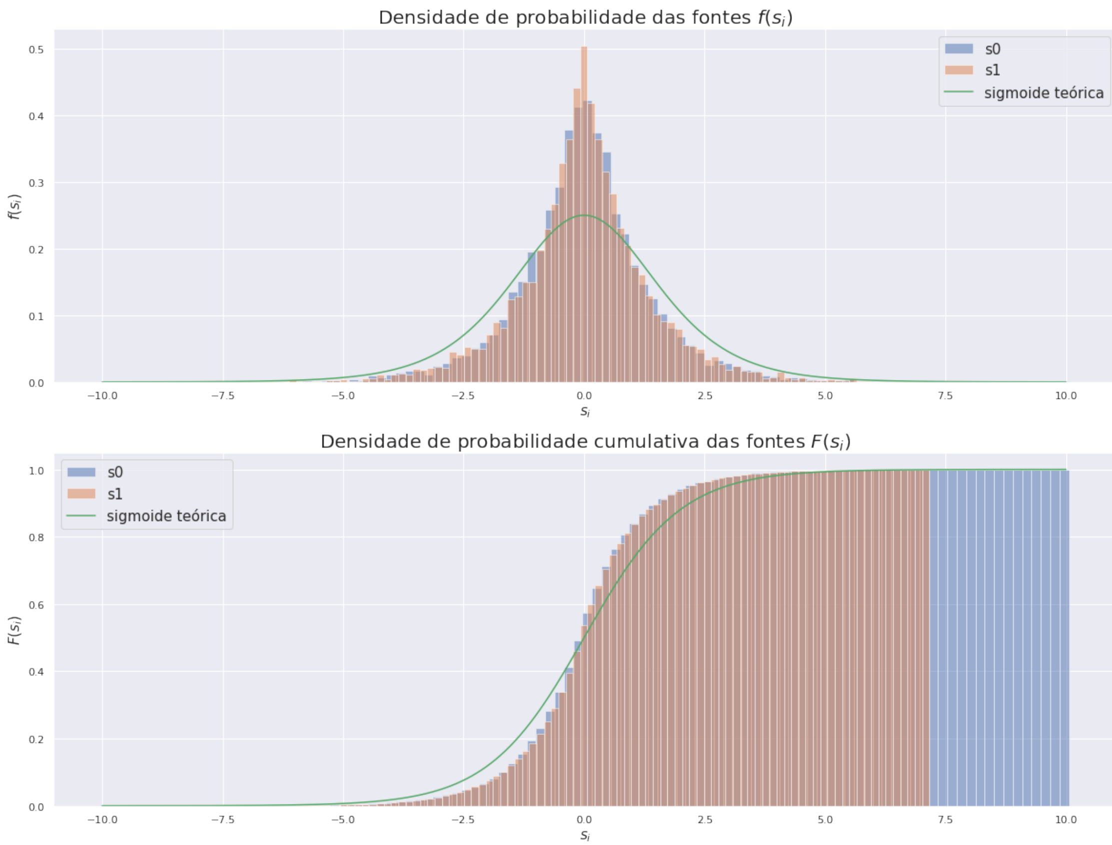
[159]:
```
NSOURCES=2
NOBS=7500
MU=0
SCALE=1

s = source_generator(
    nsources=NSOURCES,
    nobs=NOBS,
    mu=MU,
    scale=SCALE
)

print('-'*100)
print('NÚMERO DE FONTES: {}'.format(NSOURCES))
print('TAMANHO DOS SINAIS DAS FONTES: {}'.format(NOBS))
for i in range(s.shape[0]):
    print(
        'CURTOSE s{}: {}'.format(
            i,
            st.kurtosis(a=s[i,:])
        )
    )

print('-'*100)
```

```
--------------------------------------------------------------------------------
--------------------
NÚMERO DE FONTES: 2
TAMANHO DOS SINAIS DAS FONTES: 7500
CURTOSE s0: 3.229859534811859
CURTOSE s1: 2.483467270108604
--------------------------------------------------------------------------------
--------------------
```

[160]:
```
fig, (ax1, ax2) = plt.subplots(
    nrows=2, ncols=1,
    figsize=(20,15)
)
```

```
for i in range(1, NSOURCES+1):
    ax1.hist(
        x=s[i-1,:],
        bins=100,
        density=True,
        label='s{}'.format(i-1),
        alpha=0.5
    )
    ax2.hist(
        x=s[i-1,:],
        bins=100,
        cumulative=True,
        density=True,
        label='s{}'.format(i-1),
        alpha=0.5
    )

ax1.plot(
    np.linspace(-10,10,1000),
    [source_pdf(x) for x in np.linspace(-10,10,1000)],
    label='sigmoide teórica'
)


ax2.plot(
    np.linspace(-10,10,1000),
    [source_cumulative(x) for x in np.linspace(-10,10,1000)],
    label='sigmoide teórica'
)

ax1.set_xlabel(
    '$s_{i}$',
    fontsize=15
)
ax1.set_ylabel(
    '$f(s_{i})$',
    fontsize=15
)
ax1.set_title(
    'Densidade de probabilidade das fontes $f(s_{i})$',
    fontsize=20
)

ax2.set_xlabel(
    '$s_{i}$',
```

```
    fontsize=15
)
ax2.set_ylabel(
    '$F(s_{i})$',
    fontsize=15
)
ax2.set_title(
    'Densidade de probabilidade cumulativa das fontes $F(s_{i})$',
    fontsize=20
)

l1=ax1.legend(fontsize=15)
l2=ax2.legend(fontsize=15)
```



7.2 2.2. Mix sources and generate observations

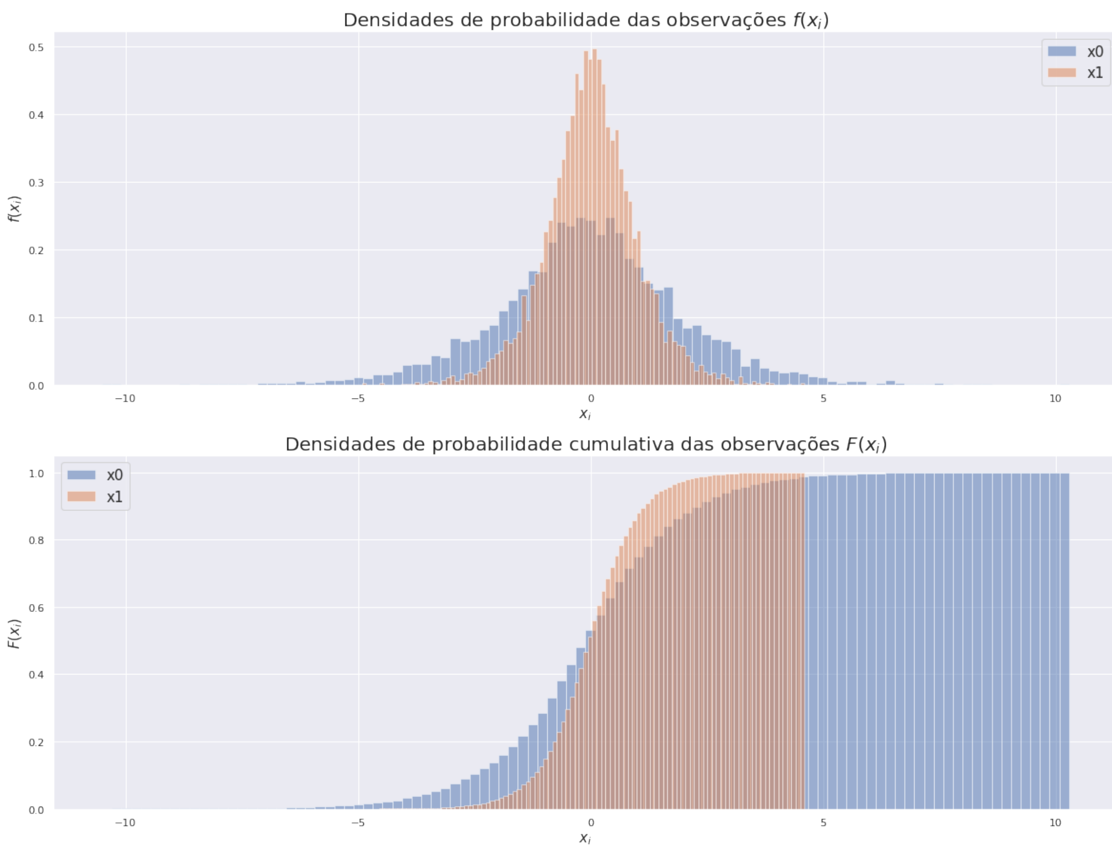
[161]:
```
# Mixing matrix
A = np.array([
    [1, 1],
    [-0.5, 0.5]
```

```
])
print('MIXING MATRIX A:')
print(A)
```

```
MIXING MATRIX A:
[[ 1.   1. ]
 [-0.5  0.5]]
```

[162]:
```
# Observed sources
x = A@s

fig, (ax1, ax2) = plt.subplots(
    nrows=2, ncols=1,
    figsize=(20,15)
)

for i in range(1, NSOURCES+1):
    ax1.hist(
        x=x[i-1,:],
        bins=100,
        density=True,
        label='x{}'.format(i-1),
        alpha=0.5
    )
    ax2.hist(
        x=x[i-1,:],
        bins=100,
        cumulative=True,
        density=True,
        label='x{}'.format(i-1),
        alpha=0.5
    )

ax1.set_xlabel(
    '$x_{i}$',
    fontsize=15
)
ax1.set_ylabel(
    '$f(x_{i})$',
    fontsize=15
)
ax1.set_title(
    'Densidades de probabilidade das observações $f(x_{i})$',
    fontsize=20
)

ax2.set_xlabel(
```

```
    '$x_{i}$',
    fontsize=15
)
ax2.set_ylabel(
    '$F(x_{i})$',
    fontsize=15
)
ax2.set_title(
    'Densidades de probabilidade cumulativa das observações $F(x_{i})$',
    fontsize=20
)

l1=ax1.legend(fontsize=15)
l2=ax2.legend(fontsize=15)
```



7.3 2.3. Performance Curves

[163]:
```
def get_posteriori_neighborhood(
    A,
    x,
    T,
    u_vec,
    v_vec,
    n_points,
    source_pdf_fn,
    prior_pdf_fn,
    njobs=1
):

    def __get_posteriori(
        A,
        x,
        T,
        source_pdf_fn,
        prior_pdf_fn,
        idx_info
    ):
        i, u = idx_info[0]
        j, v = idx_info[-1]

        A_shifted = A + u*symmetric_basis + v*skew_symmetric_basis

        B = np.linalg.inv(A_shifted)

        likelihood = np.log(np.abs(np.linalg.det(B)))

        likelihood_iterator = [
            (t,i) for t,i in itertools.product(
                range(x.shape[-1]),
                range(x.shape[0]))
        ]

        s_est = B@x

        likelihood += (1/T)*np.sum([
            np.log(source_pdf_fn(s_est[i,t])) for t,i in likelihood_iterator
        ])

        prior = np.log(prior_pdf_fn(B))/T

        return {
            'i': i,
            'j': j,
```

```
            'log_posteriori': likelihood + prior
        }


    symmetric_basis = np.array([
        [0, 1],
        [1, 0]
    ])

    skew_symmetric_basis = np.array([
        [0, -1],
        [1, 0]
    ])

    iterator = itertools.product(
        enumerate(u_vec),
        enumerate(v_vec)
    )
    # it = [x for x in iterator]

    exec_fn = partial(
        __get_posteriori,
        A,
        x,
        T,
        source_pdf_fn,
        prior_pdf_fn
    )

    z = np.empty(
        shape=(
            u_vec.shape[0],
            v_vec.shape[0]
        )
    )
    with pathos.multiprocessing.ProcessingPool(njobs) as p:
        results = p.map(exec_fn, iterator)

    for r in results:
        i=r['i']
        j=r['j']
        z[i, j]=r['log_posteriori']

    return z
```

7.3.1 2.3.1. Likelihood

[164]:
```
def source_pdf(x):
    return np.exp(-x)/np.square(1+np.exp(-x))

def prior_pdf(B):
    return 1
```

[165]:
```
%%time

u_lims = (-1, 1)
v_lims = (-1, 1)
n_points = 200

u_vec = np.linspace(u_lims[0], u_lims[-1], n_points)
v_vec = np.linspace(v_lims[0], v_lims[-1], n_points)

z = get_posteriori_neighborhood(
    A=A,
    x=x,
    T=NOBS,
    u_vec=u_vec,
    v_vec=v_vec,
    n_points=n_points,
    source_pdf_fn=source_pdf,
    prior_pdf_fn=prior_pdf,
    njobs=os.cpu_count()-1
)
```

```
---------------------------------------------------------------------------
RemoteTraceback                           Traceback (most recent call last)
RemoteTraceback:
"""
Traceback (most recent call last):
  File "/home/leonardo/py3env/lib/python3.8/site-packages/multiprocess/pool.py"
↪line 125, in worker
    result = (True, func(*args, **kwds))
  File "/home/leonardo/py3env/lib/python3.8/site-packages/multiprocess/pool.py"
↪line 48, in mapstar
    return list(map(*args))
  File "/home/leonardo/py3env/lib/python3.8/site-packages/pathos/helpers/
↪mp_helper.py", line 15, in <lambda>
    func = lambda args: f(*args)
  File "/tmp/ipykernel_45077/2236618269.py", line 26, in __get_posteriori
    B = np.linalg.inv(A_shifted)
  File "<__array_function__ internals>", line 180, in inv
  File "/home/leonardo/py3env/lib/python3.8/site-packages/numpy/linalg/linalg.
↪py", line 552, in inv
```

```
    ainv = _umath_linalg.inv(a, signature=signature, extobj=extobj)
  File "/home/leonardo/py3env/lib/python3.8/site-packages/numpy/linalg/linalg.
↪py", line 89, in _raise_linalgerror_singular
    raise LinAlgError("Singular matrix")
numpy.linalg.LinAlgError: Singular matrix
"""

The above exception was the direct cause of the following exception:

LinAlgError                                Traceback (most recent call last)
<timed exec> in <module>

/tmp/ipykernel_45077/2236618269.py in get_posteriori_neighborhood(A, x, T,␣
↪u_vec, v_vec, n_points, source_pdf_fn, prior_pdf_fn, njobs)
     81     )
     82     with pathos.multiprocessing.ProcessingPool(njobs) as p:
---> 83         results = p.map(exec_fn, iterator)
     84
     85     for r in results:

~/py3env/lib/python3.8/site-packages/pathos/multiprocessing.py in map(self, f,␣
↪*args, **kwds)
    137         AbstractWorkerPool._AbstractWorkerPool__map(self, f, *args,␣
↪**kwds)
    138         _pool = self._serve()
--> 139         return _pool.map(star(f), zip(*args)) # chunksize
    140     map.__doc__ = AbstractWorkerPool.map.__doc__
    141     def imap(self, f, *args, **kwds):

~/py3env/lib/python3.8/site-packages/multiprocess/pool.py in map(self, func,␣
↪iterable, chunksize)
    362         in a list that is returned.
    363         '''
--> 364         return self._map_async(func, iterable, mapstar, chunksize).get(
    365
    366     def starmap(self, func, iterable, chunksize=None):

~/py3env/lib/python3.8/site-packages/multiprocess/pool.py in get(self, timeout)
    769             return self._value
    770         else:
--> 771             raise self._value
    772
    773     def _set(self, i, obj):

LinAlgError: Singular matrix
```

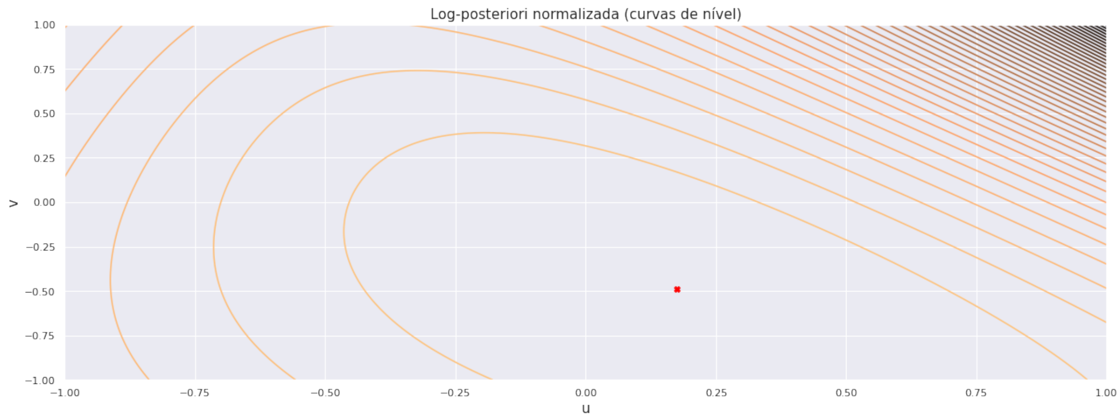
[166]:
```
# Ponto de máximo
max_post_point = np.unravel_index(
    z.argmax(),
    z.shape
)

#
max_post = z[max_post_point]

print('-'*100)
print('Ponto de máximo: u={}, v={}'.format(u_vec[max_post_point[0]],␣
 ↪v_vec[max_post_point[1]]))
print('-'*100)
```

```
--------------------------------------------------------------------------------
--------------------
Ponto de máximo: u=0.1758793969849246, v=-0.48743718592964824
--------------------------------------------------------------------------------
--------------------
```

[167]:
```
# Plot de curvas de nível
fig = plt.figure(figsize=(20,7))

U, V = np.meshgrid(u_vec, v_vec)

plt.contour(
    U,
    V,
    z.T,
    levels=50,
    cmap='copper'
)

plt.scatter(
    u_vec[max_post_point[0]],
    v_vec[max_post_point[-1]],
    marker='X',
    color='red'
)

plt.xlabel(
    'u',
    fontsize=15
)
plt.ylabel(
    'v',
    fontsize=15
```

```python
)

plt.title(
    'Log-posteriori normalizada (curvas de nível)',
    fontsize=15
)
```

[167]: Text(0.5, 1.0, 'Log-posteriori normalizada (curvas de nível)')



[168]:
```python
# Plot interativo
fig = go.Figure(
    go.Surface(
        x=u_vec,
        y=v_vec,
        z=z
    )
)

fig.update_layout(
    title='Log-posteriori normalizada (superfície)',
    autosize=False,
    width=500, height=500,
    scene=dict(
        xaxis_title='U',
        yaxis_title='V',
        zaxis_title='Log-posteriori',
    )
)

fig.show()
```

```
[169]: def source_pdf(x):
    """
        This method takes SCALAR value x and return SCALAR source pdf evaluated␣
    ↪at x.
    """
    return np.exp(-x)/np.square(1+np.exp(-x))


def source_pdf_derivative(x):
    """
        This method takes SCALAR value x and return SCALAR source pdf␣
    ↪derivative evaluated at x.
    """
    return (2*np.exp(-2*x)*(1+np.exp(-x)) - np.exp(-x)*np.square(1+np.exp(-x)))/
    ↪np.power([1+np.exp(-x)],[4])


def prior_pdf(X):
    """
        This method takes VECTOR value X and return SCALAR prior pdf evaluated␣
    ↪at X.
    """
    return 1

def prior_pdf_derivative(X):
    """
        This method takes VECTOR value x and return VECTOR prior pdf derivative␣
    ↪evaluated at X.
    """
    return np.zeros(
        shape=X.shape
    )
```

7.4 3.1. Standard Gradient

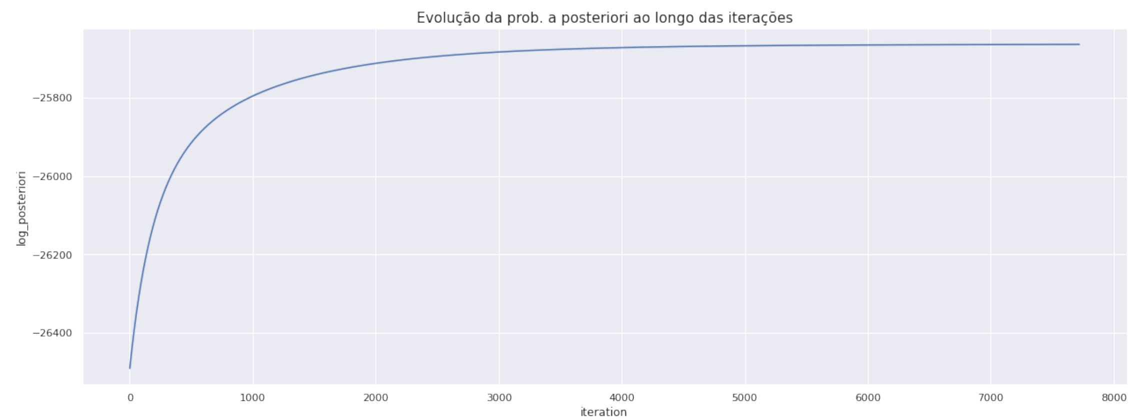
```
[170]: # Initialize estimator
estimator = MAPGradientEstimator(
    learning_rate=learning_rate,
    thresh=thresh,
    max_it=max_it,
    source_pdf=source_pdf,
    source_pdf_derivative=source_pdf_derivative,
    prior_pdf=prior_pdf,
    prior_pdf_derivative=prior_pdf_derivative,
    is_natural_gradient=False
)
```

[171]:
```
%%time

# Run optimization
estimator.fit(
    s=s,
    x=x,
    initial_condition=initial_B,
    n_initializations=1,
    n_jobs=os.cpu_count()-1
)

# Parse results
B=estimator.fit_results[0]['unmixing_matrix']
logs=estimator.fit_results[0]['logs']

print('-'*100)
print('Total iterations: {}'.format(logs.shape[0]))
print('Final log-posteriori: {}'.format(logs.iloc[-1]['log_posteriori']))
print('-'*100)
```

```
/home/leonardo/git/bayesian_bss/src/estimator.py:144: FutureWarning:

elementwise comparison failed; returning scalar instead, but in the future will
perform elementwise comparison


--------------------------------------------------------------------------------
--------------------
Total iterations: 7719
Final log-posteriori: -25663.923247346498
--------------------------------------------------------------------------------
--------------------
CPU times: user 39min 14s, sys: 6min 58s, total: 46min 12s
Wall time: 32min 47s
```

[172]:
```
fig = plt.figure(figsize=(20,7))
sns.lineplot(
    data=logs,
    x='iteration',
    y='log_posteriori'
)
plt.title(
    'Evolução da prob. a posteriori ao longo das iterações',
    fontsize=15
)
```

[172]:
```
Text(0.5, 1.0, 'Evolução da prob. a posteriori ao longo das iterações')
```



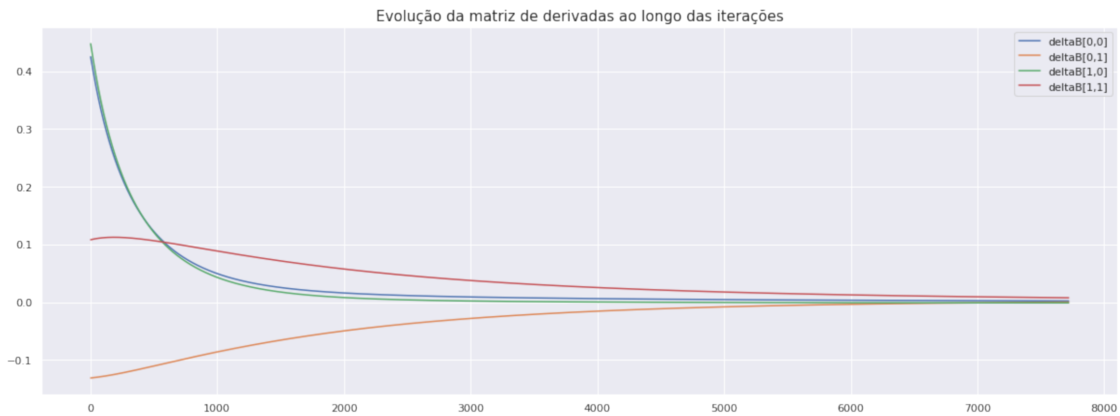
[173]:
```
fig = plt.figure(figsize=(20,7))
sns.lineplot(
    data=logs,
    x='iteration',
    y='detB'
)
plt.title(
    'Evolução do determinante de B ao longo das iterações',
    fontsize=15
)
```

[173]: Text(0.5, 1.0, 'Evolução do determinante de B ao longo das iterações')

[174]:
```
fig = plt.figure(figsize=(20,7))
for i, j in np.ndindex(B.shape):
    plt.plot(
        logs.iteration.values,
```

```
        [row['gradient'][i,j] for n, row in logs.iterrows()],
        label='deltaB[{},{}]'.format(i,j)
    )

plt.title(
    'Evolução da matriz de derivadas ao longo das iterações',
    fontsize=15
)
plt.legend()
```

[174]: <matplotlib.legend.Legend at 0x7f60ad60dca0>



[175]:
```
B
```

[175]:
```
array([[ 0.6551155 , -1.39683907],
       [ 0.6990903 ,  1.31486254]])
```

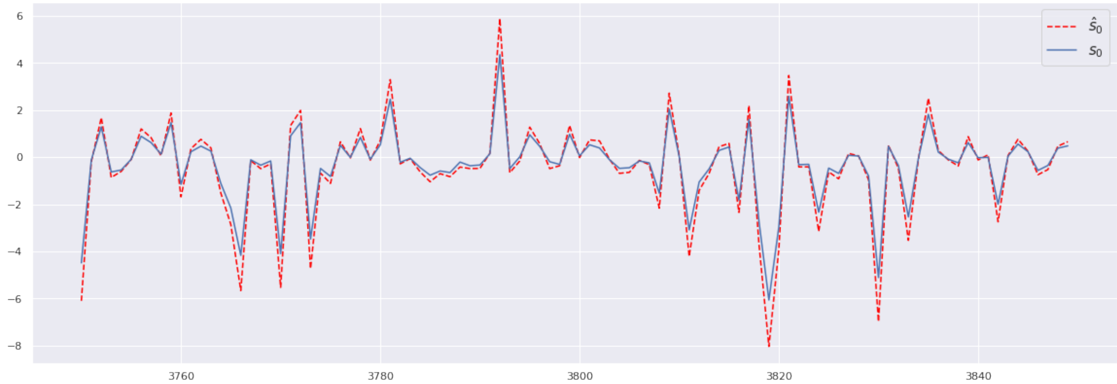
[176]:
```
np.linalg.inv(A)
```

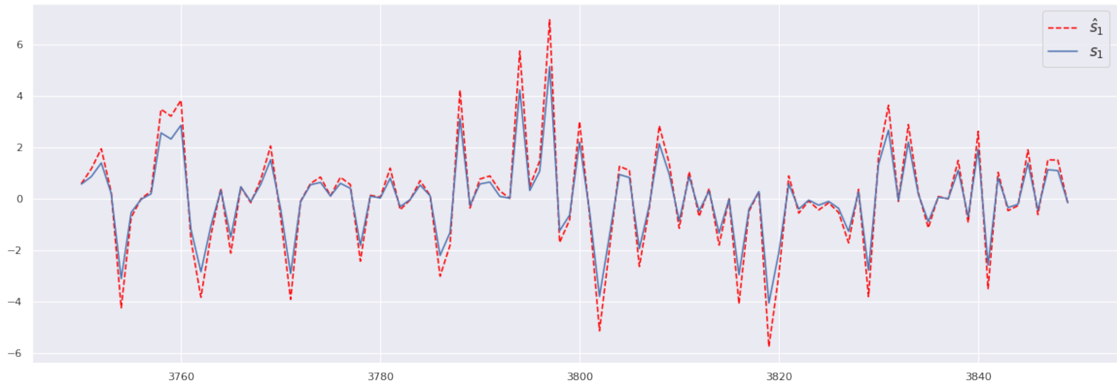
[176]:
```
array([[ 0.5, -1. ],
       [ 0.5,  1. ]])
```

[177]:
```
fig, (ax1, ax2) = plt.subplots(
    nrows=2, ncols=1,
    figsize=(20,15)
)
NPOINTS=100
PLOT_START=NOBS//2
PLOT_END=PLOT_START+NPOINTS

t=range(NOBS)
s_est = B@x
```

```python
#Ax1
ax1.plot(
    t[PLOT_START:PLOT_END],
    s_est[0,PLOT_START:PLOT_END],
    label='$\hat{s}_{0}$',
    color='red',
    linestyle='--'
)
ax1.plot(
    t[PLOT_START:PLOT_END],
    s[0,PLOT_START:PLOT_END],
    label='$s_{0}$'
)
ax1.legend(fontsize=15)

#Ax2
ax2.plot(
    t[PLOT_START:PLOT_END],
    s_est[1,PLOT_START:PLOT_END],
    label='$\hat{s}_{1}$',
    color='red',
    linestyle='--'
)
ax2.plot(
    t[PLOT_START:PLOT_END],
    s[1,PLOT_START:PLOT_END],
    label='$s_{1}$'
)
ax2.legend(fontsize=15)
```

```
[177]: <matplotlib.legend.Legend at 0x7f60b2019f10>
```





[178]:
```
# Error norm
print('Norma do erro de estimação: {}'.format(
np.linalg.norm(
    np.subtract(s,s_est)
)/np.size(s)))
```

```
Norma do erro de estimação: 0.0040996567821268756
```

7.4.1 2.3.2. MAP - determinant equal to one

[179]:
```
def source_pdf(x):
    return np.exp(-x)/np.square(1+np.exp(-x))

def prior_pdf(B):
    sig=0.1
    desired_det=1
    return (1/np.sqrt(2*np.pi*np.square(sig)))*np.exp(-np.square(np.linalg.
 ↪det(B)-desired_det)/(2*np.square(sig)))
```

[180]:
```
%%time
```

```python
u_lims = (-1, 1)
v_lims = (-1, 1)
n_points = 200

u_vec = np.linspace(u_lims[0], u_lims[-1], n_points)
v_vec = np.linspace(v_lims[0], v_lims[-1], n_points)

z = get_posteriori_neighborhood(
    A=A,
    x=x,
    T=NOBS,
    u_vec=u_vec,
    v_vec=v_vec,
    n_points=n_points,
    source_pdf_fn=source_pdf,
    prior_pdf_fn=prior_pdf,
    njobs=os.cpu_count()-1
)
```

```
---------------------------------------------------------------------------
RemoteTraceback                           Traceback (most recent call last)
RemoteTraceback:
"""
Traceback (most recent call last):
  File "/home/leonardo/py3env/lib/python3.8/site-packages/multiprocess/pool.py"
  ↪line 125, in worker
    result = (True, func(*args, **kwds))
  File "/home/leonardo/py3env/lib/python3.8/site-packages/multiprocess/pool.py"
  ↪line 48, in mapstar
    return list(map(*args))
  File "/home/leonardo/py3env/lib/python3.8/site-packages/pathos/helpers/
  ↪mp_helper.py", line 15, in <lambda>
    func = lambda args: f(*args)
  File "/tmp/ipykernel_45077/2236618269.py", line 26, in __get_posteriori
    B = np.linalg.inv(A_shifted)
  File "<__array_function__ internals>", line 180, in inv
  File "/home/leonardo/py3env/lib/python3.8/site-packages/numpy/linalg/linalg.
  ↪py", line 552, in inv
    ainv = _umath_linalg.inv(a, signature=signature, extobj=extobj)
  File "/home/leonardo/py3env/lib/python3.8/site-packages/numpy/linalg/linalg.
  ↪py", line 89, in _raise_linalgerror_singular
    raise LinAlgError("Singular matrix")
numpy.linalg.LinAlgError: Singular matrix
"""

The above exception was the direct cause of the following exception:
```

```
LinAlgError                               Traceback (most recent call last)
<timed exec> in <module>

/tmp/ipykernel_45077/2236618269.py in get_posteriori_neighborhood(A, x, T,␣
↪u_vec, v_vec, n_points, source_pdf_fn, prior_pdf_fn, njobs)
     81     )
     82     with pathos.multiprocessing.ProcessingPool(njobs) as p:
---> 83         results = p.map(exec_fn, iterator)
     84
     85     for r in results:

~/py3env/lib/python3.8/site-packages/pathos/multiprocessing.py in map(self, f,␣
↪*args, **kwds)
    137         AbstractWorkerPool._AbstractWorkerPool__map(self, f, *args,␣
↪**kwds)
    138         _pool = self._serve()
--> 139         return _pool.map(star(f), zip(*args)) # chunksize
    140     map.__doc__ = AbstractWorkerPool.map.__doc__
    141     def imap(self, f, *args, **kwds):

~/py3env/lib/python3.8/site-packages/multiprocess/pool.py in map(self, func,␣
↪iterable, chunksize)
    362         in a list that is returned.
    363         '''
--> 364         return self._map_async(func, iterable, mapstar, chunksize).get(
    365
    366     def starmap(self, func, iterable, chunksize=None):

~/py3env/lib/python3.8/site-packages/multiprocess/pool.py in get(self, timeout)
    769             return self._value
    770         else:
--> 771             raise self._value
    772
    773     def _set(self, i, obj):

LinAlgError: Singular matrix
```

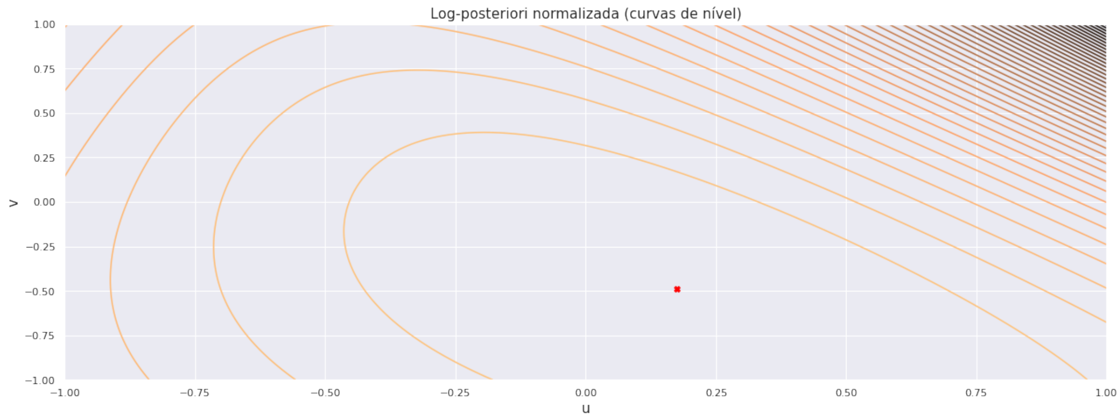
[181]:
```
# Ponto de máximo
max_post_point = np.unravel_index(
    z.argmax(),
    z.shape
)

#
max_post = z[max_post_point]

print('-'*100)
```

```
print('Ponto de máximo: u={}, v={}'.format(u_vec[max_post_point[0]],
    v_vec[max_post_point[1]]))
print('-'*100)
```

```
----------------------------------------------------------------------------------------------------
Ponto de máximo: u=0.1758793969849246, v=-0.48743718592964824
----------------------------------------------------------------------------------------------------
```

[182]:
```
# Plot de curvas de nível
fig = plt.figure(figsize=(20,7))

U, V = np.meshgrid(u_vec, v_vec)

plt.contour(
    U,
    V,
    z.T,
    levels=50,
    cmap='copper'
)

plt.scatter(
    u_vec[max_post_point[0]],
    v_vec[max_post_point[-1]],
    marker='X',
    color='red'
)

plt.xlabel(
    'u',
    fontsize=15
)
plt.ylabel(
    'v',
    fontsize=15
)

plt.title(
    'Log-posteriori normalizada (curvas de nível)',
    fontsize=15
)
```

[182]:
```
Text(0.5, 1.0, 'Log-posteriori normalizada (curvas de nível)')
```



[183]:
```
# Plot interativo
fig = go.Figure(
    go.Surface(
        x=u_vec,
        y=v_vec,
        z=z
    )
)

fig.update_layout(
    title='Log-posteriori normalizada (superfície)',
    autosize=False,
    width=500, height=500,
    scene=dict(
        xaxis_title='U',
        yaxis_title='V',
        zaxis_title='Log-posteriori',
    )
)

fig.show()
```

[184]:
```
def source_pdf(x):
    """
        This method takes SCALAR value x and return SCALAR source pdf evaluated␣
 ↪at x.
    """
    return np.exp(-x)/np.square(1+np.exp(-x))


def source_pdf_derivative(x):
    """
```

```
        This method takes SCALAR value x and return SCALAR source pdf␣
↪derivative evaluated at x.
        """
        return (2*np.exp(-2*x)*(1+np.exp(-x)) - np.exp(-x)*np.square(1+np.exp(-x)))/
↪np.power([1+np.exp(-x)],[4])

def prior_pdf(X):
    sig=0.1
    desired_det=1
    return (1/np.sqrt(2*np.pi*np.square(sig)))*np.exp(-np.square(np.linalg.
↪det(X)-desired_det)/(2*np.square(sig)))

def prior_pdf_derivative(X):
    sig=0.1
    return -1*prior_pdf(X)*(np.linalg.det(X)-1)/np.square(sig)
```

7.5 3.1. Standard Gradient

[185]:
```
# Initialize estimator
estimator = MAPGradientEstimator(
    learning_rate=learning_rate,
    thresh=thresh,
    max_it=max_it,
    source_pdf=source_pdf,
    source_pdf_derivative=source_pdf_derivative,
    prior_pdf=prior_pdf,
    prior_pdf_derivative=prior_pdf_derivative,
    is_natural_gradient=False
)
```

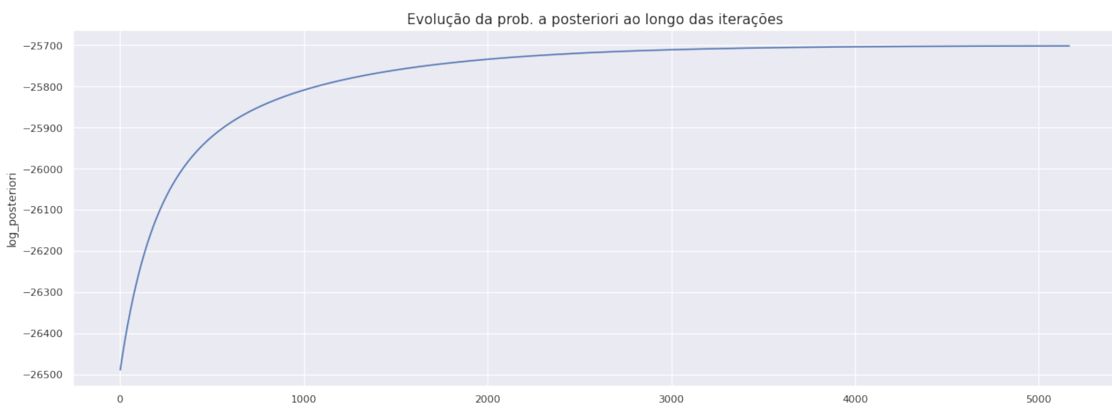
[186]:
```
%%time

# Run optimization
estimator.fit(
    s=s,
    x=x,
    initial_condition=initial_B,
    n_initializations=1,
    n_jobs=os.cpu_count()-1
)

# Parse results
B=estimator.fit_results[0]['unmixing_matrix']
logs=estimator.fit_results[0]['logs']

print('-'*100)
```

```
print('Total iterations: {}'.format(logs.shape[0]))
print('Final log-posteriori: {}'.format(logs.iloc[-1]['log_posteriori']))
print('-'*100)
```

```
/home/leonardo/git/bayesian_bss/src/estimator.py:144: FutureWarning:

elementwise comparison failed; returning scalar instead, but in the future will
perform elementwise comparison


--------------------------------------------------------------------------------
--------------------
Total iterations: 5165
Final log-posteriori: -25701.550320287548
--------------------------------------------------------------------------------
--------------------
CPU times: user 26min 12s, sys: 4min 38s, total: 30min 51s
Wall time: 21min 53s
```

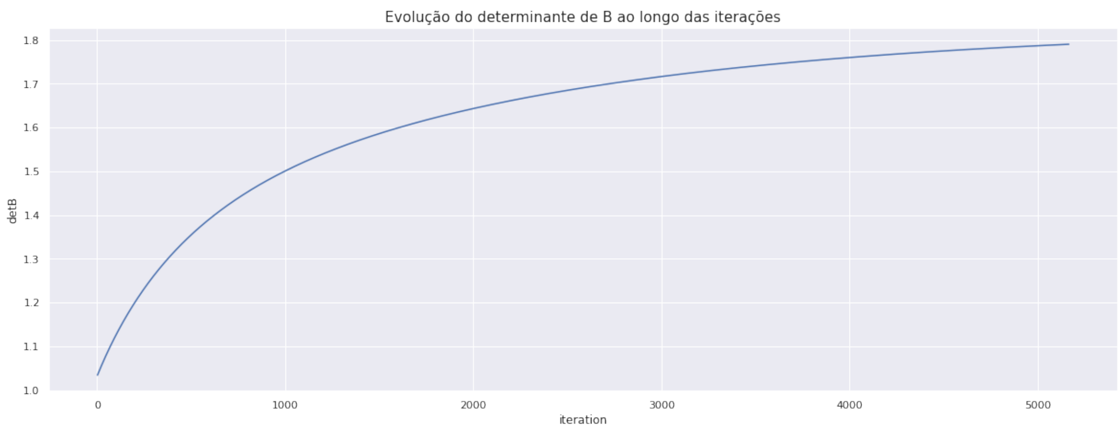
[187]:
```
fig = plt.figure(figsize=(20,7))
sns.lineplot(
    data=logs,
    x='iteration',
    y='log_posteriori'
)
plt.title(
    'Evolução da prob. a posteriori ao longo das iterações',
    fontsize=15
)
```

[187]:
```
Text(0.5, 1.0, 'Evolução da prob. a posteriori ao longo das iterações')
```



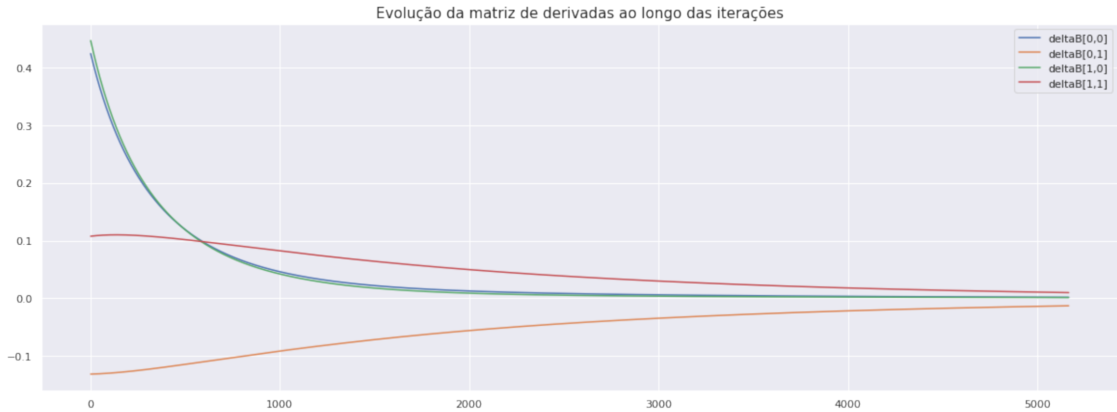
[188]:
```
fig = plt.figure(figsize=(20,7))
sns.lineplot(
    data=logs,
    x='iteration',
    y='detB'
)
plt.title(
    'Evolução do determinante de B ao longo das iterações',
    fontsize=15
)
```

[188]: `Text(0.5, 1.0, 'Evolução do determinante de B ao longo das iterações')`



[189]:
```
fig = plt.figure(figsize=(20,7))
for i, j in np.ndindex(B.shape):
    plt.plot(
        logs.iteration.values,
        [row['gradient'][i,j] for n, row in logs.iterrows()],
        label='deltaB[{},{}]'.format(i,j)
    )

plt.title(
    'Evolução da matriz de derivadas ao longo das iterações',
    fontsize=15
)
plt.legend()
```

[189]: `<matplotlib.legend.Legend at 0x7f60ad45da90>`



[190]:
```
B
```

[190]:
```
array([[ 0.63135681, -1.42064294],
       [ 0.70453392,  1.2500693 ]])
```

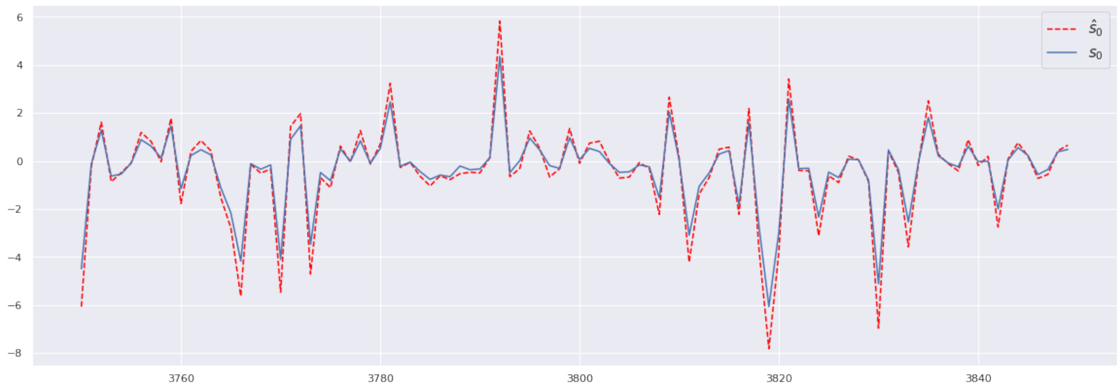
[191]:
```
np.linalg.inv(A)
```

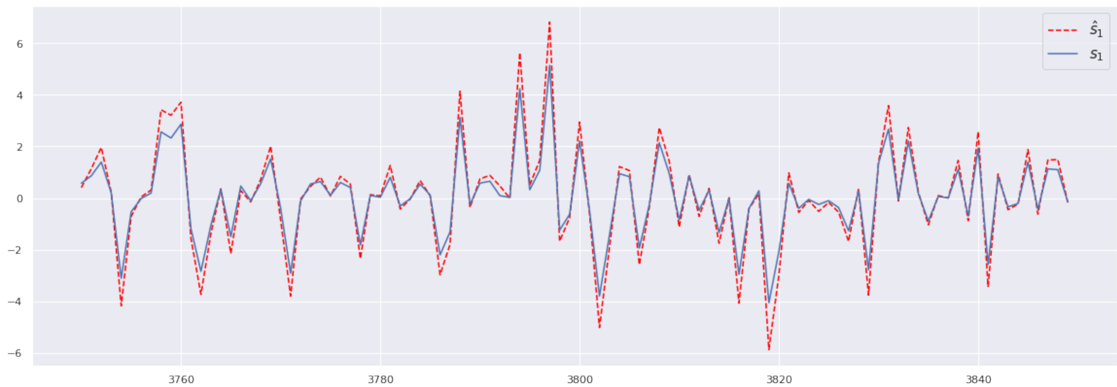
[191]:
```
array([[ 0.5, -1. ],
       [ 0.5,  1. ]])
```

[192]:
```
fig, (ax1, ax2) = plt.subplots(
    nrows=2, ncols=1,
    figsize=(20,15)
)
NPOINTS=100
PLOT_START=NOBS//2
PLOT_END=PLOT_START+NPOINTS

t=range(NOBS)
s_est = B@x

#Ax1
ax1.plot(
    t[PLOT_START:PLOT_END],
    s_est[0,PLOT_START:PLOT_END],
    label='$\hat{s}_{0}$',
    color='red',
    linestyle='--'
)
ax1.plot(
    t[PLOT_START:PLOT_END],
    s[0,PLOT_START:PLOT_END],
```

```
    label='$s_{0}$'
)
ax1.legend(fontsize=15)

#Ax2
ax2.plot(
    t[PLOT_START:PLOT_END],
    s_est[1,PLOT_START:PLOT_END],
    label='$\hat{s}_{1}$',
    color='red',
    linestyle='--'
)
ax2.plot(
    t[PLOT_START:PLOT_END],
    s[1,PLOT_START:PLOT_END],
    label='$s_{1}$'
)
ax2.legend(fontsize=15)
```

[192]: <matplotlib.legend.Legend at 0x7f60ad470070>





[193]:
```
# Error norm
print('Norma do erro de estimação: {}'.format(
np.linalg.norm(
    np.subtract(s,s_est)
)/np.size(s)))
```

```
Norma do erro de estimação: 0.0039555145624831116
```

7.6 2.3.3. MAP - near-identity transformation

[194]:
```
def source_pdf(x):
    return np.exp(-x)/np.square(1+np.exp(-x))

def prior_pdf(B):
    sig=0.1
    return np.exp(
        (-1/2/np.square(sig))*np.square(np.linalg.norm(B-np.eye(B.shape[0])))
    )
```

[195]:
```
%%time

u_lims = (-1, 1)
v_lims = (-1, 1)
n_points = 200

u_vec = np.linspace(u_lims[0], u_lims[-1], n_points)
v_vec = np.linspace(v_lims[0], v_lims[-1], n_points)

z = get_posteriori_neighborhood(
    A=A,
    x=x,
    T=NOBS,
    u_vec=u_vec,
    v_vec=v_vec,
    n_points=n_points,
    source_pdf_fn=source_pdf,
    prior_pdf_fn=prior_pdf,
    njobs=os.cpu_count()-1
)
```

```
---------------------------------------------------------------------------
RemoteTraceback                           Traceback (most recent call last)
RemoteTraceback:
"""
Traceback (most recent call last):
  File "/home/leonardo/py3env/lib/python3.8/site-packages/multiprocess/pool.py"
 ↪line 125, in worker
```

```
    result = (True, func(*args, **kwds))
  File "/home/leonardo/py3env/lib/python3.8/site-packages/multiprocess/pool.py" ␣
↪line 48, in mapstar
    return list(map(*args))
  File "/home/leonardo/py3env/lib/python3.8/site-packages/pathos/helpers/
↪mp_helper.py", line 15, in <lambda>
    func = lambda args: f(*args)
  File "/tmp/ipykernel_45077/2236618269.py", line 26, in __get_posteriori
    B = np.linalg.inv(A_shifted)
  File "<__array_function__ internals>", line 180, in inv
  File "/home/leonardo/py3env/lib/python3.8/site-packages/numpy/linalg/linalg.
↪py", line 552, in inv
    ainv = _umath_linalg.inv(a, signature=signature, extobj=extobj)
  File "/home/leonardo/py3env/lib/python3.8/site-packages/numpy/linalg/linalg.
↪py", line 89, in _raise_linalgerror_singular
    raise LinAlgError("Singular matrix")
numpy.linalg.LinAlgError: Singular matrix
"""

The above exception was the direct cause of the following exception:

LinAlgError                               Traceback (most recent call last)
<timed exec> in <module>

/tmp/ipykernel_45077/2236618269.py in get_posteriori_neighborhood(A, x, T,␣
↪u_vec, v_vec, n_points, source_pdf_fn, prior_pdf_fn, njobs)
     81     )
     82     with pathos.multiprocessing.ProcessingPool(njobs) as p:
---> 83         results = p.map(exec_fn, iterator)
     84
     85     for r in results:

~/py3env/lib/python3.8/site-packages/pathos/multiprocessing.py in map(self, f,␣
↪*args, **kwds)
    137         AbstractWorkerPool._AbstractWorkerPool__map(self, f, *args,␣
↪**kwds)
    138         _pool = self._serve()
--> 139         return _pool.map(star(f), zip(*args)) # chunksize
    140     map.__doc__ = AbstractWorkerPool.map.__doc__
    141     def imap(self, f, *args, **kwds):

~/py3env/lib/python3.8/site-packages/multiprocess/pool.py in map(self, func,␣
↪iterable, chunksize)
    362         in a list that is returned.
    363         '''
--> 364         return self._map_async(func, iterable, mapstar, chunksize).get(
    365
    366     def starmap(self, func, iterable, chunksize=None):
```

```
~/py3env/lib/python3.8/site-packages/multiprocess/pool.py in get(self, timeout)
    769                 return self._value
    770             else:
--> 771                 raise self._value
    772 
    773     def _set(self, i, obj):

LinAlgError: Singular matrix
```

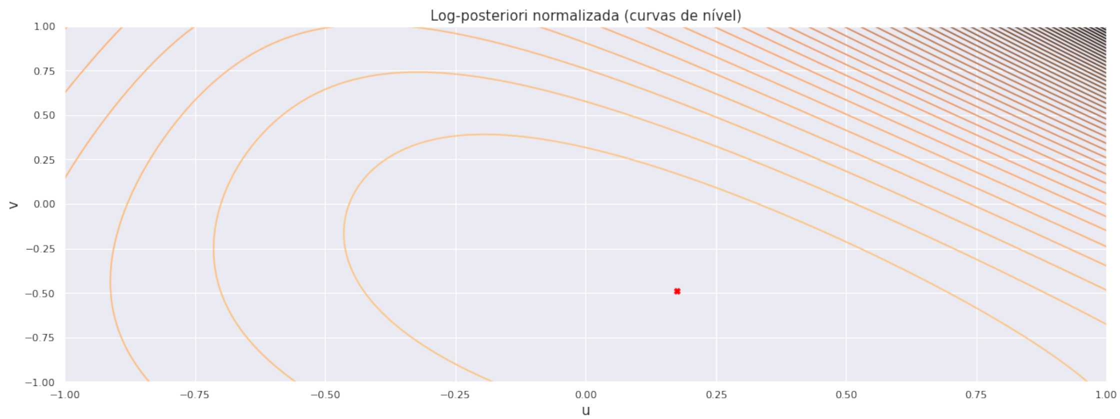
[196]:
```
# Ponto de máximo
max_post_point = np.unravel_index(
    z.argmax(),
    z.shape
)

#
max_post = z[max_post_point]

print('-'*100)
print('Ponto de máximo: u={}, v={}'.format(u_vec[max_post_point[0]],␣
 ↪v_vec[max_post_point[1]]))
print('-'*100)
```

```
--------------------------------------------------------------------------------
--------------------
Ponto de máximo: u=0.17587939698492464, v=-0.48743718592964824
--------------------------------------------------------------------------------
--------------------
```

[197]:
```
# Plot de curvas de nível
fig = plt.figure(figsize=(20,7))

U, V = np.meshgrid(u_vec, v_vec)

plt.contour(
    U,
    V,
    z.T,
    levels=50,
    cmap='copper'
)

plt.scatter(
    u_vec[max_post_point[0]],
    v_vec[max_post_point[-1]],
    marker='X',
```

```
    color='red'
)

plt.xlabel(
    'u',
    fontsize=15
)
plt.ylabel(
    'v',
    fontsize=15
)

plt.title(
    'Log-posteriori normalizada (curvas de nível)',
    fontsize=15
)
```

[197]: Text(0.5, 1.0, 'Log-posteriori normalizada (curvas de nível)')



[198]:
```
# Plot interativo
fig = go.Figure(
    go.Surface(
        x=u_vec,
        y=v_vec,
        z=z
    )
)

fig.update_layout(
    title='Log-posteriori normalizada (superfície)',
    autosize=False,
    width=500, height=500,
```

```
    scene=dict(
        xaxis_title='U',
        yaxis_title='V',
        zaxis_title='Log-posteriori',
    )
)

fig.show()
```

[199]:

```
def source_pdf(x):
    """
        This method takes SCALAR value x and return SCALAR source pdf evaluated␣
 ↪at x.
    """
    return np.exp(-x)/np.square(1+np.exp(-x))


def source_pdf_derivative(x):
    """
        This method takes SCALAR value x and return SCALAR source pdf␣
 ↪derivative evaluated at x.
    """
    return (2*np.exp(-2*x)*(1+np.exp(-x)) - np.exp(-x)*np.square(1+np.exp(-x)))/
 ↪np.power([1+np.exp(-x)],[4])

def prior_pdf(X):
    sig=0.1
    return np.exp(
        (-1/2/np.square(sig))*np.linalg.norm(X-np.eye(X.shape[0]))
    )

def prior_pdf_derivative(X):
    sig=0.1
    der_X=np.empty(X.shape)
    I=np.eye(X.shape[0])
    for i, j in np.ndindex(X.shape):
        der_X[i, j] = prior_pdf(X)*(-1/np.square(sig))*(X[i,j]-I[i,j])
    return der_X
```

7.7 3.1. Standard Gradient

[200]:

```
# Initialize estimator
estimator = MAPGradientEstimator(
    learning_rate=learning_rate,
    thresh=thresh,
    max_it=max_it,
    source_pdf=source_pdf,
```

```
    source_pdf_derivative=source_pdf_derivative,
    prior_pdf=prior_pdf,
    prior_pdf_derivative=prior_pdf_derivative,
    is_natural_gradient=False
)
```

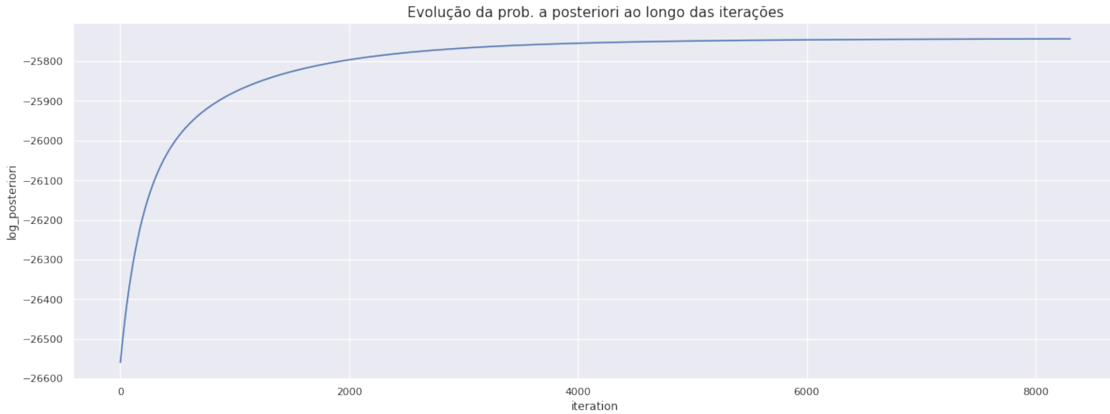
[201]:
```
%%time

# Run optimization
estimator.fit(
    s=s,
    x=x,
    initial_condition=initial_B,
    n_initializations=1,
    n_jobs=os.cpu_count()-1
)

# Parse results
B=estimator.fit_results[0]['unmixing_matrix']
logs=estimator.fit_results[0]['logs']

print('-'*100)
print('Total iterations: {}'.format(logs.shape[0]))
print('Final log-posteriori: {}'.format(logs.iloc[-1]['log_posteriori']))
print('-'*100)
```

```
/home/leonardo/git/bayesian_bss/src/estimator.py:144: FutureWarning:

elementwise comparison failed; returning scalar instead, but in the future will
perform elementwise comparison


--------------------------------------------------------------------------------
--------------------
Total iterations: 8302
Final log-posteriori: -25743.59359764015
--------------------------------------------------------------------------------
--------------------
CPU times: user 43min 1s, sys: 7min 27s, total: 50min 29s
Wall time: 36min 3s
```

[202]:
```
fig = plt.figure(figsize=(20,7))
sns.lineplot(
    data=logs,
    x='iteration',
    y='log_posteriori'
)
plt.title(
```

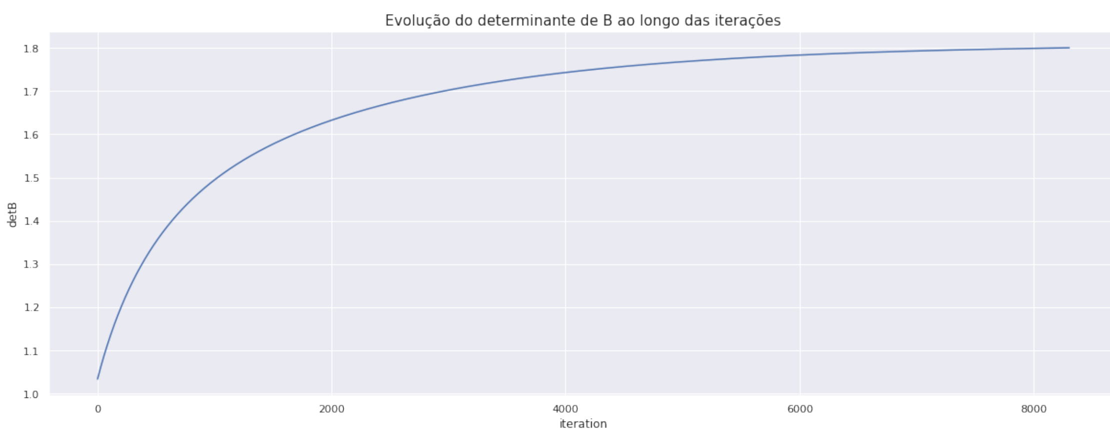
```
    'Evolução da prob. a posteriori ao longo das iterações',
    fontsize=15
)
```

[202]: Text(0.5, 1.0, 'Evolução da prob. a posteriori ao longo das iterações')



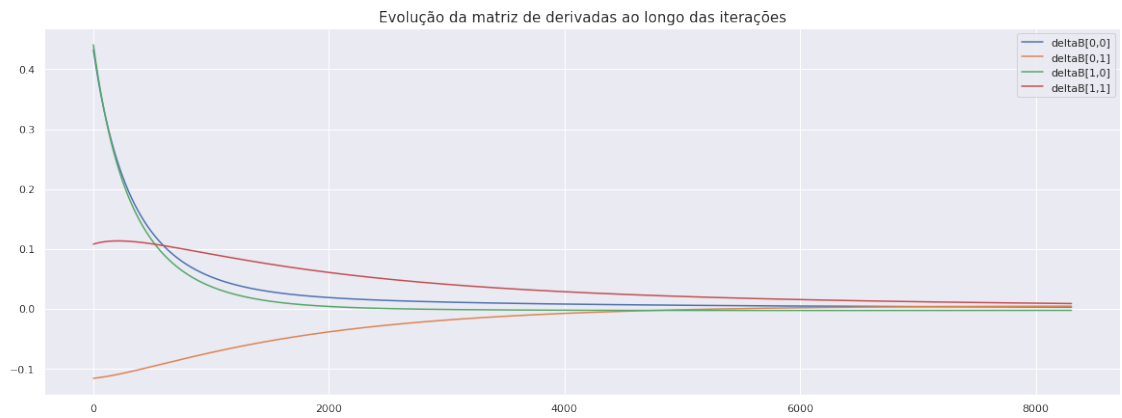
[203]:
```
fig = plt.figure(figsize=(20,7))
sns.lineplot(
    data=logs,
    x='iteration',
    y='detB'
)
plt.title(
    'Evolução do determinante de B ao longo das iterações',
    fontsize=15
)
```

[203]: Text(0.5, 1.0, 'Evolução do determinante de B ao longo das iterações')



[204]:
```
fig = plt.figure(figsize=(20,7))
for i, j in np.ndindex(B.shape):
    plt.plot(
        logs.iteration.values,
        [row['gradient'][i,j] for n, row in logs.iterrows()],
        label='deltaB[{},{}]'.format(i,j)
    )

plt.title(
    'Evolução da matriz de derivadas ao longo das iterações',
    fontsize=15
)
plt.legend()
```

[204]: <matplotlib.legend.Legend at 0x7f60aaa2cfd0>



[205]:
```
B
```

[205]:
```
array([[ 0.67536414, -1.32727789],
       [ 0.67302422,  1.34286225]])
```

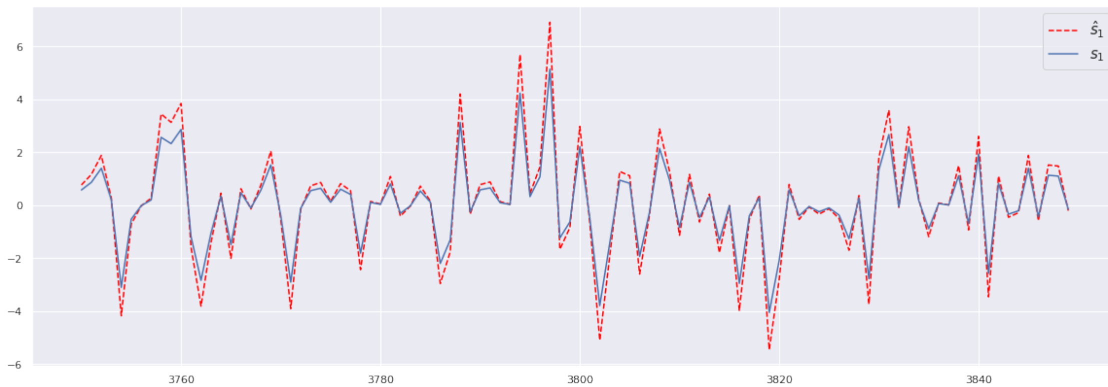
[206]:
```
np.linalg.inv(A)
```

[206]:
```
array([[ 0.5, -1. ],
       [ 0.5,  1. ]])
```

[207]:
```
fig, (ax1, ax2) = plt.subplots(
    nrows=2, ncols=1,
    figsize=(20,15)
)
NPOINTS=100
```

```python
PLOT_START=NOBS//2
PLOT_END=PLOT_START+NPOINTS

t=range(NOBS)
s_est = B@x

#Ax1
ax1.plot(
    t[PLOT_START:PLOT_END],
    s_est[0,PLOT_START:PLOT_END],
    label='$\hat{s}_{0}$',
    color='red',
    linestyle='--'
)
ax1.plot(
    t[PLOT_START:PLOT_END],
    s[0,PLOT_START:PLOT_END],
    label='$s_{0}$'
)
ax1.legend(fontsize=15)

#Ax2
ax2.plot(
    t[PLOT_START:PLOT_END],
    s_est[1,PLOT_START:PLOT_END],
    label='$\hat{s}_{1}$',
    color='red',
    linestyle='--'
)
ax2.plot(
    t[PLOT_START:PLOT_END],
    s[1,PLOT_START:PLOT_END],
    label='$s_{1}$'
)
ax2.legend(fontsize=15)
```

[207]: <matplotlib.legend.Legend at 0x7f60aad86b80>



[208]:
```
# Error norm
print('Norma do erro de estimação: {}'.format(
np.linalg.norm(
    np.subtract(s,s_est)
)/np.size(s)))
```

```
Norma do erro de estimação: 0.0039197505631380775
```

8 4. LARGELY MISSPECIFIED MODEL - SUBGAUSSIAN

8.1 3.1. Initialize Sources

[209]:
```
def source_cumulative(x):
    return 1/(1+np.exp(-x))

def source_pdf(x):
    return np.exp(-x)/np.square(1+np.exp(-x))

def source_generator(
    nsources,
    nobs,
```

```
    mu,
    scale
):
    return np.random.triangular(
        left=-2,
        mode=0,
        right=2,
        size=(nsources, nobs)
    )
```

[210]:

```
NSOURCES=2
NOBS=7500
MU=0
SCALE=1

s = source_generator(
    nsources=NSOURCES,
    nobs=NOBS,
    mu=MU,
    scale=SCALE
)

print('-'*100)
print('NÚMERO DE FONTES: {}'.format(NSOURCES))
print('TAMANHO DOS SINAIS DAS FONTES: {}'.format(NOBS))
for i in range(s.shape[0]):
    print(
        'CURTOSE s{}: {}'.format(
            i,
            st.kurtosis(a=s[i,:])
        )
    )

print('-'*100)
```

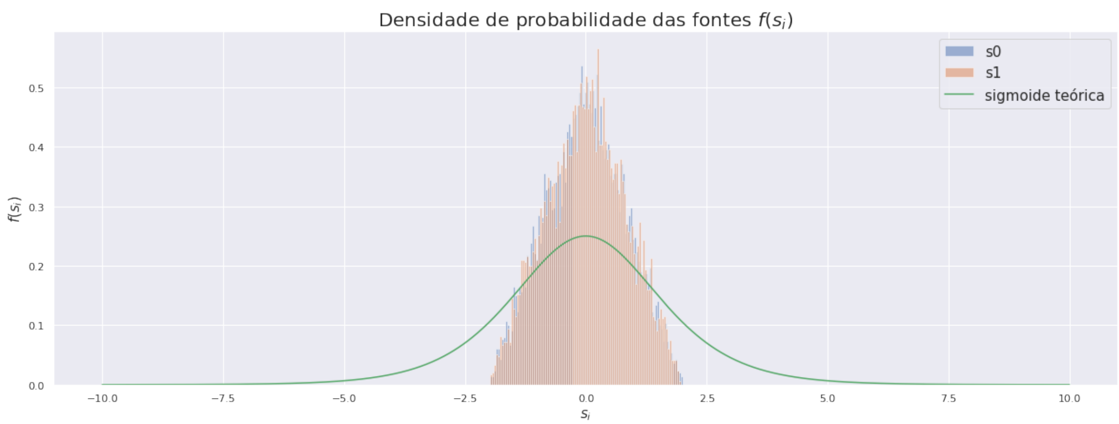
```
--------------------------------------------------------------------------------
--------------------
NÚMERO DE FONTES: 2
TAMANHO DOS SINAIS DAS FONTES: 7500
CURTOSE s0: -0.5708517855502775
CURTOSE s1: -0.5944819636332639
--------------------------------------------------------------------------------
--------------------
```

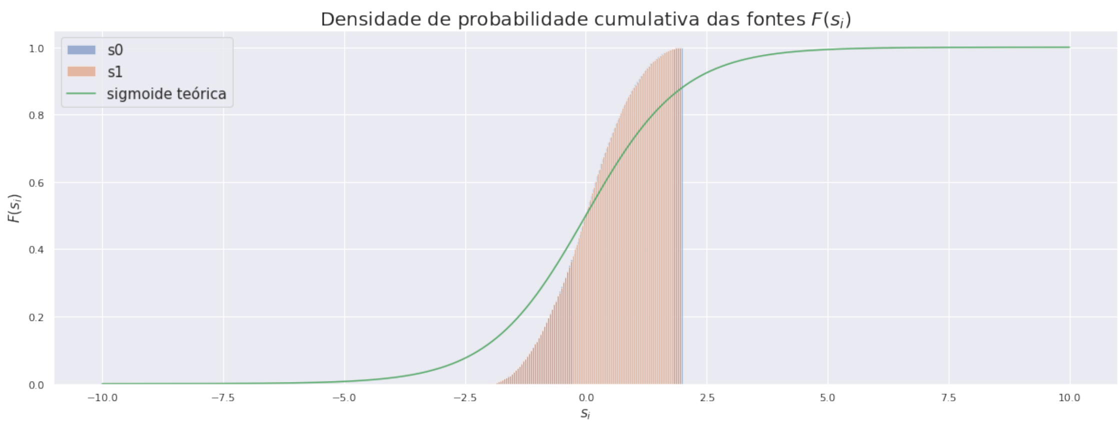
[211]:

```
fig, (ax1, ax2) = plt.subplots(
    nrows=2, ncols=1,
    figsize=(20,15)
```

```
)


for i in range(1, NSOURCES+1):
    ax1.hist(
        x=s[i-1,:],
        bins=100,
        density=True,
        label='s{}'.format(i-1),
        alpha=0.5
    )
    ax2.hist(
        x=s[i-1,:],
        bins=100,
        cumulative=True,
        density=True,
        label='s{}'.format(i-1),
        alpha=0.5
    )

ax1.plot(
    np.linspace(-10,10,1000),
    [source_pdf(x) for x in np.linspace(-10,10,1000)],
    label='sigmoide teórica'
)


ax2.plot(
    np.linspace(-10,10,1000),
    [source_cumulative(x) for x in np.linspace(-10,10,1000)],
    label='sigmoide teórica'
)

ax1.set_xlabel(
    '$s_{i}$',
    fontsize=15
)
ax1.set_ylabel(
    '$f(s_{i})$',
    fontsize=15
)
ax1.set_title(
    'Densidade de probabilidade das fontes $f(s_{i})$',
    fontsize=20
)

ax2.set_xlabel(
```

```
    '$s_{i}$',
    fontsize=15
)
ax2.set_ylabel(
    '$F(s_{i})$',
    fontsize=15
)
ax2.set_title(
    'Densidade de probabilidade cumulativa das fontes $F(s_{i})$',
    fontsize=20
)

l1=ax1.legend(fontsize=15)
l2=ax2.legend(fontsize=15)
```





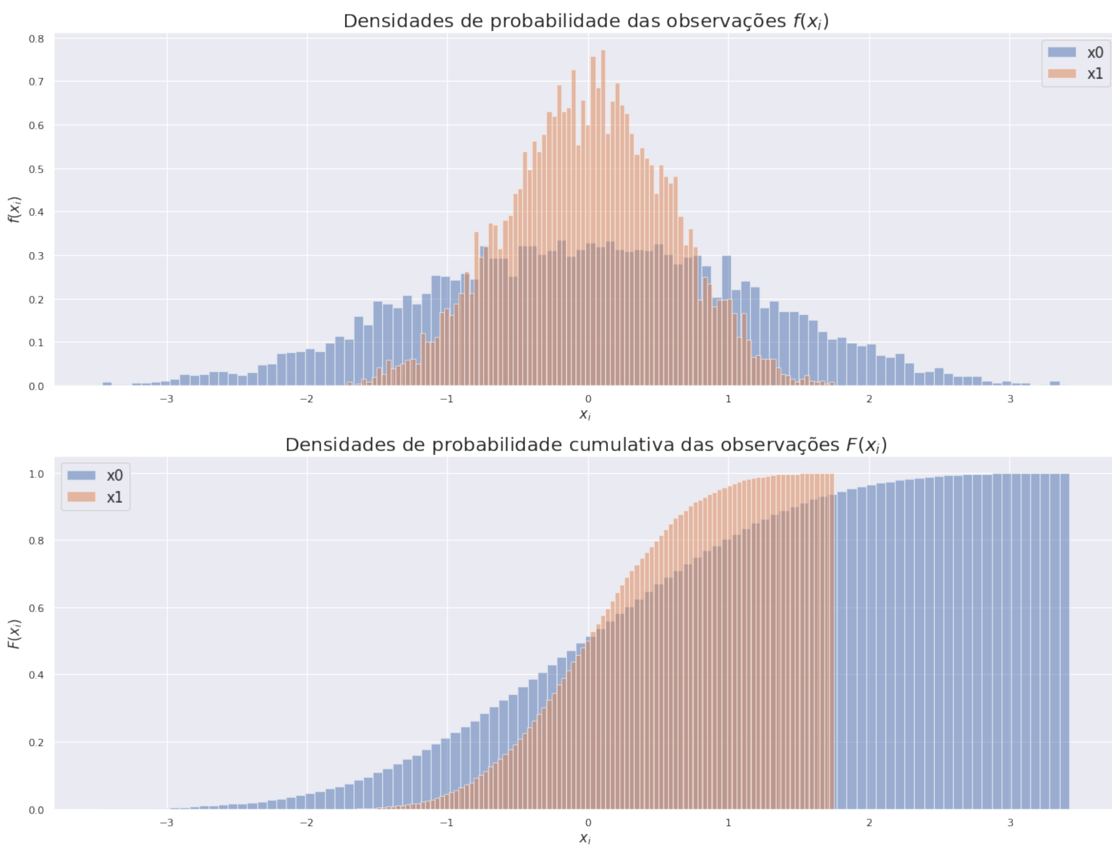
8.2 2.2. Mix sources and generate observations

[212]:
```
# Mixing matrix
A = np.array([
    [1, 1],
    [-0.5, 0.5]
])
print('MIXING MATRIX A:')
print(A)
```

```
MIXING MATRIX A:
[[ 1.   1. ]
 [-0.5  0.5]]
```

[213]:
```
# Observed sources
x = A@s

fig, (ax1, ax2) = plt.subplots(
    nrows=2, ncols=1,
    figsize=(20,15)
)

for i in range(1, NSOURCES+1):
    ax1.hist(
        x=x[i-1,:],
        bins=100,
        density=True,
        label='x{}'.format(i-1),
        alpha=0.5
    )
    ax2.hist(
        x=x[i-1,:],
        bins=100,
        cumulative=True,
        density=True,
        label='x{}'.format(i-1),
        alpha=0.5
    )

ax1.set_xlabel(
    '$x_{i}$',
    fontsize=15
)
ax1.set_ylabel(
    '$f(x_{i})$',
    fontsize=15
)
```

```
ax1.set_title(
    'Densidades de probabilidade das observações $f(x_{i})$',
    fontsize=20
)

ax2.set_xlabel(
    '$x_{i}$',
    fontsize=15
)
ax2.set_ylabel(
    '$F(x_{i})$',
    fontsize=15
)
ax2.set_title(
    'Densidades de probabilidade cumulativa das observações $F(x_{i})$',
    fontsize=20
)

l1=ax1.legend(fontsize=15)
l2=ax2.legend(fontsize=15)
```



8.3 2.3. Performance Curves

[214]:
```python
def get_posteriori_neighborhood(
    A,
    x,
    T,
    u_vec,
    v_vec,
    n_points,
    source_pdf_fn,
    prior_pdf_fn,
    njobs=1
):

    def __get_posteriori(
        A,
        x,
        T,
        source_pdf_fn,
        prior_pdf_fn,
        idx_info
    ):
        i, u = idx_info[0]
        j, v = idx_info[-1]

        A_shifted = A + u*symmetric_basis + v*skew_symmetric_basis

        B = np.linalg.inv(A_shifted)

        likelihood = np.log(np.abs(np.linalg.det(B)))

        likelihood_iterator = [
            (t,i) for t,i in itertools.product(
                range(x.shape[-1]),
                range(x.shape[0]))
        ]

        s_est = B@x

        likelihood += (1/T)*np.sum([
            np.log(source_pdf_fn(s_est[i,t])) for t,i in likelihood_iterator
        ])

        prior = np.log(prior_pdf_fn(B))/T

        return {
            'i': i,
            'j': j,
```

```
            'log_posteriori': likelihood + prior
        }


    symmetric_basis = np.array([
        [0, 1],
        [1, 0]
    ])

    skew_symmetric_basis = np.array([
        [0, -1],
        [1, 0]
    ])

    iterator = itertools.product(
        enumerate(u_vec),
        enumerate(v_vec)
    )
    # it = [x for x in iterator]

    exec_fn = partial(
        __get_posteriori,
        A,
        x,
        T,
        source_pdf_fn,
        prior_pdf_fn
    )

    z = np.empty(
        shape=(
            u_vec.shape[0],
            v_vec.shape[0]
        )
    )
    with pathos.multiprocessing.ProcessingPool(njobs) as p:
        results = p.map(exec_fn, iterator)

    for r in results:
        i=r['i']
        j=r['j']
        z[i, j]=r['log_posteriori']

    return z
```

8.3.1 2.3.1. Likelihood

[215]:
```
def source_pdf(x):
    return np.exp(-x)/np.square(1+np.exp(-x))

def prior_pdf(B):
    return 1
```

[216]:
```
%%time

u_lims = (-1, 1)
v_lims = (-1, 1)
n_points = 200

u_vec = np.linspace(u_lims[0], u_lims[-1], n_points)
v_vec = np.linspace(v_lims[0], v_lims[-1], n_points)

z = get_posteriori_neighborhood(
    A=A,
    x=x,
    T=NOBS,
    u_vec=u_vec,
    v_vec=v_vec,
    n_points=n_points,
    source_pdf_fn=source_pdf,
    prior_pdf_fn=prior_pdf,
    njobs=os.cpu_count()-1
)
```

```
---------------------------------------------------------------------------
RemoteTraceback                           Traceback (most recent call last)
RemoteTraceback:
"""
Traceback (most recent call last):
  File "/home/leonardo/py3env/lib/python3.8/site-packages/multiprocess/pool.py",
 ↪line 125, in worker
    result = (True, func(*args, **kwds))
  File "/home/leonardo/py3env/lib/python3.8/site-packages/multiprocess/pool.py",
 ↪line 48, in mapstar
    return list(map(*args))
  File "/home/leonardo/py3env/lib/python3.8/site-packages/pathos/helpers/
 ↪mp_helper.py", line 15, in <lambda>
    func = lambda args: f(*args)
  File "/tmp/ipykernel_45077/2236618269.py", line 26, in __get_posteriori
    B = np.linalg.inv(A_shifted)
  File "<__array_function__ internals>", line 180, in inv
  File "/home/leonardo/py3env/lib/python3.8/site-packages/numpy/linalg/linalg.
 ↪py", line 552, in inv
```

```
    ainv = _umath_linalg.inv(a, signature=signature, extobj=extobj)
  File "/home/leonardo/py3env/lib/python3.8/site-packages/numpy/linalg/linalg.
↪py", line 89, in _raise_linalgerror_singular
    raise LinAlgError("Singular matrix")
numpy.linalg.LinAlgError: Singular matrix
"""

The above exception was the direct cause of the following exception:

LinAlgError                               Traceback (most recent call last)
<timed exec> in <module>

/tmp/ipykernel_45077/2236618269.py in get_posteriori_neighborhood(A, x, T,␣
↪u_vec, v_vec, n_points, source_pdf_fn, prior_pdf_fn, njobs)
     81     )
     82     with pathos.multiprocessing.ProcessingPool(njobs) as p:
---> 83         results = p.map(exec_fn, iterator)
     84
     85     for r in results:

~/py3env/lib/python3.8/site-packages/pathos/multiprocessing.py in map(self, f,␣
↪*args, **kwds)
    137         AbstractWorkerPool._AbstractWorkerPool__map(self, f, *args,␣
↪**kwds)
    138         _pool = self._serve()
--> 139         return _pool.map(star(f), zip(*args)) # chunksize
    140     map.__doc__ = AbstractWorkerPool.map.__doc__
    141     def imap(self, f, *args, **kwds):

~/py3env/lib/python3.8/site-packages/multiprocess/pool.py in map(self, func,␣
↪iterable, chunksize)
    362         in a list that is returned.
    363         '''
--> 364         return self._map_async(func, iterable, mapstar, chunksize).get(
    365
    366     def starmap(self, func, iterable, chunksize=None):

~/py3env/lib/python3.8/site-packages/multiprocess/pool.py in get(self, timeout)
    769             return self._value
    770         else:
--> 771             raise self._value
    772
    773     def _set(self, i, obj):

LinAlgError: Singular matrix
```

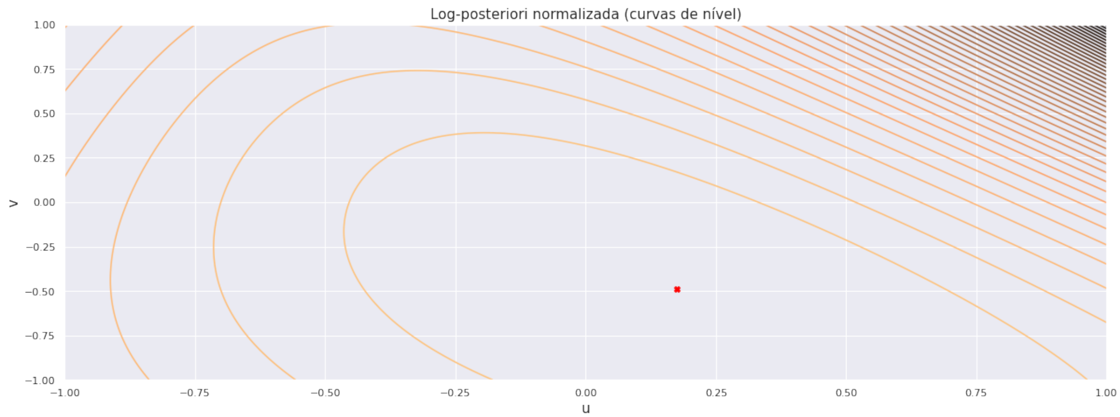
[217]:
```
# Ponto de máximo
max_post_point = np.unravel_index(
    z.argmax(),
    z.shape
)

#
max_post = z[max_post_point]

print('-'*100)
print('Ponto de máximo: u={}, v={}'.format(u_vec[max_post_point[0]],
 ↪v_vec[max_post_point[1]]))
print('-'*100)
```

```
--------------------------------------------------------------------------------
--------------------
Ponto de máximo: u=0.1758793969849246, v=-0.48743718592964824
--------------------------------------------------------------------------------
--------------------
```

[218]:
```
# Plot de curvas de nível
fig = plt.figure(figsize=(20,7))

U, V = np.meshgrid(u_vec, v_vec)

plt.contour(
    U,
    V,
    z.T,
    levels=50,
    cmap='copper'
)

plt.scatter(
    u_vec[max_post_point[0]],
    v_vec[max_post_point[-1]],
    marker='X',
    color='red'
)

plt.xlabel(
    'u',
    fontsize=15
)
plt.ylabel(
    'v',
    fontsize=15
```

```
)

plt.title(
    'Log-posteriori normalizada (curvas de nível)',
    fontsize=15
)
```

[218]: Text(0.5, 1.0, 'Log-posteriori normalizada (curvas de nível)')



[219]:
```
# Plot interativo
fig = go.Figure(
    go.Surface(
        x=u_vec,
        y=v_vec,
        z=z
    )
)

fig.update_layout(
    title='Log-posteriori normalizada (superfície)',
    autosize=False,
    width=500, height=500,
    scene=dict(
        xaxis_title='U',
        yaxis_title='V',
        zaxis_title='Log-posteriori',
    )
)

fig.show()
```

[220]:
```
def source_pdf(x):
    """
        This method takes SCALAR value x and return SCALAR source pdf evaluated␣
 ↪at x.
    """
    return np.exp(-x)/np.square(1+np.exp(-x))


def source_pdf_derivative(x):
    """
        This method takes SCALAR value x and return SCALAR source pdf␣
 ↪derivative evaluated at x.
    """
    return (2*np.exp(-2*x)*(1+np.exp(-x)) - np.exp(-x)*np.square(1+np.exp(-x)))/
 ↪np.power([1+np.exp(-x)],[4])


def prior_pdf(X):
    """
        This method takes VECTOR value X and return SCALAR prior pdf evaluated␣
 ↪at X.
    """
    return 1

def prior_pdf_derivative(X):
    """
        This method takes VECTOR value x and return VECTOR prior pdf derivative␣
 ↪evaluated at X.
    """
    return np.zeros(
        shape=X.shape
    )
```

8.4 3.1. Standard Gradient

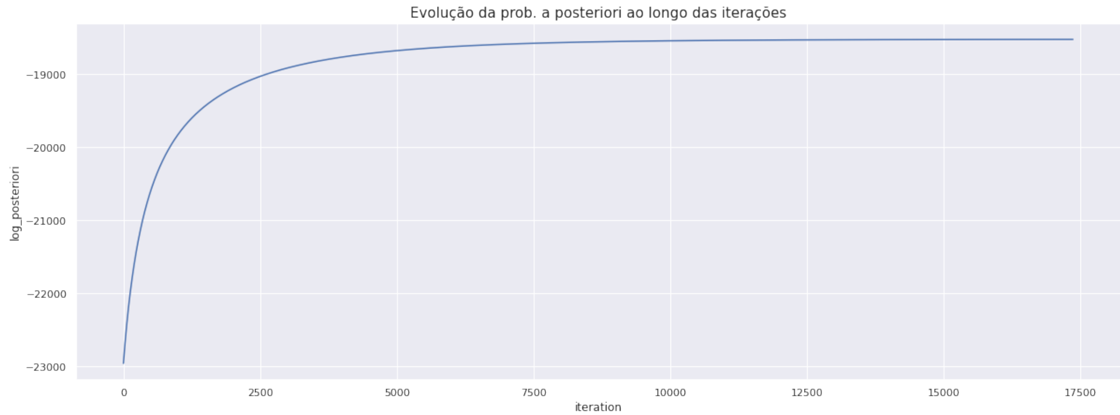
[221]:
```
# Initialize estimator
estimator = MAPGradientEstimator(
    learning_rate=learning_rate,
    thresh=thresh,
    max_it=max_it,
    source_pdf=source_pdf,
    source_pdf_derivative=source_pdf_derivative,
    prior_pdf=prior_pdf,
    prior_pdf_derivative=prior_pdf_derivative,
    is_natural_gradient=False
)
```

[222]:
```
%%time

# Run optimization
estimator.fit(
    s=s,
    x=x,
    initial_condition=initial_B,
    n_initializations=1,
    n_jobs=os.cpu_count()-1
)

# Parse results
B=estimator.fit_results[0]['unmixing_matrix']
logs=estimator.fit_results[0]['logs']

print('-'*100)
print('Total iterations: {}'.format(logs.shape[0]))
print('Final log-posteriori: {}'.format(logs.iloc[-1]['log_posteriori']))
print('-'*100)
```

```
/home/leonardo/git/bayesian_bss/src/estimator.py:144: FutureWarning:

elementwise comparison failed; returning scalar instead, but in the future will
perform elementwise comparison


--------------------------------------------------------------------------------
--------------------
Total iterations: 17358
Final log-posteriori: -18522.17889653367
--------------------------------------------------------------------------------
--------------------
CPU times: user 1h 28min 46s, sys: 15min 33s, total: 1h 44min 19s
Wall time: 1h 14min 11s
```

[223]:
```
fig = plt.figure(figsize=(20,7))
sns.lineplot(
    data=logs,
    x='iteration',
    y='log_posteriori'
)
plt.title(
    'Evolução da prob. a posteriori ao longo das iterações',
    fontsize=15
)
```

[223]:
```
Text(0.5, 1.0, 'Evolução da prob. a posteriori ao longo das iterações')
```



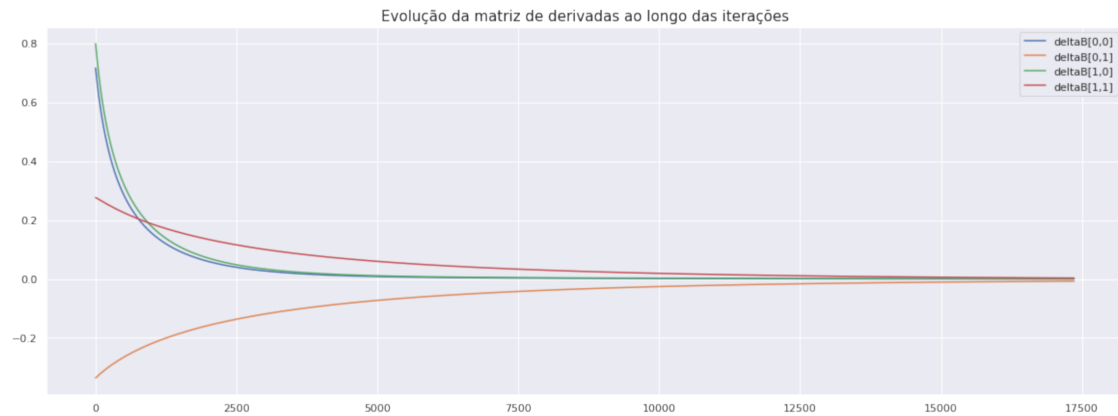
[224]:
```
fig = plt.figure(figsize=(20,7))
sns.lineplot(
    data=logs,
    x='iteration',
    y='detB'
)
plt.title(
    'Evolução do determinante de B ao longo das iterações',
    fontsize=15
)
```

[224]: Text(0.5, 1.0, 'Evolução do determinante de B ao longo das iterações')

[225]:
```
fig = plt.figure(figsize=(20,7))
for i, j in np.ndindex(B.shape):
    plt.plot(
        logs.iteration.values,
```

```
            [row['gradient'][i,j] for n, row in logs.iterrows()],
            label='deltaB[{},{}]'.format(i,j)
        )

plt.title(
    'Evolução da matriz de derivadas ao longo das iterações',
    fontsize=15
)
plt.legend()
```

[225]: <matplotlib.legend.Legend at 0x7f60b2039850>



[226]:
```
B
```

[226]:
```
array([[ 0.95583253, -2.25464401],
       [ 1.12785412,  1.90680978]])
```

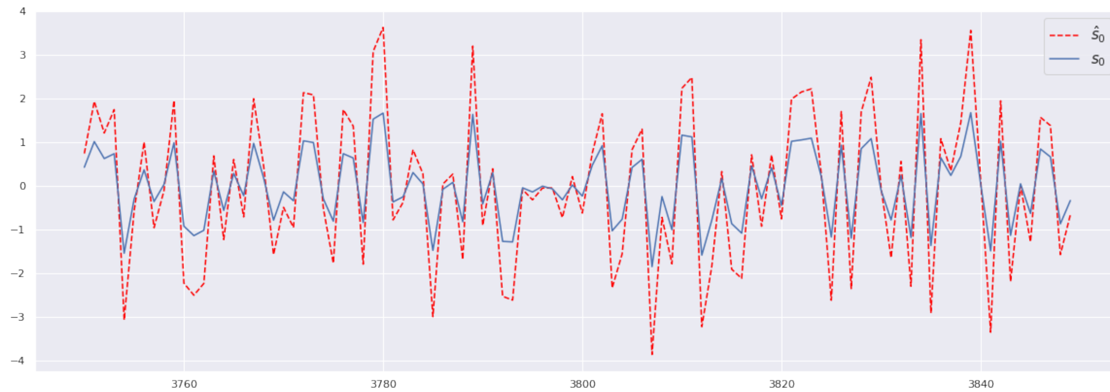
[227]:
```
np.linalg.inv(A)
```

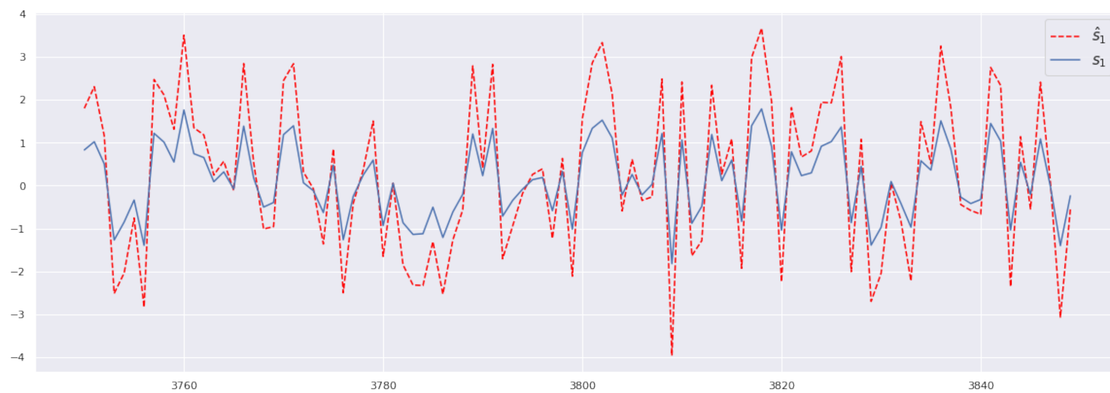
[227]:
```
array([[ 0.5, -1. ],
       [ 0.5,  1. ]])
```

[228]:
```
fig, (ax1, ax2) = plt.subplots(
    nrows=2, ncols=1,
    figsize=(20,15)
)
NPOINTS=100
PLOT_START=NOBS//2
PLOT_END=PLOT_START+NPOINTS

t=range(NOBS)
s_est = B@x
```

```
#Ax1
ax1.plot(
    t[PLOT_START:PLOT_END],
    s_est[0,PLOT_START:PLOT_END],
    label='$\hat{s}_{0}$',
    color='red',
    linestyle='--'
)
ax1.plot(
    t[PLOT_START:PLOT_END],
    s[0,PLOT_START:PLOT_END],
    label='$s_{0}$'
)
ax1.legend(fontsize=15)

#Ax2
ax2.plot(
    t[PLOT_START:PLOT_END],
    s_est[1,PLOT_START:PLOT_END],
    label='$\hat{s}_{1}$',
    color='red',
    linestyle='--'
)
ax2.plot(
    t[PLOT_START:PLOT_END],
    s[1,PLOT_START:PLOT_END],
    label='$s_{1}$'
)
ax2.legend(fontsize=15)
```

```
[228]: <matplotlib.legend.Legend at 0x7f60ad744310>
```





[229]:
```
# Error norm
print('Norma do erro de estimação: {}'.format(
np.linalg.norm(
    np.subtract(s,s_est)
)/np.size(s)))
```

```
Norma do erro de estimação: 0.0072713216386486295
```

8.4.1 2.3.2. MAP - determinant equal to one

[230]:
```
def source_pdf(x):
    return np.exp(-x)/np.square(1+np.exp(-x))

def prior_pdf(B):
    sig=0.1
    desired_det=1
    return (1/np.sqrt(2*np.pi*np.square(sig)))*np.exp(-np.square(np.linalg.
 ↪det(B)-desired_det)/(2*np.square(sig)))
```

[231]:
```
%%time
```

```python
u_lims = (-1, 1)
v_lims = (-1, 1)
n_points = 200

u_vec = np.linspace(u_lims[0], u_lims[-1], n_points)
v_vec = np.linspace(v_lims[0], v_lims[-1], n_points)

z = get_posteriori_neighborhood(
    A=A,
    x=x,
    T=NOBS,
    u_vec=u_vec,
    v_vec=v_vec,
    n_points=n_points,
    source_pdf_fn=source_pdf,
    prior_pdf_fn=prior_pdf,
    njobs=os.cpu_count()-1
)
```

```
---------------------------------------------------------------------------
RemoteTraceback                           Traceback (most recent call last)
RemoteTraceback:
"""
Traceback (most recent call last):
  File "/home/leonardo/py3env/lib/python3.8/site-packages/multiprocess/pool.py"
 ↪line 125, in worker
    result = (True, func(*args, **kwds))
  File "/home/leonardo/py3env/lib/python3.8/site-packages/multiprocess/pool.py"
 ↪line 48, in mapstar
    return list(map(*args))
  File "/home/leonardo/py3env/lib/python3.8/site-packages/pathos/helpers/
 ↪mp_helper.py", line 15, in <lambda>
    func = lambda args: f(*args)
  File "/tmp/ipykernel_45077/2236618269.py", line 26, in __get_posteriori
    B = np.linalg.inv(A_shifted)
  File "<__array_function__ internals>", line 180, in inv
  File "/home/leonardo/py3env/lib/python3.8/site-packages/numpy/linalg/linalg.
 ↪py", line 552, in inv
    ainv = _umath_linalg.inv(a, signature=signature, extobj=extobj)
  File "/home/leonardo/py3env/lib/python3.8/site-packages/numpy/linalg/linalg.
 ↪py", line 89, in _raise_linalgerror_singular
    raise LinAlgError("Singular matrix")
numpy.linalg.LinAlgError: Singular matrix
"""

The above exception was the direct cause of the following exception:
```

```
LinAlgError                               Traceback (most recent call last)
<timed exec> in <module>

/tmp/ipykernel_45077/2236618269.py in get_posteriori_neighborhood(A, x, T,␣
↪u_vec, v_vec, n_points, source_pdf_fn, prior_pdf_fn, njobs)
     81     )
     82     with pathos.multiprocessing.ProcessingPool(njobs) as p:
---> 83         results = p.map(exec_fn, iterator)
     84
     85     for r in results:

~/py3env/lib/python3.8/site-packages/pathos/multiprocessing.py in map(self, f,␣
↪*args, **kwds)
    137         AbstractWorkerPool._AbstractWorkerPool__map(self, f, *args,␣
↪**kwds)
    138         _pool = self._serve()
--> 139         return _pool.map(star(f), zip(*args)) # chunksize
    140     map.__doc__ = AbstractWorkerPool.map.__doc__
    141     def imap(self, f, *args, **kwds):

~/py3env/lib/python3.8/site-packages/multiprocess/pool.py in map(self, func,␣
↪iterable, chunksize)
    362         in a list that is returned.
    363         '''
--> 364         return self._map_async(func, iterable, mapstar, chunksize).get(
    365
    366     def starmap(self, func, iterable, chunksize=None):

~/py3env/lib/python3.8/site-packages/multiprocess/pool.py in get(self, timeout)
    769             return self._value
    770         else:
--> 771             raise self._value
    772
    773     def _set(self, i, obj):

LinAlgError: Singular matrix
```

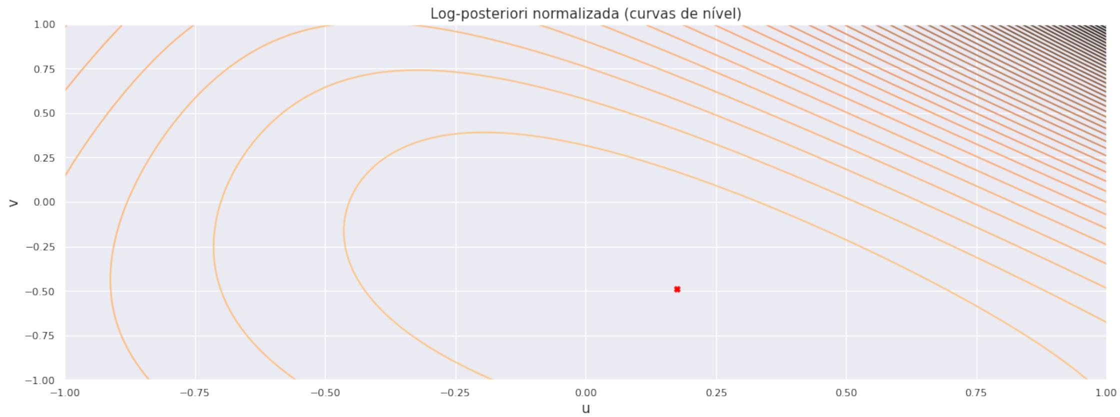
[232]:
```
# Ponto de máximo
max_post_point = np.unravel_index(
    z.argmax(),
    z.shape
)

#
max_post = z[max_post_point]

print('-'*100)
```

```
print('Ponto de máximo: u={}, v={}'.format(u_vec[max_post_point[0]],␣
↪v_vec[max_post_point[1]]))
print('-'*100)
```

```
----------------------------------------------------------------------------------------------------
Ponto de máximo: u=0.1758793969849246, v=-0.48743718592964824
----------------------------------------------------------------------------------------------------
```

[233]:
```
# Plot de curvas de nível
fig = plt.figure(figsize=(20,7))

U, V = np.meshgrid(u_vec, v_vec)

plt.contour(
    U,
    V,
    z.T,
    levels=50,
    cmap='copper'
)

plt.scatter(
    u_vec[max_post_point[0]],
    v_vec[max_post_point[-1]],
    marker='X',
    color='red'
)

plt.xlabel(
    'u',
    fontsize=15
)
plt.ylabel(
    'v',
    fontsize=15
)

plt.title(
    'Log-posteriori normalizada (curvas de nível)',
    fontsize=15
)
```

[233]:
```
Text(0.5, 1.0, 'Log-posteriori normalizada (curvas de nível)')
```



[234]:
```
# Plot interativo
fig = go.Figure(
    go.Surface(
        x=u_vec,
        y=v_vec,
        z=z
    )
)

fig.update_layout(
    title='Log-posteriori normalizada (superfície)',
    autosize=False,
    width=500, height=500,
    scene=dict(
        xaxis_title='U',
        yaxis_title='V',
        zaxis_title='Log-posteriori',
    )
)

fig.show()
```

[235]:
```
def source_pdf(x):
    """
        This method takes SCALAR value x and return SCALAR source pdf evaluated␣
 ↪at x.
    """
    return np.exp(-x)/np.square(1+np.exp(-x))


def source_pdf_derivative(x):
    """
```

```
        This method takes SCALAR value x and return SCALAR source pdf␣
↪derivative evaluated at x.
        """
    return (2*np.exp(-2*x)*(1+np.exp(-x)) - np.exp(-x)*np.square(1+np.exp(-x)))/
↪np.power([1+np.exp(-x)],[4])

def prior_pdf(X):
    sig=0.1
    desired_det=1
    return (1/np.sqrt(2*np.pi*np.square(sig)))*np.exp(-np.square(np.linalg.
↪det(X)-desired_det)/(2*np.square(sig)))

def prior_pdf_derivative(X):
    sig=0.1
    return -1*prior_pdf(X)*(np.linalg.det(X)-1)/np.square(sig)
```

8.5 3.1. Standard Gradient

[236]:
```
# Initialize estimator
estimator = MAPGradientEstimator(
    learning_rate=learning_rate,
    thresh=thresh,
    max_it=max_it,
    source_pdf=source_pdf,
    source_pdf_derivative=source_pdf_derivative,
    prior_pdf=prior_pdf,
    prior_pdf_derivative=prior_pdf_derivative,
    is_natural_gradient=False
)
```

[237]:
```
%%time

# Run optimization
estimator.fit(
    s=s,
    x=x,
    initial_condition=initial_B,
    n_initializations=1,
    n_jobs=os.cpu_count()-1
)

# Parse results
B=estimator.fit_results[0]['unmixing_matrix']
logs=estimator.fit_results[0]['logs']

print('-'*100)
```

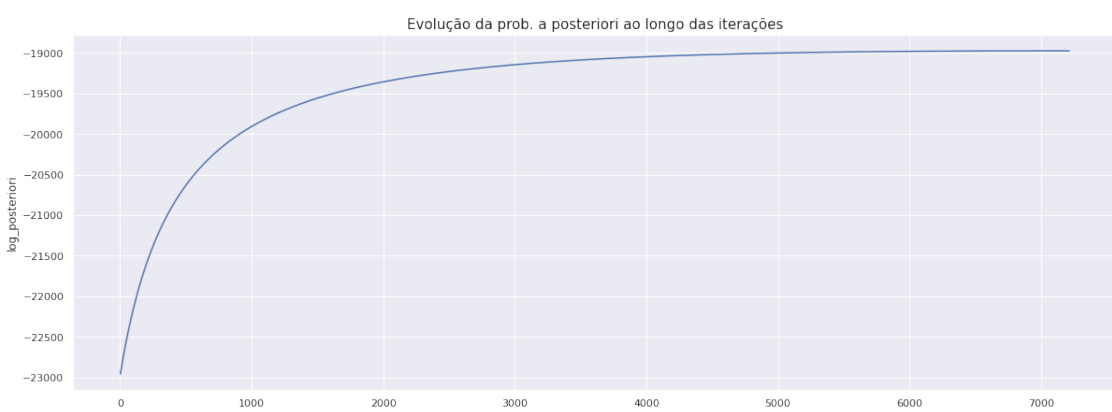
```
print('Total iterations: {}'.format(logs.shape[0]))
print('Final log-posteriori: {}'.format(logs.iloc[-1]['log_posteriori']))
print('-'*100)
```

```
/home/leonardo/git/bayesian_bss/src/estimator.py:144: FutureWarning:

elementwise comparison failed; returning scalar instead, but in the future will
perform elementwise comparison


--------------------------------------------------------------------------------
--------------------
Total iterations: 7215
Final log-posteriori: -18974.77913887492
--------------------------------------------------------------------------------
--------------------
CPU times: user 35min 47s, sys: 6min 32s, total: 42min 19s
Wall time: 29min 42s
```

[238]:

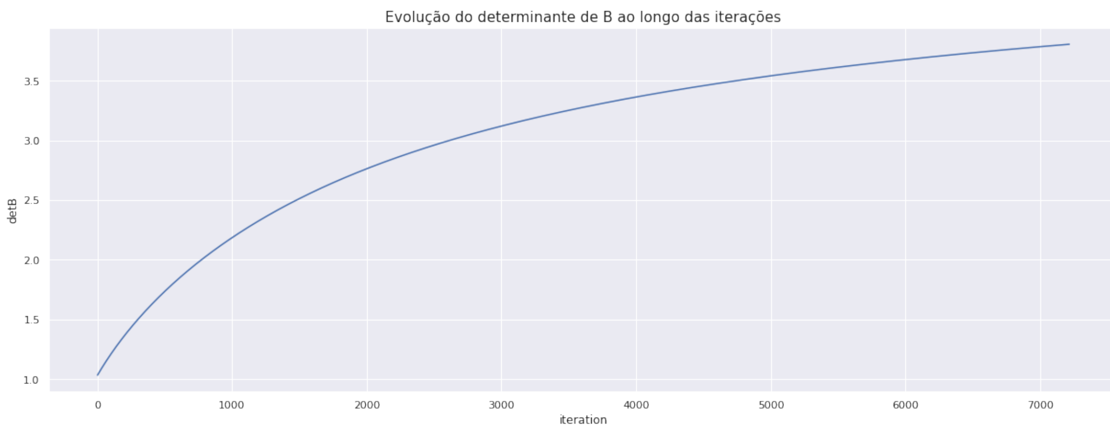
```
fig = plt.figure(figsize=(20,7))
sns.lineplot(
    data=logs,
    x='iteration',
    y='log_posteriori'
)
plt.title(
    'Evolução da prob. a posteriori ao longo das iterações',
    fontsize=15
)
```

[238]: Text(0.5, 1.0, 'Evolução da prob. a posteriori ao longo das iterações')



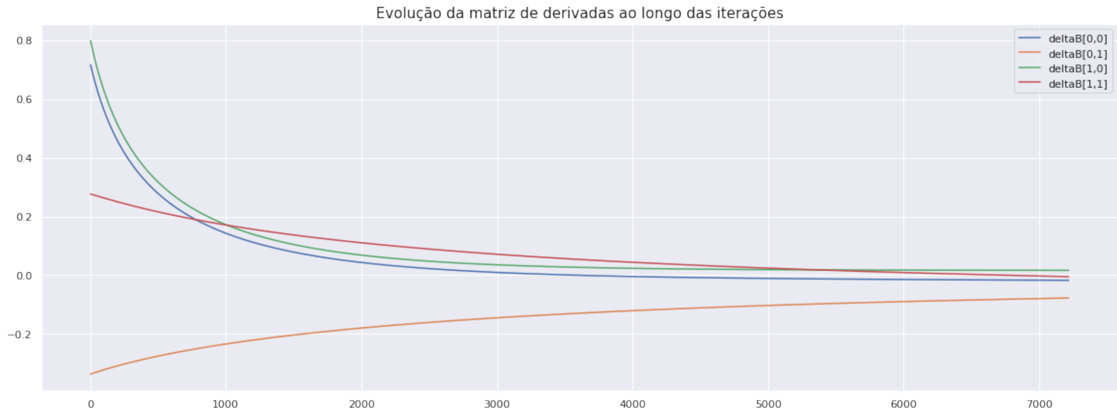
[239]:
```
fig = plt.figure(figsize=(20,7))
sns.lineplot(
    data=logs,
    x='iteration',
    y='detB'
)
plt.title(
    'Evolução do determinante de B ao longo das iterações',
    fontsize=15
)
```

[239]: Text(0.5, 1.0, 'Evolução do determinante de B ao longo das iterações')



[240]:
```
fig = plt.figure(figsize=(20,7))
for i, j in np.ndindex(B.shape):
    plt.plot(
        logs.iteration.values,
        [row['gradient'][i,j] for n, row in logs.iterrows()],
        label='deltaB[{},{}]'.format(i,j)
    )

plt.title(
    'Evolução da matriz de derivadas ao longo das iterações',
    fontsize=15
)
plt.legend()
```

[240]: <matplotlib.legend.Legend at 0x7f60aa84b9d0>



[241]:
```
B
```

[241]:
```
array([[ 0.82494616, -2.23120031],
       [ 1.12810925,  1.56060663]])
```

[242]:
```
np.linalg.inv(A)
```

[242]:
```
array([[ 0.5, -1. ],
       [ 0.5,  1. ]])
```

[243]:
```
fig, (ax1, ax2) = plt.subplots(
    nrows=2, ncols=1,
    figsize=(20,15)
)
NPOINTS=100
PLOT_START=NOBS//2
PLOT_END=PLOT_START+NPOINTS

t=range(NOBS)
s_est = B@x

#Ax1
ax1.plot(
    t[PLOT_START:PLOT_END],
    s_est[0,PLOT_START:PLOT_END],
    label='$\hat{s}_{0}$',
    color='red',
    linestyle='--'
)
ax1.plot(
    t[PLOT_START:PLOT_END],
    s[0,PLOT_START:PLOT_END],
```

```
    label='$s_{0}$'
)
ax1.legend(fontsize=15)

#Ax2
ax2.plot(
    t[PLOT_START:PLOT_END],
    s_est[1,PLOT_START:PLOT_END],
    label='$\hat{s}_{1}$',
    color='red',
    linestyle='--'
)
ax2.plot(
    t[PLOT_START:PLOT_END],
    s[1,PLOT_START:PLOT_END],
    label='$s_{1}$'
)
ax2.legend(fontsize=15)
```

[243]: <matplotlib.legend.Legend at 0x7f60a8748370>

[244]:
```python
# Error norm
print('Norma do erro de estimação: {}'.format(
np.linalg.norm(
    np.subtract(s,s_est)
)/np.size(s)))
```

```
Norma do erro de estimação: 0.006493109914702625
```

8.6 2.3.3. MAP - near-identity transformation

[245]:
```python
def source_pdf(x):
    return np.exp(-x)/np.square(1+np.exp(-x))

def prior_pdf(B):
    sig=0.1
    return np.exp(
        (-1/2/np.square(sig))*np.square(np.linalg.norm(B-np.eye(B.shape[0])))
    )
```

[246]:
```python
%%time

u_lims = (-1, 1)
v_lims = (-1, 1)
n_points = 200

u_vec = np.linspace(u_lims[0], u_lims[-1], n_points)
v_vec = np.linspace(v_lims[0], v_lims[-1], n_points)

z = get_posteriori_neighborhood(
    A=A,
    x=x,
    T=NOBS,
    u_vec=u_vec,
    v_vec=v_vec,
    n_points=n_points,
    source_pdf_fn=source_pdf,
    prior_pdf_fn=prior_pdf,
    njobs=os.cpu_count()-1
)
```

```
---------------------------------------------------------------------------
RemoteTraceback                           Traceback (most recent call last)
RemoteTraceback:
"""
Traceback (most recent call last):
  File "/home/leonardo/py3env/lib/python3.8/site-packages/multiprocess/pool.py"
 ↪line 125, in worker
```

```
    result = (True, func(*args, **kwds))
  File "/home/leonardo/py3env/lib/python3.8/site-packages/multiprocess/pool.py"␣
↪line 48, in mapstar
    return list(map(*args))
  File "/home/leonardo/py3env/lib/python3.8/site-packages/pathos/helpers/
↪mp_helper.py", line 15, in <lambda>
    func = lambda args: f(*args)
  File "/tmp/ipykernel_45077/2236618269.py", line 26, in __get_posteriori
    B = np.linalg.inv(A_shifted)
  File "<__array_function__ internals>", line 180, in inv
  File "/home/leonardo/py3env/lib/python3.8/site-packages/numpy/linalg/linalg.
↪py", line 552, in inv
    ainv = _umath_linalg.inv(a, signature=signature, extobj=extobj)
  File "/home/leonardo/py3env/lib/python3.8/site-packages/numpy/linalg/linalg.
↪py", line 89, in _raise_linalgerror_singular
    raise LinAlgError("Singular matrix")
numpy.linalg.LinAlgError: Singular matrix
"""

The above exception was the direct cause of the following exception:

LinAlgError                               Traceback (most recent call last)
<timed exec> in <module>

/tmp/ipykernel_45077/2236618269.py in get_posteriori_neighborhood(A, x, T,␣
↪u_vec, v_vec, n_points, source_pdf_fn, prior_pdf_fn, njobs)
     81     )
     82     with pathos.multiprocessing.ProcessingPool(njobs) as p:
---> 83         results = p.map(exec_fn, iterator)
     84 
     85     for r in results:

~/py3env/lib/python3.8/site-packages/pathos/multiprocessing.py in map(self, f,␣
↪*args, **kwds)
    137         AbstractWorkerPool._AbstractWorkerPool__map(self, f, *args,␣
↪**kwds)
    138         _pool = self._serve()
--> 139         return _pool.map(star(f), zip(*args)) # chunksize
    140     map.__doc__ = AbstractWorkerPool.map.__doc__
    141     def imap(self, f, *args, **kwds):

~/py3env/lib/python3.8/site-packages/multiprocess/pool.py in map(self, func,␣
↪iterable, chunksize)
    362         in a list that is returned.
    363         '''
--> 364         return self._map_async(func, iterable, mapstar, chunksize).get(
    365 
    366     def starmap(self, func, iterable, chunksize=None):
```

```
~/py3env/lib/python3.8/site-packages/multiprocess/pool.py in get(self, timeout)
    769             return self._value
    770         else:
--> 771             raise self._value
    772 
    773     def _set(self, i, obj):

LinAlgError: Singular matrix
```

[247]:

```
# Ponto de máximo
max_post_point = np.unravel_index(
    z.argmax(),
    z.shape
)

#
max_post = z[max_post_point]

print('-'*100)
print('Ponto de máximo: u={}, v={}'.format(u_vec[max_post_point[0]],␣
 ↪v_vec[max_post_point[1]]))
print('-'*100)
```

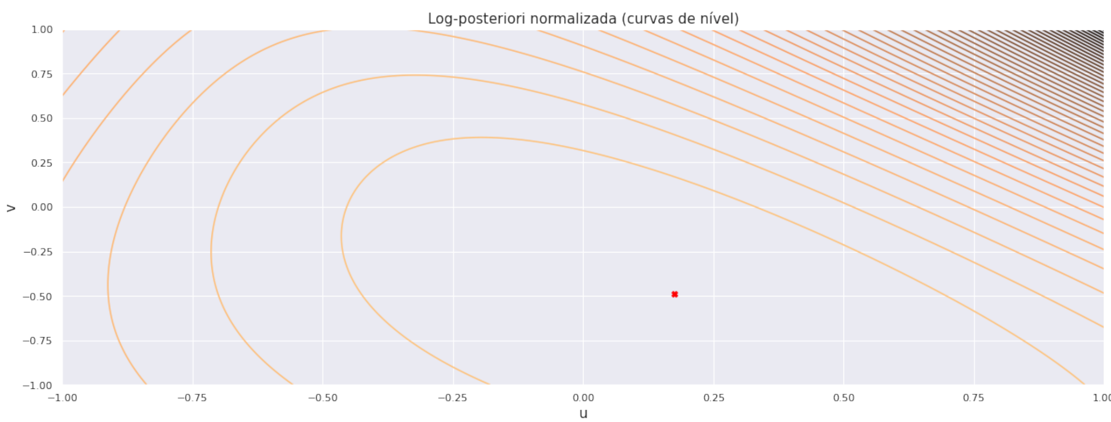
```
--------------------------------------------------------------------------------
--------------------
Ponto de máximo: u=0.1758793969849246, v=-0.48743718592964824
--------------------------------------------------------------------------------
--------------------
```

[248]:

```
# Plot de curvas de nível
fig = plt.figure(figsize=(20,7))

U, V = np.meshgrid(u_vec, v_vec)

plt.contour(
    U,
    V,
    z.T,
    levels=50,
    cmap='copper'
)

plt.scatter(
    u_vec[max_post_point[0]],
    v_vec[max_post_point[-1]],
    marker='X',
```

```
    color='red'
)

plt.xlabel(
    'u',
    fontsize=15
)
plt.ylabel(
    'v',
    fontsize=15
)

plt.title(
    'Log-posteriori normalizada (curvas de nível)',
    fontsize=15
)
```

[248]: Text(0.5, 1.0, 'Log-posteriori normalizada (curvas de nível)')



[249]:
```
# Plot interativo
fig = go.Figure(
    go.Surface(
        x=u_vec,
        y=v_vec,
        z=z
    )
)

fig.update_layout(
    title='Log-posteriori normalizada (superfície)',
    autosize=False,
    width=500, height=500,
```

```
    scene=dict(
        xaxis_title='U',
        yaxis_title='V',
        zaxis_title='Log-posteriori',
    )
)

fig.show()
```

[250]:
```
def source_pdf(x):
    """
        This method takes SCALAR value x and return SCALAR source pdf evaluated␣
 ↪at x.
    """
    return np.exp(-x)/np.square(1+np.exp(-x))


def source_pdf_derivative(x):
    """
        This method takes SCALAR value x and return SCALAR source pdf␣
 ↪derivative evaluated at x.
    """
    return (2*np.exp(-2*x)*(1+np.exp(-x)) - np.exp(-x)*np.square(1+np.exp(-x)))/
 ↪np.power([1+np.exp(-x)],[4])

def prior_pdf(X):
    sig=0.1
    return np.exp(
        (-1/2/np.square(sig))*np.linalg.norm(X-np.eye(X.shape[0]))
    )

def prior_pdf_derivative(X):
    sig=0.1
    der_X=np.empty(X.shape)
    I=np.eye(X.shape[0])
    for i, j in np.ndindex(X.shape):
        der_X[i, j] = prior_pdf(X)*(-1/np.square(sig))*(X[i,j]-I[i,j])
    return der_X
```

8.7 3.1. Standard Gradient

[251]:
```
# Initialize estimator
estimator = MAPGradientEstimator(
    learning_rate=learning_rate,
    thresh=thresh,
    max_it=max_it,
    source_pdf=source_pdf,
```

```
    source_pdf_derivative=source_pdf_derivative,
    prior_pdf=prior_pdf,
    prior_pdf_derivative=prior_pdf_derivative,
    is_natural_gradient=False
)
```

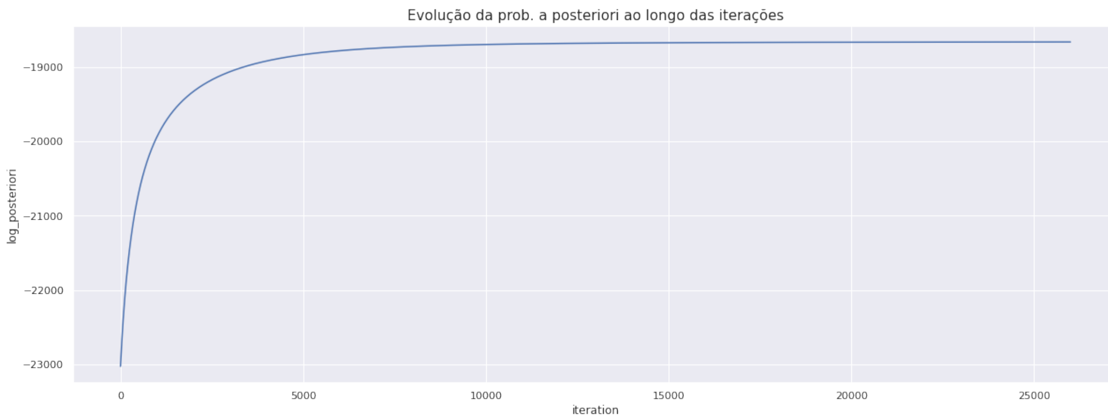
[252]:
```
%%time

# Run optimization
estimator.fit(
    s=s,
    x=x,
    initial_condition=initial_B,
    n_initializations=1,
    n_jobs=os.cpu_count()-1
)

# Parse results
B=estimator.fit_results[0]['unmixing_matrix']
logs=estimator.fit_results[0]['logs']

print('-'*100)
print('Total iterations: {}'.format(logs.shape[0]))
print('Final log-posteriori: {}'.format(logs.iloc[-1]['log_posteriori']))
print('-'*100)
```

```
/home/leonardo/git/bayesian_bss/src/estimator.py:144: FutureWarning:

elementwise comparison failed; returning scalar instead, but in the future will
perform elementwise comparison


--------------------------------------------------------------------------------
--------------------
Total iterations: 25995
Final log-posteriori: -18662.43819753326
--------------------------------------------------------------------------------
--------------------
CPU times: user 2h 9min 12s, sys: 23min 18s, total: 2h 32min 31s
Wall time: 1h 47min
```

[253]:
```
fig = plt.figure(figsize=(20,7))
sns.lineplot(
    data=logs,
    x='iteration',
    y='log_posteriori'
)
plt.title(
```

```
    'Evolução da prob. a posteriori ao longo das iterações',
    fontsize=15
)
```

[253]: Text(0.5, 1.0, 'Evolução da prob. a posteriori ao longo das iterações')

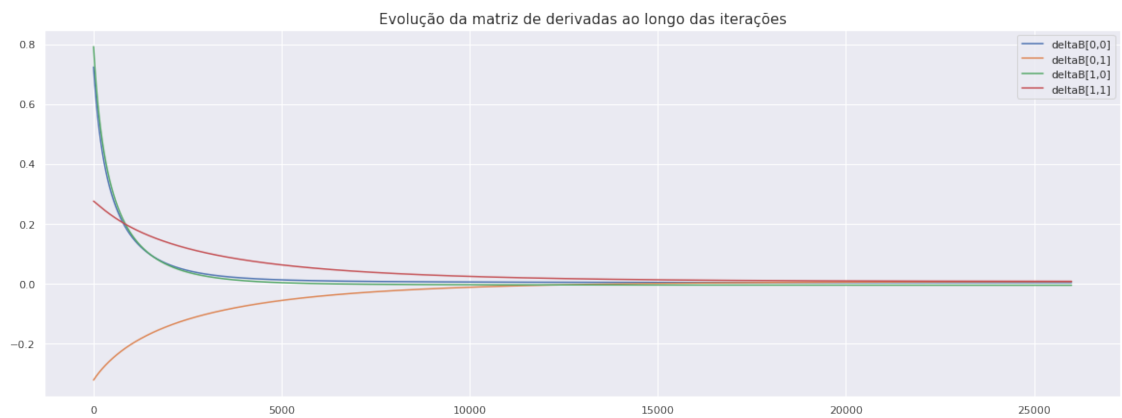


[254]:
```
fig = plt.figure(figsize=(20,7))
sns.lineplot(
    data=logs,
    x='iteration',
    y='detB'
)
plt.title(
    'Evolução do determinante de B ao longo das iterações',
    fontsize=15
)
```

[254]: Text(0.5, 1.0, 'Evolução do determinante de B ao longo das iterações')

[255]:
```
fig = plt.figure(figsize=(20,7))
for i, j in np.ndindex(B.shape):
    plt.plot(
        logs.iteration.values,
        [row['gradient'][i,j] for n, row in logs.iterrows()],
        label='deltaB[{},{}]'.format(i,j)
    )

plt.title(
    'Evolução da matriz de derivadas ao longo das iterações',
    fontsize=15
)
plt.legend()
```

[255]: <matplotlib.legend.Legend at 0x7f6089c99fd0>



[256]:
```
B
```

[256]:
```
array([[ 1.08159803, -1.93411187],
       [ 0.97957127,  2.0802218 ]])
```

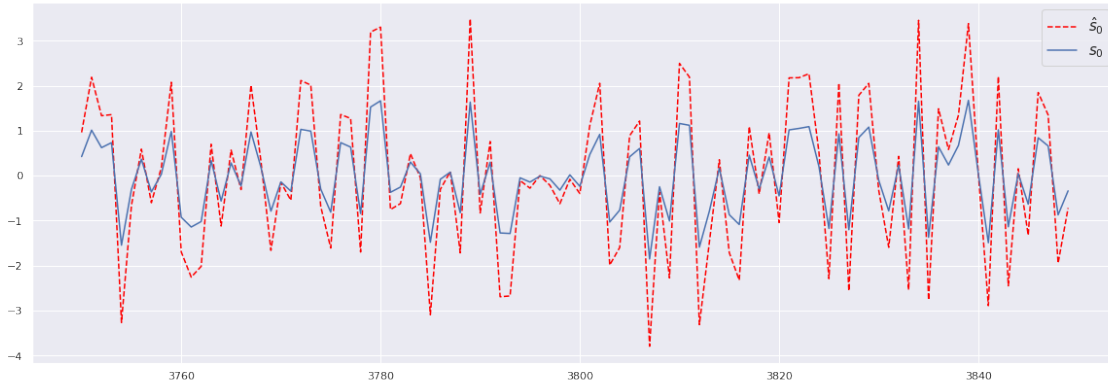
[257]:
```
np.linalg.inv(A)
```

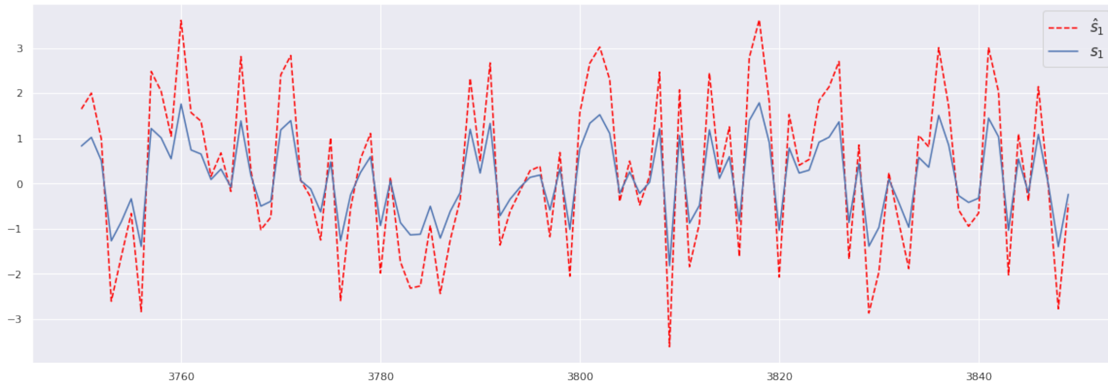
[257]:
```
array([[ 0.5, -1. ],
       [ 0.5,  1. ]])
```

[258]:
```
fig, (ax1, ax2) = plt.subplots(
    nrows=2, ncols=1,
    figsize=(20,15)
)
NPOINTS=100
```

```python
PLOT_START=NOBS//2
PLOT_END=PLOT_START+NPOINTS

t=range(NOBS)
s_est = B@x

#Ax1
ax1.plot(
    t[PLOT_START:PLOT_END],
    s_est[0,PLOT_START:PLOT_END],
    label='$\hat{s}_{0}$',
    color='red',
    linestyle='--'
)
ax1.plot(
    t[PLOT_START:PLOT_END],
    s[0,PLOT_START:PLOT_END],
    label='$s_{0}$'
)
ax1.legend(fontsize=15)

#Ax2
ax2.plot(
    t[PLOT_START:PLOT_END],
    s_est[1,PLOT_START:PLOT_END],
    label='$\hat{s}_{1}$',
    color='red',
    linestyle='--'
)
ax2.plot(
    t[PLOT_START:PLOT_END],
    s[1,PLOT_START:PLOT_END],
    label='$s_{1}$'
)
ax2.legend(fontsize=15)
```

[258]:
```
<matplotlib.legend.Legend at 0x7f6089cc7850>
```





[259]:
```python
# Error norm
print('Norma do erro de estimação: {}'.format(
np.linalg.norm(
    np.subtract(s,s_est)
)/np.size(s)))
```

```
Norma do erro de estimação: 0.0068900262851350515
```

[ ]:

[ ]: