1-MMSE_Metropolis_Hastings

August 6, 2024

[1]:
```
import os
import pickle
import pprint
import itertools
import pathos
import functools
from functools import partial
from pathlib import Path

import pandas as pd
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 MMSEMetropolisHastingsEstimator, InstantaneousMixtureModel,␣
 ↪MCMCGraphPlotter

np.random.seed(2000)
```

[2]:
```
# Whether or not to create folder structure
CREATE_FOLDER_STRUCTURE=False

# Whether or not to save graphs
SAVE_GRAPHS=True

# Whether or not to save source and mixture signals
SAVE_SIGNALS=True

# Whether or not to execute sampling
EXECUTE_SAMPLING=False
```

```
# Number of sources and observations
NSOURCES=2
NOBS=20000

# Execution hyperparameters
N_INITIALIZATIONS = 30
N_WORKERS = os.cpu_count()

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

# MCMC configs
EXPLORATION_VAR=1E-4
N_SAMPLES=200
BURN_IN=0.5

# Plot configs
NBINS=20

# Variável para armazenar erros de estimação em cada caso
errors = {
    'perfect_model': {},
    'slightly_misspecified_model': {}
}
```

[3]:
```
# Folder which will contain output directory tree
OUTPUT_DIR='./output'
base_output_path=Path(OUTPUT_DIR)

# Experiment params
EXPERIMENT_NAME='nobs_20k'
EXPERIMENT_DESCRIPTION="""
    Runs perfect model and slightly misspecified model with 20k observations␣
 ↪and 20k Metropolis-Hastings steps.
"""

# Create experiment directory tree, if so specified
experiment_dir = base_output_path / EXPERIMENT_NAME
perfect_model_dir = experiment_dir / 'perfect_model'
slightly_misspecified_model_dir = experiment_dir / 'slightly_misspecified_model'
if CREATE_FOLDER_STRUCTURE:
    if not experiment_dir.is_dir():
        experiment_dir.mkdir()
```

```
    if not perfect_model_dir.is_dir():
        perfect_model_dir.mkdir()
    if not slightly_misspecified_model_dir.is_dir():
        slightly_misspecified_model_dir.mkdir()

# Write experiment description
if EXECUTE_SAMPLING:
    with (experiment_dir/'experiment_description.txt').open('w') as f:
        f.write(EXPERIMENT_DESCRIPTION)
```

1 ENSAIO - MARKOV CHAIN MONTE CARLO - ALGORITMO METROPOLIS-HASTINGS

Objetivos:

- Implementar o algoritmo Metropolis para obter amostras da distribuição a posteriori da matriz de separação B;

- Estimar a matriz de separação B, variando a escolha de distribuição a priori a configuração de casamento entre estatística das fontes e modelo. Estimações são feitas para critério MMSE, onde se toma o valor esperado da distribuição a posteriori;

- Avaliar qualidade de estimações nos vários casos;

- Avaliar desempenho de algoritmo Metropolis: convergência das amostras, custo computacional, influência de pontos de inicialização, qualidade de estimativas;

Para tal:

- Executa-se algoritmo de amostragem, obtendo estimativa MMSE via simulação de Monte Carlo.

- Avalia-se a convergência da distribuição do processo estocástico resultante do algoritmo Metropolis, bem como da log-posteriori resultante;

- Observam-se as distribuições a posteriori dos coeficientes da matriz B individualmente;

- Utilizam-se as amostras obtidas para estimar matriz B, a qual é utilizada para recuperar as fontes não-observadas;

2 Estimação Pontual Bayesiana via critério Minimum Mean Square Error (MMSE)

De Paulino, Turkman e Murteira (2003): estimativa $\hat{B}_{MMSE}$ é obtida como sendo a média (valor esperado) da distribuição a posteriori. Considerando $\mathcal{B}$ como sendo o conjunto de todas as matrizes B possíveis:

$$\hat{B}_{MMSE} = E[B] = \int_{\mathcal{B}} B \cdot P(B|X) dB$$

Demonstra-se, de maneira simples, que este critério coincide com a minimização do erro quadrático médio, sendo portanto chamado de MMSE. Considera-se que o erro quadrático médio para uma estimativa qualquer $\hat{B}$ seja dada por:

$$MSE = \int_{\mathcal{B}} (\hat{B} - B)^2 \cdot P(B|X) dB$$

3 Simulação de Monte Carlo

De Brémaud (2020): quando não possível ou muito complexa em termos de resolução analítica, a integral para estimação de $\hat{B}_{MMSE}$ pode ser aproximada gerando-se amostras B_i de B segundo a distribuição $P(B|X)$, e computando a média destas amostras. A Lei Forte dos Grandes Números garante a convergência.

$$\hat{B}_{MMSE} = E[B] = \int_{\mathcal{B}} B \cdot P(B|X) dB = \lim_{N \to \infty} \sum_{i=1}^{N} B_i$$

Há, ainda, o desafio de se obter amostras B_i distribuídas de acordo com $P(B|X)$. Para este fim, utilizam-se os métodos MCMC.

4 Métodos de Monte Carlo via Cadeia de Markov (Markov Chain Monte Carlo - MCMC)

De Paulino, Turkman e Murteira (2003):

"A ideia básica por detrás desses métodos é a da transformar o problema estático em consideração num problema de natureza dinâmica, construindo para o efeito um processo estocástico temporal, artificial, que seja fácil de simular, e que convirja, fracamente, para a distribuição original. Este processo temporal é, em geral, uma cadeia de Markov homogênea cuja distribuição de equilíbrio é a distribuição que se pretende simular. Para implementar este método há necessidade de saber construir cadeias de Markov com distribuições de equilíbrio específicas."

5 Algoritmo Metropolis-Hastings

De Mira (2005): no caso de se desejar obter amostras da distribuição a posteriori $P(B|X)$, o algoritmo Metropolis-Hastings (Metropolis et al., 1953; Hastings, 1970) se constitui pela aplicação sequencial da seguinte regra:

1. Dado B_{i}, o valor de B num instante qualquer i, uma movimentação para B_{i+1} é proposta segundo uma distribuição $Q(B_i, B_{i+1})$. Por exemplo, Q pode ser uma distribuição normal centrada em B_i, e com variância pré-determinada.

2. O valor proposto de B_{i+1} será aceito com probabilidade $\alpha(B_i, B_{i+1})$, onde:

$$\alpha(B_i, B_{i+1}) = \min \left[1, \ \frac{P(B_{i+1}|X) \cdot Q(B_i, B_{i+1})}{P(B_i|X) \cdot Q(B_i, B_{i+1})} \right]$$

3. Caso não seja aceito o novo valor, faz-se $B_{i+1} = B_i$.

Obs: caso a função Q seja simétrica, como por exemplo uma distribuição normal, tem-se que $Q(B_i, B_{i+1}) = Q(B_{i+1}, B_i)$. Neste caso, a expressão da probabilidade de aceite se reduz a:

$$\alpha(B_i, B_{i+1}) = \min \left[1, \ \frac{P(B_{i+1}|X)}{P(B_i|X)} \right]$$

Este é o algoritmo originalmente proposto em Metropolis et al. (1953), com a generalização para funções Q não simétricas tendo sido proposta em Hastings, 1970. Neste ensaio, utiliza-se Q normal, e portanto simétrica.

6 Cálculo de $\alpha(B_i, B_{i+1})$

Uma vez que as probabilidades a posteriori envolvidas no cálculo de $\alpha(B_i, B_{i+1})$ são muito pequenas, faz-se uma adaptação das equações envolvidas para que estejam expressas em termos das log-posterioris:

$$\log \alpha(B_i, B_{i+1}) = \min \left[0, \ \log P(B_{i+1}|X) - \log P(B_i|X) \ \right]$$

Portanto, tem-se:

$$\alpha(B_i, B_{i+1}) = \exp \left(\min \left[0, \ \log P(B_{i+1}|X) - \log P(B_i|X) \ \right] \right)$$

Utiliza-se a equação da log-posteriori de Djafari (2000):

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

7 Pseudocódigo Algoritmo

** 1) Inicialização **B(i) ~ N(0,1) B_samples = lista_vazia** 2) Loop principal **para i de 1:N_PASSOS** Gera novo candidato a amostra do processo, baseado na função probabilística Q **B(i+1) ~ Q(B(i), B(i+1))** Calcula probabilidade de aceite alpha(B(i), B(i+1)) **alpha(i+1) <- alpha(B(i), B(i+1))** Obtém variável uniforme u, para acolher nova amostra com probabilidade alpha(B(i), B(i+1)) **u ~ U(0,1)** Acolhe nova amostra de acordo com u e alpha(B(i), B(i+1)) **se u < alpha(i+1): B(i+1) <- B(i+1) se u >= alpha(i+1): B(i+1) <- B(i)** Registra nova amostra ** B_samples.append(B(i+1)) retorna B_samples

8 1. PERFECT MODEL

[4]:
```
model_dir = perfect_model_dir
```

8.1 1.1. Initialize Sources

[5]:
```
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)
    )
```

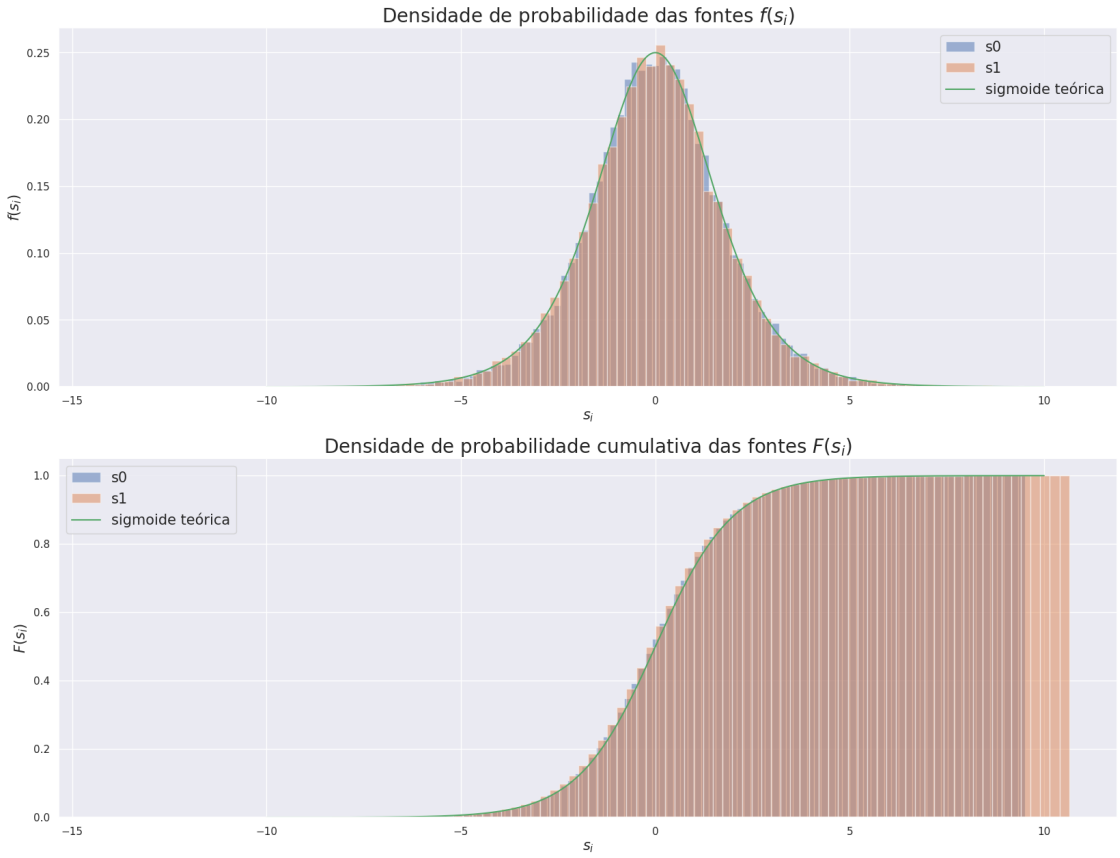
[6]:
```
MU=0.0

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: 20000
CURTOSE s0: 1.0628396785103567
CURTOSE s1: 1.2726001053056777
----------------------------------------------------------------------------------------------------
```

[7]:
```
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)
```



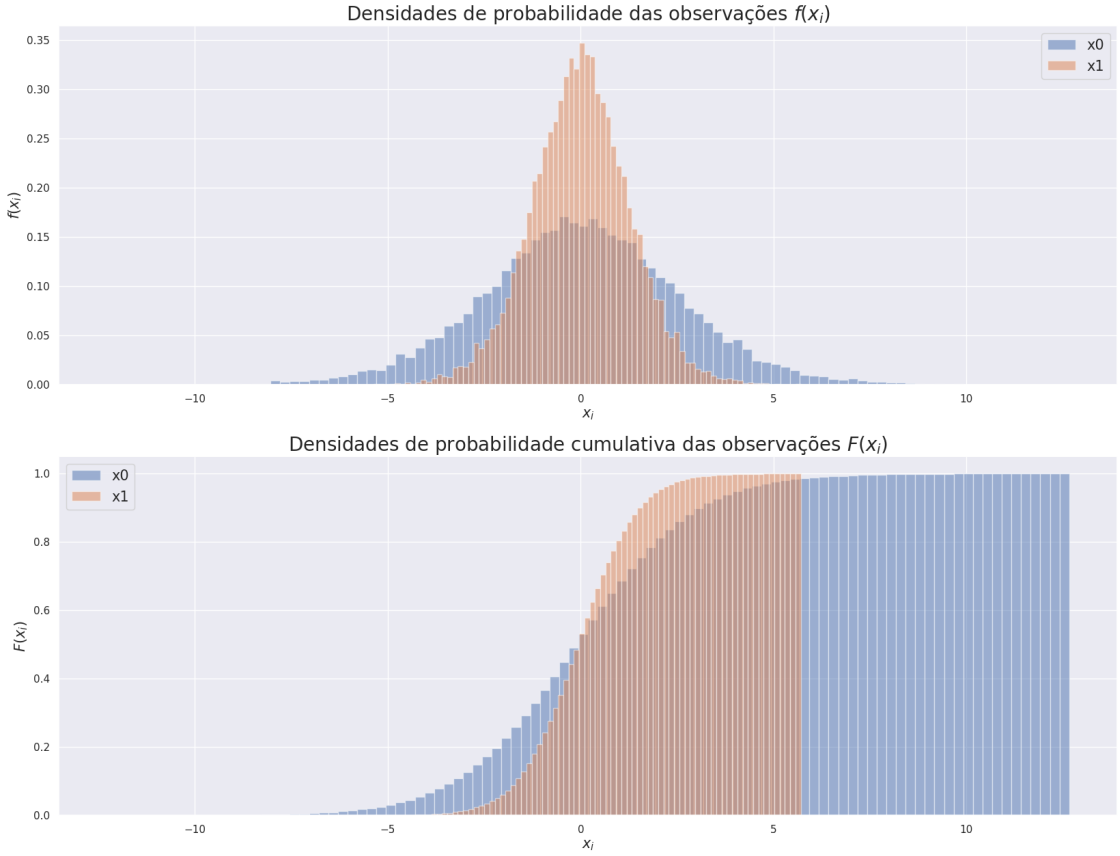
8.2 1.2. Mix sources and generate observations

[8]:
```
# 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]]
```

[9]:
```
# 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)
```



[10]:
```
signal_save_dir = model_dir / 'signals'
if CREATE_FOLDER_STRUCTURE:
    if not signal_save_dir.is_dir():
        signal_save_dir.mkdir()

if SAVE_SIGNALS:
    with (signal_save_dir / 'sources.pkl').open(mode='wb') as f:
        pickle.dump(s, f)

    with (signal_save_dir / 'mixtures.pkl').open(mode='wb') as f:
        pickle.dump(x, f)
```

8.3 1.3. Perform Analysis - LIKELIHOOD

8.3.1 1.3.1 Execute MCMC Sampling

[11]:
```
def source_pdf(x):
    return np.exp(-x)/np.square(1+np.exp(-x))

def prior_pdf(B):
```

```
    return 1

def log_posteriori_fn(
    x,
    source_pdf_fn,
    prior_pdf_fn,
    B
):
    NOBS=x.shape[-1]

    # Cálculo de posteriori para registros
    posteriori = NOBS*np.log(np.abs(np.linalg.det(B)))
    y=B@x
    for i, j in np.ndindex(x.shape):
        posteriori += np.log(source_pdf(y[i,j]))
    posteriori += np.log(prior_pdf(B))

    return posteriori

def proposal_fn(
    exploration_var,
    B
):
    # Calculate shift in parameter
    shift = np.random.normal(
        loc=0,
        scale=np.sqrt(exploration_var),
        size=B.shape
    )

    # Sum shift to obtain new parameter value
    new_B = np.add(
        B, shift
    )

    return new_B
```

[12]:
```
%%time

# Create folder for combination of model specification + prior
combination_dir = model_dir / 'likelihood'
if CREATE_FOLDER_STRUCTURE:
    if not combination_dir.is_dir():
        combination_dir.mkdir()

if EXECUTE_SAMPLING:
    # Fixate arguments in log posteriori fn
```

```
    wrapper_posteriori_fn = functools.partial(
        log_posteriori_fn,
        x,
        source_pdf,
        prior_pdf
    )

    # Fixate arguments in proposal_fn
    wrapper_proposal_fn = functools.partial(
        proposal_fn,
        EXPLORATION_VAR
    )

    # Initialize MH estimator
    estimator = MMSEMetropolisHastingsEstimator(
        n_samples=N_SAMPLES,
        log_posterior_fn=wrapper_posteriori_fn,
        Q=wrapper_proposal_fn,
        burn_in=BURN_IN
    )

    # Execute MCMC estimation
    estimator.fit(
        s,
        x,
        initial_condition=initial_B,
        n_jobs=N_WORKERS
    )

    # Save artifact
    with (combination_dir / 'MH_Estimator.pkl').open('wb') as f:
        pickle.dump(estimator, f)
else:
    # Read artifact
    with (combination_dir / 'MH_Estimator.pkl').open('rb') as f:
        estimator=pickle.load(f)
```

```
CPU times: user 49.5 ms, sys: 88.2 ms, total: 138 ms
Wall time: 137 ms
```

8.3.2 1.3.2. Parse MCMC Results

[13]:
```
# Get results for analyzed model (best model by default)
# analyzed_model_idx = estimator.B_est_idx
analyzed_model_idx=0
B_est = estimator.mcmc_results[analyzed_model_idx]['B_est']
```

```
s_est = B_est@x
samples = estimator.mcmc_results[analyzed_model_idx]['samples']
valid_samples = estimator.mcmc_results[analyzed_model_idx]['valid_samples']
logs = estimator.mcmc_results[analyzed_model_idx]['logs']
```

[14]:
```
print('-'*100)
print('Estimated B:\n{}'.format(B_est))
print('True B:\n{}'.format(np.linalg.inv(A)))
print('-'*100)
```

```
----------------------------------------------------------------------------------------------------
Estimated B:
[[ 0.49840627 -1.00676935]
 [ 0.50055214  0.98801388]]
True B:
[[ 0.5 -1. ]
 [ 0.5  1. ]]
----------------------------------------------------------------------------------------------------
```

[15]:
```
# Posteriors
[r['max_posterior'] for r in estimator.mcmc_results]
```

[15]:
```
[-80053.62176732118,
 -80053.62771851268,
 -80053.63239115704,
 -80053.62884758298,
 -80053.63221666348,
 -80053.66240126094,
 -80053.6347818967,
 -80053.63147347944,
 -80053.63949330601,
 -80053.63958554454,
 -80053.62792769306,
 -80053.65732701735,
 -80053.6557192498,
 -80053.6359056936,
 -80053.62182431784,
 -80053.63298373202,
 -80053.6396446366,
 -80053.62003542551,
 -80053.63051082939,
 -80053.61917193013,
 -80053.6137165805,
 -80053.63378437891,
 -80053.61204634693,
```

```
-80053.66895176163,
-80053.62188382217,
-80053.63202395932,
-80053.60679765086,
-80053.64300021685,
-80053.63288850794,
-80053.62563516496]
```

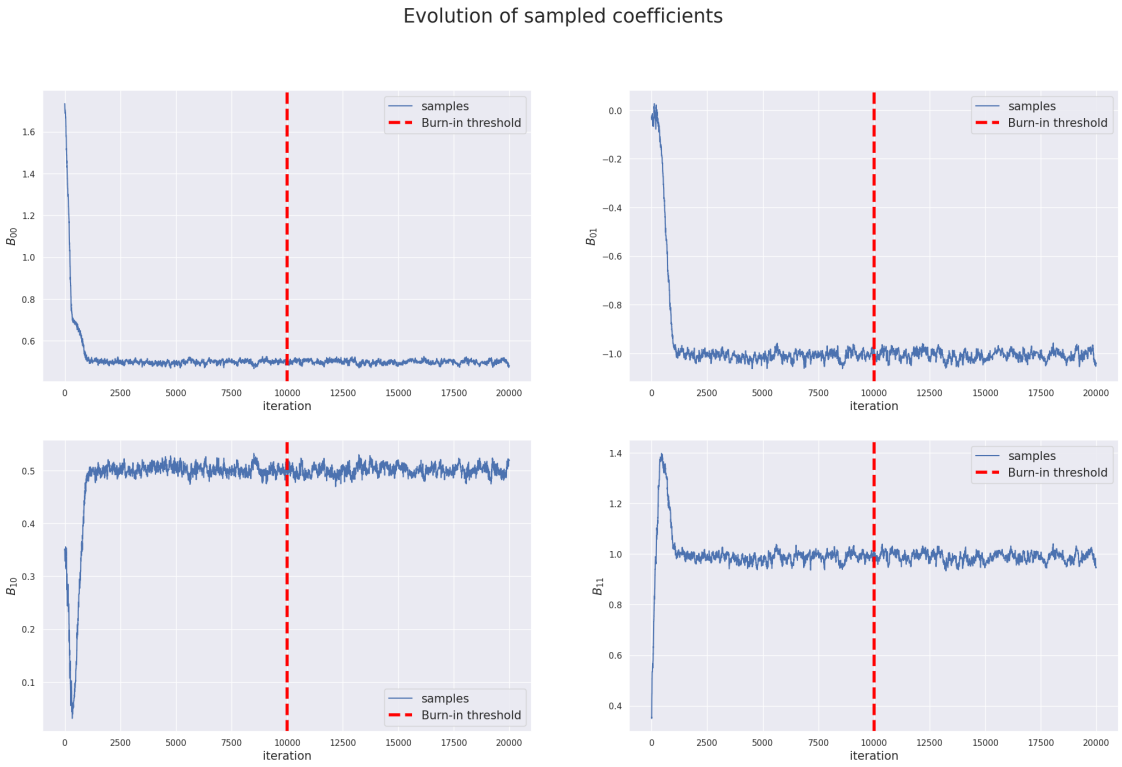
[16]:
```
[r['B_est'] for r in estimator.mcmc_results]
```

[16]:
```
[array([[ 0.49840627, -1.00676935],
        [ 0.50055214,  0.98801388]]),
 array([[ 0.50060068,  0.98823702],
        [-0.4986098 ,  1.00741184]]),
 array([[-0.4967223 ,  1.01182121],
        [-0.50305731, -0.98361323]]),
 array([[-0.49731303,  1.01102617],
        [-0.50236252, -0.98491364]]),
 array([[ 0.49900547, -1.00529921],
        [ 0.50002298,  0.98999177]]),
 array([[-0.50368718, -0.98205282],
        [-0.49594537,  1.01298205]]),
 array([[ 0.5017604 ,  0.98498794],
        [-0.49709398,  1.01085127]]),
 array([[-0.50195003, -0.98595377],
        [ 0.49748504, -1.00932667]]),
 array([[-0.49651651,  1.01241323],
        [-0.50315403, -0.98342302]]),
 array([[-0.49821186,  1.00921793],
        [ 0.50140803,  0.98739873]]),
 array([[ 0.49988858, -1.00320014],
        [ 0.49909556,  0.99152512]]),
 array([[-0.50156644, -0.9873026 ],
        [ 0.49816432, -1.00819486]]),
 array([[-0.49735871,  1.00976618],
        [ 0.50210725,  0.98605317]]),
 array([[-0.50006009,  1.00475664],
        [ 0.4995602 ,  0.99080622]]),
 array([[-0.49763366,  1.00874964],
        [-0.50159929, -0.98595508]]),
 array([[ 0.5011973 ,  0.98719333],
        [-0.4983398 ,  1.00794774]]),
 array([[ 0.49683139, -1.01055384],
        [-0.50242788, -0.98446241]]),
 array([[-0.49887653,  1.00648523],
        [ 0.50052288,  0.98928714]]),
 array([[-0.49879558,  1.007128  ],
```

```
       [ 0.50068275,  0.98814021]]),
 array([[-0.50123833, -0.98778789],
       [-0.49844706,  1.00783565]]),
 array([[-0.49715941,  1.01085411],
       [ 0.5024667 ,  0.98454377]]),
 array([[-0.49856837,  1.00730457],
       [ 0.50083926,  0.98784902]]),
 array([[-0.49826331,  1.00813188],
       [ 0.50114835,  0.98756407]]),
 array([[ 0.50217361,  0.98467908],
       [ 0.49703175, -1.01060703]]),
 array([[ 0.50108431,  0.98719371],
       [-0.49821266,  1.00837356]]),
 array([[-0.49808512,  1.00768263],
       [ 0.50121673,  0.98720037]]),
 array([[-0.49820246,  1.00887881],
       [ 0.50137584,  0.98740409]]),
 array([[-0.49902588,  1.0065207 ],
       [ 0.50040525,  0.98894698]]),
 array([[ 0.50163354,  0.98640551],
       [-0.49761313,  1.00905505]]),
 array([[-0.49866565,  1.00709926],
       [-0.50057493, -0.98782233]])]
```

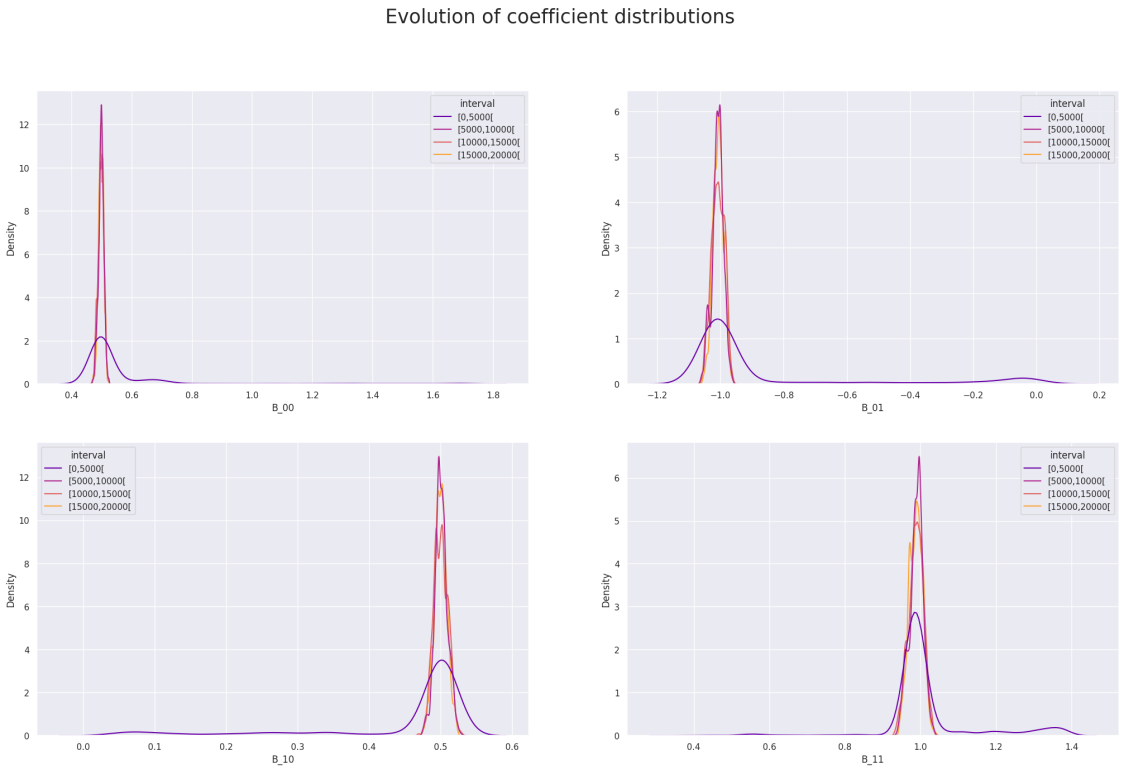
8.3.3 1.3.3. Plot sampled coefficients stochastic process - Markov Chain evolution

[17]:
```
MCMCGraphPlotter.evolution_of_sampled_coefficients(
    samples=samples,
    estimator=estimator,
    logs=logs,
    save_dir=combination_dir if SAVE_GRAPHS else None
)
```



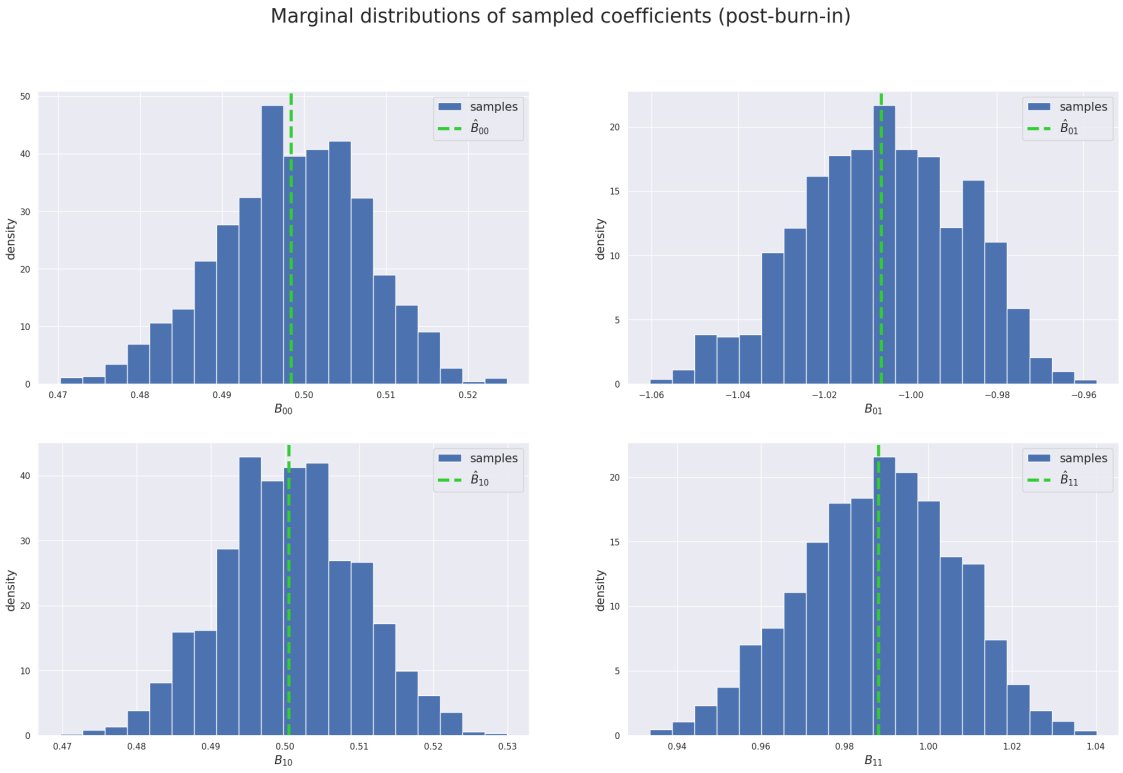
8.3.4 1.3.4. Plot sampled coefficients distributions - Markov Chain evolution

[18]:
```
STEP_SIZE=5000
PALETTE='plasma'

MCMCGraphPlotter.evolution_of_samples_distribution(
    B_est=B_est,
    samples=samples,
    step_size=STEP_SIZE,
    palette=PALETTE,
    save_dir=combination_dir if SAVE_GRAPHS else None
)
```



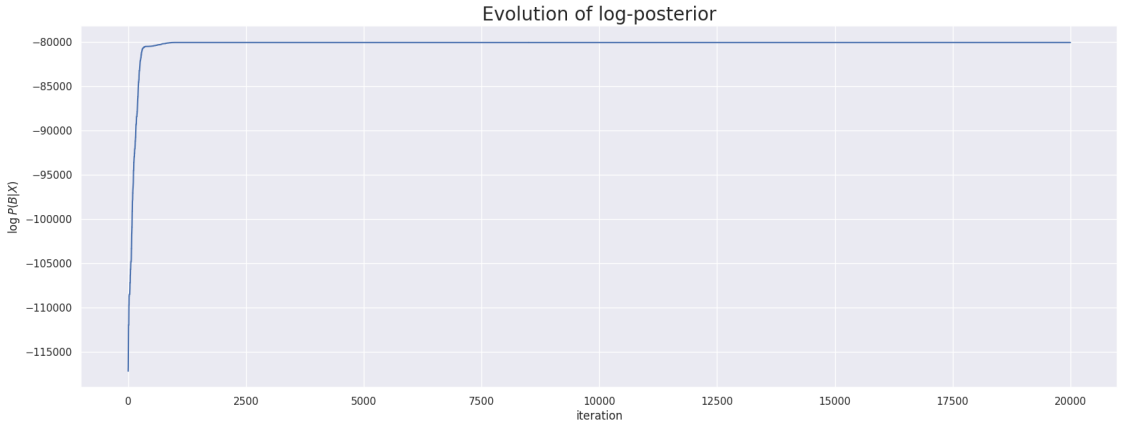
8.3.5 1.3.5. Plot Marginal Posteriors for Coefficients - Post-burn-in

[19]:
```
MCMCGraphPlotter.steady_state_marginal_distributions(
    valid_samples=valid_samples,
    nbins=NBINS,
    B_est=B_est,
    save_dir=combination_dir if SAVE_GRAPHS else None
)
```



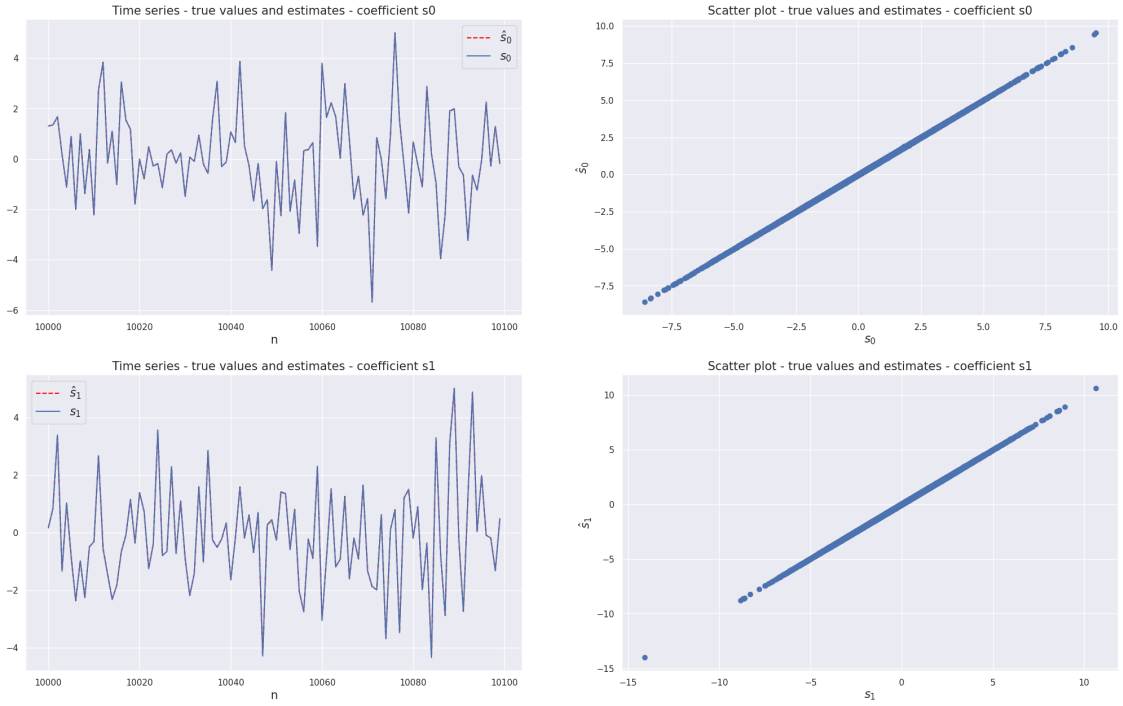
8.3.6 1.3.6. Plot evolution of log-posterior

[20]:
```
MCMCGraphPlotter.evolution_log_posterior(
    logs=logs,
    save_dir=combination_dir if SAVE_GRAPHS else None
)
```



8.4 1.3.7. Plot source separation results

[21]:
```
NPOINTS=100
PLOT_START=NOBS//2
PLOT_END=PLOT_START+NPOINTS

MCMCGraphPlotter.source_separation_results(
    plot_start=PLOT_START,
    plot_end=PLOT_END,
    B_est=B_est,
    s=s,
    s_est=s_est,
    x=x,
    nobs=NOBS,
    save_dir=combination_dir if SAVE_GRAPHS else None
)
```



[22]:
```
# Error norm
err=np.linalg.norm(
    np.subtract(s,s_est)
)/np.size(s)

# Log error
errors['perfect_model']['likelihood'] = err
```

```
print('Norma do erro de estimação: {}'.format(err))
```

```
Norma do erro de estimação: 6.443230754863028e-05
```

8.5 1.4. Perform Analysis - DETERMINANT PRIOR

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

8.5.1 1.4.1 Execute MCMC Sampling

[23]:
```
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)))

def log_posteriori_fn(
    x,
    source_pdf_fn,
    prior_pdf_fn,
    B
):
    NOBS=x.shape[-1]

    # Cálculo de posteriori para registros
    posteriori = NOBS*np.log(np.abs(np.linalg.det(B)))
    y=B@x
    for i, j in np.ndindex(x.shape):
        posteriori += np.log(source_pdf(y[i,j]))
    posteriori += np.log(prior_pdf(B))

    return posteriori

def proposal_fn(
    exploration_var,
    B
):
    # Calculate shift in parameter
    shift = np.random.normal(
        loc=0,
        scale=np.sqrt(exploration_var),
        size=B.shape
    )
```

```
    # Sum shift to obtain new parameter value
    new_B = np.add(
        B, shift
    )

    return new_B
```

[24]:
```
%%time

# Create folder for combination of model specification + prior
combination_dir = model_dir / 'determinant_prior'
if CREATE_FOLDER_STRUCTURE:
    if not combination_dir.is_dir():
        combination_dir.mkdir()

if EXECUTE_SAMPLING:
    # Fixate arguments in log posteriori fn
    wrapper_posteriori_fn = functools.partial(
        log_posteriori_fn,
        x,
        source_pdf,
        prior_pdf
    )

    # Fixate arguments in proposal_fn
    wrapper_proposal_fn = functools.partial(
        proposal_fn,
        EXPLORATION_VAR
    )

    # Initialize MH estimator
    estimator = MMSEMetropolisHastingsEstimator(
        n_samples=N_SAMPLES,
        log_posterior_fn=wrapper_posteriori_fn,
        Q=wrapper_proposal_fn,
        burn_in=BURN_IN
    )

    # Execute MCMC estimation
    estimator.fit(
        s,
        x,
        initial_condition=initial_B,
        n_jobs=N_WORKERS
    )

    # Save artifact
```

```
    with (combination_dir / 'MH_Estimator.pkl').open('wb') as f:
        pickle.dump(estimator, f)
else:
    # Read artifact
    with (combination_dir / 'MH_Estimator.pkl').open('rb') as f:
        estimator=pickle.load(f)
```

```
CPU times: user 58.9 ms, sys: 23.5 ms, total: 82.5 ms
Wall time: 86.7 ms
```

8.5.2 1.4.2. Parse MCMC Results

[25]:
```
# Get results for analyzed model (best model by default)
# analyzed_model_idx = estimator.B_est_idx
analyzed_model_idx = 0
B_est = estimator.mcmc_results[analyzed_model_idx]['B_est']
s_est = B_est@x
samples = estimator.mcmc_results[analyzed_model_idx]['samples']
valid_samples = estimator.mcmc_results[analyzed_model_idx]['valid_samples']
logs = estimator.mcmc_results[analyzed_model_idx]['logs']
```

[26]:
```
print('-'*100)
print('Estimated B:\n{}'.format(B_est))
print('True B:\n{}'.format(np.linalg.inv(A)))
print('-'*100)
```

```
----------------------------------------------------------------------------------------
--------------------
Estimated B:
[[ 0.49871611 -1.00743   ]
 [ 0.50097232  0.98869315]]
True B:
[[ 0.5 -1. ]
 [ 0.5  1. ]]
----------------------------------------------------------------------------------------
--------------------
```

[27]:
```
# Posteriors
[r['max_posterior'] for r in estimator.mcmc_results]
```

[27]:
```
[-80053.60660143394,
 -80053.6343697393,
 -80053.63215393704,
 -80053.6289123958,
 -80053.62113600581,
 -80053.61700970615,
```

```
-80053.63514549304,
-80053.65505414097,
-80053.64497313174,
-80053.64859238772,
-80053.64063177472,
-80053.6488442792,
-80053.62816852065,
-80053.61961635797,
-80053.61960027601,
-80053.62660110756,
-80053.63724373339,
-80053.63920657207,
-80053.6200407372,
-80053.61636565282,
-80053.62881372598,
-80053.61539842251,
-80053.6067639257,
-80053.68365641253,
-80053.6316916774,
-80053.62175274704,
-80053.62532382393,
-80053.61780760702,
-80053.6276333792,
-80053.64438022519]
```

[28]:
```
[r['B_est'] for r in estimator.mcmc_results]
```

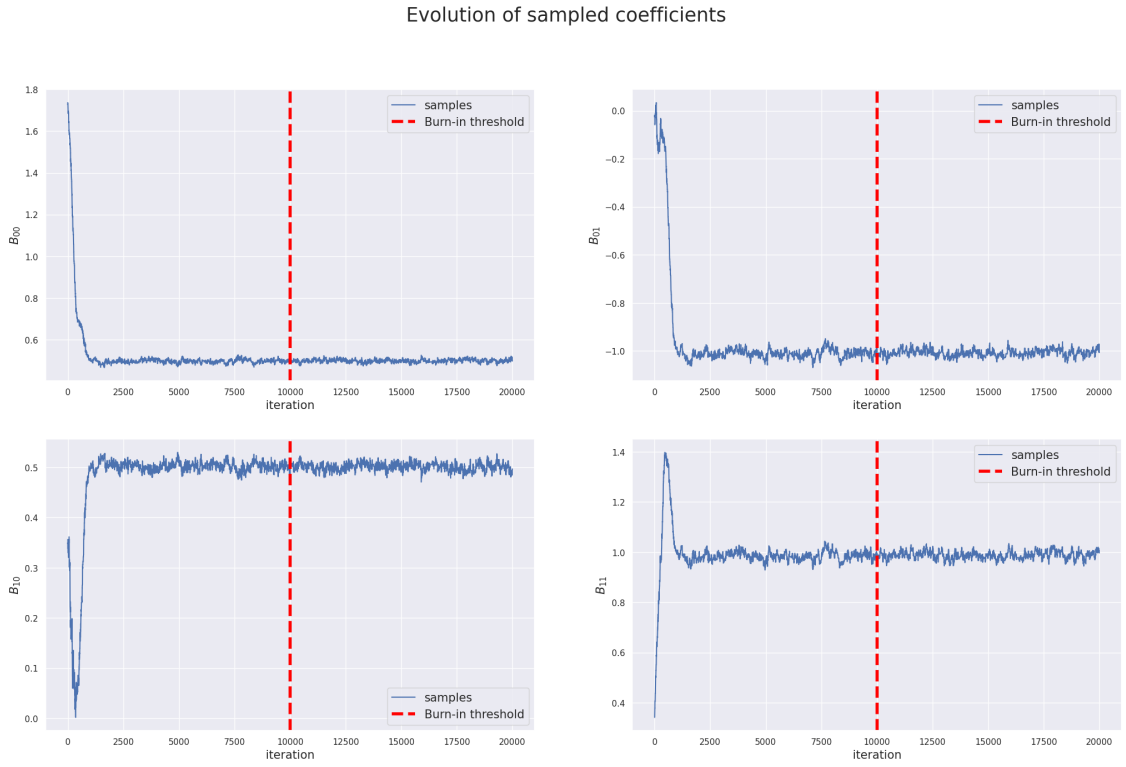
[28]:
```
[array([[ 0.49871611, -1.00743   ],
        [ 0.50097232,  0.98869315]]),
 array([[ 0.50198265,  0.98557608],
        [-0.49713648,  1.00996934]]),
 array([[-0.49843681,  1.0074883 ],
        [-0.50076877, -0.98789725]]),
 array([[-0.49769572,  1.0088967 ],
        [-0.50154268, -0.98601422]]),
 array([[-0.50244781, -0.98499765],
        [ 0.49711006, -1.01070955]]),
 array([[-0.501246  , -0.987231  ],
        [-0.49791571,  1.00886801]]),
 array([[ 0.50139325,  0.98679648],
        [-0.49804155,  1.00876862]]),
 array([[-0.50025163, -0.98939074],
        [ 0.49928822, -1.00607123]]),
 array([[-0.49772669,  1.00896518],
        [-0.5015766 , -0.98637275]]),
 array([[-0.50109241, -0.98747496],
        [-0.49824435,  1.00820399]]),
```

```
array([[ 0.49803165, -1.00878179],
       [ 0.50132798,  0.98673094]]),
array([[-0.50062996, -0.98877587],
       [ 0.49863496, -1.00623269]]),
array([[-0.49709529,  1.01068658],
       [ 0.50209077,  0.98471868]]),
array([[-0.49830148,  1.00747832],
       [ 0.50111256,  0.98809247]]),
array([[-0.49940142,  1.00560914],
       [-0.50014314, -0.98961188]]),
array([[ 0.50028424,  0.98849619],
       [-0.49931046,  1.00668283]]),
array([[ 0.49717345, -1.01026506],
       [-0.50233469, -0.98478475]]),
array([[-0.4965101 ,  1.01079456],
       [ 0.50265087,  0.98393483]]),
array([[-0.49703448,  1.01081812],
       [ 0.50221263,  0.98471655]]),
array([[-0.50143072, -0.98735675],
       [-0.49826836,  1.00926874]]),
array([[ 0.50394735,  0.98097684],
       [ 0.49557896, -1.01377463]]),
array([[-0.49842211,  1.00738171],
       [ 0.50110332,  0.98796691]]),
array([[-0.49842495,  1.00799523],
       [ 0.50103805,  0.98782034]]),
array([[ 0.50171702,  0.98632345],
       [ 0.49769521, -1.00916781]]),
array([[ 0.50241359,  0.9852568 ],
       [-0.49727   ,  1.01032129]]),
array([[-0.49785016,  1.00919515],
       [ 0.50184016,  0.98607442]]),
array([[-0.49631933,  1.0128472 ],
       [ 0.50353305,  0.98321149]]),
array([[-0.4965011 ,  1.01157322],
       [ 0.50294211,  0.98371831]]),
array([[ 0.50012715,  0.98869847],
       [-0.49904377,  1.00590895]]),
array([[-0.49911988,  1.00557256],
       [-0.50038573, -0.98936491]])]
```

8.5.3 1.4.3. Plot sampled coefficients stochastic process - Markov Chain evolution

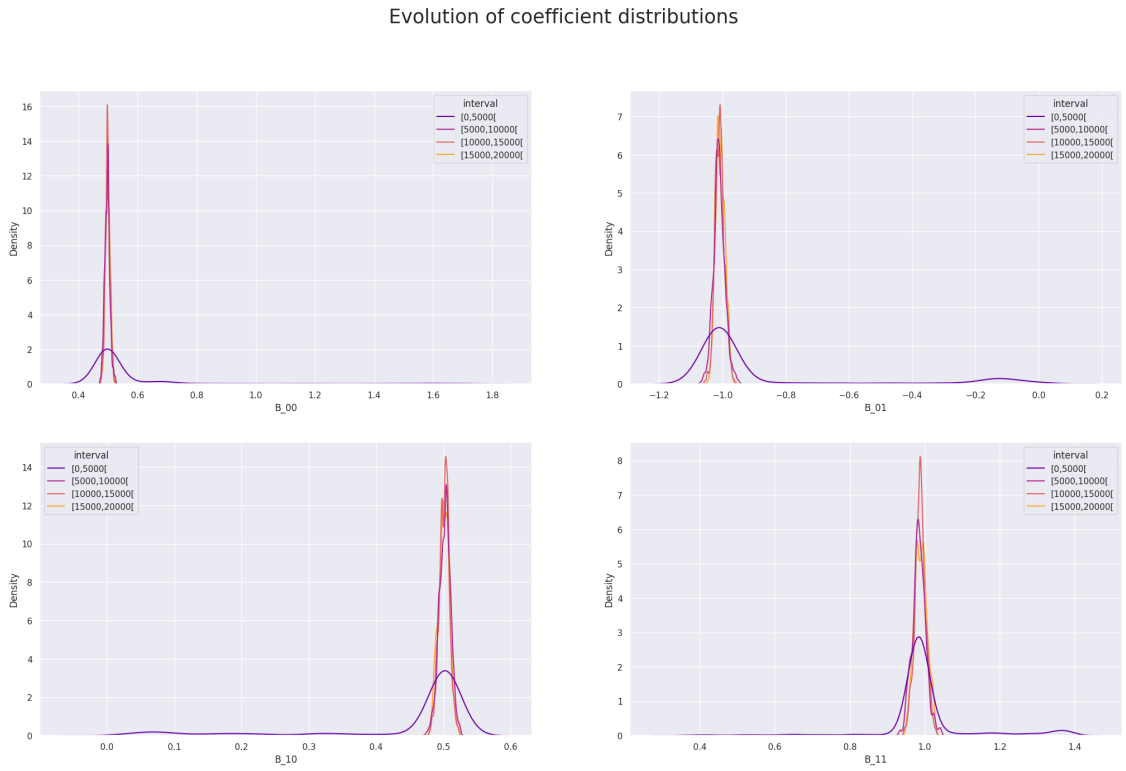
[29]:
```
MCMCGraphPlotter.evolution_of_sampled_coefficients(
    samples=samples,
    estimator=estimator,
    logs=logs,
```

```
    save_dir=combination_dir if SAVE_GRAPHS else None
)
```



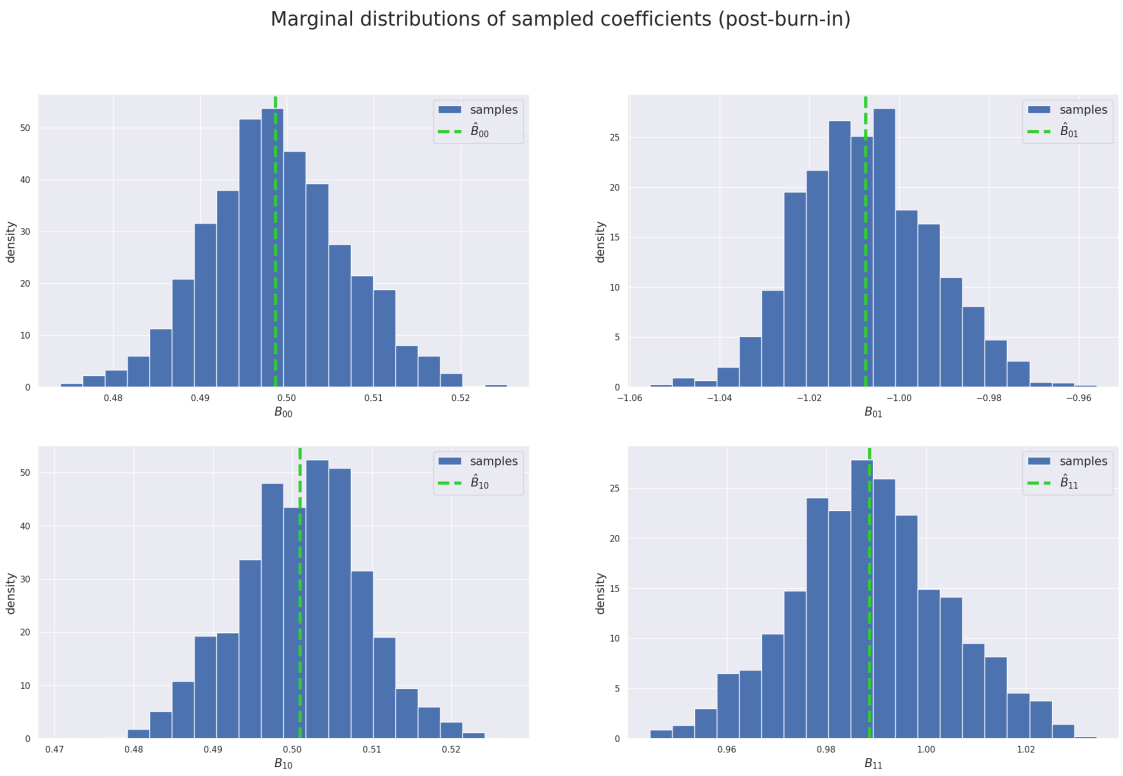
8.5.4 1.4.4. Plot sampled coefficients distributions - Markov Chain evolution

[30]:
```
STEP_SIZE=5000
PALETTE='plasma'

MCMCGraphPlotter.evolution_of_samples_distribution(
    B_est=B_est,
    samples=samples,
    step_size=STEP_SIZE,
    palette=PALETTE,
    save_dir=combination_dir if SAVE_GRAPHS else None
)
```



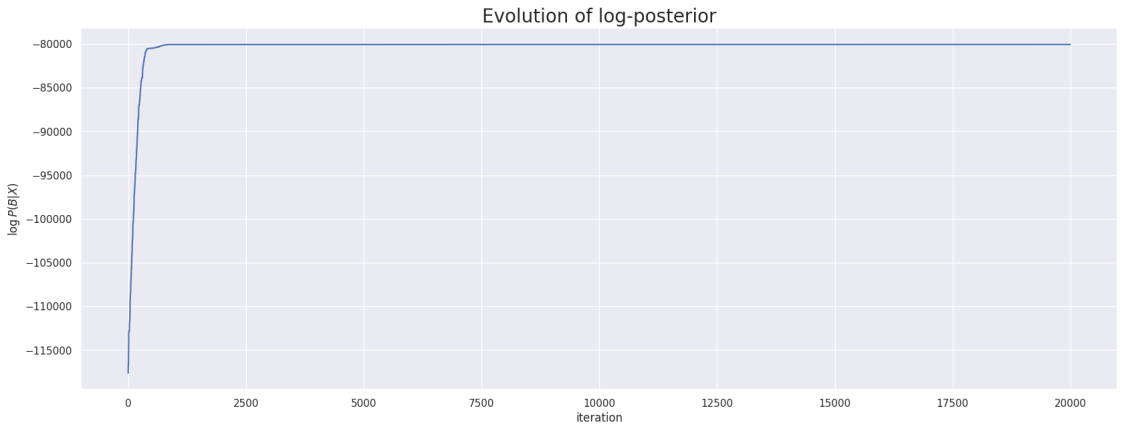
8.5.5 1.4.5. Plot Marginal Posteriors for Coefficients - Post-burn-in

[31]:
```
MCMCGraphPlotter.steady_state_marginal_distributions(
    valid_samples=valid_samples,
    nbins=NBINS,
    B_est=B_est,
    save_dir=combination_dir if SAVE_GRAPHS else None
)
```



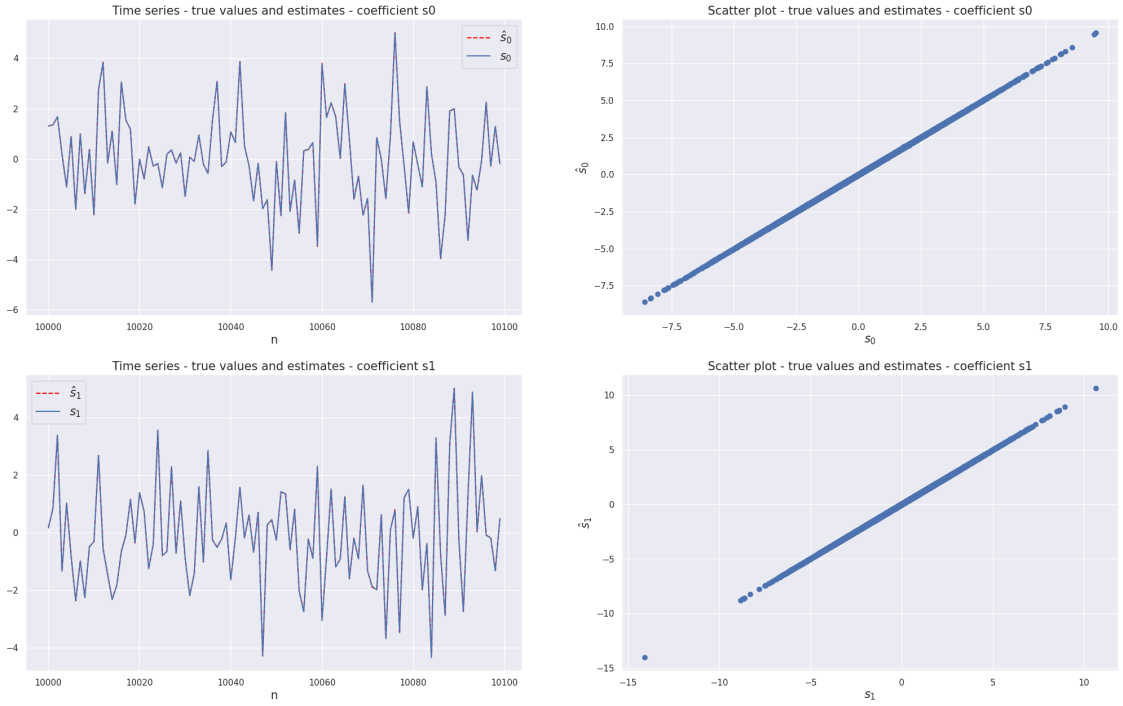
8.5.6 1.4.6. Plot evolution of log-posterior

[32]:
```
MCMCGraphPlotter.evolution_log_posterior(
    logs=logs,
    save_dir=combination_dir if SAVE_GRAPHS else None
)
```



8.6 1.4.7. Plot source separation results

[33]:
```
NPOINTS=100
PLOT_START=NOBS//2
PLOT_END=PLOT_START+NPOINTS

MCMCGraphPlotter.source_separation_results(
    plot_start=PLOT_START,
    plot_end=PLOT_END,
    B_est=B_est,
    s=s,
    s_est=s_est,
    x=x,
    nobs=NOBS,
    save_dir=combination_dir if SAVE_GRAPHS else None
)
```



[34]:
```
# Error norm
err=np.linalg.norm(
    np.subtract(s,s_est)
)/np.size(s)

# Log error
errors['perfect_model']['determinant_prior'] = err
```

```
print('Norma do erro de estimação: {}'.format(err))
```

```
Norma do erro de estimação: 6.319216406611947e-05
```

8.7 1.5. Perform Analysis - near-identity transformation

Prior:

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

8.7.1 1.5.1 Execute MCMC Sampling

[35]:
```
def source_pdf(x):
    return np.exp(-x)/np.square(1+np.exp(-x))

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 log_posteriori_fn(
    x,
    source_pdf_fn,
    prior_pdf_fn,
    B
):
    NOBS=x.shape[-1]

    # Cálculo de posteriori para registros
    posteriori = NOBS*np.log(np.abs(np.linalg.det(B)))
    y=B@x
    for i, j in np.ndindex(x.shape):
        posteriori += np.log(source_pdf(y[i,j]))
    posteriori += np.log(prior_pdf(B))

    return posteriori

def proposal_fn(
    exploration_var,
    B
):
    # Calculate shift in parameter
    shift = np.random.normal(
        loc=0,
        scale=np.sqrt(exploration_var),
        size=B.shape
```

```
    )

    # Sum shift to obtain new parameter value
    new_B = np.add(
        B, shift
    )

    return new_B
```

[36]:
```
%%time

# Create folder for combination of model specification + prior
combination_dir = model_dir / 'identity_transformation'
if CREATE_FOLDER_STRUCTURE:
    if not combination_dir.is_dir():
        combination_dir.mkdir()

if EXECUTE_SAMPLING:
    # Fixate arguments in log posteriori fn
    wrapper_posteriori_fn = functools.partial(
        log_posteriori_fn,
        x,
        source_pdf,
        prior_pdf
    )

    # Fixate arguments in proposal_fn
    wrapper_proposal_fn = functools.partial(
        proposal_fn,
        EXPLORATION_VAR
    )

    # Initialize MH estimator
    estimator = MMSEMetropolisHastingsEstimator(
        n_samples=N_SAMPLES,
        log_posterior_fn=wrapper_posteriori_fn,
        Q=wrapper_proposal_fn,
        burn_in=BURN_IN
    )

    # Execute MCMC estimation
    estimator.fit(
        s,
        x,
        initial_condition=initial_B,
        n_jobs=N_WORKERS
    )
```

```
    # Save artifact
    with (combination_dir / 'MH_Estimator.pkl').open('wb') as f:
        pickle.dump(estimator, f)
else:
    # Read artifact
    with (combination_dir / 'MH_Estimator.pkl').open('rb') as f:
        estimator=pickle.load(f)
```

```
CPU times: user 48.3 ms, sys: 28 ms, total: 76.2 ms
Wall time: 76.4 ms
```

8.7.2 1.5.2. Parse MCMC Results

[37]:
```
# Get results for analyzed model (best model by default)
# analyzed_model_idx = estimator.B_est_idx
analyzed_model_idx = 0
B_est = estimator.mcmc_results[analyzed_model_idx]['B_est']
s_est = B_est@x
samples = estimator.mcmc_results[analyzed_model_idx]['samples']
valid_samples = estimator.mcmc_results[analyzed_model_idx]['valid_samples']
logs = estimator.mcmc_results[analyzed_model_idx]['logs']
```

[38]:
```
print('-'*100)
print('Estimated B:\n{}'.format(B_est))
print('True B:\n{}'.format(np.linalg.inv(A)))
print('-'*100)
```

```
----------------------------------------------------------------------------------------
--------------------
Estimated B:
[[ 0.5032727   0.98352083]
 [-0.49635927  1.01186169]]
True B:
[[ 0.5 -1. ]
 [ 0.5  1. ]]
----------------------------------------------------------------------------------------
--------------------
```

[39]:
```
# Posteriors
[r['max_posterior'] for r in estimator.mcmc_results]
```

[39]:
```
[-80053.6177038962,
 -80053.61430482875,
 -80053.62561235623,
 -80053.6649199096,
```

```
-80053.62535281955,
-80053.61985544483,
-80053.62377264959,
-80053.61532748281,
-80053.61556015376,
-80053.62072291563,
-80053.61727844161,
-80053.62160554595,
-80053.6427582238,
-80053.63921699148,
-80053.61724944801,
-80053.63993418998,
-80053.61102581878,
-80053.61921194491,
-80053.60295179226,
-80053.64208389429,
-80053.62522903606,
-80053.63824202196,
-80053.65014770906,
-80053.61914966135,
-80053.62705803945,
-80053.63812693131,
-80053.62139438765,
-80053.65382877886,
-80053.63493454228,
-80053.61601510951]
```

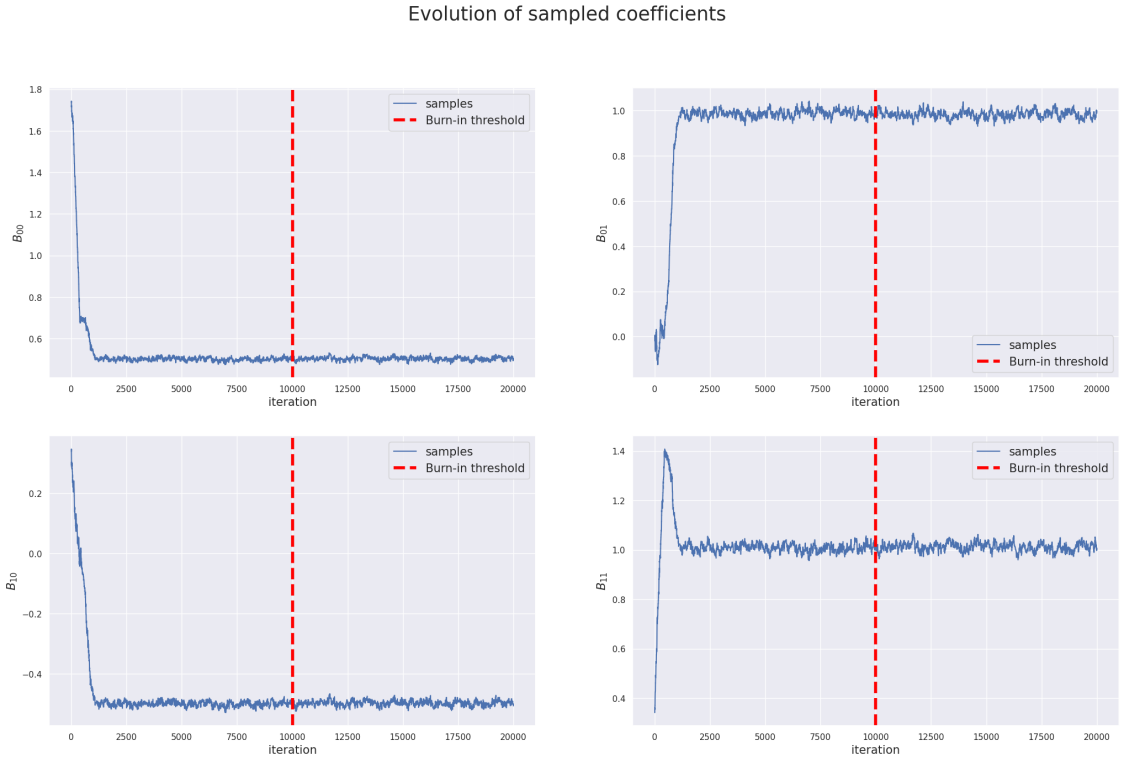
[40]:
```
[r['B_est'] for r in estimator.mcmc_results]
```

[40]:
```
[array([[ 0.5032727 ,  0.98352083],
        [-0.49635927,  1.01186169]]),
 array([[ 0.50031031,  0.98899863],
        [-0.49915191,  1.0062691 ]]),
 array([[-0.50038066,  1.00330093],
        [-0.49902308, -0.99218316]]),
 array([[ 0.50103531,  0.98826264],
        [-0.49857842,  1.0080236 ]]),
 array([[-0.50376705, -0.98293135],
        [ 0.49583836, -1.0135065 ]]),
 array([[-0.49977361, -0.99067316],
        [-0.4997653 ,  1.00531062]]),
 array([[ 0.50322521,  0.98270901],
        [-0.49588728,  1.01328398]]),
 array([[-0.50187854, -0.98533827],
        [ 0.49733537, -1.00963074]]),
 array([[-0.49752044,  1.00988664],
        [-0.50169133, -0.98597751]]),
```

```
array([[ 0.49851246, -1.00697554],
       [ 0.50082664,  0.98775283]]),
array([[ 0.49772475, -1.00835497],
       [ 0.50164807,  0.98641535]]),
array([[-0.50146194, -0.98712629],
       [ 0.49823739, -1.00803344]]),
array([[-0.49841144,  1.00750605],
       [ 0.5010373 ,  0.98768583]]),
array([[-0.49846667,  1.00788348],
       [ 0.50098998,  0.98770391]]),
array([[-0.49810027,  1.0082788 ],
       [-0.50118503, -0.98677721]]),
array([[ 0.50283158,  0.98429775],
       [-0.49681183,  1.01142788]]),
array([[ 0.49823279, -1.00790114],
       [-0.50092943, -0.98832473]]),
array([[-0.49905086,  1.00604735],
       [ 0.50031516,  0.9887821 ]]),
array([[-0.49775661,  1.00929521],
       [ 0.50187041,  0.98627187]]),
array([[-0.5013615 , -0.98770632],
       [-0.49807999,  1.00814781]]),
array([[-0.49875731,  1.00671696],
       [ 0.50080165,  0.98867754]]),
array([[-0.49694765,  1.01116168],
       [ 0.50272075,  0.98438882]]),
array([[-0.49700507,  1.01024537],
       [ 0.50246774,  0.98508573]]),
array([[ 0.50159202,  0.98687272],
       [ 0.4979079 , -1.00869547]]),
array([[ 0.50080903,  0.98787893],
       [-0.49852227,  1.00735122]]),
array([[-0.49918639,  1.00657336],
       [ 0.50045859,  0.98873132]]),
array([[-0.50180691, -0.98645307],
       [-0.49765196,  1.00948993]]),
array([[-0.50145357, -0.98760991],
       [-0.49837461,  1.00828334]]),
array([[ 0.50161237,  0.9861869 ],
       [-0.49753775,  1.00954677]]),
array([[-0.49707577,  1.0110613 ],
       [-0.502374  , -0.98499315]])]
```

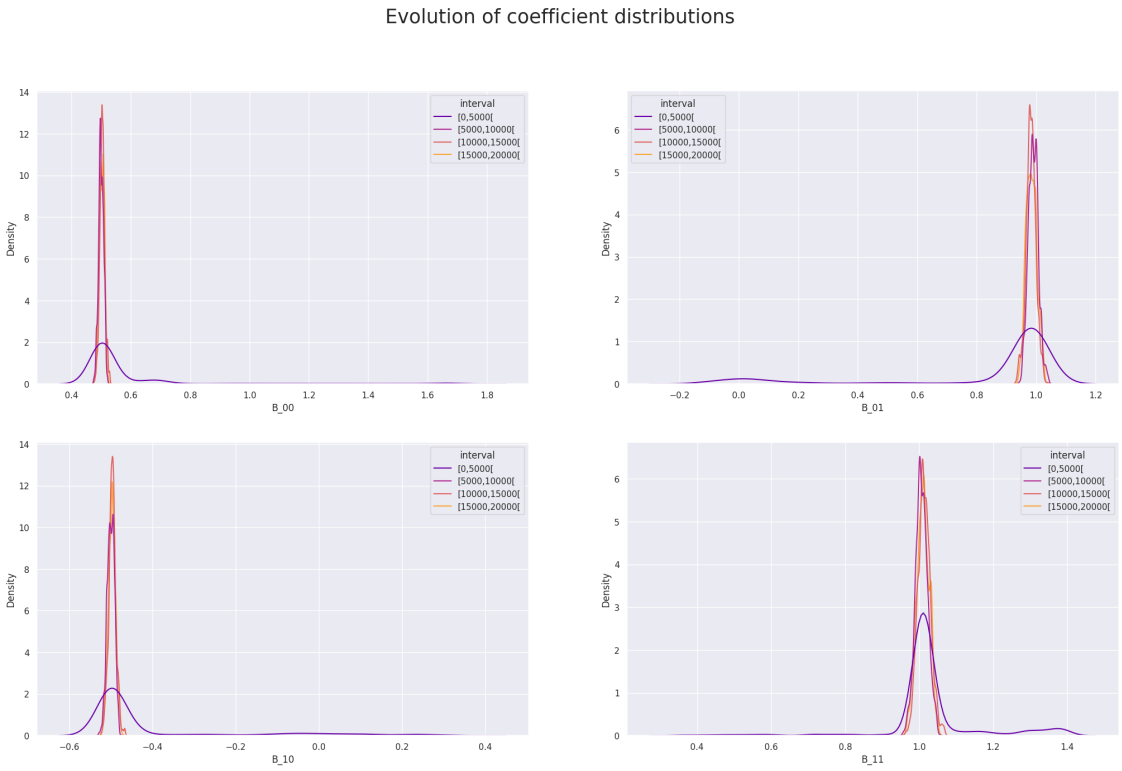
8.7.3 1.5.3. Plot sampled coefficients stochastic process - Markov Chain evolution

[41]:
```
MCMCGraphPlotter.evolution_of_sampled_coefficients(
    samples=samples,
    estimator=estimator,
    logs=logs,
    save_dir=combination_dir if SAVE_GRAPHS else None
)
```



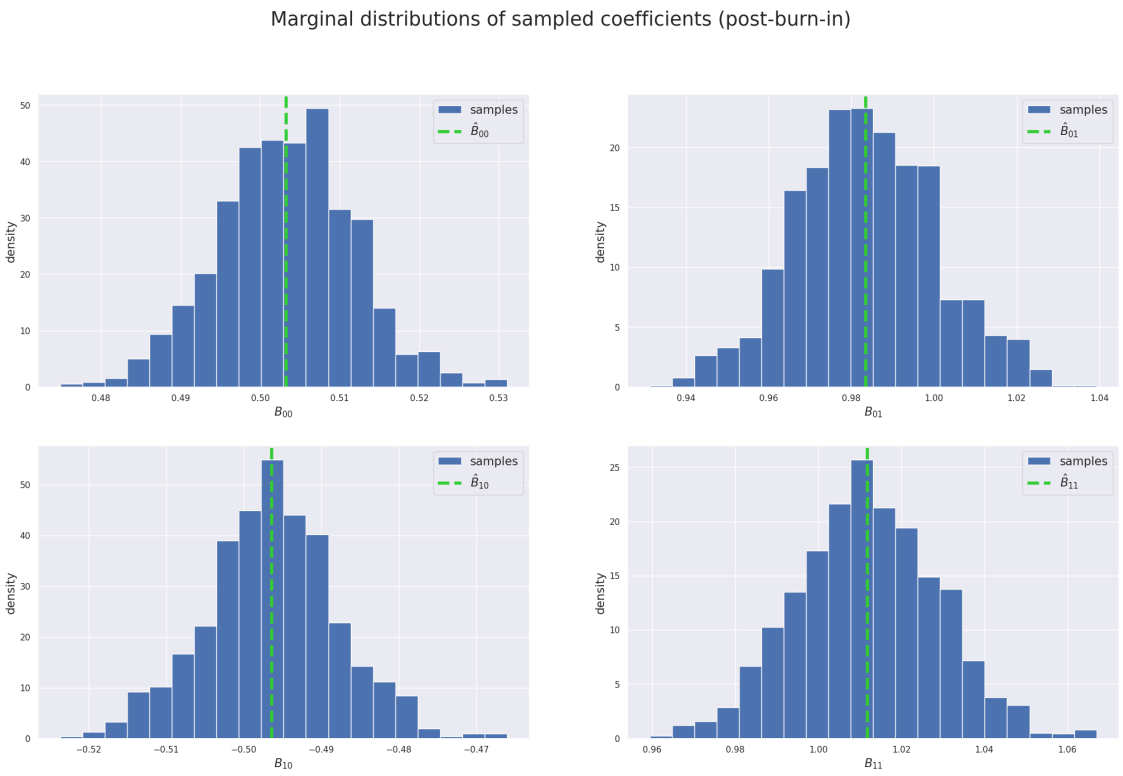
8.7.4 1.5.4. Plot sampled coefficients distributions - Markov Chain evolution

[42]:
```
STEP_SIZE=5000
PALETTE='plasma'

MCMCGraphPlotter.evolution_of_samples_distribution(
    B_est=B_est,
    samples=samples,
    step_size=STEP_SIZE,
    palette=PALETTE,
    save_dir=combination_dir if SAVE_GRAPHS else None
)
```



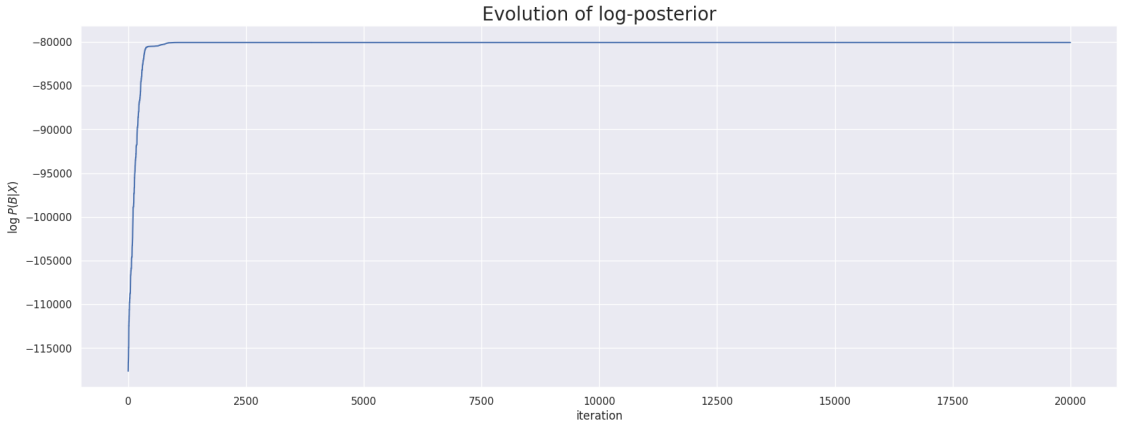
8.7.5 1.5.5. Plot Marginal Posteriors for Coefficients - Post-burn-in

[43]:
```
MCMCGraphPlotter.steady_state_marginal_distributions(
    valid_samples=valid_samples,
    nbins=NBINS,
    B_est=B_est,
    save_dir=combination_dir if SAVE_GRAPHS else None
)
```



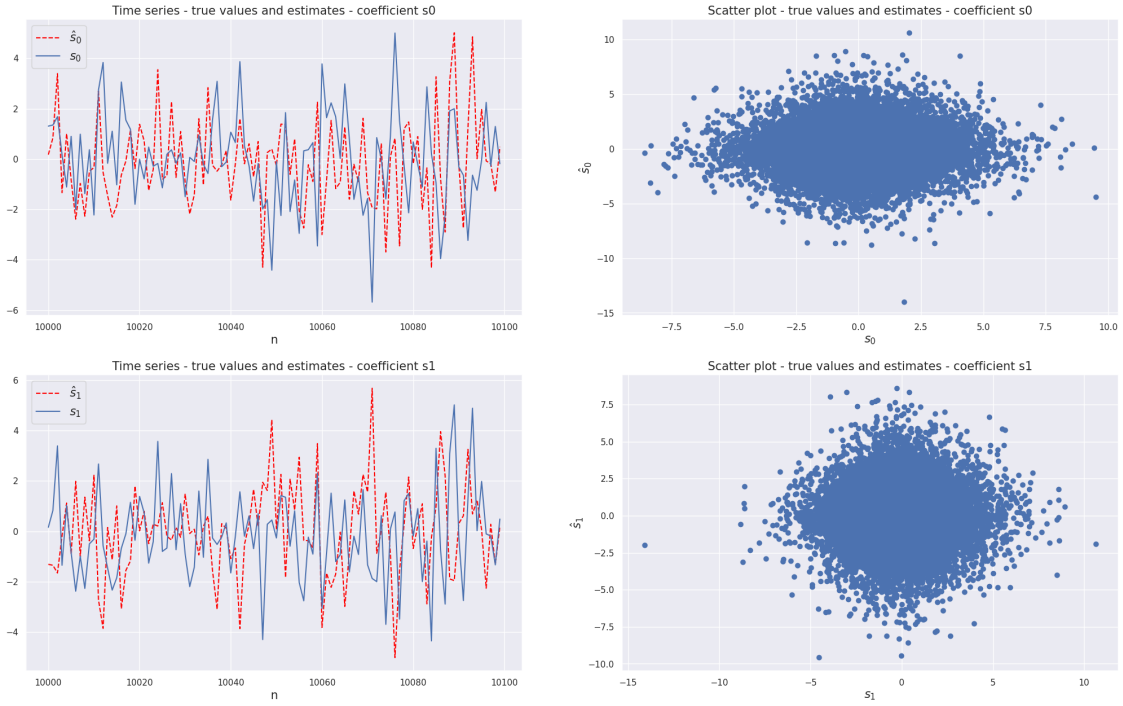
8.7.6 1.5.6. Plot evolution of log-posterior

[44]:
```
MCMCGraphPlotter.evolution_log_posterior(
    logs=logs,
    save_dir=combination_dir if SAVE_GRAPHS else None
)
```



8.8 1.5.7. Plot source separation results

[45]:
```
NPOINTS=100
PLOT_START=NOBS//2
PLOT_END=PLOT_START+NPOINTS

MCMCGraphPlotter.source_separation_results(
    plot_start=PLOT_START,
    plot_end=PLOT_END,
    B_est=B_est,
    s=s,
    s_est=s_est,
    x=x,
    nobs=NOBS,
    save_dir=combination_dir if SAVE_GRAPHS else None
)
```



[46]:
```
# Error norm
err=np.linalg.norm(
    np.subtract(s,s_est)
)/np.size(s)

# Log error
errors['perfect_model']['near_identity_prior'] = err
```

```
print('Norma do erro de estimação: {}'.format(err))
```

```
Norma do erro de estimação: 0.012761931151685916
```

9 2. SLIGHTLY MISSPECIFIED MODEL

[47]:
```
model_dir = slightly_misspecified_model_dir
```

9.1 2.1. Initialize Sources

[48]:
```
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)
    )
```

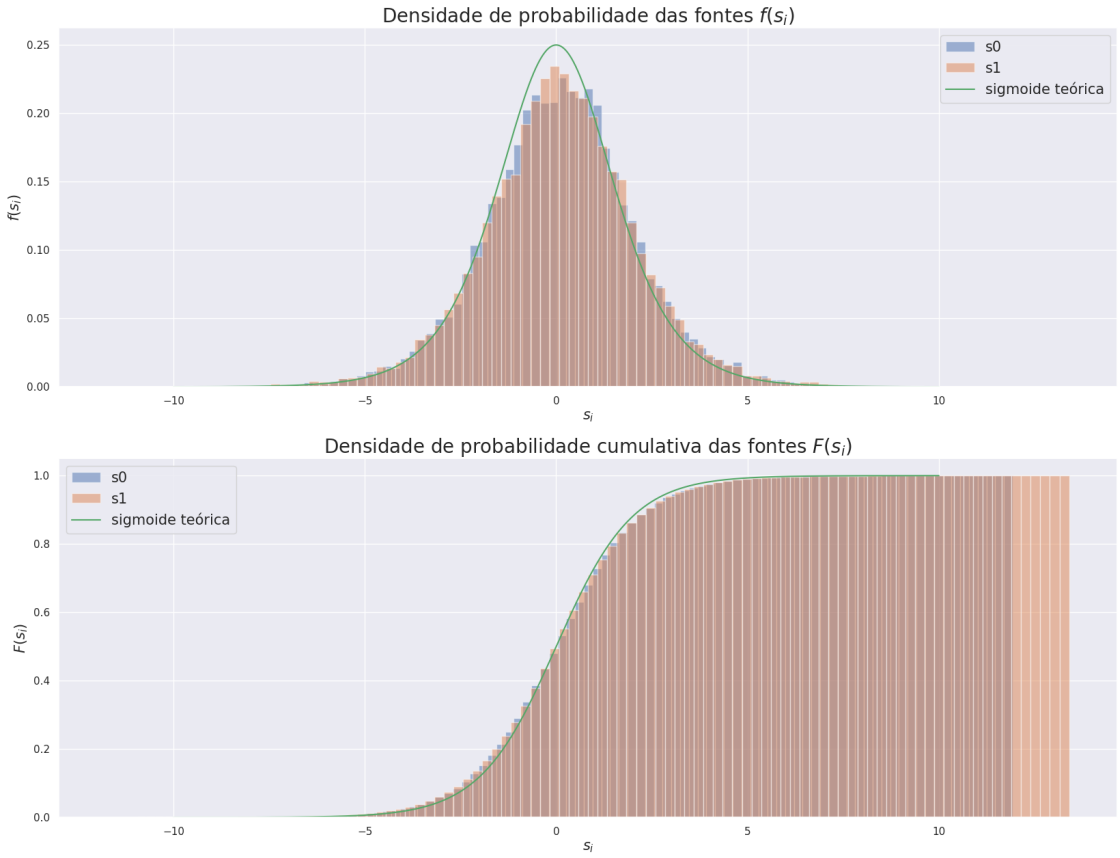
[49]:
```
MU=0.1
SCALE=1.1

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: 20000
CURTOSE s0: 1.1486915949312184
CURTOSE s1: 1.2403528187038688
----------------------------------------------------------------------------------------
--------------------
```

[50]:
```
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)
```



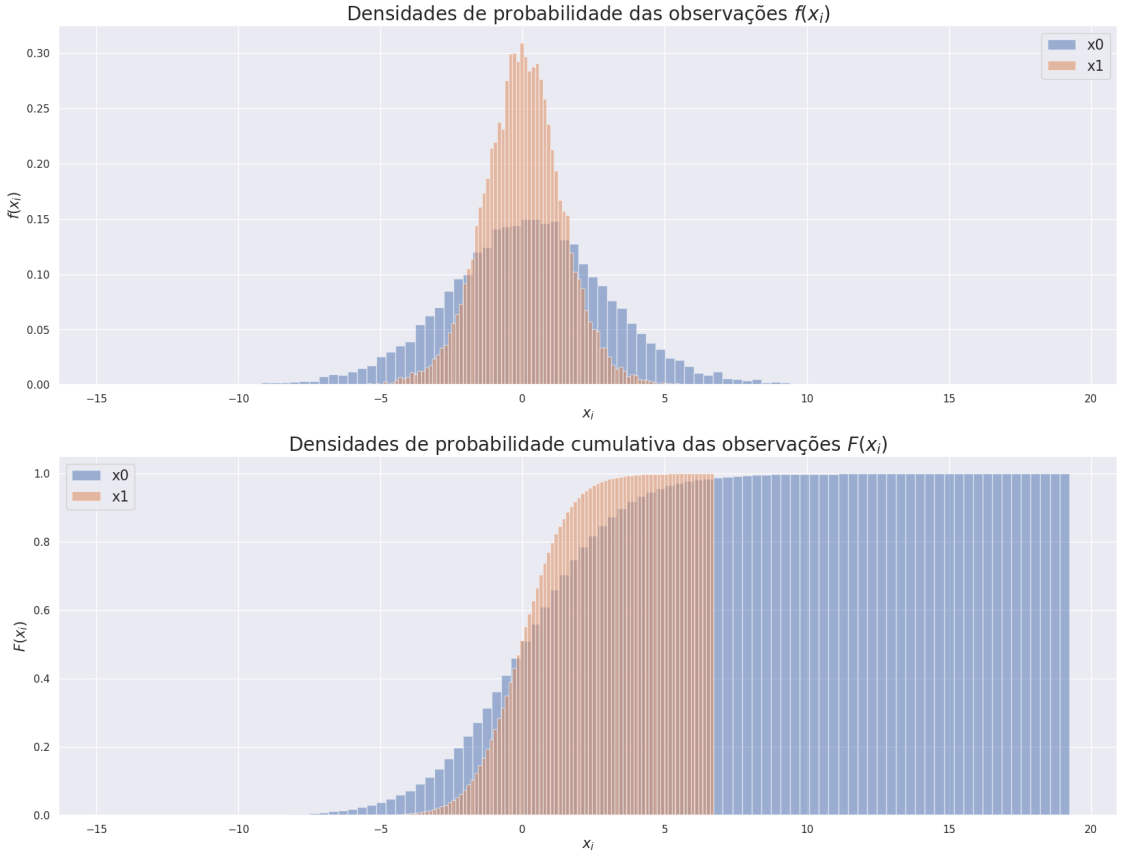
9.2 2.2. Mix sources and generate observations

[51]:
```
# 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]]
```

[52]:
```
# 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)
```



[53]:
```
if SAVE_SIGNALS:
    signal_save_dir = model_dir / 'signals'
    if not signal_save_dir.is_dir():
        signal_save_dir.mkdir()

    with (signal_save_dir / 'sources.pkl').open(mode='wb') as f:
        pickle.dump(s, f)

    with (signal_save_dir / 'mixtures.pkl').open(mode='wb') as f:
        pickle.dump(x, f)
```

9.3 2.3. Perform Analysis - LIKELIHOOD

9.3.1 2.3.1 Execute MCMC Sampling

[54]:
```
def source_pdf(x):
    return np.exp(-x)/np.square(1+np.exp(-x))

def prior_pdf(B):
    return 1
```

```
def log_posteriori_fn(
    x,
    source_pdf_fn,
    prior_pdf_fn,
    B
):
    NOBS=x.shape[-1]

    # Cálculo de posteriori para registros
    posteriori = NOBS*np.log(np.abs(np.linalg.det(B)))
    y=B@x
    for i, j in np.ndindex(x.shape):
        posteriori += np.log(source_pdf(y[i,j]))
    posteriori += np.log(prior_pdf(B))

    return posteriori

def proposal_fn(
    exploration_var,
    B
):
    # Calculate shift in parameter
    shift = np.random.normal(
        loc=0,
        scale=np.sqrt(exploration_var),
        size=B.shape
    )

    # Sum shift to obtain new parameter value
    new_B = np.add(
        B, shift
    )

    return new_B
```

[55]:
```
%%time

# Create folder for combination of model specification + prior
combination_dir = model_dir / 'likelihood'
if CREATE_FOLDER_STRUCTURE:
    if not combination_dir.is_dir():
        combination_dir.mkdir()

if EXECUTE_SAMPLING:
    # Fixate arguments in log posteriori fn
    wrapper_posteriori_fn = functools.partial(
```

```
        log_posteriori_fn,
        x,
        source_pdf,
        prior_pdf
    )

    # Fixate arguments in proposal_fn
    wrapper_proposal_fn = functools.partial(
        proposal_fn,
        EXPLORATION_VAR
    )

    # Initialize MH estimator
    estimator = MMSEMetropolisHastingsEstimator(
        n_samples=N_SAMPLES,
        log_posterior_fn=wrapper_posteriori_fn,
        Q=wrapper_proposal_fn,
        burn_in=BURN_IN
    )

    # Execute MCMC estimation
    estimator.fit(
        s,
        x,
        initial_condition=initial_B,
        n_jobs=N_WORKERS
    )

    # Save artifact
    with (combination_dir / 'MH_Estimator.pkl').open('wb') as f:
        pickle.dump(estimator, f)
else:
    # Read artifact
    with (combination_dir / 'MH_Estimator.pkl').open('rb') as f:
        estimator=pickle.load(f)
```

```
CPU times: user 44.4 ms, sys: 40 ms, total: 84.3 ms
Wall time: 86.7 ms
```

9.3.2 2.3.2. Parse MCMC Results

[56]:
```
# Get results for analyzed model (best model by default)
# analyzed_model_idx = estimator.B_est_idx
analyzed_model_idx = 0
B_est = estimator.mcmc_results[analyzed_model_idx]['B_est']
s_est = B_est@x
```

```
samples = estimator.mcmc_results[analyzed_model_idx]['samples']
valid_samples = estimator.mcmc_results[analyzed_model_idx]['valid_samples']
logs = estimator.mcmc_results[analyzed_model_idx]['logs']
```

[57]:
```
print('-'*100)
print('Estimated B:\n{}'.format(B_est))
print('True B:\n{}'.format(np.linalg.inv(A)))
print('-'*100)
```

```
----------------------------------------------------------------------------------------------------
Estimated B:
[[ 0.45254658 -0.9167764 ]
 [ 0.45459147  0.91221383]]
True B:
[[ 0.5 -1. ]
 [ 0.5  1. ]]
----------------------------------------------------------------------------------------------------
```

[58]:
```
# Posteriors
[r['max_posterior'] for r in estimator.mcmc_results]
```

[58]:
```
[-83724.47309094525,
 -83724.49383450838,
 -83724.47356231448,
 -83724.5171294933,
 -83724.48083518964,
 -83724.49294898611,
 -83724.5266463708,
 -83724.5029706083,
 -83724.49241285544,
 -83724.48310084369,
 -83724.52062803158,
 -83724.52715191284,
 -83724.5406187758,
 -83724.49555465161,
 -83724.4891727077,
 -83724.50090889196,
 -83724.49640933186,
 -83724.50842165216,
 -83724.50757620552,
 -83724.46459461523,
 -83724.4888596844,
 -83724.48659308869,
 -83724.49622701957,
 -83724.49272578672,
```

```
-83724.49360194187,
-83724.4702482457,
-83724.51283261436,
-83724.47300453267,
-83724.4994965744,
-83724.47047235841]
```

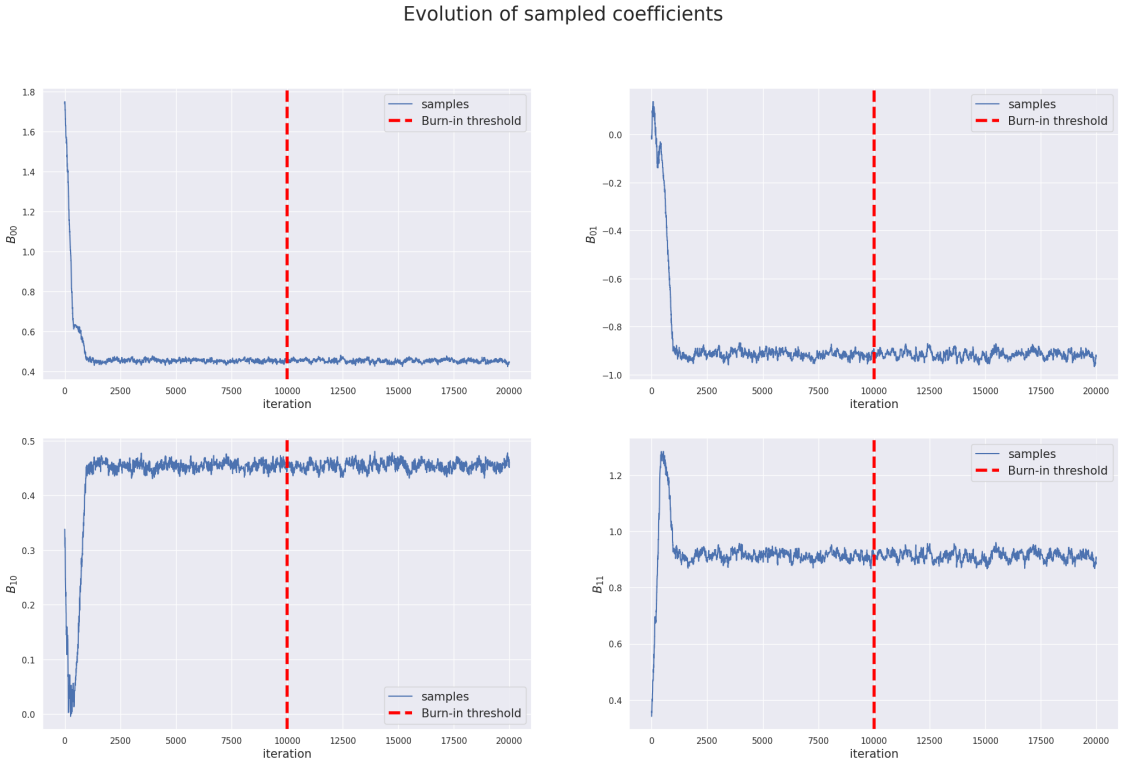
[59]:
```
[r['B_est'] for r in estimator.mcmc_results]
```

[59]:
```
[array([[ 0.45254658, -0.9167764 ],
        [ 0.45459147,  0.91221383]]),
 array([[ 0.45446249,  0.91170711],
        [-0.45264055,  0.91634685]]),
 array([[-0.45309575,  0.91575595],
        [-0.45434403, -0.91277066]]),
 array([[-0.45376731,  0.91444911],
        [-0.45357011, -0.9136017 ]]),
 array([[-0.45249695, -0.91588982],
        [ 0.45480835, -0.91214669]]),
 array([[-0.45396171, -0.91273594],
        [-0.45297678,  0.91473293]]),
 array([[ 0.45304219,  0.91449332],
        [-0.45396665,  0.91304501]]),
 array([[-0.45324008, -0.91421242],
        [ 0.45387766, -0.91413474]]),
 array([[-0.45487804,  0.91146696],
        [-0.45207145, -0.91592469]]),
 array([[ 0.45185283, -0.91776451],
        [ 0.4553749 ,  0.91006635]]),
 array([[ 0.45126535, -0.91930211],
        [ 0.45592404,  0.90837363]]),
 array([[-0.45220217, -0.91709123],
        [ 0.45505499, -0.91103093]]),
 array([[-0.45214195,  0.91698111],
        [ 0.45512261,  0.91037411]]),
 array([[-0.45246692,  0.91709097],
        [ 0.45490309,  0.91155936]]),
 array([[-0.45408482,  0.91353725],
        [-0.45310601, -0.91454235]]),
 array([[ 0.45406704,  0.91289324],
        [-0.45281403,  0.91539859]]),
 array([[ 0.45411829, -0.91321178],
        [-0.45316435, -0.9149923 ]]),
 array([[-0.45440737,  0.91327728],
        [ 0.45297236,  0.91520781]]),
 array([[-0.45407918,  0.9141409 ],
        [ 0.45321791,  0.91410028]]),
```

```
array([[-0.45471639, -0.91168967],
       [-0.45244655,  0.91667769]]),
array([[-0.45426382,  0.91374965],
       [ 0.45325366,  0.91463932]]),
array([[-0.45502432,  0.91175958],
       [ 0.45237612,  0.9162646 ]]),
array([[-0.45300992, -0.91553181],
       [-0.45456141,  0.91239532]]),
array([[ 0.45427145,  0.91150111],
       [ 0.45267954, -0.91608362]]),
array([[ 0.45421535,  0.91315224],
       [-0.45338402,  0.91522411]]),
array([[-0.45426322,  0.9130032 ],
       [ 0.45288842,  0.91519792]]),
array([[-0.45429975,  0.91310752],
       [ 0.45307195,  0.91486748]]),
array([[-0.45440003,  0.91254279],
       [ 0.45266221,  0.91570627]]),
array([[ 0.45326505,  0.91435057],
       [-0.45385622,  0.91322775]]),
array([[-0.45414573,  0.9129537 ],
       [-0.45299522, -0.91551713]])]
```

9.3.3 2.3.3. Plot sampled coefficients stochastic process - Markov Chain evolution

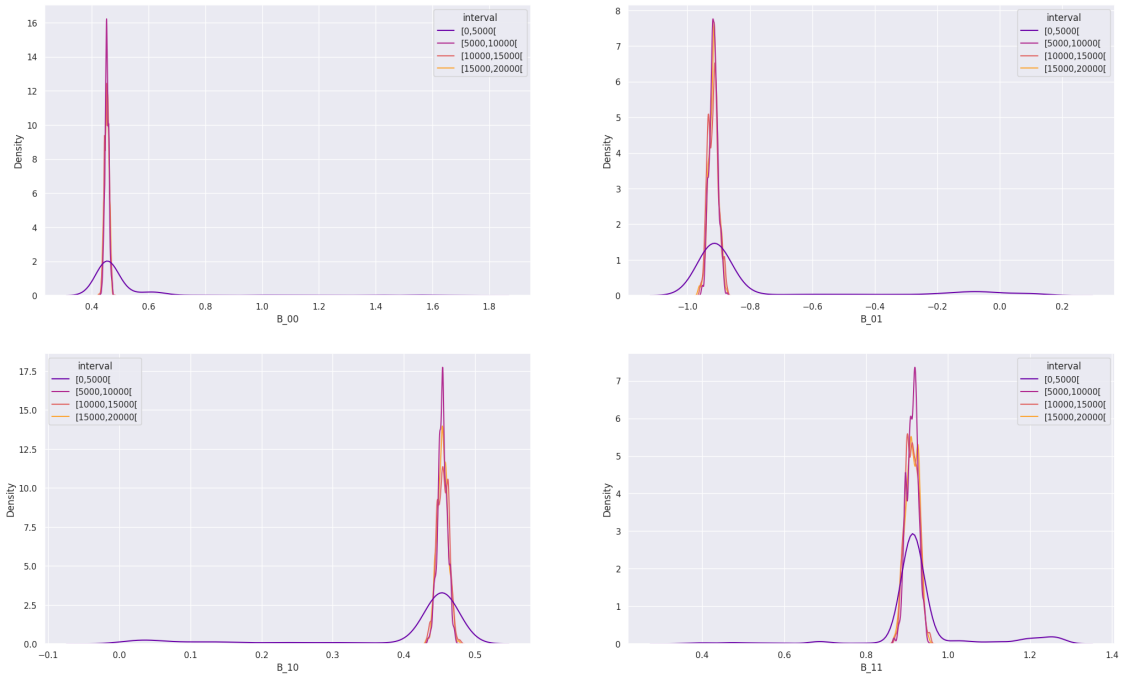
[60]:
```
MCMCGraphPlotter.evolution_of_sampled_coefficients(
    samples=samples,
    estimator=estimator,
    logs=logs,
    save_dir=combination_dir if SAVE_GRAPHS else None
)
```



9.3.4 2.3.4. Plot sampled coefficients distributions - Markov Chain evolution

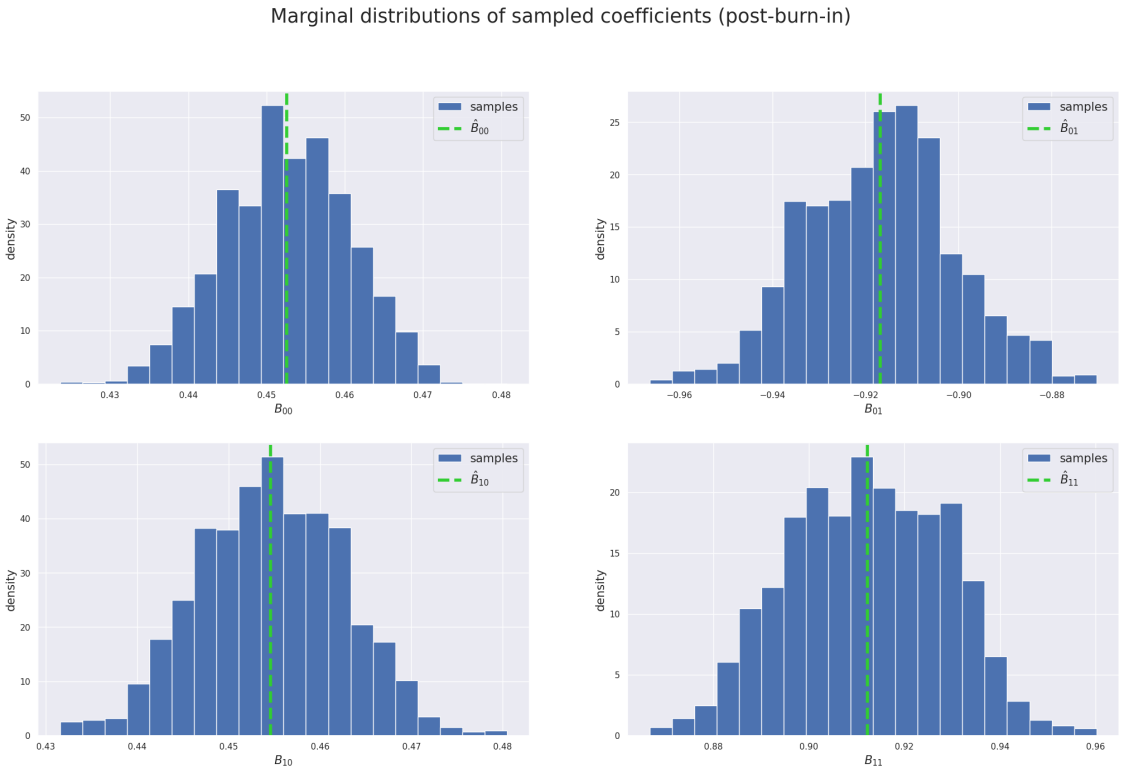
[61]:
```
STEP_SIZE=5000
PALETTE='plasma'

MCMCGraphPlotter.evolution_of_samples_distribution(
    B_est=B_est,
    samples=samples,
    step_size=STEP_SIZE,
    palette=PALETTE,
    save_dir=combination_dir if SAVE_GRAPHS else None
)
```

Evolution of coefficient distributions



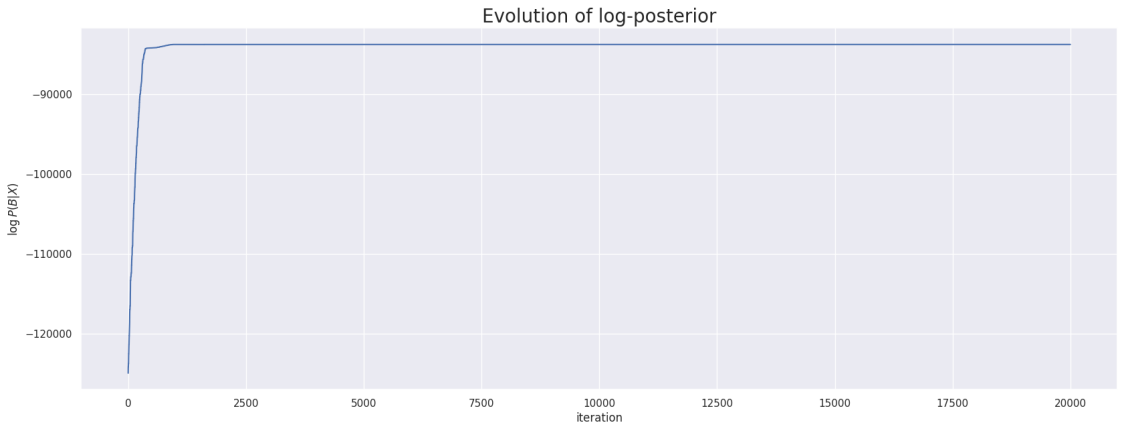
9.3.5 2.3.5. Plot Marginal Posteriors for Coefficients - Post-burn-in

[62]:
```
MCMCGraphPlotter.steady_state_marginal_distributions(
    valid_samples=valid_samples,
    nbins=NBINS,
    B_est=B_est,
    save_dir=combination_dir if SAVE_GRAPHS else None
)
```



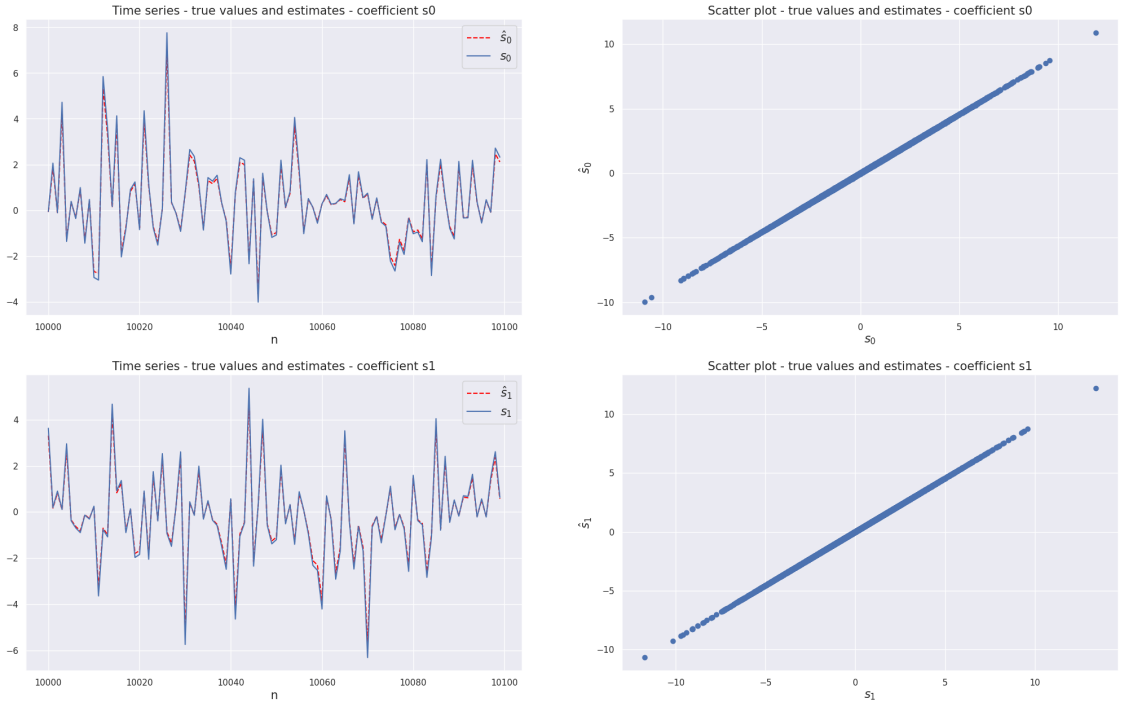
9.3.6 2.3.6. Plot evolution of log-posterior

[63]:
```
MCMCGraphPlotter.evolution_log_posterior(
    logs=logs,
    save_dir=combination_dir if SAVE_GRAPHS else None
)
```



9.4 2.3.7. Plot source separation results

[64]:
```
NPOINTS=100
PLOT_START=NOBS//2
PLOT_END=PLOT_START+NPOINTS

MCMCGraphPlotter.source_separation_results(
    plot_start=PLOT_START,
    plot_end=PLOT_END,
    B_est=B_est,
    s=s,
    s_est=s_est,
    x=x,
    nobs=NOBS,
    save_dir=combination_dir if SAVE_GRAPHS else None
)
```



[65]:
```
# Error norm
err=np.linalg.norm(
    np.subtract(s,s_est)
)/np.size(s)

# Log error
errors['slightly_misspecified_model']['likelihood'] = err
```

```
print('Norma do erro de estimação: {}'.format(err))
```

```
Norma do erro de estimação: 0.0008887084177311825
```

9.5 2.4. Perform Analysis - DETERMINANT PRIOR

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

9.5.1 2.4.1 Execute MCMC Sampling

[66]:
```
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)))

def log_posteriori_fn(
    x,
    source_pdf_fn,
    prior_pdf_fn,
    B
):
    NOBS=x.shape[-1]

    # Cálculo de posteriori para registros
    posteriori = NOBS*np.log(np.abs(np.linalg.det(B)))
    y=B@x
    for i, j in np.ndindex(x.shape):
        posteriori += np.log(source_pdf(y[i,j]))
    posteriori += np.log(prior_pdf(B))

    return posteriori

def proposal_fn(
    exploration_var,
    B
):
    # Calculate shift in parameter
    shift = np.random.normal(
        loc=0,
        scale=np.sqrt(exploration_var),
        size=B.shape
    )
```

```
    # Sum shift to obtain new parameter value
    new_B = np.add(
        B, shift
    )

    return new_B
```

[67]:
```
%%time

# Create folder for combination of model specification + prior
combination_dir = model_dir / 'determinant_prior'
if CREATE_FOLDER_STRUCTURE:
    if not combination_dir.is_dir():
        combination_dir.mkdir()

if EXECUTE_SAMPLING:
    # Fixate arguments in log posteriori fn
    wrapper_posteriori_fn = functools.partial(
        log_posteriori_fn,
        x,
        source_pdf,
        prior_pdf
    )

    # Fixate arguments in proposal_fn
    wrapper_proposal_fn = functools.partial(
        proposal_fn,
        EXPLORATION_VAR
    )

    # Initialize MH estimator
    estimator = MMSEMetropolisHastingsEstimator(
        n_samples=N_SAMPLES,
        log_posterior_fn=wrapper_posteriori_fn,
        Q=wrapper_proposal_fn,
        burn_in=BURN_IN
    )

    # Execute MCMC estimation
    estimator.fit(
        s,
        x,
        initial_condition=initial_B,
        n_jobs=N_WORKERS
    )

    # Save artifact
```

```
    with (combination_dir / 'MH_Estimator.pkl').open('wb') as f:
        pickle.dump(estimator, f)
else:
    # Read artifact
    with (combination_dir / 'MH_Estimator.pkl').open('rb') as f:
        estimator=pickle.load(f)
```

```
CPU times: user 61.9 ms, sys: 16.3 ms, total: 78.1 ms
Wall time: 80 ms
```

9.5.2 2.4.2. Parse MCMC Results

[68]:
```
# Get results for analyzed model (best model by default)
# analyzed_model_idx = estimator.B_est_idx
analyzed_model_idx = 0
B_est = estimator.mcmc_results[analyzed_model_idx]['B_est']
s_est = B_est@x
samples = estimator.mcmc_results[analyzed_model_idx]['samples']
valid_samples = estimator.mcmc_results[analyzed_model_idx]['valid_samples']
logs = estimator.mcmc_results[analyzed_model_idx]['logs']
```

[69]:
```
print('-'*100)
print('Estimated B:\n{}'.format(B_est))
print('True B:\n{}'.format(np.linalg.inv(A)))
print('-'*100)
```

```
----------------------------------------------------------------------------------------
--------------------
Estimated B:
[[ 0.45212338 -0.91850741]
 [ 0.45543706  0.91031807]]
True B:
[[ 0.5 -1. ]
 [ 0.5  1. ]]
----------------------------------------------------------------------------------------
--------------------
```

[70]:
```
# Posteriors
[r['max_posterior'] for r in estimator.mcmc_results]
```

[70]:
```
[-83724.51526568584,
 -83724.52735361346,
 -83724.51193879756,
 -83724.49355862965,
 -83724.51313093999,
 -83724.5030019633,
```

```
-83724.4834209965,
-83724.4759776923,
-83724.49709917007,
-83724.48512781507,
-83724.49459418523,
-83724.51527616222,
-83724.48938862114,
-83724.48332132805,
-83724.48368205412,
-83724.4673195939,
-83724.48844168875,
-83724.47575044444,
-83724.50838335854,
-83724.52048779959,
-83724.52554537512,
-83724.51441812892,
-83724.4907183338,
-83724.49524884674,
-83724.4690512942,
-83724.4947220375,
-83724.49181050074,
-83724.50921095397,
-83724.47777849137,
-83724.49580358909]
```

[71]:
```
[r['B_est'] for r in estimator.mcmc_results]
```

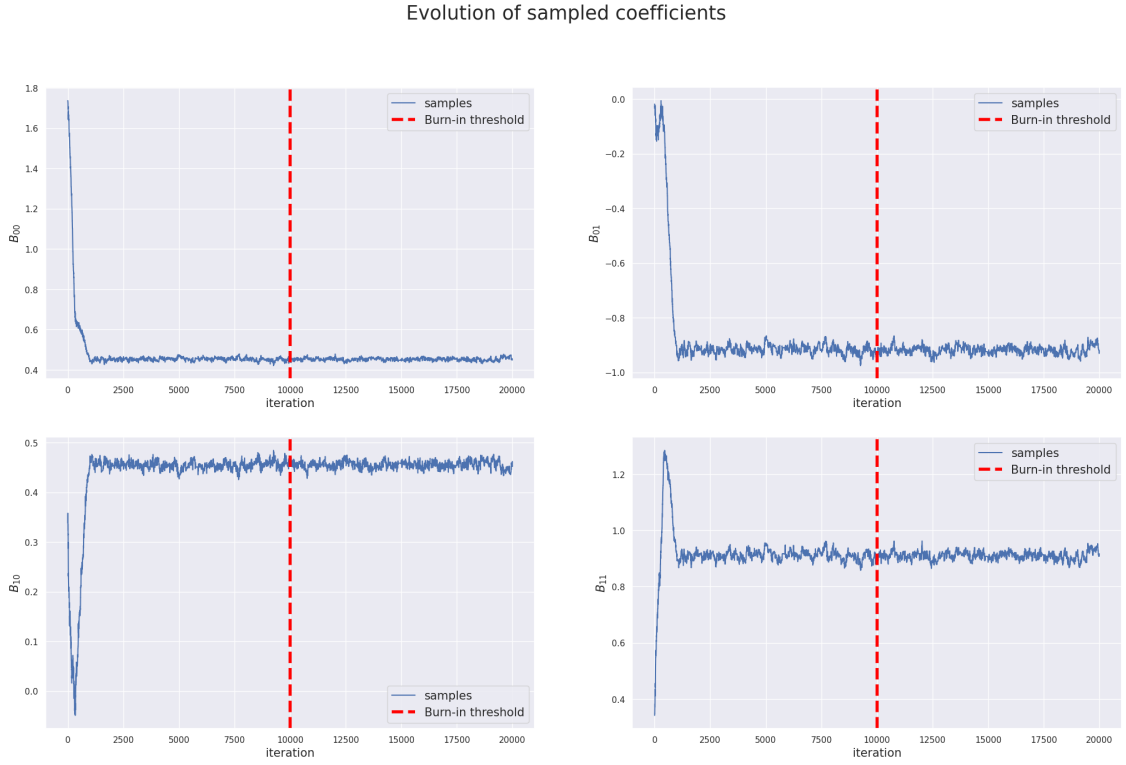
[71]:
```
[array([[ 0.45212338, -0.91850741],
        [ 0.45543706,  0.91031807]]),
 array([[ 0.45276916,  0.91429334],
        [-0.45433239,  0.91264819]]),
 array([[-0.45432665,  0.91314036],
        [-0.45269725, -0.91518129]]),
 array([[-0.45459201,  0.91260201],
        [-0.45266582, -0.91524748]]),
 array([[-0.45394623, -0.91334678],
        [ 0.45335754, -0.91489962]]),
 array([[-0.45309356, -0.9150313 ],
        [-0.45411812,  0.91306767]]),
 array([[ 0.45329474,  0.91391983],
        [-0.45383075,  0.91404334]]),
 array([[-0.45417037, -0.91343647],
        [ 0.45330497, -0.91538754]]),
 array([[-0.45452354,  0.9130346 ],
        [-0.45280786, -0.91596294]]),
 array([[ 0.45224516, -0.91703055],
        [ 0.45502758,  0.91137764]]),
```

```
array([[ 0.45289953, -0.91534268],
       [ 0.4542068 ,  0.91129824]]),
array([[-0.45291687, -0.91486491],
       [ 0.45414444, -0.91324175]]),
array([[-0.45309892,  0.91541933],
       [ 0.45396738,  0.91228535]]),
array([[-0.45365978,  0.91431069],
       [ 0.45351349,  0.91369821]]),
array([[-0.45449039, -0.91201796],
       [ 0.45274861, -0.91613102]]),
array([[ 0.45297644,  0.91527933],
       [-0.45428821,  0.91301867]]),
array([[ 0.45244589, -0.91678914],
       [-0.45485643, -0.91068216]]),
array([[-0.45259405,  0.91609509],
       [ 0.4544476 ,  0.91142452]]),
array([[-0.45193677,  0.91789062],
       [ 0.45521917,  0.90973177]]),
array([[-0.45403135, -0.91289816],
       [-0.45304888,  0.91499946]]),
array([[ 0.45430104,  0.91124893],
       [ 0.45273245, -0.91718589]]),
array([[-0.45360131,  0.91453497],
       [ 0.45371622,  0.91333186]]),
array([[-0.45343711,  0.91474889],
       [ 0.45387591,  0.91278203]]),
array([[ 0.45353361,  0.91305315],
       [ 0.45364299, -0.9144968 ]]),
array([[ 0.45216236,  0.91655311],
       [-0.45480245,  0.91144973]]),
array([[-0.45298424,  0.9155213 ],
       [ 0.45420097,  0.91209349]]),
array([[-0.45327722,  0.91502362],
       [ 0.45384526,  0.9129387 ]]),
array([[-0.45203712,  0.91807564],
       [ 0.45509793,  0.90961609]]),
array([[ 0.4543156 ,  0.91238618],
       [-0.45269953,  0.91577005]]),
array([[-0.45546543,  0.91087038],
       [-0.45181488, -0.9175368 ]])]
```

9.5.3 2.4.3. Plot sampled coefficients stochastic process - Markov Chain evolution

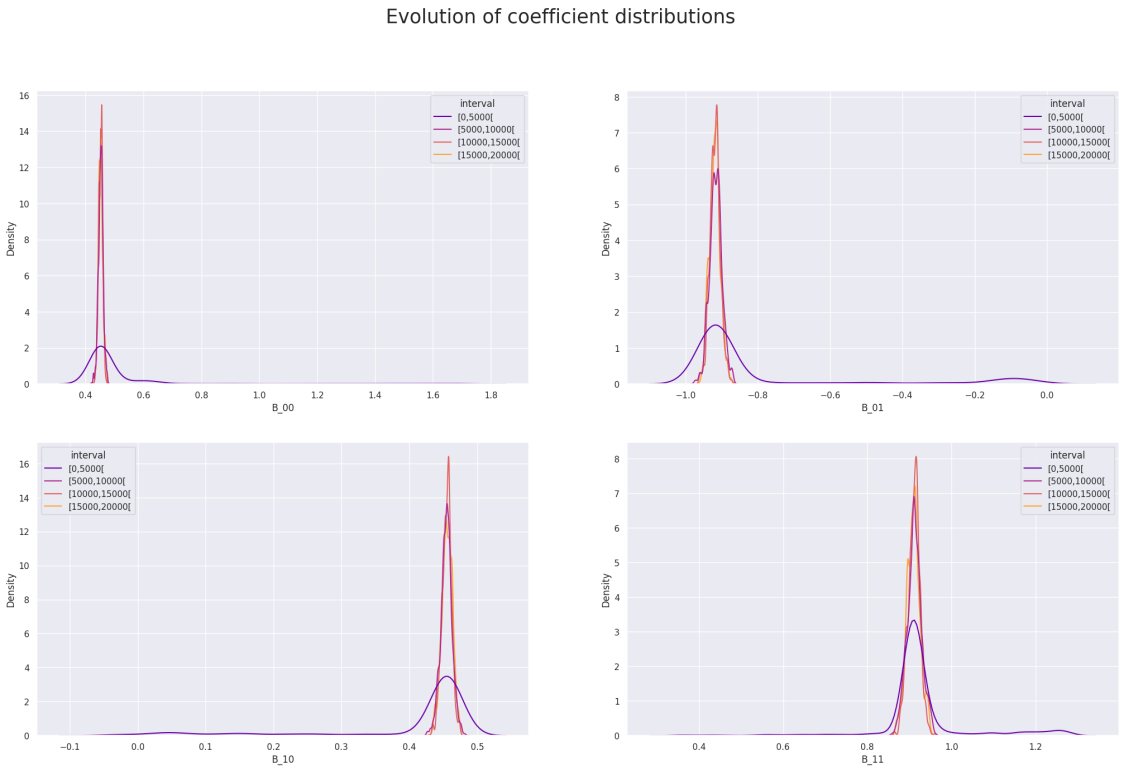
[72]:
```
MCMCGraphPlotter.evolution_of_sampled_coefficients(
    samples=samples,
    estimator=estimator,
    logs=logs,
```

```
    save_dir=combination_dir if SAVE_GRAPHS else None
)
```



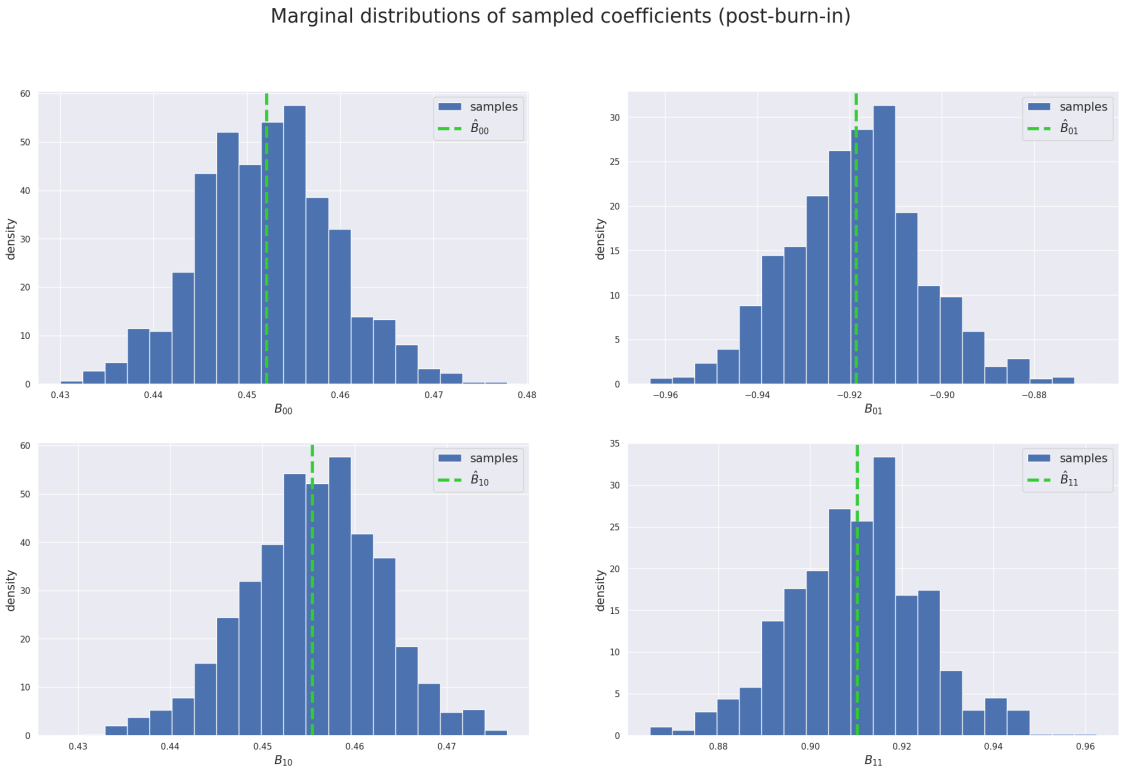
9.5.4 2.4.4. Plot sampled coefficients distributions - Markov Chain evolution

[73]:
```
STEP_SIZE=5000
PALETTE='plasma'

MCMCGraphPlotter.evolution_of_samples_distribution(
    B_est=B_est,
    samples=samples,
    step_size=STEP_SIZE,
    palette=PALETTE,
    save_dir=combination_dir if SAVE_GRAPHS else None
)
```



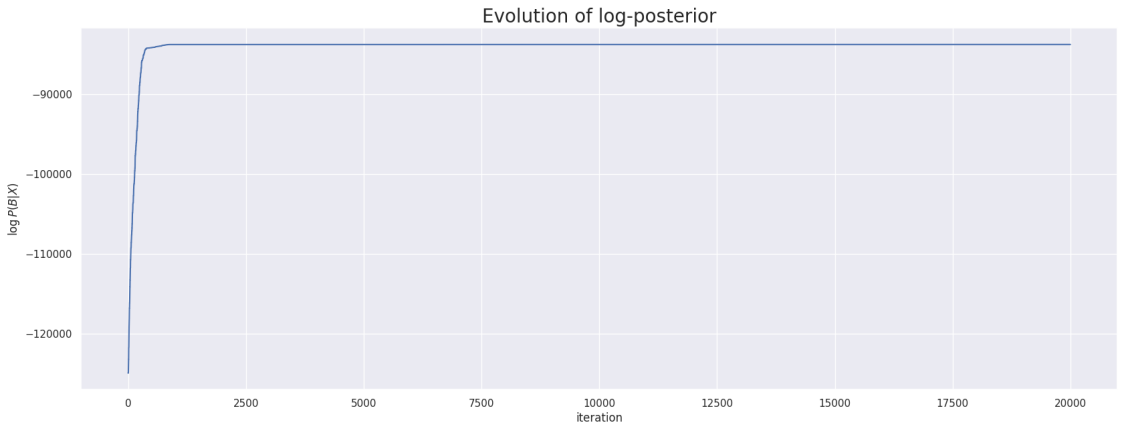
9.5.5 2.4.5. Plot Marginal Posteriors for Coefficients - Post-burn-in

[74]:
```
MCMCGraphPlotter.steady_state_marginal_distributions(
    valid_samples=valid_samples,
    nbins=NBINS,
    B_est=B_est,
    save_dir=combination_dir if SAVE_GRAPHS else None
)
```



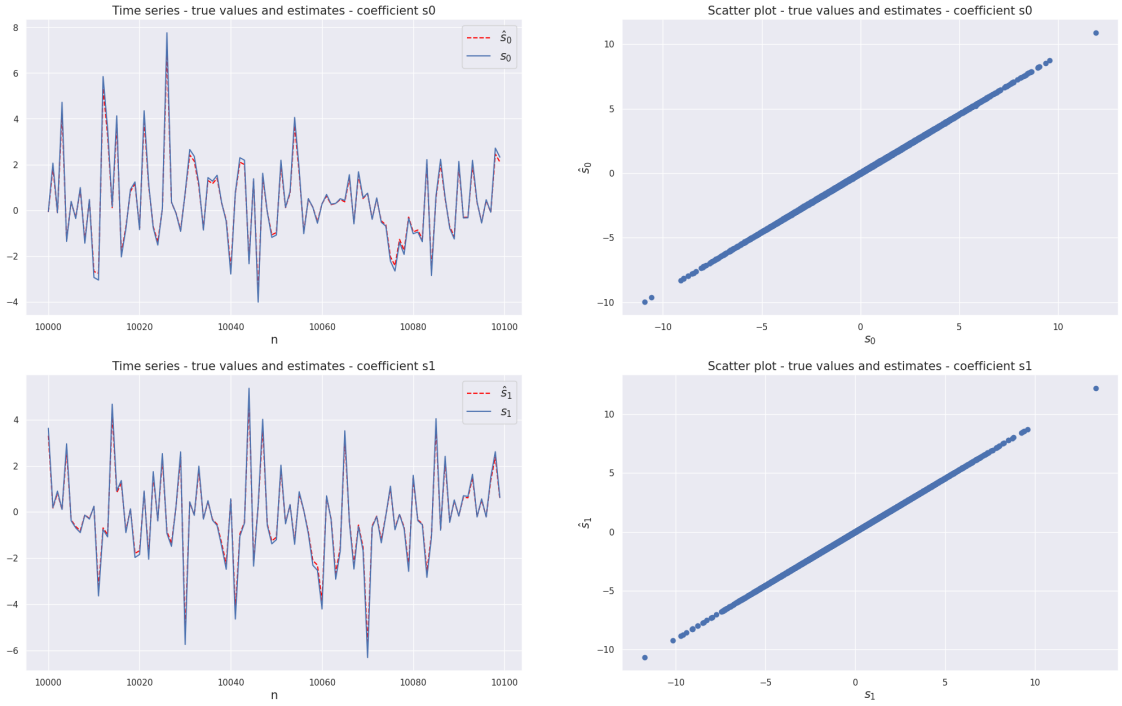
9.5.6 2.4.6. Plot evolution of log-posterior

[75]:
```
MCMCGraphPlotter.evolution_log_posterior(
    logs=logs,
    save_dir=combination_dir if SAVE_GRAPHS else None
)
```



9.6 2.4.7. Plot source separation results

[76]:
```
NPOINTS=100
PLOT_START=NOBS//2
PLOT_END=PLOT_START+NPOINTS

MCMCGraphPlotter.source_separation_results(
    plot_start=PLOT_START,
    plot_end=PLOT_END,
    B_est=B_est,
    s=s,
    s_est=s_est,
    x=x,
    nobs=NOBS,
    save_dir=combination_dir if SAVE_GRAPHS else None
)
```



[77]:
```
# Error norm
err=np.linalg.norm(
    np.subtract(s,s_est)
)/np.size(s)

# Log error
errors['slightly_misspecified_model']['determinant_prior'] = err
```

```
print('Norma do erro de estimação: {}'.format(err))
```

```
Norma do erro de estimação: 0.00088742115196529
```

9.7 2.5. Perform Analysis - near-identity transformation

Prior:

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

9.7.1 2.5.1 Execute MCMC Sampling

[78]:
```
def source_pdf(x):
    return np.exp(-x)/np.square(1+np.exp(-x))

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 log_posteriori_fn(
    x,
    source_pdf_fn,
    prior_pdf_fn,
    B
):
    NOBS=x.shape[-1]

    # Cálculo de posteriori para registros
    posteriori = NOBS*np.log(np.abs(np.linalg.det(B)))
    y=B@x
    for i, j in np.ndindex(x.shape):
        posteriori += np.log(source_pdf(y[i,j]))
    posteriori += np.log(prior_pdf(B))

    return posteriori

def proposal_fn(
    exploration_var,
    B
):
    # Calculate shift in parameter
    shift = np.random.normal(
        loc=0,
        scale=np.sqrt(exploration_var),
        size=B.shape
```

```
    )

    # Sum shift to obtain new parameter value
    new_B = np.add(
        B, shift
    )

    return new_B
```

[79]:
```
%%time

# Create folder for combination of model specification + prior
combination_dir = model_dir / 'identity_transform'
if CREATE_FOLDER_STRUCTURE:
    if not combination_dir.is_dir():
        combination_dir.mkdir()

if EXECUTE_SAMPLING:
    # Fixate arguments in log posteriori fn
    wrapper_posteriori_fn = functools.partial(
        log_posteriori_fn,
        x,
        source_pdf,
        prior_pdf
    )

    # Fixate arguments in proposal_fn
    wrapper_proposal_fn = functools.partial(
        proposal_fn,
        EXPLORATION_VAR
    )

    # Initialize MH estimator
    estimator = MMSEMetropolisHastingsEstimator(
        n_samples=N_SAMPLES,
        log_posterior_fn=wrapper_posteriori_fn,
        Q=wrapper_proposal_fn,
        burn_in=BURN_IN
    )

    # Execute MCMC estimation
    estimator.fit(
        s,
        x,
        initial_condition=initial_B,
        n_jobs=N_WORKERS
    )
```

```
    # Save artifact
    with (combination_dir / 'MH_Estimator.pkl').open('wb') as f:
        pickle.dump(estimator, f)
else:
    # Read artifact
    with (combination_dir / 'MH_Estimator.pkl').open('rb') as f:
        estimator=pickle.load(f)
```

```
CPU times: user 42.3 ms, sys: 47.9 ms, total: 90.2 ms
Wall time: 95.1 ms
```

9.7.2 2.5.2. Parse MCMC Results

[80]:
```
# Get results for analyzed model (best model by default)
# analyzed_model_idx = estimator.B_est_idx
analyzed_model_idx = 0
B_est = estimator.mcmc_results[analyzed_model_idx]['B_est']
s_est = B_est@x
samples = estimator.mcmc_results[analyzed_model_idx]['samples']
valid_samples = estimator.mcmc_results[analyzed_model_idx]['valid_samples']
logs = estimator.mcmc_results[analyzed_model_idx]['logs']
```

[81]:
```
print('-'*100)
print('Estimated B:\n{}'.format(B_est))
print('True B:\n{}'.format(np.linalg.inv(A)))
print('-'*100)
```

```
----------------------------------------------------------------------------------------
--------------------
Estimated B:
[[ 0.45208099 -0.9169157 ]
 [ 0.45486892  0.91034756]]
True B:
[[ 0.5 -1. ]
 [ 0.5  1. ]]
----------------------------------------------------------------------------------------
--------------------
```

[82]:
```
# Posteriors
[r['max_posterior'] for r in estimator.mcmc_results]
```

[82]:
```
[-83724.51258316879,
 -83724.46768447662,
 -83724.50134385555,
 -83724.55710950652,
```

```
-83724.4963298515,
-83724.47057095333,
-83724.48546185096,
-83724.50979129308,
-83724.51833034439,
-83724.47197490079,
-83724.5095143327,
-83724.4770484742,
-83724.50498011183,
-83724.49874900182,
-83724.51726255147,
-83724.48185750854,
-83724.50558677419,
-83724.49930510536,
-83724.52825512033,
-83724.50259510633,
-83724.52554537512,
-83724.46991883867,
-83724.47533384728,
-83724.53537328474,
-83724.51089899358,
-83724.47194911318,
-83724.49181050074,
-83724.50025797174,
-83724.51896448735,
-83724.51597292005]
```

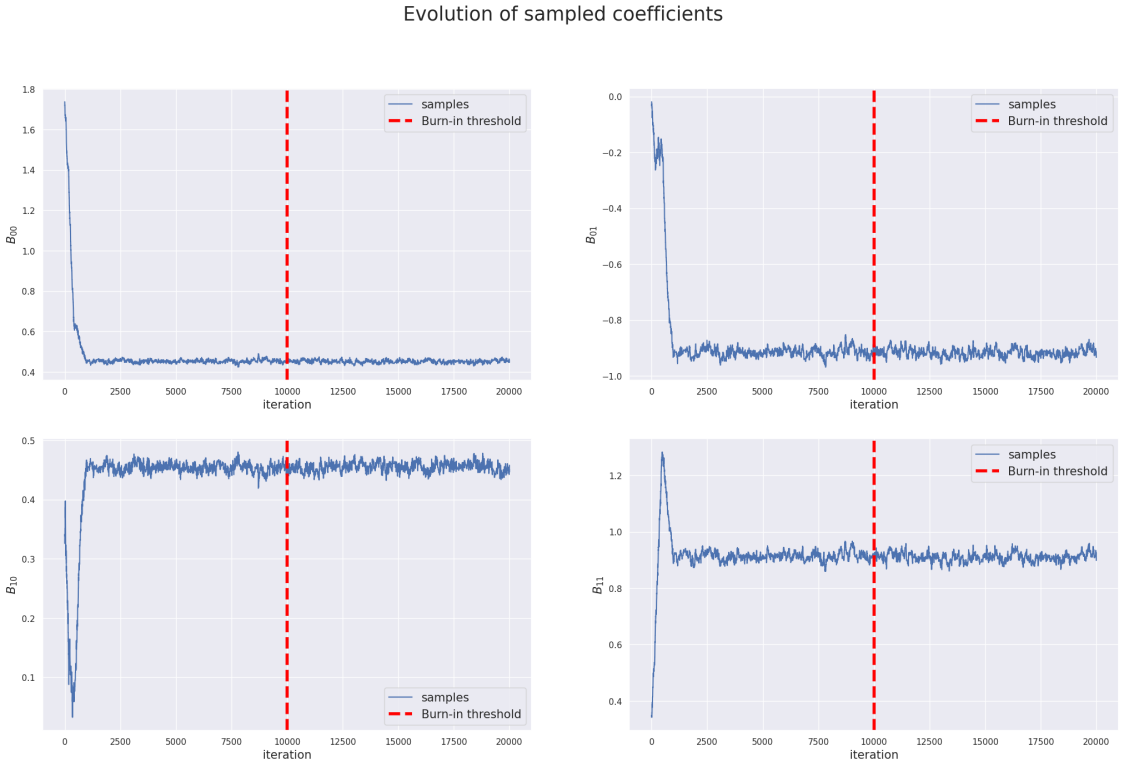
[83]:
```
[r['B_est'] for r in estimator.mcmc_results]
```

[83]:
```
[array([[ 0.45208099, -0.9169157 ],
        [ 0.45486892,  0.91034756]]),
 array([[ 0.45289387, -0.91542144],
        [ 0.45441262,  0.91234286]]),
 array([[-0.45349743,  0.91464595],
        [-0.45357053, -0.9133748 ]]),
 array([[-0.45284098,  0.9163315 ],
        [-0.45428754, -0.9117788 ]]),
 array([[-0.45307472, -0.91419235],
        [ 0.45384976, -0.91352122]]),
 array([[-0.45437665, -0.91228284],
        [-0.45276503,  0.91615079]]),
 array([[ 0.45278058,  0.91526068],
        [-0.45440442,  0.91302663]]),
 array([[-0.45441488, -0.91222069],
        [ 0.45301276, -0.91517073]]),
 array([[-0.4526366 ,  0.91611587],
        [-0.45450068, -0.91120627]]),
```

```
array([[-0.45350574,  0.91546656],
       [ 0.45373466,  0.91362192]]),
array([[ 0.45285846, -0.91571552],
       [ 0.45421305,  0.91192586]]),
array([[-0.45546301, -0.90971528],
       [ 0.45207168, -0.91876898]]),
array([[-0.45267975,  0.91621204],
       [ 0.45457278,  0.91149965]]),
array([[-0.45365041,  0.91421001],
       [ 0.45363949,  0.91317008]]),
array([[-0.45423158,  0.91317476],
       [-0.45310494, -0.91500701]]),
array([[ 0.45376282,  0.91285771],
       [-0.45339064,  0.91468109]]),
array([[ 0.45286656, -0.91595645],
       [-0.45422509, -0.91322982]]),
array([[-0.45244648,  0.91726778],
       [ 0.45493676,  0.91024197]]),
array([[-0.45399099,  0.91401041],
       [ 0.45313127,  0.91504987]]),
array([[-0.45316558, -0.91413534],
       [-0.45414607,  0.91382694]]),
array([[ 0.45430104,  0.91124893],
       [ 0.45273245, -0.91718589]]),
array([[-0.45451987,  0.91242585],
       [ 0.45262939,  0.91563719]]),
array([[-0.45357664,  0.91507464],
       [ 0.4535839 ,  0.91323205]]),
array([[ 0.45480052,  0.91082512],
       [ 0.45230848, -0.9175291 ]]),
array([[ 0.45343056,  0.91470343],
       [-0.45380653,  0.91380139]]),
array([[-0.45250633,  0.91687167],
       [ 0.45472053,  0.91099609]]),
array([[-0.45327722,  0.91502362],
       [ 0.45384526,  0.9129387 ]]),
array([[-0.45380396,  0.9138094 ],
       [ 0.45353168,  0.91381272]]),
array([[ 0.45382954,  0.91348217],
       [-0.45343284,  0.91492639]]),
array([[-0.45388256,  0.91356253],
       [-0.45324976, -0.91409261]])]
```

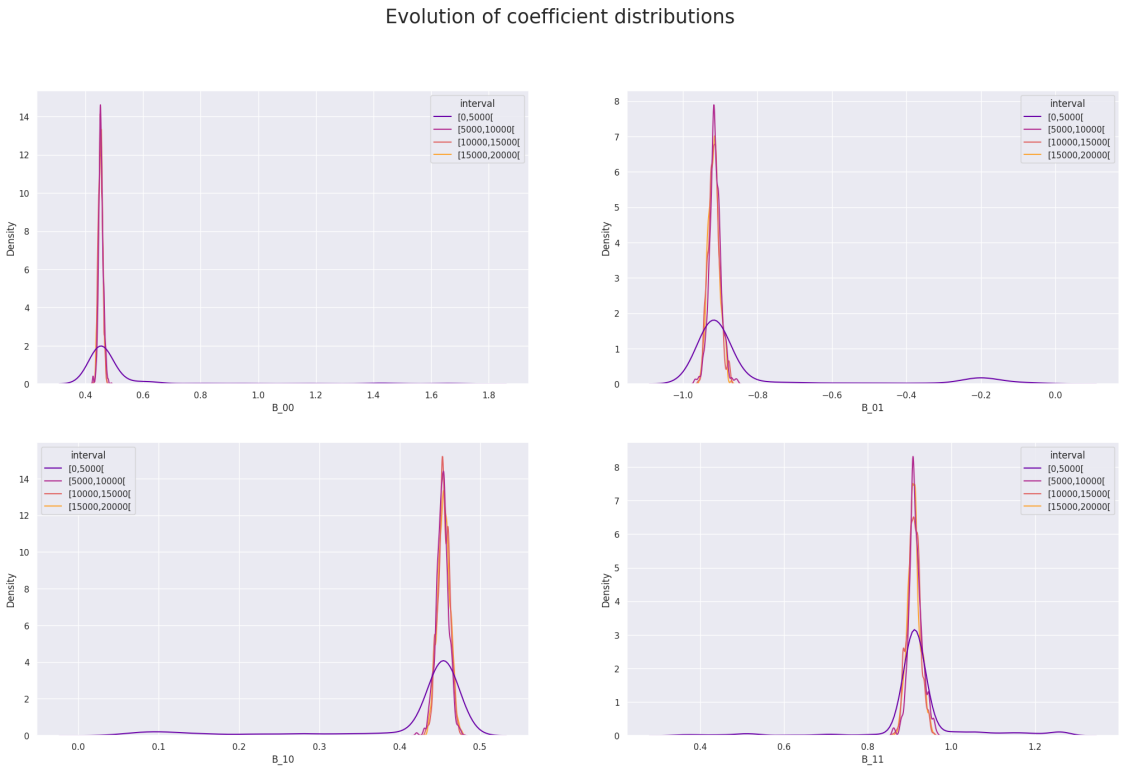
9.7.3 2.5.3. Plot sampled coefficients stochastic process - Markov Chain evolution

[84]:
```
MCMCGraphPlotter.evolution_of_sampled_coefficients(
    samples=samples,
    estimator=estimator,
    logs=logs,
    save_dir=combination_dir if SAVE_GRAPHS else None
)
```



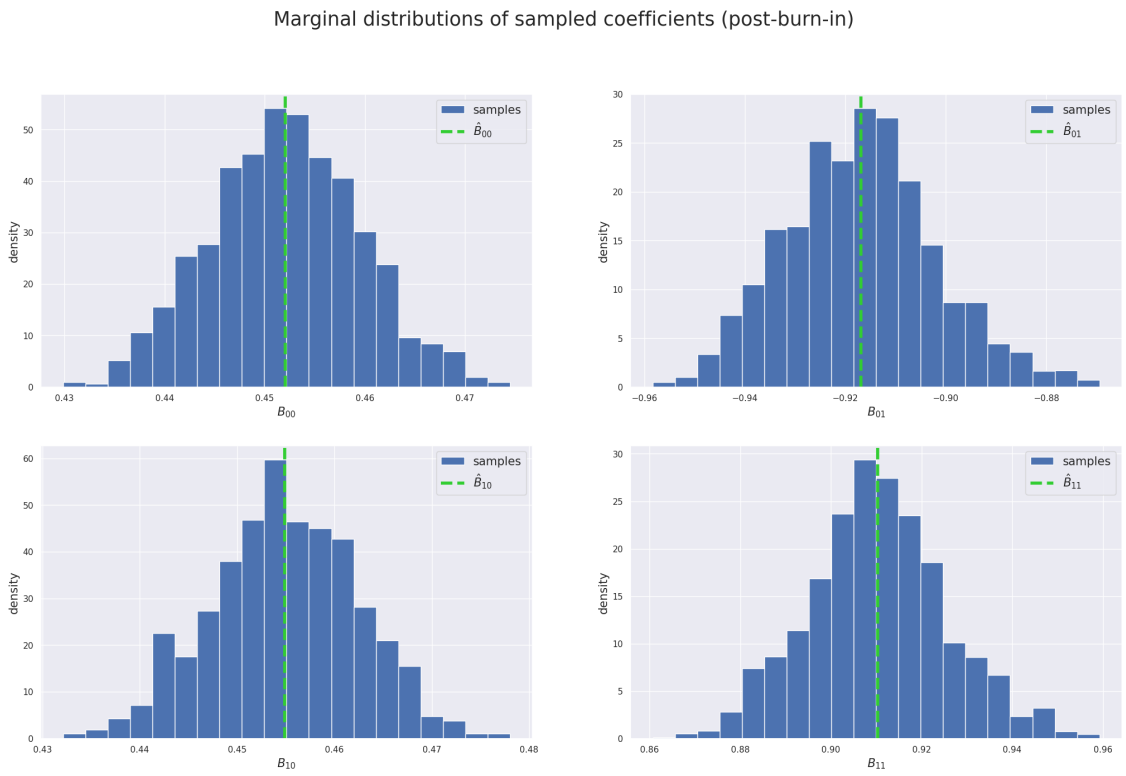
9.7.4 2.5.4. Plot sampled coefficients distributions - Markov Chain evolution

[85]:
```
STEP_SIZE=5000
PALETTE='plasma'

MCMCGraphPlotter.evolution_of_samples_distribution(
    B_est=B_est,
    samples=samples,
    step_size=STEP_SIZE,
    palette=PALETTE,
    save_dir=combination_dir if SAVE_GRAPHS else None
)
```



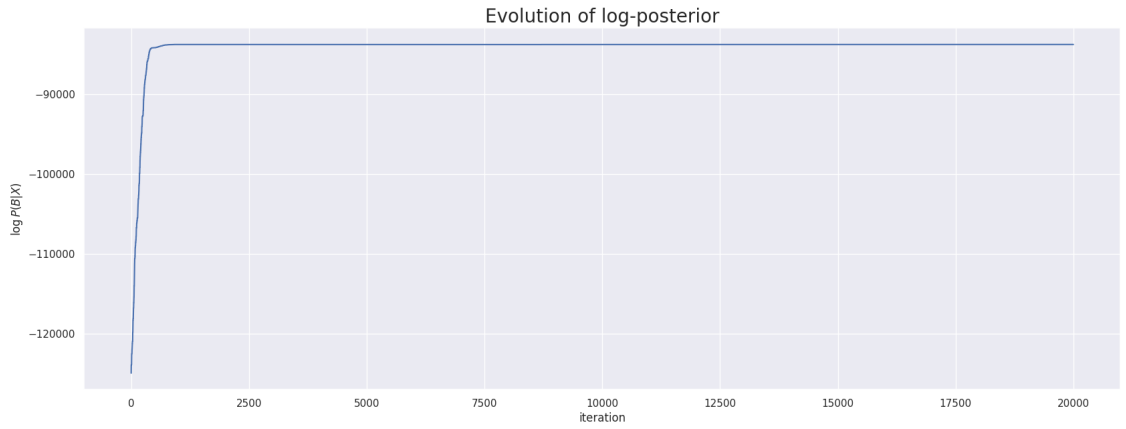
9.7.5 2.5.5. Plot Marginal Posteriors for Coefficients - Post-burn-in

[86]:
```
MCMCGraphPlotter.steady_state_marginal_distributions(
    valid_samples=valid_samples,
    nbins=NBINS,
    B_est=B_est,
    save_dir=combination_dir if SAVE_GRAPHS else None
)
```



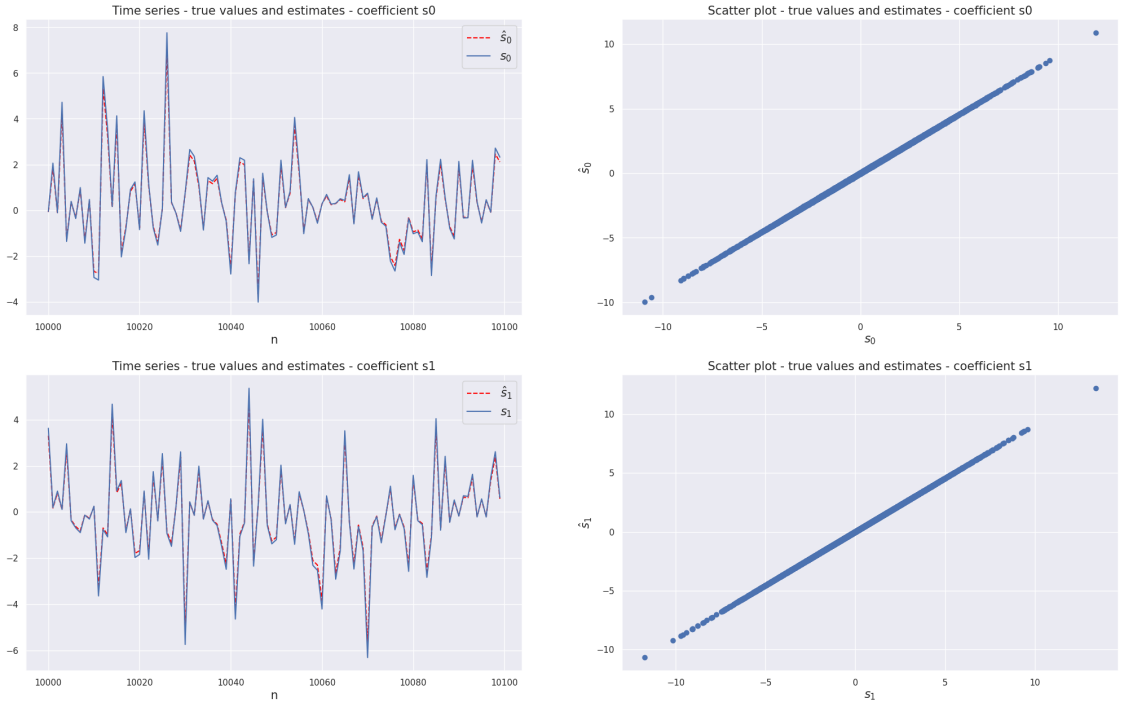
9.7.6 2.5.6. Plot evolution of log-posterior

[87]:
```
MCMCGraphPlotter.evolution_log_posterior(
    logs=logs,
    save_dir=combination_dir if SAVE_GRAPHS else None
)
```



9.8 2.5.7. Plot source separation results

[88]:
```
NPOINTS=100
PLOT_START=NOBS//2
PLOT_END=PLOT_START+NPOINTS

MCMCGraphPlotter.source_separation_results(
    plot_start=PLOT_START,
    plot_end=PLOT_END,
    B_est=B_est,
    s=s,
    s_est=s_est,
    x=x,
    nobs=NOBS,
    save_dir=combination_dir if SAVE_GRAPHS else None
)
```



[89]:
```
# Error norm
err=np.linalg.norm(
    np.subtract(s,s_est)
)/np.size(s)

# Log error
errors['slightly_misspecified_model']['near_identity_prior'] = err
```

```
print('Norma do erro de estimação: {}'.format(err))
```

```
Norma do erro de estimação: 0.0008940354077953767
```

[90]:
```
# Save errors, if new execution was made
if EXECUTE_SAMPLING:
    with (experiment_dir / 'errors.pkl').open('wb') as f:
        pickle.dump(errors, f)
errors
```

[90]:
```
{'perfect_model': {'likelihood': 6.443230754863028e-05,
  'determinant_prior': 6.319216406611947e-05,
  'near_identity_prior': 0.012761931151685916},
 'slightly_misspecified_model': {'likelihood': 0.0008887084177311825,
  'determinant_prior': 0.00088742115196529,
  'near_identity_prior': 0.0008940354077953767}}
```

[ ]:
```

```