1-MMSE_Metropolis_Hastings

July 16, 2024

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

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

np.random.seed(2000)
```

[2]:
```
# Whether or not to execute sampling
EXECUTE_SAMPLING=False

# Number of sources and observations
NSOURCES=2
NOBS=10000

# 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=10000
BURN_IN=0.5

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

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):

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

7 1. PERFECT MODEL

7.1 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)
    )
```

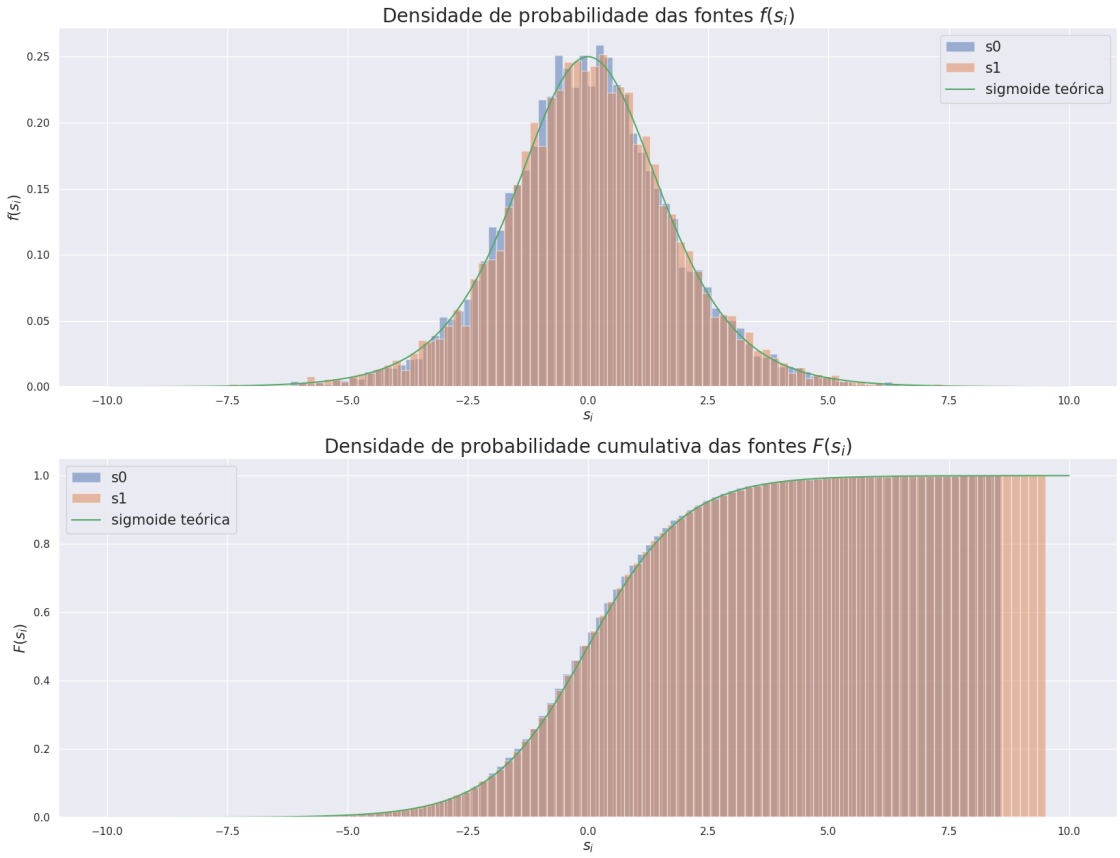
[4]:
```
MU=0.0

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: 1000
CURTOSE s0: 1.2000000000000002
CURTOSE s1: 1.2000000000000002
----------------------------------------------------------------------------------------------------
```

```
NÚMERO DE FONTES: 2
TAMANHO DOS SINAIS DAS FONTES: 10000
CURTOSE s0: 0.924747949970139
CURTOSE s1: 1.1835518264795413
--------------------------------------------------------------------------------
--------------------
```

[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)
```



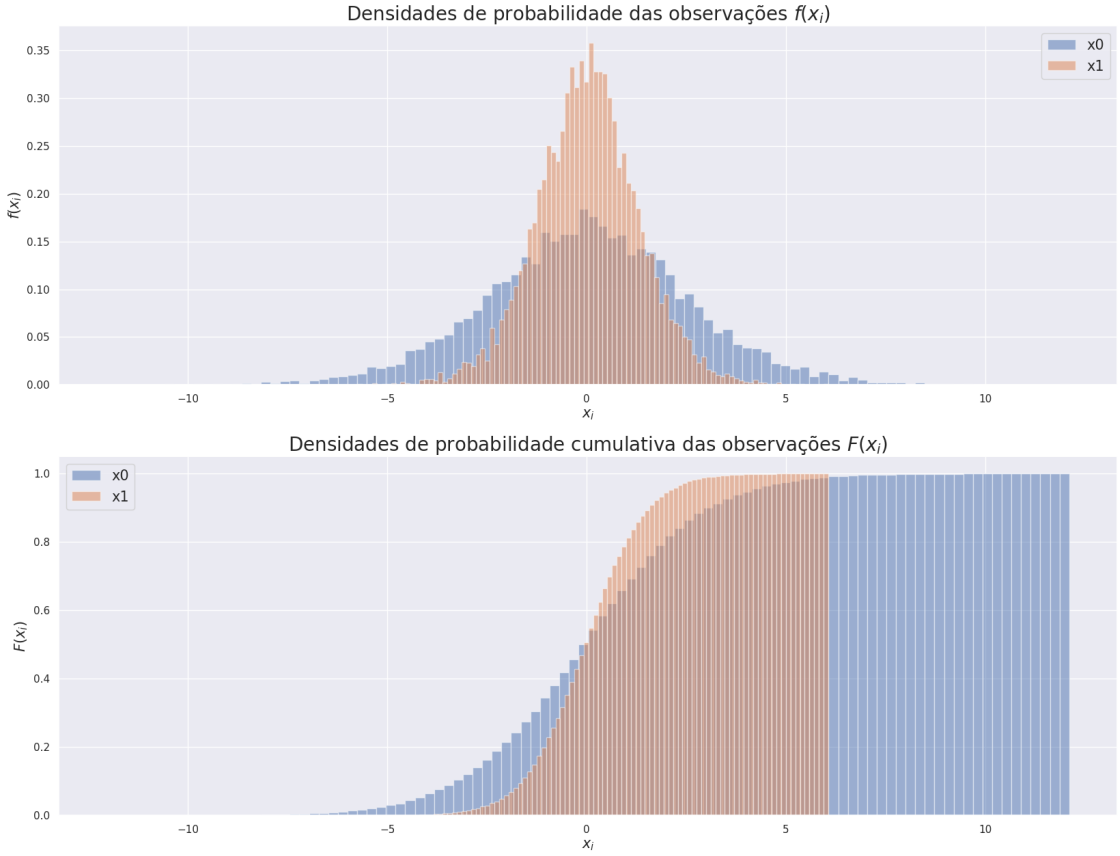
7.2 1.2. Mix sources and generate observations

[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]:
```
# 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 1.3. Perform Analysis - LIKELIHOOD

7.3.1 1.3.1 Execute MCMC Sampling

[8]:
```
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
```

[9]:
```
%%time

SAVE_PATH='./artifacts/perfect_model_specification/ML.pkl'
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 open(SAVE_PATH, 'wb') as f:
        pickle.dump(estimator, f)

else:
    # Read artifact
    with open(SAVE_PATH, 'rb') as f:
        estimator=pickle.load(f)
```

```
CPU times: user 14.8 ms, sys: 3.78 ms, total: 18.6 ms
Wall time: 19.3 ms
```

7.3.2 1.3.2. Parse MCMC Results

[10]:
```
# Get results for analyzed model (best model by default)
# analyzed_model_idx = estimator.B_est_idx
analyzed_model_idx = 10
B_est = estimator.mcmc_results[analyzed_model_idx]['B_est']
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']
```

[11]:
```
print('-'*100)
print('Estimated B:\n{}'.format(B_est))
print('True B:\n{}'.format(np.linalg.inv(A)))
print('-'*100)
```

```
----------------------------------------------------------------------------------
--------------------
Estimated B:
[[ 0.51105554 -0.99906802]
 [ 0.4977057   0.99458036]]
True B:
[[ 0.5 -1. ]
 [ 0.5  1. ]]
```

```
--------------------------------------------------------------------------------
--------------------
```

[12]:
```
# Posteriors
[r['max_posterior'] for r in estimator.mcmc_results]
```

[12]:
```
[-39943.42282851319,
 -39943.42896446798,
 -39943.41709346319,
 -39943.4284763818,
 -39943.42585288991,
 -39943.440684275796,
 -39943.45586670964,
 -39943.45010555786,
 -39943.42985963637,
 -39943.431416595595,
 -39943.432438187294,
 -39943.45908433995,
 -39943.45934658459,
 -39943.427485364904,
 -39943.42641296887,
 -39943.46352682999,
 -39943.44976930605,
 -39943.421375863676,
 -39943.42983964741,
 -39943.44835529387,
 -39943.457337976426,
 -39943.4298806864,
 -39943.44358741392,
 -39943.41849209318,
 -39943.48530443943,
 -39943.45575617601,
 -39943.416124674026,
 -39943.45076913215,
 -39943.4192334433,
 -39943.49254260917]
```

[13]:
```
[r['B_est'] for r in estimator.mcmc_results]
```

[13]:
```
[array([[ 0.5006023 ,  0.98698079],
        [-0.50778459,  1.00517431]]),
 array([[ 0.5006521 ,  0.98646763],
        [-0.50762462,  1.00590607]]),
 array([[-0.50909148,  1.00205889],
        [-0.49910418, -0.98942673]]),
 array([[-0.50910727,  1.00196721],
        [-0.49899042, -0.99023131]]),
```

```
array([[-0.49688417, -0.99606649],
       [ 0.51179544, -0.997116  ]]),
array([[ 0.50798391, -1.00523381],
       [-0.5003555 , -0.98723268]]),
array([[ 0.49881596,  0.99025468],
       [-0.50903503,  1.00195257]]),
array([[-0.4969553 , -0.99492928],
       [ 0.5118217 , -0.99674108]]),
array([[-0.50905055,  1.00144639],
       [-0.4990576 , -0.98915919]]),
array([[ 0.50997345, -1.0006843 ],
       [ 0.49872515,  0.99325287]]),
array([[ 0.51105554, -0.99906802],
       [ 0.4977057 ,  0.99458036]]),
array([[-0.49620988, -0.99758792],
       [ 0.5125244 , -0.99512235]]),
array([[-0.51290279,  0.99465526],
       [ 0.49576185,  0.99774154]]),
array([[-0.51166291,  0.99772042],
       [ 0.49665805,  0.99538265]]),
array([[-0.50821544,  1.00442509],
       [-0.50051836, -0.98772286]]),
array([[ 0.49937669,  0.98967576],
       [-0.50925688,  1.00398556]]),
array([[ 0.50986934, -0.99987405],
       [-0.49865543, -0.99195219]]),
array([[-0.51096   ,  0.99801047],
       [ 0.49744745,  0.99405056]]),
array([[-0.51256287,  0.99559976],
       [ 0.49580183,  0.99669196]]),
array([[-0.49658716, -0.99544922],
       [-0.51241382,  0.99561703]]),
array([[-0.51054246,  1.00033799],
       [ 0.49799629,  0.99222983]]),
array([[-0.51180752,  0.99825206],
       [ 0.49710597,  0.99515306]]),
array([[-0.51084992,  0.99936399],
       [ 0.49759948,  0.99359875]]),
array([[ 0.49915354,  0.98985258],
       [ 0.50885931, -1.00304471]]),
array([[ 0.49702135,  0.99478161],
       [-0.51134975,  0.99701165]]),
array([[-0.50784078,  1.00594516],
       [ 0.50059031,  0.98648245]]),
array([[-0.50647081,  1.00838945],
       [ 0.5023543 ,  0.98430501]]),
array([[-0.50833541,  1.00353597],
```

```
        [ 0.50038241,  0.98831706]]),
 array([[ 0.49495847,  0.99903326],
        [-0.51318977,  0.99246735]]),
 array([[-0.51326118,  0.99387738],
        [-0.49534203, -0.99962459]])]
```

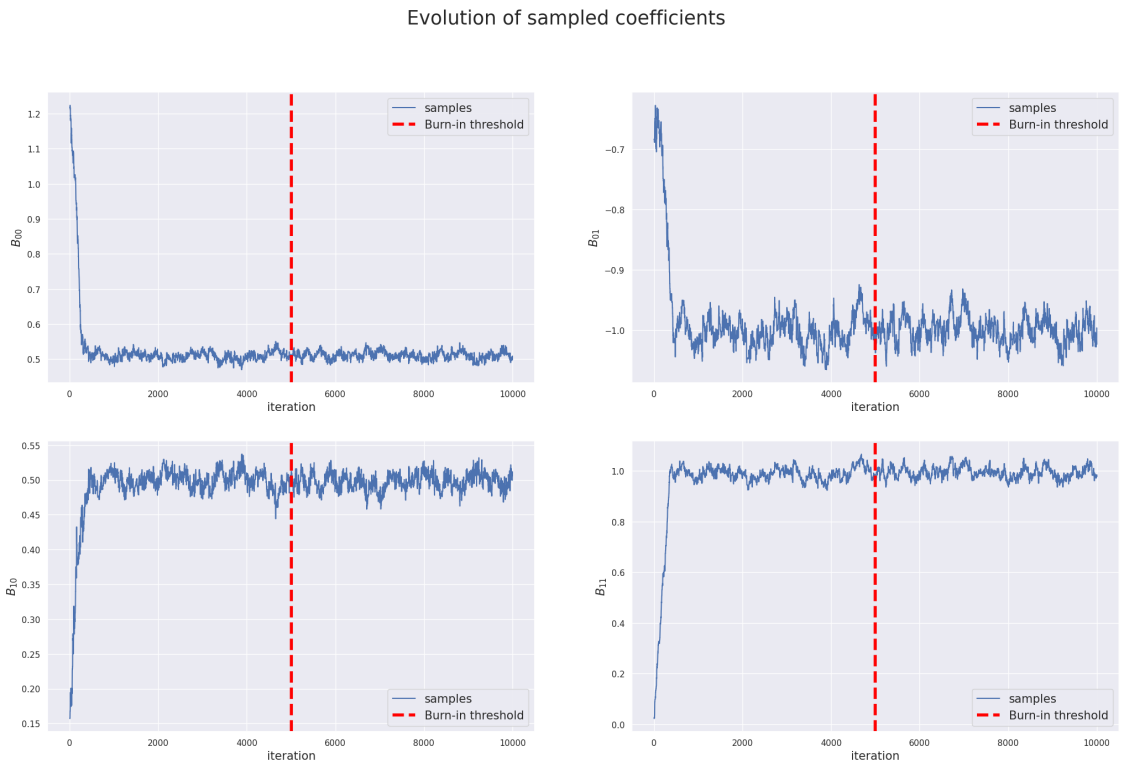
7.3.3 1.3.3. Plot sampled coefficients stochastic process - Markov Chain evolution

[14]:
```
# Plot sampled coefficients
fig, axs = plt.subplots(
    nrows=2, ncols=2,
    figsize=(25,15)
)

fig.suptitle(
    'Evolution of sampled coefficients',
    fontsize=25
)

# B_00
axs[0,0].plot(
    logs.iteration,
    samples[:, 0, 0],
    label='samples'
)
axs[0,0].axvline(
    estimator.burn_in_start,
    color='red',
    linestyle='--',
    linewidth=4,
    label='Burn-in threshold'
)
axs[0,0].set_xlabel(
    'iteration',
    fontsize=15
)
axs[0,0].set_ylabel(
    '$B_{00}$',
    fontsize=15
)
axs[0,0].legend(
    fontsize=15
)

# B_01
axs[0,1].plot(
    logs.iteration,
```

```
    samples[:, 0, 1],
    label='samples'
)
axs[0,1].axvline(
    estimator.burn_in_start,
    color='red',
    linestyle='--',
    linewidth=4,
    label='Burn-in threshold'
)
axs[0,1].set_xlabel(
    'iteration',
    fontsize=15
)
axs[0,1].set_ylabel(
    '$B_{01}$',
    fontsize=15
)
axs[0,1].legend(
    fontsize=15
)

# B_10
axs[1,0].plot(
    logs.iteration,
    samples[:, 1, 0],
    label='samples'
)
axs[1,0].axvline(
    estimator.burn_in_start,
    color='red',
    linestyle='--',
    linewidth=4,
    label='Burn-in threshold'
)
axs[1,0].set_xlabel(
    'iteration',
    fontsize=15
)
axs[1,0].set_ylabel(
    '$B_{10}$',
    fontsize=15
)
axs[1,0].legend(
    fontsize=15
)
```

```
# B_11
axs[1,1].plot(
    logs.iteration,
    samples[:, 1, 1],
    label='samples'
)
axs[1,1].axvline(
    estimator.burn_in_start,
    color='red',
    linestyle='--',
    linewidth=4,
    label='Burn-in threshold'
)
axs[1,1].set_xlabel(
    'iteration',
    fontsize=15
)
axs[1,1].set_ylabel(
    '$B_{11}$',
    fontsize=15
)
axs[1,1].legend(
    fontsize=15
)
```

[14]: <matplotlib.legend.Legend at 0x7f487a59e010>



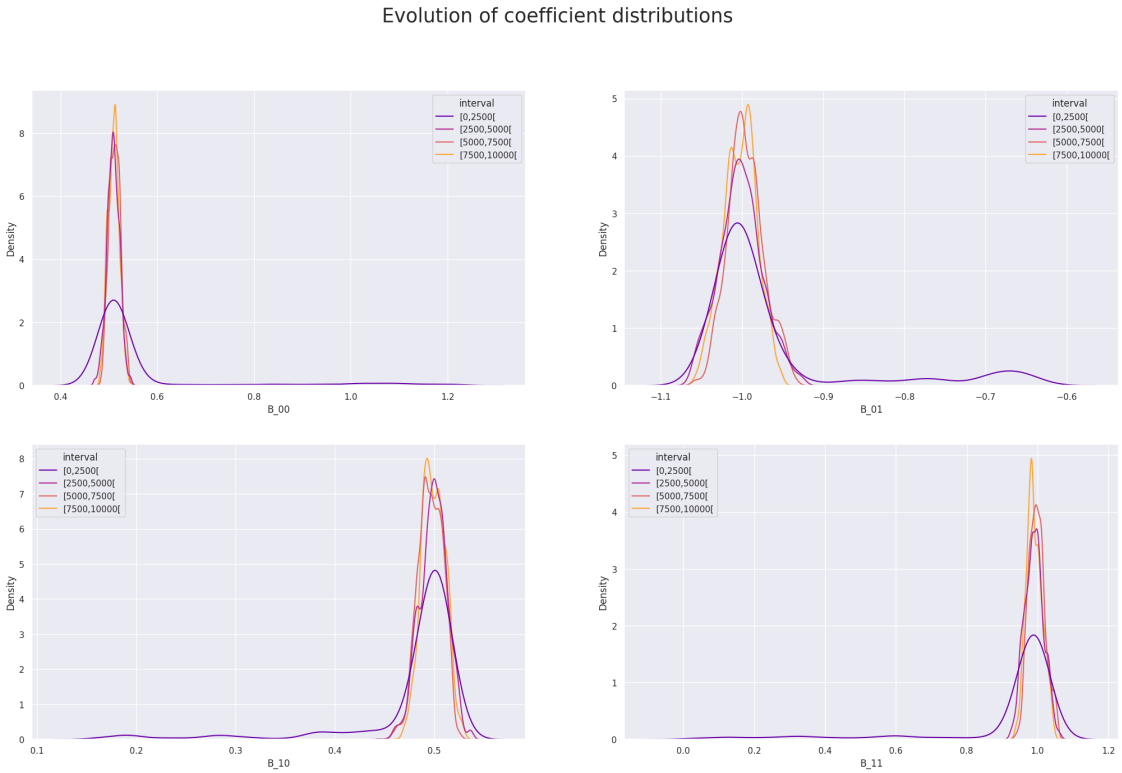
7.3.4 1.3.4. Plot sampled coefficients distributions - Markov Chain evolution

[15]:
```
################################
# Evolution of distributions #
################################

# Get step size and evaluated points
STEP_SIZE=2500
PALETTE='plasma'

i=0
evaluated_intervals=[]
while i<N_SAMPLES:
    start=i
    end=min(
        i+STEP_SIZE,
        N_SAMPLES
    )
    evaluated_intervals.append(
        (start, end)
    )
    i = i + STEP_SIZE
```

```
# Window samples and construct dataframe for plotting
plot_df=pd.DataFrame()
for start, end in evaluated_intervals:
    wdw_df = pd.DataFrame(
        data={
            'interval': ['[{},{}['.format(start, end)]*(end-start)
        }
    )
    wdw_samples = samples[start:end,:,:]
    for i, j in np.ndindex(B_est.shape):
        wdw_df['B_{}{}'.format(i, j)] = wdw_samples[:,i,j]

    plot_df = pd.concat(
        [
            plot_df,
            wdw_df
        ],
        axis=0
    ).reset_index(
        drop=True
    )

# Plot coefficient distribution evolution
fig, axs = plt.subplots(
    nrows=2, ncols=2,
    figsize=(25,15)
)

fig.suptitle(
    'Evolution of coefficient distributions',
    fontsize=25
)

# B_00
sns.kdeplot(
    data=plot_df,
    x='B_00',
    hue='interval',
    ax=axs[0,0],
    palette=PALETTE
)

# B_01
sns.kdeplot(
    data=plot_df,
    x='B_01',
    hue='interval',
```

```
    ax=axs[0,1],
    palette=PALETTE
)

# B_10
sns.kdeplot(
    data=plot_df,
    x='B_10',
    hue='interval',
    ax=axs[1,0],
    palette=PALETTE
)

# B_11
sns.kdeplot(
    data=plot_df,
    x='B_11',
    hue='interval',
    ax=axs[1,1],
    palette=PALETTE
)
```

[15]: <Axes: xlabel='B_11', ylabel='Density'>



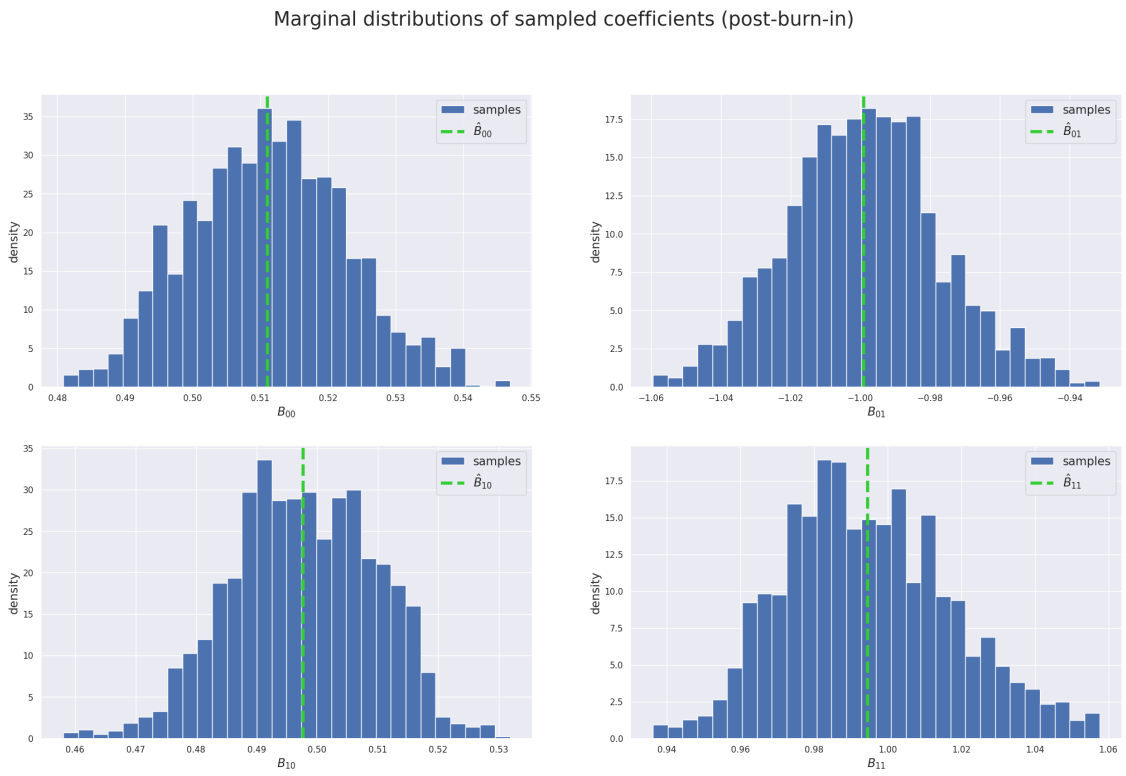
7.3.5 1.3.5. Plot Marginal Posteriors for Coefficients - Post-burn-in

[16]:
```
# Plot sampled coefficients
fig, axs = plt.subplots(
    nrows=2, ncols=2,
    figsize=(25,15)
)

fig.suptitle(
    'Marginal distributions of sampled coefficients (post-burn-in)',
    fontsize=25
)

# B_00
axs[0,0].hist(
    valid_samples[:, 0, 0],
    density=True,
    bins=30,
    label='samples'
)
axs[0,0].axvline(
    B_est[0, 0],
    color='limegreen',
    linestyle='--',
    linewidth=4,
    label='$\hat{B}_{00}$'
)
axs[0,0].set_xlabel(
    '$B_{00}$',
    fontsize=15
)
axs[0,0].set_ylabel(
    'density',
    fontsize=15
)
axs[0,0].legend(
    loc='upper right',
    fontsize=15
)


# B_01
axs[0,1].hist(
    valid_samples[:, 0, 1],
    density=True,
```

```
    bins=30,
    label='samples'
)
axs[0,1].axvline(
    B_est[0, 1],
    color='limegreen',
    linestyle='--',
    linewidth=4,
    label='$\hat{B}_{01}$'
)
axs[0,1].set_xlabel(
    '$B_{01}$',
    fontsize=15
)
axs[0,1].set_ylabel(
    'density',
    fontsize=15
)
axs[0,1].legend(
    loc='upper right',
    fontsize=15
)

# B_10
axs[1,0].hist(
    valid_samples[:, 1, 0],
    density=True,
    bins=30,
    label='samples'
)
axs[1,0].axvline(
    B_est[1, 0],
    color='limegreen',
    linestyle='--',
    linewidth=4,
    label='$\hat{B}_{10}$'
)
axs[1,0].set_xlabel(
    '$B_{10}$',
    fontsize=15
)
axs[1,0].set_ylabel(
    'density',
    fontsize=15
)
axs[1,0].legend(
    loc='upper right',
```

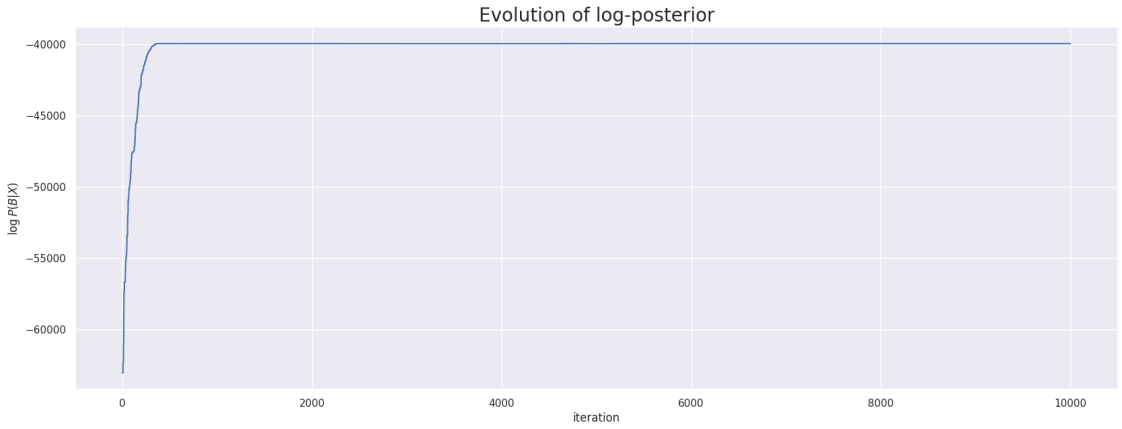
```
    fontsize=15
)

# B_11
axs[1,1].hist(
    valid_samples[:, 1, 1],
    density=True,
    bins=30,
    label='samples'
)
axs[1,1].axvline(
    B_est[1, 1],
    color='limegreen',
    linestyle='--',
    linewidth=4,
    label='$\hat{B}_{11}$'
)
axs[1,1].set_xlabel(
    '$B_{11}$',
    fontsize=15
)
axs[1,1].set_ylabel(
    'density',
    fontsize=15
)
axs[1,1].legend(
    loc='upper right',
    fontsize=15
)
```

[16]: <matplotlib.legend.Legend at 0x7f487b03bd90>



7.3.6 1.3.6. Plot evolution of log-posterior

[17]:
```
fig = plt.figure(figsize=(20,7))
sns.lineplot(
    data=logs,
    x='iteration',
    y='log_posterior'
)
plt.ylabel(
    '$\log P(B|X)$'
)
t = plt.title(
    'Evolution of log-posterior',
    fontsize=20
)
```

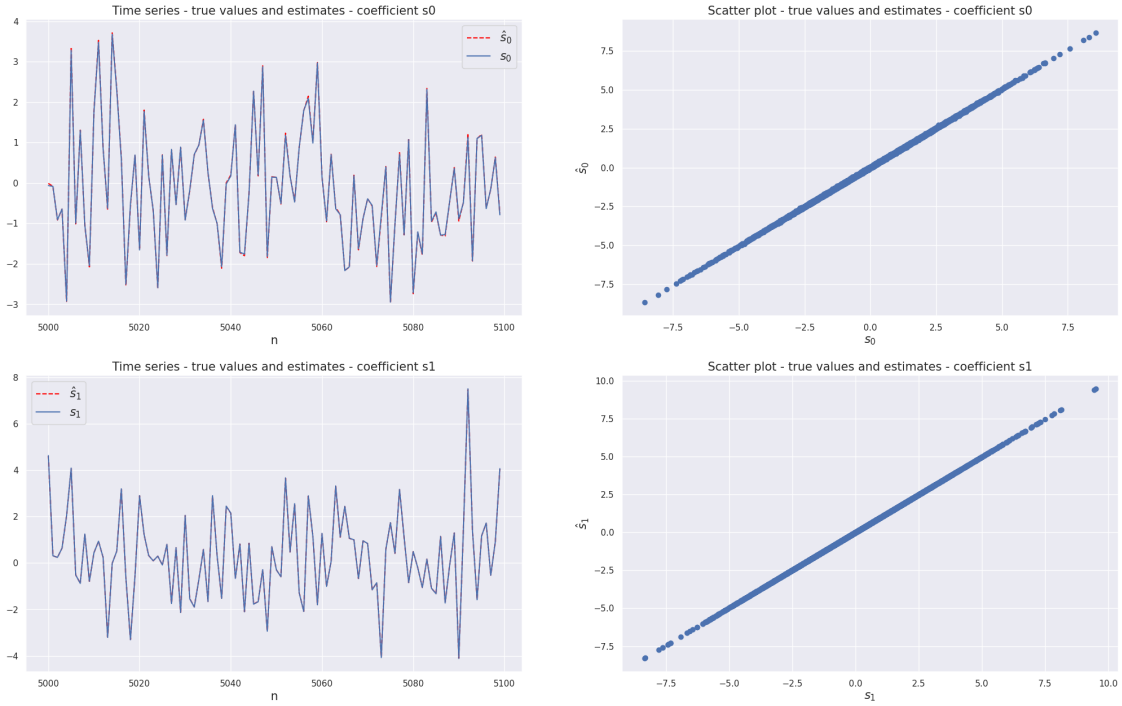


[18]:
```
fig, axs = plt.subplots(
    nrows=2, ncols=2,
    figsize=(25,15)
)
NPOINTS=100
PLOT_START=NOBS//2
PLOT_END=PLOT_START+NPOINTS

t=range(NOBS)
s_est = B_est@x

# Axs 00
axs[0,0].plot(
    t[PLOT_START:PLOT_END],
    s_est[0,PLOT_START:PLOT_END],
    label='$\hat{s}_{0}$',
    color='red',
    linestyle='--'
)
axs[0,0].plot(
    t[PLOT_START:PLOT_END],
    s[0,PLOT_START:PLOT_END],
    label='$s_{0}$'
)
axs[0,0].set_xlabel(
    'n',
    fontsize=15
)
axs[0,0].set_title(
    'Time series - true values and estimates - coefficient s0',
    fontsize=15
)
```

```
axs[0,0].legend(fontsize=15)

# Axs 01
axs[0,1].scatter(
    s[0,:],
    s_est[0,:],
)
axs[0,1].set_xlabel(
    '$s_{0}$',
    fontsize=15
)
axs[0,1].set_ylabel(
    '$\hat{s}_{0}$',
    fontsize=15
)
axs[0,1].set_title(
    'Scatter plot - true values and estimates - coefficient s0',
    fontsize=15
)

# Axs 10
axs[1,0].plot(
    t[PLOT_START:PLOT_END],
    s_est[1,PLOT_START:PLOT_END],
    label='$\hat{s}_{1}$',
    color='red',
    linestyle='--'
)
axs[1,0].plot(
    t[PLOT_START:PLOT_END],
    s[1,PLOT_START:PLOT_END],
    label='$s_{1}$'
)
axs[1,0].set_xlabel(
    'n',
    fontsize=15
)
axs[1,0].set_title(
    'Time series - true values and estimates - coefficient s1',
    fontsize=15
)
axs[1,0].legend(fontsize=15)

# Axs 11
axs[1,1].scatter(
    s[1,:],
    s_est[1,:],
```

```
)
axs[1,1].set_xlabel(
    '$s_{1}$',
    fontsize=15
)
axs[1,1].set_ylabel(
    '$\hat{s}_{1}$',
    fontsize=15
)
axs[1,1].set_title(
    'Scatter plot - true values and estimates - coefficient s1',
    fontsize=15
)
```

[18]: Text(0.5, 1.0, 'Scatter plot - true values and estimates - coefficient s1')



[19]:
```
# 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: 0.00014791877114411317
```

7.4 1.4. Perform Analysis - DETERMINANT PRIOR

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

7.4.1 1.4.1 Execute MCMC Sampling

[20]:
```
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
```

[21]:
```
%%time

SAVE_PATH='./artifacts/perfect_model_specification/Det_1.pkl'
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 open(SAVE_PATH, 'wb') as f:
        pickle.dump(estimator, f)

else:
    # Read artifact
```

```
    with open(SAVE_PATH, 'rb') as f:
        estimator=pickle.load(f)
```

```
CPU times: user 76 ms, sys: 48.4 ms, total: 124 ms
Wall time: 20.4 ms
```

7.4.2 1.4.2. Parse MCMC Results

[22]:
```
# Get results for analyzed model (best model by default)
# analyzed_model_idx = estimator.B_est_idx
analyzed_model_idx = 10
B_est = estimator.mcmc_results[analyzed_model_idx]['B_est']
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']
```

[23]:
```
print('-'*100)
print('Estimated B:\n{}'.format(B_est))
print('True B:\n{}'.format(np.linalg.inv(A)))
print('-'*100)
```

```
----------------------------------------------------------------------------------------
--------------------
Estimated B:
[[ 0.511216   -0.9982714 ]
 [ 0.49742711  0.99457323]]
True B:
[[ 0.5 -1. ]
 [ 0.5  1. ]]
----------------------------------------------------------------------------------------
--------------------
```

[24]:
```
# Posteriors
[r['max_posterior'] for r in estimator.mcmc_results]
```

[24]:
```
[-39942.087270799886,
 -39942.04882357925,
 -39942.0501270747,
 -39942.04194048764,
 -39942.04648401399,
 -40140.31131374631,
 -39942.101992415475,
 -39942.04460195968,
 -39942.056497326485,
 -39942.06321798317,
 -39942.05099628422,
```

```
-39942.03932546771,
-inf,
-40140.3061782553,
-39942.06248563203,
-39942.06714759611,
-inf,
-40140.31091606717,
-inf,
-40140.315495132185,
-40140.30640814799,
-40140.309981635626,
-inf,
-inf,
-39942.046827667094,
-40140.30850474406,
-40140.29554015431,
-40140.296583337826,
-39942.0342601322,
-39942.06444776925]
```

[25]:
```
[r['B_est'] for r in estimator.mcmc_results]
```

[25]:
```
[array([[ 0.50201998,  0.98412438],
        [-0.50611382,  1.00772184]]),
 array([[ 0.50203779,  0.9830094 ],
        [-0.50620671,  1.00816235]]),
 array([[-0.51103773,  0.99851137],
        [-0.49741919, -0.9939602 ]]),
 array([[-0.50888974,  1.00275772],
        [-0.49950901, -0.98959353]]),
 array([[-0.49561432, -0.99850475],
        [ 0.51275611, -0.99342547]]),
 array([[ 0.50334208, -0.98658035],
        [-0.49107725, -0.9789222 ]]),
 array([[ 0.50094899,  0.986337  ],
        [-0.50727469,  1.00538684]]),
 array([[-0.49765305, -0.99556677],
        [ 0.5112469 , -0.99742463]]),
 array([[-0.50864933,  1.00281299],
        [-0.50002558, -0.98823065]]),
 array([[ 0.51015845, -1.00144234],
        [ 0.49882085,  0.991284  ]]),
 array([[ 0.511216  , -0.9982714 ],
        [ 0.49742711,  0.99457323]]),
 array([[-0.49638694, -0.9973222 ],
        [ 0.51246134, -0.99575969]]),
 array([[0.80691921, 2.5052391 ],
```

```
       [2.60165615, 2.15296942]]),
 array([[-0.50601827,  0.98164283],
       [ 0.48892475,  0.98382349]]),
 array([[-0.50764492,  1.00578747],
       [-0.50132146, -0.98714519]]),
 array([[ 0.49811291,  0.99341339],
       [-0.51038133,  1.00039451]]),
 array([[ 1.10083444, -0.59172085],
       [-1.7380585 , -1.82884332]]),
 array([[-0.50665036,  0.97982882],
       [ 0.48818713,  0.98581928]]),
 array([[-0.00437558,  2.47639208],
       [ 2.78081332,  1.60621876]]),
 array([[-0.49254304, -0.97632271],
       [-0.50262078,  0.98842731]]),
 array([[-0.50487449,  0.98469083],
       [ 0.49031219,  0.98093001]]),
 array([[-0.50379788,  0.98606685],
       [ 0.49092459,  0.98010997]]),
 array([[-1.91064012,  0.30439887],
       [-0.37064289,  1.95697512]]),
 array([[ 1.30645535,  1.17660585],
       [ 2.42957998, -0.37927439]]),
 array([[ 0.49858843,  0.99127815],
       [-0.5098613 ,  0.99988537]]),
 array([[-0.50186317,  0.98918906],
       [ 0.49299929,  0.97508148]]),
 array([[-0.50073896,  0.99231703],
       [ 0.49411638,  0.97235079]]),
 array([[-0.50101421,  0.99172936],
       [ 0.49399161,  0.97274755]]),
 array([[ 0.49734804,  0.9951518 ],
       [-0.51116027,  0.99832114]]),
 array([[-0.51076257,  1.00124032],
       [-0.4979432 , -0.99353443]])]
```

7.4.3 1.4.3. Plot sampled coefficients stochastic process - Markov Chain evolution

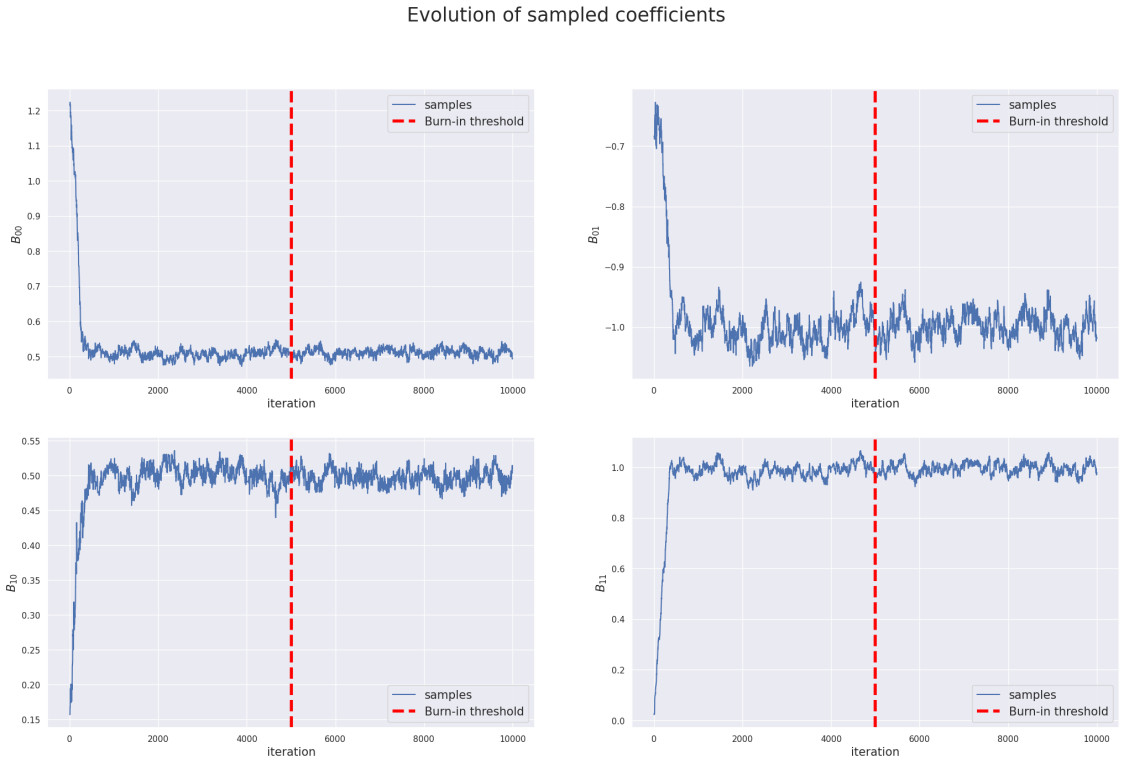
[26]:
```
# Plot sampled coefficients
fig, axs = plt.subplots(
    nrows=2, ncols=2,
    figsize=(25,15)
)

fig.suptitle(
    'Evolution of sampled coefficients',
    fontsize=25
```

```
)

# B_00
axs[0,0].plot(
    logs.iteration,
    samples[:, 0, 0],
    label='samples'
)
axs[0,0].axvline(
    estimator.burn_in_start,
    color='red',
    linestyle='--',
    linewidth=4,
    label='Burn-in threshold'
)
axs[0,0].set_xlabel(
    'iteration',
    fontsize=15
)
axs[0,0].set_ylabel(
    '$B_{00}$',
    fontsize=15
)
axs[0,0].legend(
    fontsize=15
)

# B_01
axs[0,1].plot(
    logs.iteration,
    samples[:, 0, 1],
    label='samples'
)
axs[0,1].axvline(
    estimator.burn_in_start,
    color='red',
    linestyle='--',
    linewidth=4,
    label='Burn-in threshold'
)
axs[0,1].set_xlabel(
    'iteration',
    fontsize=15
)
axs[0,1].set_ylabel(
    '$B_{01}$',
    fontsize=15
```

```
)
axs[0,1].legend(
    fontsize=15
)

# B_10
axs[1,0].plot(
    logs.iteration,
    samples[:, 1, 0],
    label='samples'
)
axs[1,0].axvline(
    estimator.burn_in_start,
    color='red',
    linestyle='--',
    linewidth=4,
    label='Burn-in threshold'
)
axs[1,0].set_xlabel(
    'iteration',
    fontsize=15
)
axs[1,0].set_ylabel(
    '$B_{10}$',
    fontsize=15
)
axs[1,0].legend(
    fontsize=15
)

# B_11
axs[1,1].plot(
    logs.iteration,
    samples[:, 1, 1],
    label='samples'
)
axs[1,1].axvline(
    estimator.burn_in_start,
    color='red',
    linestyle='--',
    linewidth=4,
    label='Burn-in threshold'
)
axs[1,1].set_xlabel(
    'iteration',
    fontsize=15
)
```

```
axs[1,1].set_ylabel(
    '$B_{11}$',
    fontsize=15
)
axs[1,1].legend(
    fontsize=15
)
```

[26]: <matplotlib.legend.Legend at 0x7f487ac30d50>



7.4.4 1.4.4. Plot sampled coefficients distributions - Markov Chain evolution

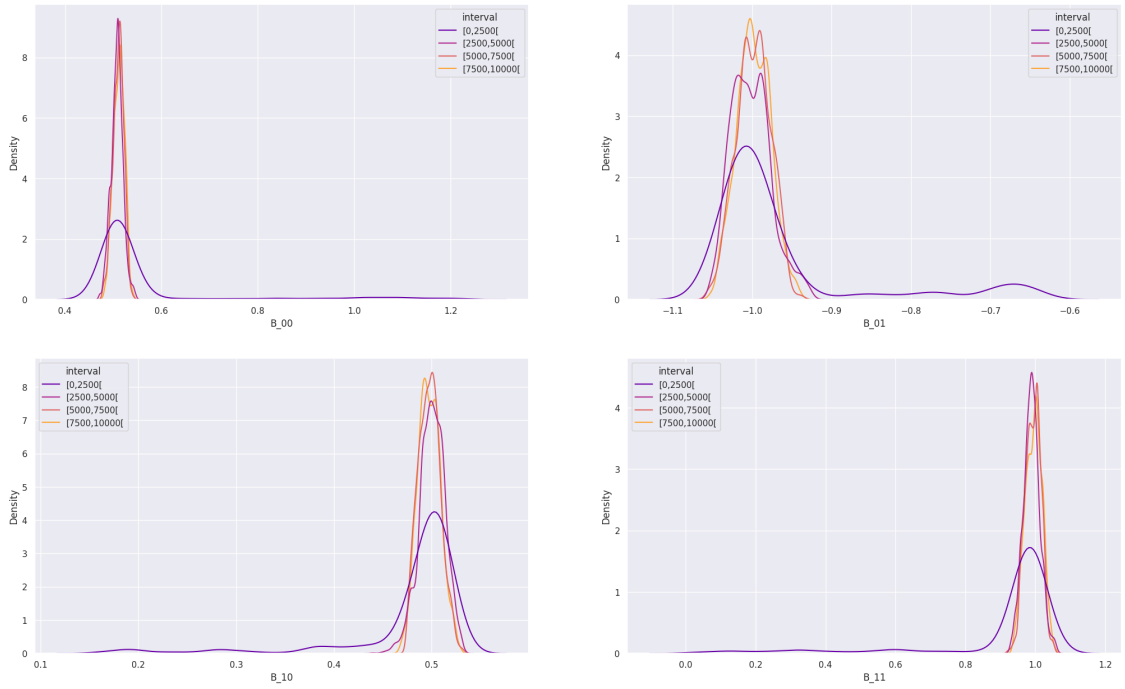
[27]:
```
##############################
# Evolution of distributions #
##############################

# Get step size and evaluated points
STEP_SIZE=2500
PALETTE='plasma'

i=0
evaluated_intervals=[]
while i<N_SAMPLES:
```

```
    start=i
    end=min(
        i+STEP_SIZE,
        N_SAMPLES
    )
    evaluated_intervals.append(
        (start, end)
    )
    i = i + STEP_SIZE

# Window samples and construct dataframe for plotting
plot_df=pd.DataFrame()
for start, end in evaluated_intervals:
    wdw_df = pd.DataFrame(
        data={
            'interval': ['[{},{}['.format(start, end)]*(end-start)
        }
    )
    wdw_samples = samples[start:end,:,:]
    for i, j in np.ndindex(B_est.shape):
        wdw_df['B_{}{}'.format(i, j)] = wdw_samples[:,i,j]

    plot_df = pd.concat(
        [
            plot_df,
            wdw_df
        ],
        axis=0
    ).reset_index(
        drop=True
    )

# Plot coefficient distribution evolution
fig, axs = plt.subplots(
    nrows=2, ncols=2,
    figsize=(25,15)
)

fig.suptitle(
    'Evolution of coefficient distributions',
    fontsize=25
)

# B_00
sns.kdeplot(
    data=plot_df,
    x='B_00',
```

```
    hue='interval',
    ax=axs[0,0],
    palette=PALETTE
)

# B_01
sns.kdeplot(
    data=plot_df,
    x='B_01',
    hue='interval',
    ax=axs[0,1],
    palette=PALETTE
)

# B_10
sns.kdeplot(
    data=plot_df,
    x='B_10',
    hue='interval',
    ax=axs[1,0],
    palette=PALETTE
)

# B_11
sns.kdeplot(
    data=plot_df,
    x='B_11',
    hue='interval',
    ax=axs[1,1],
    palette=PALETTE
)
```

[27]: <Axes: xlabel='B_11', ylabel='Density'>

Evolution of coefficient distributions



7.4.5 1.4.5. Plot Marginal Posteriors for Coefficients - Post-burn-in

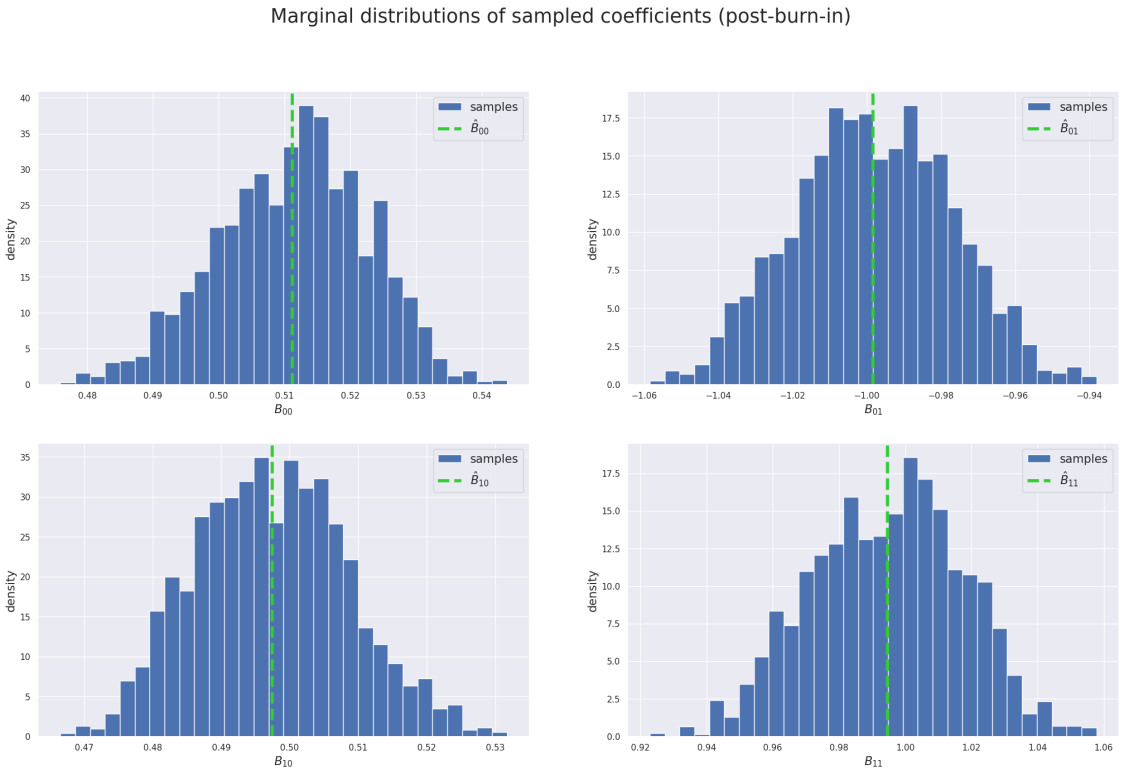
[28]:
```
# Plot sampled coefficients
fig, axs = plt.subplots(
    nrows=2, ncols=2,
    figsize=(25,15)
)

fig.suptitle(
    'Marginal distributions of sampled coefficients (post-burn-in)',
    fontsize=25
)

# B_00
axs[0,0].hist(
    valid_samples[:, 0, 0],
    density=True,
    bins=30,
    label='samples'
)
axs[0,0].axvline(
    B_est[0, 0],
```

```
    color='limegreen',
    linestyle='--',
    linewidth=4,
    label='$\hat{B}_{00}$'
)
axs[0,0].set_xlabel(
    '$B_{00}$',
    fontsize=15
)
axs[0,0].set_ylabel(
    'density',
    fontsize=15
)
axs[0,0].legend(
    loc='upper right',
    fontsize=15
)


# B_01
axs[0,1].hist(
    valid_samples[:, 0, 1],
    density=True,
    bins=30,
    label='samples'
)
axs[0,1].axvline(
    B_est[0, 1],
    color='limegreen',
    linestyle='--',
    linewidth=4,
    label='$\hat{B}_{01}$'
)
axs[0,1].set_xlabel(
    '$B_{01}$',
    fontsize=15
)
axs[0,1].set_ylabel(
    'density',
    fontsize=15
)
axs[0,1].legend(
    loc='upper right',
    fontsize=15
)

# B_10
```

```
axs[1,0].hist(
    valid_samples[:, 1, 0],
    density=True,
    bins=30,
    label='samples'
)
axs[1,0].axvline(
    B_est[1, 0],
    color='limegreen',
    linestyle='--',
    linewidth=4,
    label='$\hat{B}_{10}$'
)
axs[1,0].set_xlabel(
    '$B_{10}$',
    fontsize=15
)
axs[1,0].set_ylabel(
    'density',
    fontsize=15
)
axs[1,0].legend(
    loc='upper right',
    fontsize=15
)

# B_11
axs[1,1].hist(
    valid_samples[:, 1, 1],
    density=True,
    bins=30,
    label='samples'
)
axs[1,1].axvline(
    B_est[1, 1],
    color='limegreen',
    linestyle='--',
    linewidth=4,
    label='$\hat{B}_{11}$'
)
axs[1,1].set_xlabel(
    '$B_{11}$',
    fontsize=15
)
axs[1,1].set_ylabel(
    'density',
    fontsize=15
```

```
)
axs[1,1].legend(
    loc='upper right',
    fontsize=15
)
```

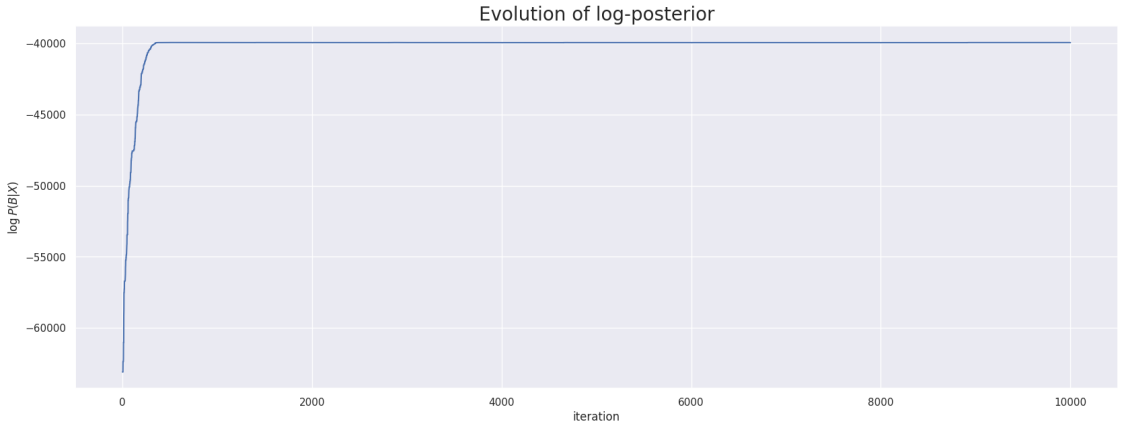
[28]: <matplotlib.legend.Legend at 0x7f487a77df10>



7.4.6 1.4.6. Plot evolution of log-posterior

[29]:
```
fig = plt.figure(figsize=(20,7))
sns.lineplot(
    data=logs,
    x='iteration',
    y='log_posterior'
)
plt.ylabel(
    '$\log P(B|X)$'
)
t = plt.title(
    'Evolution of log-posterior',
    fontsize=20
```

```
)
```



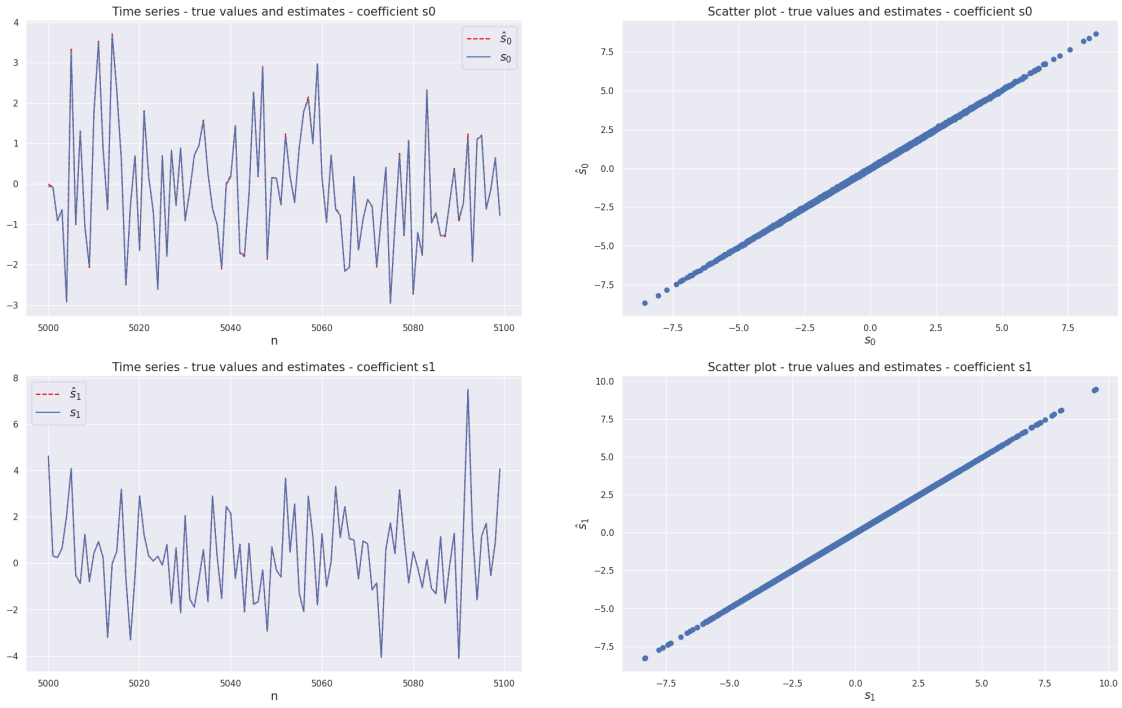
7.4.7 1.4.7. Plot Source Separation Results

[30]:
```
fig, axs = plt.subplots(
    nrows=2, ncols=2,
    figsize=(25,15)
)
NPOINTS=100
PLOT_START=NOBS//2
PLOT_END=PLOT_START+NPOINTS

t=range(NOBS)
s_est = B_est@x

# Axs 00
axs[0,0].plot(
    t[PLOT_START:PLOT_END],
    s_est[0,PLOT_START:PLOT_END],
    label='$\hat{s}_{0}$',
    color='red',
    linestyle='--'
)
axs[0,0].plot(
    t[PLOT_START:PLOT_END],
    s[0,PLOT_START:PLOT_END],
    label='$s_{0}$'
)
axs[0,0].set_xlabel(
    'n',
    fontsize=15
)
```

```
axs[0,0].set_title(
    'Time series - true values and estimates - coefficient s0',
    fontsize=15
)
axs[0,0].legend(fontsize=15)

# Axs 01
axs[0,1].scatter(
    s[0,:],
    s_est[0,:],
)
axs[0,1].set_xlabel(
    '$s_{0}$',
    fontsize=15
)
axs[0,1].set_ylabel(
    '$\hat{s}_{0}$',
    fontsize=15
)
axs[0,1].set_title(
    'Scatter plot - true values and estimates - coefficient s0',
    fontsize=15
)

# Axs 10
axs[1,0].plot(
    t[PLOT_START:PLOT_END],
    s_est[1,PLOT_START:PLOT_END],
    label='$\hat{s}_{1}$',
    color='red',
    linestyle='--'
)
axs[1,0].plot(
    t[PLOT_START:PLOT_END],
    s[1,PLOT_START:PLOT_END],
    label='$s_{1}$'
)
axs[1,0].set_xlabel(
    'n',
    fontsize=15
)
axs[1,0].set_title(
    'Time series - true values and estimates - coefficient s1',
    fontsize=15
)
axs[1,0].legend(fontsize=15)
```

```
# Axs 11
axs[1,1].scatter(
    s[1,:],
    s_est[1,:],
)
axs[1,1].set_xlabel(
    '$s_{1}$',
    fontsize=15
)
axs[1,1].set_ylabel(
    '$\hat{s}_{1}$',
    fontsize=15
)
axs[1,1].set_title(
    'Scatter plot - true values and estimates - coefficient s1',
    fontsize=15
)
```

[30]: Text(0.5, 1.0, 'Scatter plot - true values and estimates - coefficient s1')



[31]:
```
# 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: 0.00015101624153912317
```

7.5 1.5. Perform Analysis - near-identity transformation

Prior:

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

7.5.1 1.5.1 Execute MCMC Sampling

[32]:
```
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
```

[33]:
```
%%time

SAVE_PATH='./artifacts/perfect_model_specification/identity.pkl'
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 open(SAVE_PATH, 'wb') as f:
        pickle.dump(estimator, f)

else:
    # Read artifact
    with open(SAVE_PATH, 'rb') as f:
        estimator=pickle.load(f)
```

```
CPU times: user 27.6 ms, sys: 73.7 ms, total: 101 ms
Wall time: 17.2 ms
```

7.5.2 1.5.2. Parse MCMC Results

[34]:
```
# Get results for analyzed model (best model by default)
analyzed_model_idx = estimator.B_est_idx
# analyzed_model_idx = 10
B_est = estimator.mcmc_results[analyzed_model_idx]['B_est']
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']
```

[35]:
```
print('-'*100)
print('Estimated B:\n{}'.format(B_est))
print('True B:\n{}'.format(np.linalg.inv(A)))
print('-'*100)
```

```
----------------------------------------------------------------------------------------
--------------------
Estimated B:
[[ 0.52541629 -0.96509125]
 [ 0.4819765   1.02202536]]
True B:
[[ 0.5 -1. ]
 [ 0.5  1. ]]
----------------------------------------------------------------------------------------
--------------------
```

[36]:
```
# Posteriors
[r['max_posterior'] for r in estimator.mcmc_results]
```

[36]:
```
[-40003.33986286617,
 -40003.33412990008,
 -40080.03240232674,
 -40080.01846910019,
 -40079.98361290401,
```

```
 -40036.7281212995,
 -40003.47864060574,
 -40079.960189195626,
 -40080.048367538315,
 -40037.23205428763,
 -40003.33819303506,
 -40003.333963847675,
 -40037.25617979229,
 -40037.23354565022,
 -40080.02504220005,
 -40003.45428200776,
 -40060.10397272788,
 -40037.24905176969,
 -40037.267097064956,
 -40036.71719124863,
 -40060.50390461173,
 -40037.27102482258,
 -40036.74095206561,
 -40060.49921831398,
 -40003.44592451662,
 -40037.24606122002,
 -40036.70551715535,
 -40036.70978456624,
 -40003.47098648538,
 -40080.00570511746]
```

[37]:
```
[r['B_est'] for r in estimator.mcmc_results]
```

[37]:
```
[array([[ 0.52657144, -0.96101113],
        [ 0.47987664,  1.02512749]]),
 array([[ 0.52810804, -0.95779922],
        [ 0.47797423,  1.02864512]]),
 array([[-0.49846808,  1.01966263],
        [-0.50856609, -0.96585629]]),
 array([[-0.50166723,  1.0127349 ],
        [-0.50550397, -0.97272185]]),
 array([[-0.48868009, -1.00647242],
        [ 0.51751248, -0.97868359]]),
 array([[-0.4989353 , -0.98502877],
        [-0.50841087,  1.00312275]]),
 array([[ 0.51431539,  0.957138  ],
        [-0.4935368 ,  1.0291718 ]]),
 array([[-0.49276917, -0.99857336],
        [ 0.51413119, -0.98724762]]),
 array([[-0.50210331,  1.01245383],
        [-0.50480857, -0.9747335 ]]),
 array([[-0.50992389,  0.99442963],
```

```
       [ 0.49676675,  0.99372429]]),
array([[ 0.52531552, -0.96407822],
       [ 0.48161634,  1.02172254]]),
array([[ 0.52541629, -0.96509125],
       [ 0.4819765 ,  1.02202536]]),
array([[-0.50977375,  0.99476192],
       [ 0.4969846 ,  0.9937229 ]]),
array([[-0.50915019,  0.99595801],
       [ 0.49779097,  0.99261434]]),
array([[-0.50142164,  1.01441395],
       [-0.50605465, -0.97228117]]),
array([[ 0.51410866,  0.95578652],
       [-0.4933724 ,  1.02820817]]),
array([[ 0.50884085, -1.00082955],
       [-0.4991057 , -0.98470467]]),
array([[-0.51258081,  0.9882738 ],
       [ 0.49416124,  1.00031679]]),
array([[-0.51378732,  0.98694454],
       [ 0.49263562,  1.00335718]]),
array([[-0.49915446, -0.98534682],
       [-0.50781053,  1.00442581]]),
array([[ 0.49673545,  0.99289255],
       [ 0.51078344, -0.99149865]]),
array([[-0.51508808,  0.98542393],
       [ 0.49141615,  1.00516335]]),
array([[-0.49981428, -0.98302748],
       [-0.50711388,  1.0057289 ]]),
array([[ 0.49771994,  0.99094546],
       [ 0.50996901, -0.99300103]]),
array([[ 0.51667548,  0.94961805],
       [-0.49103584,  1.03631741]]),
array([[-0.5158674 ,  0.98205537],
       [ 0.49055426,  1.00697186]]),
array([[-0.50046823, -0.98156419],
       [-0.50617951,  1.00774887]]),
array([[-0.50066596, -0.98175135],
       [-0.50581618,  1.00679373]]),
array([[ 0.51521337,  0.9532719 ],
       [-0.49270868,  1.03228817]]),
array([[-0.50070701,  1.0137425 ],
       [-0.50591999, -0.97063889]])]
```

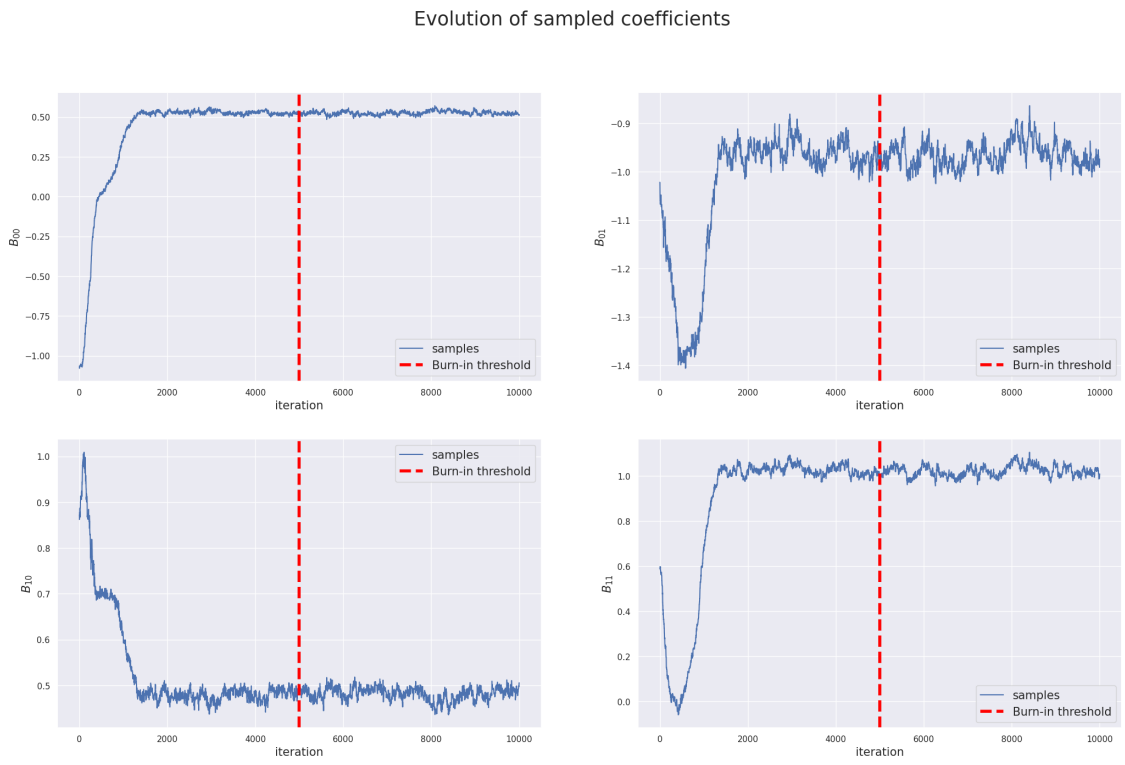
7.5.3 1.5.3. Plot sampled coefficients stochastic process - Markov Chain evolution

[38]:
```
# Plot sampled coefficients
fig, axs = plt.subplots(
    nrows=2, ncols=2,
    figsize=(25,15)
)

fig.suptitle(
    'Evolution of sampled coefficients',
    fontsize=25
)

# B_00
axs[0,0].plot(
    logs.iteration,
    samples[:, 0, 0],
    label='samples'
)
axs[0,0].axvline(
    estimator.burn_in_start,
    color='red',
    linestyle='--',
    linewidth=4,
    label='Burn-in threshold'
)
axs[0,0].set_xlabel(
    'iteration',
    fontsize=15
)
axs[0,0].set_ylabel(
    '$B_{00}$',
    fontsize=15
)
axs[0,0].legend(
    fontsize=15
)

# B_01
axs[0,1].plot(
    logs.iteration,
    samples[:, 0, 1],
    label='samples'
)
axs[0,1].axvline(
    estimator.burn_in_start,
    color='red',
```

```
    linestyle='--',
    linewidth=4,
    label='Burn-in threshold'
)
axs[0,1].set_xlabel(
    'iteration',
    fontsize=15
)
axs[0,1].set_ylabel(
    '$B_{01}$',
    fontsize=15
)
axs[0,1].legend(
    fontsize=15
)

# B_10
axs[1,0].plot(
    logs.iteration,
    samples[:, 1, 0],
    label='samples'
)
axs[1,0].axvline(
    estimator.burn_in_start,
    color='red',
    linestyle='--',
    linewidth=4,
    label='Burn-in threshold'
)
axs[1,0].set_xlabel(
    'iteration',
    fontsize=15
)
axs[1,0].set_ylabel(
    '$B_{10}$',
    fontsize=15
)
axs[1,0].legend(
    fontsize=15
)

# B_11
axs[1,1].plot(
    logs.iteration,
    samples[:, 1, 1],
    label='samples'
)
```

```
axs[1,1].axvline(
    estimator.burn_in_start,
    color='red',
    linestyle='--',
    linewidth=4,
    label='Burn-in threshold'
)
axs[1,1].set_xlabel(
    'iteration',
    fontsize=15
)
axs[1,1].set_ylabel(
    '$B_{11}$',
    fontsize=15
)
axs[1,1].legend(
    fontsize=15
)
```

[38]: <matplotlib.legend.Legend at 0x7f487a87c990>



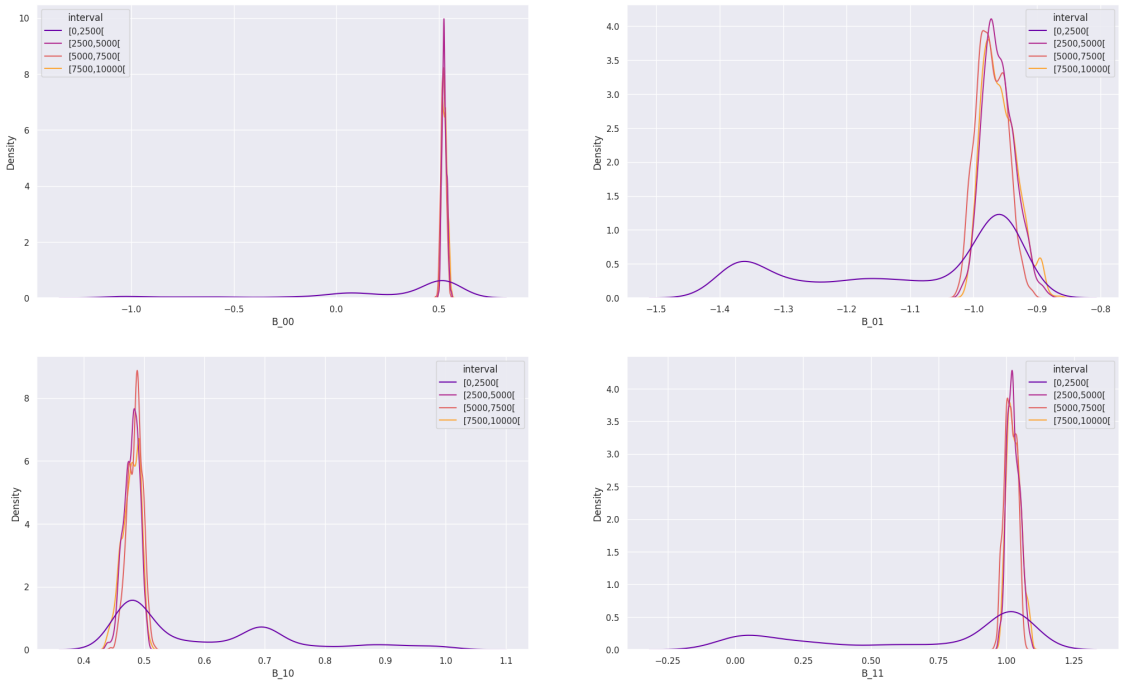
7.5.4 1.5.4. Plot sampled coefficients distributions - Markov Chain evolution

[39]:
```
##############################
# Evolution of distributions #
##############################

# Get step size and evaluated points
STEP_SIZE=2500
PALETTE='plasma'

i=0
evaluated_intervals=[]
while i<N_SAMPLES:
    start=i
    end=min(
        i+STEP_SIZE,
        N_SAMPLES
    )
    evaluated_intervals.append(
        (start, end)
    )
    i = i + STEP_SIZE

# Window samples and construct dataframe for plotting
plot_df=pd.DataFrame()
for start, end in evaluated_intervals:
    wdw_df = pd.DataFrame(
        data={
            'interval': ['[{},{}['.format(start, end)]*(end-start)
        }
    )
    wdw_samples = samples[start:end,:,:]
    for i, j in np.ndindex(B_est.shape):
        wdw_df['B_{}{}'.format(i, j)] = wdw_samples[:,i,j]

    plot_df = pd.concat(
        [
            plot_df,
            wdw_df
        ],
        axis=0
    ).reset_index(
        drop=True
    )

# Plot coefficient distribution evolution
fig, axs = plt.subplots(
```

```
    nrows=2, ncols=2,
    figsize=(25,15)
)

fig.suptitle(
    'Evolution of coefficient distributions',
    fontsize=25
)

# B_00
sns.kdeplot(
    data=plot_df,
    x='B_00',
    hue='interval',
    ax=axs[0,0],
    palette=PALETTE
)

# B_01
sns.kdeplot(
    data=plot_df,
    x='B_01',
    hue='interval',
    ax=axs[0,1],
    palette=PALETTE
)

# B_10
sns.kdeplot(
    data=plot_df,
    x='B_10',
    hue='interval',
    ax=axs[1,0],
    palette=PALETTE
)

# B_11
sns.kdeplot(
    data=plot_df,
    x='B_11',
    hue='interval',
    ax=axs[1,1],
    palette=PALETTE
)
```

[39]: <Axes: xlabel='B_11', ylabel='Density'>

Evolution of coefficient distributions



7.5.5 1.5.5. Plot Marginal Posteriors for Coefficients - Post-burn-in

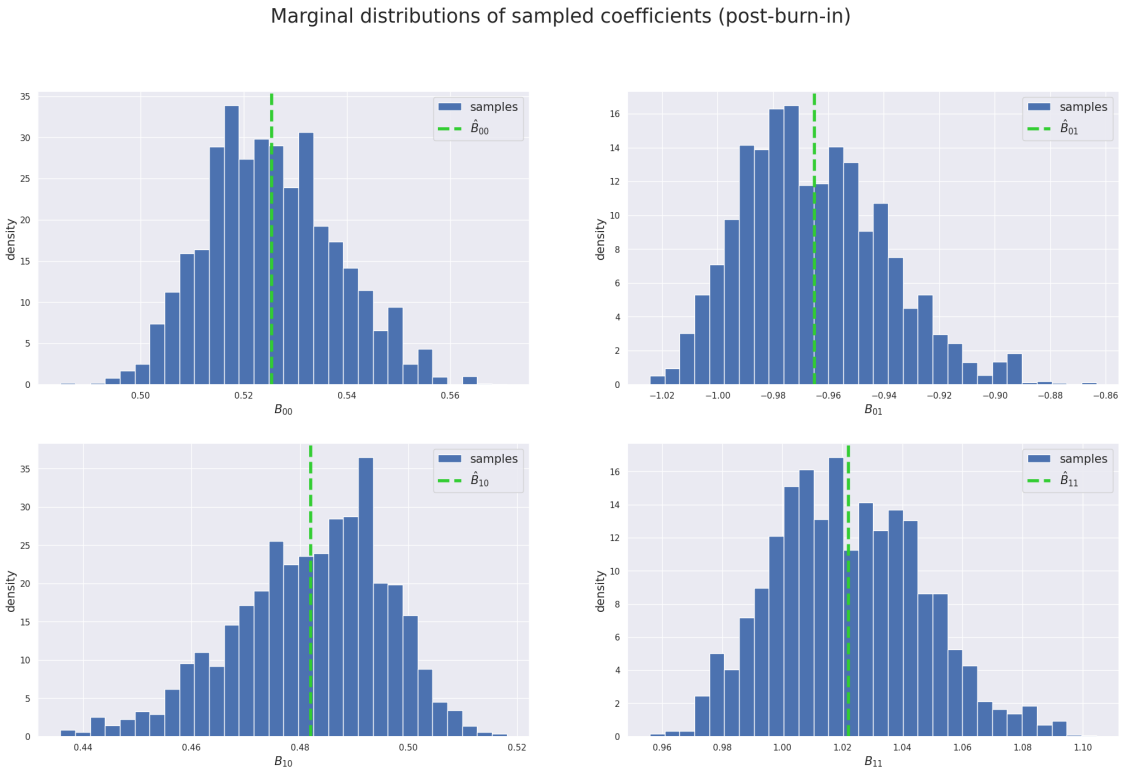
[40]:
```
# Plot sampled coefficients
fig, axs = plt.subplots(
    nrows=2, ncols=2,
    figsize=(25,15)
)

fig.suptitle(
    'Marginal distributions of sampled coefficients (post-burn-in)',
    fontsize=25
)

# B_00
axs[0,0].hist(
    valid_samples[:, 0, 0],
    density=True,
    bins=30,
    label='samples'
)
axs[0,0].axvline(
    B_est[0, 0],
```

```
    color='limegreen',
    linestyle='--',
    linewidth=4,
    label='$\hat{B}_{00}$'
)
axs[0,0].set_xlabel(
    '$B_{00}$',
    fontsize=15
)
axs[0,0].set_ylabel(
    'density',
    fontsize=15
)
axs[0,0].legend(
    loc='upper right',
    fontsize=15
)


# B_01
axs[0,1].hist(
    valid_samples[:, 0, 1],
    density=True,
    bins=30,
    label='samples'
)
axs[0,1].axvline(
    B_est[0, 1],
    color='limegreen',
    linestyle='--',
    linewidth=4,
    label='$\hat{B}_{01}$'
)
axs[0,1].set_xlabel(
    '$B_{01}$',
    fontsize=15
)
axs[0,1].set_ylabel(
    'density',
    fontsize=15
)
axs[0,1].legend(
    loc='upper right',
    fontsize=15
)

# B_10
```

```
axs[1,0].hist(
    valid_samples[:, 1, 0],
    density=True,
    bins=30,
    label='samples'
)
axs[1,0].axvline(
    B_est[1, 0],
    color='limegreen',
    linestyle='--',
    linewidth=4,
    label='$\hat{B}_{10}$'
)
axs[1,0].set_xlabel(
    '$B_{10}$',
    fontsize=15
)
axs[1,0].set_ylabel(
    'density',
    fontsize=15
)
axs[1,0].legend(
    loc='upper right',
    fontsize=15
)

# B_11
axs[1,1].hist(
    valid_samples[:, 1, 1],
    density=True,
    bins=30,
    label='samples'
)
axs[1,1].axvline(
    B_est[1, 1],
    color='limegreen',
    linestyle='--',
    linewidth=4,
    label='$\hat{B}_{11}$'
)
axs[1,1].set_xlabel(
    '$B_{11}$',
    fontsize=15
)
axs[1,1].set_ylabel(
    'density',
    fontsize=15
```

```
)
axs[1,1].legend(
    loc='upper right',
    fontsize=15
)
```

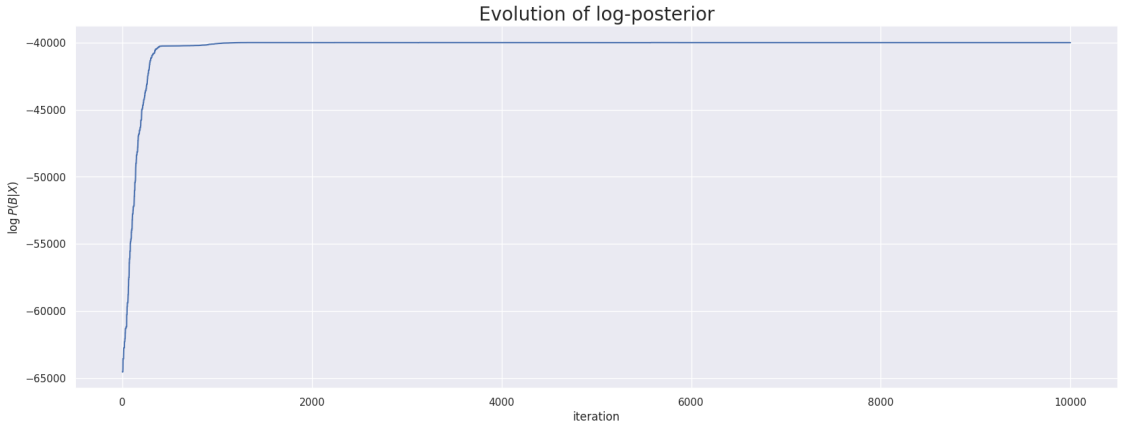
[40]: <matplotlib.legend.Legend at 0x7f487a0b3e10>



7.5.6 1.5.6. Plot evolution of log-posterior

[41]:
```
fig = plt.figure(figsize=(20,7))
sns.lineplot(
    data=logs,
    x='iteration',
    y='log_posterior'
)
plt.ylabel(
    '$\log P(B|X)$'
)
t = plt.title(
    'Evolution of log-posterior',
    fontsize=20
```

```
)
```



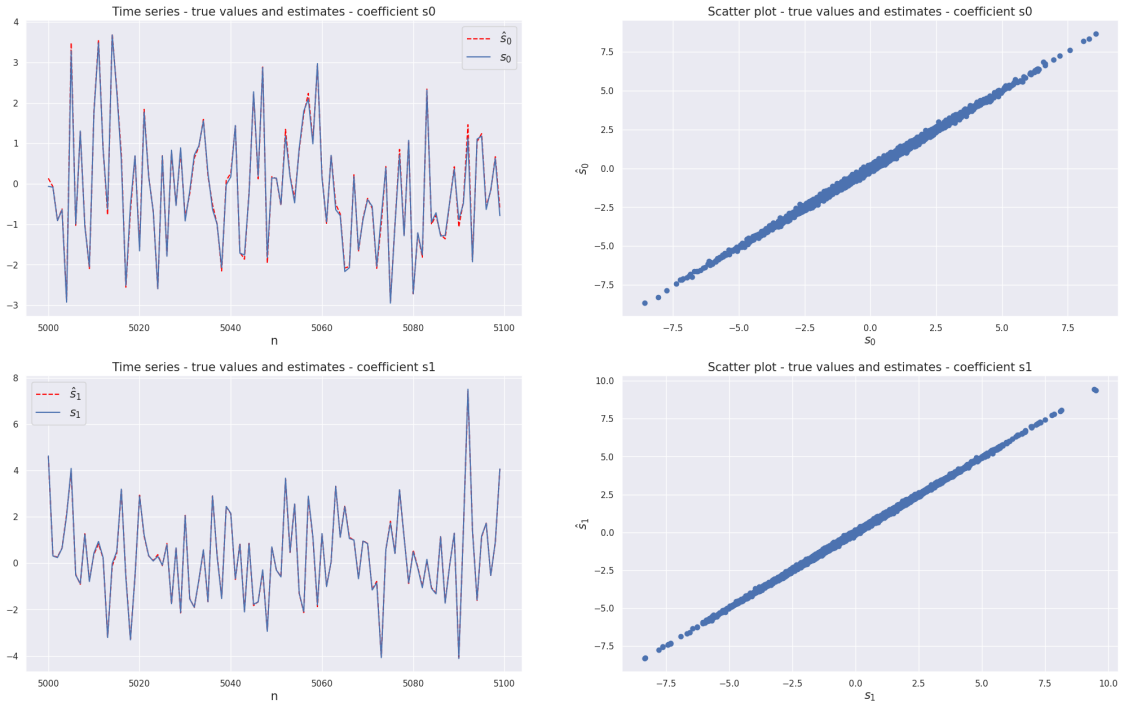
7.5.7 1.5.7. Plot Source Separation Results

[42]:
```
fig, axs = plt.subplots(
    nrows=2, ncols=2,
    figsize=(25,15)
)
NPOINTS=100
PLOT_START=NOBS//2
PLOT_END=PLOT_START+NPOINTS

t=range(NOBS)
s_est = B_est@x

# Axs 00
axs[0,0].plot(
    t[PLOT_START:PLOT_END],
    s_est[0,PLOT_START:PLOT_END],
    label='$\hat{s}_{0}$',
    color='red',
    linestyle='--'
)
axs[0,0].plot(
    t[PLOT_START:PLOT_END],
    s[0,PLOT_START:PLOT_END],
    label='$s_{0}$'
)
axs[0,0].set_xlabel(
    'n',
    fontsize=15
)
```

```
axs[0,0].set_title(
    'Time series - true values and estimates - coefficient s0',
    fontsize=15
)
axs[0,0].legend(fontsize=15)

# Axs 01
axs[0,1].scatter(
    s[0,:],
    s_est[0,:],
)
axs[0,1].set_xlabel(
    '$s_{0}$',
    fontsize=15
)
axs[0,1].set_ylabel(
    '$\hat{s}_{0}$',
    fontsize=15
)
axs[0,1].set_title(
    'Scatter plot - true values and estimates - coefficient s0',
    fontsize=15
)

# Axs 10
axs[1,0].plot(
    t[PLOT_START:PLOT_END],
    s_est[1,PLOT_START:PLOT_END],
    label='$\hat{s}_{1}$',
    color='red',
    linestyle='--'
)
axs[1,0].plot(
    t[PLOT_START:PLOT_END],
    s[1,PLOT_START:PLOT_END],
    label='$s_{1}$'
)
axs[1,0].set_xlabel(
    'n',
    fontsize=15
)
axs[1,0].set_title(
    'Time series - true values and estimates - coefficient s1',
    fontsize=15
)
axs[1,0].legend(fontsize=15)
```

```
# Axs 11
axs[1,1].scatter(
    s[1,:],
    s_est[1,:],
)
axs[1,1].set_xlabel(
    '$s_{1}$',
    fontsize=15
)
axs[1,1].set_ylabel(
    '$\hat{s}_{1}$',
    fontsize=15
)
axs[1,1].set_title(
    'Scatter plot - true values and estimates - coefficient s1',
    fontsize=15
)
```

[42]: Text(0.5, 1.0, 'Scatter plot - true values and estimates - coefficient s1')



[43]:
```
# 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: 0.00047823481320951455
```

8 2. SLIGHTLY MISSPECIFIED MODEL

8.1 2.1. Initialize Sources

[44]:
```
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)
    )
```

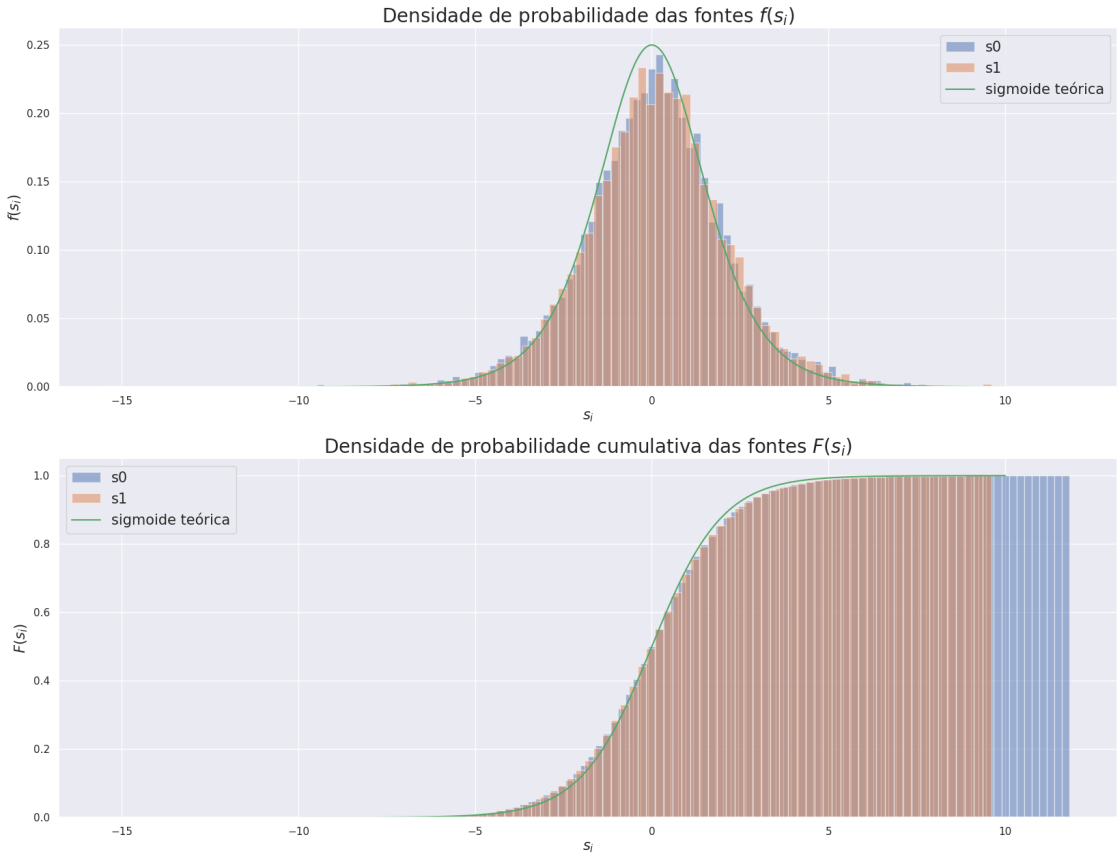
[45]:
```
MU=0.1
SCALE=1.1

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: 10000
CURTOSE s0: 1.1372995111879556
CURTOSE s1: 1.409004826288526
----------------------------------------------------------------------------------------------------
```

[46]:
```
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)
```



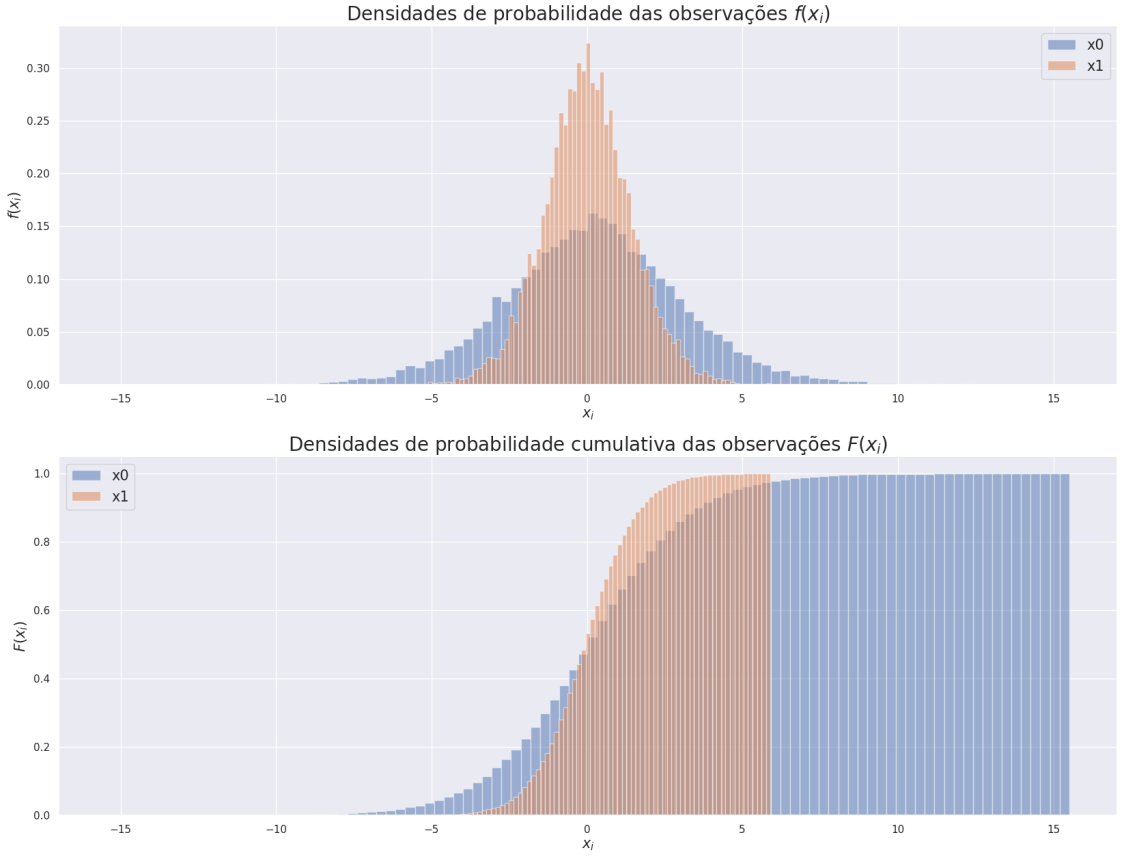
8.2 2.2. Mix sources and generate observations

[47]:
```
# 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]]
```

[48]:
```
# 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. Perform Analysis - LIKELIHOOD

8.3.1 2.3.1 Execute MCMC Sampling

[49]:
```
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
```

[50]:
```
%%time

SAVE_PATH='./artifacts/slightly_misspecified_model/ML.pkl'
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 open(SAVE_PATH, 'wb') as f:
        pickle.dump(estimator, f)

else:
    # Read artifact
    with open(SAVE_PATH, 'rb') as f:
        estimator=pickle.load(f)
```

```
CPU times: user 18.7 ms, sys: 19.3 ms, total: 38 ms
Wall time: 37 ms
```

8.3.2 2.3.2. Parse MCMC Results

[51]:
```
# Get results for analyzed model (best model by default)
# analyzed_model_idx = estimator.B_est_idx
analyzed_model_idx = 10
B_est = estimator.mcmc_results[analyzed_model_idx]['B_est']
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']
```

[52]:
```
print('-'*100)
print('Estimated B:\n{}'.format(B_est))
print('True B:\n{}'.format(np.linalg.inv(A)))
print('-'*100)
```

```
----------------------------------------------------------------------------------
--------------------
Estimated B:
[[ 0.46099008 -0.91320985]
 [ 0.45291944  0.89730503]]
True B:
[[ 0.5 -1. ]
 [ 0.5  1. ]]
```

```
--------------------------------------------------------------------------------
--------------------
```

[53]:
```
# Posteriors
[r['max_posterior'] for r in estimator.mcmc_results]
```

[53]:
```
[-41881.829192605335,
 -41881.845057690145,
 -41881.82963715858,
 -41881.827976049644,
 -41881.85474890447,
 -41881.85643535599,
 -41881.80918010041,
 -41881.83656423422,
 -41881.83305161601,
 -41881.848115063876,
 -41881.84880617575,
 -41881.82610334039,
 -41881.83482555457,
 -41881.81818643405,
 -41881.8168228029,
 -41881.84429849275,
 -41881.813942000794,
 -41881.85630720661,
 -41881.8020270848,
 -41881.836348193014,
 -41881.816063785605,
 -41881.82272870979,
 -41881.84167270034,
 -41881.80929636772,
 -41881.84204396548,
 -41881.81966360558,
 -41881.818163597345,
 -41881.82040382651,
 -41881.82962029725,
 -41881.80367177005]
```

[54]:
```
[r['B_est'] for r in estimator.mcmc_results]
```

[54]:
```
[array([[ 0.454514  ,  0.89429605],
        [-0.45926989,  0.91700742]]),
 array([[ 0.45530822,  0.89221675],
        [-0.45826992,  0.91825193]]),
 array([[-0.46349626,  0.90800927],
        [-0.45018332, -0.90285083]]),
 array([[-0.46229384,  0.91094563],
        [-0.45143924, -0.90007634]]),
```

```
array([[-0.45048108, -0.90387914],
       [ 0.46322317, -0.90819752]]),
array([[ 0.46045823, -0.91419565],
       [-0.45264875, -0.89690265]]),
array([[ 0.452101  ,  0.898925  ],
       [-0.46125197,  0.91207149]]),
array([[-0.44951498, -0.90622725],
       [ 0.46454222, -0.9044776 ]]),
array([[-0.46212201,  0.91179206],
       [-0.45185399, -0.90026292]]),
array([[ 0.46094168, -0.91403222],
       [ 0.45318015,  0.89807777]]),
array([[ 0.46099008, -0.91320985],
       [ 0.45291944,  0.89730503]]),
array([[ 0.46247522, -0.91176768],
       [ 0.45158625,  0.90143764]]),
array([[-0.46398724,  0.90662303],
       [ 0.44982529,  0.90402058]]),
array([[-0.46285817,  0.90916142],
       [ 0.45097968,  0.90204864]]),
array([[-0.45186997, -0.8997976 ],
       [ 0.46182765, -0.9107711 ]]),
array([[ 0.45127802,  0.90090578],
       [-0.46238551,  0.91127511]]),
array([[ 0.46025414, -0.91443875],
       [-0.45351586, -0.89656009]]),
array([[-0.46446328,  0.90650831],
       [ 0.44967362,  0.90477972]]),
array([[-0.46174461,  0.91199157],
       [ 0.45208372,  0.89918124]]),
array([[-0.45025164, -0.90349511],
       [-0.46389631,  0.90702945]]),
array([[-0.46188983,  0.91245783],
       [ 0.45204227,  0.89910305]]),
array([[-0.46318062,  0.9094643 ],
       [ 0.45065523,  0.90245979]]),
array([[-0.46151424,  0.91271958],
       [ 0.45183424,  0.89891032]]),
array([[ 0.45286912,  0.89752833],
       [ 0.46060025, -0.91399775]]),
array([[ 0.45164494,  0.90118742],
       [-0.46231672,  0.91011242]]),
array([[-0.46107523,  0.9131045 ],
       [ 0.4525512 ,  0.89804346]]),
array([[-0.45832891,  0.91853344],
       [ 0.45546643,  0.89232555]]),
array([[-0.46041418,  0.9153477 ],
```

```
        [ 0.45354703,  0.89635502]]),
 array([[ 0.4493767 ,  0.90579429],
        [-0.46429221,  0.90573016]]),
 array([[-0.4645457 ,  0.90708629],
        [-0.44946797, -0.90522897]])]
```

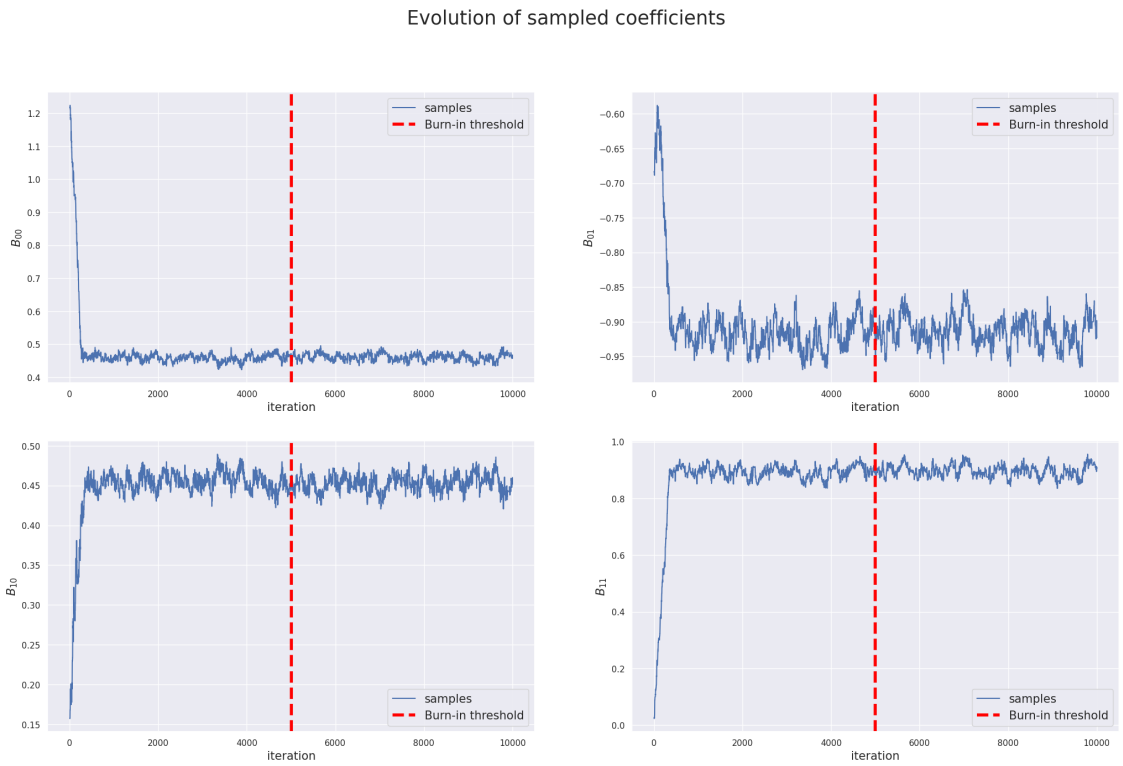
8.3.3 2.3.3. Plot sampled coefficients stochastic process - Markov Chain evolution

[55]:
```
# Plot sampled coefficients
fig, axs = plt.subplots(
    nrows=2, ncols=2,
    figsize=(25,15)
)

fig.suptitle(
    'Evolution of sampled coefficients',
    fontsize=25
)

# B_00
axs[0,0].plot(
    logs.iteration,
    samples[:, 0, 0],
    label='samples'
)
axs[0,0].axvline(
    estimator.burn_in_start,
    color='red',
    linestyle='--',
    linewidth=4,
    label='Burn-in threshold'
)
axs[0,0].set_xlabel(
    'iteration',
    fontsize=15
)
axs[0,0].set_ylabel(
    '$B_{00}$',
    fontsize=15
)
axs[0,0].legend(
    fontsize=15
)

# B_01
axs[0,1].plot(
    logs.iteration,
```

```
    samples[:, 0, 1],
    label='samples'
)
axs[0,1].axvline(
    estimator.burn_in_start,
    color='red',
    linestyle='--',
    linewidth=4,
    label='Burn-in threshold'
)
axs[0,1].set_xlabel(
    'iteration',
    fontsize=15
)
axs[0,1].set_ylabel(
    '$B_{01}$',
    fontsize=15
)
axs[0,1].legend(
    fontsize=15
)

# B_10
axs[1,0].plot(
    logs.iteration,
    samples[:, 1, 0],
    label='samples'
)
axs[1,0].axvline(
    estimator.burn_in_start,
    color='red',
    linestyle='--',
    linewidth=4,
    label='Burn-in threshold'
)
axs[1,0].set_xlabel(
    'iteration',
    fontsize=15
)
axs[1,0].set_ylabel(
    '$B_{10}$',
    fontsize=15
)
axs[1,0].legend(
    fontsize=15
)
```

```
# B_11
axs[1,1].plot(
    logs.iteration,
    samples[:, 1, 1],
    label='samples'
)
axs[1,1].axvline(
    estimator.burn_in_start,
    color='red',
    linestyle='--',
    linewidth=4,
    label='Burn-in threshold'
)
axs[1,1].set_xlabel(
    'iteration',
    fontsize=15
)
axs[1,1].set_ylabel(
    '$B_{11}$',
    fontsize=15
)
axs[1,1].legend(
    fontsize=15
)
```

[55]: <matplotlib.legend.Legend at 0x7f487142f3d0>



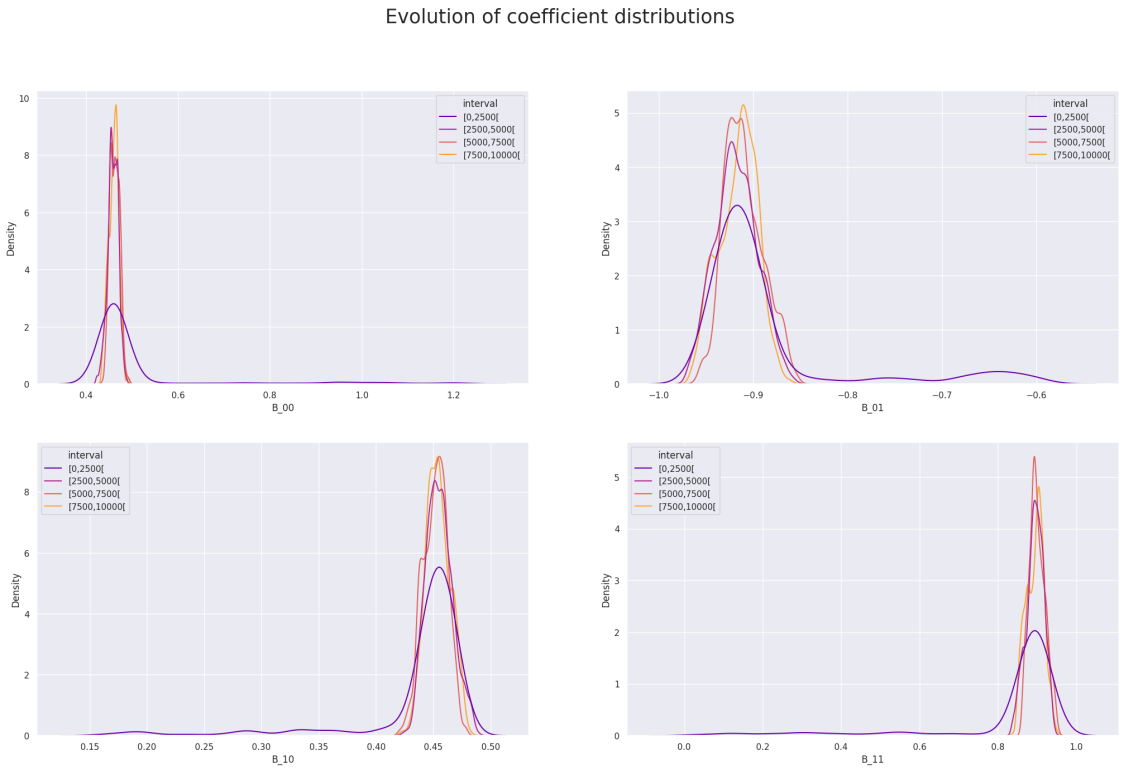
8.3.4 2.3.4. Plot sampled coefficients distributions - Markov Chain evolution

[56]:
```
################################
# Evolution of distributions #
################################

# Get step size and evaluated points
STEP_SIZE=2500
PALETTE='plasma'

i=0
evaluated_intervals=[]
while i<N_SAMPLES:
    start=i
    end=min(
        i+STEP_SIZE,
        N_SAMPLES
    )
    evaluated_intervals.append(
        (start, end)
    )
    i = i + STEP_SIZE
```

```
# Window samples and construct dataframe for plotting
plot_df=pd.DataFrame()
for start, end in evaluated_intervals:
    wdw_df = pd.DataFrame(
        data={
            'interval': ['[{},{}['.format(start, end)]*(end-start)
        }
    )
    wdw_samples = samples[start:end,:,:]
    for i, j in np.ndindex(B_est.shape):
        wdw_df['B_{}{}'.format(i, j)] = wdw_samples[:,i,j]

    plot_df = pd.concat(
        [
            plot_df,
            wdw_df
        ],
        axis=0
    ).reset_index(
        drop=True
    )

# Plot coefficient distribution evolution
fig, axs = plt.subplots(
    nrows=2, ncols=2,
    figsize=(25,15)
)

fig.suptitle(
    'Evolution of coefficient distributions',
    fontsize=25
)

# B_00
sns.kdeplot(
    data=plot_df,
    x='B_00',
    hue='interval',
    ax=axs[0,0],
    palette=PALETTE
)

# B_01
sns.kdeplot(
    data=plot_df,
    x='B_01',
    hue='interval',
```

```
    ax=axs[0,1],
    palette=PALETTE
)

# B_10
sns.kdeplot(
    data=plot_df,
    x='B_10',
    hue='interval',
    ax=axs[1,0],
    palette=PALETTE
)

# B_11
sns.kdeplot(
    data=plot_df,
    x='B_11',
    hue='interval',
    ax=axs[1,1],
    palette=PALETTE
)
```

[56]: <Axes: xlabel='B_11', ylabel='Density'>



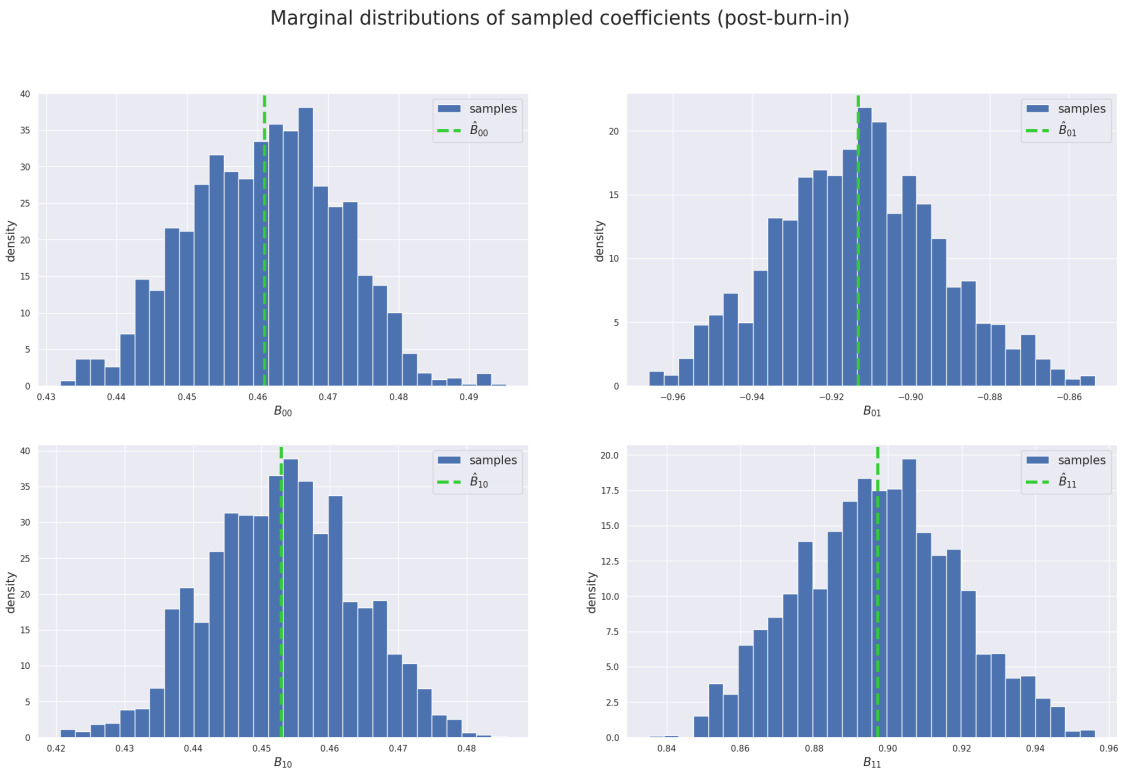
8.3.5 2.3.5. Plot Marginal Posteriors for Coefficients - Post-burn-in

[57]:
```
# Plot sampled coefficients
fig, axs = plt.subplots(
    nrows=2, ncols=2,
    figsize=(25,15)
)

fig.suptitle(
    'Marginal distributions of sampled coefficients (post-burn-in)',
    fontsize=25
)

# B_00
axs[0,0].hist(
    valid_samples[:, 0, 0],
    density=True,
    bins=30,
    label='samples'
)
axs[0,0].axvline(
    B_est[0, 0],
    color='limegreen',
    linestyle='--',
    linewidth=4,
    label='$\hat{B}_{00}$'
)
axs[0,0].set_xlabel(
    '$B_{00}$',
    fontsize=15
)
axs[0,0].set_ylabel(
    'density',
    fontsize=15
)
axs[0,0].legend(
    loc='upper right',
    fontsize=15
)


# B_01
axs[0,1].hist(
    valid_samples[:, 0, 1],
    density=True,
    bins=30,
    label='samples'
```

```
)
axs[0,1].axvline(
    B_est[0, 1],
    color='limegreen',
    linestyle='--',
    linewidth=4,
    label='$\hat{B}_{01}$'
)
axs[0,1].set_xlabel(
    '$B_{01}$',
    fontsize=15
)
axs[0,1].set_ylabel(
    'density',
    fontsize=15
)
axs[0,1].legend(
    loc='upper right',
    fontsize=15
)

# B_10
axs[1,0].hist(
    valid_samples[:, 1, 0],
    density=True,
    bins=30,
    label='samples'
)
axs[1,0].axvline(
    B_est[1, 0],
    color='limegreen',
    linestyle='--',
    linewidth=4,
    label='$\hat{B}_{10}$'
)
axs[1,0].set_xlabel(
    '$B_{10}$',
    fontsize=15
)
axs[1,0].set_ylabel(
    'density',
    fontsize=15
)
axs[1,0].legend(
    loc='upper right',
    fontsize=15
)
```

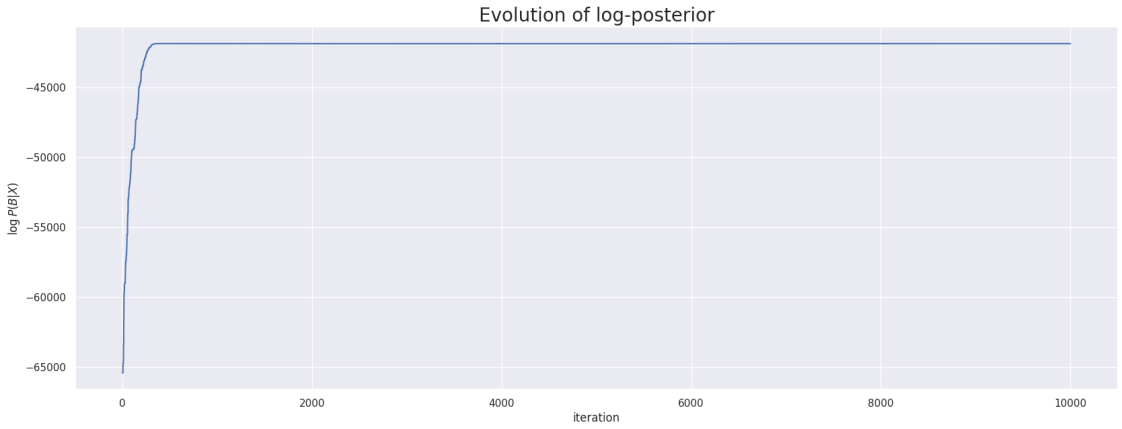
```
# B_11
axs[1,1].hist(
    valid_samples[:, 1, 1],
    density=True,
    bins=30,
    label='samples'
)
axs[1,1].axvline(
    B_est[1, 1],
    color='limegreen',
    linestyle='--',
    linewidth=4,
    label='$\hat{B}_{11}$'
)
axs[1,1].set_xlabel(
    '$B_{11}$',
    fontsize=15
)
axs[1,1].set_ylabel(
    'density',
    fontsize=15
)
axs[1,1].legend(
    loc='upper right',
    fontsize=15
)
```

[57]: <matplotlib.legend.Legend at 0x7f4873cf5f10>



8.3.6 2.3.6. Plot evolution of log-posterior

[58]:
```
fig = plt.figure(figsize=(20,7))
sns.lineplot(
    data=logs,
    x='iteration',
    y='log_posterior'
)
plt.ylabel(
    '$\log P(B|X)$'
)
t = plt.title(
    'Evolution of log-posterior',
    fontsize=20
)
```

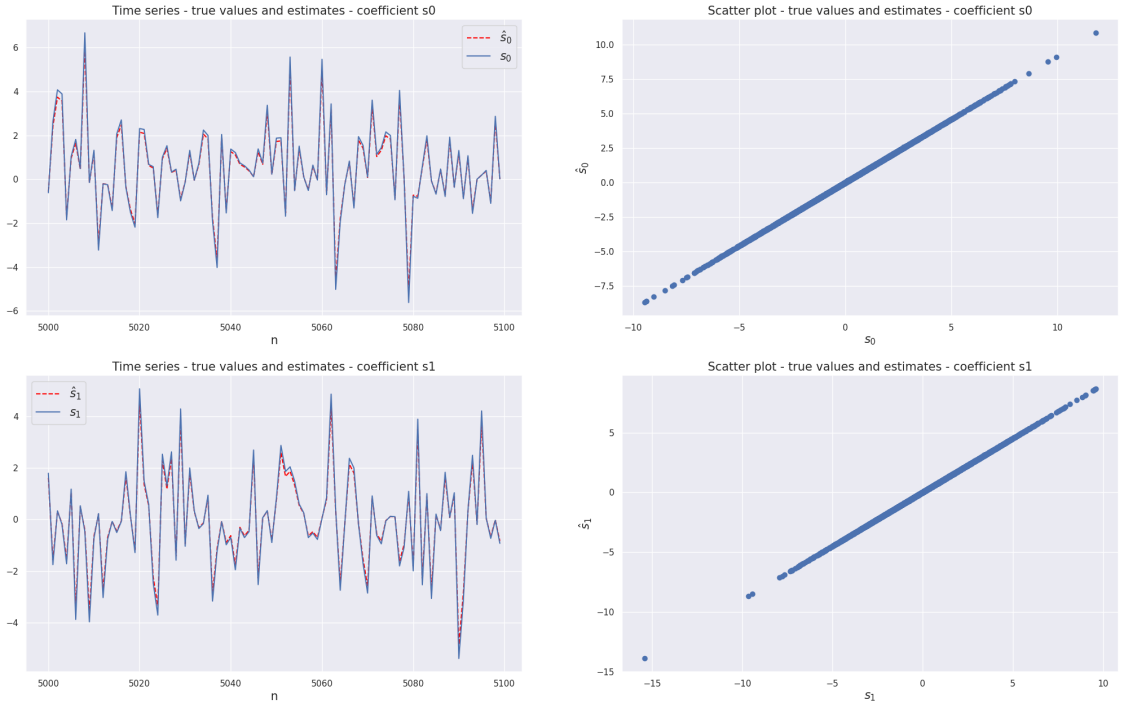


[59]:
```
fig, axs = plt.subplots(
    nrows=2, ncols=2,
    figsize=(25,15)
)
NPOINTS=100
PLOT_START=NOBS//2
PLOT_END=PLOT_START+NPOINTS

t=range(NOBS)
s_est = B_est@x

# Axs 00
axs[0,0].plot(
    t[PLOT_START:PLOT_END],
    s_est[0,PLOT_START:PLOT_END],
    label='$\hat{s}_{0}$',
    color='red',
    linestyle='--'
)
axs[0,0].plot(
    t[PLOT_START:PLOT_END],
    s[0,PLOT_START:PLOT_END],
    label='$s_{0}$'
)
axs[0,0].set_xlabel(
    'n',
    fontsize=15
)
axs[0,0].set_title(
    'Time series - true values and estimates - coefficient s0',
    fontsize=15
)
```

```
axs[0,0].legend(fontsize=15)

# Axs 01
axs[0,1].scatter(
    s[0,:],
    s_est[0,:],
)
axs[0,1].set_xlabel(
    '$s_{0}$',
    fontsize=15
)
axs[0,1].set_ylabel(
    '$\hat{s}_{0}$',
    fontsize=15
)
axs[0,1].set_title(
    'Scatter plot - true values and estimates - coefficient s0',
    fontsize=15
)

# Axs 10
axs[1,0].plot(
    t[PLOT_START:PLOT_END],
    s_est[1,PLOT_START:PLOT_END],
    label='$\hat{s}_{1}$',
    color='red',
    linestyle='--'
)
axs[1,0].plot(
    t[PLOT_START:PLOT_END],
    s[1,PLOT_START:PLOT_END],
    label='$s_{1}$'
)
axs[1,0].set_xlabel(
    'n',
    fontsize=15
)
axs[1,0].set_title(
    'Time series - true values and estimates - coefficient s1',
    fontsize=15
)
axs[1,0].legend(fontsize=15)

# Axs 11
axs[1,1].scatter(
    s[1,:],
    s_est[1,:],
```

```
)
axs[1,1].set_xlabel(
    '$s_{1}$',
    fontsize=15
)
axs[1,1].set_ylabel(
    '$\hat{s}_{1}$',
    fontsize=15
)
axs[1,1].set_title(
    'Scatter plot - true values and estimates - coefficient s1',
    fontsize=15
)
```

[59]: Text(0.5, 1.0, 'Scatter plot - true values and estimates - coefficient s1')



[60]:
```
# Error norm
err=np.linalg.norm(
    np.subtract(s,s_est)
)/np.size(s)

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

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

```
Norma do erro de estimação: 0.0012914937731572473
```

8.4 2.4. Perform Analysis - DETERMINANT PRIOR

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

8.4.1 2.4.1 Execute MCMC Sampling

[61]:
```
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
```

[62]:
```
%%time

SAVE_PATH='./artifacts/slightly_misspecified_model/Det_1.pkl'
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 open(SAVE_PATH, 'wb') as f:
        pickle.dump(estimator, f)

else:
    # Read artifact
```

```
    with open(SAVE_PATH, 'rb') as f:
        estimator=pickle.load(f)
```

```
CPU times: user 78.7 ms, sys: 109 ms, total: 188 ms
Wall time: 23.6 ms
```

8.4.2 2.4.2. Parse MCMC Results

[63]:
```
# Get results for analyzed model (best model by default)
# analyzed_model_idx = estimator.B_est_idx
analyzed_model_idx = 10
B_est = estimator.mcmc_results[analyzed_model_idx]['B_est']
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']
```

[64]:
```
print('-'*100)
print('Estimated B:\n{}'.format(B_est))
print('True B:\n{}'.format(np.linalg.inv(A)))
print('-'*100)
```

```
----------------------------------------------------------------------------------------
--------------------
Estimated B:
[[ 0.46249789 -0.91085758]
 [ 0.45125366  0.90231344]]
True B:
[[ 0.5 -1. ]
 [ 0.5  1. ]]
----------------------------------------------------------------------------------------
--------------------
```

[65]:
```
# Posteriors
[r['max_posterior'] for r in estimator.mcmc_results]
```

[65]:
```
[-41881.82648294251,
 -41881.84357256397,
 -41881.84017897975,
 -41881.81995983159,
 -41881.81499834952,
 -41881.84607366815,
 -41881.831736898006,
 -41881.82019164272,
 -41881.8158869228,
 -41881.82504773269,
 -41881.80866734508,
```

```
 -41881.848635927556,
 -41881.8265472787,
 -41881.8042340089,
 -41881.82541209858,
 -41881.85033894496,
 -41881.81781075382,
 -41881.83549335458,
 -41881.80383136671,
 -41881.808024104306,
 -41881.82185895512,
 -41881.83640515857,
 -41881.827239649596,
 -41881.84001742282,
 -41881.82675656568,
 -41881.81948057642,
 -41881.83048992674,
 -41881.80435416372,
 -41881.84337511899,
 -41881.81310856113]
```

[66]:
```
[r['B_est'] for r in estimator.mcmc_results]
```

[66]:
```
[array([[ 0.46195056, -0.91039276],
        [ 0.45186088,  0.9008521 ]]),
 array([[ 0.44881211,  0.90694395],
        [-0.4647487 ,  0.9048323 ]]),
 array([[-0.46195361,  0.91261087],
        [-0.45188924, -0.90019145]]),
 array([[-0.46151092,  0.91329021],
        [-0.45211276, -0.89942545]]),
 array([[-0.45318326, -0.89649799],
        [ 0.46021756, -0.91418083]]),
 array([[-0.44971972, -0.90504733],
        [-0.46425876,  0.90635637]]),
 array([[ 0.45279023,  0.89866876],
        [-0.4614143 ,  0.91346837]]),
 array([[-0.45435196, -0.89542426],
        [ 0.45953042, -0.91630728]]),
 array([[-0.46149048,  0.91271424],
        [-0.4526202 , -0.89788599]]),
 array([[-0.45021092, -0.90366795],
        [-0.46366544,  0.90846289]]),
 array([[ 0.46249789, -0.91085758],
        [ 0.45125366,  0.90231344]]),
 array([[ 0.46457464, -0.90517731],
        [ 0.44913128,  0.90637421]]),
 array([[-0.46499715,  0.90640771],
```

```
       [ 0.44894905,  0.90634651]]),
 array([[-0.46330135,  0.9090556 ],
       [ 0.45040073,  0.9026958 ]]),
 array([[-0.4516012 , -0.90124524],
       [ 0.46237104, -0.91010312]]),
 array([[ 0.45024271,  0.90237628],
       [-0.4630698 ,  0.90961955]]),
 array([[ 0.46027198, -0.91472425],
       [-0.45366803, -0.89744084]]),
 array([[-0.46369993,  0.90853245],
       [ 0.45015781,  0.90360136]]),
 array([[-0.46457897,  0.90643709],
       [ 0.44931653,  0.90576681]]),
 array([[-0.44938214, -0.90563686],
       [-0.46431107,  0.90570032]]),
 array([[ 0.45408006,  0.89465958],
       [ 0.45965113, -0.91564937]]),
 array([[-0.46484615,  0.90605829],
       [ 0.44905104,  0.9057876 ]]),
 array([[-0.44960427, -0.90475283],
       [-0.46380271,  0.90650987]]),
 array([[ 0.45469732,  0.89485869],
       [ 0.45917495, -0.91674558]]),
 array([[ 0.45094233,  0.90179464],
       [-0.46293965,  0.90908187]]),
 array([[-0.46440898,  0.90673328],
       [ 0.44960704,  0.90484243]]),
 array([[-0.46386916,  0.9081661 ],
       [ 0.44991839,  0.90325949]]),
 array([[-0.46363197,  0.90712471],
       [ 0.44992001,  0.90437135]]),
 array([[ 0.44922573,  0.90518059],
       [-0.46423595,  0.90645181]]),
 array([[-0.46205848,  0.9099924 ],
       [-0.45149559, -0.90065376]])]
```

8.4.3 2.4.3. Plot sampled coefficients stochastic process - Markov Chain evolution

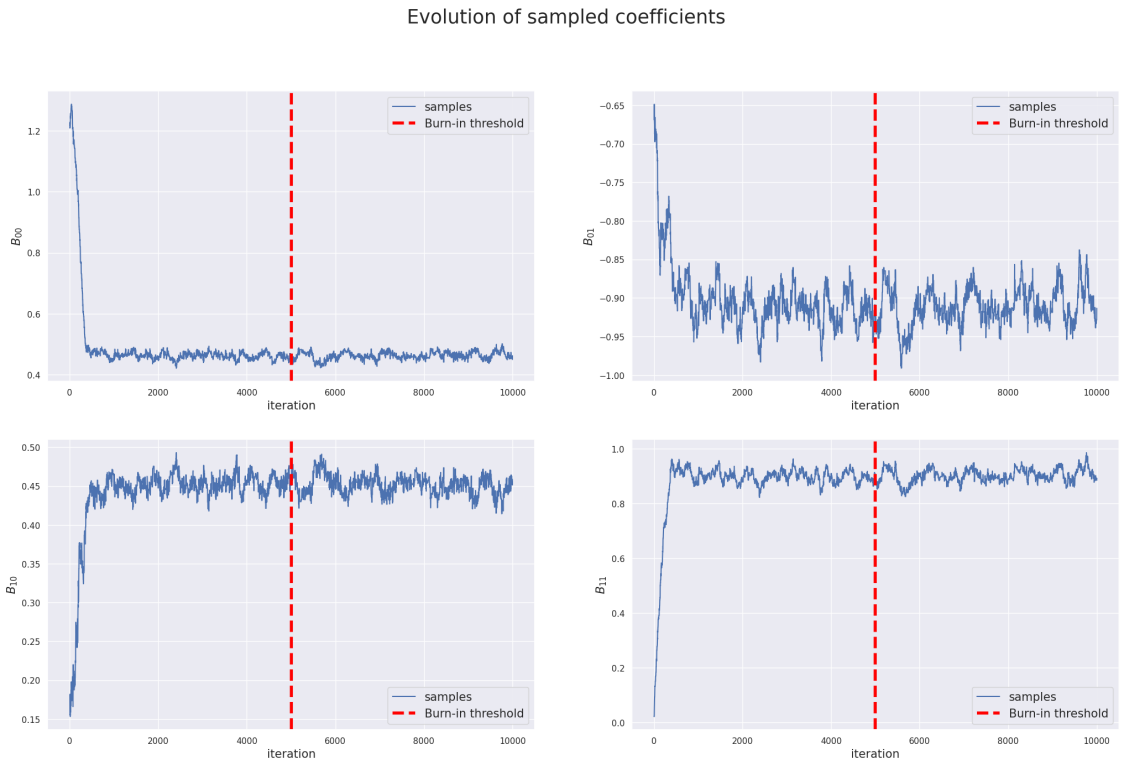
[67]:
```
# Plot sampled coefficients
fig, axs = plt.subplots(
    nrows=2, ncols=2,
    figsize=(25,15)
)

fig.suptitle(
    'Evolution of sampled coefficients',
    fontsize=25
```

```
)

# B_00
axs[0,0].plot(
    logs.iteration,
    samples[:, 0, 0],
    label='samples'
)
axs[0,0].axvline(
    estimator.burn_in_start,
    color='red',
    linestyle='--',
    linewidth=4,
    label='Burn-in threshold'
)
axs[0,0].set_xlabel(
    'iteration',
    fontsize=15
)
axs[0,0].set_ylabel(
    '$B_{00}$',
    fontsize=15
)
axs[0,0].legend(
    fontsize=15
)

# B_01
axs[0,1].plot(
    logs.iteration,
    samples[:, 0, 1],
    label='samples'
)
axs[0,1].axvline(
    estimator.burn_in_start,
    color='red',
    linestyle='--',
    linewidth=4,
    label='Burn-in threshold'
)
axs[0,1].set_xlabel(
    'iteration',
    fontsize=15
)
axs[0,1].set_ylabel(
    '$B_{01}$',
    fontsize=15
```

```
)
axs[0,1].legend(
    fontsize=15
)

# B_10
axs[1,0].plot(
    logs.iteration,
    samples[:, 1, 0],
    label='samples'
)
axs[1,0].axvline(
    estimator.burn_in_start,
    color='red',
    linestyle='--',
    linewidth=4,
    label='Burn-in threshold'
)
axs[1,0].set_xlabel(
    'iteration',
    fontsize=15
)
axs[1,0].set_ylabel(
    '$B_{10}$',
    fontsize=15
)
axs[1,0].legend(
    fontsize=15
)

# B_11
axs[1,1].plot(
    logs.iteration,
    samples[:, 1, 1],
    label='samples'
)
axs[1,1].axvline(
    estimator.burn_in_start,
    color='red',
    linestyle='--',
    linewidth=4,
    label='Burn-in threshold'
)
axs[1,1].set_xlabel(
    'iteration',
    fontsize=15
)
```

```
axs[1,1].set_ylabel(
    '$B_{11}$',
    fontsize=15
)
axs[1,1].legend(
    fontsize=15
)
```

[67]: <matplotlib.legend.Legend at 0x7f48784b2810>



8.4.4 2.4.4. Plot sampled coefficients distributions - Markov Chain evolution

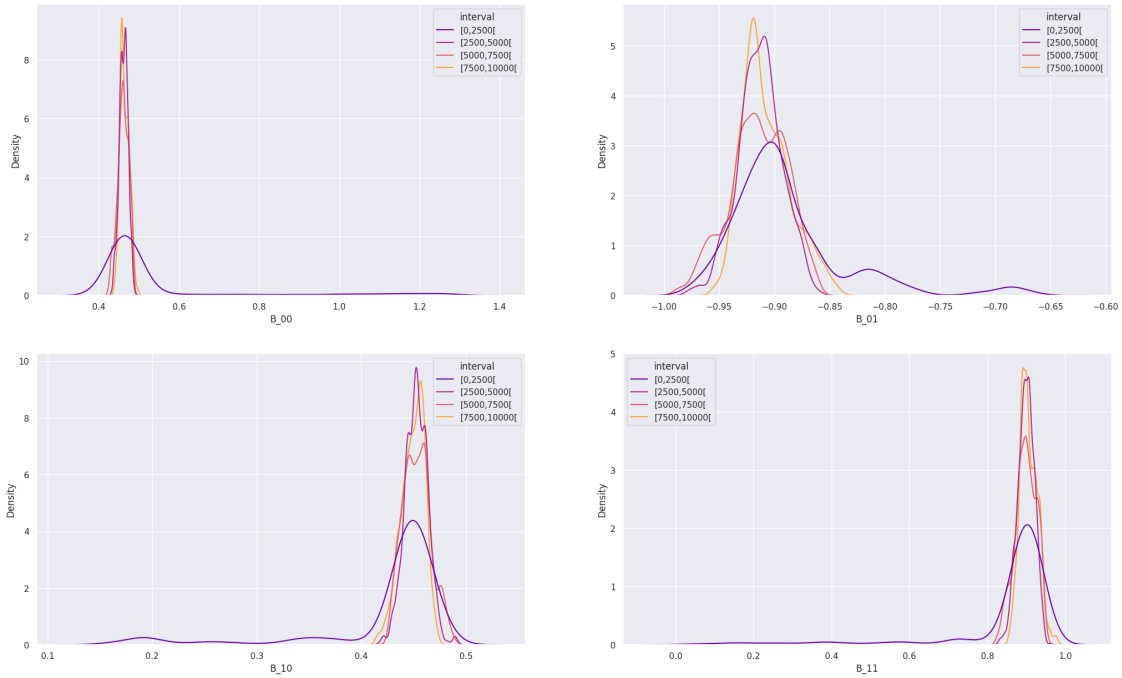
[68]:
```
################################
# Evolution of distributions #
################################

# Get step size and evaluated points
STEP_SIZE=2500
PALETTE='plasma'

i=0
evaluated_intervals=[]
while i<N_SAMPLES:
```

```
    start=i
    end=min(
        i+STEP_SIZE,
        N_SAMPLES
    )
    evaluated_intervals.append(
        (start, end)
    )
    i = i + STEP_SIZE

# Window samples and construct dataframe for plotting
plot_df=pd.DataFrame()
for start, end in evaluated_intervals:
    wdw_df = pd.DataFrame(
        data={
            'interval': ['[{},{}['.format(start, end)]*(end-start)
        }
    )
    wdw_samples = samples[start:end,:,:]
    for i, j in np.ndindex(B_est.shape):
        wdw_df['B_{}{}'.format(i, j)] = wdw_samples[:,i,j]

    plot_df = pd.concat(
        [
            plot_df,
            wdw_df
        ],
        axis=0
    ).reset_index(
        drop=True
    )

# Plot coefficient distribution evolution
fig, axs = plt.subplots(
    nrows=2, ncols=2,
    figsize=(25,15)
)

fig.suptitle(
    'Evolution of coefficient distributions',
    fontsize=25
)

# B_00
sns.kdeplot(
    data=plot_df,
    x='B_00',
```

```
    hue='interval',
    ax=axs[0,0],
    palette=PALETTE
)

# B_01
sns.kdeplot(
    data=plot_df,
    x='B_01',
    hue='interval',
    ax=axs[0,1],
    palette=PALETTE
)

# B_10
sns.kdeplot(
    data=plot_df,
    x='B_10',
    hue='interval',
    ax=axs[1,0],
    palette=PALETTE
)

# B_11
sns.kdeplot(
    data=plot_df,
    x='B_11',
    hue='interval',
    ax=axs[1,1],
    palette=PALETTE
)
```

[68]: <Axes: xlabel='B_11', ylabel='Density'>

Evolution of coefficient distributions



8.4.5 2.4.5. Plot Marginal Posteriors for Coefficients - Post-burn-in

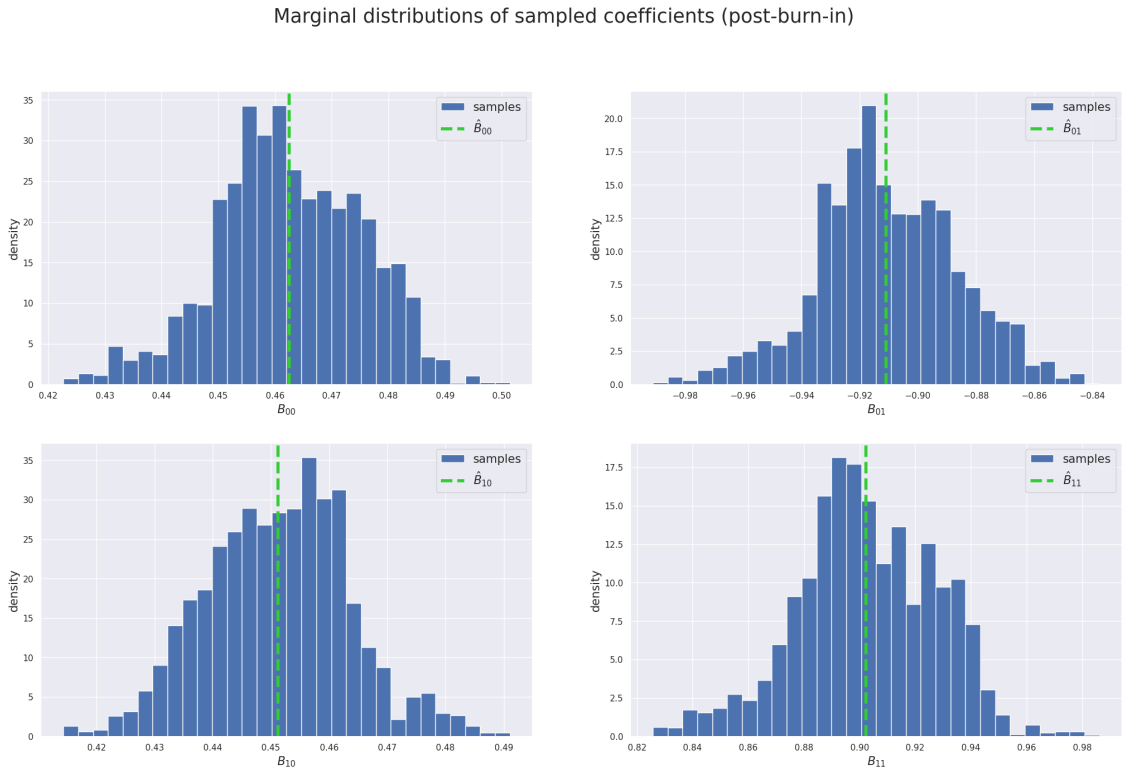
[69]:
```
# Plot sampled coefficients
fig, axs = plt.subplots(
    nrows=2, ncols=2,
    figsize=(25,15)
)

fig.suptitle(
    'Marginal distributions of sampled coefficients (post-burn-in)',
    fontsize=25
)

# B_00
axs[0,0].hist(
    valid_samples[:, 0, 0],
    density=True,
    bins=30,
    label='samples'
)
axs[0,0].axvline(
    B_est[0, 0],
    color='limegreen',
```

```
    linestyle='--',
    linewidth=4,
    label='$\hat{B}_{00}$'
)
axs[0,0].set_xlabel(
    '$B_{00}$',
    fontsize=15
)
axs[0,0].set_ylabel(
    'density',
    fontsize=15
)
axs[0,0].legend(
    loc='upper right',
    fontsize=15
)


# B_01
axs[0,1].hist(
    valid_samples[:, 0, 1],
    density=True,
    bins=30,
    label='samples'
)
axs[0,1].axvline(
    B_est[0, 1],
    color='limegreen',
    linestyle='--',
    linewidth=4,
    label='$\hat{B}_{01}$'
)
axs[0,1].set_xlabel(
    '$B_{01}$',
    fontsize=15
)
axs[0,1].set_ylabel(
    'density',
    fontsize=15
)
axs[0,1].legend(
    loc='upper right',
    fontsize=15
)

# B_10
axs[1,0].hist(
```

```
    valid_samples[:, 1, 0],
    density=True,
    bins=30,
    label='samples'
)
axs[1,0].axvline(
    B_est[1, 0],
    color='limegreen',
    linestyle='--',
    linewidth=4,
    label='$\hat{B}_{10}$'
)
axs[1,0].set_xlabel(
    '$B_{10}$',
    fontsize=15
)
axs[1,0].set_ylabel(
    'density',
    fontsize=15
)
axs[1,0].legend(
    loc='upper right',
    fontsize=15
)

# B_11
axs[1,1].hist(
    valid_samples[:, 1, 1],
    density=True,
    bins=30,
    label='samples'
)
axs[1,1].axvline(
    B_est[1, 1],
    color='limegreen',
    linestyle='--',
    linewidth=4,
    label='$\hat{B}_{11}$'
)
axs[1,1].set_xlabel(
    '$B_{11}$',
    fontsize=15
)
axs[1,1].set_ylabel(
    'density',
    fontsize=15
)
```

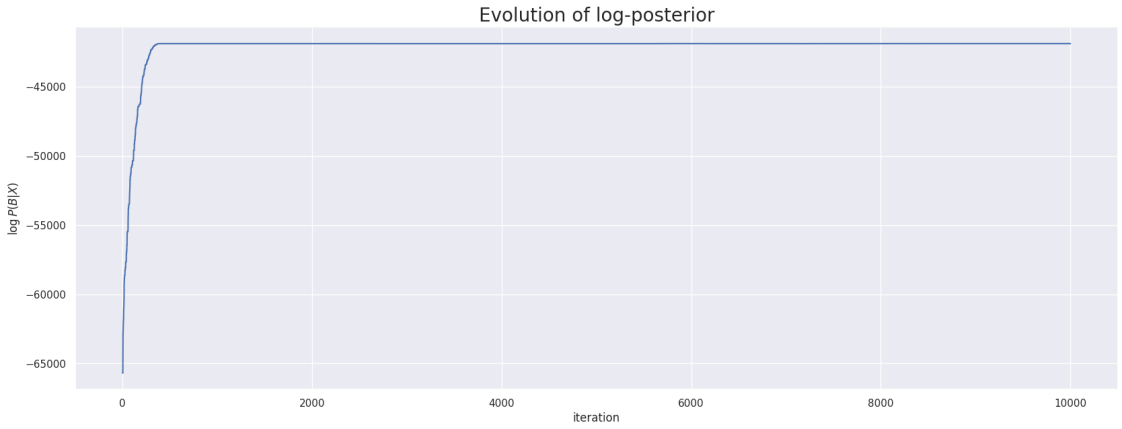
```
axs[1,1].legend(
    loc='upper right',
    fontsize=15
)
```

[69]: <matplotlib.legend.Legend at 0x7f4873b69490>



8.4.6 2.4.6. Plot evolution of log-posterior

[70]:
```
fig = plt.figure(figsize=(20,7))
sns.lineplot(
    data=logs,
    x='iteration',
    y='log_posterior'
)
plt.ylabel(
    '$\log P(B|X)$'
)
t = plt.title(
    'Evolution of log-posterior',
    fontsize=20
)
```



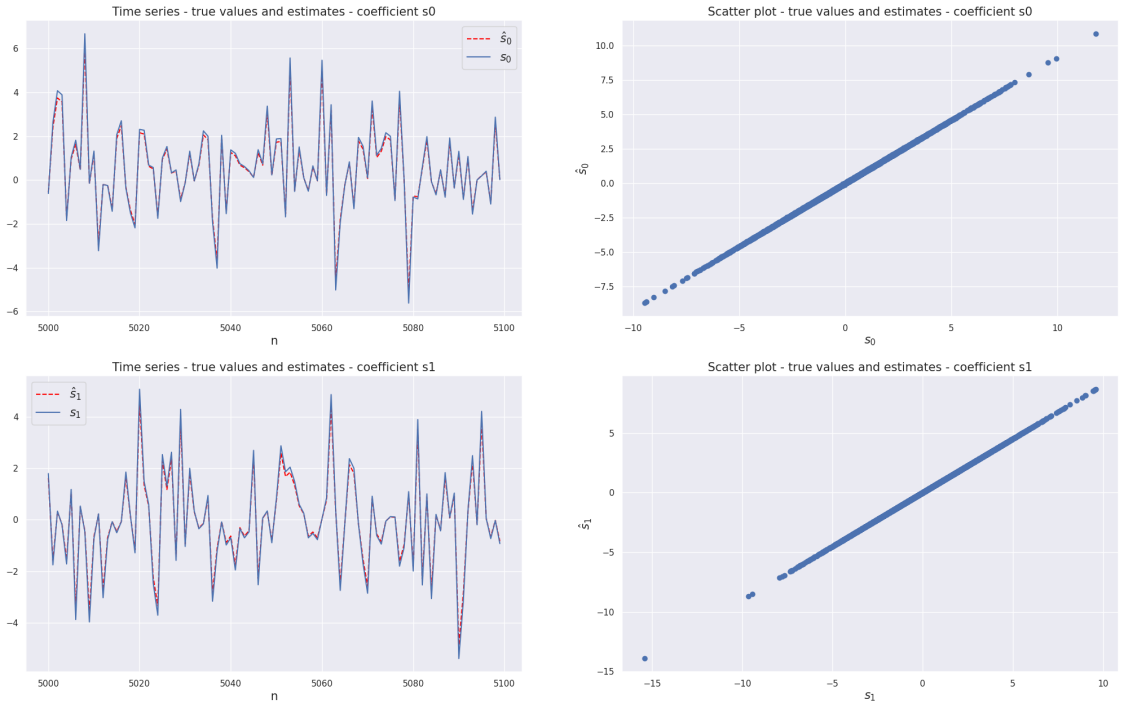
8.4.7 2.4.7. Plot Source Separation Results

[71]:
```
fig, axs = plt.subplots(
    nrows=2, ncols=2,
    figsize=(25,15)
)
NPOINTS=100
PLOT_START=NOBS//2
PLOT_END=PLOT_START+NPOINTS

t=range(NOBS)
s_est = B_est@x

# Axs 00
axs[0,0].plot(
    t[PLOT_START:PLOT_END],
    s_est[0,PLOT_START:PLOT_END],
    label='$\hat{s}_{0}$',
    color='red',
    linestyle='--'
)
axs[0,0].plot(
    t[PLOT_START:PLOT_END],
    s[0,PLOT_START:PLOT_END],
    label='$s_{0}$'
)
axs[0,0].set_xlabel(
    'n',
    fontsize=15
)
axs[0,0].set_title(
    'Time series - true values and estimates - coefficient s0',
```

```
    fontsize=15
)
axs[0,0].legend(fontsize=15)

# Axs 01
axs[0,1].scatter(
    s[0,:],
    s_est[0,:],
)
axs[0,1].set_xlabel(
    '$s_{0}$',
    fontsize=15
)
axs[0,1].set_ylabel(
    '$\hat{s}_{0}$',
    fontsize=15
)
axs[0,1].set_title(
    'Scatter plot - true values and estimates - coefficient s0',
    fontsize=15
)

# Axs 10
axs[1,0].plot(
    t[PLOT_START:PLOT_END],
    s_est[1,PLOT_START:PLOT_END],
    label='$\hat{s}_{1}$',
    color='red',
    linestyle='--'
)
axs[1,0].plot(
    t[PLOT_START:PLOT_END],
    s[1,PLOT_START:PLOT_END],
    label='$s_{1}$'
)
axs[1,0].set_xlabel(
    'n',
    fontsize=15
)
axs[1,0].set_title(
    'Time series - true values and estimates - coefficient s1',
    fontsize=15
)
axs[1,0].legend(fontsize=15)

# Axs 11
axs[1,1].scatter(
```

```
    s[1,:],
    s_est[1,:],
)
axs[1,1].set_xlabel(
    '$s_{1}$',
    fontsize=15
)
axs[1,1].set_ylabel(
    '$\hat{s}_{1}$',
    fontsize=15
)
axs[1,1].set_title(
    'Scatter plot - true values and estimates - coefficient s1',
    fontsize=15
)
```

[71]: Text(0.5, 1.0, 'Scatter plot - true values and estimates - coefficient s1')



[72]:
```
# Error norm
err=np.linalg.norm(
    np.subtract(s,s_est)
)/np.size(s)

# Log error
```

```
errors['slightly_misspecified']['determinant_prior'] = err

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

```
Norma do erro de estimação: 0.001283389775758889
```

8.5 2.5. Perform Analysis - near-identity transformation

Prior:

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

8.5.1 1.5.1 Execute MCMC Sampling

[73]:
```
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
```

[74]:
```
%%time

SAVE_PATH='./artifacts/slightly_misspecified_model/identity.pkl'
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 open(SAVE_PATH, 'wb') as f:
```

```
        pickle.dump(estimator, f)

else:
    # Read artifact
    with open(SAVE_PATH, 'rb') as f:
        estimator=pickle.load(f)
```

```
CPU times: user 3.38 ms, sys: 11.3 ms, total: 14.7 ms
Wall time: 14.5 ms
```

8.5.2 2.5.2. Parse MCMC Results

[75]:
```
# 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']
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']
```

[76]:
```
print('-'*100)
print('Estimated B:\n{}'.format(B_est))
print('True B:\n{}'.format(np.linalg.inv(A)))
print('-'*100)
```

```
----------------------------------------------------------------------------------------------------
Estimated B:
[[ 0.46038294 -0.91476356]
 [ 0.45361317  0.897238  ]]
True B:
[[ 0.5 -1. ]
 [ 0.5  1. ]]
----------------------------------------------------------------------------------------------------
```

[77]:
```
# Posteriors
[r['max_posterior'] for r in estimator.mcmc_results]
```

[77]:
```
[-41881.82767751246,
 -41881.83221326981,
 -41881.84312909353,
 -41881.817088597985,
 -41881.825020334836,
 -41881.83356741597,
 -41881.841500247065,
```

```
-41881.846408694844,
-41881.812157238324,
-41881.80983003557,
-41881.8484135974,
-41881.84471158533,
-41881.82195458354,
-41881.83259082017,
-41881.83241449067,
-41881.82761164671,
-41881.80633217238,
-41881.819427069044,
-41881.81162011927,
-41881.80846535233,
-41881.838700795255,
-41881.817914988766,
-41881.809473237394,
-41881.83650949703,
-41881.86245088612,
-41881.8279622609,
-41881.82768000485,
-41881.8007840359,
-41881.81990722639,
-41881.857021743184]
```

[78]:
```
[r['B_est'] for r in estimator.mcmc_results]
```

[78]:
```
[array([[ 0.46038294, -0.91476356],
        [ 0.45361317,  0.897238  ]]),
 array([[ 0.4600777 , -0.91642292],
        [ 0.45450464,  0.89589705]]),
 array([[-0.46245701,  0.91076917],
        [-0.45143476, -0.90154547]]),
 array([[-0.46140335,  0.91242907],
        [-0.45219765, -0.89935671]]),
 array([[-0.45427057, -0.89499946],
        [ 0.45980682, -0.91681569]]),
 array([[-0.4514364 , -0.90129237],
        [-0.46226633,  0.91038806]]),
 array([[ 0.45247197,  0.8986984 ],
        [-0.46114318,  0.9118283 ]]),
 array([[-0.44883296, -0.90705924],
        [ 0.46480478, -0.9053657 ]]),
 array([[-0.46310552,  0.90755814],
        [-0.45034094, -0.90307394]]),
 array([[-0.45225831, -0.89863412],
        [-0.46199271,  0.91234338]]),
 array([[ 0.46229181, -0.90894194],
```

```
       [ 0.45108801,  0.90103433]]),
 array([[-0.45437525, -0.89457436],
       [ 0.45980813, -0.91508082]]),
 array([[-0.46428024,  0.90583946],
       [ 0.4493518 ,  0.90598409]]),
 array([[-0.46355537,  0.9086193 ],
       [ 0.45021376,  0.90334858]]),
 array([[-0.46213487,  0.9103624 ],
       [-0.45167511, -0.90118979]]),
 array([[ 0.44976228,  0.90494002],
       [-0.46416863,  0.90741739]]),
 array([[ 0.46108323, -0.91296352],
       [-0.45283787, -0.89854946]]),
 array([[-0.46565281,  0.90300718],
       [ 0.44757765,  0.90884867]]),
 array([[-0.46303761,  0.9084941 ],
       [ 0.45057398,  0.90371867]]),
 array([[-0.44915707, -0.90520285],
       [-0.46468571,  0.90578137]]),
 array([[ 0.45251788,  0.89869708],
       [ 0.46096936, -0.91257338]]),
 array([[-0.45278729, -0.89790271],
       [-0.46153375,  0.91330985]]),
 array([[-0.46589026,  0.90317688],
       [ 0.44802042,  0.90802567]]),
 array([[ 0.45284367,  0.89762585],
       [ 0.46084119, -0.91226189]]),
 array([[ 0.45078667,  0.90089983],
       [-0.46230849,  0.90977044]]),
 array([[-0.46296488,  0.90863484],
       [ 0.4507992 ,  0.90247387]]),
 array([[-0.46320505,  0.90922202],
       [ 0.45051748,  0.90259339]]),
 array([[-0.46245357,  0.91072162],
       [ 0.4514174 ,  0.90039948]]),
 array([[ 0.446552  ,  0.91120988],
       [-0.46685386,  0.90094449]]),
 array([[-0.46337712,  0.90769347],
       [-0.45033865, -0.90334322]])]
```

8.5.3 2.5.3. Plot sampled coefficients stochastic process - Markov Chain evolution

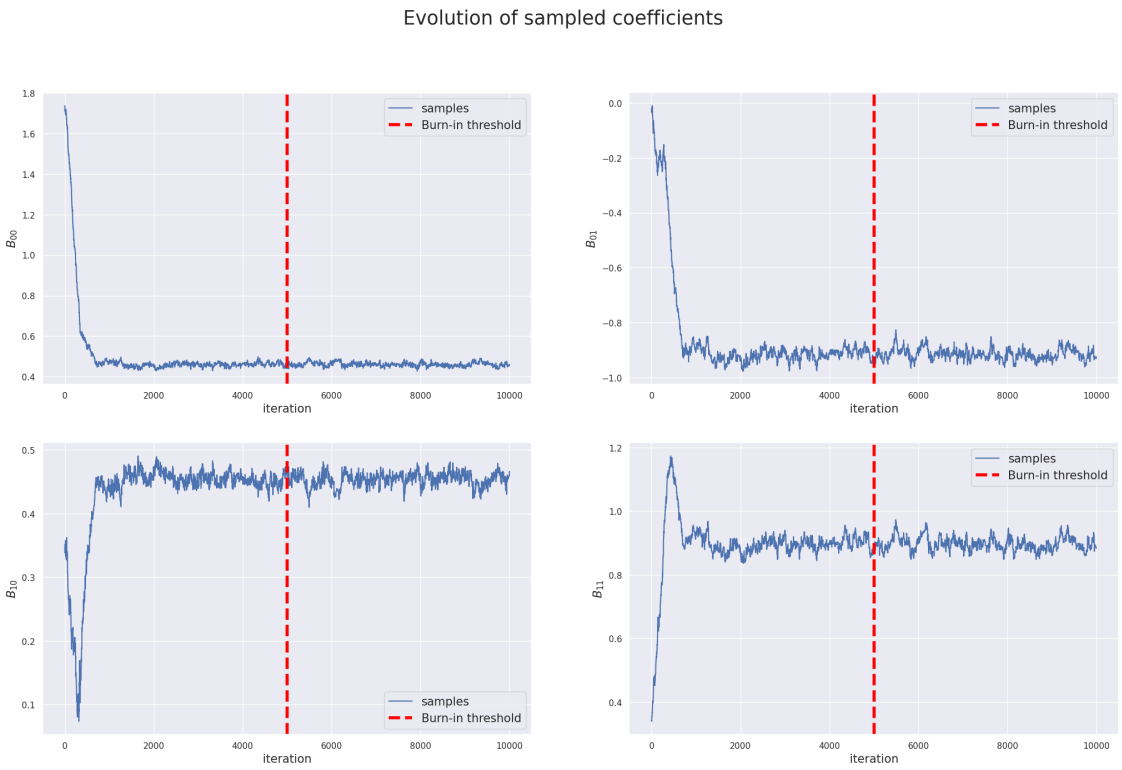
[79]:
```
# Plot sampled coefficients
fig, axs = plt.subplots(
    nrows=2, ncols=2,
    figsize=(25,15)
)
```

```
fig.suptitle(
    'Evolution of sampled coefficients',
    fontsize=25
)

# B_00
axs[0,0].plot(
    logs.iteration,
    samples[:, 0, 0],
    label='samples'
)
axs[0,0].axvline(
    estimator.burn_in_start,
    color='red',
    linestyle='--',
    linewidth=4,
    label='Burn-in threshold'
)
axs[0,0].set_xlabel(
    'iteration',
    fontsize=15
)
axs[0,0].set_ylabel(
    '$B_{00}$',
    fontsize=15
)
axs[0,0].legend(
    fontsize=15
)

# B_01
axs[0,1].plot(
    logs.iteration,
    samples[:, 0, 1],
    label='samples'
)
axs[0,1].axvline(
    estimator.burn_in_start,
    color='red',
    linestyle='--',
    linewidth=4,
    label='Burn-in threshold'
)
axs[0,1].set_xlabel(
    'iteration',
    fontsize=15
```

```
)
axs[0,1].set_ylabel(
    '$B_{01}$',
    fontsize=15
)
axs[0,1].legend(
    fontsize=15
)

# B_10
axs[1,0].plot(
    logs.iteration,
    samples[:, 1, 0],
    label='samples'
)
axs[1,0].axvline(
    estimator.burn_in_start,
    color='red',
    linestyle='--',
    linewidth=4,
    label='Burn-in threshold'
)
axs[1,0].set_xlabel(
    'iteration',
    fontsize=15
)
axs[1,0].set_ylabel(
    '$B_{10}$',
    fontsize=15
)
axs[1,0].legend(
    fontsize=15
)

# B_11
axs[1,1].plot(
    logs.iteration,
    samples[:, 1, 1],
    label='samples'
)
axs[1,1].axvline(
    estimator.burn_in_start,
    color='red',
    linestyle='--',
    linewidth=4,
    label='Burn-in threshold'
)
```

```
axs[1,1].set_xlabel(
    'iteration',
    fontsize=15
)
axs[1,1].set_ylabel(
    '$B_{11}$',
    fontsize=15
)
axs[1,1].legend(
    fontsize=15
)
```

[79]: <matplotlib.legend.Legend at 0x7f4870a12f50>



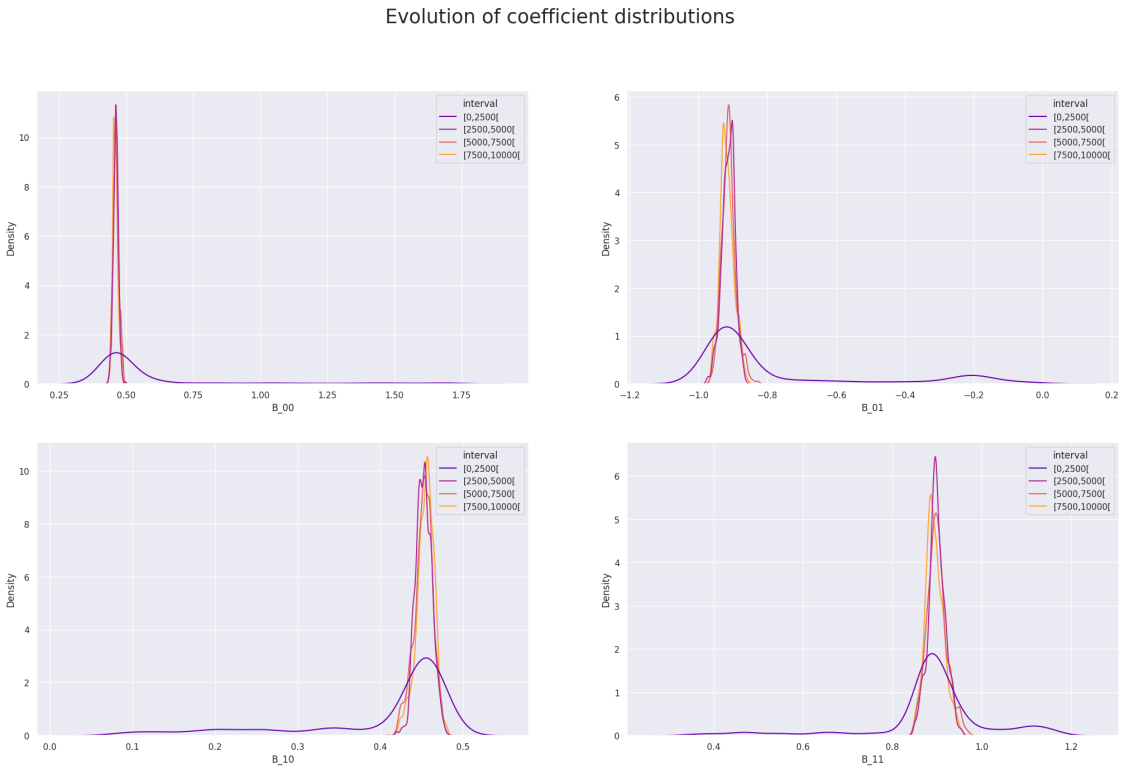
8.5.4 2.5.4. Plot sampled coefficients distributions - Markov Chain evolution

[80]:
```
################################
# Evolution of distributions #
################################

# Get step size and evaluated points
STEP_SIZE=2500
```

```
PALETTE='plasma'

i=0
evaluated_intervals=[]
while i<N_SAMPLES:
    start=i
    end=min(
        i+STEP_SIZE,
        N_SAMPLES
    )
    evaluated_intervals.append(
        (start, end)
    )
    i = i + STEP_SIZE

# Window samples and construct dataframe for plotting
plot_df=pd.DataFrame()
for start, end in evaluated_intervals:
    wdw_df = pd.DataFrame(
        data={
            'interval': ['[{},{}['.format(start, end)]*(end-start)
        }
    )
    wdw_samples = samples[start:end,:,:]
    for i, j in np.ndindex(B_est.shape):
        wdw_df['B_{}{}'.format(i, j)] = wdw_samples[:,i,j]

    plot_df = pd.concat(
        [
            plot_df,
            wdw_df
        ],
        axis=0
    ).reset_index(
        drop=True
    )

# Plot coefficient distribution evolution
fig, axs = plt.subplots(
    nrows=2, ncols=2,
    figsize=(25,15)
)

fig.suptitle(
    'Evolution of coefficient distributions',
    fontsize=25
)
```

```
# B_00
sns.kdeplot(
    data=plot_df,
    x='B_00',
    hue='interval',
    ax=axs[0,0],
    palette=PALETTE
)

# B_01
sns.kdeplot(
    data=plot_df,
    x='B_01',
    hue='interval',
    ax=axs[0,1],
    palette=PALETTE
)

# B_10
sns.kdeplot(
    data=plot_df,
    x='B_10',
    hue='interval',
    ax=axs[1,0],
    palette=PALETTE
)

# B_11
sns.kdeplot(
    data=plot_df,
    x='B_11',
    hue='interval',
    ax=axs[1,1],
    palette=PALETTE
)
```

[80]: <Axes: xlabel='B_11', ylabel='Density'>



8.5.5 2.5.5. Plot Marginal Posteriors for Coefficients - Post-burn-in

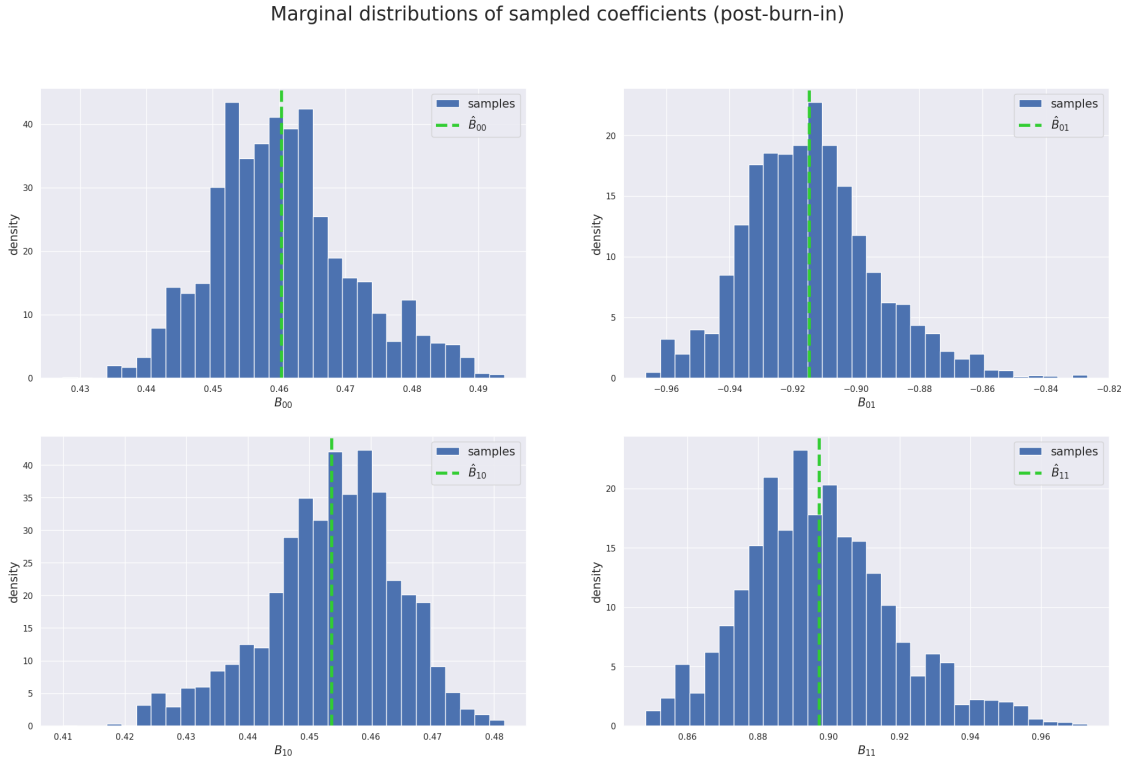
[81]:
```
# Plot sampled coefficients
fig, axs = plt.subplots(
    nrows=2, ncols=2,
    figsize=(25,15)
)

fig.suptitle(
    'Marginal distributions of sampled coefficients (post-burn-in)',
    fontsize=25
)

# B_00
axs[0,0].hist(
    valid_samples[:, 0, 0],
    density=True,
    bins=30,
    label='samples'
)
axs[0,0].axvline(
    B_est[0, 0],
```

```
    color='limegreen',
    linestyle='--',
    linewidth=4,
    label='$\hat{B}_{00}$'
)
axs[0,0].set_xlabel(
    '$B_{00}$',
    fontsize=15
)
axs[0,0].set_ylabel(
    'density',
    fontsize=15
)
axs[0,0].legend(
    loc='upper right',
    fontsize=15
)


# B_01
axs[0,1].hist(
    valid_samples[:, 0, 1],
    density=True,
    bins=30,
    label='samples'
)
axs[0,1].axvline(
    B_est[0, 1],
    color='limegreen',
    linestyle='--',
    linewidth=4,
    label='$\hat{B}_{01}$'
)
axs[0,1].set_xlabel(
    '$B_{01}$',
    fontsize=15
)
axs[0,1].set_ylabel(
    'density',
    fontsize=15
)
axs[0,1].legend(
    loc='upper right',
    fontsize=15
)

# B_10
```

```
axs[1,0].hist(
    valid_samples[:, 1, 0],
    density=True,
    bins=30,
    label='samples'
)
axs[1,0].axvline(
    B_est[1, 0],
    color='limegreen',
    linestyle='--',
    linewidth=4,
    label='$\hat{B}_{10}$'
)
axs[1,0].set_xlabel(
    '$B_{10}$',
    fontsize=15
)
axs[1,0].set_ylabel(
    'density',
    fontsize=15
)
axs[1,0].legend(
    loc='upper right',
    fontsize=15
)

# B_11
axs[1,1].hist(
    valid_samples[:, 1, 1],
    density=True,
    bins=30,
    label='samples'
)
axs[1,1].axvline(
    B_est[1, 1],
    color='limegreen',
    linestyle='--',
    linewidth=4,
    label='$\hat{B}_{11}$'
)
axs[1,1].set_xlabel(
    '$B_{11}$',
    fontsize=15
)
axs[1,1].set_ylabel(
    'density',
    fontsize=15
```

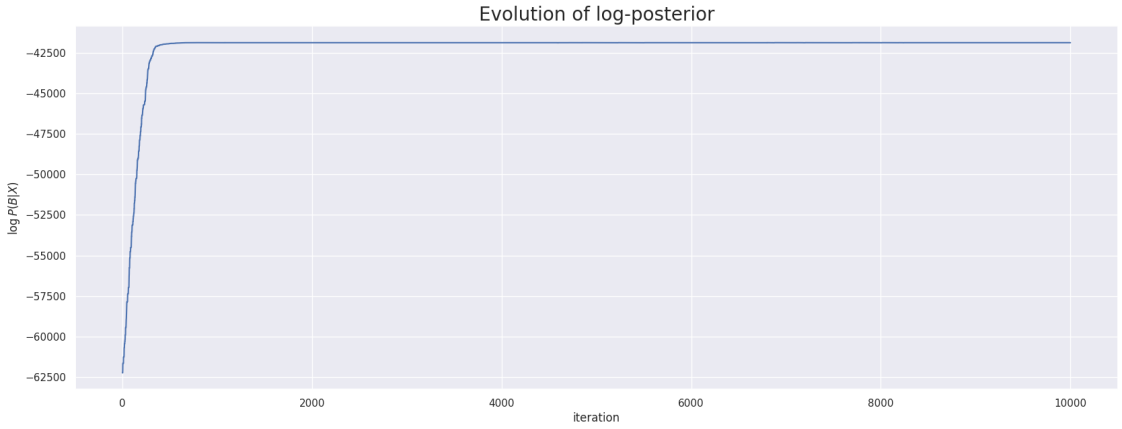
```
)
axs[1,1].legend(
    loc='upper right',
    fontsize=15
)
```

[81]: <matplotlib.legend.Legend at 0x7f48705d1950>



8.5.6 2.5.6. Plot evolution of log-posterior

[82]:
```
fig = plt.figure(figsize=(20,7))
sns.lineplot(
    data=logs,
    x='iteration',
    y='log_posterior'
)
plt.ylabel(
    '$\log P(B|X)$'
)
t = plt.title(
    'Evolution of log-posterior',
    fontsize=20
)
```



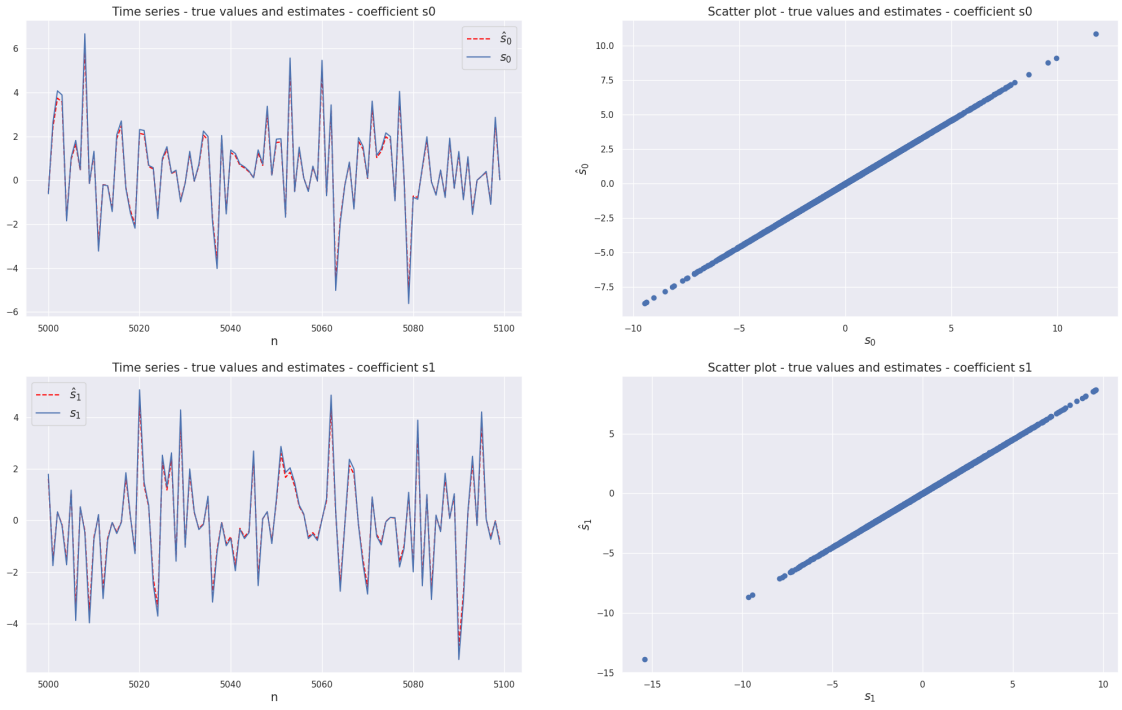
8.5.7 2.5.7. Plot Source Separation Results

[83]:
```
fig, axs = plt.subplots(
    nrows=2, ncols=2,
    figsize=(25,15)
)
NPOINTS=100
PLOT_START=NOBS//2
PLOT_END=PLOT_START+NPOINTS

t=range(NOBS)
s_est = B_est@x

# Axs 00
axs[0,0].plot(
    t[PLOT_START:PLOT_END],
    s_est[0,PLOT_START:PLOT_END],
    label='$\hat{s}_{0}$',
```

```
    color='red',
    linestyle='--'
)
axs[0,0].plot(
    t[PLOT_START:PLOT_END],
    s[0,PLOT_START:PLOT_END],
    label='$s_{0}$'
)
axs[0,0].set_xlabel(
    'n',
    fontsize=15
)
axs[0,0].set_title(
    'Time series - true values and estimates - coefficient s0',
    fontsize=15
)
axs[0,0].legend(fontsize=15)

# Axs 01
axs[0,1].scatter(
    s[0,:],
    s_est[0,:],
)
axs[0,1].set_xlabel(
    '$s_{0}$',
    fontsize=15
)
axs[0,1].set_ylabel(
    '$\hat{s}_{0}$',
    fontsize=15
)
axs[0,1].set_title(
    'Scatter plot - true values and estimates - coefficient s0',
    fontsize=15
)

# Axs 10
axs[1,0].plot(
    t[PLOT_START:PLOT_END],
    s_est[1,PLOT_START:PLOT_END],
    label='$\hat{s}_{1}$',
    color='red',
    linestyle='--'
)
axs[1,0].plot(
    t[PLOT_START:PLOT_END],
    s[1,PLOT_START:PLOT_END],
```

```
    label='$s_{1}$'
)
axs[1,0].set_xlabel(
    'n',
    fontsize=15
)
axs[1,0].set_title(
    'Time series - true values and estimates - coefficient s1',
    fontsize=15
)
axs[1,0].legend(fontsize=15)

# Axs 11
axs[1,1].scatter(
    s[1,:],
    s_est[1,:],
)
axs[1,1].set_xlabel(
    '$s_{1}$',
    fontsize=15
)
axs[1,1].set_ylabel(
    '$\hat{s}_{1}$',
    fontsize=15
)
axs[1,1].set_title(
    'Scatter plot - true values and estimates - coefficient s1',
    fontsize=15
)
```

[83]: Text(0.5, 1.0, 'Scatter plot - true values and estimates - coefficient s1')



[84]:
```
# Error norm
err=np.linalg.norm(
    np.subtract(s,s_est)
)/np.size(s)

# Log error
errors['slightly_misspecified']['near_identity_prior'] = err

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

```
Norma do erro de estimação: 0.0012851809941576843
```

[85]:
```
pprint.pp(errors)
```

```
{'perfect_model': {'likelihood': 0.00014791877114411317,
                   'determinant_prior': 0.00015101624153912317,
                   'near_identity_prior': 0.00047823481320951455},
 'slightly_misspecified': {'likelihood': 0.0012914937731572473,
                           'determinant_prior': 0.001283389775758889,
                           'near_identity_prior': 0.0012851809941576843}}
```