ČESKÉ VYSOKÉ
UČENÍ TECHNICKÉ
V PRAZE

FAKULTA STAVEBNÍ

HABILITAČNÍ PRÁCE

PŘÍLOHA 3: STOCHASTICKÉ MODELOVÁNÍ

2026

ING. VJAČESLAV USMANOV, PH.D.

OBSAH

Stochastické modelování 00: MATLAB

Stochastické modelování 01: Matice přechodů

Stochastické modelování 02: Algoritmus MCMC (Metropolis–Hastings)

Stochastické modelování 03: Algoritmus MCMC (Přímé vzorkování)

Stochastické modelování 04: Ověření modelu

STOCHASTIC MODEL 00: MATLAB

```
clc;
clear;
format long;

%% Definice parametrů
dt = 1;   % časový krok (s)

% Průměrné doby trvání stavů (s)
states = { ...
    'S1','S2','S3','S4','S5','S6','S7','S8', ...
    'S9','S10','S11','S12'};

T = containers.Map( ...
    states, ...
    [2 2 5 2 5 7 3 6 60 70 160 20]);

% Intenzity poruch (1/s)
lambda_vals = containers.Map( ...
    {'S9','S10','S11','S12'}, ...
    [4/8896, 7/8896, 3/8896, 13/8896]);

n = length(states);

%% Inicializace přechodové matice
P = zeros(n,n);

%% Definice funkcí
exit_probability = @(Ti) 1 - exp(-dt / Ti);
fault_probability = @(lam) 1 - exp(-lam * dt);

%% Definice přechodů
transitions = containers.Map;

transitions('S1')  = {'S2','S9','S11'};
transitions('S2')  = {'S3','S9','S11'};
transitions('S3')  = {'S4','S9','S10','S11'};
transitions('S4')  = {'S5','S9','S11'};
transitions('S5')  = {'S6','S9','S10','S11','S12'};
transitions('S6')  = {'S7','S9','S11'};
transitions('S7')  = {'S8','S9','S10','S11'};
transitions('S8')  = {'S1','S9','S11'};
transitions('S9')  = {'S1'};
transitions('S10') = {'S3'};
transitions('S11') = {'S1'};
transitions('S12') = {'S5'};

%% Výpočet přechodové matice
for i = 1:n
    si = states{i};
    p_exit = exit_probability(T(si));

    outgoing = transitions(si);

    for k = 1:length(outgoing)
        sj = outgoing{k};
        j = find(strcmp(states, sj));
```

```
        if isKey(lambda_vals, sj)
            P(i,j) = fault_probability(lambda_vals(sj));
        else
            P(i,j) = p_exit;
        end
    end

    % pravděpodobnost setrvání
    P(i,i) = 1 - sum(P(i,:));
end

%% Kontrola součtů řádků
disp("Součty řádků:");
disp(sum(P,2));

disp("Přechodová matice P:");
disp(round(P,8));

%% Stacionární rozdělení
[V,D] = eig(P');

[~,idx] = min(abs(diag(D) - 1));
pi_vec = V(:,idx);

pi_vec = real(pi_vec / sum(pi_vec));

disp("Stacionární rozdělení pi:");
disp(round(pi_vec,6));

%% Dostupnost systému
availability = sum(pi_vec(1:8));
disp("Dostupnost systému:");
disp(round(availability,4));

%% Podíl neprovozního času
downtime = sum(pi_vec(9:12));
disp("Podíl neprovozního času:");
disp(round(downtime,4));

%% Tabulka přechodové matice
P_table = array2table(P, 'VariableNames', states, 'RowNames', states);
disp(P_table);

%% Heatmapa (logaritmická škála)
figure;
imagesc(log10(P + eps));
colorbar;
colormap('hot');
xticks(1:n);
yticks(1:n);
xticklabels(states);
yticklabels(states);
xtickangle(45);
title('Heatmapa přechodové matice (log10 škála)');
xlabel('Do stavu');
ylabel('Ze stavu');
```

Stochastic Model 01: Matice přechodů (Transition matrix)

In [1]:
```
# Instalace potřebných knihoven
#%pip install pandas
#%pip install numpy
#%pip install seaborn matplotlib
```

In [2]:
```
# Import potřebných knihoven
import pandas as pd
import numpy as np

import seaborn as sns
import matplotlib.pyplot as plt
```

Definice modelových parametrů

In [3]:
```
# nastavení formátu výpisu pro zobrazení až 8 desetinných míst.
np.set_printoptions(precision=8, suppress=True)

# Definice průměrných dob trvání jednotlivých stavů systému a intenzit přechodu do neprovozníc
dt = 1  # časový krok (s)

# prumerne doby trvani (s)
T = {
    "S1": 2, "S2": 2, "S3": 5, "S4": 2,
    "S5": 5, "S6": 7, "S7": 3, "S8": 6,
    "S9": 60, "S10": 70, "S11": 160, "S12": 20
}


# intenzity poruch (1/s)
lambda_vals = {
    "S9": 4/8896,
    "S10": 7/8896,
    "S11": 3/8896,
    "S12": 13/8896
}
```

In [4]:
```
# Inicializace přechodové matice Markovského řetězce
states = list(T.keys())
n = len(states)

P = np.zeros((n,n))
```

Pomocne funkce

In [5]:
```
def exit_probability(Ti):
    return 1 - np.exp(-dt / Ti)

def fault_probability(lam):
    return 1 - np.exp(-lam * dt)
```

Definice prechodu

In [6]:
```
transitions = {
```

```
    "S1": ["S2", "S9", "S11"],
    "S2": ["S3", "S9", "S11"],
    "S3": ["S4", "S9", "S10", "S11"],
    "S4": ["S5", "S9", "S11"],
    "S5": ["S6", "S9", "S10", "S11", "S12"],
    "S6": ["S7", "S9", "S11"],
    "S7": ["S8", "S9", "S10", "S11"],
    "S8": ["S1", "S9", "S11"],
    "S9": ["S1"],
    "S10": ["S3"],
    "S11": ["S1"],
    "S12": ["S5"]
}
```

Sestaveni matice

In [7]:
```
for i, si in enumerate(states):

    p_exit = exit_probability(T[si])
    total_fault_prob = 0

    # poruchove prechody
    for sj in transitions[si]:
        if sj in lambda_vals:
            total_fault_prob += fault_probability(lambda_vals[sj])

    # rozdeleni pravdepodobnosti
    for sj in transitions[si]:
        j = states.index(sj)

        if sj in lambda_vals:
            P[i, j] = fault_probability(lambda_vals[sj])
        else:
            P[i, j] = p_exit

    # pravdepodobnost setrvani
    P[i, i] = 1 - P[i].sum()
```

Kontrola matice

In [8]:
```
print("Soucty radku:", P.sum(axis=1))

print("Prechodova matice P:")
print(np.round(P, 8))
```

```
Soucty radku: [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
Prechodova matice P:
[[0.60574395 0.39346934 0.         0.         0.         0.
  0.         0.         0.00044954 0.         0.00033717 0.        ]
 [0.         0.60574395 0.39346934 0.         0.         0.
  0.         0.         0.00044954 0.         0.00033717 0.        ]
 [0.         0.         0.81715748 0.18126925 0.         0.
  0.         0.         0.00044954 0.00078656 0.00033717 0.        ]
 [0.         0.         0.         0.60574395 0.39346934 0.
  0.         0.         0.00044954 0.         0.00033717 0.        ]
 [0.         0.         0.         0.         0.81569722 0.18126925
  0.         0.         0.00044954 0.00078656 0.00033717 0.00146026]
 [0.         0.         0.         0.         0.         0.86609119
  0.1331221  0.         0.00044954 0.         0.00033717 0.        ]
 [0.         0.         0.         0.         0.         0.
  0.71495804 0.28346869 0.00044954 0.00078656 0.00033717 0.        ]
 [0.15351828 0.         0.         0.         0.         0.
  0.         0.84569501 0.00044954 0.         0.00033717 0.        ]
 [0.01652855 0.         0.         0.         0.         0.
  0.         0.         0.98347145 0.         0.         0.        ]
 [0.         0.         0.01418416 0.         0.         0.
  0.         0.         0.         0.98581584 0.         0.        ]
 [0.00623051 0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.99376949 0.        ]
 [0.         0.         0.         0.         0.04877058 0.
  0.         0.         0.         0.         0.         0.95122942]]
```

Výpočet stacionárního rozdělení

In [9]:
```
# Výpočet stacionárního rozdělení pravděpodobností pomocí vlastního řešení matice.
eigvals, eigvecs = np.linalg.eig(P.T)

idx = np.argmin(np.abs(eigvals-1))

pi = eigvecs[:,idx]
pi = np.real(pi / np.sum(pi))

print("\nStacionární rozdělení pi:")
print(np.round(pi,6))
```

```
Stacionární rozdělení pi:
[0.064078 0.06395  0.139183 0.063993 0.13771  0.186414 0.08706  0.159936
 0.024541 0.020182 0.048831 0.004123]
```

Vyhodnocení dostupnosti systému

In [10]:
```
# Stanovení dlouhodobé provozní dostupnosti systému
availability = np.sum(pi[0:8])
print("\nDostupnost systému:")
print(round(availability,4))

# Stanovení podílu neprovozního času
downtime = np.sum(pi[8:12])
print("\nPodíl neprovozního času:")
print(round(downtime,4))
```

```
Dostupnost systému:
0.9023

Podíl neprovozního času:
0.0977
```

Export matice formátu CSV

In [11]:
```
states = [
    "S1","S2","S3","S4","S5","S6","S7","S8",
    "S9","S10","S11","S12"
]

df = pd.DataFrame(P, index=states, columns=states)
```

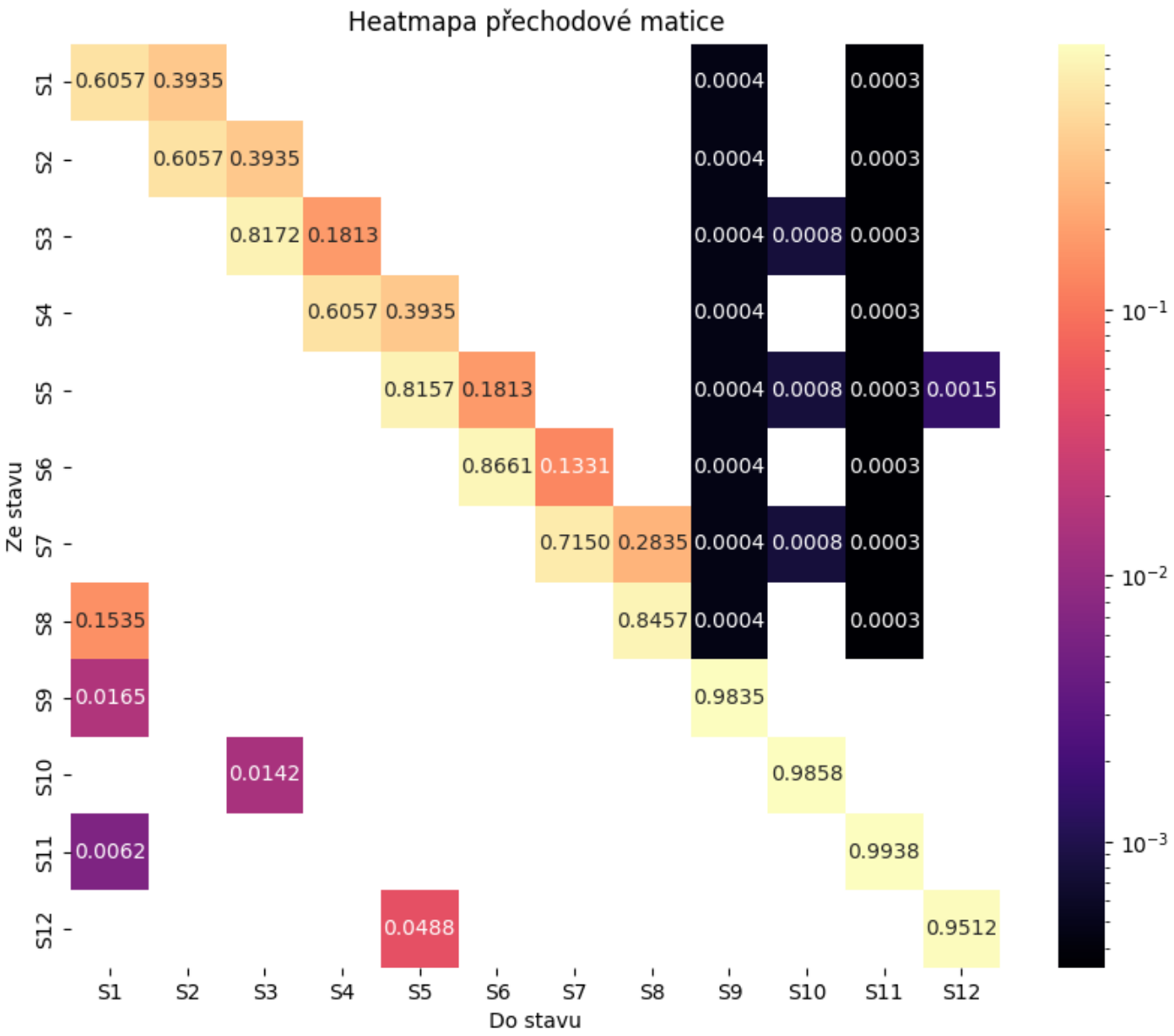
In [12]:
```
df
```

Out[12]:

	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10
S1	0.605744	0.393469	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000450	0.000000
S2	0.000000	0.605744	0.393469	0.000000	0.000000	0.000000	0.000000	0.000000	0.000450	0.000000
S3	0.000000	0.000000	0.817157	0.181269	0.000000	0.000000	0.000000	0.000000	0.000450	0.000787
S4	0.000000	0.000000	0.000000	0.605744	0.393469	0.000000	0.000000	0.000000	0.000450	0.000000
S5	0.000000	0.000000	0.000000	0.000000	0.815697	0.181269	0.000000	0.000000	0.000450	0.000787
S6	0.000000	0.000000	0.000000	0.000000	0.000000	0.866091	0.133122	0.000000	0.000450	0.000000
S7	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.714958	0.283469	0.000450	0.000787
S8	0.153518	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.845695	0.000450	0.000000
S9	0.016529	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.983471	0.000000
S10	0.000000	0.000000	0.014184	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.985816
S11	0.006231	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
S12	0.000000	0.000000	0.000000	0.000000	0.048771	0.000000	0.000000	0.000000	0.000000	0.000000

Vykreslení heatmapy

In [13]:
```
plt.figure(figsize=(10,8))

sns.heatmap(
    df,
    annot=True,
    fmt=".4f",
    cmap="magma",
    norm=plt.matplotlib.colors.LogNorm()
)

plt.title("Heatmapa přechodové matice")
plt.xlabel("Do stavu")
plt.ylabel("Ze stavu")
plt.show()
```



In [14]:
```
df.to_csv('../../data/03_StochModel/transition_matrix.csv', index=False)
```

Zdroj: vlastní implementace na základě [Stewart, 2009]

- Stewart, W.J. (2009). Probability, Markov Chains, Queues, and Simulation. Princeton University Press.
- Norris, J.R. (1998). Markov Chains. Cambridge University Press.

Autor / Organizace / Datum

Vjačeslav Usmanov, ČVUT v Praze, Fakulta stavební

Přehled změn

Datum (YYYY-MM-DD)	Verze	Autor změny	Popis změny
2026-01-25	1.1	Vjačeslav Usmanov	added SM_01_Transition_matrix.ipynb
2026-02-15	1.2	Vjačeslav Usmanov	changed SM_01_Transition_matrix.ipynb

Stochastic Model 02: Algoritmus MCMC (Metropolis–Hastings)

In [1]:
```
# Instalace potřebných knihoven
#%pip install pandas
#%pip install numpy
#%pip install seaborn matplotlib
```

In [2]:
```
# Import potřebných knihoven
import pandas as pd
import numpy as np

import seaborn as sns
import matplotlib.pyplot as plt
```

Načtení přechodové matice

In [3]:
```
# Soubor je načten a přiřazen do proměnné ‚df‘
other_path = '../../data/03_StochModel/transition_matrix.csv'
df = pd.read_csv(other_path, header=0)
P = df.to_numpy()
P
```

Out[3]:
```
array([[6.05743947e-01, 3.93469340e-01, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        4.49539215e-04, 0.00000000e+00, 3.37173360e-04, 0.00000000e+00],
       [0.00000000e+00, 6.05743947e-01, 3.93469340e-01, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        4.49539215e-04, 0.00000000e+00, 3.37173360e-04, 0.00000000e+00],
       [0.00000000e+00, 0.00000000e+00, 8.17157480e-01, 1.81269247e-01,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        4.49539215e-04, 7.86561002e-04, 3.37173360e-04, 0.00000000e+00],
       [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 6.05743947e-01,
        3.93469340e-01, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        4.49539215e-04, 0.00000000e+00, 3.37173360e-04, 0.00000000e+00],
       [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        8.15697216e-01, 1.81269247e-01, 0.00000000e+00, 0.00000000e+00,
        4.49539215e-04, 7.86561002e-04, 3.37173360e-04, 1.46026371e-03],
       [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 8.66091187e-01, 1.33122100e-01, 0.00000000e+00,
        4.49539215e-04, 0.00000000e+00, 3.37173360e-04, 0.00000000e+00],
       [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 7.14958037e-01, 2.83468689e-01,
        4.49539215e-04, 7.86561002e-04, 3.37173360e-04, 0.00000000e+00],
       [1.53518275e-01, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 8.45695012e-01,
        4.49539215e-04, 0.00000000e+00, 3.37173360e-04, 0.00000000e+00],
       [1.65285462e-02, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        9.83471454e-01, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [0.00000000e+00, 0.00000000e+00, 1.41841576e-02, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 9.85815842e-01, 0.00000000e+00, 0.00000000e+00],
       [6.23050938e-03, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 9.93769491e-01, 0.00000000e+00],
       [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        4.87705755e-02, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 9.51229425e-01]])
```

Výpočet stacionárního rozdělení

In [4]:
```
eigvals, eigvecs = np.linalg.eig(P.T)
idx = np.argmin(np.abs(eigvals-1))
pi = np.real(eigvecs[:,idx])
pi = pi / np.sum(pi)
```

Parametry simulace

In [5]:
```
n_states = 12

# počet iteračních kroků (sekund)
n_iter = 100_000

# směrodatná odchylka návrhového normálního rozdělení
sigma = 1.5

samples = np.zeros(n_iter, dtype=int)
```

```
# počáteční stav S1 (index 0)
current_state = 0
samples[0] = current_state

# nastavení seedu (počátečního stavu generátoru náhodných čísel)
np.random.seed(122)
```

Metropolis–Hastings simulace

In [6]:
```
for t in range(1, n_iter):

    # --- návrh nového stavu ---
    proposal = int(np.round(
        np.random.normal(loc=current_state, scale=sigma)
    ))

    # omezení na interval stavů
    proposal = np.clip(proposal, 0, n_states-1)

    # --- pravděpodobnosti ---
    p_current = pi[current_state]
    p_proposal = pi[proposal]

    # symetrické návrhové rozdělení → zkrácení poměru
    alpha = min(1, p_proposal / p_current)

    # --- přijetí / zamítnutí ---
    if np.random.rand() < alpha:
        current_state = proposal

    samples[t] = current_state
```

Výsledná trajektorie stavů

In [7]:
```
# převod na stavy S1–S12
states = samples + 1
```

Empirické rozdělení

In [8]:
```
hist = np.bincount(samples, minlength=n_states)
empirical_pi = hist / np.sum(hist)

print("Empirické rozdělení:")
print(empirical_pi)
```

```
Empirické rozdělení:
[0.08948 0.05646 0.13601 0.06233 0.13584 0.18241 0.08404 0.15545 0.02463
 0.02046 0.04638 0.00651]
```

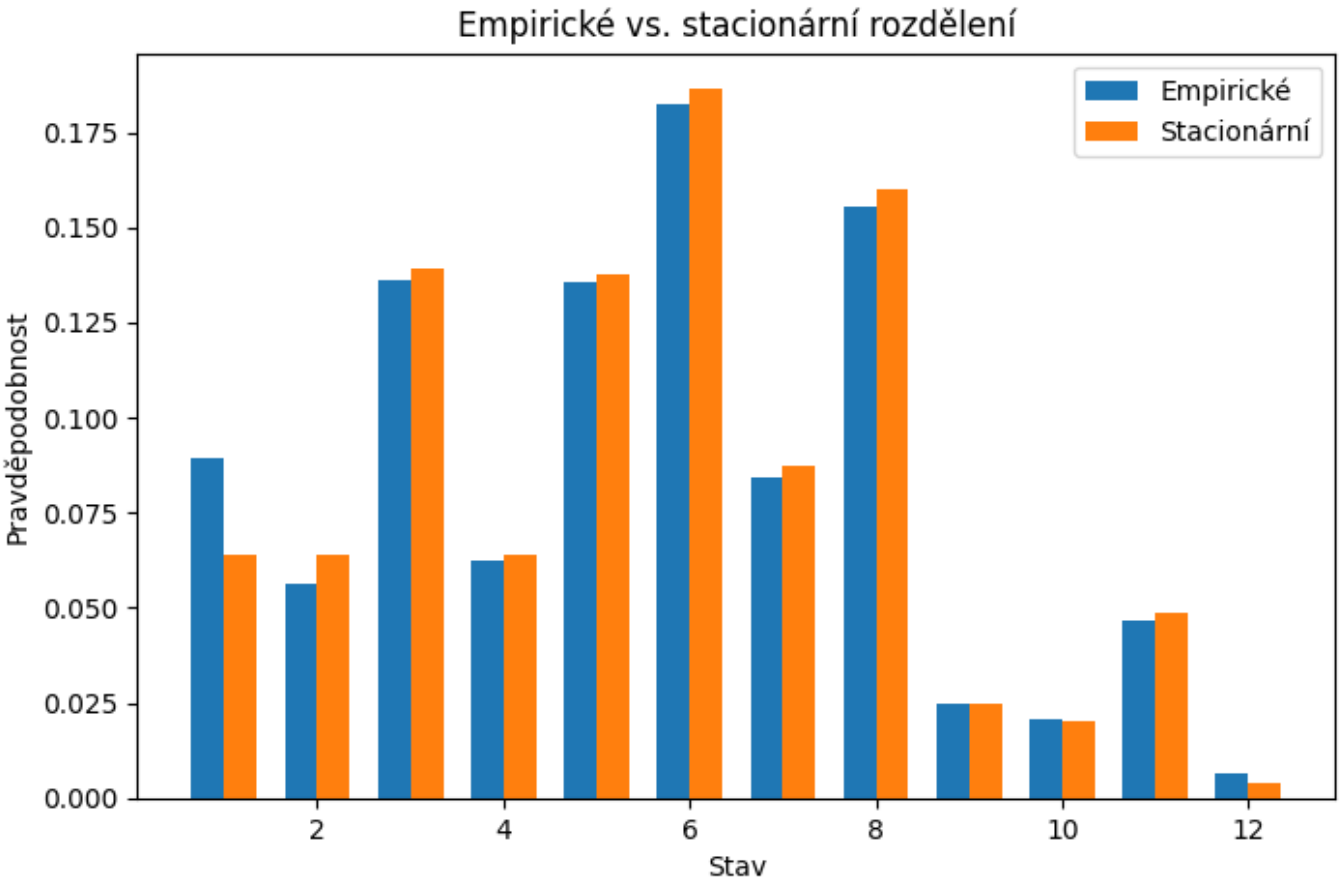
Porovnání se stacionárním rozdělením

In [9]:
```
print("Teoretické pi:")
print(pi)
```

```
Teoretické pi:
[0.0640775  0.06394964 0.13918257 0.06399273 0.13770963 0.18641433
 0.0870604  0.15993584 0.02454114 0.02018244 0.04883054 0.00412323]
```

Sloupcový density graf

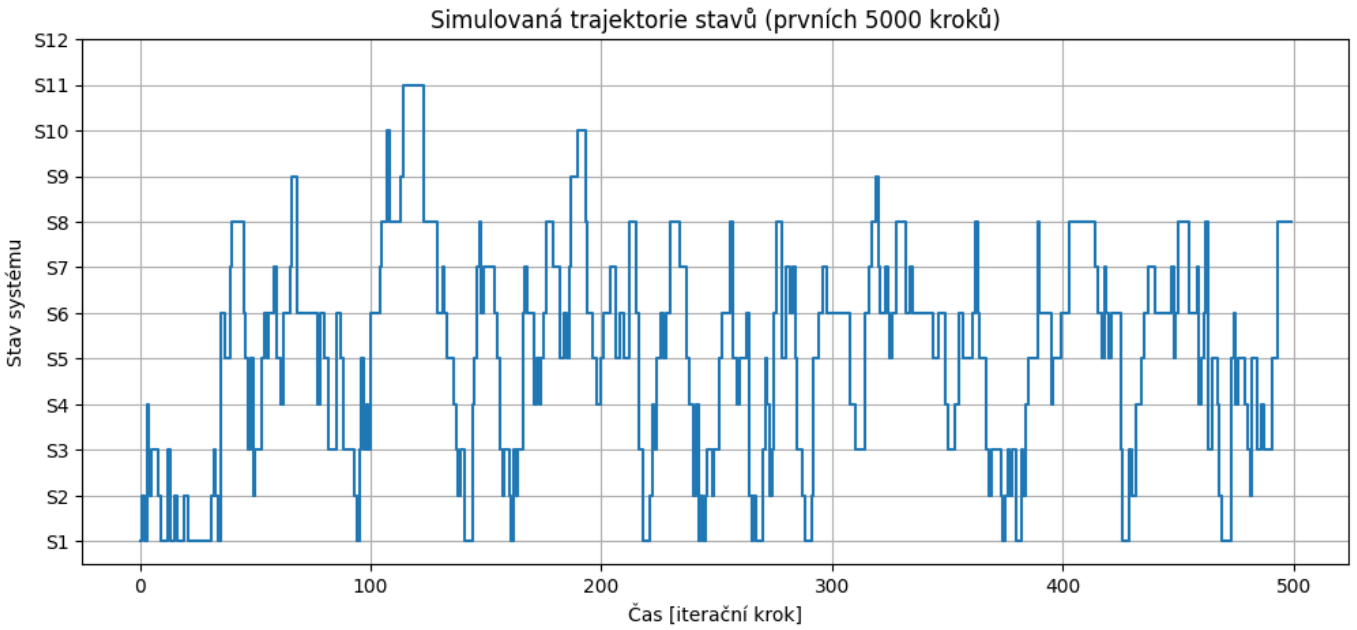
In [10]:
```
width = 0.35
states = np.arange(1,13)
plt.figure(figsize=(8,5))

plt.bar(states - width/2, empirical_pi, width, label='Empirické')
plt.bar(states + width/2, pi, width, label='Stacionární')

plt.xlabel("Stav")
plt.ylabel("Pravděpodobnost")
plt.title("Empirické vs. stacionární rozdělení")
plt.legend()
plt.show()
```



Ilustrační graf simulace

In [11]:
```
time = np.arange(len(samples))
states = samples + 1   # převod na S1–S12
```

In [12]:
```
N = 500

plt.figure(figsize=(12,5))
plt.step(time[:N], states[:N], where='post')

plt.xlabel("Čas [iterační krok]")
plt.ylabel("Stav systému")
plt.title("Simulovaná trajektorie stavů (prvních 5000 kroků)")

plt.yticks(np.arange(1,13), [f"S{i}" for i in range(1,13)])
plt.grid(True)

plt.show()
```

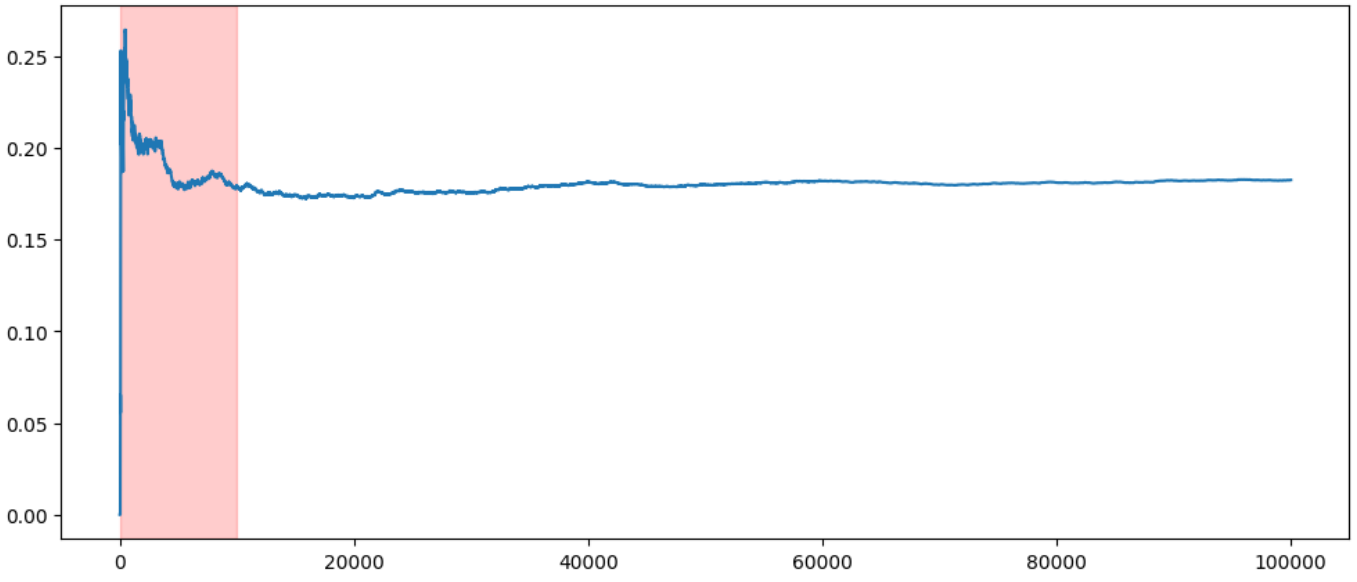


Spálení počáteční části simulace (Burn-in)

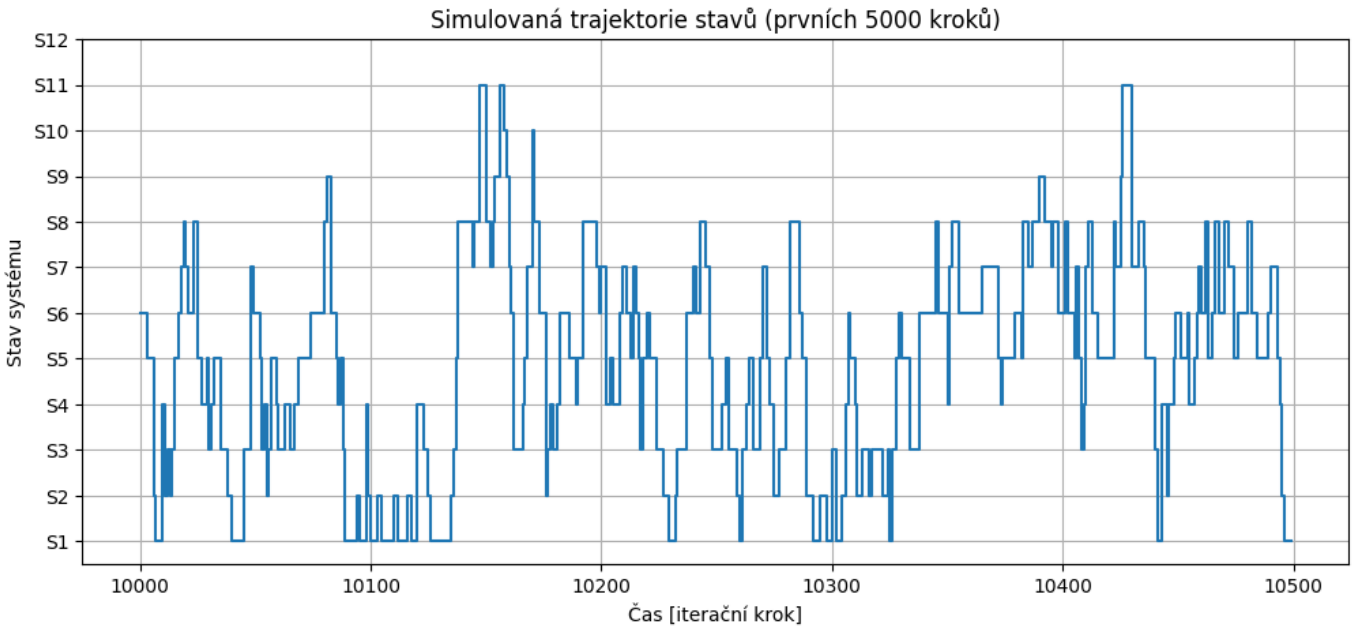
Počáteční transientní fáze simulace (burn-in) byla identifikována a odstraněna z další analýzy. Tato oblast je v grafu vyznačena červeně.

In [13]:
```
burn_in = 10_000
running_mean = np.cumsum(samples==5)/np.arange(1,len(samples)+1)
plt.figure(figsize=(12,5))
plt.axvspan(0, burn_in, color='red', alpha=0.2, label='Burn-in')
plt.plot(running_mean)
```

Out[13]:
```
[<matplotlib.lines.Line2D at 0x1d5eace6f10>]
```



In [14]:
```
plt.figure(figsize=(12,5))
plt.step(time[burn_in:burn_in+N], states[burn_in:burn_in+N], where='post')

plt.xlabel("Čas [iterační krok]")
plt.ylabel("Stav systému")
plt.title("Simulovaná trajektorie stavů (prvních 5000 kroků)")

plt.yticks(np.arange(1,13), [f"S{i}" for i in range(1,13)])
plt.grid(True)

plt.show()
```



Odstranění burn-in

In [15]:
```
samples_burned = samples[burn_in:]
```

Empirické rozdělení

In [16]:
```
hist = np.bincount(samples_burned, minlength=n_states)
empirical_pi = hist / np.sum(hist)

print("Empirické rozdělení:")
print(empirical_pi)
```

```
Empirické rozdělení:
[0.09033333 0.05648889 0.1359     0.06225556 0.13518889 0.18282222
 0.08408889 0.15475556 0.02436667 0.02058889 0.04686667 0.00634444]
```

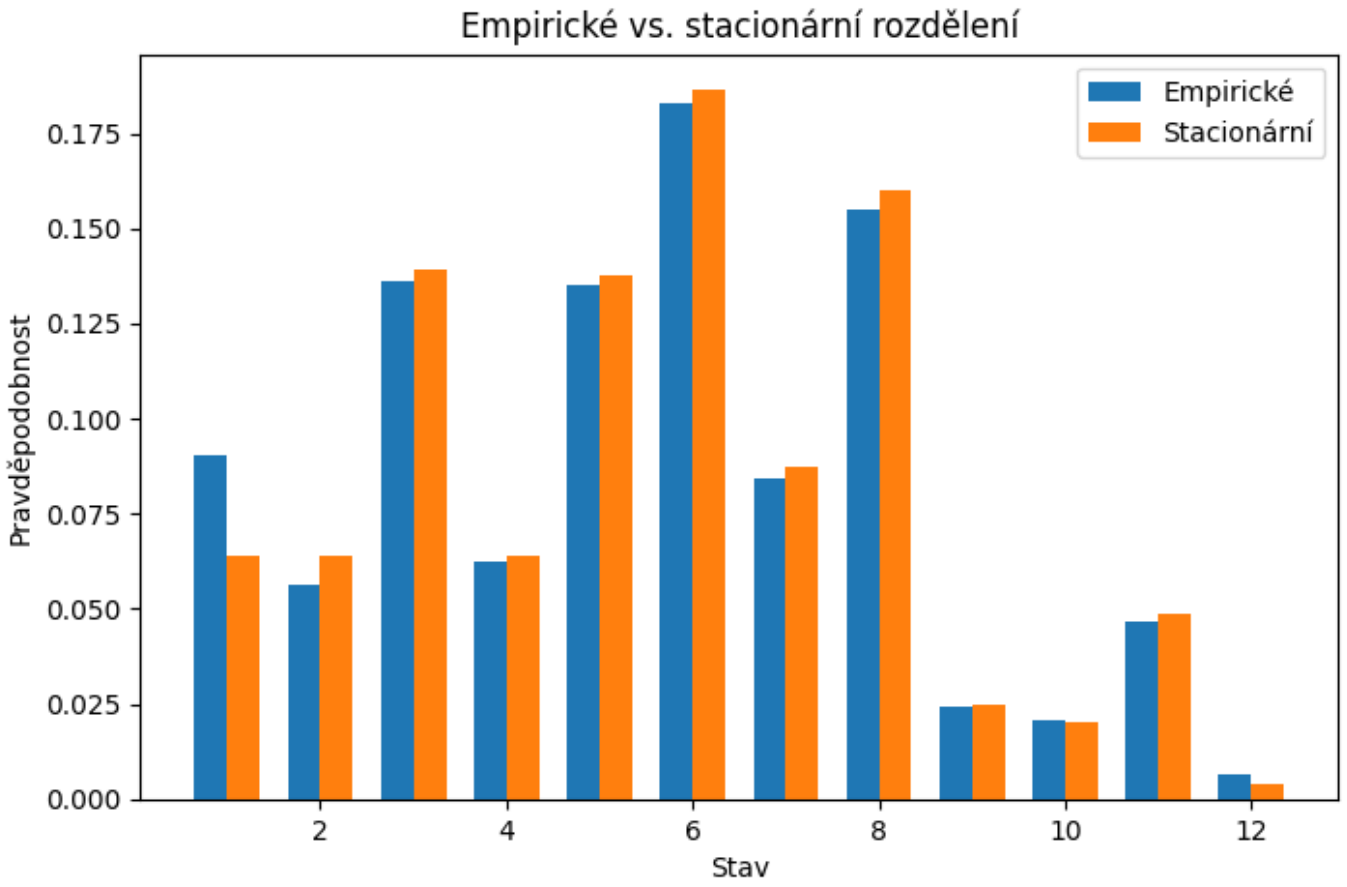
Porovnání se stacionárním rozdělením

In [17]:
```
print("Teoretické pi:")
print(pi)
```

```
Teoretické pi:
[0.0640775  0.06394964 0.13918257 0.06399273 0.13770963 0.18641433
 0.0870604  0.15993584 0.02454114 0.02018244 0.04883054 0.00412323]
```

Sloupcový density graf

In [18]:
```
width = 0.35
states = np.arange(1,13)
plt.figure(figsize=(8,5))

plt.bar(states - width/2, empirical_pi, width, label='Empirické')
plt.bar(states + width/2, pi, width, label='Stacionární')

plt.xlabel("Stav")
plt.ylabel("Pravděpodobnost")
plt.title("Empirické vs. stacionární rozdělení")
plt.legend()
plt.show()
```



Export simulace formátu CSV

In [19]:
```
df_sim = pd.DataFrame({
    "time": np.arange(len(samples_burned)),
    "state_index": samples_burned,
    "state": samples_burned + 1
})
```

In [20]:
```
df_sim.to_csv('../../data/03_StochModel/simulation_MCMC_MH.csv', index=False)
```

Autor / Organizace / Datum

Vjačeslav Usmanov, ČVUT v Praze, Fakulta stavební

Přehled změn

Datum (YYYY-MM-DD)	Verze	Autor změny	Popis změny
2026-01-25	1.1	Vjačeslav Usmanov	added SM_02_MCMC_MH.ipynb
2026-02-15	1.2	Vjačeslav Usmanov	changed SM_02_MCMC_MH.ipynb

Stochastic Model 03: Algoritmus MCMC (Přímé vzorkování)

In [1]:
```
# Instalace potřebných knihoven
#%pip install pandas
#%pip install numpy
#%pip install seaborn matplotlib
```

In [2]:
```
# Import potřebných knihoven
import pandas as pd
import numpy as np

import seaborn as sns
import matplotlib.pyplot as plt
```

Načtení přechodové matice

In [3]:
```
# Soubor je načten a přiřazen do proměnné ‚df‘
other_path = '../../data/03_StochModel/transition_matrix.csv'
df = pd.read_csv(other_path, header=0)
P = df.to_numpy()
```

Parametry simulace

In [4]:
```
n_states = 12

dt = 1                              # sekunda

# počet iteračních kroků (sekund)
simulation_time = 2_000_000       # délka simulace

# Doby setrvání (s)
T = {
    "S1": 2, "S2": 2, "S3": 5, "S4": 2,
    "S5": 5, "S6": 7, "S7": 3, "S8": 6,
    "S9": 60, "S10": 70, "S11": 160, "S12": 20
}

# Stav
T_S = {
    "S1": 1, "S2": 2, "S3": 3, "S4": 4,
    "S5": 5, "S6": 6, "S7": 7, "S8": 8,
    "S9": 9, "S10": 10, "S11": 11, "S12": 12
}

# Intenzity poruch (1/s)
lambda_vals = {
    "S9": 4/8896,
    "S10": 7/8896,
    "S11": 3/8896,
    "S12": 13/8896
}

# nastavení seedu (počátečního stavu generátoru náhodných čísel)
rng = np.random.default_rng(seed=1122)

# Sekvence provozních stavů
```

```
operational_sequence = ["S1","S2","S3","S4","S5","S6","S7","S8"]

# Návraty z poruch
returns = {
    "S9": "S1",
    "S10": "S3",
    "S11": "S1",
    "S12": "S5"
}

burn_in = 100_000
```

Nastavení počátečního stavu

In [5]:
```
# počáteční stav S1 (index 0)

current_state = "S1"
time_in_state = 0

state_time_counter = {s:0 for s in T.keys()}
state_time_counter_burn_in = {s:0 for s in T.keys()}
```

Generování stochastických vzorků pomocí metody MCMC

In [6]:
```
state_history = []
samples = []

for t in range(simulation_time):

    if t > burn_in:
        state_time_counter_burn_in[current_state] += dt
    state_time_counter[current_state] += dt
    time_in_state += dt

    # ===== PROVOZNÍ STAVY =====
    if current_state in operational_sequence:

        # 1 Kontrola poruch
        fault_triggered = False

        for fault_state, lam in lambda_vals.items():
            if rng.random() < lam * dt:
                current_state = fault_state
                time_in_state = 0
                fault_triggered = True
                break

        if fault_triggered:
            continue

        # 2 Kontrola dokončení provozního stavu
        if time_in_state >= T[current_state]:

            idx = operational_sequence.index(current_state)
            next_idx = (idx + 1) % len(operational_sequence)
            current_state = operational_sequence[next_idx]
            time_in_state = 0

    # ===== PORUCHOVÉ STAVY =====
    else:

        if time_in_state >= T[current_state]:
```

```
            current_state = returns[current_state]
            time_in_state = 0

    state_history.append(T_S[current_state])
    samples.append(T_S[current_state])
```

Výpočet empirického stacionárního rozdělení

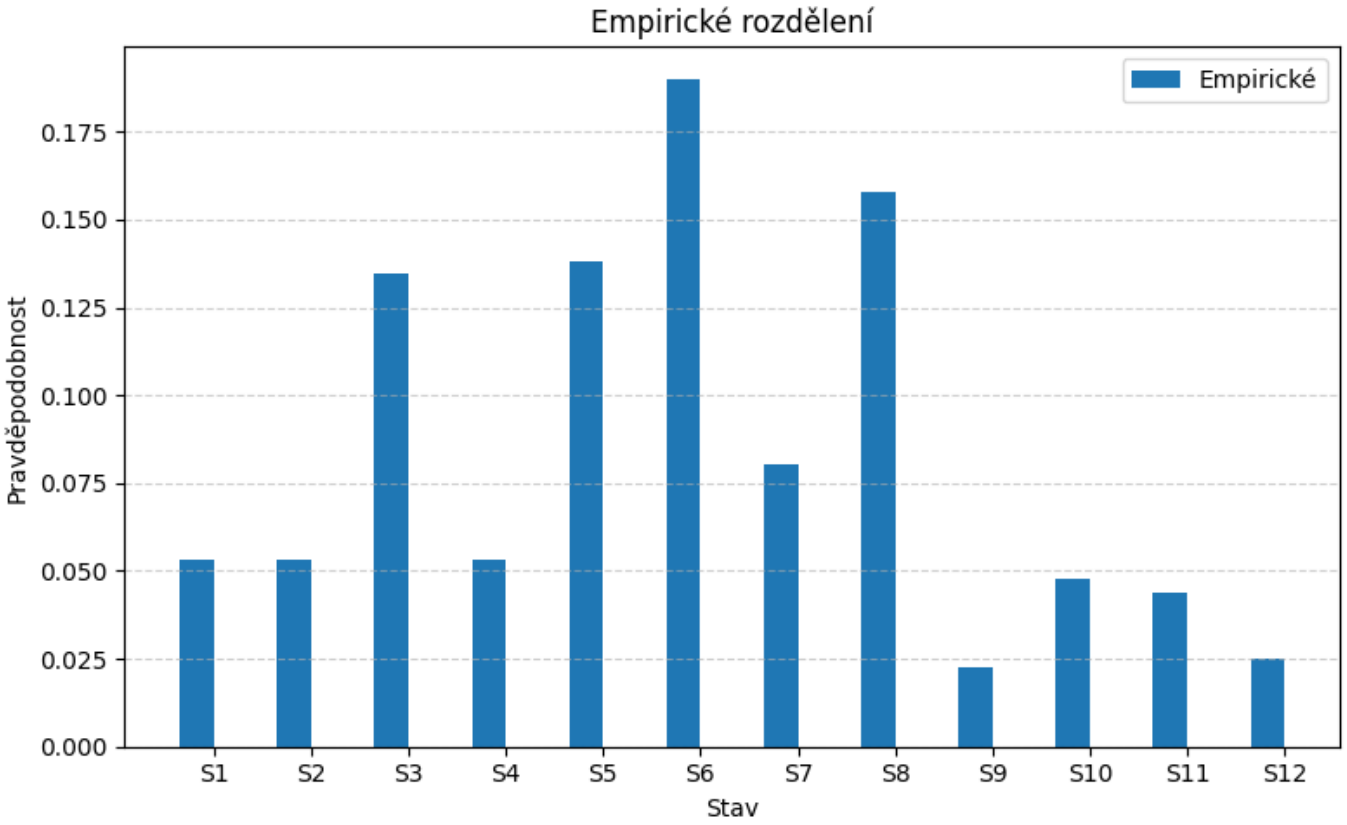
In [7]:
```
# Výpočet empirického stacionárního rozdělení π ze simulace
# jako podílu času stráveného v jednotlivých stavech
total_time = sum(state_time_counter.values())
pi_sim = np.array([state_time_counter[s]/total_time for s in T.keys()])

print("Stacionární rozdělení ze simulace:")

cycle_times = np.round(pi_sim, 8)
cycle_times
```

```
Stacionární rozdělení ze simulace:
```

Out[7]:
```
array([0.0533695, 0.05306  , 0.1346495, 0.0532855, 0.138159 , 0.189942 ,
       0.080186 , 0.1581135, 0.02253  , 0.047845 , 0.04376  , 0.0251   ])
```

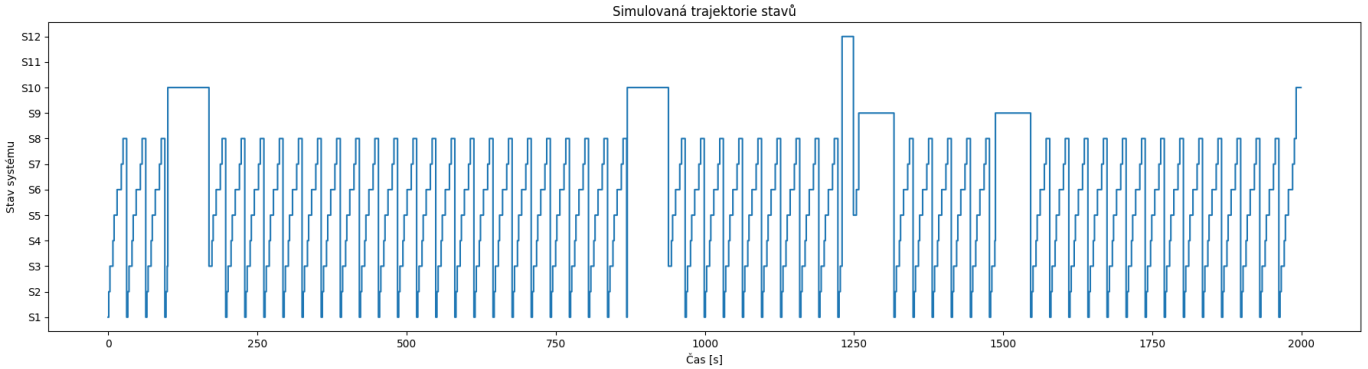
Sloupcový density graf

In [8]:
```
width = 0.35

states = np.arange(1,13)
states = np.arange(1,13)
state_labels = [f"S{i}" for i in states]

plt.figure(figsize=(8,5))

plt.bar(states - width/2, cycle_times, width, label='Empirické')

# --- osa X pro každý stav ---
plt.xticks(states, state_labels)

plt.xlabel("Stav")
plt.ylabel("Pravděpodobnost")
plt.title("Empirické rozdělení")

plt.grid(axis='y', linestyle='--', alpha=0.6)
plt.legend()
plt.tight_layout()
plt.show()
```



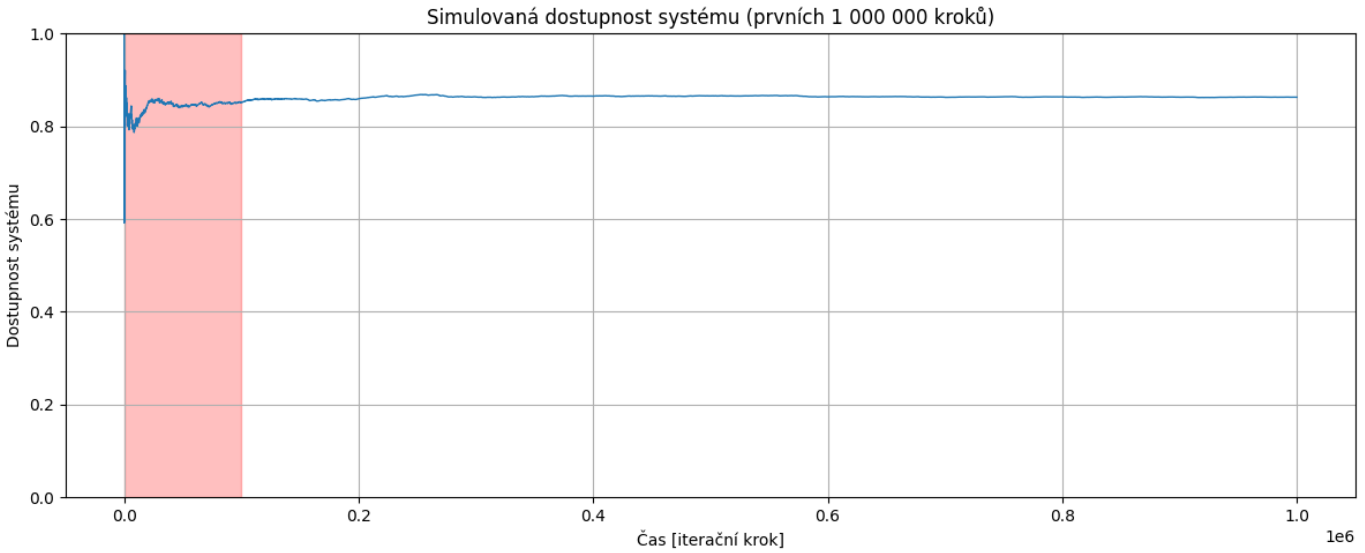
Ilustrační graf simulace

In [9]:
```
N = 2_000

# pořadí stavů
states = ["S1","S2","S3","S4","S5","S6",
          "S7","S8","S9","S10","S11","S12"]

# převod stavů na čísla 0-11
numeric_trajectory = [s-1 for s in state_history]

# časová osa (např. prvních N s pro přehlednost)
time_window = N
t = np.arange(time_window)

plt.figure(figsize=(18,5))
plt.step(t, numeric_trajectory[:time_window], where="post")

plt.yticks(range(len(states)), states)
plt.xlabel("Čas [s]")
plt.ylabel("Stav systému")
plt.title("Simulovaná trajektorie stavů")
plt.tight_layout()
plt.show()
```



Simulovaná dostupnost systému (prvních 1 000 000 kroků)

In [10]:
```
N = 1_000_000

# ===== Indikátor provozu =====
# S1–S8 jsou provozní stavy (index 1–8)
samples = np.array(samples)
operational = (samples < 9).astype(int)

# ===== Běžící dostupnost =====
availability_running = np.cumsum(operational[:N]) / np.arange(1, N+1)

# ===== Graf =====
plt.figure(figsize=(12,5))
plt.plot(availability_running, linewidth=1)

plt.axvspan(0, burn_in, color='red', alpha=0.25)

plt.xlabel("Čas [iterační krok]")
plt.ylabel("Dostupnost systému")
plt.title("Simulovaná dostupnost systému (prvních 1 000 000 kroků)")

plt.ylim(0, 1)
plt.grid(True)
plt.tight_layout()
plt.show()
```



Výpočet empirického stacionárního rozdělení po Burn-in

In [11]:
```
# Výpočet empirického stacionárního rozdělení π ze simulace po Burn-in
# jako podílu času stráveného v jednotlivých stavech
total_time = sum(state_time_counter_burn_in.values())
pi_sim = np.array([state_time_counter_burn_in[s]/total_time for s in T.keys()])

print("Stacionární rozdělení ze simulace po Burn_in:")

cycle_times = np.round(pi_sim, 8)
cycle_times
```

```
Stacionární rozdělení ze simulace po Burn_in:
```

Out[11]:
```
array([0.05340371, 0.05308898, 0.13469849, 0.05331055, 0.13826586,
       0.19008747, 0.08025636, 0.1582464 , 0.0225158 , 0.04752634,
       0.04336844, 0.02523159])
```

Export simulace formátu CSV

In [12]:
```
df_sim = pd.DataFrame({
    "time": np.arange(len(samples[burn_in:burn_in+1_000_000])),
    "state_index": samples[burn_in:burn_in+1_000_000],
})
```

In [13]:
```
df_sim
```

Out[13]:

	time	state_index
0	0	7
1	1	7
2	2	7
3	3	8
4	4	8
...	...	...
999995	999995	6
999996	999996	6
999997	999997	6
999998	999998	7
999999	999999	7

1000000 rows × 2 columns

In [14]:
```
df_sim.to_csv('../../data/03_StochModel/simulation_MCMC_samples.csv', index=False)
```

Autor / Organizace / Datum

Vjačeslav Usmanov, ČVUT v Praze, Fakulta stavební

Přehled změn

Datum (YYYY-MM-DD)	Verze	Autor změny	Popis změny
2026-01-25	1.1	Vjačeslav Usmanov	added SM_03_MCMC_samples.ipynb
2026-02-15	1.2	Vjačeslav Usmanov	changed SM_03_MCMC_samples.ipynb

Stochastic Model 04: Ověření modelu (Model Verification)

In [1]:
```
# Instalace potřebných knihoven
#%pip install pandas
#%pip install numpy
#%pip install seaborn matplotlib
```

In [2]:
```
# Import potřebných knihoven
import pandas as pd
import numpy as np

import seaborn as sns
import matplotlib.pyplot as plt

from scipy.stats import entropy
```

Vstupní data

In [3]:
```
# Soubor je načten a přiřazen do proměnné ‚df‘
other_path = '../../data/03_StochModel/transition_matrix.csv'
df = pd.read_csv(other_path, header=0)
P = df.to_numpy()
```

In [4]:
```
# analytické stacionární rozdělení (např. z P)
w, v = np.linalg.eig(P.T)

pi = np.real(v[:, np.isclose(w,1)])
pi = pi[:,0]
pi = pi / np.sum(pi)

pi_analyt = pi

# Empirické rozdělení
empirical_pi = np.array([
    0.05340371, 0.05308898, 0.13469849, 0.05331055,
    0.13826586, 0.19008747, 0.08025636, 0.1582464,
    0.0225158, 0.04752634, 0.04336844, 0.02523159
])

# empirické rozdělení ze simulace (např. z histogramu stavů)
pi_sim = empirical_pi
```

Maximální absolutní odchylka

In [5]:
```
# výpočet maximální absolutní odchylky mezi analytickým a simulovaným rozdělením

max_abs_dev = np.max(np.abs(pi_sim - pi_analyt))

print("Maximální absolutní odchylka:")
print(round(max_abs_dev, 6))
```

```
Maximální absolutní odchylka:
0.027344
```

Výpočet RMSE

In [6]:
```
# výpočet střední kvadratické chyby (Root Mean Square Error)

rmse = np.sqrt(
    np.mean(
        (pi_sim - pi_analyt)**2
    )
)

print("RMSE:")
print(round(rmse, 6))
```

```
RMSE:
0.011748
```

Výpočet KL divergence

In [7]:
```
# malá konstanta pro zamezení log(0)
epsilon = 1e-12

# výpočet Kullbackovy-Leiblerovy divergence
kl_div = np.sum(
    pi_analyt * np.log(
        (pi_analyt + epsilon) / (pi_sim + epsilon)
    )
)

print("KL divergence:")
print(round(kl_div, 6))
```

```
KL divergence:
0.027566
```

In [8]:
```
KL = entropy(pi_analyt, pi_sim)

print("KL divergence =", KL)
```

```
KL divergence = 0.027565786043595625
```

Shrnutí výsledků

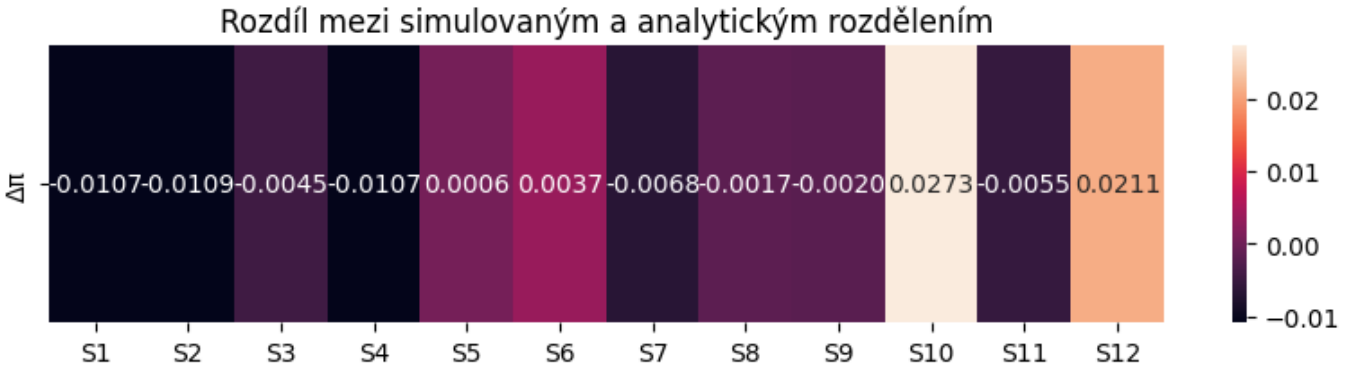
In [9]:
```
print("\n===== VERIFIKACE MODELU =====")
print(f"Max |pi_sim - pi_analyt| = {max_abs_dev:.6f}")
print(f"RMSE = {rmse:.6f}")
print(f"D_KL = {kl_div:.6f}")
```

```
===== VERIFIKACE MODELU =====
Max |pi_sim - pi_analyt| = 0.027344
RMSE = 0.011748
D_KL = 0.027566
```

Heatmapa rozdílu mezi analytickým a simulovaným rozdělením

In [10]:
```
states = np.arange(1,13)
width = 0.35

diff = (pi_sim - pi_analyt).reshape(1,-1)

plt.figure(figsize=(10,2))

sns.heatmap(diff,
            annot=True,
            fmt=".4f",
            xticklabels=[f"S{i}" for i in states],
```

```
            yticklabels=["Δπ"],
            cbar=True)

plt.title("Rozdíl mezi simulovaným a analytickým rozdělením")

plt.show()
```



Verifikace stochastického modelu pomocí porovnání analytického a simulovaného stacionárního rozdělení ukázala přijatelnou shodu (RMSE = 0.011748; max. odchylka 0.027344). Nízká hodnota KL divergence (0.027566) indikuje pouze minimální strukturální rozdíly způsobené diskretizací a konečnou délkou simulace.

Autor / Organizace / Datum

Vjačeslav Usmanov, ČVUT v Praze, Fakulta stavební

Přehled změn

Datum (YYYY-MM-DD)	Verze	Autor změny	Popis změny
2026-01-26	1.1	Vjačeslav Usmanov	added SM_04_Model_Verification.ipynb
2026-02-16	1.2	Vjačeslav Usmanov	changed SM_04_Model_Verification.ipynb