Appendix 3. Reproducible code (inline)

Copy each section into a file with the indicated name. The code expects two CSVs in data/: pacific_landmarks_os.csv and pacific_landmarks_pfs.csv with columns: time_months,value (0–1).

requirements.txt

```
# requirements.txt
numpy>=1.26
scipy>=1.11
pandas>=2.0
matplotlib>=3.8
```

utils_markov.py

```
# utils_markov.py
import numpy as np
from scipy.linalg import expm

def build_Q(rates):
    lam = rates['lambda_prog']
    mu0 = rates['mu0']
    mu1 = rates['mu1']
    return np.array([[-(lam + mu0),   lam,          mu0],
                     [0.0,             -mu1,         mu1],
                     [0.0,              0.0,         0.0]], dtype=float)

def softplus(x):
    return np.log1p(np.exp(x))

def rates_from_theta(theta):
    return {'lambda_prog': softplus(theta[0]), 'mu0': softplus(theta[1]),
'mu1': softplus(theta[2])}

def assemble_piecewise_Qs(theta_vec, knots):
    Qs = []
    for i in range(len(knots)):
        th = theta_vec[3*i:3*(i+1)]
        Qs.append(build_Q(rates_from_theta(th)))
    return Qs

def forward_piecewise(times, knots, Qs, dt=0.25):
    times = np.array(times, dtype=float)
    T_max = float(np.max(times))
    t = 0.0
    pi = np.array([1.0, 0.0, 0.0])  # start in pre-progression
    os_map, pfs_map = {}, {}
    while t <= T_max + 1e-9:
        idx = 0
        for k in range(len(knots)-1):
            if knots[k] <= t < knots[k+1]:
```

```
                idx = k
                break
        else:
            idx = len(knots)-1
        Q = Qs[idx]
        P_dt = expm(Q * (dt/12.0))  # months->years if rates are per-year
        pi = pi @ P_dt
        os_map[round(t, 4)] = 1.0 - pi[2]
        pfs_map[round(t, 4)] = pi[0]
        t += dt
    os_vals = np.array([os_map[min(os_map.keys(), key=lambda k: abs(k - tt))]
for tt in times])
    pfs_vals = np.array([pfs_map[min(pfs_map.keys(), key=lambda k: abs(k -
tt))] for tt in times])
    return os_vals, pfs_vals
```

01_landmarks_ingest.py

```
# 01_landmarks_ingest.py
import pandas as pd

def load_landmarks(os_path="data/pacific_landmarks_os.csv",
                   pfs_path="data/pacific_landmarks_pfs.csv"):
    os_df = pd.read_csv(os_path)
    pfs_df = pd.read_csv(pfs_path)
    required = {'time_months','value'}
    assert required.issubset(os_df.columns) and
required.issubset(pfs_df.columns), "CSV must contain time_months,value"
    os_df = os_df.sort_values('time_months').reset_index(drop=True)
    pfs_df = pfs_df.sort_values('time_months').reset_index(drop=True)
    return os_df, pfs_df

if __name__ == "__main__":
    os_df, pfs_df = load_landmarks()
    print(os_df.head(), pfs_df.head(), sep="\n")
```

02_ctmc_calibration.py

```
# 02_ctmc_calibration.py
import numpy as np
import pandas as pd
from scipy.optimize import minimize
from utils_markov import assemble_piecewise_Qs, forward_piecewise

def rmse(a, b):
    a = np.array(a); b = np.array(b)
    return float(np.sqrt(np.mean((a-b)**2)))

def weighted_loss(os_m, os_y, pfs_m, pfs_y, w_os=0.5, w_pfs=0.5):
    return w_os*rmse(os_m, os_y) + w_pfs*rmse(pfs_m, pfs_y)

def calibrate_ctmc(os_df, pfs_df, knots=(0,12,24,36), seed=2025, w_os=0.5,
w_pfs=0.5):
    rng = np.random.default_rng(seed)
```

```
    n_intervals = len(knots)
    theta0 = rng.normal(0.0, 0.5, size=3*n_intervals)

    t_os = os_df['time_months'].values
    y_os = os_df['value'].values
    t_pfs = pfs_df['time_months'].values
    y_pfs = pfs_df['value'].values
    times_grid = np.unique(np.concatenate([t_os, t_pfs]))

    def objective(theta):
        Qs = assemble_piecewise_Qs(theta, knots)
        os_model, pfs_model = forward_piecewise(times_grid, knots, Qs, dt=0.25)
        idx_os = [np.where(times_grid == t)[0][0] for t in t_os]
        idx_pfs = [np.where(times_grid == t)[0][0] for t in t_pfs]
        return weighted_loss(os_model[idx_os], y_os, pfs_model[idx_pfs], y_pfs,
w_os, w_pfs)

    res = minimize(objective, theta0, method='L-BFGS-B', options={'maxiter':
600, 'ftol': 1e-9})
    return res

if __name__ == "__main__":
    from importlib.machinery import SourceFileLoader
    load_landmarks =
SourceFileLoader('ing','01_landmarks_ingest.py').load_module().load_landmarks
    os_df, pfs_df = load_landmarks()
    res = calibrate_ctmc(os_df, pfs_df)
    print("Success:", res.success, "Loss:", res.fun)
    np.savetxt("results/theta_star.csv", res.x.reshape(1,-1), delimiter=",")
```

03_generate_figures.py

```
# 03_generate_figures.py
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from utils_markov import assemble_piecewise_Qs, forward_piecewise

def plot_os_pfs(os_df, pfs_df, theta_star, knots=(0,12,24,36),
out_prefix="results/"):
    times = np.arange(0, 72.01, 0.25)
    Qs = assemble_piecewise_Qs(theta_star, knots)
    os_vals, pfs_vals = forward_piecewise(times, knots, Qs, dt=0.25)

    # OS
    plt.figure()
    plt.plot(times, os_vals, label="CTMC OS")
    plt.scatter(os_df['time_months'], os_df['value'], s=30, label="Landmarks
OS")
    plt.xlabel("Months"); plt.ylabel("Overall survival"); plt.legend();
plt.tight_layout()
    plt.savefig(out_prefix + "fig3_os_model_vs_landmarks.pdf"); plt.close()

    # PFS
```

```
    plt.figure()
    plt.plot(times, pfs_vals, label="CTMC PFS")
    plt.scatter(pfs_df['time_months'], pfs_df['value'], s=30, label="Landmarks
PFS")
    plt.xlabel("Months"); plt.ylabel("Progression-free survival");
plt.legend(); plt.tight_layout()
    plt.savefig(out_prefix + "fig4_pfs_model_vs_landmarks.pdf"); plt.close()

if __name__ == "__main__":
    from importlib.machinery import SourceFileLoader
    load_landmarks =
SourceFileLoader('ing','01_landmarks_ingest.py').load_module().load_landmarks
    os_df, pfs_df = load_landmarks()
    th = pd.read_csv("results/theta_star.csv", header=None).values.flatten()
    plot_os_pfs(os_df, pfs_df, th)
```

04_uncertainty_bootstrap.py

```
# 04_uncertainty_bootstrap.py
import numpy as np
import pandas as pd
from scipy.optimize import minimize
from utils_markov import assemble_piecewise_Qs, forward_piecewise

def rmse(a, b):
    a = np.array(a); b = np.array(b)
    return float(np.sqrt(np.mean((a-b)**2)))

def weighted_loss(os_m, os_y, pfs_m, pfs_y, w_os=0.5, w_pfs=0.5):
    return w_os*rmse(os_m, os_y) + w_pfs*rmse(pfs_m, pfs_y)

def bootstrap_params(os_df, pfs_df, knots=(0,12,24,36), B=100, seed=1234,
w_os=0.5, w_pfs=0.5):
    rng = np.random.default_rng(seed)
    t_os, y_os = os_df['time_months'].values, os_df['value'].values
    t_pfs, y_pfs = pfs_df['time_months'].values, pfs_df['value'].values
    times_grid = np.unique(np.concatenate([t_os, t_pfs]))
    sd_os = max(1e-3, 0.02); sd_pfs = max(1e-3, 0.02)

    thetas = []
    for b in range(B):
        y_os_b = np.clip(y_os + rng.normal(0, sd_os, size=y_os.shape), 0.0,
1.0)
        y_pfs_b = np.clip(y_pfs + rng.normal(0, sd_pfs, size=y_pfs.shape), 0.0,
1.0)

        def objective(theta):
            Qs = assemble_piecewise_Qs(theta, knots)
            os_model, pfs_model = forward_piecewise(times_grid, knots, Qs,
dt=0.25)
            idx_os = [np.where(times_grid == t)[0][0] for t in t_os]
            idx_pfs = [np.where(times_grid == t)[0][0] for t in t_pfs]
            return weighted_loss(os_model[idx_os], y_os_b, pfs_model[idx_pfs],
y_pfs_b, w_os, w_pfs)
```

```
        theta0 = rng.normal(0.0, 0.5, size=3*len(knots))
        res = minimize(objective, theta0, method='L-BFGS-B',
options={'maxiter': 300, 'ftol': 1e-7})
        thetas.append(res.x)

    th = np.array(thetas)
    np.savetxt("results/theta_bootstrap.csv", th, delimiter=",")
    return th

if __name__ == "__main__":
    from importlib.machinery import SourceFileLoader
    load_landmarks =
SourceFileLoader('ing','01_landmarks_ingest.py').load_module().load_landmarks
    os_df, pfs_df = load_landmarks()
    th = bootstrap_params(os_df, pfs_df, B=50)
    print("Bootstrap shapes:", th.shape)
```

README.md (excerpt)

```
# README.md (excerpt)
## Reproducible pipeline
1) Create a virtual environment and install `requirements.txt`.
2) Place landmark CSVs under `data/`.
3) Calibrate:
   python 02_ctmc_calibration.py
4) Generate figures:
   python 03_generate_figures.py
5) Optional: bootstrap uncertainty:
   python 04_uncertainty_bootstrap.py

## Notes
- Time in months; rates treated per-year (months/12.0 in exponentials).
- 3-state structure (control → progression → death) is a pragmatic abstraction.
```

Appendix 4. Landmark tables (OS & PFS)

Landmarks derived from the manuscript's CTMC cohort state distributions (Table 4). Overall survival (OS) = 1 − death proportion; progression-free survival (PFS) = CR + PR + SD. Values are expressed as proportions (0–1).

A) Overall survival (OS) landmarks

time_months	value
12	0.9400
24	0.8100
36	0.6700
60	0.4300

B) Progression-free survival (PFS) landmarks

time_months	value
12	0.5800
24	0.3400
36	0.2000
60	0.0700

CSV files saved alongside the manuscript:

- OS: /mnt/data/data/pacific_landmarks_os.csv
- PFS: /mnt/data/data/pacific_landmarks_pfs.csv

These files can be used directly with the Appendix 3 scripts.