Supplementary Information

Software, validation scenarios, reference calculations and file inventory.

Online Resource 1

Contents

Software

Validation scenarios

Reference inputs

Reference calculations

Run times and metrics

File inventory

A computational framework for posterior probability vectors over discrete pharmacokinetic states
Journal of Pharmacokinetics and Pharmacodynamics
Jinming Zhang, Zheng Jiao
Corresponding author: Zheng Jiao
Shanghai Chest Hospital, Shanghai Jiao Tong University School of Medicine, Shanghai, China
Email: zjiao@sjtu.edu.cn

Reading guide

Online Resource 1 provides the software, reference inputs, execution examples, and comparison rules. Scenario-by-scenario results for OXC and TAC are presented in Online Resource 2 and Online Resource 3, respectively, and can be located via the scenario maps and tables of contents. For each scenario, results are presented across four metric categories; individual run times and numerical checks are compiled later in each volume, and summaries of the sensitivity scenarios are placed at the end of each volume. Code, inputs, saved outputs, and logs are listed in the scenario file index in Online Resource 4.

1. Software

`pmx-discrete-posterior` calculates Bayesian posterior probabilities for discrete pharmacokinetic states. Here, the states represent candidate dosing histories.

Component	Contents
Inputs	Model information, observation histories, candidate states and their prior probabilities
Output	Posterior probability for each candidate state

Software Information

Item	Details
Python import name	`pmx_discrete_posterior`
Package version used in this study	`0.2.0`
Python version	`3.12.0`
Main dependencies	NumPy `2.5.2`; SciPy `1.16.2`
Source repository	pmx-discrete-posterior

Installation and Reference Examples

In a new Python 3.12 environment, install the SHA-256-verified wheel from the 25 October 2026 release:

```
python -m pip install "numpy==2.5.2" "scipy==1.16.2" ./pmx_discrete_posterior-0.2.0-py3-none-any.whl
```

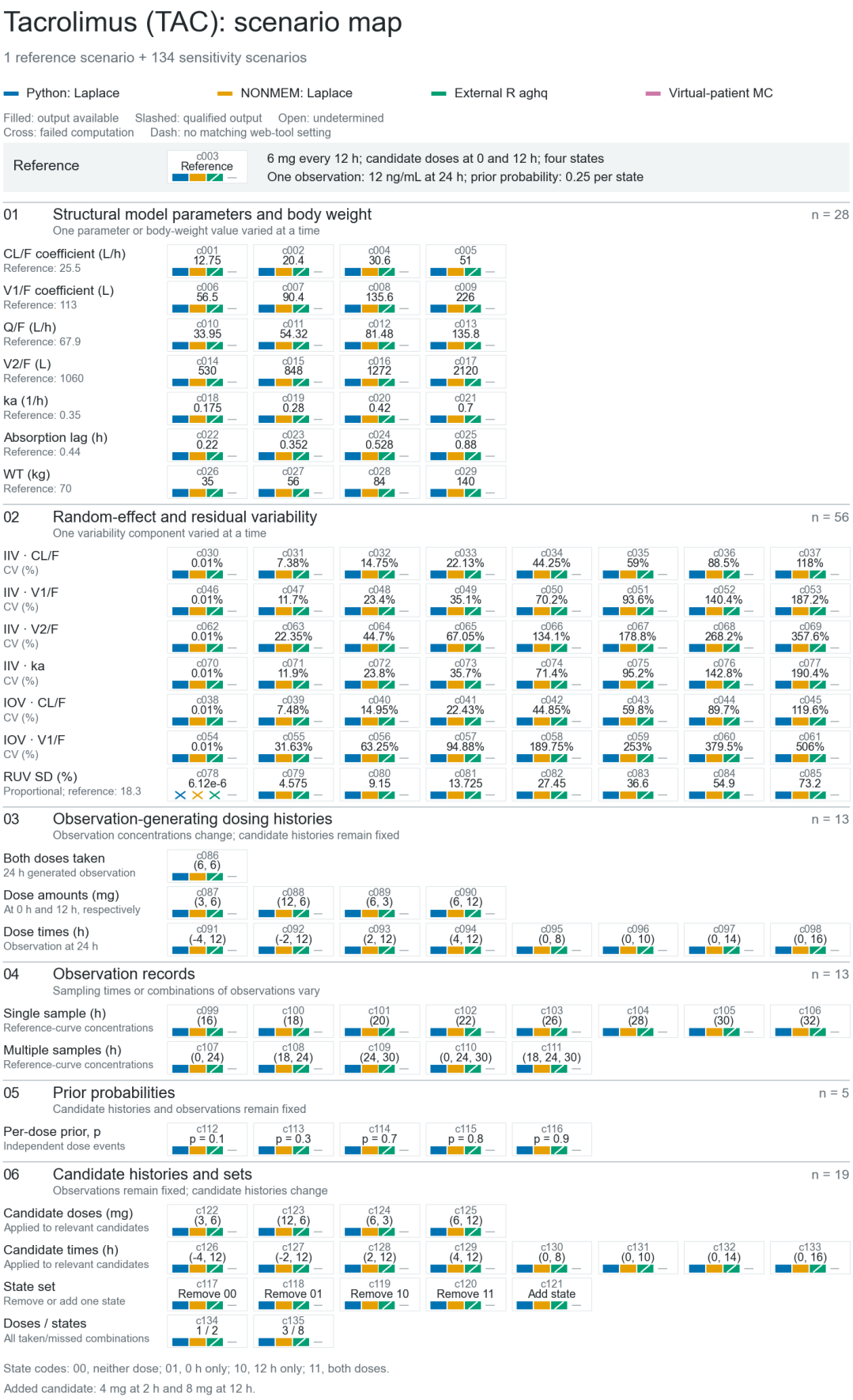
Material	Location
Wheel, source distribution, examples, inputs and checksums	Release files
Source code and version for each scenario	Source mapping
OXC and TAC reference settings	Section 3
Invocation code and example outputs	Section 4

Material	Location
Complete code, inputs and recorded outputs	Online Resource 4

2. Validation Scenarios

OXC comprises one reference and 82 sensitivity scenarios; TAC comprises one reference and 134 sensitivity scenarios. Scenario identifiers are case-sensitive.





The maps show the reference scenario and six sensitivity categories. Blue, orange, green and magenta identify Python-Laplace, NONMEM-Laplace, R aghq and virtual-patient MC. Filled indicators denote available outputs; slashes denote qualified outputs; crosses denote failed computations; open indicators denote undetermined coverage. A dash indicates no corresponding MC setting.

Model	Method	Designs	Displayed vectors	Comparison-eligible vectors	Failed designs
OXC	Python-Laplace	83	83	83	0
OXC	NONMEM-Laplace	83	83	83	0
OXC	R aghq	83	82	82	1
OXC	Virtual-patient MC	83	37	37	0
TAC	Python-Laplace	135	134	134	1
TAC	NONMEM-Laplace	135	134	134	1
TAC	R aghq	135	134	134	1
TAC	Virtual-patient MC	135	0	0	0

Counts include the reference scenario. R aghq outputs retain stability flags; see Section 5 for comparison rules. The MC count excludes seven captures with unmatched settings or totals differing from 100%; all 44 captures are retained.

3. Reference Inputs

For both models, the Python, NONMEM and R aghq calculations assume that residual errors are independent across observations conditional on the candidate state and all random effects. The joint observation likelihood is the product of the conditional densities.

3.1. OXC C003

Parameters

The one-compartment oral model predicts concentrations of the active oxcarbazepine metabolite, MHD. Population parameters are fixed. Clearance varies between individuals through a normally distributed log-scale random effect.

Input	Specification
Body weight	25 kg
Clearance coefficient, CLp	1.68 L/h
Volume of distribution, V	14.7 L
Absorption rate constant, ka	0.83 h^-1
Weight relationship	CL = CLp x (BW/70)^b x exp(eta); b = 0.624 - 0.233 x BW^219/(8.97^219 + BW^219)
Clearance random effect	eta follows a normal distribution with mean 0 and variance 0.0122457142458844; clearance CV = 11.1%
Residual error	Additive normal error with mean 0 and SD 2.992 mg/L

BW denotes body weight in kilograms; CV, coefficient of variation; SD, standard deviation.

Records

Input	Specification
Common dosing history	300 mg orally every 12 h at steady state, with the last common dose at -12 h
Observation	12 mg/L at 24 h

The steady-state initial amounts are calculated for each evaluated clearance random effect, immediately before the common -12 h dose.

Candidates

The four states combine taken or missed doses at 0 and 12 h; each dose taken is 300 mg.

Candidate state	Dose at 0 h (mg)	Dose at 12 h (mg)	Prior probability
omega_00	0	0	0.25
omega_01	300	0	0.25
omega_10	0	300	0.25
omega_11	300	300	0.25

For each history, the observation likelihood is integrated over the clearance random effect and weighted by the state prior. These four weights are normalized to posterior probabilities. Python and NONMEM use Laplace approximation; the R example uses adaptive Gauss-Hermite quadrature.

3.2. TAC c003

Input	Recorded setting
Observation	12.0 ng/mL at 24.0 h
Candidate dose	6.0 mg orally at 0 or 12 h
Shared dosing history	6.0 mg every 12.0 h; last shared dose at -12.0 h
Initialization	Conditional steady state before the shared -12 h dose, using the first occasion's parameters
Typical apparent clearance, CL/F	25.5 L/h
Typical apparent central volume, V1/F	113.0 L
Apparent intercompartmental clearance, Q/F	67.9 L/h
Typical apparent peripheral volume, V2/F	1060.0 L
Absorption rate and lag	0.35 1/h and 0.44 h
Covariates	WT = 70.0 kg; X = 0; HEM = 0.33; POD = 22.7 days; Pred = 155.5 nmol/L
Residual error	Conditional on the candidate state and random effects, the observed concentration is normally distributed with mean F and SD 0.183F, where F is the predicted concentration.
Random effects	Four interindividual effects and two interoccasion effects per occasion; ten coordinates in the source covariance matrix

WT denotes body weight; X is the CYP3A5 expresser indicator (0, nonexpresser); HEM denotes hematocrit; POD denotes postoperative day; Pred denotes peak free prednisolone concentration. Interindividual variability (IIV) affects CL/F, V1/F, V2/F and absorption rate. Interoccasion variability (IOV) affects CL/F and V1/F in each of the three occasions. The complete covariance matrix, coordinate order and fixed covariate coefficients are retained below. SD denotes standard deviation.

The unit of `theta_Pred` is L/nmol, not mL/ng as labeled in some archived code and output records.

The source random-effect vector follows a zero-mean multivariate normal distribution. The IIV effects on central volume, peripheral volume and absorption rate are correlated. IIV and IOV are independent, as are IOV effects within and across occasions; all other off-diagonal covariances are zero.

Candidate States

State	Machine bitstring in chronological order	Candidate dosing history	Prior	NONMEM SUBPOP
`omega_00`	`00`	Neither candidate dose	0.25	1
`omega_01`	`10`	6.0 mg at 0 h	0.25	2
`omega_10`	`01`	6.0 mg at 12 h	0.25	3
`omega_11`	`11`	6.0 mg at both 0 and 12 h	0.25	4

The shared -12 h dose is present in every candidate. Occasion changes at 0 and 12 h remain present when a candidate dose is omitted. State-event identifiers in the input below link the JSON posterior vectors to these labels.

Complete Input

Complete c003.json

Original source | Reader listing


```
{
  "schema": "tac_resolved_v1",
  "numeric_id": "tac_52001ebae53c0b4ac12eb02f752a2cc4fdffd3426b16241e73a582882aae4f84",
  "parameters": {
    "CL": 25.5,
    "V1": 113.0,
    "Q": 67.9,
    "V2": 1060.0,
    "ka": 0.35,
    "lag": 0.44,
    "WT": 70.0
  },
  "fixed": {
    "X": 0,
    "HEM": 0.33,
    "POD": 22.7,
    "Pred": 155.5,
    "theta_CYP3A5": 1.6,
    "theta_HEM": -1.01,
    "theta_POD": -0.0021,
    "theta_Pred": -0.0028,
    "POD_cap": 180,
    "WT_exponent": 0.75,
    "normalization_weight": 70
  },
  "slots": [
    0,
    12
  ],
  "shared_last": -12.0,
  "shared_dose": 6.0,
  "tau": 12.0,
  "states": [
    {
      "state_event_id": "state_944c7ce98f10ddebf79258da1fee55f539dacafb4c78ffb6746e1a1d71ee1681",
      "manuscript_id": "omega_00",
      "machine_bitstring": "00",
      "prior": 0.25,
      "events": []
    },
    {
      "state_event_id": "state_257522c31a5edf1ff5037764858097c7686f87db82abac1ed065432159708fee",
      "manuscript_id": "omega_01",
      "machine_bitstring": "10",
      "prior": 0.25,
      "events": [
        {
          "nominal_slot_h": 0.0,
          "time_h": 0.0,
          "amount_mg": 6.0
        }
      ]
    },
    {
      "state_event_id": "state_fc8bab0939921bb3816e4299c1228b86366b0eb2992227642bc76fb934151ed4",
      "manuscript_id": "omega_10",
      "machine_bitstring": "01",
      "prior": 0.25,
      "events": [
        {
          "nominal_slot_h": 12.0,
          "time_h": 12.0,
          "amount_mg": 6.0
        }
      ]
    },
    {
      "state_event_id": "state_f2f85477955a8a8c552c99078c66f34a36df9bcf2dedc540c807e4dd02899286",
      "manuscript_id": "omega_11",
      "machine_bitstring": "11",
      "prior": 0.25,
      "events": [
        {
          "nominal_slot_h": 0.0,
          "time_h": 0.0,
          "amount_mg": 6.0
        },
        {
```

```
        "nominal_slot_h": 12.0,
        "time_h": 12.0,
        "amount_mg": 6.0
      }
    ]
  }
],
"omega": [
  [
    0.08344460695463439,
    0.0,
    0.0,
    0.0,
    0.0,
    0.0,
    0.0,
    0.0,
    0.0,
    0.0
  ],
  [
    0.0,
    0.19805053857439603,
    -0.01671233240450257,
    0.13615474483258513,
    0.0,
    0.0,
    0.0,
    0.0,
    0.0,
    0.0
  ],
  [
    0.0,
    -0.01671233240450257,
    0.5873621303556349,
    -0.004502486522586959,
    0.0,
    0.0,
    0.0,
    0.0,
    0.0,
    0.0
  ],
  [
    0.0,
    0.13615474483258513,
    -0.004502486522586959,
    0.20422654773754803,
    0.0,
    0.0,
    0.0,
    0.0,
    0.0,
    0.0
  ],
  [
    0.0,
    0.0,
    0.0,
    0.0,
    0.0,
    0.08562800390349452,
    0.0,
    0.0,
    0.0,
    0.0,
    0.0
  ],
  [
    0.0,
    0.0,
    0.0,
    0.0,
    0.0,
    0.0,
    0.955597979744738,
    0.0,
    0.0,
    0.0,
    0.0
  ],
  [
    0.0,
    0.0,
    0.0,
    0.0,
    0.0,
    0.0,
    0.0,
    0.08562800390349452,
    0.0,
    0.0,
    0.0
```

```
    ],
    [
      0.0,
      0.0,
      0.0,
      0.0,
      0.0,
      0.0,
      0.0,
      0.955597979744738,
      0.0,
      0.0
    ],
    [
      0.0,
      0.0,
      0.0,
      0.0,
      0.0,
      0.0,
      0.0,
      0.0,
      0.08562800390349452,
      0.0
    ],
    [
      0.0,
      0.0,
      0.0,
      0.0,
      0.0,
      0.0,
      0.0,
      0.0,
      0.0,
      0.955597979744738
    ]
  ],
  "eta_order": [
    "Clearance IIV",
    "Central-volume IIV",
    "Peripheral-volume IIV",
    "Absorption-rate IIV",
    "Clearance IOV@occ_-12",
    "Central-volume IOV@occ_-12",
    "Clearance IOV@occ_0",
    "Central-volume IOV@occ_0",
    "Clearance IOV@occ_12",
    "Central-volume IOV@occ_12"
  ],
  "residual_sd": 0.183,
  "times": [
    24.0
  ],
  "observations": [
    12.0
  ]
}
```


4. Reference Calculations

Reproduction directory: runnable OXC and TAC examples for the three computational methods, installation requirements, launch commands and scenario-to-file mappings.

Material index: original code and recorded outputs.

Result summaries and figure reconstruction: rebuild tables and figures from saved outputs without rerunning the models.

4.1. Python-Laplace

OXC C003

Dashboard

Select **OXC manuscript** and **Reset reference** to load C003. The **Parameters**, **Records** and **Candidates** tabs contain the inputs in Section 3.1. Select **Compute** to obtain the posterior probabilities; **Result JSON** and **Probability CSV** export the output.

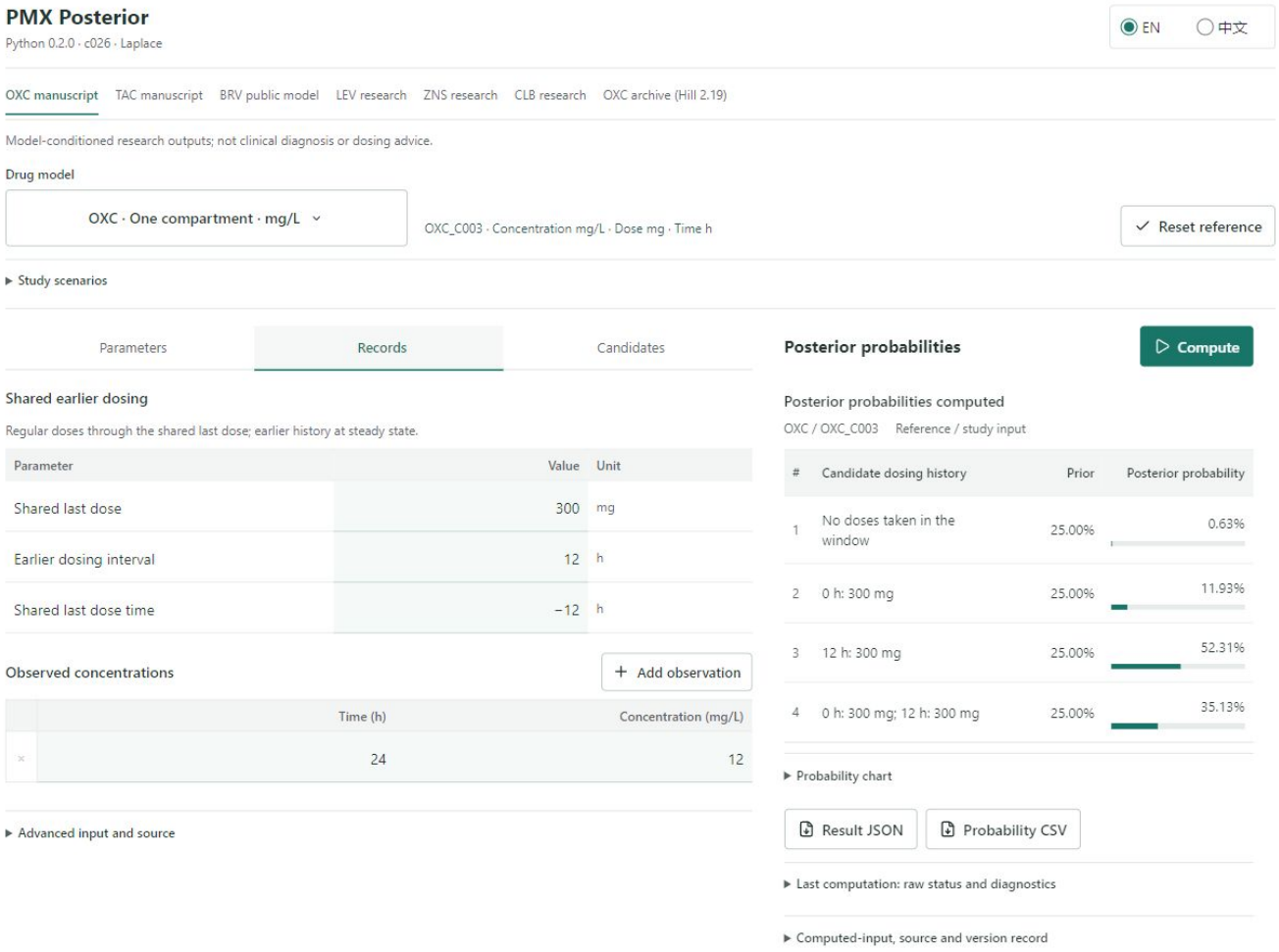


Figure PY1. Dosing and observation records with Python-Laplace posterior probabilities for C003.

Complete Python Example

The script below specifies the same reference inputs, calls `compute_posterior` and writes the posterior probabilities and numerical diagnostics to `output/result.json`.

The recorded computational environment used Python 3.12.0, `pmx-discrete-posterior` 0.2.0, NumPy 2.5.2 and SciPy 1.16.2. The script uses the package's default Laplace settings. The source hash manifest identifies the archived build.

From the directory containing `c003_laplace.py`, run:

```
python c003_laplace.py
```

Listing 1. Complete Python example.

c003_laplace.py

```
"""Calculate OXC C003 posterior probabilities using the Python package."""

# 1. Dependencies
import json
import math
from pathlib import Path

from scipy.special import expit
from pmx_discrete_posterior import compute_posterior
from pmx_discrete_posterior.interfaces.validation import InputError, scalar
from pmx_discrete_posterior.models.prediction.linear import LinearPredictor


# 2. Model-specific parameter and concentration-prediction interfaces
def oxc_relations(*, parameters, covariates, iiv_values, iov_values,
                  occasions, evaluation_interval):
    p = parameters["values"]
    bw = scalar(covariates["BW"], positive=True)
    exponent = p["k0"] - p["kmax"] * expit(
        p["gamma"] * math.log(bw / p["k50"]))
    cl = (p["CL_p"] * (bw / p["normalization_weight"]) ** exponent
          * math.exp(iiv_values["CL"]))
```

```
    values = {"CL": scalar(cl, positive=True),
              "V": scalar(p["V_p"], positive=True),
              "ka": scalar(p["ka"], positive=True)}
    return {
        "units": {"CL": "L/h", "V": "L", "ka": "1/h"},
        "segments": [{
            "start": evaluation_interval[0],
            "end": max(evaluation_interval[1], evaluation_interval[0] + 1),
            "occasion_id": None, "values": values,
        }],
    }


def oxc_system(*, values, units):
    if units != {"CL": "L/h", "V": "L", "ka": "1/h"}:
        raise InputError("OXC units differ")
    cl, volume, ka = (scalar(values[k], positive=True)
                      for k in ("CL", "V", "ka"))
    return {
        "state_names": ["depot", "central"],
        "amount_unit": "mg", "time_unit": "h",
        "concentration_unit": "mg/L",
        "matrix": [[-ka, 0], [ka, -cl / volume]],
        "observation_vector": [0, 1 / volume],
        "inputs": {("oral", "depot"): {"fractions": [1, 0], "lag": 0}},
    }


def dose(time):
    return {"time": time, "amount": 300.0, "amount_unit": "mg",
            "route": "oral", "compartment": "depot"}


# 3. Population parameters, random effects and residual error
def build_inputs():
    values = {
        "CL_p": 1.68, "V_p": 14.7, "ka": 0.83,
        "k0": 0.624, "kmax": 0.233, "k50": 8.97, "gamma": 219.0,
        "normalization_weight": 70.0,
        "omega_0_0": 0.0122457142458844, "sigma": 2.992,
    }
    units = {name: "1" for name in values}
    units.update(CL_p="L/h", V_p="L", ka="1/h", k50="kg",
                 normalization_weight="kg", sigma="mg/L")
    model = {
        "parameters": {"values": values, "units": units},
        "parameter_model": {"covariate_names": ["BW"],
                            "relations": oxc_relations},
        "iiv": {"names": ["CL"], "distribution": "normal",
                "covariance_parameters": [["omega_0_0"]]},
        "iov": None,
        "random_effects_dependence": "independent_blocks",
        "residual_model": {
            "distribution": "normal", "error_type": "additive",
            "parameter_names": {"sd": "sigma"},
            "dependence": "conditionally_independent",
        },
        "predict": LinearPredictor(oxc_system),
    }

    # 4. Shared history: steady state immediately before the -12 h dose
    periodic_dose = {"offset": 0.0, "amount": 300.0, "amount_unit": "mg",
                     "route": "oral", "compartment": "depot"}
    individual_data = {
        "covariates": {"BW": 25.0}, "occasions": [],
        "dosing_prehistory": {
            "events": [dose(-12.0)],
            "regimen": {"period": 12.0, "anchor_time": -12.0,
                        "time_unit": "h", "events": [periodic_dose]},
        },
        "initialization": {
            "mode": "steady_state", "start_time": -12.0,
            "event_phase": "before_events",
            "steady_state": {"parameter_rule": "fixed_reference",
                             "reference_occasion_id": None},
        },
    }

    # 5. Observation, candidate histories and state priors
    observations = [{"record_id": "0", "time": 24.0, "value": 12.0,
                     "time_unit": "h", "value_unit": "mg/L"}]
    candidate_states = [
        {"state_id": "omega_00", "events": []},
        {"state_id": "omega_01", "events": [dose(0.0)]},
        {"state_id": "omega_10", "events": [dose(12.0)]},
        {"state_id": "omega_11", "events": [dose(0.0), dose(12.0)]},
    ]
    priors = {state["state_id"]: 0.25 for state in candidate_states}
    return dict(model=model, individual_data=individual_data,
                observations=observations, candidate_states=candidate_states,
                priors=priors)
```

```
# 6. Laplace calculation and complete output, including diagnostics
def jsonable(value):
    """Preserve nonfinite diagnostic settings without invalid JSON literals."""
    if isinstance(value, dict):
        return {str(key): jsonable(item) for key, item in value.items()}
    if isinstance(value, (tuple, list)):
        return [jsonable(item) for item in value]
    if hasattr(value, "tolist"):
        return jsonable(value.tolist())
    if isinstance(value, float) and not math.isfinite(value):
        return {"nonfinite_diagnostic": repr(value)}
    return value


def main():
    result = compute_posterior(
        **build_inputs(), integration_method="laplace", numerical_options=None)
    output = Path(__file__).resolve().parent / "output"
    output.mkdir(exist_ok=True)
    (output / "result.json").write_text(
        json.dumps(jsonable(result), indent=2, allow_nan=False) + "\n",
        encoding="utf-8")
    if result["status"] != "success":
        raise RuntimeError(result.get("failure_reason", "Calculation failed"))
    print("State\tPosterior probability")
    for state in result["state_results"]:
        print(f"{state['state_id']}\t{state['posterior_probability']:.12g}")


if __name__ == "__main__":
    main()
```

TAC c003

The recorded driver mapped the source JSON with `build_inputs` and called `compute_posterior(**inputs, integration_method="laplace", numerical_options=None)`. Its recorded environment was Python 3.12.0, `pmx-discrete-posterior` 0.2.0, NumPy 2.5.2 and SciPy 1.16.2. The actual default optimizer tolerance was checked as `gtol=1e-6`; the complete resolved options are provided below.

Recorded Invocation

```
& "D:\AI-ready PMX benchmark engine\rerun_c026_full_20261021\runtime\.venv\Scripts\python.exe" -B -X utf8 "D:\AI-ready
  PMX benchmark engine\rerun_c026_full_20261021\02_laplace_tac\run_python_case.py" TAC "D:\AI-ready PMX benchmark
  engine\rerun_c026_full_20261021\02_laplace_tac\inputs\TAC\c003.json" "D:\AI-ready PMX benchmark
  engine\rerun_c026_full_20261021\02_laplace_tac\runs\formal\TAC\c003\r01"
```

Original execution record.

Complete Driver

run_python_case.py

Original source | Reader listing

```
"""Function: one fresh released-package calculation
Inputs: model, frozen input, output directory
Outputs: complete posterior/diagnostics or explicit failure
"""
import argparse
from pathlib import Path
import sys
import time
import traceback
import warnings

ROOT = Path(__file__).resolve().parent
sys.path.insert(0, str(ROOT / "shared"))
from run_support import environment, now, read, verify_seal, write
from shared.scenario_inputs import build_inputs
from pmx_discrete_posterior import compute_posterior
from pmx_discrete_posterior.interfaces.inputs.numerical_options import validate_numerical_options


def main():
    """Function: dispatch one scenario
    Inputs: CLI paths Outputs: immutable result and process exit
    """
    parser = argparse.ArgumentParser()
    parser.add_argument("model", choices=("OXC", "TAC"))
    parser.add_argument("input")
    parser.add_argument("output")
    args = parser.parse_args()
    out = Path(args.output)
```

```
    out.mkdir(parents=True, exist_ok=True)
    started = time.monotonic()
    verify_seal(ROOT)
    write(out / "actual_environment.json", environment())
    try:
        # Record installed defaults without an override.
        resolved = validate_numerical_options(None)
        write(out / "numerical_options.json", {"requested": None, "resolved": resolved})
        if resolved["optimization"]["backend_options"]["gtol"] != 1e-6:
            raise ValueError("installed default gtol differs from the approved 1e-6")
        with warnings.catch_warnings(record=True) as recorded:
            warnings.simplefilter("always")
            inputs = build_inputs(read(args.input), args.model)
        write(out / "input_mapping_warnings.json", [
            {"category": item.category.__name__, "message": str(item.message)} for item in recorded])
        mapped = {key: value for key, value in inputs.items() if key != "model"}
        mapped["model"] = {key: value for key, value in inputs["model"].items()
                           if key not in ("predict", "parameter_model")}
        mapped["model"]["parameter_model"] = {"covariate_names": inputs["model"]["parameter_model"]["covariate_names"],
                                              "relations_source": "shared/reference_models.py"}
        mapped["model"]["predict_source"] = "shared/reference_models.py"
        write(out / "mapped_scientific_inputs.json", mapped)
        result = compute_posterior(**inputs, integration_method="laplace", numerical_options=None)
        # Check actual optimizer options when used.
        for state in result.get("state_results", []):
            optimization = state.get("method_details", {}).get("optimization")
            if optimization is not None and optimization["backend_options"]["gtol"] != 1e-6:
                raise ValueError("actual optimizer gtol differs from the recorded default")
    except Exception as exc:
        result = {"status": "failed", "failure_reason": f"{type(exc).__name__}: {exc}",
                  "state_results": [], "traceback": traceback.format_exc()}
    verify_seal(ROOT)
    result["rerun"] = {"model": args.model, "input": str(Path(args.input).resolve()),
                       "ended_utc": now(), "elapsed_seconds": time.monotonic() - started}
    write(out / "result.json", result)
    print(f"{args.model} {result['status']} {result['rerun']['elapsed_seconds']:.3f}s", flush=True)
    return 0 if result["status"] == "success" else 1


if __name__ == "__main__":
    raise SystemExit(main())
```

Required Shared Model Adapters

The driver requires `shared/scenario_inputs.py`, `shared/reference_models.py`, `shared/run_support.py`, the source seal and the recorded `pmx_discrete_posterior` package. The first two adapters are in the OXC shared listings linked below, including their TAC parameter relations, compartment system, occasions, initialization and package-interface mapping. The dependency manifest lists the remaining package modules and support files individually.

Required file	Complete shared listing	Byte-identical original
`shared/scenario_inputs.py`	OXC shared listing	Original
`shared/reference_models.py`	OXC shared listing	Original

Additional execution helper: shared/run_support.py.

Recorded Numerical Options

Resolved numerical_options.json

Original source | Reader listing


```
{
  "requested": null,
  "resolved": {
    "optimization": {
      "method": "BFGS",
      "stationarity_tolerance": 1e-05,
      "backend_options": {
        "maxiter": 1000,
        "gtol": 1e-06,
        "xrtol": 0.0,
        "c1": 0.0001,
        "c2": 0.9
      }
    },
    "derivatives": {
      "backend": "scipy",
      "atol": 1e-06,
      "rtol": 1e-06,
      "backend_options": {
        "initial_step": 0.5,
```

```
        "step_factor": 2.0,
        "maxiter": 10,
        "order": 8
      }
    }
  }
}
```

Outputs and Runtime

Run	Recorded status	omega_00	omega_01	omega_10	omega_11
1	valid	0.20711803268279594	0.2564854354109252	0.2726783493908101	0.2637181825154687
2	valid	0.20711803268279594	0.2564854354109252	0.2726783493908101	0.2637181825154687
3	valid	0.20711803268279594	0.2564854354109252	0.2726783493908101	0.2637181825154687
4	valid	0.20711803268279594	0.2564854354109252	0.2726783493908101	0.2637181825154687
5	valid	0.20711803268279594	0.2564854354109252	0.2726783493908101	0.2637181825154687

Run	Time (s)	Sources
1	74.3440000000046	result; receipt
2	61.18800000000192	result; receipt
3	56.0309999999954	result; receipt
4	59.90600000000268	result; receipt
5	57.04700000000594	result; receipt

Times are copied from the controller receipts, without rounding or recomputation. The driver's internal `rerun.elapsed_seconds` is a separate timer retained in each full result. The recorded environment, input mapping, warnings, commands, receipts and complete state diagnostics are linked in the source manifest.

4.2. NONMEM-Laplace

OXC C003

This NONMEM 7.5.0 example shows saved formal run 1. Images are rendered file previews, not fresh application screenshots.

Control Stream

model.mod:

```
$PROBLEM OXC C003 category 1 formal native SS
$INPUT ID TIME AMT RATE II SS EVID MDV CMT DV WT AGE HT
$DATA data.csv IGNORE=@
$SUBROUTINES ADVAN2 TRANS2

$PK
CALLFL=-1
WTLOG=THETA(6)*LOG(WT/THETA(7))
IF (WTLOG.GE.0) THEN
 WTRAT=1/(1+EXP(-WTLOG))
ELSE
 WTNEG=EXP(WTLOG)
 WTRAT=WTNEG/(1+WTNEG)
ENDIF
WTEXP=THETA(4)-THETA(5)*WTRAT
CL=THETA(1)*(WT/70)**WTEXP*EXP(ETA(1))
V=THETA(2)
KA=THETA(3)
S2=V

; F1 gates candidate doses; the shared -12 h SS dose stays enabled.
F1=1
IF (EVID.EQ.1.AND.TIME.EQ.0) THEN
 IF (MIXNUM.EQ.1.OR.MIXNUM.EQ.3) F1=0
ENDIF
IF (EVID.EQ.1.AND.TIME.EQ.12) THEN
 IF (MIXNUM.EQ.1.OR.MIXNUM.EQ.2) F1=0
ENDIF

$ERROR
```

```
IPRED=F
ET1=ETA(1)
MX=MIXNUM
; Preserve the original EPS reference and explicit likelihood.
POPIND=EPS(1)
F_FLAG=2
Y=LOG(8.952064)+(DV-IPRED)**2/(8.952064)-(0)

$MIX NSPOP=4
P(1)=0.25
P(2)=0.25
P(3)=0.25
P(4)=0.25
$THETA 1.6799999999999999 FIX
$THETA 14.699999999999999 FIX
$THETA 0.82999999999999996 FIX
$THETA 0.624 FIX
$THETA 0.23300000000000001 FIX
$THETA 219 FIX
$THETA 8.9700000000000006 FIX
$OMEGA 0.012245714245884399 FIX
$SIGMA 8.952064 FIX
$ESTIMATION PRINT=20 METHOD=1 INTERACTION LAPLACIAN POSTHOC NOABORT MAXEVAL=0 OPTMAP=1 FORMAT=s1PE24.16
$TABLE ID TIME AMT II SS EVID CMT DV MDV WT AGE HT IPRED ET1 MX NOPR ONEHEADER FILE=model.fit FORMAT=s1PE24.16
```

NONMEM C003: model.mod

File preview | Saved formal run 1

```
$PROBLEM OXC C003 category 1 formal native SS
$INPUT ID TIME AMT RATE II SS EVID MDV CMT DV WT AGE HT
$DATA data.csv IGNORE=@
$SUBROUTINES ADVAN2 TRANS2

$PK
CALLFL=-1
WTLOG=THETA(6)*LOG(WT/THETA(7))
IF (WTLOG.GE.0) THEN
 WTRAT=1/(1+EXP(-WTLOG))
ELSE
 WTNEG=EXP(WTLOG)
 WTRAT=WTNEG/(1+WTNEG)
ENDIF
WTEXP=THETA(4)-THETA(5)*WTRAT
CL=THETA(1)*(WT/70)**WTEXP*EXP(ETA(1))
V=THETA(2)
KA=THETA(3)
S2=V

; F1 gates candidate doses; the shared -12 h SS dose stays enabled.
F1=1
IF (EVID.EQ.1.AND.TIME.EQ.0) THEN
 IF (MIXNUM.EQ.1.OR.MIXNUM.EQ.3) F1=0
ENDIF
IF (EVID.EQ.1.AND.TIME.EQ.12) THEN
 IF (MIXNUM.EQ.1.OR.MIXNUM.EQ.2) F1=0
ENDIF

$ERROR
IPRED=F
ET1=ETA(1)
MX=MIXNUM
; Preserve the original EPS reference and explicit likelihood.
POPIND=EPS(1)
F_FLAG=2
Y=LOG(8.952064)+(DV-IPRED)**2/(8.952064)-(0)

$MIX NSPOP=4
P(1)=0.25
P(2)=0.25
P(3)=0.25
P(4)=0.25
$THETA 1.6799999999999999 FIX
$THETA 14.699999999999999 FIX
$THETA 0.82999999999999996 FIX
$THETA 0.624 FIX
$THETA 0.23300000000000001 FIX
$THETA 219 FIX
$THETA 8.9700000000000006 FIX
$OMEGA 0.012245714245884399 FIX
$SIGMA 8.952064 FIX
$ESTIMATION PRINT=20 METHOD=1 INTERACTION LAPLACIAN POSTHOC NOABORT MAXEVAL=0 OPTMAP=1 FORMAT=s1PE24.16
$TABLE ID TIME AMT II SS EVID CMT DV MDV WT AGE HT IPRED ET1 MX NOPR ONEHEADER FILE=model.fit FORMAT=s1PE24.16
```

Figure NM0. C003 control stream, saved formal run 1.

Dataset

data.csv:

```
@ID,TIME,AMT,RATE,II,SS,EVID,MDV,CMT,DV,WT,AGE,HT
1,-12,300,0,12,1,1,1,1,0,25,0,0
1,0,300,0,0,0,1,1,1,0,25,0,0
1,12,300,0,0,0,1,1,1,0,25,0,0
1,24,0,0,0,0,0,0,2,12,25,0,0
```

The `SS=1` record establishes the common predose history. `F1` switches the two candidate doses by `MIXNUM`; `AGE` and `HT` are unused, zero-valued input columns.

NONMEM C003: data.csv

File preview | Saved formal run 1

```
@ID,TIME,AMT,RATE,II,SS,EVID,MDV,CMT,DV,WT,AGE,HT
1,-12,300,0,12,1,1,1,1,0,25,0,0
1,0,300,0,0,0,1,1,1,0,25,0,0
1,12,300,0,0,0,1,1,1,0,25,0,0
1,24,0,0,0,0,0,0,2,12,25,0,0
```

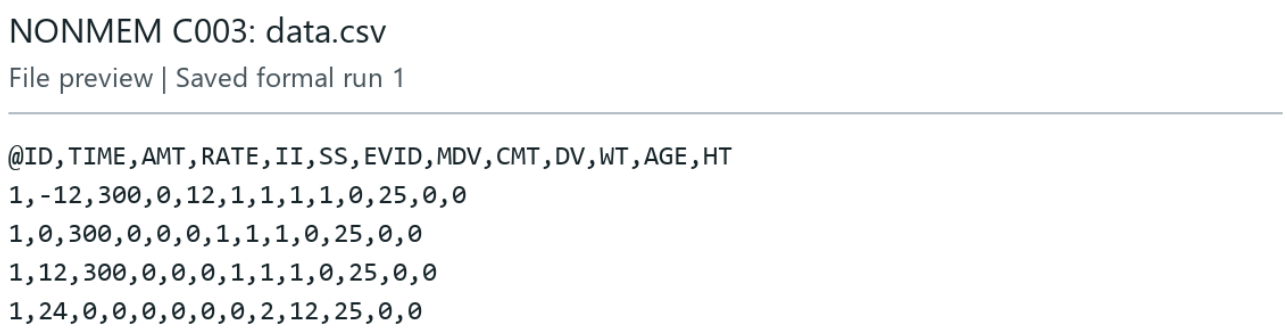


Figure NM1. Data file for C003.

Native Posterior Output

model.phm:

```
TABLE NO.     1: Laplacian Conditional Estimation with Interaction (Evaluation): Problem=1 Subproblem=0 Superproblem1=0
  Iteration1=0 Superproblem2=0 Iteration2=0
 SUBJECT_NO              ID                      SUBPOP                  PMIX                    ETA(1)
        ETC(1,1)                OBJ
            1                                  1            1              6.2965600911769372E-03
  -1.2189457950464128E-01   1.4102276677589252E-02    11.333190172487161
            1                                  1            2              1.1928606268061692E-01
  -1.0159238353455496E-01   1.0098757025019984E-02    5.4501481135574839
            1                                  1            3              5.2307287690581028E-01
  -9.9071748015784195E-03   9.3587522679364245E-03    2.4937555098004665
            1                                  1            4              3.5134450032239589E-01
  5.2070317423414086E-02   7.5892663620020992E-03    3.2896626574378427
```

NONMEM C003: model.phm

File preview | Saved formal run 1

```
TABLE NO.     1: Laplacian Conditional Estimation with Interaction (Evaluation): Problem=1 Subproblem=0 Superproblem1=0 Iteration1=0 Superproblem2=0 Iteration2=0
 SUBJECT_NO              ID                      SUBPOP                  PMIX                    ETA(1)                  ETC(1,1)                OBJ
            1                                  1            1              6.2965600911769372E-03  -1.2189457950464128E-01   1.4102276677589252E-02    11.333190172487161
            1                                  1            2              1.1928606268061692E-01  -1.0159238353455496E-01   1.0098757025019984E-02    5.4501481135574839
            1                                  1            3              5.2307287690581028E-01  -9.9071748015784195E-03   9.3587522679364245E-03    2.4937555098004665
            1                                  1            4              3.5134450032239589E-01   5.2070317423414086E-02   7.5892663620020992E-03    3.2896626574378427
```

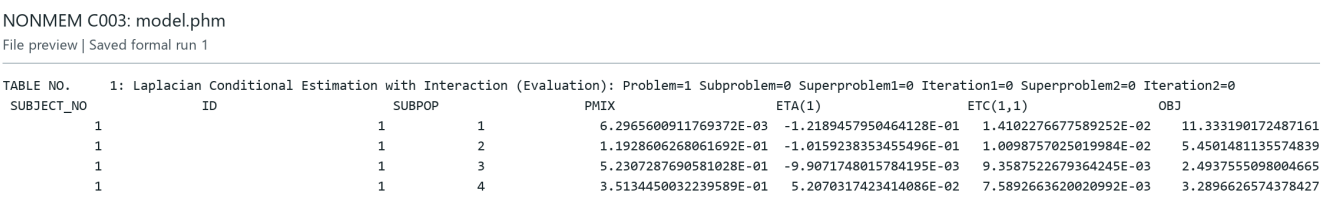


Figure NM2. Native posterior output for C003 (one run).

SUBPOP	State label
1	`omega_00`
2	`omega_01`
3	`omega_10`
4	`omega_11`

`PMIX` contains probabilities, not percentages, and matches model.shm exactly. With `F_FLAG=2`, `PRED`, `RES`, and `WRES` are not concentration residual diagnostics. NM-TRAN warnings: inferred population data (2); automatically supplied `NUMERICAL` and `SLOW` (66).

Invocation

Keep `model.mod` and `data.csv` together in a separate working directory. Set `NMFE` to the local licensed NONMEM launcher and run in PowerShell:

```
& $env:NMFE model.mod model.lst -prdefault
```

TAC c003

The original listing identifies NONMEM 7.5.0 and PREDPP 7.5.0. The control stream used `$MIX NSPOP=4` and `METHOD=1 INTERACTION LAPLACIAN POSTHOC MAXEVAL=0`. Posterior probabilities are the native `PMIX` values in `run.phm`, with `SUBPOP` mapped to the candidate table above. The control-stream file is named `run.mod` in the source.

Complete Control Stream

run.mod

Original source | Reader listing

```
$PROBLEM TAC reference: native SS, four dosing histories, three occasions
; Dose: mg. TIME: h. DV: ng/mL. WT: kg. Volumes: L. Flows: L/h.
; Shared SS dose at -12 h; candidate doses at 0/12 h; observation at 24 h.
$INPUT ID TIME AMT RATE II SS EVID MDV CMT DV OCC DOSEEQ WT HEM POD PRDF X
; At lagged dose times, use OCC from NEXT and DOSEEQ from the DOSE record.
$BIND - - - - - - - - - - NEXT DOSE - - - - -
$DATA data.csv IGNORE=@
$SUBROUTINES ADVAN4 TRANS4 SS4

$PK
CALLFL=-2

; 1. Typical parameters and covariates. X=0: CYP3A5 nonexpresser; X=1: expresser.
; POD is postoperative day, a fixed covariate rather than the TIME column.
PODC=POD
IF (POD.GT.180) PODC=180
TVCLF=THETA(1)*THETA(7)**X*(1+THETA(8)*(HEM-0.33))
TVCLF=TVCLF*(WT/70)**0.75
TVCLF=TVCLF*(1+THETA(9)*(PODC-22.7))
TVV1F=THETA(2)*(1+THETA(10)*(PRDF-155.5))

; 2. Occasion-specific IOV. Switches remain even when a dose is omitted.
KCL=0
KV1=0
; OCC 1: shared prehistory, before the 0 h switch.
IF (OCC.EQ.1) THEN
  KCL=ETA(5)
  KV1=ETA(6)
ENDIF
; OCC 2: after the 0 h switch, before the 12 h switch.
IF (OCC.EQ.2) THEN
  KCL=ETA(7)
  KV1=ETA(8)
ENDIF
; OCC 3: after the 12 h switch, through the 24 h observation.
IF (OCC.EQ.3) THEN
  KCL=ETA(9)
  KV1=ETA(10)
ENDIF

; 3. Table notation: CL/F, V1/F (central), Q/F, V2/F (peripheral).
; ETA(1:4) are shared IIV; CL/F and V1/F also include the current IOV.
CLF=TVCLF*EXP(ETA(1)+KCL)
V1F=TVV1F*EXP(ETA(2)+KV1)
QF=THETA(3)
V2F=THETA(4)*EXP(ETA(3))
KA=THETA(5)*EXP(ETA(4))
ALAG1=THETA(6)

CL=CLF                        ; ADVAN4 TRANS4 clearance
V2=V1F                        ; ADVAN4 TRANS4 central volume
Q=QF                          ; ADVAN4 TRANS4 intercompartmental clearance
V3=V2F                        ; ADVAN4 TRANS4 peripheral volume
S2=V1F/1000                   ; Central concentration in ng/mL

; 4. DOSEEQ=0 keeps the shared SS dose; 1/2 identify the 0/12 h doses.
; F1 controls whether each candidate receives a dose, including after ALAG1.
F1=1
IF (DOSEEQ.EQ.1) THEN
  F1=0
  IF (MIXNUM.EQ.2) F1=1
  IF (MIXNUM.EQ.4) F1=1
ENDIF
IF (DOSEEQ.EQ.2) THEN
  F1=0
  IF (MIXNUM.EQ.3) F1=1
  IF (MIXNUM.EQ.4) F1=1
ENDIF

$ERROR
IPRED=F
; Proportional residual SD = 0.183 (18.3%).
Y=F*(1+EPS(1))
```

```
$MIX NSPOP=4
; SUBPOP 1: neither dose; 2: only 0 h; 3: only 12 h; 4: both doses.
P(1)=0.25
P(2)=0.25
P(3)=0.25
P(4)=0.25

; Fixed population parameters. Numeric literals match the archived model.
$THETA 25.5 FIX                 ; 1: typical CL/F
$THETA 113 FIX                  ; 2: typical V1/F (central)
$THETA 67.900000000000006 FIX    ; 3: Q/F
$THETA 1060 FIX                 ; 4: typical V2/F (peripheral)
$THETA 0.34999999999999998 FIX   ; 5: typical absorption rate
$THETA 0.44 FIX                 ; 6: absorption lag (h)
$THETA 1.6000000000000001 FIX   ; 7: CYP3A5 expresser multiplier on CL/F
$THETA -1.01 FIX                ; 8: HEM coefficient on CL/F
$THETA -0.0020999999999999999 FIX ; 9: POD coefficient on CL/F
$THETA -0.0028 FIX              ; 10: PRDF coefficient on V1/F

$OMEGA 0.08344460695463439 FIX   ; IIV for CL/F
$OMEGA BLOCK(3) FIX             ; Correlated IIV for V1/F, V2/F, KA
0.19805053857439603             ; IIV for V1/F
-0.01671233240450257 0.58736213035563489 ; IIV for V2/F
0.13615474483258513 -0.0045024865225869589 0.20422654773754803 ; IIV for KA
$OMEGA 0.085628003903494523 FIX  ; OCC1 IOV for CL/F
$OMEGA 0.95559797974473804 FIX   ; OCC1 IOV for V1/F
$OMEGA 0.085628003903494523 FIX  ; OCC2 IOV for CL/F
$OMEGA 0.95559797974473804 FIX   ; OCC2 IOV for V1/F
$OMEGA 0.085628003903494523 FIX  ; OCC3 IOV for CL/F
$OMEGA 0.95559797974473804 FIX   ; OCC3 IOV for V1/F
$SIGMA 0.033488999999999998 FIX  ; proportional residual variance

$ESTIMATION METHOD=1 INTERACTION LAPLACIAN POSTHOC MAXEVAL=0
NUMERICAL SLOW FORMAT=s1PE23.16
SIGDIGITS=4 SIGL=12 SIGLO=12
; Observations are identified by EVID=0 and MDV=0; no RID/QID are needed.
$TABLE ID TIME EVID MDV AMT DV IPRED OCC DOSEEQ CLF V1F V2F KA SS II F1
ETA(1) ETA(2) ETA(3) ETA(4) ETA(5) ETA(6)
ETA(7) ETA(8) ETA(9) ETA(10)
NOPRINT ONEHEADER NOAPPEND FORMAT=s1PE23.16 FILE=run.tab
```

Complete Dataset

The units are mg for AMT, h for TIME, and ng/mL for DV. The records retain both occasion-boundary rows and the candidate dose rows.

data.csv

Original source | Reader listing

```
ID,TIME,AMT,RATE,II,SS,EVID,MDV,CMT,DV,OCC,DOSEEQ,WT,HEM,POD,PRDF,X
1,-12.0,6.0,0,12.0,1,1,1,1,0,1,0,70.0,0.33,22.7,155.5,0
1,0.0,0,0,0,0,2,1,0,0,1,0,70.0,0.33,22.7,155.5,0
1,0.0,0,0,0,0,2,1,0,0,2,0,70.0,0.33,22.7,155.5,0
1,0.0,6.0,0,0,0,1,1,1,0,2,1,70.0,0.33,22.7,155.5,0
1,12.0,0,0,0,0,2,1,0,0,2,0,70.0,0.33,22.7,155.5,0
1,12.0,0,0,0,0,2,1,0,0,3,0,70.0,0.33,22.7,155.5,0
1,12.0,6.0,0,0,0,1,1,1,0,3,2,70.0,0.33,22.7,155.5,0
1,24.0,0,0,0,0,0,0,2,12.0,3,0,70.0,0.33,22.7,155.5,0
```

TAC c003: complete NONMEM dataset

```
ID,TIME,AMT,RATE,II,SS,EVID,MDV,CMT,DV,OCC,DOSEEQ,WT,HEM,POD,PRDF,X
1,-12.0,6.0,0,12.0,1,1,1,1,0,1,0,70.0,0.33,22.7,155.5,0
1,0.0,0,0,0,0,0,2,1,0,0,1,0,70.0,0.33,22.7,155.5,0
1,0.0,0,0,0,0,0,2,1,0,0,2,0,70.0,0.33,22.7,155.5,0
1,0.0,6.0,0,0,0,0,1,1,1,0,2,1,70.0,0.33,22.7,155.5,0
1,12.0,0,0,0,0,0,2,1,0,0,2,0,70.0,0.33,22.7,155.5,0
1,12.0,0,0,0,0,0,2,1,0,0,3,0,70.0,0.33,22.7,155.5,0
1,12.0,6.0,0,0,0,0,1,1,1,0,3,2,70.0,0.33,22.7,155.5,0
1,24.0,0,0,0,0,0,0,0,2,12.0,3,0,70.0,0.33,22.7,155.5,0
```

Text preview of all dataset records; the selectable listing above preserves the source file.

Complete Native Posterior Output

The complete `.phm` listing includes ETA, covariance, objective and `PMIX` columns. `PMIX` values are probabilities, not percentages.

run.phm

Original source | Reader listing

```
TABLE NO.     1: Laplacian Conditional Estimation with Interaction (Evaluation): Problem=1 Subproblem=0 Superproblem1=0
  Iteration1=0 Superproblem2=0 Iteration2=0
 SUBJECT_NO              ID                    SUBPOP                  PMIX                    ETA(1)
   ETA(2)                  ETA(3)                  ETA(4)                  ETA(5)                  ETA(6)
     ETA(7)                  ETA(8)                  ETA(9)                  ETA(10)                 ETC(1,1)
       ETC(2,1)                ETC(2,2)                ETC(3,1)                ETC(3,2)                ETC(3,3)
         ETC(4,1)                ETC(4,2)                ETC(4,3)                ETC(4,4)                ETC(5,1)
           ETC(5,2)                ETC(5,3)                ETC(5,4)                ETC(5,5)                ETC(6,1)
             ETC(6,2)                ETC(6,3)                ETC(6,4)                ETC(6,5)                ETC(6,6)
               ETC(7,1)                ETC(7,2)                ETC(7,3)                ETC(7,4)                ETC(7,5)
                 ETC(7,6)                ETC(7,7)                ETC(8,1)                ETC(8,2)
 ETC(8,3)                ETC(8,4)                ETC(8,5)                ETC(8,6)                ETC(8,7)
   ETC(8,8)                ETC(9,1)                ETC(9,2)                ETC(9,3)                ETC(9,4)
     ETC(9,5)                ETC(9,6)                ETC(9,7)                ETC(9,8)                ETC(9,9)
       ETC(10,1)               ETC(10,2)               ETC(10,3)               ETC(10,4)               ETC(10,5)
         ETC(10,6)               ETC(10,7)               ETC(10,8)               ETC(10,9)               ETC(10,10)
           OBJ
           1                                 1           1            2.0711804476327172E-01
 -6.2249991005755366E-02 -1.0768969120695080E-04  9.3113765302045862E-02 -1.1961354492827777E-03
 -4.2986199855052594E-02  3.3821124233109658E-02 -5.5783047514311491E-03  4.3290267003886010E-03
 -1.5314308744883843E-02 -1.2570930334375412E-02  4.2039505640931257E-02 -4.9085252736406188E-06
  1.9960881781561599E-01  5.9272354539398428E-02 -1.8085655120385555E-02  4.7259679703455454E-01
 -6.7512584881513466E-04  1.3753122327448267E-01 -4.0051870300568870E-03  2.0566197373004533E-01
 -2.8641937497255653E-02 -1.1402004006962856E-04  3.8594540853127975E-02 -6.6990063999977677E-04
  6.5285668894525153E-02  2.1039864212569415E-02  7.2801873499964177E-03 -4.8139225150965491E-02
  5.2269665422600255E-03  1.5398318622268011E-02  9.7581415351425516E-01 -3.5306918030644559E-03
 -5.1341193375582788E-05  7.3410668201290916E-03 -2.6593659575790457E-05 -2.3414404748623116E-03
  1.1314228614712945E-03  8.5099624258897999E-02  2.5067717733977140E-03  9.5966940476195769E-04
 -7.7996429700017645E-03  6.2187913243836050E-04  1.7229621147002141E-03 -7.0735513139435569E-04
  7.9526462046522158E-04  9.5857387534093708E-01 -1.0315870705105883E-02  1.6032530232306904E-04
  1.4887656852904283E-02  3.7039531427657797E-06 -6.7076019880180386E-03  5.0606481766009674E-03
 -7.5325515794224607E-04  5.4136672639339516E-05  8.2503067061760027E-02 -7.2087800303025276E-03
 -2.7966783772645598E-03  1.6770733267569109E-02 -1.9905783470863794E-03 -6.1025021799361850E-03
  4.3835094100911649E-03 -6.9585222666600157E-04  9.7731877761901673E-04 -5.9904927492194615E-04
  9.4059636082128895E-01    4.2826383596866462
           1                                 1           2            2.5648541606976905E-01
  1.7785162599159179E-04  9.5202520045144483E-07 -1.6831460369051473E-04  7.5147188058509104E-06
  1.0798205782936541E-04 -8.8316204590633524E-05  2.3713767702915898E-05 -1.9444026425635459E-05
  5.0809434908717757E-05  3.6431942135953681E-05  3.6785897839891481E-02 -2.4990513490737387E-04
  1.9804427990236803E-01  4.4159135556843025E-02 -1.6473494628414694E-02  5.4563722272110438E-01
 -1.9718372177557190E-03  1.3613919838727459E-01 -2.6374846891917617E-03  2.0413718262837929E-01
 -2.8328207044797708E-02 -1.5132083788714886E-04  2.6824450992060843E-02 -1.1963861819982696E-03
  6.8429944838237167E-02  2.3171458830404702E-02  1.0584735380757616E-04 -2.1889342475404805E-02
  9.6678995377971614E-04  1.4066698935017009E-02  9.4401428833551826E-01 -6.2217698172238713E-03
 -3.3115002570779741E-05  5.8785751342265895E-03 -2.6291771919130540E-04 -3.7776023410500091E-03
```

```
 3.0911557994183626E-03  8.4799200004804551E-02  5.1019693516039218E-03  2.3123343002225674E-05
-4.8091272193889418E-03  2.1293828261616064E-04  3.0974551938073203E-03 -2.5345923229383245E-03
6.7821400266031772E-04  9.5502576878164847E-01 -1.3329604376727968E-02 -7.2007835457348068E-05
1.2611576636151964E-02 -5.6412400301055334E-04 -8.0937826708979456E-03  6.6199121789204769E-03
-1.7781632434500714E-03  1.4597909130415220E-03  8.1821563096822217E-02 -9.5609254463541130E-03
-4.2551058702666453E-05  9.0306016530240993E-03 -3.9780231447958311E-04 -5.8012861328451473E-03
4.7448796553618954E-03 -1.2744404105701800E-03  1.0431144875636888E-03 -2.7353793686613697E-03
9.5368076912898758E-01    3.8550725391198806
          1                                   1            3             2.7267835663387596E-01
3.4263786856522628E-02  2.6827286547421547E-03 -2.3244377035226707E-02  6.5670855594822707E-03
2.0136434173412446E-02 -1.6451682121466738E-02  3.0505536571172353E-03 -2.5064005631677926E-03
1.1973339262794543E-02 -1.0078378672072359E-03  3.6408824529763566E-02 -3.6465055517803688E-03
1.9614807030884493E-01  3.1750325829997206E-02 -1.3522494728680225E-02  5.7078266231131103E-01
-8.9496618907367945E-03  1.3378898862517510E-01  1.8863468878252869E-03  2.0045969622715709E-01
-2.7619844195040565E-02 -2.0065779836429564E-03  2.1459055968536856E-02 -4.9656008302767151E-03
6.9621588584356187E-02  2.2876368036440779E-02 -1.6150245722142426E-03 -8.1620090427673535E-03
1.9320843780380784E-03  1.3182823130988333E-02  9.3058295724901241E-01 -4.2174383208907441E-03
-3.1173597228129996E-04  1.6535451396450405E-03 -8.2364130998312940E-04 -2.5343737936420821E-03
2.3194915232148693E-03  8.5375370089596919E-02  3.5246050073201914E-03 -2.5694347015981352E-04
8.0089768683453151E-05  3.4179173379506435E-04  2.0915176973801044E-03 -1.8850578068392010E-03
-5.1186213838175847E-06  9.5357225480484142E-01 -1.6429229994589400E-02 -1.4236054043535314E-03
9.4684982495588377E-03 -3.3945950294522265E-03 -9.8017511175319018E-03  7.9726324934635707E-03
-1.5407834728154162E-03  1.5304302828228690E-03  8.0111424119549163E-02  9.0903999697201227E-04
-3.7948116358652587E-04 -4.9022065044705279E-03  3.4178343826819617E-04  8.2845368093985880E-04
-7.3087995288318277E-04  1.4751347366391025E-04 -4.0239226729806666E-04 -4.3141695919607306E-05
9.5028409796466085E-01    3.7326303107410292
          1                                   1            4             2.6371818253308321E-01
1.0987313809512293E-01  8.7219139209422890E-03 -8.5832168497448624E-03  2.2801093452252178E-02
5.5935024140274771E-02 -4.3144485412161496E-02  1.3906573041149830E-02 -1.0773955433413399E-02
4.2906462172648027E-02 -5.6545402260795271E-03  3.4358701230184703E-02 -3.7136630179520954E-03
1.9250597290378427E-01  2.3575174909733899E-03 -1.5116861843926068E-02  5.9671263365652005E-01
-9.8004588357484632E-03  1.2963660002544622E-01 -4.1675292610577541E-03  1.9473997747393496E-01
-2.5356192306486135E-02 -1.5143384400087766E-03  1.4179644618582810E-02 -4.1041976261001816E-03
7.3018721183924357E-02  1.9298433271404809E-02 -7.1061259516101739E-03  1.3610372779602441E-02
-2.3557925811343609E-03  9.8630295731206019E-03  9.1226119619109958E-01 -6.1017164883226827E-03
-3.6966898362244719E-04 -5.1993855391266651E-03 -1.2074368470538007E-03 -3.2880825004316244E-03
2.8873553973887663E-03  8.5384885104949237E-02  4.6779536634577882E-03 -1.8842162225413835E-03
1.0332084089830072E-02 -5.3047768921829497E-04  2.5210523138032557E-03 -1.9022234739965163E-03
-6.5462638781000314E-04  9.4742930538167358E-01 -1.8912370173486404E-02 -1.9268266850876279E-03
-6.5610551006758597E-03 -4.7452614809574481E-03 -1.0122292790422384E-02  7.0530077786744045E-03
-2.7301716463024716E-03  2.9339303540812140E-03  7.9073240389245195E-02  1.6664446427246795E-03
-1.4192404561964270E-03 -9.7312218795738703E-03  1.1828991935502998E-03  1.4712757522254614E-03
-1.3892989314945097E-03  5.6640518158077599E-04 -1.4508742240570826E-03 -3.2763236955846343E-04
9.3668806313955788E-01    3.7994540651177573
```

TAC c003: native PMIX columns

SUBJECT_NO	ID	SUBPOP	PMIX
1	1	1	2.0711804476327172E-01
1	1	2	2.5648541606976905E-01
1	1	3	2.7267835663387596E-01
1	1	4	2.6371818253308321E-01

Text preview of SUBJECT_NO, ID, SUBPOP and PMIX. The complete `.phm` listing above retains all columns.

run.shm

Original source | Reader listing

```
TABLE NO.     1: Laplacian Conditional Estimation with Interaction (Evaluation): Problem=1 Subproblem=0 Superproblem1=0
  Iteration1=0 Superproblem2=0 Iteration2=0
 SUBJECT_NO             ID                   SUBPOP                PMIX                   ETS(1)    ETS(2)
  ETS(3)    ETS(4)    ETS(5)    ETS(6)    ETS(7)    ETS(8)    ETS(9)    ETS(10)
          1                                   1           1            2.0711804476327172E-01  1.010000  1.010000
  1.010000  1.010000  1.010000  1.010000  1.010000  1.010000  1.010000  1.010000
          1                                   1           2            2.5648541606976905E-01  1.010000  1.010000
  1.010000  1.010000  1.010000  1.010000  1.010000  1.010000  1.010000  1.010000
          1                                   1           3            2.7267835663387596E-01  1.010000  1.010000
```

```
 1.010000  1.010000  1.010000  1.010000  1.010000  1.010000  1.010000  1.010000
         1                                 1           4            2.6371818253308321E-01  1.010000  1.010000
 1.010000  1.010000  1.010000  1.010000  1.010000  1.010000  1.010000  1.010000
```

Recorded Invocation and Timing

```
cmd.exe /d /c nmfe75_no_retry.bat run.mod run.lst -prdefault -locfile=nmloc.bat
```

Original execution record.

Run	Recorded status	omega_00	omega_01	omega_10	omega_11
1	valid	2.0711804476327172E-01	2.5648541606976905E-01	2.7267835663387596E-01	2.6371818253308321E-01
2	valid	2.0711804476327172E-01	2.5648541606976905E-01	2.7267835663387596E-01	2.6371818253308321E-01
3	valid	2.0711804476327172E-01	2.5648541606976905E-01	2.7267835663387596E-01	2.6371818253308321E-01
4	valid	2.0711804476327172E-01	2.5648541606976905E-01	2.7267835663387596E-01	2.6371818253308321E-01
5	valid	2.0711804476327172E-01	2.5648541606976905E-01	2.7267835663387596E-01	2.6371818253308321E-01

Run	Time (s)	Sources
1	1.5779999999940628	run.phm; timing
2	1.5789999999979045	run.phm; timing
3	1.562000000005355	run.phm; timing
4	1.562000000005355	run.phm; timing
5	1.562000000005355	run.phm; timing

Times are copied from `command.result.json.elapsed_seconds`, which records the wrapper and its descendant-process deadline scope. The original wrapper, location file, command results, stdout, `.lst` and native tables are individually linked in the source manifest. The recorded NM-TRAN population-data warning is retained; the scenario record states that optimizer stationarity was not established.

4.3. R aghq

OXC C003

R performs optimization, numerical differentiation and adaptive Gauss-Hermite quadrature over the clearance random effect. Python supplies the model likelihood, random-effect density and posterior normalization.

Comments in the listings are translated into English. The following call records the original environment; use the portable R instructions to run a reference or sensitivity scenario on another computer.

Call and Settings

The four arguments are input JSON, model name, settings JSON and output directory. Configure the local executable and library paths to match the source-pinned runtime.

The recorded environment used R 4.4.3, aghq 0.4.1, reticulate 1.46.0, numDeriv 2016.8.1.1, mvQuad 1.0.10, jsonlite 2.0.0 and data.table 1.18.2.1; Python 3.12.0 with pmx-discrete-posterior 0.2.0, NumPy 2.5.2 and SciPy 1.16.2.

```
$workspace = 'D:/AI-ready PMX benchmark engine'
$root = "$workspace/outputs/supplement_intake_20261022/c003/code/r_formal"
$env:PMX_RERUN_ROOT = $root
$env:RETICULATE_PYTHON = "$workspace/rerun_c026_full_20261021/runtime/.venv/Scripts/python.exe"
$env:PMX_R_LIBRARY = "$workspace/aghq_reference_validation_20261020/r_library"
$env:R_LIBS_USER = $env:PMX_R_LIBRARY
$env:PYTHONDONTWRITEBYTECODE = '1'
$env:RETICULATE_USE_MANAGED_VENV = 'no'
$env:OMP_NUM_THREADS = '1'
$env:OPENBLAS_NUM_THREADS = '1'
$env:MKL_NUM_THREADS = '1'
$env:NUMEXPR_NUM_THREADS = '1'
Push-Location -LiteralPath $root
try {
    New-Item -ItemType Directory -Path 'tmp' -Force | Out-Null
    $env:TMPDIR = 'tmp'
    $env:TMP = 'tmp'
    $env:TEMP = 'tmp'
    if (Test-Path -LiteralPath 'c003_manual_001') { throw 'Choose a fresh output name' }
```

```
    & 'C:/Program Files/R/R-4.4.3/bin/Rscript.exe' --vanilla `
        'r_external/run_case.R' 'inputs/OXC/c003.json' 'OXC' `
        'settings/OXC.json' 'c003_manual_001'
    if ($LASTEXITCODE -ne 0) { throw "Rscript failed: $LASTEXITCODE" }
} finally {
    Pop-Location
}
```

Complete Settings

OXC.json

```
{
  "acceptance_mode": "external_comparison",
  "orders": [
    7,
    15,
    31,
    61
  ],
  "log_integral_atol": 1e-06,
  "posterior_atol": 1e-06,
  "max_nodes": 1000,
  "max_evaluations": 20000,
  "curvature_atol": 0.001,
  "stationarity_atol": 0.001,
  "construction": "product",
  "rule": "GHe",
  "derivative_policy": "library_default",
  "optimizer": "aghq_BFGS",
  "coordinate_system": "prior_standardized"
}
```

The integral uses prior-standardized coordinates and includes the Jacobian once. Orders 7, 15, 31 and 61 are evaluated; the last order is returned, without selecting the closest agreement with another route. `external_comparison` retains order-sensitivity warnings, but actual integration failures still prevent posterior normalization. These checks do not certify absolute quadrature error.

r_external/run_case.R

run_case.R

```
# Function: Run one explicitly configured external case
# Inputs: raw JSON, OXC/TAC, settings JSON, output directory
# Outputs: result.json and actual_environment.json

# Function: Resolve an explicit environment/config value
# Inputs: Environment name, config field, optional default
# Outputs: One nonempty path; conflicting values fail
runtime_value <- function(name, configured = NULL, default = NULL) {
  value <- Sys.getenv(name, unset = "")
  if (!is.null(configured) && (length(configured) != 1L || !is.character(configured) || !nzchar(configured)))
    stop(paste("invalid runtime field for", name))
  if (nzchar(value) && !is.null(configured) &&
      !identical(normalizePath(value, winslash = "/", mustWork = TRUE),
                 normalizePath(configured, winslash = "/", mustWork = TRUE)))
    stop(paste("conflicting environment and config paths:", name))
  if (!nzchar(value)) value <- if (!is.null(configured)) configured else default
  if (is.null(value) || length(value) != 1L || !nzchar(value)) stop(paste("explicit path required:", name))
  normalizePath(value, winslash = "/", mustWork = TRUE)
}

# Function: Verify and bind the installed runtime once
# Inputs: Explicit root, Python, R library
# Outputs: Actual paths and exact pinned versions
initialize_external_runtime <- function(root, python, r_library) {
  locale <- if (.Platform$OS.type == "windows") "English_United States.utf8" else "C.UTF-8"
  if (!nzchar(Sys.setlocale("LC_CTYPE", locale)) || !isTRUE(l10n_info()[["UTF-8"]]))
    stop("a UTF-8 LC_CTYPE locale is required to source the archived bilingual bridge")
  expected <- normalizePath("D:/AI-ready PMX benchmark
  engine/rerun_c026_full_20261021/runtime/.venv/Scripts/python.exe", winslash = "/", mustWork = TRUE)
  if (!identical(python, expected)) stop("Python must be the installed .venv belonging to the explicit root")
  .libPaths(c(r_library, .libPaths()))
  Sys.setenv(RETICULATE_PYTHON = python, PYTHONDONTWRITEBYTECODE = "1",
             OMP_NUM_THREADS = "1", OPENBLAS_NUM_THREADS = "1", MKL_NUM_THREADS = "1")
  pinned <- c(aghq = "0.4.1", reticulate = "1.46.0", numDeriv = "2016.8.1.1",
              mvQuad = "1.0.10", jsonlite = "2.0.0")
  versions <- vapply(names(pinned), function(name) {
    # Function: Check namespace version and origin
    # Inputs: Package name Outputs: Exact version or failure
    if (!requireNamespace(name, quietly = TRUE)) stop(paste("missing R dependency:", name))
    origin <- normalizePath(getNamespaceInfo(asNamespace(name), "path"), winslash = "/", mustWork = TRUE)
    if (!startsWith(tolower(origin), paste0(tolower(r_library), "/")))
      stop(paste("R dependency loaded outside explicit library:", name))
    as.character(utils::packageVersion(name))
  }, character(1))
```

```
  if (!identical(versions, pinned)) stop("R dependency versions differ from the audited harness")
  data.table::setDTthreads(1L)
  reticulate::use_python(python, required = TRUE)
  python_sys <- reticulate::import("sys", convert = FALSE)
  python_sys$path$insert(0L, root)
  python_sys$path$insert(0L, file.path(root, "bridge"))
  adapter <- reticulate::import_from_path("external_scenario", path = file.path(root, "r_external"), convert = TRUE)
  provenance <- adapter$verify_runtime(python)
  list(adapter = adapter, environment = list(root = root, R = R.version.string, LC_CTYPE = Sys.getlocale("LC_CTYPE"),
       R_versions = as.list(versions), R_session_info = capture.output(utils::sessionInfo()),
       r_library = r_library, Python = provenance))
}

# Function: Enforce preregistered external-comparison stages
# Inputs: Model and integration settings
# Outputs: Validated settings; no best-order selection
external_settings <- function(model_name, settings) {
  settings <- validate_settings(settings)
  orders <- if (model_name == "OXC") c(7L, 15L, 31L, 61L) else c(4L, 5L, 6L)
  construction <- if (model_name == "OXC") "product" else "sparse"
  rule <- if (model_name == "OXC") "GHe" else "nHe"
  if (!identical(settings$orders, orders) || settings$construction != construction ||
      settings$rule != rule || settings$acceptance_mode != "external_comparison")
    stop("settings must use the prespecified model stages/rule and external_comparison")
  settings
}

# Function: Execute one case using only command arguments and explicit runtime
# Inputs: Four command arguments
# Outputs: Final result.json, including failures after runtime initialization
main <- function(args = commandArgs(trailingOnly = TRUE)) {
  if (length(args) != 4L) stop("usage: run_case.R RAW_JSON OXC|TAC SETTINGS_JSON OUTPUT_DIR")
  if (!args[[2]] %in% c("OXC", "TAC")) stop("model must be OXC or TAC")
  # jsonlite must be located before parsing runtime config; bootstrap from explicit R library.
  # Locate jsonlite using the explicit R library or R_LIBS_USER before reading settings.
  library_env <- Sys.getenv("PMX_R_LIBRARY", unset = Sys.getenv("R_LIBS_USER", unset = ""))
  if (nzchar(library_env)) .libPaths(c(library_env, .libPaths()))
  if (!requireNamespace("jsonlite", quietly = TRUE)) stop("set PMX_R_LIBRARY or R_LIBS_USER to bootstrap jsonlite")
  config <- jsonlite::read_json(args[[3]], simplifyVector = FALSE)
  runtime_config <- if (is.null(config$runtime)) list() else config$runtime
  root <- runtime_value("PMX_RERUN_ROOT", runtime_config$root)
  if (!identical(root, normalizePath(getwd(), winslash = "/", mustWork = TRUE)))
    stop("process cwd must equal the explicitly configured rerun root")
  python <- runtime_value("RETICULATE_PYTHON", runtime_config$python,
                          file.path(root, ".venv/Scripts/python.exe"))
  r_library <- runtime_value("PMX_R_LIBRARY", runtime_config$r_library,
                             if (nzchar(library_env)) library_env else NULL)
  raw_path <- normalizePath(args[[1]], winslash = "/", mustWork = TRUE)
  output_parent <- normalizePath(dirname(args[[4]]), winslash = "/", mustWork = TRUE)
  if (!startsWith(tolower(paste0(output_parent, "/")), paste0(tolower(root), "/")) ||
      basename(args[[4]]) %in% c(".", ".."))
    stop("output parent must already exist under the explicit rerun root")
  dir.create(args[[4]], showWarnings = FALSE)
  output <- normalizePath(args[[4]], winslash = "/", mustWork = TRUE)
  if (!startsWith(tolower(paste0(output, "/")), paste0(tolower(root), "/")))
    stop("output directory must remain under the explicit rerun root")
  if (file.exists(file.path(output, "result.json"))) stop("result.json already exists; use a fresh output directory")
  Sys.setenv(VALIDATION_RUN_DIR = output)
  started <- proc.time()[["elapsed"]]
  environment <- NULL
  metadata <- NULL
  settings <- NULL
  result <- tryCatch({
    bound <- initialize_external_runtime(root, python, r_library)
    environment <- bound$environment
    jsonlite::write_json(environment, file.path(output, "actual_environment.json"),
                         auto_unbox = TRUE, pretty = TRUE, null = "null", digits = NA)
    withCallingHandlers(source(file.path(root, "bridge/scenario.R"), encoding = "UTF-8"),
      warning = function(warning) {
        # Function: Reject incomplete source loading
        # Inputs: Source warning Outputs: Explicit failure
        stop(paste("archived bridge source warning:", conditionMessage(warning)))
      })
    settings <- config$integration
    if (is.null(settings)) settings <- config[setdiff(names(config), "runtime")]
    settings <- external_settings(args[[2]], settings)
    scenario <- bound$adapter$make_scenario(raw_path, args[[2]])
    metadata <- scenario$metadata()
    run_scenario(scenario, settings)
  }, error = function(error) {
    # Function: Preserve an explicit failure without substitute posterior
    # Inputs: Original exception Outputs: Failure record
    list(status = "failed", backend = "R aghq", posterior = NULL, state_results = list(),
         failure_reason = conditionMessage(error), absolute_error_certified = FALSE)
  })
  result$input <- metadata
  result$environment <- environment
  result$settings <- settings
  result$source_path <- raw_path
  result$model_name <- args[[2]]
  result$elapsed_seconds <- proc.time()[["elapsed"]] - started
```

```
  jsonlite::write_json(result, file.path(output, "result.json"), auto_unbox = TRUE,
                       pretty = TRUE, null = "null", digits = NA)
  if (!identical(result$status, "success")) stop(paste("external case failed:", result$failure_reason))
  invisible(result)
}

if (sys.nframe() == 0L) main()
```

bridge/scenario.R

scenario.R

```
source("bridge/integration.R", encoding = "UTF-8")

# Function: Schedule external integrals and reconnect normalization.
# Inputs: Python scenario and explicit settings
# Outputs: complete result including failed/skipped candidates
run_scenario <- function(scenario, settings) {
  settings <- validate_settings(settings)
  meta <- scenario$metadata()
  rows <- list()
  for (id in unlist(meta$state_ids, use.names = FALSE)) {
    if (meta$priors[[id]] == 0) {
      rows[[id]] <- list(status = "skipped_zero_prior", log_marginal_likelihood = NULL, failure_reason = NULL)
    } else {
      rows[[id]] <- integrate_target(scenario$target(id), settings)
    }
    if (nzchar(Sys.getenv("VALIDATION_RUN_DIR"))) {
      jsonlite::write_json(list(input=meta,state_results=rows,completed_states=length(rows)),
        file.path(Sys.getenv("VALIDATION_RUN_DIR"),"candidate_checkpoint.json"),
        auto_unbox=TRUE,pretty=TRUE,null="null",digits=NA)
    }
  }
  scenario$finalize(jsonlite::toJSON(rows, auto_unbox = TRUE, null = "null", digits = NA),
                    jsonlite::toJSON(settings, auto_unbox = TRUE, digits = NA))
}
```

bridge/integration.R

integration.R

```
# Function: External integration for verification only.
# Inputs: Python normalized log-joint callback and explicit settings
# Outputs: logZ, actual checks, failure reason
# All optimization, differentiation, nodes, adaptation and quadrature are library calls.
# Libraries perform optimization, differentiation and quadrature; this module connects and checks them.
source("bridge/runtime.R", encoding = "UTF-8")
source("bridge/coordinates.R", encoding = "UTF-8")

# Function: Validate acceptance configuration.
# Inputs: named settings Outputs: validated list or error
validate_settings <- function(settings) {
  expected <- c("orders", "log_integral_atol", "posterior_atol", "max_nodes",
                "max_evaluations", "curvature_atol", "stationarity_atol")
  optional <- c("construction", "rule", "derivative_policy", "optimizer", "acceptance_mode", "coordinate_system")
  if (anyDuplicated(names(settings)) || !all(expected %in% names(settings)) ||
      length(setdiff(names(settings), c(expected,optional)))) stop("unexpected or missing integration setting")
  if (is.null(settings$construction)) settings$construction <- "product"
  if (is.null(settings$rule)) settings$rule <- "GHe"
  if (is.null(settings$derivative_policy)) settings$derivative_policy <- "library_default"
  if (is.null(settings$optimizer)) settings$optimizer <- "aghq_BFGS"
  if (is.null(settings$acceptance_mode)) settings$acceptance_mode <- "stability_required"
  if (is.null(settings$coordinate_system)) settings$coordinate_system <- "identity"
  if (!is.character(settings$coordinate_system) || length(settings$coordinate_system)!=1 ||
      is.na(settings$coordinate_system) || !settings$coordinate_system %in%
      c("identity","prior_standardized")) stop("invalid coordinate system")
  if (length(settings$acceptance_mode)!=1 || !settings$acceptance_mode %in%
      c("stability_required","external_comparison")) stop("invalid acceptance mode")
  if (length(settings$optimizer)!=1 || !settings$optimizer %in%
      c("aghq_BFGS","R_BFGS_scaled")) stop("invalid optimizer")
  if (length(settings$derivative_policy)!=1 || !settings$derivative_policy %in%
      c("library_default","floored_richardson")) stop("invalid derivative policy")
  if (length(settings$construction)!=1 || !settings$construction %in% c("product","sparse")) stop("invalid
  construction")
  if (length(settings$rule)!=1 || !settings$rule %in% c("GHe","nHe")) stop("invalid integration measure or rule")
  orders <- unlist(settings$orders, use.names = FALSE)
  if (!is.numeric(orders) || length(orders) < 3L || any(!is.finite(orders)) ||
      any(orders != floor(orders)) || any(orders < 3) || any(diff(orders) <= 0)) {
    stop("orders must contain at least three strictly increasing integers >=3")
  }
  for (key in setdiff(expected, "orders")) {
    value <- settings[[key]]
    if (!is.numeric(value) || length(value) != 1L || !is.finite(value) || value <= 0) stop(paste("invalid", key))
  }
  for (key in c("max_nodes", "max_evaluations")) {
    if (settings[[key]] != floor(settings[[key]])) stop(paste("budget must be integer:", key))
  }
  settings$orders <- as.integer(orders)
```

```
  settings
}

# Function: Make library derivative steps explicit.
# Inputs: validated policy Outputs: numDeriv method.args
# always-floor uses the library's eps+abs(d*x) steps without modifying the Hessian.
derivative_arguments <- function(policy) {
  if (policy=="library_default") return(list(gradient=list(),hessian=list(),check=list(eps=2e-4,d=.2)))
  if (policy!="floored_richardson") stop("unknown derivative policy")
  list(gradient=list(eps=1e-4,d=1e-4,zero.tol=Inf,r=4L,v=2),
       hessian=list(eps=1e-3,d=.1,zero.tol=Inf,r=4L,v=2),
       check=list(eps=2e-3,d=.2,zero.tol=Inf,r=4L,v=2))
}

# Function: Diagnose the library's signed sum only.
# Inputs: returned node table and library logZ
# Outputs: cancellation diagnostics or explicit failure
check_quadrature_table <- function(result) {
  if (!all(c("nodesandweights","grid","lognormconst") %in% names(result))) stop("aghq return schema changed")
  tab <- result$nodesandweights
  if (!is.data.frame(tab) || !all(c("weights","logpost") %in% names(tab)) ||
      !is.numeric(result$lognormconst) || length(result$lognormconst)!=1 || !is.finite(result$lognormconst) ||
      any(!is.finite(as.matrix(tab)))) stop("invalid library quadrature result")
  w <- tab$weights
  if (!any(w>0)) stop("quadrature has no positive contribution")
  active <- w!=0
  a <- log(abs(w[active])) + tab$logpost[active]
  shift <- max(a)
  positive <- sum(exp(a[w[active]>0]-shift))
  negative <- sum(exp(a[w[active]<0]-shift))
  if (!is.finite(positive) || !is.finite(negative) || positive<=negative) stop("nonpositive signed quadrature total")
  log_condition <- shift+log(positive+negative)-result$lognormconst
  # Condition sensitivity is eps*kappa. The n*eps*kappa serial-sum bound is
  # retained separately: it is not an observed error of R/library accumulation.
  # eps*kappa measures rounding sensitivity; the separately stored serial-sum bound is not an error estimate.
  roundoff_indicator <- .Machine$double.eps * exp(log_condition)
  if (!is.finite(roundoff_indicator) || roundoff_indicator>1e-8) stop("severe signed-weight cancellation")
  list(nodes=nrow(tab),negative_weights=sum(w<0),negative_positive_ratio=negative/positive,
       log_condition_number=log_condition,estimated_digits_lost=max(0,log_condition/log(10)),
       roundoff_indicator=roundoff_indicator,roundoff_limit=1e-8,
       conservative_serial_sum_bound=nrow(tab)*roundoff_indicator,
       interpretation="summation sensitivity diagnostic, not a quadrature error bound")
}

# Function: Check unmodified curvature.
# Inputs: negative log-joint Hessian
# Outputs: original positive-definite matrix, or error
check_curvature <- function(hessian) {
  hessian <- as.matrix(hessian)
  if (nrow(hessian) != ncol(hessian) || any(!is.finite(hessian)) ||
      max(abs(hessian - t(hessian))) > 1e-10 * max(1, max(abs(hessian)))) stop("invalid raw curvature")
  chol(hessian)  # No jitter, nearest-SPD repair or eigenvalue replacement
  hessian
}

# Function: Compare curvature in all directions.
# Inputs: two raw positive-definite Hessians
# Outputs: empirical relative perturbation and log-volume bound
compare_curvature <- function(hessian, alternate) {
  hessian <- check_curvature(hessian)
  alternate <- check_curvature(alternate)
  inverse_factor <- backsolve(chol(hessian), diag(nrow(hessian)))
  perturbation <- t(inverse_factor) %*% (alternate - hessian) %*% inverse_factor
  values <- eigen(perturbation, symmetric = TRUE, only.values = TRUE)$values
  if (any(!is.finite(values)) || max(abs(values)) >= 1) stop("curvature direction check failed")
  list(relative_spectral_change = max(abs(values)),
       log_volume_impact = 0.5 * length(values) * max(abs(log1p(values))))
}

# Function: Delegate the complete integral to aghq.
# Inputs: target metadata/log_joint; validated settings
# Outputs: common integration fields and per-order evidence
integrate_target <- function(target, settings) {
  settings <- validate_settings(settings)
  details <- list(backend = "R aghq", variant = paste0("adaptive_",settings$construction), settings = settings,
                  order_results = list(), evaluations = 0L, nodes = 0,
                  stability_passed = FALSE, absolute_error_certified = FALSE, warnings = list())
  phase <- "initial"
  last_point <- NULL
  # Function: Preserve failure phase and both coordinates.
  # Inputs: condition and last attempted point
  # Outputs: first failure trace, without replacing errors
  record_failure <- function(e) {
    if (is.null(details$failure_trace)) {
      details$failure_trace <<- c(list(phase=phase, coordinate_system=settings$coordinate_system,
        evaluation_index=details$evaluations, reason=conditionMessage(e)), last_point)
    }
  }
  # Function: Label derivative evaluation phases.
  # Inputs: phase and library expression
  # Outputs: original library result or recorded error
```

```
in_phase <- function(name, expression) {
  previous_phase <- phase
  phase <<- name
  on.exit(phase <<- previous_phase)
  withCallingHandlers(force(expression), error=record_failure)
}
output <- tryCatch(withCallingHandlers({
  meta <- target$metadata()
  adapter <- make_coordinate_adapter(meta, settings$coordinate_system)
  start <- adapter$start
  dimension <- adapter$dimension
  details$coordinates <- adapter[c("coordinate_system","dimension","mean","covariance",
    "offset","L","log_jacobian","coordinate_label","start")]
  details$coordinates$raw_start <- adapter$to_raw(start)
  # Function: Budget and deterministic evaluation.
  # Inputs: x or z Outputs: log density in that measure
  fn <- function(value) {
    last_point <<- list(coordinates=as.numeric(value), transformed_coordinates=as.numeric(value),
      raw_coordinates=NULL, coordinate_label=adapter$coordinate_label)
    tryCatch({
      x <- adapter$to_raw(value)
      last_point$raw_coordinates <<- x
      if (details$evaluations >= settings$max_evaluations) stop("model evaluation budget exhausted")
      details$evaluations <<- details$evaluations + 1L
      coordinate_log_joint(evaluate_target(target, x, dimension), adapter)
    }, error=function(e) {
      # Function: Preserve the failed point without replacing density.
      # Inputs: exact evaluated coordinate and error
      # Outputs: failure evidence and original exception
      record_failure(e)
      details$failed_evaluation <<- details$failure_trace
      stop(e)
    })
  }
  first <- fn(start)
  if (!identical(first, fn(start))) stop("target is not deterministic at the initial point")
  if (dimension == 0L) {
    details$scope <- "direct"
    details$stability_passed <- TRUE
    list(status = "success", log_marginal_likelihood = first, failure_reason = NULL)
  } else {
    if (settings$construction=="product" && settings$rule=="GHe" &&
        sum(as.double(settings$orders)^dimension) > settings$max_nodes) stop("quadrature node budget exceeded before
grid allocation")
    # Function/Inputs/Outputs: use independent R numerical derivatives of fn.
    # Differentiate the complete fn in R without using Python modes or Hessians.
    derivative_args <- derivative_arguments(settings$derivative_policy)
    details$derivatives <- list(policy=settings$derivative_policy,
      arguments=derivative_args,step_formula="abs(d*x)+eps*(abs(x)<zero.tol)",
      coordinate_system=adapter$coordinate_system, coordinate_label=adapter$coordinate_label)
    # Function: Save both modes before post-optimization derivatives.
    # Inputs: library mode and convergence
    # Outputs: optimizer evidence, no Python optimizer fields
    save_mode <- function(value, convergence=NULL) {
      details$optimizer <<- list(method=settings$optimizer, convergence=convergence,
        mode=as.numeric(value), transformed_mode=as.numeric(value), raw_mode=adapter$to_raw(value),
        coordinate_system=adapter$coordinate_system, coordinate_label=adapter$coordinate_label,
        controls=optimizer_controls)
    }
    ff <- list(fn = fn,
      gr = function(x) in_phase(if (phase=="optimizer_objective") "optimizer_gradient_stencil" else phase,
        numDeriv::grad(fn, x, method = "Richardson",method.args=derivative_args$gradient)),
      he = function(x) {
        if (is.null(details$optimizer)) save_mode(x)
        in_phase("post_optimization_hessian",
          numDeriv::hessian(fn, x, method = "Richardson",method.args=derivative_args$hessian))
      })
    control <- aghq::default_control(method = "BFGS", negate = FALSE,
                                     ndConstruction = settings$construction, onlynormconst = TRUE)
    # aghq 0.4.1's BFGS branch ignores optcontrol; do not advertise it here.
    # This BFGS branch ignores optcontrol; record the effective defaults and outer evaluation budget.
    phase <- "optimizer_objective"
    if (settings$optimizer=="aghq_BFGS") {
      optimizer_controls <- "aghq 0.4.1 BFGS defaults"
      optimized <- aghq::optimize_theta(ff, start, control = control)
    } else {
      # Function: Use explicit scales in R's BFGS.
      # Inputs: same complete log joint and R derivatives
      # Outputs: aghq optresults in integration coordinates
      # aghq's BFGS wrapper discards optcontrol; call the same mature optimizer directly.
      # This aghq version ignores optcontrol; call the same optimizer directly without altering the integration
region.
      optimizer_controls <- list(parscale=rep(.1,dimension),maxit=1000L,reltol=1e-10)
      fit <- stats::optim(start,function(x) -ff$fn(x),function(x) -ff$gr(x),
                          method="BFGS",control=optimizer_controls)
      # Preserve optimizer evidence before any post-optimization derivative can fail.
      # Store the mode and exit status before subsequent differentiation can fail.
      save_mode(fit$par, fit$convergence)
      if (!identical(as.integer(fit$convergence),0L)) stop("R optimization did not converge")
      optimized <- list(ff=ff,mode=fit$par,hessian=-ff$he(fit$par),convergence=fit$convergence)
    }
```

```
      save_mode(optimized$mode, optimized$convergence)
      if (!identical(as.integer(optimized$convergence), 0L)) stop("R optimization did not converge")
      phase <- "post_optimization_hessian"
      hessian <- check_curvature(optimized$hessian)
      phase <- "check_hessian"
      check_hessian <- check_curvature(-numDeriv::hessian(fn, optimized$mode, method = "Richardson",
                                                          method.args = derivative_args$check))
      comparison <- compare_curvature(hessian, check_hessian)
      curvature_change <- comparison$log_volume_impact
      phase <- "stationarity_gradient_stencil"
      gradient <- as.numeric(ff$gr(optimized$mode))
      stationarity <- as.numeric(crossprod(gradient, solve(hessian, gradient))) / 2
      details$curvature <- list(raw_hessian = unname(hessian), alternate_hessian = unname(check_hessian),
                                coordinate_system=adapter$coordinate_system,
                                coordinate_label=adapter$coordinate_label, gradient=gradient,
                                gradient_coordinate_label=adapter$coordinate_label,
                                hessian_coordinate_label=adapter$coordinate_label,
                                raw_hessian_meaning="unmodified negative-log-density Hessian in integration
  coordinates",
                                logdet_impact_estimate = curvature_change,
                                relative_spectral_change = comparison$relative_spectral_change,
                                stationarity_impact_estimate = stationarity,
                                interpretation = "empirical checks, not strict error bounds")
      if (!is.finite(curvature_change) || curvature_change > settings$curvature_atol) stop("curvature check failed")
      if (!is.finite(stationarity) || stationarity > settings$stationarity_atol) stop("stationarity check failed")
      # Function: Reuse only R's adaptation across orders.
      # Inputs: R optresults and k Outputs: library log integrals
      previous <- NULL
      phase <- "quadrature"
      for (order in settings$orders) {
        # Function: Delegate grid creation and count actual nodes.
        # Inputs: dimension, construction, level Outputs: library grid
        grid <- mvQuad::createNIGrid(dim=dimension,type=settings$rule,level=order,
                                     ndConstruction=settings$construction)
        nodes <- nrow(mvQuad::getNodes(grid))
        if (details$nodes+nodes>settings$max_nodes) stop("quadrature node budget exceeded before stage evaluation")
        before <- details$evaluations
        # normalize_logpost is aghq()'s exported integration implementation.
        # Call the exported aghq integral for diagnostics, omitting unreliable sparse-grid marginal summaries.
        normalized <- aghq::normalize_logpost(optresults=optimized,k=order,basegrid=grid,
                                              ndConstruction=settings$construction)
        diagnostics <- check_quadrature_table(normalized)
        if (diagnostics$nodes!=nodes) stop("library grid accounting differs")
        logz <- normalized$lognormconst
        if (!is.numeric(logz) || length(logz) != 1L || !is.finite(logz)) stop("aghq returned invalid log integral")
        details$nodes <- details$nodes + nodes
        details$order_results[[length(details$order_results) + 1L]] <- list(order = order,
          log_marginal_likelihood = as.numeric(logz), change = if (is.null(previous)) NULL else abs(logz - previous),
          nodes = nodes, evaluations = details$evaluations - before, summation=diagnostics)
        previous <- logz
      }
      details$stability_passed <- all(vapply(tail(details$order_results, 2),
        function(row) row$change <= settings$log_integral_atol, logical(1)))
      # Function: Separate returned integrals from order diagnostics.
      # Inputs: complete finite stages and explicit mode
      # Outputs: last configured integral; strict mode may reject
      # External comparisons do not require threshold passage or agreement with Laplace; errors are caught upstream.
      if (!details$stability_passed && settings$acceptance_mode=="stability_required") {
        stop("last two log-integral changes exceed tolerance")
      }
      if (!details$stability_passed) {
        details$warnings[[length(details$warnings)+1L]] <- list(phase="quadrature",
          reason="last two log-integral changes exceed tolerance")
      }
      list(status = "success", log_marginal_likelihood = as.numeric(previous), failure_reason = NULL)
    }
  }, warning = function(w) {
    # Function: Record warnings and preserve the original failure policy.
    # Inputs: backend warning Outputs: evidence and failure in both modes
    details$warnings[[length(details$warnings)+1L]] <<- c(list(phase=phase,
      reason=conditionMessage(w)), last_point)
    stop(paste("backend warning:", conditionMessage(w)), call. = FALSE)
  }), error = function(e) {
    record_failure(e)
    list(status = "failed", log_marginal_likelihood = NULL, failure_reason = conditionMessage(e))
  })
  output$method_details <- details
  output
}
```

bridge/coordinates.R

coordinates.R

```
# Function: Validate a coordinate vector.
# Inputs: numeric vector and dimension
# Outputs: unchanged finite numeric vector or error
coordinate_vector <- function(value, dimension) {
  if (!is.numeric(value) || !is.null(dim(value)) || length(value) != dimension ||
```

```
      any(!is.finite(value))) stop("coordinates must be a finite numeric vector of declared dimension")
  as.numeric(value)
}

# Function: Read covariance without repair.
# Inputs: metadata matrix or row lists and dimension
# Outputs: finite exactly symmetric matrix
coordinate_covariance <- function(value, dimension) {
  if (is.list(value) && is.null(dim(value))) {
    if (length(value) != dimension) stop("invalid covariance dimensions")
    rows <- lapply(value, function(row) coordinate_vector(unlist(row, use.names = FALSE), dimension))
    value <- if (dimension == 0L) matrix(numeric(), 0L, 0L) else do.call(rbind, rows)
  }
  if (!is.matrix(value) || !is.numeric(value) ||
      !identical(dim(value), c(dimension, dimension)) || any(!is.finite(value))) {
    stop("invalid covariance matrix")
  }
  if (any(value != t(value))) stop("covariance must be symmetric; no symmetrization is permitted")
  unname(value)
}

# Function: Create an explicit invertible coordinate adapter.
# Inputs: target metadata and identity/prior_standardized
# Outputs: raw/transformed maps, start and Jacobian
make_coordinate_adapter <- function(meta, coordinate_system = "identity") {
  if (!is.character(coordinate_system) || length(coordinate_system) != 1L ||
      is.na(coordinate_system) || !coordinate_system %in% c("identity", "prior_standardized")) {
    stop("invalid coordinate system")
  }
  dimension <- meta$dimension
  if (!is.numeric(dimension) || length(dimension) != 1L || !is.finite(dimension) ||
      dimension < 0 || dimension != floor(dimension) || dimension > .Machine$integer.max) {
    stop("invalid target metadata dimension")
  }
  dimension <- as.integer(dimension)
  mean <- unlist(meta$mean, use.names = FALSE)
  if (dimension == 0L && is.null(mean)) mean <- numeric()
  mean <- coordinate_vector(mean, dimension)
  if (coordinate_system == "identity") {
    # Function: Preserve the original numeric path.
    # Inputs: x Outputs: x, without centering or Jacobian arithmetic
    L <- diag(dimension)
    covariance <- NULL
    log_jacobian <- 0
    to_raw <- from_raw <- function(value) coordinate_vector(value, dimension)
    start <- mean
    offset <- numeric(dimension)
    label <- "x"
  } else {
    covariance <- coordinate_covariance(meta$covariance, dimension)
    L <- if (dimension == 0L) matrix(numeric(), 0L, 0L) else t(chol(covariance))
    log_jacobian <- sum(log(diag(L)))
    if (!is.finite(log_jacobian)) stop("non-finite coordinate log Jacobian")
    # Function: Apply x=mean+L*z.
    # Inputs: z Outputs: original target x
    to_raw <- function(value) {
      z <- coordinate_vector(value, dimension)
      coordinate_vector(as.numeric(mean + L %*% z), dimension)
    }
    # Function: Invert the map without clipping.
    # Inputs: x Outputs: z
    from_raw <- function(value) {
      x <- coordinate_vector(value, dimension)
      if (dimension == 0L) return(numeric())
      coordinate_vector(as.numeric(forwardsolve(L, x - mean)), dimension)
    }
    start <- numeric(dimension)
    offset <- mean
    label <- "z"
  }
  list(coordinate_system = coordinate_system, dimension = dimension, mean = mean,
       covariance = covariance, offset = offset, L = L, log_jacobian = log_jacobian,
       coordinate_label = label, start = start, to_raw = to_raw, from_raw = from_raw)
}

# Function: Add the coordinate Jacobian exactly once.
# Inputs: finite raw log joint and adapter
# Outputs: integration-coordinate log density
coordinate_log_joint <- function(raw_value, adapter) {
  if (!is.numeric(raw_value) || length(raw_value) != 1L || !is.finite(raw_value)) {
    stop("target returned a non-finite or non-scalar log joint")
  }
  if (adapter$coordinate_system == "identity") return(raw_value)
  value <- raw_value + adapter$log_jacobian
  if (!is.finite(value)) stop("transformed log joint is not finite")
  value
}
```

bridge/runtime.R

runtime.R

```
# Function: Bind the workspace Python environment.
# Inputs: fixed validation cwd and environment
# Outputs: imported modules and version information
initialize_runtime <- function() {
  root <- normalizePath(getwd(), winslash = "/", mustWork = TRUE)
  expected <- normalizePath(file.path(root, ".venv/Scripts/python.exe"), winslash = "/", mustWork = TRUE)
  if (!identical(normalizePath(Sys.getenv("RETICULATE_PYTHON"), winslash = "/"), expected)) {
    stop("RETICULATE_PYTHON must name the verification environment")
  }
  reticulate::use_python(expected, required = TRUE)
  pkg <- reticulate::import("pmx_discrete_posterior", convert = TRUE)
  origin <- normalizePath(pkg$`__file__`, winslash = "/", mustWork = TRUE)
  if (!startsWith(tolower(origin), tolower(paste0(root, "/.venv/Lib/site-packages/")))) {
    stop("package must be loaded from the installed verification wheel")
  }
  versions <- vapply(c("aghq", "reticulate", "numDeriv", "mvQuad", "jsonlite"),
    function(name) {
      # Verify loadability first; Windows sandbox file.access(mode=5) can reject
      # an unloaded package directory even though its namespace is readable.
      # Test namespace loading directly; directory permission checks are insufficient on Windows.
      if (!requireNamespace(name, quietly = TRUE)) stop(paste("missing R dependency", name))
      as.character(utils::packageVersion(name))
    }, character(1))
  pinned <- c(aghq="0.4.1", reticulate="1.46.0", numDeriv="2016.8.1.1", mvQuad="1.0.10", jsonlite="2.0.0")
  if (!identical(versions, pinned)) stop("R dependency versions changed; requalify the environment")
  data.table::setDTthreads(1L)
  provenance <- reticulate::import_from_path("provenance", path=file.path(root, "bridge"),
  convert=TRUE)$verify_runtime()
  if (nzchar(Sys.getenv("VALIDATION_RUN_DIR"))) {
    jsonlite::write_json(list(R=R.version.string, R_versions=as.list(versions), Python=provenance),
      file.path(Sys.getenv("VALIDATION_RUN_DIR"), "actual_environment.json"), auto_unbox=TRUE, pretty=TRUE)
  }
  list(root = root, python = expected, package_origin = origin, R = R.version.string,
       versions = as.list(versions), fixtures = reticulate::import_from_path(
         "analytic_targets", path = file.path(root, "fixtures"), convert = TRUE))
}

# Function: Pass finite vectors and preserve failures.
# Inputs: Python target, R eta and optional validated dimension
# Outputs: finite scalar or error, never replacement weights
evaluate_target <- function(target, eta, dimension=NULL) {
  if (is.null(dimension)) dimension <- as.integer(target$metadata()$dimension)
  if (length(dimension)!=1 || !is.numeric(dimension) || !is.finite(dimension) ||
      dimension<0 || dimension!=floor(dimension)) stop("invalid target dimension")
  if (!is.numeric(eta) || length(eta) != dimension || any(!is.finite(eta))) {
    stop("eta must be a finite numeric vector of declared dimension")
  }
  value <- target$log_joint(as.list(as.numeric(eta)))
  if (!is.numeric(value) || length(value) != 1L || !is.finite(value)) {
    stop("target returned a non-finite or non-scalar log joint")
  }
  as.numeric(value)
}
```

r_external/external_scenario.py

external_scenario.py

```
"""Function: Bind raw reruns to the archived R bridge
Inputs: Raw JSON, shared mapper and installed 0.2.0
Outputs: Exact targets and source metadata, no quadrature
"""

import hashlib
import importlib
import importlib.metadata
import inspect
import json
import math
from functools import lru_cache
from pathlib import Path
import sys
import warnings

import numpy as np
from scipy.stats import multivariate_normal

import pmx_discrete_posterior
from pmx_discrete_posterior.interfaces.inputs import validate_inputs
from pmx_discrete_posterior.models.prediction import prepare_prediction_records
from pmx_discrete_posterior.models.prediction.linear import LinearPredictor
from pmx_discrete_posterior.models.random_effects import resolve_random_effects
from pmx_discrete_posterior.posterior import computation

from model_bridge import ModelScenario, ModelTarget
```

```
PACKAGE_TREE_SHA256 = "5d65cd8f75f1e457a6e326f499702d41d3905690989d2cfd4a38b58f192b33d5"
BINDING_SHA256 = "dd42654fbae69118d52e8db04f5789110865c9ca86a68d80207b51d38c8713bd"
REFERENCE_SHA256 = "d4c07d73fc4521da244d6ed715fcda7e6cc03f3172488d931f39a09857c3e9cc"
PINNED_VERSIONS = {"pmx-discrete-posterior": "0.2.0", "numpy": "2.5.2", "scipy": "1.16.2"}


def file_sha256(path):
    """Function: Hash source bytes
    Inputs: File path Outputs: SHA256 hex digest
    """
    return hashlib.sha256(Path(path).read_bytes()).hexdigest()


@lru_cache(maxsize=1)
def verify_sources():
    """Function: Check the pinned implementation once
    Inputs: Installed package and archived reference module
    Outputs: Versions and source hashes, or explicit failure
    """
    reference = importlib.import_module("shared.reference_models")
    versions = {name: importlib.metadata.version(name) for name in PINNED_VERSIONS}
    if versions != PINNED_VERSIONS:
        raise RuntimeError(f"pinned Python dependency versions changed: {versions}")
    package = Path(pmx_discrete_posterior.__file__).resolve().parent
    files = sorted(package.rglob("*.py"), key=lambda path: path.relative_to(package).as_posix())
    manifest = "".join(f"{path.relative_to(package).as_posix()}\0{file_sha256(path)}\n" for path in files)
    tree_hash = hashlib.sha256(manifest.encode("utf-8")).hexdigest()
    if tree_hash != PACKAGE_TREE_SHA256 or file_sha256(computation.__file__) != BINDING_SHA256:
        raise RuntimeError("installed package differs from the audited 0.2.0 Python source tree")
    if file_sha256(reference.__file__) != REFERENCE_SHA256:
        raise RuntimeError("shared/reference_models.py differs from the archived audited factory")
    return {"versions": versions, "package_origin": str(package), "package_source_sha256": tree_hash,
            "binding_sha256": BINDING_SHA256, "reference_source_sha256": REFERENCE_SHA256,
            "reference_origin": str(Path(reference.__file__).resolve())}


def verify_runtime(expected_python):
    """Function: Verify the explicitly selected installed environment
    Inputs: Absolute Python executable
    Outputs: Actual executable, versions and provenance
    """
    expected = Path(expected_python).resolve(strict=True)
    if Path(sys.executable).resolve() != expected:
        raise RuntimeError("reticulate is not using the explicitly selected Python executable")
    package = Path(pmx_discrete_posterior.__file__).resolve()
    site_packages = expected.parent.parent / "Lib" / "site-packages"
    if not package.is_relative_to(site_packages.resolve()):
        raise RuntimeError("pmx_discrete_posterior must come from the selected installed venv")
    return {**verify_sources(), "python": sys.version, "executable": str(expected)}


def effect_names(keys):
    """Function: Label coordinates without losing occasions
    Inputs: Ordered effect keys Outputs: Ordered names
    """
    return [".".join(str(part) for part in key if part is not None) for key in keys]


class NamedModelTarget(ModelTarget):
    """Function: Add names to the unchanged archived target
    Inputs: Bound target inputs Outputs: Named target
    """

    def metadata(self):
        """Function: Export full-coordinate metadata
        Inputs: None Outputs: Dimension, names and covariance
        """
        keys = self.distribution.effect_keys
        return {**super().metadata(), "representation": "full", "effect_names": effect_names(keys),
                "effect_keys": [list(key) for key in keys],
                "covariance": self.distribution.covariance.tolist()}


def prove_tac_sum_structure(target, model, data):
    """Function: Restrict projection to the audited CL/V1 sums
    Inputs: Bound target, validated model and data
    Outputs: Ordered occasion IDs or failure; not a numerical proof
    """
    verify_sources()
    reference = importlib.import_module("shared.reference_models")
    if model["parameter_model"]["relations"] is not reference._tac_relations:
        raise ValueError("exact TAC projection requires the audited shared TAC relations")
    predictor = model["predict"]
    if type(predictor) is not LinearPredictor or predictor._system is not reference._tac_system:
        raise ValueError("exact TAC projection requires the audited TAC LinearPredictor")
    likelihood_code = next(code for code in computation._bind_likelihood.__code__.co_consts
                           if inspect.iscode(code) and code.co_name == "likelihood")
    if not inspect.isfunction(target.likelihood) or target.likelihood.__code__ is not likelihood_code:
        raise ValueError("exact TAC projection requires the pinned package likelihood binding")
    closure = inspect.getclosurevars(target.likelihood).nonlocals
    if (closure.get("model") is not model or closure.get("individual_data") is not data or
```

```
            closure.get("random_effect_distribution") is not target.distribution):
        raise ValueError("projection proof and bound likelihood do not use the same inputs")
    occasions = [occasion["occasion_id"] for occasion in data["occasions"]]
    if not occasions or len(set(occasions)) != len(occasions):
        raise ValueError("exact TAC projection requires distinct nonempty occasions")
    expected = [("iiv", None, name) for name in ("CL", "V1", "V2", "ka")]
    expected += [("iov", occasion, name) for occasion in occasions for name in ("CL", "V1")]
    if target.distribution.effect_keys != tuple(expected):
        raise ValueError("unsupported TAC active effects; expected all CL/V1 IIV and IOV plus V2/ka IIV")
    return occasions


class TACGaussianProjection:
    """Function: Push the Gaussian through exact dynamic TAC sums
    Inputs: Audited full bound target and validated inputs
    Outputs: Equivalent normalized (2*n+2)D target
    """

    def __init__(self, target, model, data):
        """Function: Build A, a right inverse and A Omega A^T
        Inputs: Target, model and data
        Outputs: Proven exact projection; no jitter or fallback
        """
        self.occasions = prove_tac_sum_structure(target, model, data)
        self.original = target
        count = len(self.occasions)
        self.dimension = 2 * count + 2
        self.projection = np.zeros((self.dimension, 4 + 2 * count))
        self.right_inverse = np.zeros((4 + 2 * count, self.dimension))
        self.effect_names = []
        for occasion_index, occasion in enumerate(self.occasions):
            for parameter_index, parameter in enumerate(("CL", "V1")):
                row = 2 * occasion_index + parameter_index
                self.projection[row, parameter_index] = 1
                self.projection[row, 4 + row] = 1
                self.right_inverse[4 + row, row] = 1
                self.effect_names.append(f"iiv.{parameter}+iov.{occasion}.{parameter}")
        for row, column, name in ((2 * count, 2, "V2"), (2 * count + 1, 3, "ka")):
            self.projection[row, column] = self.right_inverse[column, row] = 1
            self.effect_names.append(f"iiv.{name}")
        if not np.array_equal(self.projection @ self.right_inverse, np.eye(self.dimension)):
            raise ValueError("invalid exact Gaussian projection right inverse")
        self.mean = self.projection @ target.distribution.mean
        self.covariance = self.projection @ target.distribution.covariance @ self.projection.T
        self.density = multivariate_normal(mean=self.mean, cov=self.covariance, allow_singular=False)

    def metadata(self):
        """Function: Export exact representation and proof boundaries
        Inputs: None Outputs: Maps, names, covariance and source proof
        """
        return {"dimension": self.dimension, "original_dimension": len(self.original.distribution.mean),
                "representation": "exact_gaussian_projection", "mean": self.mean.tolist(),
                "effect_names": self.effect_names, "original_effect_names":
  effect_names(self.original.distribution.effect_keys),
                "original_effect_keys": [list(key) for key in self.original.distribution.effect_keys],
                "occasion_ids": self.occasions, "projection": self.projection.tolist(),
                "right_inverse": self.right_inverse.tolist(), "covariance": self.covariance.tolist(),
                "proof": {"kind": "audited_source_identity_and_linear_gaussian_pushforward",
                          "reference_source_sha256": REFERENCE_SHA256, "binding_sha256": BINDING_SHA256,
                          "likelihood_dependence": "CL/V1 IIV+IOV per occasion; V2/ka IIV",
                          "integrated_null_dimension": 2, "approximation": False}}

    def log_joint(self, coordinates):
        """Function: Evaluate the exact normalized reduced target
        Inputs: Finite effective coordinates
        Outputs: Log likelihood plus pushed Gaussian log density
        """
        point = np.asarray(coordinates, dtype=float)
        if point.shape != (self.dimension,) or not np.all(np.isfinite(point)):
            raise ValueError(f"projection expects {self.dimension} finite coordinates")
        return math.fsum((self.original.likelihood(self.right_inverse @ point),
                          float(self.density.logpdf(point))))


class ExternalScenario(ModelScenario):
    """Function: Bind shared raw inputs and inherit archived finalize
    Inputs: Raw JSON path and OXC/TAC
    Outputs: Source-prior scenario; never calls the old constructor
    """

    def __init__(self, raw_json_path, model_name):
        """Function: Validate source priors and bind package likelihoods
        Inputs: Raw path and exact model name
        Outputs: Full OXC or exactly projected TAC targets
        """
        if model_name not in ("OXC", "TAC"):
            raise ValueError("model_name must be OXC or TAC")
        self.source = Path(raw_json_path).resolve(strict=True)
        payload = self.source.read_bytes()
        self.source_sha256 = hashlib.sha256(payload).hexdigest()
        self.raw = json.loads(payload.decode("utf-8-sig"))
```

```
        if not isinstance(self.raw, dict):
            raise ValueError("raw JSON must contain an object")
        self.provenance = verify_sources()
        builder = importlib.import_module("shared.scenario_inputs")
        with warnings.catch_warnings(record=True) as recorded:
            warnings.simplefilter("always")
            inputs = builder.build_inputs(self.raw, model_name)
        self.input_warnings = [{"category": warning.category.__name__, "message": str(warning.message)}
                               for warning in recorded]
        for warning in recorded:
            warnings.warn(warning.message, warning.category, stacklevel=2)
        if isinstance(inputs, dict):
            case = {key: inputs[key] for key in
                    ("model", "individual_data", "observations", "candidate_states", "priors")}
        else:
            model, data, observations, states, priors = inputs
            case = dict(model=model, individual_data=data, observations=observations,
                        candidate_states=states, priors=priors)
        # "laplace" selects shared input validation only; no optimizer/integrator is called.
        # This identifier selects input validation only, not Laplace optimization or integration.
        checked = validate_inputs(**case, integration_method="laplace", numerical_options=None)
        self.model_name = model_name
        self.case_id = self.raw.get("case_id", self.raw.get("numeric_id", self.source.stem))
        self.case = {key: checked[key] for key in ("model", "individual_data", "observations", "candidate_states")}
        self.states, self.priors = checked["candidate_states"], checked["priors"]
        model, data = self.case["model"], self.case["individual_data"]
        self.effects = resolve_random_effects(iiv=model["iiv"], iov=model["iov"], parameters=model["parameters"],
                                              occasions=data["occasions"], dependence=model["random_effects_dependence"])
        request = prepare_prediction_records(checked["observations"], data)
        self.targets = {state["state_id"]: NamedModelTarget(
            computation._bind_likelihood(model=model, individual_data=data, observations=checked["observations"],
                                         candidate_state=state, random_effect_distribution=self.effects,
                                         prediction_request=request), self.effects) for state in self.states}
        self.representation = "full" if model_name == "OXC" else "exact_gaussian_projection"
        if model_name == "OXC":
            if self.effects.effect_keys != (("iiv", None, "CL"),):
                raise ValueError("external OXC route requires the full one-dimensional CL effect")
        else:
            self.targets = {key: TACGaussianProjection(target, model, data) for key, target in self.targets.items()}
        self.builder_sha256 = file_sha256(builder.__file__)

    def metadata(self):
        """Function: Preserve input provenance, source priors and coordinates
        Inputs: None Outputs: Complete JSON-compatible scenario metadata
        """
        target = next(iter(self.targets.values())).metadata()
        model, data = self.case["model"], self.case["individual_data"]
        return {**super().metadata(), "effect_names": target["effect_names"],
                "full_effect_names": effect_names(self.effects.effect_keys),
                "full_effect_keys": [list(key) for key in self.effects.effect_keys],
                "full_mean": self.effects.mean.tolist(), "full_covariance": self.effects.covariance.tolist(),
                "target_coordinates": target, "source_path": str(self.source), "source_sha256": self.source_sha256,
                "source_schema": self.raw.get("schema"), "source_numeric_id": self.raw.get("numeric_id"),
                "source_covariance": (self.raw["variability"]["joint_omega"] if self.model_name == "OXC"
                                      else self.raw["omega"]),
                "input_warnings": self.input_warnings,
                "prior_source": "raw.priors" if self.model_name == "OXC" else "raw.states[].prior",
                "builder_source_sha256": self.builder_sha256, "provenance": self.provenance,
                "parameters": model["parameters"], "covariates": data["covariates"],
                "occasions": data["occasions"], "initialization": data["initialization"],
                "dosing_prehistory": data["dosing_prehistory"], "residual_model": model["residual_model"],
                "observations": self.case["observations"], "candidate_states": self.states,
                "random_effects_dependence": model["random_effects_dependence"]}


def make_scenario(raw_json_path, model_name):
    """Function: Stable reticulate constructor
    Inputs: Raw JSON path, OXC/TAC
    Outputs: Validated external scenario
    """
    return ExternalScenario(raw_json_path, model_name)
```

bridge/model_bridge.py

model_bridge.py

```
"""Function: Reuse package reference targets and normalization.
Inputs: frozen OXC/TAC inputs and external results
Outputs: Python targets, full posterior and stage checks
No PK equations, optimizer, Hessian or GHQ algorithm are implemented here.
This module binds existing interfaces; it does not implement PK equations, optimization, Hessians or GHQ.
"""

import hashlib
import importlib.util
import json
import math
from pathlib import Path
```

```
import numpy as np
from pmx_discrete_posterior.interfaces.inputs import validate_inputs
from pmx_discrete_posterior.interfaces.results import validate_integration_result
from pmx_discrete_posterior.models.prediction import prepare_prediction_records
from pmx_discrete_posterior.models.random_effects import resolve_random_effects
from pmx_discrete_posterior.posterior import computation
from pmx_discrete_posterior.posterior.normalization import normalize
from target_coordinates import TACGaussianProjection

ROOT = Path(__file__).resolve().parents[1]
_BINDING_HASH = "f7b6f0fa45fae01f0b4dd39bf022b3b1e0667e2a5f42f8be3159ba1197fa3608"


def load_reference_inputs(model_name, case_id=None):
    """Function: Load inputs from the pinned example.
    Inputs: OXC or TAC; local immutable fixture
    Outputs: declared model inputs
    """
    if hashlib.sha256(Path(computation.__file__).read_bytes()).hexdigest() != _BINDING_HASH:
        raise RuntimeError("private likelihood binding differs from the audited 0.2.0 source")
    path = ROOT / "fixtures/reference_inputs/examples/reference_models.py"
    if hashlib.sha256(path.read_bytes()).hexdigest() !=
  "d4c07d73fc4521da244d6ed715fcda7e6cc03f3172488d931f39a09857c3e9cc":
        raise RuntimeError("reference model factory changed")
    spec = importlib.util.spec_from_file_location("frozen_reference_models", path)
    module = importlib.util.module_from_spec(spec)
    spec.loader.exec_module(module)
    case = module.load_reference(model_name)
    case["model"]["random_effects_dependence"] = "independent_blocks"
    case["model"]["residual_model"]["dependence"] = "conditionally_independent"
    if case_id is not None:
        if model_name != "TAC":
            raise ValueError("archived sensitivity input is TAC-only")
        manifest = json.loads((ROOT / "fixtures/tac_cases/manifest.json").read_text())
        if case_id not in manifest["cases"]:
            raise ValueError("TAC case not frozen for this verification")
        payload = (ROOT / "fixtures/tac_cases" / case_id / "input.json").read_bytes()
        if hashlib.sha256(payload).hexdigest() != manifest["cases"][case_id]["input_sha256"]:
            raise ValueError("TAC frozen case hash mismatch")
        raw = json.loads(payload)
        baseline = json.loads((ROOT / "fixtures/reference_inputs/tests/fixtures/TAC_C003.json").read_text())
        # Function: Map only frozen input changes.
        # Inputs: historical JSON Outputs: existing package model inputs
        allowed = {"numeric_id", "parameters", "omega", "residual_sd", "times", "observations"}
        if {k:v for k,v in raw.items() if k not in allowed} != {k:v for k,v in baseline.items() if k not in allowed}:
            raise ValueError("unsupported changes in TAC history, covariates or structure")
        if set(raw["parameters"]) != set(baseline["parameters"]):
            raise ValueError("TAC parameter structure changed")
        if len(raw["times"]) != len(raw["observations"]) or not raw["times"] or min(raw["times"]) < 12 or
  max(raw["times"]) >= 48:
            raise ValueError("observation times exceed the frozen occasion structure")
        model = case["model"]
        values, units = model["parameters"]["values"], model["parameters"]["units"]
        for key, value in raw["parameters"].items():
            if key == "WT":
                case["individual_data"]["covariates"][key] = value
            else:
                values[key] = value
        values["sigma"] = raw["residual_sd"]
        omega = np.asarray(raw["omega"], dtype=float)
        if omega.shape != (10,10):
            raise ValueError("expected full ten-dimensional TAC covariance")
        blocks = np.zeros((10,10))
        blocks[:4,:4] = omega[:4,:4]
        for offset in (4,6,8):
            if not np.array_equal(omega[offset:offset+2,offset:offset+2],omega[4:6,4:6]):
                raise ValueError("occasion-specific covariance differs from supported shared IOV block")
            blocks[offset:offset+2,offset:offset+2] = omega[4:6,4:6]
        if not np.array_equal(omega,blocks):
            raise ValueError("unsupported cross-block dependence")
        model["iiv"]["covariance_parameters"] = module._covariance_refs(omega[:4,:4],"omega",values,units)
        model["iov"]["covariance_parameters"] = module._covariance_refs(omega[4:6,4:6],"pi",values,units)
        case["observations"] = [dict(record_id=str(i),time=t,value=y,time_unit="h",value_unit="ng/mL")
                                for i,(t,y) in enumerate(zip(raw["times"],raw["observations"]))]
    return case


class ModelTarget:
    """Function: Combine existing likelihood and effect density.
    Inputs: bound likelihood, distribution
    Outputs: complete normalized log-joint callback
    """

    def __init__(self, likelihood, distribution):
        """Function: Store bindings.
        Inputs: likelihood, distribution Outputs: target
        """
        self.likelihood, self.distribution = likelihood, distribution

    def metadata(self):
        """Function: Export coordinate metadata.
```

```
        Inputs: none Outputs: dimension and start mean
        """
        return {"dimension": len(self.distribution.mean), "mean": self.distribution.mean.tolist()}

    def log_joint(self, eta):
        """Function: Evaluate joint density excluding discrete priors.
        Inputs: active effects in declared order
        Outputs: finite scalar or failure
        """
        eta = np.asarray(eta, dtype=float)
        if eta.shape != self.distribution.mean.shape or not np.all(np.isfinite(eta)):
            raise ValueError("invalid random-effect vector")
        return math.fsum((self.likelihood(eta), self.distribution.log_density(eta)))


class ModelScenario:
    """Function: Candidate set for verification.
    Inputs: optional explicit normalized priors
    Outputs: targets and existing posterior normalization
    """

    def __init__(self, priors_json=None, model_name="OXC", case_id=None, representation="full"):
        """Function: Validate and bind common inputs.
        Inputs: optional JSON priors, model name
        Outputs: validated scenario
        """
        case = load_reference_inputs(model_name, case_id)
        self.model_name = model_name
        self.case_id = case_id or "c003"
        if representation not in ("full","exact_gaussian_projection") or (model_name!="TAC" and representation!="full"):
            raise ValueError("unsupported target representation")
        self.representation = representation
        priors = json.loads(priors_json) if priors_json is not None else {s["state_id"]:
  1/len(case["candidate_states"]) for s in case["candidate_states"]}
        # Only use existing shared input validation; no Laplace calculation here.
        # Reuse input validation only; this identifier does not execute Laplace or an external integration method.
        checked = validate_inputs(**case, priors=priors, integration_method="laplace", numerical_options=None)
        self.case = {key: checked[key] for key in case}
        self.states, self.priors = checked["candidate_states"], checked["priors"]
        model, data = checked["model"], checked["individual_data"]
        self.effects = resolve_random_effects(iiv=model["iiv"], iov=model["iov"], parameters=model["parameters"],
                        occasions=data["occasions"], dependence=model["random_effects_dependence"])
        request = prepare_prediction_records(checked["observations"], data)
        self.targets = {s["state_id"]: ModelTarget(computation._bind_likelihood(model=model, individual_data=data,
            observations=checked["observations"], candidate_state=s, random_effect_distribution=self.effects,
            prediction_request=request), self.effects) for s in self.states}
        if representation == "exact_gaussian_projection":
            self.targets = {key:TACGaussianProjection(value) for key,value in self.targets.items()}

    def metadata(self):
        """Function: Export candidate IDs and priors.
        Inputs: none Outputs: ordered complete state metadata
        """
        return {"model_name": self.model_name, "case_id": self.case_id,
                "representation":self.representation,
                "state_ids": [s["state_id"] for s in self.states], "priors": self.priors,
                "events": {s["state_id"]: s["events"] for s in self.states},

  "full_dimension":len(self.effects.mean),"dimension":next(iter(self.targets.values())).metadata()["dimension"]}

    def target(self, state_id):
        """Function: Get a candidate callback.
        Inputs: exact state_id Outputs: target, no prior weighting
        """
        return self.targets[state_id]

    def finalize(self, rows_json, settings_json):
        """Function: Validate external results and reuse normalization.
        Inputs: complete state-result mapping and stages
        Outputs: all rows, posterior history and final status; no partial normalization
        """
        rows, settings = json.loads(rows_json), json.loads(settings_json)
        mode = settings.get("acceptance_mode", "stability_required")
        output = {"status": "failed", "backend": "R aghq", "state_results": rows,
                  "posterior": None, "posterior_stages": [], "failure_reason": None,
                  "acceptance_mode": mode, "stability_passed": None,
                  "absolute_error_certified": False, "diagnostics": {}}
        try:
            if mode not in ("stability_required", "external_comparison"):
                raise ValueError("invalid acceptance mode")
            if set(rows) != set(self.priors):
                raise ValueError("external results must contain every candidate exactly once")
            active = [key for key, prior in self.priors.items() if prior > 0]
            log_stability = {}
            for key, row in rows.items():
                if self.priors[key] == 0:
                    if row.get("status") != "skipped_zero_prior" or row.get("log_marginal_likelihood") is not None:
                        raise ValueError("zero-prior state must be explicitly skipped")
                    continue
                validate_integration_result(row)
                if row["status"] != "success":
```

```
                    raise ValueError(f"positive-prior candidate failed: {key}; no posterior normalization")
                history = row.get("method_details", {}).get("order_results", [])
                if [entry["order"] for entry in history] != settings["orders"]:
                    raise ValueError("incomplete or mismatched precision stages")
                logs = [entry["log_marginal_likelihood"] for entry in history]
                if len(logs) < 3 or not all(math.isfinite(x) for x in logs):
                    raise ValueError("invalid log-integral history")
                for value in logs:
                    validate_integration_result({"status": "success", "log_marginal_likelihood": value,
                                                 "failure_reason": None})
                if logs[-1] != row["log_marginal_likelihood"]:
                    raise ValueError("final log integral differs from history")
                log_stability[key] = max(abs(logs[-1] - logs[-2]), abs(logs[-2] - logs[-3])) <=
  settings["log_integral_atol"]
            previous = None
            for index, order in enumerate(settings["orders"]):
                logs = {key: (rows[key]["method_details"]["order_results"][index]["log_marginal_likelihood"]
                              if key in active else None) for key in self.priors}
                posterior = normalize(logs, self.priors, candidate_states=self.states)
                change = max(abs(posterior[key] - previous[key]) for key in posterior) if previous is not None else None
                output["posterior_stages"].append({"order": order, "posterior": posterior, "max_change": change})
                previous = posterior
            # Function: Keep order sensitivity beside external results.
            # Inputs: complete same-order posteriors
            # Outputs: diagnostic flags; optional historical strict rejection
            posterior_stable = len(output["posterior_stages"]) >= 3 and all(
                row["max_change"] <= settings["posterior_atol"] for row in output["posterior_stages"][-2:])
            output["diagnostics"] = {
                "log_integral_stability_by_state": log_stability,
                "log_integral_stability_passed": all(log_stability.values()),
                "posterior_stability_passed": posterior_stable,
                "log_integral_atol": settings["log_integral_atol"],
                "posterior_atol": settings["posterior_atol"],
                "interpretation": "Order sensitivity only; not an absolute error certificate or agreement requirement.",
            }
            output["stability_passed"] = all(log_stability.values()) and posterior_stable
            if mode == "stability_required":
                if not all(log_stability.values()):
                    raise ValueError("log-integral stability failed")
                if not posterior_stable:
                    raise ValueError("last two posterior changes exceed tolerance")
            output.update(status="success", posterior=previous)
        except (ValueError, TypeError, KeyError, FloatingPointError) as exc:
            output["failure_reason"] = f"{type(exc).__name__}: {exc}"
        return output


def make_scenario(priors_json=None, model_name="OXC", case_id=None, representation="full"):
    """Function: Create a reference verification scenario.
    Inputs: optional priors JSON, OXC/TAC
    Outputs: validated reference scenario
    """
    return ModelScenario(priors_json, model_name, case_id, representation)
```

bridge/target_coordinates.py

target_coordinates.py

```
"""Function: Represent exact effective TAC Gaussian coordinates.
Inputs: full bound likelihood and Gaussian distribution
Outputs: equivalent eight-dimensional integral target
This changes target coordinates only; it contains no quadrature implementation.
Only the mathematically equivalent coordinates change; quadrature and PK equations are unchanged.
"""
import math
import numpy as np
from scipy.stats import multivariate_normal


class TACGaussianProjection:
    """Function: Integrate out two likelihood-null directions.
    Inputs: TAC ModelTarget with ten declared coordinates
    Outputs: L(Bu) times normalized N8(u;AΩA^T)
    """

    def __init__(self, target):
        """Function: Establish an explicit linear map.
        Inputs: original target Outputs: A, right inverse B, pushed density
        """
        expected = [("iiv",None,name) for name in ("CL","V1","V2","ka")]
        expected += [("iov",f"occ_{i}",name) for i in range(3) for name in ("CL","V1")]
        if tuple(expected) != target.distribution.effect_keys:
            raise ValueError("exact TAC projection requires the frozen ten-coordinate structure")
        self.original = target
        self.projection = np.zeros((8,10))
        self.right_inverse = np.zeros((10,8))
        for occasion in range(3):
            for parameter in range(2):
                row = 2*occasion+parameter
```

```
                self.projection[row,parameter] = 1
                self.projection[row,4+row] = 1
                self.right_inverse[4+row,row] = 1
        self.projection[6,2] = self.projection[7,3] = 1
        self.right_inverse[2,6] = self.right_inverse[3,7] = 1
        if not np.array_equal(self.projection @ self.right_inverse,np.eye(8)):
            raise ValueError("invalid Gaussian projection right inverse")
        self.mean = self.projection @ target.distribution.mean
        self.covariance = self.projection @ target.distribution.covariance @ self.projection.T
        self.density = multivariate_normal(mean=self.mean,cov=self.covariance,allow_singular=False)

    def metadata(self):
        """Function: Export exact coordinate representation.
        Inputs: none Outputs: dimensions, maps and covariance
        """
        return dict(dimension=8,mean=self.mean.tolist(),original_dimension=10,
                    representation="exact_gaussian_projection",projection=self.projection.tolist(),
                    right_inverse=self.right_inverse.tolist(),covariance=self.covariance.tolist())

    def log_joint(self, coordinates):
        """Function: Evaluate equivalent complete joint density.
        Inputs: eight finite effective effects
        Outputs: scalar normalized joint log density
        """
        u = np.asarray(coordinates,dtype=float)
        if u.shape!=(8,) or not np.all(np.isfinite(u)):
            raise ValueError("projection expects eight finite coordinates")
        return math.fsum((self.original.likelihood(self.right_inverse @ u),float(self.density.logpdf(u))))
```

shared/scenario_inputs.py

scenario_inputs.py

```
"""Function: Experiment-only raw OXC/TAC adapter.
Inputs: Source JSON and model name.
Outputs: Five scientific keyword arguments.

No integration, file access, reference-case substitution, unit conversion, prior
normalization, covariance rank repair, or production-code changes occur here. Source
IDs and display labels are provenance, not instructions for constructing doses.
TAC v1 defines oral mg doses, h clocks, ng/mL observations, proportional Gaussian
residuals, and the reference callback's conditional steady-state convention.
Archived TAC covariance construction can differ across triangles by 1-2 ULP.
Only nonzero, same-sign pairs within two ULP are canonicalized to the source
upper triangle, with a CovarianceRoundoffWarning recording every changed pair.
Callers should capture these warnings in their experiment audit logs. All other
asymmetry, cross-block dependence and unequal IOV blocks are rejected.
"""

import math
import warnings

import numpy as np

from pmx_discrete_posterior.interfaces.inputs.candidate_states import (
    validate_candidate_history,
    validate_candidate_states,
)
from pmx_discrete_posterior.interfaces.inputs.individual_data import validate_individual_data
from pmx_discrete_posterior.interfaces.inputs.model_inputs import validate_model
from pmx_discrete_posterior.interfaces.inputs.observations import validate_observations
from pmx_discrete_posterior.interfaces.inputs.priors import validate_priors
from pmx_discrete_posterior.interfaces.validation import (
    InputError, fields, identifier, numeric_array, scalar, validate_covariance,
)
from pmx_discrete_posterior.models.prediction.linear import LinearPredictor
from pmx_discrete_posterior.models.random_effects import resolve_random_effects

from .reference_models import (
    _covariance_refs, _event, _oxc_relations, _oxc_system, _tac_relations, _tac_system,
)


_OXC_IIV = ["Clearance IIV"]
_TAC_IIV = ["Clearance IIV", "Central-volume IIV", "Peripheral-volume IIV", "Absorption-rate IIV"]
_OXC_RELATIONSHIPS = {
    "CL": "CL_p*(BW/70)**(k0-kmax*BW**gamma/(k50**gamma+BW**gamma))*exp(eta_CL)",
    "V": "V_p", "ka": "ka", "concentration": "A_central/V_p",
}


class CovarianceRoundoffWarning(UserWarning):
    """Function: Report archive covariance roundoff canonicalization.
    Inputs: Coordinates and original triangle values.
    Outputs: Capturable warning; source object unchanged.
    """


def _keys(value, required, path):
```

```
    """Function: Reject unknown or missing fields.
    Inputs: Mapping, key string, path.
    Outputs: Checked mapping or InputError.
    """
    return fields(value, required.split(), name=path)


def _equal(actual, expected, path):
    """Function: Check fixed semantics and duplicate declarations.
    Inputs: Actual value, expected value, path.
    Outputs: None; mismatch raises InputError.
    """
    if (isinstance(expected, bool) and type(actual) is not bool) or actual != expected:
        raise InputError(f"{path}: unsupported or inconsistent value; expected {expected!r}")


def _sequence(value, path, *, nonempty=False):
    """Function: Validate a raw sequence.
    Inputs: Value, path, nonempty constraint.
    Outputs: Original sequence or InputError.
    """
    if not isinstance(value, (list, tuple)) or (nonempty and not value):
        raise InputError(f"{path} must be a {'nonempty ' if nonempty else ''}sequence")
    return value


def _number_map(raw, keys, path, *, positive=(), nonnegative=()):
    """Function: Validate and retain source parameters.
    Inputs: Mapping, keys and bounds.
    Outputs: Finite numeric mapping.
    """
    _keys(raw, keys, path)
    return {k: scalar(v, name=f"{path}.{k}", positive=k in positive,
                      minimum=0 if k in nonnegative else None) for k, v in raw.items()}


def _matrix(raw, dimension, path):
    """Function: Validate the complete covariance.
    Inputs: Matrix, dimension, path.
    Outputs: Covariance; <=2 ULP canonicalization warns.
    """
    matrix = numeric_array(raw, ndim=2, name=path)
    if matrix.shape != (dimension, dimension):
        raise InputError(f"{path} must have shape ({dimension}, {dimension})")
    changes = []
    for i in range(dimension):
        for j in range(i+1, dimension):
            upper, lower = float(matrix[i, j]), float(matrix[j, i])
            if upper == lower:
                continue
            if (upper == 0 or lower == 0 or (upper > 0) != (lower > 0)
                    or abs(upper-lower) > 2*max(math.ulp(upper), math.ulp(lower))):
                raise InputError(f"{path} is asymmetric at ({i}, {j}); outside two-ULP roundoff policy")
            changes.append(f"({i},{j}): upper={upper!r}, lower={lower!r}")
            matrix[j, i] = upper
    if changes:
        warnings.warn(f"{path}: canonicalized lower to source upper triangle (<=2 ULP); " + "; ".join(changes),
                      CovarianceRoundoffWarning, stacklevel=3)
    validate_covariance(matrix)
    return matrix


def _slots(raw, shared):
    """Function: Validate nominal window boundaries.
    Inputs: Slots and shared last-dose time.
    Outputs: Strictly increasing slots after shared history.
    """
    slots = [scalar(t, name="slot") for t in _sequence(raw, "slots", nonempty=True)]
    if any(b <= a for a, b in zip([shared, *slots], slots)):
        raise InputError("slots must be strictly increasing after shared_last")
    return slots


def _raw_event(raw, *, oxc, offset=0):
    """Function: Map actual dose time and amount.
    Inputs: Source event, model flag, local offset.
    Outputs: Public event and nominal slot.
    """
    keys = "time_h amount_mg nominal_slot_h"
    if oxc:
        keys += " event_id local_time_h local_nominal_slot_h route compartment"
    _keys(raw, keys, "dose event")
    time = scalar(raw["time_h"], name="time_h")
    nominal = scalar(raw["nominal_slot_h"], name="nominal_slot_h")
    amount = scalar(raw["amount_mg"], name="amount_mg", minimum=0)
    if oxc:
        identifier(raw["event_id"], name="event_id")
        _equal(raw["route"], "oral", "event.route")
        _equal(raw["compartment"], "depot", "event.compartment")
        _equal(scalar(raw["local_time_h"]), time + offset, "event.local_time_h")
        _equal(scalar(raw["local_nominal_slot_h"]), nominal + offset, "event.local_nominal_slot_h")
```

```
    return _event(time, amount), nominal


def _states(raw_states, *, oxc, slots, offset=0):
    """Function: Preserve candidate order and explicit events.
    Inputs: States, model, slots, offset.
    Outputs: Public candidates, never inferred from labels.
    """
    result = []
    for state in _sequence(raw_states, "states", nonempty=True):
        keys = "state_event_id manuscript_id machine_bitstring "
        keys += ("machine_chronological_bits nominal_slots_h taken_by_slot dose_events"
                 if oxc else "prior events")
        _keys(state, keys, "state")
        identifier(state["manuscript_id"], name="manuscript_id")
        if oxc:
            _equal(state["nominal_slots_h"], slots, "state.nominal_slots_h")
            for taken in _sequence(state["taken_by_slot"], "taken_by_slot"):
                _keys(taken, "nominal_time_h taken", "taken_by_slot")
                if scalar(taken["nominal_time_h"]) not in slots or type(taken["taken"]) is not bool:
                    raise InputError("invalid taken_by_slot metadata")
        events = []
        for event in _sequence(state["dose_events" if oxc else "events"], "events"):
            mapped, nominal = _raw_event(event, oxc=oxc, offset=offset)
            if nominal not in slots:
                raise InputError("event nominal_slot_h is not a declared slot")
            events.append(mapped)
        result.append({"state_id": state["state_event_id"], "events": events})
    return validate_candidate_states(result)


def _history(shared, dose, period, occasions, *, historical_dose=None):
    """Function: Build conditional steady state and common history.
    Inputs: Shared dose, period, occasions, historical dose.
    Outputs: Dosing prehistory and initialization.
    """
    event = _event(shared, dose)
    periodic = _event(shared, dose if historical_dose is None else historical_dose)
    periodic["offset"] = 0
    del periodic["time"]
    return {
        "dosing_prehistory": {
            "events": [event],
            "regimen": {"period": period, "anchor_time": shared, "time_unit": "h", "events": [periodic]},
        },
        "initialization": {
            "mode": "steady_state", "start_time": shared, "event_phase": "before_events",
            "steady_state": {"parameter_rule": "fixed_reference",
                             "reference_occasion_id": occasions[0]["occasion_id"] if occasions else None},
        },
    }


def _oxc_initialization(raw):
    """Function: Validate OXC steady history and time axes.
    Inputs: Raw initialization.
    Outputs: History values, slots and offset.
    """
    _keys(raw, "definition_status interval_h shared_last_dose_h initial_state_time_h initial_state_phase "
          "initial_dose_event earlier_history explicit_occasion_slots_h no_0h_dose_in_one_dose_window "
          "numerical_initial_state time_axes method conditioning iov_present", "initialization")
    for key, expected in {
        "definition_status": "author_approved", "method": "analytic_steady_state",
        "conditioning": "individual_eta_CL", "iov_present": False,
        "initial_state_phase": "immediately_before_shared_last_dose",
        "numerical_initial_state": None, "explicit_occasion_slots_h": [],
    }.items():
        _equal(raw[key], expected, f"initialization.{key}")
    shared = scalar(raw["shared_last_dose_h"])
    period = scalar(raw["interval_h"], positive=True)
    _equal(scalar(raw["initial_state_time_h"]), shared, "initial_state_time_h")
    earlier = _keys(raw["earlier_history"], "interval_h dose_mg last_implicit_dose_h extends_to", "earlier_history")
    _equal(scalar(earlier["interval_h"]), period, "earlier_history.interval_h")
    _equal(scalar(earlier["last_implicit_dose_h"]), shared-period, "last_implicit_dose_h")
    _equal(earlier["extends_to"], "minus_infinity", "earlier_history.extends_to")
    historical_dose = scalar(earlier["dose_mg"], minimum=0)
    axes = _keys(raw["time_axes"], "time_h_axis local_axis manuscript_to_local_offset_h "
                 "manuscript_candidate_slots_h local_candidate_slots_h manuscript_shared_last_h local_shared_last_h "
                 "manuscript_last_implicit_dose_h local_last_implicit_dose_h "
                 "all_events_and_observations_shift_equally native_runtime_axis_adapter_verified", "time_axes")
    _equal(axes["time_h_axis"], "manuscript_study", "time_h_axis")
    _equal(axes["local_axis"], "earliest_candidate_slot_is_zero", "local_axis")
    _equal(axes["all_events_and_observations_shift_equally"], True, "all_events_and_observations_shift_equally")
    if type(axes["native_runtime_axis_adapter_verified"]) is not bool:
        raise InputError("native_runtime_axis_adapter_verified must be boolean")
    slots = _slots(axes["manuscript_candidate_slots_h"], shared)
    offset = scalar(axes["manuscript_to_local_offset_h"])
    _equal(offset, -slots[0], "manuscript_to_local_offset_h")
    _equal(axes["local_candidate_slots_h"], [t+offset for t in slots], "local_candidate_slots_h")
    for key, expected in {
        "manuscript_shared_last_h": shared, "local_shared_last_h": shared+offset,
```

```
        "manuscript_last_implicit_dose_h": shared-period, "local_last_implicit_dose_h": shared-period+offset,
    }.items():
        _equal(scalar(axes[key]), expected, f"time_axes.{key}")
    _equal(raw["no_0h_dose_in_one_dose_window"], len(slots) == 1 and 0 not in slots,
           "no_0h_dose_in_one_dose_window")
    event, nominal = _raw_event(raw["initial_dose_event"], oxc=True, offset=offset)
    _equal(event["time"], shared, "initial_dose_event.time_h")
    _equal(nominal, shared, "initial_dose_event.nominal_slot_h")
    return shared, event["amount"], period, historical_dose, slots, offset


def _assemble(values, units, iiv_names, iiv_matrix, iov_matrix, relations, system, error_type):
    """Function: Reuse reference callbacks with explicit independence.
    Inputs: Parameters, units, covariance and original callbacks.
    Outputs: Public model without integration.
    """
    iov = None
    if iov_matrix is not None:
        iov = {"names": ["CL", "V1"], "distribution": "normal", "occasion_key": "occasion_id",
               "covariance_parameters": _covariance_refs(iov_matrix, "pi", values, units)}
    iiv = {"names": iiv_names, "distribution": "normal",
           "covariance_parameters": _covariance_refs(iiv_matrix, "omega", values, units)}
    role = "sd" if error_type == "additive" else "coefficient_sd"
    return {
        "parameters": {"values": values, "units": units},
        "parameter_model": {"covariate_names": [], "relations": relations},
        "iiv": iiv, "iov": iov, "random_effects_dependence": "independent_blocks",
        "residual_model": {"distribution": "normal", "error_type": error_type,
                           "parameter_names": {role: "sigma"}, "dependence": "conditionally_independent"},
        "predict": LinearPredictor(system),
    }


def _oxc(raw):
    """Function: Map an OXC experiment directly.
    Inputs: Complete raw OXC object.
    Outputs: Five groups without prior rewriting.
    """
    _keys(raw, "model initialization variability residual candidate_states priors schema observations", "OXC")
    _equal(raw["schema"], "oxc_numeric_source_ruv_v1", "schema")
    source = _keys(raw["model"], "model_id compartments analyte dose_unit concentration_unit parameters fixed "
                   "relationships covariates_constant_over_record", "model")
    for key, expected in {
        "model_id": "Lin_OXC_MHD_Hill219", "compartments": ["depot", "central"], "analyte": "MHD",
        "dose_unit": "mg", "concentration_unit": "mg/L", "covariates_constant_over_record": True,
        "relationships": _OXC_RELATIONSHIPS,
    }.items():
        _equal(source[key], expected, f"model.{key}")
    parameters = _number_map(source["parameters"], "CL_p V_p ka BW", "model.parameters",
                             positive=("CL_p", "V_p", "ka", "BW"))
    fixed = _number_map(source["fixed"], "k0 kmax k50 gamma normalization_weight", "model.fixed",
                        positive=("k50", "normalization_weight"))
    _equal(fixed["normalization_weight"], 70, "fixed.normalization_weight (source relationship uses BW/70)")
    covariates = {"BW": parameters.pop("BW")}
    variability = _keys(raw["variability"], "iiv_order iiv_omega iiv_correlation iov_order iov_omega_per_occasion "
                        "iov_occasion_ids joint_order joint_omega source_reference_log_variances distribution",
  "variability")
    for key, expected in {
        "iiv_order": _OXC_IIV, "joint_order": _OXC_IIV, "iiv_correlation": [[1]],
        "iov_order": [], "iov_omega_per_occasion": [], "iov_occasion_ids": [],
        "distribution": "zero_mean_multivariate_normal_log_effects",
    }.items():
        _equal(variability[key], expected, f"variability.{key}")
    source_variances = fields(variability["source_reference_log_variances"], _OXC_IIV,
                              name="source_reference_log_variances")
    scalar(source_variances["Clearance IIV"], minimum=0)
    omega = _matrix(variability["joint_omega"], 1, "joint_omega")
    if not np.array_equal(omega, _matrix(variability["iiv_omega"], 1, "iiv_omega")):
        raise InputError("iiv_omega and joint_omega disagree")
    residual = _keys(raw["residual"], "model sd variance additive_component_present residual_floor normal_zero_mean",
  "residual")
    for key, expected in {"model": "additive", "additive_component_present": True,
                          "residual_floor": None, "normal_zero_mean": True}.items():
        _equal(residual[key], expected, f"residual.{key}")
    sigma = scalar(residual["sd"], positive=True)
    variance = scalar(residual["variance"], minimum=0)
    if not math.isclose(variance, sigma*sigma, rel_tol=8*np.finfo(float).eps, abs_tol=0):
        raise InputError("residual.variance must equal residual.sd squared")
    values = {**fixed, **parameters, "sigma": sigma}
    units = {k: "1" for k in values}
    units.update(CL_p="L/h", V_p="L", ka="1/h", k50="kg", normalization_weight="kg", sigma="mg/L")
    model = _assemble(values, units, ["CL"], omega, None, _oxc_relations, _oxc_system, "additive")
    shared, dose, period, historical_dose, slots, offset = _oxc_initialization(raw["initialization"])
    states = _states(raw["candidate_states"], oxc=True, slots=slots, offset=offset)
    priors = {}
    labels = {s["state_event_id"]: s["manuscript_id"] for s in raw["candidate_states"]}
    for prior in _sequence(raw["priors"], "priors", nonempty=True):
        _keys(prior, "state_event_id manuscript_id probability", "prior")
        key = identifier(prior["state_event_id"])
        if key in priors:
            raise InputError("duplicate prior state_event_id")
```

```
        if key not in labels or labels[key] != prior["manuscript_id"]:
            raise InputError("prior labels do not match candidate states")
        priors[key] = prior["probability"]
    observations = []
    for i, record in enumerate(_sequence(raw["observations"], "observations", nonempty=True)):
        _keys(record, "time_h concentration_mg_L", "observation")
        observations.append({"record_id": str(i), "time": record["time_h"], "value": record["concentration_mg_L"],
                             "time_unit": "h", "value_unit": "mg/L"})
    data = {"covariates": covariates, "occasions": [],
            **_history(shared, dose, period, [], historical_dose=historical_dose)}
    return dict(model=model, individual_data=data, observations=observations, candidate_states=states, priors=priors)


def _tac(raw):
    """Function: Map full TAC block structure and occasions.
    Inputs: Complete raw TAC object.
    Outputs: Five groups; unrepresentable covariance rejected.
    """
    _keys(raw, "schema numeric_id parameters fixed slots shared_last shared_dose tau states omega eta_order "
          "residual_sd times observations", "TAC")
    _equal(raw["schema"], "tac_resolved_v1", "schema")
    identifier(raw["numeric_id"], name="numeric_id")
    parameters = _number_map(raw["parameters"], "CL V1 Q V2 ka lag WT", "parameters",
                             positive=("CL", "V1", "Q", "V2", "ka", "WT"), nonnegative=("lag",))
    fixed = _number_map(raw["fixed"], "X HEM POD Pred theta_CYP3A5 theta_HEM theta_POD theta_Pred "
                        "POD_cap WT_exponent normalization_weight", "fixed",
                        positive=("theta_CYP3A5", "normalization_weight"))
    covariates = {k: fixed.pop(k) for k in ("X", "HEM", "POD", "Pred")}
    covariates["WT"] = parameters.pop("WT")
    values = {**fixed, **parameters, "sigma": scalar(raw["residual_sd"], positive=True)}
    units = {k: "1" for k in values}
    units.update(CL="L/h", V1="L", V2="L", Q="L/h", ka="1/h", lag="h", normalization_weight="kg",
                 theta_POD="1/day", POD_cap="day", theta_Pred="mL/ng")
    shared = scalar(raw["shared_last"])
    dose, period = scalar(raw["shared_dose"], minimum=0), scalar(raw["tau"], positive=True)
    slots = _slots(raw["slots"], shared)
    times, concentrations = numeric_array(raw["times"], name="times"), numeric_array(raw["observations"],
  name="observations")
    if not len(times) or times.shape != concentrations.shape:
        raise InputError("times and observations must be nonempty and have equal lengths")
    boundaries = [shared, *slots, max(48, float(max(times))+1, slots[-1]+1)]
    occasions = [{"occasion_id": f"occ_{i}", "start": a, "end": b}
                 for i, (a, b) in enumerate(zip(boundaries[:-1], boundaries[1:]))]
    order = [*_TAC_IIV]
    for start in boundaries[:-1]:
        order.extend([f"Clearance IOV@occ_{start:g}", f"Central-volume IOV@occ_{start:g}"])
    _equal(raw["eta_order"], order, "eta_order")
    omega = _matrix(raw["omega"], 4+2*len(occasions), "omega")
    represented = np.zeros_like(omega)
    represented[:4, :4] = omega[:4, :4]
    for i in range(4, len(omega), 2):
        represented[i:i+2, i:i+2] = omega[4:6, 4:6]
    if not np.array_equal(omega, represented):
        raise InputError("omega cannot be represented by 0.2.0 independent_blocks: "
                         "cross-block covariance or unequal occasion-specific IOV blocks")
    model = _assemble(values, units, ["CL", "V1", "V2", "ka"], omega[:4, :4], omega[4:6, 4:6],
                      _tac_relations, _tac_system, "proportional")
    states = _states(raw["states"], oxc=False, slots=slots)
    observations = [{"record_id": str(i), "time": float(t), "value": float(y), "time_unit": "h", "value_unit": "ng/mL"}
                    for i, (t, y) in enumerate(zip(times, concentrations))]
    data = {"covariates": covariates, "occasions": occasions, **_history(shared, dose, period, occasions)}
    return dict(model=model, individual_data=data, observations=observations, candidate_states=states,
                priors={s["state_event_id"]: s["prior"] for s in raw["states"]})


def build_inputs(raw, model_name):
    """Function: Adapt raw experiments to public posterior inputs.
    Inputs: Raw JSON mapping and case-insensitive OXC/TAC.
    Outputs: Five keyword argument groups:
        Five kwargs for compute_posterior; caller supplies integration_method and numerical_options.

    Reject unknown keys, unsupported source semantics and unrepresentable random
    effects with InputError. Preserve exact zero priors, source ordering and
    strict zero-variance coordinates. Never mutate raw or load a C003 fixture.
    """
    if not isinstance(model_name, str) or model_name.upper() not in ("OXC", "TAC"):
        raise InputError("model_name must be OXC or TAC")
    result = _oxc(raw) if model_name.upper() == "OXC" else _tac(raw)
    result["model"]["parameter_model"]["covariate_names"] = list(result["individual_data"]["covariates"])
    model = validate_model(result["model"])
    observations = validate_observations(result["observations"])
    data = validate_individual_data(result["individual_data"], model=model, observations=observations)
    states = [validate_candidate_history(s, initialization_start=data["initialization"]["start_time"],
                                         common_events=data["dosing_prehistory"]["events"])
              for s in result["candidate_states"]]
    resolve_random_effects(iiv=model["iiv"], iov=model["iov"], parameters=model["parameters"],
                           occasions=data["occasions"], dependence=model["random_effects_dependence"])
    return dict(model=model, individual_data=data, observations=observations, candidate_states=states,
                priors=validate_priors(result["priors"], candidate_states=states))
```

shared/reference_models.py

reference_models.py

```
"""Function: Demonstrate reference inputs through the new model layer.
Inputs: Frozen local C003 reference JSON.
Outputs: Explicit inputs, predictions and conditional likelihoods only.
"""

import json
import hashlib
import math
from pathlib import Path

import numpy as np
from scipy.special import expit

from pmx_discrete_posterior.interfaces.validation import InputError, scalar
from pmx_discrete_posterior.models.likelihood import conditional_log_likelihood
from pmx_discrete_posterior.models.parameter_resolution import resolve_parameters
from pmx_discrete_posterior.models.prediction import evaluate_prediction, prepare_prediction_records
from pmx_discrete_posterior.models.prediction.linear import LinearPredictor
from pmx_discrete_posterior.models.random_effects import resolve_random_effects
from pmx_discrete_posterior.models.residuals import resolve_residuals


def _event(time, amount):
    """Function: Explicit oral event.
    Inputs: h, mg. Outputs: Event mapping.
    """
    return dict(time=time, amount=amount, amount_unit="mg", route="oral", compartment="depot")


def _covariance_refs(matrix, prefix, values, units):
    """Function: Store frozen covariance in the single parameter source.
    Inputs: Symmetric matrix and parameter maps.
    Outputs: Reference matrix and updated maps.
    """
    matrix = np.asarray(matrix)
    if not np.array_equal(matrix, matrix.T):
        raise InputError("reference covariance is not symmetric")
    refs = []
    for i in range(len(matrix)):
        row = []
        for j in range(len(matrix)):
            if matrix[i, j] == 0:
                row.append(None)
            else:
                name = f"{prefix}_{min(i,j)}_{max(i,j)}"
                values[name], units[name] = float(matrix[i, j]), "1"
                row.append(name)
        refs.append(row)
    return refs


def _oxc_relations(*, parameters, covariates, iiv_values, iov_values, occasions, evaluation_interval):
    """Function: OXC weight and clearance IIV relationship.
    Inputs: Parameters, BW and current CL effect.
    Outputs: Resolved CL, V and ka segment.
    """
    p, bw = parameters["values"], scalar(covariates["BW"], positive=True)
    exponent = p["k0"] - p["kmax"] * expit(p["gamma"] * math.log(bw / p["k50"]))
    cl = p["CL_p"] * (bw/p["normalization_weight"])**exponent * math.exp(iiv_values["CL"])
    values = {"CL": scalar(cl, positive=True), "V": scalar(p["V_p"], positive=True), "ka": scalar(p["ka"],
  positive=True)}
    return {"units": {"CL": "L/h", "V": "L", "ka": "1/h"}, "segments": [
        {"start": evaluation_interval[0], "end": max(evaluation_interval[1], evaluation_interval[0]+1),
         "occasion_id": None, "values": values}]}


def _oxc_system(*, values, units):
    """Function: OXC one-compartment oral system.
    Inputs: Individual kinetics and units.
    Outputs: Linear compartment system.
    """
    if units != {"CL": "L/h", "V": "L", "ka": "1/h"}:
        raise InputError("OXC units differ")
    cl, volume, ka = (scalar(values[k], positive=True) for k in ("CL", "V", "ka"))
    return {"state_names": ["depot", "central"], "amount_unit": "mg", "time_unit": "h", "concentration_unit": "mg/L",
            "matrix": [[-ka, 0], [ka, -cl/volume]], "observation_vector": [0, 1/volume],
            "inputs": {("oral", "depot"): {"fractions": [1, 0], "lag": 0}}}


def _tac_relations(*, parameters, covariates, iiv_values, iov_values, occasions, evaluation_interval):
    """Function: TAC covariate, IIV and IOV relationships.
    Inputs: Parameters, covariates and occasion-indexed effects.
    Outputs: Occasion-specific resolved kinetics.
    """
    p, c, segments = parameters["values"], covariates, []
    for occasion in occasions:
```

```
        io = iov_values[occasion["occasion_id"]]
        cl = (p["CL"] * p["theta_CYP3A5"]**c["X"] * (1+p["theta_HEM"]*(c["HEM"]-.33))
              * (c["WT"]/p["normalization_weight"])**p["WT_exponent"]
              * (1+p["theta_POD"]*(min(c["POD"], p["POD_cap"])-22.7)) * math.exp(iiv_values["CL"]+io["CL"]))
        v1 = p["V1"] * (1+p["theta_Pred"]*(c["Pred"]-155.5)) * math.exp(iiv_values["V1"]+io["V1"])
        values = {"CL": cl, "V1": v1, "V2": p["V2"]*math.exp(iiv_values["V2"]), "Q": p["Q"],
                  "ka": p["ka"]*math.exp(iiv_values["ka"]), "lag": p["lag"]}
        values = {k: scalar(v, positive=(k != "lag"), minimum=0) for k, v in values.items()}
        segments.append({**occasion, "values": values})
    return {"units": {"CL": "L/h", "V1": "L", "V2": "L", "Q": "L/h", "ka": "1/h", "lag": "h"}, "segments": segments}


def _tac_system(*, values, units):
    """Function: TAC two-compartment oral system.
    Inputs: Resolved kinetics.
    Outputs: System with mg amounts and ng/mL concentration.
    """
    if units != {"CL": "L/h", "V1": "L", "V2": "L", "Q": "L/h", "ka": "1/h", "lag": "h"}:
        raise InputError("TAC units differ")
    cl, v1, v2, q, ka = (scalar(values[k], positive=True) for k in ("CL", "V1", "V2", "Q", "ka"))
    return {"state_names": ["depot", "central", "peripheral"], "amount_unit": "mg", "time_unit": "h",
  "concentration_unit": "ng/mL",
            "matrix": [[-ka, 0, 0], [ka, -(cl+q)/v1, q/v2], [0, q/v1, -q/v2]],
            "observation_vector": [0, 1000/v1, 0],
            "inputs": {("oral", "depot"): {"fractions": [1, 0, 0], "lag": values["lag"]}}}


def load_reference(name):
    """Function: Load and explicitly map a local reference case.
    Inputs: OXC or TAC. Outputs: Complete model-layer inputs.
    """
    if name not in ("OXC", "TAC"):
        raise InputError("reference name must be OXC or TAC")
    source = Path(__file__).resolve().parents[1] / "tests" / "fixtures" / f"{name}_C003.json"
    expected_hash = {"OXC": "67d268cbc4e7268474753403ea195ec52a5ce95986e40772e2e1a8ab03db3d5a",
                     "TAC": "3aa3ae1408c8c8707e3644f1924217350847b3a3555d9dcec90b89a574ab14e6"}
    payload = source.read_bytes()
    if hashlib.sha256(payload).hexdigest() != expected_hash[name]:
        raise InputError("frozen reference changed; modify the returned explicit model inputs instead")
    raw = json.loads(payload.decode("utf-8"))
    if name == "OXC":
        values = {**raw["model"]["fixed"], **{k: v for k, v in raw["model"]["parameters"].items() if k != "BW"},
  "sigma": raw["residual"]["sd"]}
        units = {k: "1" for k in values}
        units.update(CL_p="L/h", V_p="L", ka="1/h", k50="kg", normalization_weight="kg", sigma="mg/L")
        iiv_names, covariates = ["CL"], {"BW": raw["model"]["parameters"]["BW"]}
        iiv_matrix = raw["variability"]["joint_omega"]
        iov, occasions = None, []
        shared, dose, period = raw["initialization"]["shared_last_dose_h"], 300, 12
        candidates = [{"state_id": s["state_event_id"], "events": [_event(e["time_h"], e["amount_mg"]) for e in
  s["dose_events"]]} for s in raw["candidate_states"]]
        observations = [{"record_id": str(i), "time": r["time_h"], "value": r["concentration_mg_L"], "time_unit": "h",
  "value_unit": "mg/L"} for i, r in enumerate(raw["observations"])]
        relations, predictor, error_type, role = _oxc_relations, LinearPredictor(_oxc_system), "additive", "sd"
    else:
        covariates = {k: raw["fixed"][k] for k in ("X", "HEM", "POD", "Pred")}
        covariates["WT"] = raw["parameters"]["WT"]
        values = {**{k: v for k, v in raw["fixed"].items() if k not in covariates},
                  **{k: v for k, v in raw["parameters"].items() if k != "WT"}, "sigma": raw["residual_sd"]}
        units = {k: "1" for k in values}
        units.update(CL="L/h", V1="L", V2="L", Q="L/h", ka="1/h", lag="h", normalization_weight="kg",
                     theta_POD="1/day", POD_cap="day", theta_Pred="mL/ng")
        iiv_names, iiv_matrix = ["CL", "V1", "V2", "ka"], np.array(raw["omega"])[:4, :4]
        shared, dose, period = raw["shared_last"], raw["shared_dose"], raw["tau"]
        boundaries = [shared, *raw["slots"], max(48, max(raw["times"])+1)]
        occasions = [{"occasion_id": f"occ_{i}", "start": a, "end": b} for i, (a, b) in enumerate(zip(boundaries[:-1],
  boundaries[1:]))]
        iov = {"names": ["CL", "V1"], "distribution": "normal", "occasion_key": "occasion_id",
               "covariance_parameters": _covariance_refs(np.array(raw["omega"])[4:6, 4:6], "pi", values, units)}
        candidates = [{"state_id": s["state_event_id"], "events": [_event(e["time_h"], e["amount_mg"]) for e in
  s["events"]]} for s in raw["states"]]
        observations = [{"record_id": str(i), "time": t, "value": y, "time_unit": "h", "value_unit": "ng/mL"} for i,
  (t, y) in enumerate(zip(raw["times"], raw["observations"]))]
        relations, predictor, error_type, role = _tac_relations, LinearPredictor(_tac_system), "proportional",
  "coefficient_sd"
    refs = _covariance_refs(iiv_matrix, "omega", values, units)
    model = {"parameters": {"values": values, "units": units}, "parameter_model": {"covariate_names": list(covariates),
  "relations": relations},
             "iiv": {"names": iiv_names, "distribution": "normal", "covariance_parameters": refs}, "iov": iov,
             "residual_model": {"distribution": "normal", "error_type": error_type, "parameter_names": {role:
  "sigma"}}, "predict": predictor}
    periodic = _event(shared, dose)
    periodic["offset"] = 0
    del periodic["time"]
    individual_data = {"covariates": covariates, "occasions": occasions,
        "dosing_prehistory": {"events": [_event(shared, dose)], "regimen": {"period": period, "anchor_time": shared,
  "time_unit": "h", "events": [periodic]}},
        "initialization": {"mode": "steady_state", "start_time": shared, "event_phase": "before_events",
                           "steady_state": {"parameter_rule": "fixed_reference", "reference_occasion_id":
  occasions[0]["occasion_id"] if occasions else None}}}
    return {"model": model, "individual_data": individual_data, "observations": observations, "candidate_states":
```

```
  candidates}


def evaluate_reference(case, candidate_state, random_effect_values):
    """Function: Evaluate the complete model layer once.
    Inputs: Reference inputs, one candidate and current effects.
    Outputs: Predictions and conditional log-likelihood, not posterior probabilities.
    """
    model, data = case["model"], case["individual_data"]
    distribution = resolve_random_effects(iiv=model["iiv"], iov=model["iov"], parameters=model["parameters"],
                                          occasions=data["occasions"], dependence="independent_blocks")
    request = prepare_prediction_records(case["observations"], data)
    individual_parameters = resolve_parameters(parameter_model=model["parameter_model"], parameters=model["parameters"],
        covariates=data["covariates"], **distribution.expand(random_effect_values), occasions=data["occasions"],
  evaluation_interval=request["evaluation_interval"])
    predictions = evaluate_prediction(predict=model["predict"], individual_parameters=individual_parameters,
  candidate_state=candidate_state,
        prediction_records=request["prediction_records"], individual_data=data)
    residual_distribution = resolve_residuals(residual_model=model["residual_model"], parameters=model["parameters"],
  predictions=predictions,
        prediction_records=request["prediction_records"], dependence="conditionally_independent")
    return predictions, conditional_log_likelihood(observations=case["observations"], predictions=predictions,
  residual_distribution=residual_distribution)


def main():
    """Function: Print conditional reference evaluations.
    Inputs: Local references at zero random effects.
    Outputs: JSON concentrations and conditional likelihoods, not posteriors.
    """
    output = {}
    for name, dimension in (("OXC", 1), ("TAC", 10)):
        case = load_reference(name)
        output[name] = []
        for state in case["candidate_states"]:
            predictions, likelihood = evaluate_reference(case, state, np.zeros(dimension))
            output[name].append({"state_id": state["state_id"], "predictions": predictions["values"].tolist(),
                                 "units": predictions["units"], "conditional_log_likelihood": likelihood})
    print(json.dumps(output, indent=2, allow_nan=False))


if __name__ == "__main__":
    main()
```

Complete C003 Input

c003.json

```
{
  "model": {
    "model_id": "Lin_OXC_MHD_Hill219",
    "compartments": [
      "depot",
      "central"
    ],
    "analyte": "MHD",
    "dose_unit": "mg",
    "concentration_unit": "mg/L",
    "parameters": {
      "CL_p": 1.68,
      "V_p": 14.7,
      "ka": 0.83,
      "BW": 25.0
    },
    "fixed": {
      "k0": 0.624,
      "kmax": 0.233,
      "k50": 8.97,
      "gamma": 219,
      "normalization_weight": 70
    },
    "relationships": {
      "CL": "CL_p*(BW/70)**(k0-kmax*BW**gamma/(k50**gamma+BW**gamma))*exp(eta_CL)",
      "V": "V_p",
      "ka": "ka",
      "concentration": "A_central/V_p"
    },
    "covariates_constant_over_record": true
  },
  "initialization": {
    "definition_status": "author_approved",
    "interval_h": 12,
    "shared_last_dose_h": -12,
    "initial_state_time_h": -12,
    "initial_state_phase": "immediately_before_shared_last_dose",
    "initial_dose_event": {
      "event_id": "shared_last",
      "nominal_slot_h": -12,
      "time_h": -12,
```

```
      "local_time_h": -12,
      "local_nominal_slot_h": -12,
      "amount_mg": 300.0,
      "route": "oral",
      "compartment": "depot"
    },
    "earlier_history": {
      "interval_h": 12,
      "dose_mg": 300.0,
      "last_implicit_dose_h": -24,
      "extends_to": "minus_infinity"
    },
    "explicit_occasion_slots_h": [],
    "no_0h_dose_in_one_dose_window": false,
    "numerical_initial_state": null,
    "time_axes": {
      "time_h_axis": "manuscript_study",
      "local_axis": "earliest_candidate_slot_is_zero",
      "manuscript_to_local_offset_h": 0,
      "manuscript_candidate_slots_h": [
        0,
        12
      ],
      "local_candidate_slots_h": [
        0,
        12
      ],
      "manuscript_shared_last_h": -12,
      "local_shared_last_h": -12,
      "manuscript_last_implicit_dose_h": -24,
      "local_last_implicit_dose_h": -24,
      "all_events_and_observations_shift_equally": true,
      "native_runtime_axis_adapter_verified": false
    },
    "method": "analytic_steady_state",
    "conditioning": "individual_eta_CL",
    "iov_present": false
  },
  "variability": {
    "iiv_order": [
      "Clearance IIV"
    ],
    "iiv_omega": [
      [
        0.0122457142458844
      ]
    ],
    "iiv_correlation": [
      [
        1
      ]
    ],
    "iov_order": [],
    "iov_omega_per_occasion": [],
    "iov_occasion_ids": [],
    "joint_order": [
      "Clearance IIV"
    ],
    "joint_omega": [
      [
        0.0122457142458844
      ]
    ],
    "source_reference_log_variances": {
      "Clearance IIV": 0.0122457142458844
    },
    "distribution": "zero_mean_multivariate_normal_log_effects"
  },
  "residual": {
    "model": "additive",
    "sd": 2.992,
    "variance": 8.952064,
    "additive_component_present": true,
    "residual_floor": null,
    "normal_zero_mean": true
  },
  "candidate_states": [
    {
      "manuscript_id": "omega_00",
      "machine_chronological_bits": [
        0,
        0
      ],
      "machine_bitstring": "00",
      "nominal_slots_h": [
        0,
        12
      ],
      "taken_by_slot": [
        {
          "nominal_time_h": 0,
```

```
        "taken": false
      },
      {
        "nominal_time_h": 12,
        "taken": false
      }
    ],
    "dose_events": [],
    "state_event_id": "state_944c7ce98f10ddebf79258da1fee55f539dacafb4c78ffb6746e1a1d71ee1681"
  },
  {
    "manuscript_id": "omega_01",
    "machine_chronological_bits": [
      1,
      0
    ],
    "machine_bitstring": "10",
    "nominal_slots_h": [
      0,
      12
    ],
    "taken_by_slot": [
      {
        "nominal_time_h": 0,
        "taken": true
      },
      {
        "nominal_time_h": 12,
        "taken": false
      }
    ],
    "dose_events": [
      {
        "event_id": "slot_0",
        "nominal_slot_h": 0,
        "time_h": 0,
        "local_time_h": 0,
        "local_nominal_slot_h": 0,
        "amount_mg": 300.0,
        "route": "oral",
        "compartment": "depot"
      }
    ],
    "state_event_id": "state_6cf511a7706e8318ba9c59ca6e9e74e347b31829a559efe8fa33b861b64644e9"
  },
  {
    "manuscript_id": "omega_10",
    "machine_chronological_bits": [
      0,
      1
    ],
    "machine_bitstring": "01",
    "nominal_slots_h": [
      0,
      12
    ],
    "taken_by_slot": [
      {
        "nominal_time_h": 0,
        "taken": false
      },
      {
        "nominal_time_h": 12,
        "taken": true
      }
    ],
    "dose_events": [
      {
        "event_id": "slot_12",
        "nominal_slot_h": 12,
        "time_h": 12,
        "local_time_h": 12,
        "local_nominal_slot_h": 12,
        "amount_mg": 300.0,
        "route": "oral",
        "compartment": "depot"
      }
    ],
    "state_event_id": "state_207324eb5bdd241659fde078e9c0b6feeeb160423e270da304daaa5b8390de69"
  },
  {
    "manuscript_id": "omega_11",
    "machine_chronological_bits": [
      1,
      1
    ],
    "machine_bitstring": "11",
    "nominal_slots_h": [
      0,
      12
    ],
```

```
      "taken_by_slot": [
        {
          "nominal_time_h": 0,
          "taken": true
        },
        {
          "nominal_time_h": 12,
          "taken": true
        }
      ],
      "dose_events": [
        {
          "event_id": "slot_0",
          "nominal_slot_h": 0,
          "time_h": 0,
          "local_time_h": 0,
          "local_nominal_slot_h": 0,
          "amount_mg": 300.0,
          "route": "oral",
          "compartment": "depot"
        },
        {
          "event_id": "slot_12",
          "nominal_slot_h": 12,
          "time_h": 12,
          "local_time_h": 12,
          "local_nominal_slot_h": 12,
          "amount_mg": 300.0,
          "route": "oral",
          "compartment": "depot"
        }
      ],
      "state_event_id": "state_50241cf258a49601695e85381a33eb515611e03f51a0bad796d3089b8665d42f"
    }
  ],
  "priors": [
    {
      "state_event_id": "state_944c7ce98f10ddebf79258da1fee55f539dacafb4c78ffb6746e1a1d71ee1681",
      "manuscript_id": "omega_00",
      "probability": 0.25
    },
    {
      "state_event_id": "state_6cf511a7706e8318ba9c59ca6e9e74e347b31829a559efe8fa33b861b64644e9",
      "manuscript_id": "omega_01",
      "probability": 0.25
    },
    {
      "state_event_id": "state_207324eb5bdd241659fde078e9c0b6feeeb160423e270da304daaa5b8390de69",
      "manuscript_id": "omega_10",
      "probability": 0.25
    },
    {
      "state_event_id": "state_50241cf258a49601695e85381a33eb515611e03f51a0bad796d3089b8665d42f",
      "manuscript_id": "omega_11",
      "probability": 0.25
    }
  ],
  "schema": "oxc_numeric_source_ruv_v1",
  "observations": [
    {
      "time_h": 24.0,
      "concentration_mg_L": 12.0
    }
  ]
}
```


Output: `result.json` contains the posterior vector and per-candidate diagnostics; `candidate_checkpoint.json` records intermediate candidate results. C003 result.

TAC c003

Five probability vectors were returned for `c003`. **All five remain qualified because the prescribed log-integral and posterior order-stability checks were not met.** The recorded execution status `success` and acceptance mode `external_comparison` do not replace those check outcomes.

The recorded integration used sparse nHe orders 4, 5 and 6 in prior-standardized coordinates. Both stability tolerances were `1e-6`; the criteria required the last two changes to meet the tolerance for every candidate log integral and for the maximum statewise posterior change, respectively. The TAC interface records an eight-dimensional `exact_gaussian_projection` from the ten source random-effect coordinates; the adapter and shared listings are identified below.

The recorded environment was R 4.4.3 (2025-02-28 ucrt), `aghq` 0.4.1, `reticulate` 1.46.0, `numDeriv` 2016.8.1.1, `mvQuad` 1.0.10 and `jsonlite` 2.0.0, with Python 3.12.0, `pmx-discrete-posterior` 0.2.0, NumPy 2.5.2 and SciPy 1.16.2. The R session record identifies Windows 11 x64 build 26200. Complete session and source-origin records are included in the manifest.

Recorded Invocation and Settings

```
& "C:\Program Files\R\R-4.4.3\bin\Rscript.exe" --vanilla "D:\AI-ready PMX benchmark
  engine\rerun_c026_full_20261021\04_r_aghq_tac\r_external\run_case.R" "D:\AI-ready PMX benchmark
  engine\rerun_c026_full_20261021\04_r_aghq_tac\inputs\TAC\c003.json" TAC "D:\AI-ready PMX benchmark
  engine\rerun_c026_full_20261021\04_r_aghq_tac\settings\TAC.json" "D:\AI-ready PMX benchmark
  engine\rerun_c026_full_20261021\04_r_aghq_tac\runs\formal_retry_20261022\TAC\c003\r01"
```

Original execution record.

settings/TAC.json

Original source | Reader listing

```
{
  "acceptance_mode": "external_comparison",
  "orders": [
    4,
    5,
    6
  ],
  "log_integral_atol": 1e-06,
  "posterior_atol": 1e-06,
  "max_nodes": 50000,
  "max_evaluations": 70000,
  "curvature_atol": 0.001,
  "stationarity_atol": 0.001,
  "construction": "sparse",
  "rule": "nHe",
  "derivative_policy": "floored_richardson",
  "optimizer": "R_BFGS_scaled",
  "coordinate_system": "prior_standardized"
}
```

Shared Driver and Interface: Runs 1-2

Complete driver and interface: OXC shared listings in Section 4.3, including the TAC Gaussian-projection code in `r_external/external_scenario.py`.

Source identities: match the collected files used by TAC runs 1-2.

Required dependencies: shared model adapters in Section 4.3 and package and runtime dependencies in the manifest.

Required file	Complete shared listing	Byte-identical original
`r_external/run_case.R`	OXC shared listing	Original
`bridge/scenario.R`	OXC shared listing	Original
`bridge/integration.R`	OXC shared listing	Original
`bridge/coordinates.R`	OXC shared listing	Original
`bridge/runtime.R`	OXC shared listing	Original
`r_external/external_scenario.py`	OXC shared listing	Original
`bridge/model_bridge.py`	OXC shared listing	Original
`bridge/target_coordinates.py`	OXC shared listing	Original

Source Variant: Runs 3-5

Runs 3-5 record the same distribution version, 0.2.0, but a different Python source tree from runs 1-2. Their full driver and changed interface/mapper files follow. The source seals, tree hashes and package-module identities are separately listed in the manifest.

Runs 3-5: r_external/run_case.R

Original source | Reader listing

```
# Function: Run one explicitly configured external case
# Inputs: raw JSON, OXC/TAC, settings JSON, output directory
# Outputs: result.json and actual_environment.json

# Function: Resolve an explicit environment/config value
# Inputs: Environment name, config field, optional default
# Outputs: One nonempty path; conflicting values fail
runtime_value <- function(name, configured = NULL, default = NULL) {
```

```
  value <- Sys.getenv(name, unset = "")
  if (!is.null(configured) && (length(configured) != 1L || !is.character(configured) || !nzchar(configured)))
    stop(paste("invalid runtime field for", name))
  if (nzchar(value) && !is.null(configured) &&
      !identical(normalizePath(value, winslash = "/", mustWork = TRUE),
                 normalizePath(configured, winslash = "/", mustWork = TRUE)))
    stop(paste("conflicting environment and config paths:", name))
  if (!nzchar(value)) value <- if (!is.null(configured)) configured else default
  if (is.null(value) || length(value) != 1L || !nzchar(value)) stop(paste("explicit path required:", name))
  normalizePath(value, winslash = "/", mustWork = TRUE)
}

# Function: Verify and bind the installed runtime once
# Inputs: Explicit root, Python, R library
# Outputs: Actual paths and exact pinned versions
initialize_external_runtime <- function(root, python, r_library) {
  locale <- if (.Platform$OS.type == "windows") "English_United States.utf8" else "C.UTF-8"
  if (!nzchar(Sys.setlocale("LC_CTYPE", locale)) || !isTRUE(l10n_info()[["UTF-8"]]))
    stop("a UTF-8 LC_CTYPE locale is required to source the archived bilingual bridge")
  expected <- normalizePath("D:/AI-ready PMX benchmark
  engine/rerun_c026_full_20261021/runtime/.venv/Scripts/python.exe", winslash = "/", mustWork = TRUE)
  if (!identical(python, expected)) stop("Python must be the installed .venv belonging to the explicit root")
  .libPaths(c(r_library, .libPaths()))
  Sys.setenv(RETICULATE_PYTHON = python, PYTHONDONTWRITEBYTECODE = "1",
             OMP_NUM_THREADS = "1", OPENBLAS_NUM_THREADS = "1", MKL_NUM_THREADS = "1")
  pinned <- c(aghq = "0.4.1", reticulate = "1.46.0", numDeriv = "2016.8.1.1",
              mvQuad = "1.0.10", jsonlite = "2.0.0")
  versions <- vapply(names(pinned), function(name) {
    # Function: Check namespace version and origin
    # Inputs: Package name Outputs: Exact version or failure
    if (!requireNamespace(name, quietly = TRUE)) stop(paste("missing R dependency:", name))
    origin <- normalizePath(getNamespaceInfo(asNamespace(name), "path"), winslash = "/", mustWork = TRUE)
    if (!startsWith(tolower(origin), paste0(tolower(r_library), "/")))
      stop(paste("R dependency loaded outside explicit library:", name))
    as.character(utils::packageVersion(name))
  }, character(1))
  if (!identical(versions, pinned)) stop("R dependency versions differ from the audited harness")
  data.table::setDTthreads(1L)
  reticulate::use_python(python, required = TRUE)
  python_sys <- reticulate::import("sys", convert = FALSE)
  python_sys$path$insert(0L, root)
  python_sys$path$insert(0L, file.path(root, "bridge"))
  # Pin pure sources before any pmx import; keep the interpreter read-only.
  # Bind the source snapshot before importing pmx; reuse the interpreter read-only.
  expected_source <- normalizePath(file.path(dirname(root), "runtime/src"), winslash = "/", mustWork = TRUE)
  source_root <- runtime_value("PMX_SOURCE_ROOT", default = expected_source)
  if (!identical(source_root, expected_source)) stop("PMX_SOURCE_ROOT must name the repaired snapshot")
  python_sys$path$insert(0L, source_root)
  adapter <- reticulate::import_from_path("external_scenario", path = file.path(root, "r_external"), convert = TRUE)
  provenance <- adapter$verify_runtime(python)
  list(adapter = adapter, environment = list(root = root, R = R.version.string, LC_CTYPE = Sys.getlocale("LC_CTYPE"),
       R_versions = as.list(versions), R_session_info = capture.output(utils::sessionInfo()),
       r_library = r_library, Python = provenance))
}

# Function: Enforce preregistered external-comparison stages
# Inputs: Model and integration settings
# Outputs: Validated settings; no best-order selection
external_settings <- function(model_name, settings) {
  settings <- validate_settings(settings)
  orders <- if (model_name == "OXC") c(7L, 15L, 31L, 61L) else c(4L, 5L, 6L)
  construction <- if (model_name == "OXC") "product" else "sparse"
  rule <- if (model_name == "OXC") "GHe" else "nHe"
  if (!identical(settings$orders, orders) || settings$construction != construction ||
      settings$rule != rule || settings$acceptance_mode != "external_comparison")
    stop("settings must use the prespecified model stages/rule and external_comparison")
  settings
}

# Function: Execute one case using only command arguments and explicit runtime
# Inputs: Four command arguments
# Outputs: Final result.json, including failures after runtime initialization
main <- function(args = commandArgs(trailingOnly = TRUE)) {
  if (length(args) != 4L) stop("usage: run_case.R RAW_JSON OXC|TAC SETTINGS_JSON OUTPUT_DIR")
  if (!args[[2]] %in% c("OXC", "TAC")) stop("model must be OXC or TAC")
  # jsonlite must be located before parsing runtime config; bootstrap from explicit R library.
  # Locate jsonlite using the explicit R library or R_LIBS_USER before reading settings.
  library_env <- Sys.getenv("PMX_R_LIBRARY", unset = Sys.getenv("R_LIBS_USER", unset = ""))
  if (nzchar(library_env)) .libPaths(c(library_env, .libPaths()))
  if (!requireNamespace("jsonlite", quietly = TRUE)) stop("set PMX_R_LIBRARY or R_LIBS_USER to bootstrap jsonlite")
  config <- jsonlite::read_json(args[[3]], simplifyVector = FALSE)
  runtime_config <- if (is.null(config$runtime)) list() else config$runtime
  root <- runtime_value("PMX_RERUN_ROOT", runtime_config$root)
  if (!identical(root, normalizePath(getwd(), winslash = "/", mustWork = TRUE)))
    stop("process cwd must equal the explicitly configured rerun root")
  python <- runtime_value("RETICULATE_PYTHON", runtime_config$python,
                          file.path(root, ".venv/Scripts/python.exe"))
  r_library <- runtime_value("PMX_R_LIBRARY", runtime_config$r_library,
                             if (nzchar(library_env)) library_env else NULL)
  raw_path <- normalizePath(args[[1]], winslash = "/", mustWork = TRUE)
  output_parent <- normalizePath(dirname(args[[4]]), winslash = "/", mustWork = TRUE)
```

```
  if (!startsWith(tolower(paste0(output_parent, "/")), paste0(tolower(root), "/")) ||
      basename(args[[4]]) %in% c(".", ".."))
    stop("output parent must already exist under the explicit rerun root")
  dir.create(args[[4]], showWarnings = FALSE)
  output <- normalizePath(args[[4]], winslash = "/", mustWork = TRUE)
  if (!startsWith(tolower(paste0(output, "/")), paste0(tolower(root), "/")))
    stop("output directory must remain under the explicit rerun root")
  if (file.exists(file.path(output, "result.json"))) stop("result.json already exists; use a fresh output directory")
  Sys.setenv(VALIDATION_RUN_DIR = output)
  started <- proc.time()[["elapsed"]]
  environment <- NULL
  metadata <- NULL
  settings <- NULL
  result <- tryCatch({
    bound <- initialize_external_runtime(root, python, r_library)
    environment <- bound$environment
    jsonlite::write_json(environment, file.path(output, "actual_environment.json"),
                         auto_unbox = TRUE, pretty = TRUE, null = "null", digits = NA)
    withCallingHandlers(source(file.path(root, "bridge/scenario.R"), encoding = "UTF-8"),
      warning = function(warning) {
        # Function: Reject incomplete source loading
        # Inputs: Source warning Outputs: Explicit failure
        stop(paste("archived bridge source warning:", conditionMessage(warning)))
      })
    settings <- config$integration
    if (is.null(settings)) settings <- config[setdiff(names(config), "runtime")]
    settings <- external_settings(args[[2]], settings)
    scenario <- bound$adapter$make_scenario(raw_path, args[[2]])
    metadata <- scenario$metadata()
    run_scenario(scenario, settings)
  }, error = function(error) {
    # Function: Preserve an explicit failure without substitute posterior
    # Inputs: Original exception Outputs: Failure record
    list(status = "failed", backend = "R aghq", posterior = NULL, state_results = list(),
         failure_reason = conditionMessage(error), absolute_error_certified = FALSE)
  })
  result$input <- metadata
  result$environment <- environment
  result$settings <- settings
  result$source_path <- raw_path
  result$model_name <- args[[2]]
  result$elapsed_seconds <- proc.time()[["elapsed"]] - started
  jsonlite::write_json(result, file.path(output, "result.json"), auto_unbox = TRUE,
                       pretty = TRUE, null = "null", digits = NA)
  if (!identical(result$status, "success")) stop(paste("external case failed:", result$failure_reason))
  invisible(result)
}

if (sys.nframe() == 0L) main()
```

Runs 3-5: r_external/external_scenario.py

Original source | Reader listing

```
"""Function: Bind raw reruns to the archived R bridge
Inputs: Raw JSON, shared mapper and installed 0.2.0
Outputs: Exact targets and source metadata, no quadrature
"""

import hashlib
import importlib
import importlib.metadata
import inspect
import json
import math
from functools import lru_cache
from pathlib import Path
import sys
import warnings

import numpy as np
from scipy.stats import multivariate_normal

import pmx_discrete_posterior
from pmx_discrete_posterior.interfaces.inputs import validate_inputs
from pmx_discrete_posterior.models.prediction import prepare_prediction_records
from pmx_discrete_posterior.models.prediction.linear import LinearPredictor
from pmx_discrete_posterior.models.random_effects import resolve_random_effects
from pmx_discrete_posterior.posterior import computation

from model_bridge import ModelScenario, ModelTarget


PACKAGE_TREE_SHA256 = "2e553f124d5d27e1b95be3f4dd2d28c145f408f22c59fc7ac23f2c5f49e52f8e"
BINDING_SHA256 = "dd42654fbae69118d52e8db04f5789110865c9ca86a68d80207b51d38c8713bd"
REFERENCE_SHA256 = "d4c07d73fc4521da244d6ed715fcda7e6cc03f3172488d931f39a09857c3e9cc"
PINNED_VERSIONS = {"pmx-discrete-posterior": "0.2.0", "numpy": "2.5.2", "scipy": "1.16.2"}
BUILDER_SHA256 = "ede0f03ebb0b8c590a548405362524207b699b360ffa4a1d18dcdde24118fa33"
ROOT = Path(__file__).resolve().parents[1]
```

```
SOURCE_ROOT = ROOT.parent / "runtime/src"
FROZEN_PYTHON = ROOT.parents[1] / "rerun_c026_full_20261021/runtime/.venv/Scripts/python.exe"


def file_sha256(path):
    """Function: Hash source bytes
    Inputs: File path Outputs: SHA256 hex digest
    """
    return hashlib.sha256(Path(path).read_bytes()).hexdigest()


@lru_cache(maxsize=1)
def verify_sources():
    """Function: Check the pinned implementation once
    Inputs: Repaired snapshot and local bridge modules
    Outputs: Versions and source hashes, or explicit failure
    """
    reference = importlib.import_module("shared.reference_models")
    versions = {name: importlib.metadata.version(name) for name in PINNED_VERSIONS}
    if versions != PINNED_VERSIONS:
        raise RuntimeError(f"pinned Python dependency versions changed: {versions}")
    package = Path(pmx_discrete_posterior.__file__).resolve().parent
    if Path(pmx_discrete_posterior.__file__).resolve() != (
            SOURCE_ROOT / "pmx_discrete_posterior/__init__.py").resolve(strict=True):
        raise RuntimeError("package must come from the explicitly pinned repaired source snapshot")
    for name, module in tuple(sys.modules.items()):
        if name == "pmx_discrete_posterior" or name.startswith("pmx_discrete_posterior."):
            origin = getattr(module, "__file__", None)
            if not origin or not Path(origin).resolve().is_relative_to(package):
                raise RuntimeError(f"mixed package import origin: {name}")
    files = sorted(package.rglob("*.py"), key=lambda path: path.relative_to(package).as_posix())
    manifest = "".join(f"{path.relative_to(package).as_posix()}\0{file_sha256(path)}\n" for path in files)
    tree_hash = hashlib.sha256(manifest.encode("utf-8")).hexdigest()
    if tree_hash != PACKAGE_TREE_SHA256 or file_sha256(computation.__file__) != BINDING_SHA256:
        raise RuntimeError("repaired snapshot differs from the audited 0.2.0 Python source tree")
    if file_sha256(reference.__file__) != REFERENCE_SHA256:
        raise RuntimeError("shared/reference_models.py differs from the archived audited factory")
    builder = importlib.import_module("shared.scenario_inputs")
    for module, filename in ((reference, "reference_models.py"), (builder, "scenario_inputs.py")):
        if Path(module.__file__).resolve() != (ROOT / "shared" / filename).resolve():
            raise RuntimeError("shared module must come from this isolated bridge")
    if file_sha256(builder.__file__) != BUILDER_SHA256:
        raise RuntimeError("shared/scenario_inputs.py differs from the verified pre-event mapper")
    return {"versions": versions, "package_origin": str(package), "package_source_sha256": tree_hash,
            "binding_sha256": BINDING_SHA256, "reference_source_sha256": REFERENCE_SHA256,
            "reference_origin": str(Path(reference.__file__).resolve()),
            "package_file": str(Path(pmx_discrete_posterior.__file__).resolve()), "source_root":
  str(SOURCE_ROOT.resolve()),
            "builder_origin": str(Path(builder.__file__).resolve()), "builder_source_sha256": BUILDER_SHA256,
            "source_mode": "repaired_source_snapshot_on_frozen_interpreter"}


def verify_runtime(expected_python):
    """Function: Verify frozen interpreter plus repaired sources
    Inputs: Absolute Python executable
    Outputs: Actual executable, versions and provenance
    """
    expected = Path(expected_python).resolve(strict=True)
    if expected != FROZEN_PYTHON.resolve(strict=True) or Path(sys.executable).resolve() != expected:
        raise RuntimeError("reticulate is not using the explicitly selected Python executable")
    prefix = expected.parent.parent.resolve()
    if Path(sys.prefix).resolve() != prefix:
        raise RuntimeError("Python prefix differs from the read-only frozen runtime")
    for name in ("numpy", "scipy"):
        module = importlib.import_module(name)
        distribution = importlib.metadata.distribution(name)
        if (not Path(module.__file__).resolve().is_relative_to(prefix) or
                not Path(distribution.locate_file("")).resolve().is_relative_to(prefix)):
            raise RuntimeError(f"numerical dependency outside frozen runtime: {name}")
    return {**verify_sources(), "python": sys.version, "executable": str(expected),
            "runtime_prefix": str(prefix), "distribution_metadata_origin": str(
                importlib.metadata.distribution("pmx-discrete-posterior").locate_file(""))}


def effect_names(keys):
    """Function: Label coordinates without losing occasions
    Inputs: Ordered effect keys Outputs: Ordered names
    """
    return [".".join(str(part) for part in key if part is not None) for key in keys]


class NamedModelTarget(ModelTarget):
    """Function: Add names to the unchanged archived target
    Inputs: Bound target inputs Outputs: Named target
    """

    def metadata(self):
        """Function: Export full-coordinate metadata
        Inputs: None Outputs: Dimension, names and covariance
        """
        keys = self.distribution.effect_keys
```

```
        return {**super().metadata(), "representation": "full", "effect_names": effect_names(keys),
                "effect_keys": [list(key) for key in keys],
                "covariance": self.distribution.covariance.tolist()}


def prove_tac_sum_structure(target, model, data):
    """Function: Restrict projection to the audited CL/V1 sums
    Inputs: Bound target, validated model and data
    Outputs: Ordered occasion IDs or failure; not a numerical proof
    """
    verify_sources()
    reference = importlib.import_module("shared.reference_models")
    if model["parameter_model"]["relations"] is not reference._tac_relations:
        raise ValueError("exact TAC projection requires the audited shared TAC relations")
    predictor = model["predict"]
    if type(predictor) is not LinearPredictor or predictor._system is not reference._tac_system:
        raise ValueError("exact TAC projection requires the audited TAC LinearPredictor")
    likelihood_code = next(code for code in computation._bind_likelihood.__code__.co_consts
                           if inspect.iscode(code) and code.co_name == "likelihood")
    if not inspect.isfunction(target.likelihood) or target.likelihood.__code__ is not likelihood_code:
        raise ValueError("exact TAC projection requires the pinned package likelihood binding")
    closure = inspect.getclosurevars(target.likelihood).nonlocals
    if (closure.get("model") is not model or closure.get("individual_data") is not data or
            closure.get("random_effect_distribution") is not target.distribution):
        raise ValueError("projection proof and bound likelihood do not use the same inputs")
    occasions = [occasion["occasion_id"] for occasion in data["occasions"]]
    if not occasions or len(set(occasions)) != len(occasions):
        raise ValueError("exact TAC projection requires distinct nonempty occasions")
    expected = [("iiv", None, name) for name in ("CL", "V1", "V2", "ka")]
    expected += [("iov", occasion, name) for occasion in occasions for name in ("CL", "V1")]
    if target.distribution.effect_keys != tuple(expected):
        raise ValueError("unsupported TAC active effects; expected all CL/V1 IIV and IOV plus V2/ka IIV")
    return occasions


class TACGaussianProjection:
    """Function: Push the Gaussian through exact dynamic TAC sums
    Inputs: Audited full bound target and validated inputs
    Outputs: Equivalent normalized (2*n+2)D target
    """

    def __init__(self, target, model, data):
        """Function: Build A, a right inverse and A Omega A^T
        Inputs: Target, model and data
        Outputs: Proven exact projection; no jitter or fallback
        """
        self.occasions = prove_tac_sum_structure(target, model, data)
        self.original = target
        count = len(self.occasions)
        self.dimension = 2 * count + 2
        self.projection = np.zeros((self.dimension, 4 + 2 * count))
        self.right_inverse = np.zeros((4 + 2 * count, self.dimension))
        self.effect_names = []
        for occasion_index, occasion in enumerate(self.occasions):
            for parameter_index, parameter in enumerate(("CL", "V1")):
                row = 2 * occasion_index + parameter_index
                self.projection[row, parameter_index] = 1
                self.projection[row, 4 + row] = 1
                self.right_inverse[4 + row, row] = 1
                self.effect_names.append(f"iiv.{parameter}+iov.{occasion}.{parameter}")
        for row, column, name in ((2 * count, 2, "V2"), (2 * count + 1, 3, "ka")):
            self.projection[row, column] = self.right_inverse[column, row] = 1
            self.effect_names.append(f"iiv.{name}")
        if not np.array_equal(self.projection @ self.right_inverse, np.eye(self.dimension)):
            raise ValueError("invalid exact Gaussian projection right inverse")
        self.mean = self.projection @ target.distribution.mean
        self.covariance = self.projection @ target.distribution.covariance @ self.projection.T
        self.density = multivariate_normal(mean=self.mean, cov=self.covariance, allow_singular=False)

    def metadata(self):
        """Function: Export exact representation and proof boundaries
        Inputs: None Outputs: Maps, names, covariance and source proof
        """
        return {"dimension": self.dimension, "original_dimension": len(self.original.distribution.mean),
                "representation": "exact_gaussian_projection", "mean": self.mean.tolist(),
                "effect_names": self.effect_names, "original_effect_names":
  effect_names(self.original.distribution.effect_keys),
                "original_effect_keys": [list(key) for key in self.original.distribution.effect_keys],
                "occasion_ids": self.occasions, "projection": self.projection.tolist(),
                "right_inverse": self.right_inverse.tolist(), "covariance": self.covariance.tolist(),
                "proof": {"kind": "audited_source_identity_and_linear_gaussian_pushforward",
                          "reference_source_sha256": REFERENCE_SHA256, "binding_sha256": BINDING_SHA256,
                          "likelihood_dependence": "CL/V1 IIV+IOV per occasion; V2/ka IIV",
                          "integrated_null_dimension": 2, "approximation": False}}

    def log_joint(self, coordinates):
        """Function: Evaluate the exact normalized reduced target
        Inputs: Finite effective coordinates
        Outputs: Log likelihood plus pushed Gaussian log density
        """
        point = np.asarray(coordinates, dtype=float)
```

```
        if point.shape != (self.dimension,) or not np.all(np.isfinite(point)):
            raise ValueError(f"projection expects {self.dimension} finite coordinates")
        return math.fsum((self.original.likelihood(self.right_inverse @ point),
                          float(self.density.logpdf(point))))


class ExternalScenario(ModelScenario):
    """Function: Bind shared raw inputs and inherit archived finalize
    Inputs: Raw JSON path and OXC/TAC
    Outputs: Source-prior scenario; never calls the old constructor
    """

    def __init__(self, raw_json_path, model_name):
        """Function: Validate source priors and bind package likelihoods
        Inputs: Raw path and exact model name
        Outputs: Full OXC or exactly projected TAC targets
        """
        if model_name not in ("OXC", "TAC"):
            raise ValueError("model_name must be OXC or TAC")
        self.source = Path(raw_json_path).resolve(strict=True)
        payload = self.source.read_bytes()
        self.source_sha256 = hashlib.sha256(payload).hexdigest()
        self.raw = json.loads(payload.decode("utf-8-sig"))
        if not isinstance(self.raw, dict):
            raise ValueError("raw JSON must contain an object")
        self.provenance = verify_sources()
        builder = importlib.import_module("shared.scenario_inputs")
        with warnings.catch_warnings(record=True) as recorded:
            warnings.simplefilter("always")
            inputs = builder.build_inputs(self.raw, model_name)
        self.input_warnings = [{"category": warning.category.__name__, "message": str(warning.message)}
                               for warning in recorded]
        for warning in recorded:
            warnings.warn(warning.message, warning.category, stacklevel=2)
        if isinstance(inputs, dict):
            case = {key: inputs[key] for key in
                    ("model", "individual_data", "observations", "candidate_states", "priors")}
        else:
            model, data, observations, states, priors = inputs
            case = dict(model=model, individual_data=data, observations=observations,
                        candidate_states=states, priors=priors)
        # "laplace" selects shared input validation only; no optimizer/integrator is called.
        # This identifier selects input validation only, not Laplace optimization or integration.
        checked = validate_inputs(**case, integration_method="laplace", numerical_options=None)
        self.model_name = model_name
        self.case_id = self.raw.get("case_id", self.raw.get("numeric_id", self.source.stem))
        self.case = {key: checked[key] for key in ("model", "individual_data", "observations", "candidate_states")}
        self.states, self.priors = checked["candidate_states"], checked["priors"]
        model, data = self.case["model"], self.case["individual_data"]
        self.effects = resolve_random_effects(iiv=model["iiv"], iov=model["iov"], parameters=model["parameters"],
                                              occasions=data["occasions"], dependence=model["random_effects_dependence"])
        request = prepare_prediction_records(checked["observations"], data)
        self.targets = {state["state_id"]: NamedModelTarget(
            computation._bind_likelihood(model=model, individual_data=data, observations=checked["observations"],
                                         candidate_state=state, random_effect_distribution=self.effects,
                                         prediction_request=request), self.effects) for state in self.states}
        self.representation = "full" if model_name == "OXC" else "exact_gaussian_projection"
        if model_name == "OXC":
            if self.effects.effect_keys != (("iiv", None, "CL"),):
                raise ValueError("external OXC route requires the full one-dimensional CL effect")
        else:
            self.targets = {key: TACGaussianProjection(target, model, data) for key, target in self.targets.items()}
        self.builder_sha256 = file_sha256(builder.__file__)

    def metadata(self):
        """Function: Preserve input provenance, source priors and coordinates
        Inputs: None Outputs: Complete JSON-compatible scenario metadata
        """
        target = next(iter(self.targets.values())).metadata()
        model, data = self.case["model"], self.case["individual_data"]
        return {**super().metadata(), "effect_names": target["effect_names"],
                "full_effect_names": effect_names(self.effects.effect_keys),
                "full_effect_keys": [list(key) for key in self.effects.effect_keys],
                "full_mean": self.effects.mean.tolist(), "full_covariance": self.effects.covariance.tolist(),
                "target_coordinates": target, "source_path": str(self.source), "source_sha256": self.source_sha256,
                "source_schema": self.raw.get("schema"), "source_numeric_id": self.raw.get("numeric_id"),
                "source_covariance": (self.raw["variability"]["joint_omega"] if self.model_name == "OXC"
                                      else self.raw["omega"]),
                "input_warnings": self.input_warnings,
                "prior_source": "raw.priors" if self.model_name == "OXC" else "raw.states[].prior",
                "builder_source_sha256": self.builder_sha256, "provenance": self.provenance,
                "parameters": model["parameters"], "covariates": data["covariates"],
                "occasions": data["occasions"], "initialization": data["initialization"],
                "dosing_prehistory": data["dosing_prehistory"], "residual_model": model["residual_model"],
                "observations": self.case["observations"], "candidate_states": self.states,
                "random_effects_dependence": model["random_effects_dependence"]}


def make_scenario(raw_json_path, model_name):
    """Function: Stable reticulate constructor
    Inputs: Raw JSON path, OXC/TAC
```

```
    Outputs: Validated external scenario
    """
    return ExternalScenario(raw_json_path, model_name)
```

Runs 3-5: shared/scenario_inputs.py

Original source | Reader listing

```
"""Function: Experiment-only raw OXC/TAC adapter.
Inputs: Source JSON and model name.
Outputs: Five scientific keyword arguments.

No integration, file access, reference-case substitution, unit conversion, prior
normalization, covariance rank repair, or production-code changes occur here. Source
IDs and display labels are provenance, not instructions for constructing doses.
TAC v1 defines oral mg doses, h clocks, ng/mL observations, proportional Gaussian
residuals, and the reference callback's conditional steady-state convention.
This study reads concentrations before same-time parameter switches and inputs;
the mapper explicitly carries that convention in each TAC observation record.
Archived TAC covariance construction can differ across triangles by 1-2 ULP.
Only nonzero, same-sign pairs within two ULP are canonicalized to the source
upper triangle, with a CovarianceRoundoffWarning recording every changed pair.
Callers should capture these warnings in their experiment audit logs. All other
asymmetry, cross-block dependence and unequal IOV blocks are rejected.
"""

import math
import warnings

import numpy as np

from pmx_discrete_posterior.interfaces.inputs.candidate_states import (
    validate_candidate_history,
    validate_candidate_states,
)
from pmx_discrete_posterior.interfaces.inputs.individual_data import validate_individual_data
from pmx_discrete_posterior.interfaces.inputs.model_inputs import validate_model
from pmx_discrete_posterior.interfaces.inputs.observations import validate_observations
from pmx_discrete_posterior.interfaces.inputs.priors import validate_priors
from pmx_discrete_posterior.interfaces.validation import (
    InputError, fields, identifier, numeric_array, scalar, validate_covariance,
)
from pmx_discrete_posterior.models.prediction.linear import LinearPredictor
from pmx_discrete_posterior.models.random_effects import resolve_random_effects

from .reference_models import (
    _covariance_refs, _event, _oxc_relations, _oxc_system, _tac_relations, _tac_system,
)


_OXC_IIV = ["Clearance IIV"]
_TAC_IIV = ["Clearance IIV", "Central-volume IIV", "Peripheral-volume IIV", "Absorption-rate IIV"]
_OXC_RELATIONSHIPS = {
    "CL": "CL_p*(BW/70)**(k0-kmax*BW**gamma/(k50**gamma+BW**gamma))*exp(eta_CL)",
    "V": "V_p", "ka": "ka", "concentration": "A_central/V_p",
}


class CovarianceRoundoffWarning(UserWarning):
    """Function: Report archive covariance roundoff canonicalization.
    Inputs: Coordinates and original triangle values.
    Outputs: Capturable warning; source object unchanged.
    """


def _keys(value, required, path):
    """Function: Reject unknown or missing fields.
    Inputs: Mapping, key string, path.
    Outputs: Checked mapping or InputError.
    """
    return fields(value, required.split(), name=path)


def _equal(actual, expected, path):
    """Function: Check fixed semantics and duplicate declarations.
    Inputs: Actual value, expected value, path.
    Outputs: None; mismatch raises InputError.
    """
    if (isinstance(expected, bool) and type(actual) is not bool) or actual != expected:
        raise InputError(f"{path}: unsupported or inconsistent value; expected {expected!r}")


def _sequence(value, path, *, nonempty=False):
    """Function: Validate a raw sequence.
    Inputs: Value, path, nonempty constraint.
    Outputs: Original sequence or InputError.
    """
    if not isinstance(value, (list, tuple)) or (nonempty and not value):
        raise InputError(f"{path} must be a {'nonempty ' if nonempty else ''}sequence")
```

```
    return value


def _number_map(raw, keys, path, *, positive=(), nonnegative=()):
    """Function: Validate and retain source parameters.
    Inputs: Mapping, keys and bounds.
    Outputs: Finite numeric mapping.
    """
    _keys(raw, keys, path)
    return {k: scalar(v, name=f"{path}.{k}", positive=k in positive,
                      minimum=0 if k in nonnegative else None) for k, v in raw.items()}


def _matrix(raw, dimension, path):
    """Function: Validate the complete covariance.
    Inputs: Matrix, dimension, path.
    Outputs: Covariance; <=2 ULP canonicalization warns.
    """
    matrix = numeric_array(raw, ndim=2, name=path)
    if matrix.shape != (dimension, dimension):
        raise InputError(f"{path} must have shape ({dimension}, {dimension})")
    changes = []
    for i in range(dimension):
        for j in range(i+1, dimension):
            upper, lower = float(matrix[i, j]), float(matrix[j, i])
            if upper == lower:
                continue
            if (upper == 0 or lower == 0 or (upper > 0) != (lower > 0)
                    or abs(upper-lower) > 2*max(math.ulp(upper), math.ulp(lower))):
                raise InputError(f"{path} is asymmetric at ({i}, {j}); outside two-ULP roundoff policy")
            changes.append(f"({i},{j}): upper={upper!r}, lower={lower!r}")
            matrix[j, i] = upper
    if changes:
        warnings.warn(f"{path}: canonicalized lower to source upper triangle (<=2 ULP); " + "; ".join(changes),
                      CovarianceRoundoffWarning, stacklevel=3)
    validate_covariance(matrix)
    return matrix


def _slots(raw, shared):
    """Function: Validate nominal window boundaries.
    Inputs: Slots and shared last-dose time.
    Outputs: Strictly increasing slots after shared history.
    """
    slots = [scalar(t, name="slot") for t in _sequence(raw, "slots", nonempty=True)]
    if any(b <= a for a, b in zip([shared, *slots], slots)):
        raise InputError("slots must be strictly increasing after shared_last")
    return slots


def _raw_event(raw, *, oxc, offset=0):
    """Function: Map actual dose time and amount.
    Inputs: Source event, model flag, local offset.
    Outputs: Public event and nominal slot.
    """
    keys = "time_h amount_mg nominal_slot_h"
    if oxc:
        keys += " event_id local_time_h local_nominal_slot_h route compartment"
    _keys(raw, keys, "dose event")
    time = scalar(raw["time_h"], name="time_h")
    nominal = scalar(raw["nominal_slot_h"], name="nominal_slot_h")
    amount = scalar(raw["amount_mg"], name="amount_mg", minimum=0)
    if oxc:
        identifier(raw["event_id"], name="event_id")
        _equal(raw["route"], "oral", "event.route")
        _equal(raw["compartment"], "depot", "event.compartment")
        _equal(scalar(raw["local_time_h"]), time + offset, "event.local_time_h")
        _equal(scalar(raw["local_nominal_slot_h"]), nominal + offset, "event.local_nominal_slot_h")
    return _event(time, amount), nominal


def _states(raw_states, *, oxc, slots, offset=0):
    """Function: Preserve candidate order and explicit events.
    Inputs: States, model, slots, offset.
    Outputs: Public candidates, never inferred from labels.
    """
    result = []
    for state in _sequence(raw_states, "states", nonempty=True):
        keys = "state_event_id manuscript_id machine_bitstring "
        keys += ("machine_chronological_bits nominal_slots_h taken_by_slot dose_events"
                 if oxc else "prior events")
        _keys(state, keys, "state")
        identifier(state["manuscript_id"], name="manuscript_id")
        if oxc:
            _equal(state["nominal_slots_h"], slots, "state.nominal_slots_h")
            for taken in _sequence(state["taken_by_slot"], "taken_by_slot"):
                _keys(taken, "nominal_time_h taken", "taken_by_slot")
                if scalar(taken["nominal_time_h"]) not in slots or type(taken["taken"]) is not bool:
                    raise InputError("invalid taken_by_slot metadata")
        events = []
        for event in _sequence(state["dose_events" if oxc else "events"], "events"):
```

```
            mapped, nominal = _raw_event(event, oxc=oxc, offset=offset)
            if nominal not in slots:
                raise InputError("event nominal_slot_h is not a declared slot")
            events.append(mapped)
        result.append({"state_id": state["state_event_id"], "events": events})
    return validate_candidate_states(result)


def _history(shared, dose, period, occasions, *, historical_dose=None):
    """Function: Build conditional steady state and common history.
    Inputs: Shared dose, period, occasions, historical dose.
    Outputs: Dosing prehistory and initialization.
    """
    event = _event(shared, dose)
    periodic = _event(shared, dose if historical_dose is None else historical_dose)
    periodic["offset"] = 0
    del periodic["time"]
    return {
        "dosing_prehistory": {
            "events": [event],
            "regimen": {"period": period, "anchor_time": shared, "time_unit": "h", "events": [periodic]},
        },
        "initialization": {
            "mode": "steady_state", "start_time": shared, "event_phase": "before_events",
            "steady_state": {"parameter_rule": "fixed_reference",
                             "reference_occasion_id": occasions[0]["occasion_id"] if occasions else None},
        },
    }


def _oxc_initialization(raw):
    """Function: Validate OXC steady history and time axes.
    Inputs: Raw initialization.
    Outputs: History values, slots and offset.
    """
    _keys(raw, "definition_status interval_h shared_last_dose_h initial_state_time_h initial_state_phase "
          "initial_dose_event earlier_history explicit_occasion_slots_h no_0h_dose_in_one_dose_window "
          "numerical_initial_state time_axes method conditioning iov_present", "initialization")
    for key, expected in {
        "definition_status": "author_approved", "method": "analytic_steady_state",
        "conditioning": "individual_eta_CL", "iov_present": False,
        "initial_state_phase": "immediately_before_shared_last_dose",
        "numerical_initial_state": None, "explicit_occasion_slots_h": [],
    }.items():
        _equal(raw[key], expected, f"initialization.{key}")
    shared = scalar(raw["shared_last_dose_h"])
    period = scalar(raw["interval_h"], positive=True)
    _equal(scalar(raw["initial_state_time_h"]), shared, "initial_state_time_h")
    earlier = _keys(raw["earlier_history"], "interval_h dose_mg last_implicit_dose_h extends_to", "earlier_history")
    _equal(scalar(earlier["interval_h"]), period, "earlier_history.interval_h")
    _equal(scalar(earlier["last_implicit_dose_h"]), shared-period, "last_implicit_dose_h")
    _equal(earlier["extends_to"], "minus_infinity", "earlier_history.extends_to")
    historical_dose = scalar(earlier["dose_mg"], minimum=0)
    axes = _keys(raw["time_axes"], "time_h_axis local_axis manuscript_to_local_offset_h "
                 "manuscript_candidate_slots_h local_candidate_slots_h manuscript_shared_last_h local_shared_last_h "
                 "manuscript_last_implicit_dose_h local_last_implicit_dose_h "
                 "all_events_and_observations_shift_equally native_runtime_axis_adapter_verified", "time_axes")
    _equal(axes["time_h_axis"], "manuscript_study", "time_h_axis")
    _equal(axes["local_axis"], "earliest_candidate_slot_is_zero", "local_axis")
    _equal(axes["all_events_and_observations_shift_equally"], True, "all_events_and_observations_shift_equally")
    if type(axes["native_runtime_axis_adapter_verified"]) is not bool:
        raise InputError("native_runtime_axis_adapter_verified must be boolean")
    slots = _slots(axes["manuscript_candidate_slots_h"], shared)
    offset = scalar(axes["manuscript_to_local_offset_h"])
    _equal(offset, -slots[0], "manuscript_to_local_offset_h")
    _equal(axes["local_candidate_slots_h"], [t+offset for t in slots], "local_candidate_slots_h")
    for key, expected in {
        "manuscript_shared_last_h": shared, "local_shared_last_h": shared+offset,
        "manuscript_last_implicit_dose_h": shared-period, "local_last_implicit_dose_h": shared-period+offset,
    }.items():
        _equal(scalar(axes[key]), expected, f"time_axes.{key}")
    _equal(raw["no_0h_dose_in_one_dose_window"], len(slots) == 1 and 0 not in slots,
           "no_0h_dose_in_one_dose_window")
    event, nominal = _raw_event(raw["initial_dose_event"], oxc=True, offset=offset)
    _equal(event["time"], shared, "initial_dose_event.time_h")
    _equal(nominal, shared, "initial_dose_event.nominal_slot_h")
    return shared, event["amount"], period, historical_dose, slots, offset


def _assemble(values, units, iiv_names, iiv_matrix, iov_matrix, relations, system, error_type):
    """Function: Reuse reference callbacks with explicit independence.
    Inputs: Parameters, units, covariance and original callbacks.
    Outputs: Public model without integration.
    """
    iov = None
    if iov_matrix is not None:
        iov = {"names": ["CL", "V1"], "distribution": "normal", "occasion_key": "occasion_id",
               "covariance_parameters": _covariance_refs(iov_matrix, "pi", values, units)}
    iiv = {"names": iiv_names, "distribution": "normal",
           "covariance_parameters": _covariance_refs(iiv_matrix, "omega", values, units)}
    role = "sd" if error_type == "additive" else "coefficient_sd"
```

```
    return {
        "parameters": {"values": values, "units": units},
        "parameter_model": {"covariate_names": [], "relations": relations},
        "iiv": iiv, "iov": iov, "random_effects_dependence": "independent_blocks",
        "residual_model": {"distribution": "normal", "error_type": error_type,
                           "parameter_names": {role: "sigma"}, "dependence": "conditionally_independent"},
        "predict": LinearPredictor(system),
    }


def _oxc(raw):
    """Function: Map an OXC experiment directly.
    Inputs: Complete raw OXC object.
    Outputs: Five groups without prior rewriting.
    """
    _keys(raw, "model initialization variability residual candidate_states priors schema observations", "OXC")
    _equal(raw["schema"], "oxc_numeric_source_ruv_v1", "schema")
    source = _keys(raw["model"], "model_id compartments analyte dose_unit concentration_unit parameters fixed "
                   "relationships covariates_constant_over_record", "model")
    for key, expected in {
        "model_id": "Lin_OXC_MHD_Hill219", "compartments": ["depot", "central"], "analyte": "MHD",
        "dose_unit": "mg", "concentration_unit": "mg/L", "covariates_constant_over_record": True,
        "relationships": _OXC_RELATIONSHIPS,
    }.items():
        _equal(source[key], expected, f"model.{key}")
    parameters = _number_map(source["parameters"], "CL_p V_p ka BW", "model.parameters",
                             positive=("CL_p", "V_p", "ka", "BW"))
    fixed = _number_map(source["fixed"], "k0 kmax k50 gamma normalization_weight", "model.fixed",
                        positive=("k50", "normalization_weight"))
    _equal(fixed["normalization_weight"], 70, "fixed.normalization_weight (source relationship uses BW/70)")
    covariates = {"BW": parameters.pop("BW")}
    variability = _keys(raw["variability"], "iiv_order iiv_omega iiv_correlation iov_order iov_omega_per_occasion "
                        "iov_occasion_ids joint_order joint_omega source_reference_log_variances distribution",
  "variability")
    for key, expected in {
        "iiv_order": _OXC_IIV, "joint_order": _OXC_IIV, "iiv_correlation": [[1]],
        "iov_order": [], "iov_omega_per_occasion": [], "iov_occasion_ids": [],
        "distribution": "zero_mean_multivariate_normal_log_effects",
    }.items():
        _equal(variability[key], expected, f"variability.{key}")
    source_variances = fields(variability["source_reference_log_variances"], _OXC_IIV,
                              name="source_reference_log_variances")
    scalar(source_variances["Clearance IIV"], minimum=0)
    omega = _matrix(variability["joint_omega"], 1, "joint_omega")
    if not np.array_equal(omega, _matrix(variability["iiv_omega"], 1, "iiv_omega")):
        raise InputError("iiv_omega and joint_omega disagree")
    residual = _keys(raw["residual"], "model sd variance additive_component_present residual_floor normal_zero_mean",
  "residual")
    for key, expected in {"model": "additive", "additive_component_present": True,
                          "residual_floor": None, "normal_zero_mean": True}.items():
        _equal(residual[key], expected, f"residual.{key}")
    sigma = scalar(residual["sd"], positive=True)
    variance = scalar(residual["variance"], minimum=0)
    if not math.isclose(variance, sigma*sigma, rel_tol=8*np.finfo(float).eps, abs_tol=0):
        raise InputError("residual.variance must equal residual.sd squared")
    values = {**fixed, **parameters, "sigma": sigma}
    units = {k: "1" for k in values}
    units.update(CL_p="L/h", V_p="L", ka="1/h", k50="kg", normalization_weight="kg", sigma="mg/L")
    model = _assemble(values, units, ["CL"], omega, None, _oxc_relations, _oxc_system, "additive")
    shared, dose, period, historical_dose, slots, offset = _oxc_initialization(raw["initialization"])
    states = _states(raw["candidate_states"], oxc=True, slots=slots, offset=offset)
    priors = {}
    labels = {s["state_event_id"]: s["manuscript_id"] for s in raw["candidate_states"]}
    for prior in _sequence(raw["priors"], "priors", nonempty=True):
        _keys(prior, "state_event_id manuscript_id probability", "prior")
        key = identifier(prior["state_event_id"])
        if key in priors:
            raise InputError("duplicate prior state_event_id")
        if key not in labels or labels[key] != prior["manuscript_id"]:
            raise InputError("prior labels do not match candidate states")
        priors[key] = prior["probability"]
    observations = []
    for i, record in enumerate(_sequence(raw["observations"], "observations", nonempty=True)):
        _keys(record, "time_h concentration_mg_L", "observation")
        observations.append({"record_id": str(i), "time": record["time_h"], "value": record["concentration_mg_L"],
                             "time_unit": "h", "value_unit": "mg/L"})
    data = {"covariates": covariates, "occasions": [],
            **_history(shared, dose, period, [], historical_dose=historical_dose)}
    return dict(model=model, individual_data=data, observations=observations, candidate_states=states, priors=priors)


def _tac(raw):
    """Function: Map full TAC block structure and occasions.
    Inputs: Complete raw TAC object.
    Outputs: Five groups; unrepresentable covariance rejected.
    """
    _keys(raw, "schema numeric_id parameters fixed slots shared_last shared_dose tau states omega eta_order "
          "residual_sd times observations", "TAC")
    _equal(raw["schema"], "tac_resolved_v1", "schema")
    identifier(raw["numeric_id"], name="numeric_id")
    parameters = _number_map(raw["parameters"], "CL V1 Q V2 ka lag WT", "parameters",
```

```
                            positive=("CL", "V1", "Q", "V2", "ka", "WT"), nonnegative=("lag",))
    fixed = _number_map(raw["fixed"], "X HEM POD Pred theta_CYP3A5 theta_HEM theta_POD theta_Pred "
                        "POD_cap WT_exponent normalization_weight", "fixed",
                        positive=("theta_CYP3A5", "normalization_weight"))
    covariates = {k: fixed.pop(k) for k in ("X", "HEM", "POD", "Pred")}
    covariates["WT"] = parameters.pop("WT")
    values = {**fixed, **parameters, "sigma": scalar(raw["residual_sd"], positive=True)}
    units = {k: "1" for k in values}
    units.update(CL="L/h", V1="L", V2="L", Q="L/h", ka="1/h", lag="h", normalization_weight="kg",
                 theta_POD="1/day", POD_cap="day", theta_Pred="mL/ng")
    shared = scalar(raw["shared_last"])
    dose, period = scalar(raw["shared_dose"], minimum=0), scalar(raw["tau"], positive=True)
    slots = _slots(raw["slots"], shared)
    times, concentrations = numeric_array(raw["times"], name="times"), numeric_array(raw["observations"],
  name="observations")
    if not len(times) or times.shape != concentrations.shape:
        raise InputError("times and observations must be nonempty and have equal lengths")
    boundaries = [shared, *slots, max(48, float(max(times))+1, slots[-1]+1)]
    occasions = [{"occasion_id": f"occ_{i}", "start": a, "end": b}
                 for i, (a, b) in enumerate(zip(boundaries[:-1], boundaries[1:]))]
    order = [*_TAC_IIV]
    for start in boundaries[:-1]:
        order.extend([f"Clearance IOV@occ_{start:g}", f"Central-volume IOV@occ_{start:g}"])
    _equal(raw["eta_order"], order, "eta_order")
    omega = _matrix(raw["omega"], 4+2*len(occasions), "omega")
    represented = np.zeros_like(omega)
    represented[:4, :4] = omega[:4, :4]
    for i in range(4, len(omega), 2):
        represented[i:i+2, i:i+2] = omega[4:6, 4:6]
    if not np.array_equal(omega, represented):
        raise InputError("omega cannot be represented by 0.2.0 independent_blocks: "
                         "cross-block covariance or unequal occasion-specific IOV blocks")
    model = _assemble(values, units, ["CL", "V1", "V2", "ka"], omega[:4, :4], omega[4:6, 4:6],
                      _tac_relations, _tac_system, "proportional")
    states = _states(raw["states"], oxc=False, slots=slots)
    # Function: Declare this study's pre-event readout.
    # Inputs: Original times and concentrations. Outputs: Explicit record phase.
    observations = [{"record_id": str(i), "time": float(t), "value": float(y), "time_unit": "h", "value_unit": "ng/mL",
                     "event_phase": "before_events"}
                    for i, (t, y) in enumerate(zip(times, concentrations))]
    data = {"covariates": covariates, "occasions": occasions, **_history(shared, dose, period, occasions)}
    return dict(model=model, individual_data=data, observations=observations, candidate_states=states,
                priors={s["state_event_id"]: s["prior"] for s in raw["states"]})


def build_inputs(raw, model_name):
    """Function: Adapt raw experiments to public posterior inputs.
    Inputs: Raw JSON mapping and case-insensitive OXC/TAC.
    Outputs: Five keyword argument groups:
        Five kwargs for compute_posterior; caller supplies integration_method and numerical_options.

    Reject unknown keys, unsupported source semantics and unrepresentable random
    effects with InputError. Preserve exact zero priors, source ordering and
    strict zero-variance coordinates. Never mutate raw or load a C003 fixture.
    """
    if not isinstance(model_name, str) or model_name.upper() not in ("OXC", "TAC"):
        raise InputError("model_name must be OXC or TAC")
    result = _oxc(raw) if model_name.upper() == "OXC" else _tac(raw)
    result["model"]["parameter_model"]["covariate_names"] = list(result["individual_data"]["covariates"])
    model = validate_model(result["model"])
    observations = validate_observations(result["observations"])
    data = validate_individual_data(result["individual_data"], model=model, observations=observations)
    states = [validate_candidate_history(s, initialization_start=data["initialization"]["start_time"],
                                         common_events=data["dosing_prehistory"]["events"])
              for s in result["candidate_states"]]
    resolve_random_effects(iiv=model["iiv"], iov=model["iov"], parameters=model["parameters"],
                           occasions=data["occasions"], dependence=model["random_effects_dependence"])
    return dict(model=model, individual_data=data, observations=observations, candidate_states=states,
                priors=validate_priors(result["priors"], candidate_states=states))
```

Outputs, Checks and Runtime

Run	Recorded status	omega_00	omega_01	omega_10	omega_11
1	qualified	0.208774426214603	0.243957039780046	0.275492913017266	0.271775620988086
2	qualified	0.208774426214603	0.243957039780046	0.275492913017266	0.271775620988086
3	qualified	0.208774426214603	0.243957039780046	0.275492913017266	0.271775620988086
4	qualified	0.208774426214603	0.243957039780046	0.275492913017266	0.271775620988086
5	qualified	0.208774426214603	0.243957039780046	0.275492913017266	0.271775620988086

Run	Time (s)	Sources
1	64.125	result; receipt; environment
2	64.59399999999732	result; receipt; environment
3	63.15600000000268	result; receipt; environment
4	63.39099999999962	result; receipt; environment
5	121.34399999999914	result; receipt; environment

Times are copied from the controller receipts. The full results retain the separate internal R elapsed-time field, candidate-level order results, posterior stages and check records. The complete route index also retains the separate unsuccessful attempt; it is not one of the five returned vectors listed here.

4.4. Virtual-Patient MC

OXC C003

Settings and output were captured from the externally authored Adherence ASM dashboard. The application and method are attributed to Liu XQ, Li ZR, Lin WW, Wang J, Gu FQ, Ding JJ and Jiao Z, *Journal of Medical Internet Research* 2026;28:e77917 (original article; DOI).

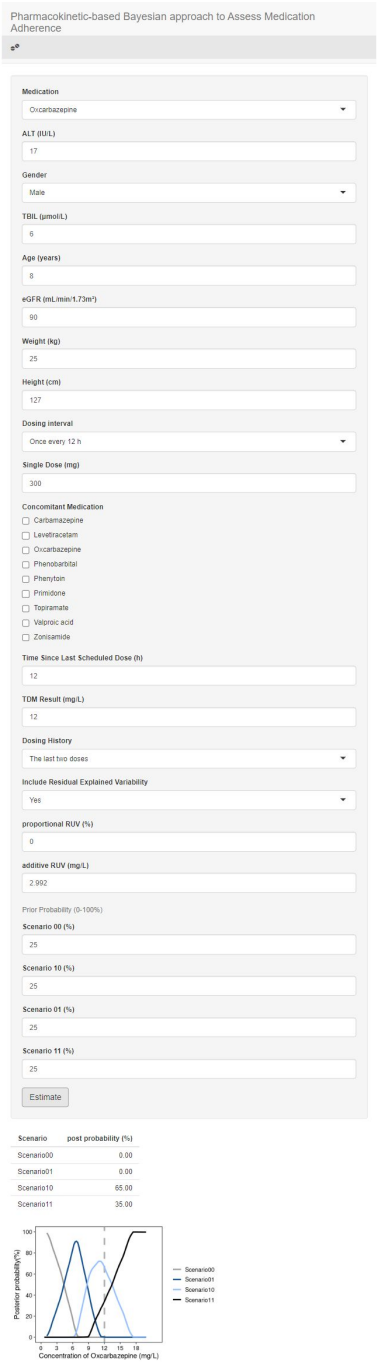


Figure MC1. External C003 dashboard capture, 19 October 2026 (Asia/Shanghai).

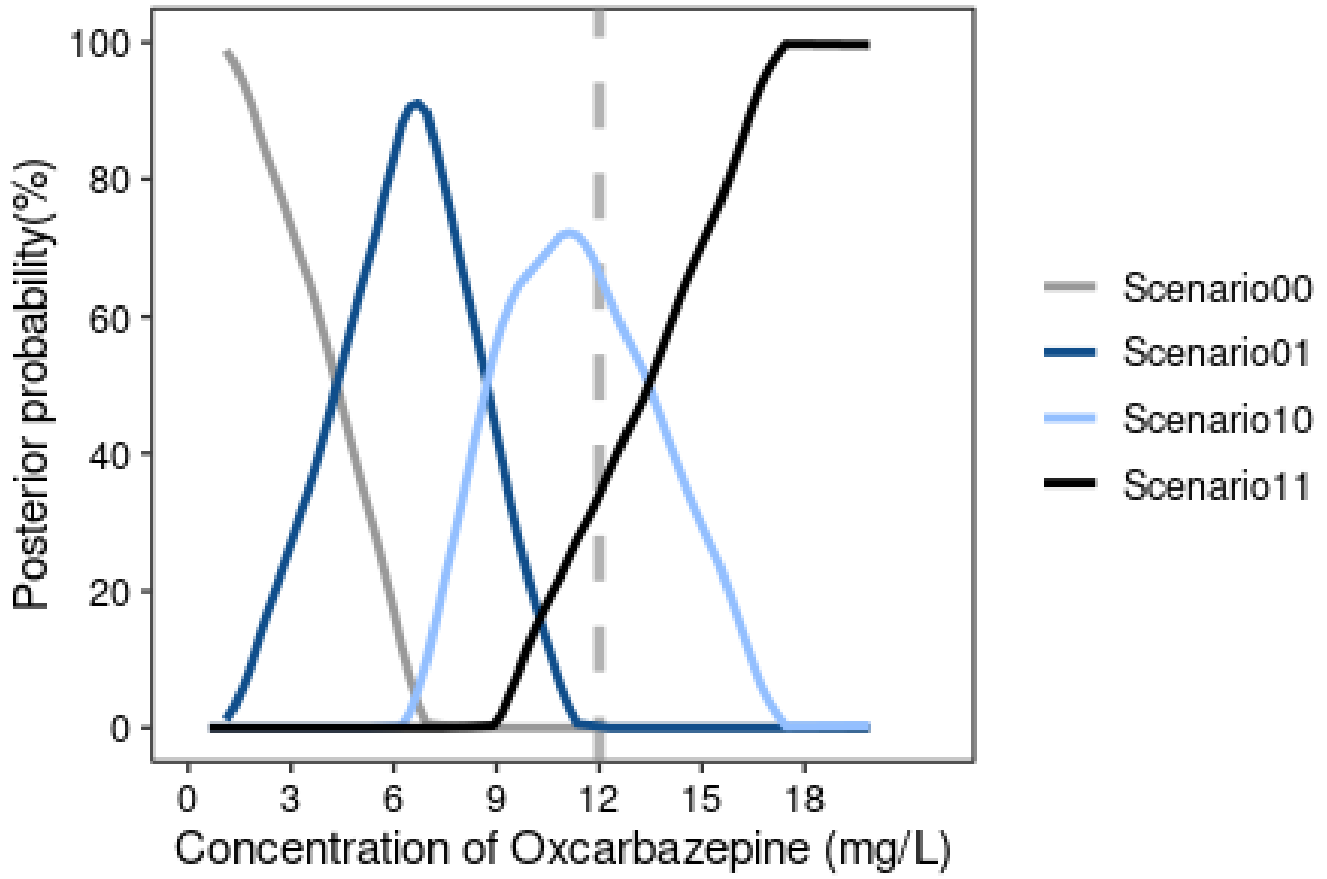


Figure MC2. C003 concentration plot exported by the external dashboard.

Recorded settings retain the requested controls and input readback. Results retain the application's displayed precision; displayed zeros need not be exact. Visible input agreement does not verify the deployed model, analyte convention or estimator.

TAC c003

Unavailable. No TAC c003 virtual-patient Monte Carlo (MC) result, runtime or executable reference was supplied in the designated materials. No external MC application is included.

5. Run Times and Metrics

Definitions of quantitative metrics are provided in Table 3 of the main text. `Unavailable` denotes missing or excluded values, distinct from reported zeros.

Output availability

For the three computational routes, a single valid output must contain posterior probabilities correctly corresponding to all candidate states, with each probability being a finite value within [0,1] and the sum of probabilities deviating from 1 by no more than 10^{-6}. The scenario tables report the number of valid outputs across five formal runs and the number of candidate states.

Repeatability

Across five runs with unchanged inputs and numerical settings, the sample standard deviation (SD) and range of the posterior probabilities for each candidate state were summarized separately before and after normalization, and their maximum values across candidate states are reported. The virtual patient approach had only a single web record per scenario; repeatability was not evaluated.

For routes obtaining five valid outputs, we normalized the probability vector from each output separately and then averaged them by candidate state to form the mean posterior probability vector, which was used for subsequent cross-method comparisons and descriptions of the posterior probability distribution.

Cross-method agreement

Pairwise comparisons among the three computational routes required both routes to obtain five complete valid outputs under matched scenarios, inputs, and prior settings. With candidate states matched by identical dosing history, the mean posterior probability vectors were used to calculate d_∞, TV, d_2, and d_H. Flags indicating failed stability checks in R aghq were retained, but they were not used alone as an exclusion criterion.

Comparisons with the virtual patient approach were matched by visible web input settings and directly used the web-displayed probabilities at their displayed precision without further normalization. Web records with mismatched candidate sets or prior settings, or with displayed probabilities not summing to 100%, were retained separately and excluded from quantitative comparisons.

Ranking comparisons examine both the highest-probability states (including ties) and the complete rankings of candidate states. For the computational routes, two sets of results are retained based on exact equality and a tie tolerance of 10^{-6}; ranking figures use the latter, whereas ties in the virtual patient results are defined by exact equality of the displayed values. The tie-grouping algorithm for the complete rankings is provided in the detailed comparison records in Online Resource 4.

Rank 1 indicates the highest probability. Open R symbols indicate failure to pass stability checks. States reported as zero were not ranked; a web-displayed zero does not necessarily represent an exact zero probability.

Reference scenarios and sensitivity scenarios are reported separately. Sensitivity scenarios are summarized by model, route pair, and category, reporting the median, 95th percentile, and maximum for distance metrics alongside ranking agreement. Each denominator counts available values for that measure; unavailable comparisons are not counted as zero or disagreement. P95 uses linear interpolation (R type 7; NumPy linear). When only one value is available, that value is returned.

Posterior probability summaries

For each scenario, the complete posterior probabilities across candidate states are listed first, followed by the maximum probability (P_{max}), top-two probability gap (G), and normalized entropy (H_N). The three computational routes use the aforementioned mean posterior probability vectors, whereas the virtual patient approach results included in the comparisons use the web-displayed probability vectors.

Run times and numerical checks

Times are wall-clock seconds. Timing scope, environment and software version are in the source records. TAC Python-Laplace times cover the parent-observed subprocess; external web times include service and communication delays.

Recorded statuses are `valid`, `qualified`, `failed`, `incomplete`, `not_assessed` and `reported`. A `qualified` output has an unresolved numerical check; `reported` denotes a web-displayed result with unavailable internal checks. Interrupted or superseded attempts are listed separately from the five formal run records.

Scope	Records
Laplace calculations	Optimization termination, stationarity and curvature diagnostics
NONMEM	Native posterior mapping; fixed-parameter echoes; random-effect and covariance outputs; objective-function/prior reconstruction discrepancy; PHM-SHM discrepancy; ETC positive-definiteness where applicable
External R aghq	Quadrature orders and refinements; changes in posterior probabilities and state-specific log marginal likelihoods; implementation-specific integration diagnostics

Detailed records give the run, state, value, criterion and outcome. Scenario-specific run times and unmet check criteria are provided in "Run times and numerical checks" in Online Resources 2 and 3. These checks are recorded separately from output validity. TAC Python-Laplace used `gtol = 10^-6` throughout.

Full-precision probabilities, distances, rankings, and summary results are provided in the comparison tables and saved output analysis files in Online Resource 4.

6. File Inventory

Online Resource 4 contains the current code, inputs, outputs and logs. Extract its folder beside the three PDFs to use the material links. On Windows, use a short extraction directory to avoid path-length limits.

Scenario file index

Method	OXC reference	TAC reference
Python-Laplace	C003	c003

Method	OXC reference	TAC reference
NONMEM-Laplace	C003	c003
R aghq	C003	c003
Virtual-patient MC	C003	Unavailable

Contents: 654 method-model-scenario entries, 3,270 run records (including failures), and 44 separate MC captures. Files are grouped by method, model, scenario and run; shared code is indexed once. The external MC application is not included.

Scenario results are in Online Resources 2 (OXC) and 3 (TAC).