Finanziato dall'Unione europea NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA

Data-Model Integration and Software Implementation Report

1. Data-model integration and calibration

Within the project "*Analyzing Climate Lobbying with a Simulation Model based on Dynamic Opinions (ALMONDO)*", funded by the European Union – NextGenerationEU under the National Recovery and Resilience Plan (NRRP), Mission 4, Component 2, Investment 1.1, Call PRIN 2022 PNRR (D.D. 1409, 14 September 2022), CUP No. J53D23015400001, we empirically calibrated the model introduced by Giachini et al. (2025) using the observed climate-belief data and adapting the approach proposed by Giachini et al. (2026). In particular, we consider the version of the model with homogeneous baseline under-reaction parameter (λ) and homogeneous strength of motivated directional reasoning (φ). The data used consists of prior and posterior beliefs about the reality of climate change of a representative sample of 1,633 adults in Italy, with respondents randomized across advocacy (IPCC), skepticism (CO2 Coalition), and active-control treatments. In practical terms, this is a data-model handshake: experimental evidence is converted into bounded numerical parameters for downstream scenario execution. The calibration of parameters occurs by substituting the true belief process in the updating model proposed by Giachini et al. (2026) with the slightly different belief updating model used in Giachini et al. (2025). Hence, we maintain the parameters that control experimenter demand effects, while considering a true belief updating model that blends under-reaction with directed motivated reasoning. After changing the name of the variable that controls for survey-induced anchor in beliefs (from λ to ρ) to avoid confusion with the baseline under-reaction parameter in Giachini et al. (2025), we estimated the parameters λ and φ using a different econometric approach than the one used in Giachini et al. (2026). In particular, we decided to find the best fitting values minimizing the cross-entropy between recorded priors and fitted ones. This is because under the specification of true belief updating of Giachini et al. (2025), the model can hardly be estimated by OLS as in Giachini et al. (2026). To do that, we implemented an Octave function and a script that does the estimation. The details of these pieces of code are reported in Appendix A and explained below.

1.1 Dataset preparation

We saved our data in a dataset called `Data_exp_complete.csv`, removed the header row, and remapped all probability-like columns from the interval [0.0001, 0.9999] to [0.01, 0.99]. This rescaling stabilizes optimization, avoids boundary problems when the model later divides by `p0` or by `1 - p0`, and makes the setting consistent with the modeling choices in terms of optimistic and pessimistic models made in Giachini et al. (2025). The code then extracts the prior and posterior vectors used for calibration as follows.

```
p0 = data(:,2); # Priors on Reality of Climate Change
p1 = data(:,16); # Posteriors on Reality of Climate Change
```

Treatment assignment is reconstructed from the first column: IPCC observations are coded as 1, CO2 Coalition observations as -1, and all remaining observations are treated as control. The grouped variable passed into the estimator is implemented as follows.

```
grp = 1; # control
grp = 2; # IPCC/advocacy
grp = 3; # CO2/skepticism
```

1.2 Calibration routine implemented in Octave

The `calibrate_model` function estimates an eight-parameter vector `beta = [mu0 mu1 mu2 rho theta1 theta2 phi lambda]`, with each parameter `mu0 mu1 mu2 rho phi lambda` constrained in `[0,1]` and `theta1 theta2` constrained in `[0 , 1 + mu1 ( 1 - rho ) / ( 1 - mu1 )]` and `[0 , 1 + mu2 rho / ( 1 - mu2 )]`, respectively. The estimation procedure fits posterior probabilities by minimizing summed cross-entropy between observed `p1` and model-implied `p1hat`. Considering that in the mathematical model of Giachini et al. (2025) the weight variables `w` indicate the probability attached to the optimistic model (that is, the skepticism model of Giachini et al., 2026) to be the correct one, the Octave function first converts priors into the weight variable using the following mapping.

```
w0 = (0.99 - p0) / 0.98;
```

Before applying treatment-specific adjustment terms based on `mu0`, `mu1`, `mu2`, `theta1`, `theta2`, and `rho`, the octave function then computes group-specific under-reaction functions as in Giachini et al. (2025) through the following implementation.

```
f1 = phi * abs(w0) + (1 - phi) * lambda;
f2 = phi * abs(1 - w0) + (1 - phi) * lambda;
```

Identifying with `idxA` the individuals assigned to the IPCC treatment, with `idxS` those assigned to the CO2 treatment, and with `idxC` the individuals assigned to the control, the posterior weights attached to the optimistic model to be correct are then computed as follows.

```
    if any(idxA)
      % A: (1-f1)*(0.01*w0/p0) + f1*w0
      w1star(idxA) = (1 - f1(idxA)) .* (0.01 .* w0(idxA) ./
p0(idxA)) + f1(idxA) .* w0(idxA);
```

```
    end

    if any(idxS)
      % S: (1-f2)*(0.99*w0/(1-p0)) + f2*w0
      w1star(idxS) = (1 - f2(idxS)) .* (0.99 .* w0(idxS) ./ (1 -
p0(idxS))) + f2(idxS) .* w0(idxS);
    end

    % w1
    w1 = zeros(N,1);
    if any(idxC)
      w1(idxC) = mu0 .* w1star(idxC) + (1 - mu0) .* rho;
    end
    if any(idxA)
      w1(idxA) = mu1 .* theta1 .* w1star(idxA) + (1 - mu1) .*
rho;
    end
    if any(idxS)
      w1(idxS) = mu2 .* (theta2 .* w1star(idxS)+ 1 - theta2) + (1
- mu2) .* rho;
    end
```

The predicted posteriors are obtained according to the following formula..

```
p1hat = 0.99 - 0.98 * w1;
```

To improve numerical robustness, the function drops non-finite observations, uses random initial values in (0,1), penalizes fitted probabilities that leave the [0,1] interval, and runs Sequential Quadratic Programming (`sqp`) from 80 starting points before selecting the best solution.

The final object returned by the function contains the estimated parameters, average log loss, total objective value, the Root Mean Square Error (RMSE), fitted posteriors, and the optimal parameter vector. This makes the calibration stage both an estimation step and a diagnostic step, since it produces the behavioural parameters needed for simulation as well as fit indicators that can be used for model validation.

1.3 Calibration output

The output of the calibration procedure outlined above is the following.

```
# Estimated parameters
mu0 = 0.8109
mu1 = 0.9054
mu2 = 0.7256
rho = 0.1819
```

```
theta1 = 1.0586
theta2 = 1.0179
phi = 0.1209
lambda = 0.8989
# Average log-loss
avg_log_loss = 0.3639
# Root Mean Square Error
rmse = 0.1667
```

Hence, the estimated values of the key parameters, `lambda` and `phi`, suggest that individuals exhibit substantial baseline underreaction, whereas the strength of directional motivated reasoning is relatively weak. Regarding the estimated parameters capturing experiment-driven biases, the results are consistent with those reported by Giachini et al. (2026). In particular, the survey-induced anchor probability, `rho`, is below 0.5. In the present framework, where `w` denotes the weight assigned to the optimistic model, this implies that the progression and wording of the questions induce individuals to lean toward the pessimistic model. The parameters `mu0`, `mu1`, `mu2` (which capture the strength of adjustment) are close to 1, indicating that the induced anchoring effect is limited. Finally, the parameters `theta1`, `theta2` (which capture the compliance component of the experimenter-demand effect) are also very close to 1, suggesting that this bias plays only a minor role. The qualitative agreement between the estimated parameters capturing experiment-driven biases here and those reported by Giachini et al. (2026) suggests that the proposed framework is highly robust.

1.4 Handoff into ALMONDO software

From a systems perspective, this Octave calibration is the bridge between empirical evidence and the Python simulation stack. The output of the estimation stage should be versioned as a calibration artifact and then mapped into the runtime configuration used by AlmondoModel and ALMONDOSimulator. The README file documents this handoff pattern by exposing empirically calibrated λ and φ values, and by showing a simulator configuration in which calibrated values are passed directly into ALMONDOSimulator.

2. System overview

Here we describe the two complementary parts of the ALMONDO platform: (i) the backend modelling layer, centred on the AlmondoModel and its ALMONDOSimulator, and (ii) the React-based web interface used to configure runs and visualize outputs. At a high level, the ALMONDO platform combines a calibrated opinion-diffusion model with a software stack that lets researchers run scenarios, manage simulation parameters, and inspect results. The model extends the NDlib DiffusionModel abstraction and uses NetworkX

graphs to represent interacting agents. The frontend provides a modern user interface, implemented in React, to bridge computational routines and end users.

The software stack matured through successive development phases. After an initial requirement-analysis stage focused on end-user needs and on the integration of the Python/NDlib mathematical model, we defined the platform architecture, selected the enabling technologies, and produced a Canva mock-up to support discussion of interface functionalities. A first local prototype then established the core workflow, that is, network generation, simulation execution, and result visualization. Subsequent iterations aligned the backend with model updates, extended customization for graph structures, initial conditions, and lobbying strategies, added user-oriented metrics, and culminated in a modular web interface publicly released through the project website and the GitHub repositories for both the model and the web application.

Layer	Role	Technologies / abstractions	Main outputs
Input / evidence	Scenario definition, empirical calibration values, lobbyist settings, initial agent states	Survey experiment evidence from Giachini et al. (2026), parameters, strategy files	Configurable scenario inputs
Modelling core	Opinion diffusion with lobbyist intervention, behavioural bias, and steady-state detection	NDlib DiffusionModel, NetworkX, AlmondoModel from Giachini et al. (2025)	System status trajectories
Execution layer	Single runs, Monte Carlo runs, experiment orchestration, metric computation, and persistence	ALMONDO Simulator	config.json, runs_data.json, figures, stored strategies, metrics
Presentation layer	Run control, parameter configuration, interactive visualization, and analysis	React and Taiwind CSS frontend, API integration, Docker Compose	Dashboard views, network visualizations, and analysis screens

3. Platform data-model integration workflow

The platform data-model integration workflow can be understood as a sequence of transformations that convert scenario definitions and empirical inputs into executable

simulations and reusable analytical outputs. The sequence below consolidates the steps explicitly described in the README files provided in the Simulator repository and the Web Application repository, while also reflecting the development choices adopted during the implementation of the simulation tool and the web interface.

1. Scenario framing. Define the analytical scenario to be simulated, including the number of agents, the time horizon, the type of initial opinion distribution, the lobbying setting, and the graph structure to be used. At this stage, users can either select predefined network topologies or prepare custom graph files for upload.

2. Calibration and parameter selection. Select the opinion-dynamics parameters p_o and p_p together with agent-level susceptibility and resistance parameters (lambda and phi). The provided README files also report empirically calibrated values derived from survey evidence, enabling a data-informed configuration consistent with the underlying behavioural model.

3. Network and agent initialization. Instantiate the network graph in NetworkX, attach node-level configuration values, and assign initial states to agents through the model configuration object. The workflow supports complete, Erdős–Rényi, Watts–Strogatz, and Barabási–Albert graphs, as well as custom edge-list or adjacency-matrix inputs; likewise, initial conditions can be generated from uniform distributions, Gaussian mixtures, fixed values, or custom vectors.

4. Lobbyist data injection. Provide the simulator with lobbyist metadata, including model type, budget, signaling cost, active time window, and strategy references. The workflow also supports the creation and management of both automatic and manual lobbying strategies, making external intervention part of the executable scenario definition.

5. Model instantiation. Create the AlmondoModel as the diffusion engine and bind it to the configured graph and random seed. At this stage, the backend links structural data to behavioural rules, including the updated treatment of agents' behavioural bias and the interaction between endogenous opinion dynamics and external lobbying.

6. Simulation execution. Run either a single simulation or multiple Monte Carlo replications. The execution layer can stop at steady state or at a defined maximum number of iterations, and it is used both for scenario analysis and for numerical validation of graph settings, model parameters, and expected outcomes.

7. Results processing, metrics, and persistence. Compute the indicators selected to support interpretation of the simulation outcomes, including graph-level metrics, node-level metrics, opinion statistics, effective number of clusters, conformity measures, pairwise opinion distances, and the lobbyist performance index. Store configuration and metrics under a scenario-specific results directory, including

configuration files, run-level data, strategy folders, uploaded inputs, and generated figures.

8. Visualization and user access. Expose run management and output exploration through the web interface, where users configure scenarios, launch analyses, and inspect results through dashboard views, interactive network visualizations, and analytical summaries designed to support reproducible and non-technical use of the platform.

4. Software implementation workflow

From a software-engineering perspective, the ALMONDO platform is organized as a layered implementation composed of a model package, a simulator, backend service exposure, and a frontend application. The system evolved from a fully local prototype into a publicly released, modular web platform, with continuous refinement of both backend and frontend components in response to numerical testing and user-oriented requirements.

1. Requirements definition and architectural design. We first identified the functional requirements of the simulation tool, with particular attention to end-user needs and to the integration of the mathematical model implemented in Python with NDlib. This phase also included the definition of the platform architecture, the selection of the main technologies, and the preparation of a Canva mock-up to guide discussion of the user interface.

2. Prototype development in a local environment. A first prototype of the simulator was implemented to establish the core workflow based on network generation, simulation execution, and result visualization. At this stage, the application operated entirely in a local environment and deliberately omitted advanced customization features in order to validate the overall logic of the tool.

3. Backend model alignment and simulator extension. The backend simulation engine was progressively aligned with the latest updates of the mathematical model, including the introduction of agents' behavioural bias and, in later stages, the integration of external lobbyists into the workflow through both automatic and manual strategy creation. The simulator therefore evolved together with the model and not as a separate downstream wrapper.

4. Customization features and input handling. The software was extended to support multiple graph-generation options and flexible initial conditions. Users can now choose among complete, Erdős–Rényi, Watts–Strogatz, and Barabási–Albert graphs, or upload custom graphs via edge-list or adjacency-matrix files; analogously, they can initialize the system through uniform distributions, Gaussian mixtures, fixed values, or custom vectors.

5. Metrics implementation and analytical support. To improve the readability and usefulness of simulation outputs, we implemented a broad set of indicators covering graph overview, node overview, opinion statistics, conformity, effective number of clusters, pairwise opinion distances, and a lobbyist performance index based on the distance between final opinions and the model supported by the lobbyist. Extensive numerical simulations were used to assess which outputs and plots were most informative for end users and to validate the relevant graph and model settings.

6. Backend organization and deployment readiness. The backend project folder was reorganized to provide a clear and intuitive structure for simulation inputs and stored outputs, including uploaded files, strategy resources, run data, and figures. This work supported reproducibility, traceability, and the later exposure of backend services to the web application.

7. Frontend implementation and public release. In the final stage, we developed and finalized a user-friendly web interface that allows users to configure model parameters, define network structures, set lobbying strategies, run simulations, and visualize key outcomes such as opinion dynamics, polarization, and resistance to influence. The interface was implemented with a modular and extensible architecture, supports interactive network visualization and real-time feedback, and was released on the project's GitHub repository together with the ALMONDO model repository and made publicly available through the project website.

8. Validation, iteration, and collaborative refinement. Throughout development, repeated simulation runs, interface reviews, and feedback from the project partners were used to test the tool, identify possible improvements, and guide both backend and frontend refinement. This iterative process was essential to ensuring the robustness, usability, and coherence of the final system.

5. End-to-end operational sequence

The following sequence provides a concise operational view for implementation or testing, reflecting the finalized workflow of the simulation tool and the web application.

- A researcher or analyst defines a scenario, including the number of agents, the behavioural parameters, the lobbying setting, the network structure, and the initial condition specification.

- The backend converts those inputs into model configuration values and simulator arguments, including graph-generation choices or uploaded graph files and, where relevant, automatic or manual lobbying strategies.

- The diffusion model is instantiated on a graph and populated with initial node states, behavioural-bias parameters, and lobbyists parameters and strategies.

- The simulator in the backend executes a single run or a batch of repeated runs across the chosen lambda-phi combinations and stopping conditions.
- Results are saved in a structured scenario directory together with configuration metadata, uploaded resources, reusable strategy files, and generated figures.
- The backend computes graph metrics, node-level indicators, opinion statistics, conformity measures, clustering indicators, and lobbyist performance measures to support interpretation of the run.
- The frontend receives the resulting outputs through the backend integration layer and presents them through dashboard views, interactive visualizations, and analytical summaries that allow users to inspect trajectories, distributions, polarization, and resistance to influence.

6. Conclusion

The ALMONDO platform is a robust research software pipeline combining a calibrated diffusion model, a simulation orchestration layer, and a user-oriented web interface. Its development progressed from requirement analysis and local prototyping to the implementation of advanced customization options, model-aligned backend extensions, user-oriented metrics, and a publicly accessible web application. The distinguishing characteristic of the platform is the tight link between empirical inputs, configurable simulation logic, and interactive analytical access, which together enhance the accessibility, reproducibility, and dissemination of the modelling framework.

7. References and Sources

Giachini, D., Ciambezi, L., Del Rosso, V., Fornari, F., Pansanella, V., Popoyan, L., and Sîrbu, A. (2025). Navigating the Lobbying Landscape: Insights from Opinion Dynamics Models. *arXiv preprint,* arXiv:2507.13767.

Giachini, D., Rossello, G., and Ciambezi, L. (2026). A Survey Experiment on Climate Change Beliefs, Opinions, and Actions. *LEM Working Paper Series*: 2026/08.

ALMONDO-Model Simulator GitHub Repository:
https://github.com/ALMONDO-Project/ALMONDO-Model

ALMONDO-Model Web Application GitHub Repository:
https://github.com/ALMONDO-Project/ALMONDO-WebInterface

Appendix A. Octave Calibration Code

```
################################################################################
################ CALIBRATE MODEL FUNCTION ######################################
################################################################################

function res = calibrate_model(p0, p1, grp, nstarts)
  % Calibrate parameters by minimizing log loss (cross-entropy)
  % between observed p1 and model-implied p1hat.
  %

  if nargin < 4, nstarts = 50; end

  p0  = p0(:);
  p1  = p1(:);
  grp = grp(:);
  N   = length(p0);
  assert(length(p1) == N && length(grp) == N);

  % Drop missing
  ok  = isfinite(p0) & isfinite(p1) & isfinite(grp);
  p0  = p0(ok);
  p1  = p1(ok);
  grp = grp(ok);
  N   = length(p0);

  % Convert p0 -> w0
  w0 = (0.99 - p0) ./ 0.98;

  % Group indices
  idxC = (grp == 1);
  idxA = (grp == 2);
  idxS = (grp == 3);

  % Box bounds for parameters
  % mu0,mu1,mu2,rho,phi,lambda remain in [0,1]
  % theta1,theta2 may exceed 1, so use large finite bounds
  THETA_BOX_UB = 1e6;

  lb = zeros(8,1);
  ub = ones(8,1);
  ub(5) = THETA_BOX_UB;   % theta1
  ub(6) = THETA_BOX_UB;   % theta2

  % ---- model prediction ----
  function p1hat = predict_p1(beta)
    mu0    = beta(1);
    mu1    = beta(2);
    mu2    = beta(3);
    rho    = beta(4);
    theta1 = beta(5);
    theta2 = beta(6);
    phi    = beta(7);
    lambda = beta(8);

    % f1, f2
    f1 = phi .* abs(w0)     + (1 - phi) .* lambda;
    f2 = phi .* abs(1 - w0) + (1 - phi) .* lambda;

    % w1star
    w1star = zeros(N,1);
    w1star(idxC) = w0(idxC);

    if any(idxA)
```

```
    % A: (1-f1)*(0.01*w0/p0) + f1*w0
    w1star(idxA) = (1 - f1(idxA)) .* (0.01 .* w0(idxA) ./ p0(idxA)) ...
                 + f1(idxA) .* w0(idxA);
  end

  if any(idxS)
    % S: (1-f2)*(0.99*w0/(1-p0)) + f2*w0
    w1star(idxS) = (1 - f2(idxS)) .* (0.99 .* w0(idxS) ./ (1 - p0(idxS))) ...
                 + f2(idxS) .* w0(idxS);
  end

  % w1
  w1 = zeros(N,1);
  if any(idxC)
    w1(idxC) = mu0 .* w1star(idxC) + (1 - mu0) .* rho;
  end
  if any(idxA)
    w1(idxA) = mu1 .* theta1 .* w1star(idxA) + (1 - mu1) .* rho;
  end
  if any(idxS)
    w1(idxS) = mu2 .* (theta2 .* w1star(idxS) + 1 - theta2) + (1 - mu2) .* rho;
  end

  % Back to p1: p = 0.99 - 0.98 w
  p1hat = 0.99 - 0.98 .* w1;
end

% ---- objective ----
function L = obj(beta)
  p1hat = predict_p1(beta);

  % penalty if predictions leave [0,1]
  pen = 0;
  below = p1hat <= 0;
  above = p1hat >= 1;

  if any(below)
    pen = pen + 1e6 * sum((p1hat(below)).^2);
  end
  if any(above)
    pen = pen + 1e6 * sum((p1hat(above) - 1).^2);
  end

  % log loss (cross-entropy)
  epslog = 1e-12;
  q = min(max(p1hat, epslog), 1 - epslog);

  L = -sum(p1 .* log(q) + (1 - p1) .* log(1 - q)) + pen;

  if ~isfinite(L)
    L = 1e18;
  end
end

% ---- nonlinear inequality constraints ----
% Octave sqp expects h(beta) >= 0
function c = ineqcon(beta)
  mu1    = beta(2);
  mu2    = beta(3);
  rho    = beta(4);
  theta1 = beta(5);
  theta2 = beta(6);

  c = [ theta1;       % theta1 >= 0
    theta2;           % theta2 >= 0
```

```
    mu1 * (1 - rho) - (1 - mu1) * (theta1 - 1);  % theta1 <= 1 + mu1*(1-rho)/(1-mu1)
    mu2 * rho       - (1 - mu2) * (theta2 - 1)   % theta2 <= 1 + mu2*rho/(1-mu2)
  ];
end

g = [];          % no equality constraints
h = @ineqcon;    % inequality constraints

best_L    = Inf;
best_beta = [];

maxiter = 500;
tol     = 1e-8;

for s = 1:nstarts
  % Draw a feasible starting point
  x0 = rand(8,1);

  % theta1/theta2 may be > 1, so sample them separately
  % using the nonlinear upper bounds implied by x0
  mu1 = x0(2);
  mu2 = x0(3);
  rho = x0(4);

  % Use the exact formulas for initialization only
  if mu1 < 1
    ub_t1 = 1 + mu1 * (1 - rho) / (1 - mu1);
  else
    ub_t1 = THETA_BOX_UB;
  end

  if mu2 < 1
    ub_t2 = 1 + mu2 * rho / (1 - mu2);
  else
    ub_t2 = THETA_BOX_UB;
  end

  ub_t1 = min(ub_t1, THETA_BOX_UB);
  ub_t2 = min(ub_t2, THETA_BOX_UB);

  x0(5) = rand() * ub_t1;
  x0(6) = rand() * ub_t2;

  [xhat, fval, info] = sqp(x0, @obj, g, h, lb, ub, maxiter, tol);

  if isfinite(fval) && any(info == [101, 102, 103, 104]) && fval < best_L
    best_L = fval;
    best_beta = xhat;
  end
end

if isempty(best_beta)
  error('Optimization failed: no feasible solution found.');
end

p1hat_best = predict_p1(best_beta);
rmse_best  = sqrt(mean((p1hat_best - p1).^2));

res = struct();
res.params = struct( ...
  'mu0',    best_beta(1), ...
  'mu1',    best_beta(2), ...
  'mu2',    best_beta(3), ...
  'rho',    best_beta(4), ...
  'theta1', best_beta(5), ...
```

```
    'theta2', best_beta(6), ...
    'phi',    best_beta(7), ...
    'lambda', best_beta(8) );

  res.logloss = best_L / length(p1);   % average log loss
  res.obj     = best_L;                % summed objective
  res.rmse    = rmse_best;
  res.p1hat   = p1hat_best;
  res.best_x  = best_beta;
end


################################################################################
##################### OCTAVE MAIN SCRIPT #######################################
################################################################################

pkg load statistics

data=csvread('Data_exp_complete.csv');
data=data(2:end,:);
lp=0.01;
hp=0.99;
data(:,2:end)=lp + (data(:,2:end) - 0.0001) .* (hp - lp) ./ (0.9999 - 0.0001);

p0=data(:,2)# Priors on Reality of Climate Change
p1=data(:,16); # Posteriors on Reality of Climate Change

N=length(data(:,1));
ipcc=zeros(N,1);
co2=zeros(N,1);
control=zeros(N,1);
for i=1:N
  if data(i,1)==1
    ipcc(i)=1;
  elseif data(i,1)==-1
    co2(i)=1;
  else
    control(i)=1;
  endif
endfor

grp=control+2.*ipcc+3.*co2;

out = calibrate_model(p0, p1, grp, 80);

# Estimated parameters
disp(out.params);
# Average log-loss
avg_log_loss=disp(out.logloss)
# Root Mean Square Error
rmse=disp(out.rmse)
```