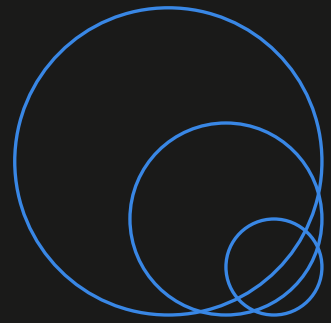
Building Surrogate Models (Proxy/Surrogate Models) — A Practicum

Practical Reservoir Engineering — Data-Driven Simulator Series, Session 2 (Detailed Edition)



Today's Agenda

- Part A: Recap of Session 1 — classification of proxy models —
- Part B: Building surrogate models with response surface methodology (RSM) and regression
- Part C: Building models with machine learning (random forest, neural network)
- Part D: Overview and implementation points of advanced methods (ROM, Fourier Neural Operator, etc.)
- Part E: Principles of accuracy validation

Part A

Recap of Session 1

› Aim of this part: briefly recap what we learned last time—'why proxy models are needed' and 'how they are classified'—as a bridge to today's hands-on parts

› Last time we started from the computational cost of conventional simulation and the challenges of uncertainty assessment (Part A), then organized the role and classification of proxy/surrogate models (Part B)

› Today we move on to the stage of 'actually building them hands-on,' following this classification

A1: Recap (1): Why Proxy Models Are Needed

■ Grid/time-step growth, history matching, Monte-Carlo-style uncertainty assessment, and well-placement optimization all share the bottleneck of 'repeating the simulation many times'

■ A proxy model is the idea of replacing this iterative computation with a fast approximate model

■ Its main purposes are reducing computational cost and applying it to uncertainty quantification and optimization

Source (internal): Research/Deliverables 'DataDrivenSimulator_S1_Detailed.pptx', Parts A and B

A2: Recap (2): Overview of the Three Categories

- (1) Statistical methods: response surface methodology (RSM), polynomial regression, kriging
- (2) Machine-learning methods: random forest, support vector machine, neural network (MLP)
- (3) Advanced methods: reduced-order models (ROM/POD), Fourier Neural Operator, deep-learning surrogates (CNN/LSTM, etc.)

Source: ResearchGate, 'Advancements in Machine Learning-Driven Proxy Modelling Techniques for Reservoir Engineering: A Systematic Review'
https://www.researchgate.net/publication/399879336_Advancements_in_Machine_Learning-Driven_Proxy_Modelling_Techniques_for_Reservoir_Engineering_A_Systematic_Review

Classification Map of Proxy Models (Review)

Statistical methods	Machine-learning methods	Advanced methods	Hybrid
Response surface methodology (RSM)	Random forest	ROM/POD	dual-driven
Polynomial regression	SVM	Fourier Neural Operator	Physics-informed NN
Kriging	Neural network	Deep-learning surrogate	

A3: Recap (3): Benefits and Limitations

■ Benefit: major reduction in computational cost; good fit with optimization and uncertainty assessment that presuppose many evaluations

■ Limitations: accuracy drop when extrapolating outside the training-data range, dependence on data quantity/quality, and challenges of model explainability

■ Choose the method (statistical / machine-learning / advanced) according to data volume, strength of nonlinearity, and interpretability requirements

Source: Academia.edu, 'Development of proxy models for petroleum reservoir simulation: a systematic literature review and state-of-the-art'
https://www.academia.edu/44346638/Development_of_proxy_models_for_petroleum_reservoir_simulation_a_systematic_literature_review_and_state-of-the-art

A4: On to Today's 'Actually Build It' Phase

■ Today we cover concrete implementation steps in the order Part B (statistical) → Part C (machine learning) → Part D (advanced)

■ Part E organizes the principles of accuracy validation

Part B

Building Surrogate Models with Response Surface Methodology (RSM) and Regression

› Aim of this part: understand the concrete procedures and tools for building surrogate models with statistical methods (RSM, polynomial regression, kriging)

› We cover the sequence: sampling design via Design of Experiments (DOE) → model fitting → validation

› We also introduce the Python libraries used (pyDOE2, scikit-learn, SMT, etc.)

B1: Overall Design of Surrogate-Model Building with RSM

■ Response Surface Methodology (RSM) is a statistical method that runs simulations for a small number of input-parameter combinations and fits a function (e.g., a polynomial) to the results to build a response surface

■ Its aim is to approximate, with few runs, the relationship between inputs (design variables such as properties and operating conditions) and outputs (responses such as production and recovery factor)

■ A classic method widely used in practice for history matching, uncertainty assessment, and optimization

Source: Penn State University, STAT 503, 'Lesson 11: Response Surface Methods and Designs' https://online.stat.psu.edu/stat503/lesson/11

B2: Purpose and Overview of Design of Experiments (DOE)

■ DOE is a sampling-design method for covering the input space as efficiently as possible with a limited number of simulation runs

■ It is broadly divided into classical factorial designs (full/fractional factorial), response-surface designs (Box-Behnken, central composite), and space-filling designs (Latin hypercube sampling, etc.)

■ Choose the design method according to the purpose (screening / response-surface estimation / approximating the whole space)

Source: SPE Annual Technical Conference and Exhibition, 'Experimental Design or Monte Carlo Simulation? Strategies for Building Robust Surrogate Models' https://onepetro.org/SPEATCE/proceedings-abstract/15ATCE/15ATCE/D031S030R005/180367

B3: Concrete Steps (1): Classical Factorial and Response-Surface Designs

■ A full factorial design covers all level combinations of each factor, but the number of runs grows exponentially with the number of factors

■ Fractional factorial and Plackett-Burman designs are used in the screening stage to narrow down the most influential factors

■ Box-Behnken and central composite designs are representative response-surface designs for fitting a quadratic (curved) response

Source: GitHub, clicumu/pyDOE2, 'Design of experiments for Python' https://github.com/clicumu/pyDOE2

B4: Concrete Steps (2): Latin Hypercube Sampling (LHS)

■ LHS is a space-filling design method that divides each input variable's range into equal-probability intervals and samples one point from each interval without overlap

■ Even with many factors, it covers the whole input space in a balanced way with few runs

Source: Viana, F.A.C., 'A Tutorial on Latin Hypercube Design of Experiments', Quality and Reliability Engineering International, Vol.32, No.5 (2016) https://onlinelibrary.wiley.com/doi/10.1002/qre.1924 (subscription required; see purchase_list)

B5: Python Tools (1): Sampling Design with pyDOE2

■ pyDOE2 is an open-source library that provides full factorial (fullfact), two-level factorial (ff2n), fractional factorial (fracfact), Plackett-Burman (pbdesign), Box-Behnken (bbdesign), central composite (ccdesign), and Latin hypercube (lhs) designs

■ Call it like `from pyDOE2 import lhs` and generate a design matrix simply by specifying the number of factors and samples

■ Based on the generated design points, run the simulator (or benchmark data) to obtain response values

Source: GitHub, clicumu/pyDOE2, 'Design of experiments for Python' https://github.com/clicumu/pyDOE2

B6: Fitting Procedure for Polynomial Regression

■ Step 1: Reduce multicollinearity by centering and coding (normalizing) the input variables

■ Step 2: Try fitting in the order first-order model (main effects only) → second-order model (including interactions and squared terms), and decide the necessary degree

■ Step 3: Estimate the regression coefficients by least squares (in Python, use `numpy.polyfit`, `statsmodels.OLS`, etc.)

Source: MDPI, 'Importance of Using Modern Regression Analysis for Response Surface Models in Science and Technology' https://www.mdpi.com/2076-3417/15/13/7206

B7: Validating the RSM Model (ANOVA, Lack-of-Fit Test)

■ Use the F-test of analysis of variance (ANOVA) to check whether the model as a whole is statistically significant

■ Use the lack-of-fit test to check whether the chosen polynomial degree adequately captures the actual curvature of the data

■ Also check the coefficient of determination (R2) and residual plots to judge the model's validity comprehensively

Source: Penn State University, STAT 503, 'Lesson 11: Response Surface Methods and Designs' https://online.stat.psu.edu/stat503/lesson/11

B8: Basics of Kriging (Gaussian Process Regression)

■ Kriging is a method derived from spatial statistics (geostatistics) that predicts the value at unknown points using a covariance function (kernel) based on distance from known points

■ A feature not found in polynomial regression is that it yields not only the predicted value but also the prediction uncertainty (variance)

■ In machine learning it is treated under almost the same theoretical framework as 'Gaussian Process Regression'

Source: Sacks, J., Welch, W.J., Mitchell, T.J., Wynn, H.P., 'Design and Analysis of Computer Experiments', Statistical Science, Vol.4, No.4 (1989) (paper, subscription required; see purchase_list)

B9: Implementation Tools — Python Libraries for Kriging and Response Surfaces

■ scikit-learn's `GaussianProcessRegressor` lets you easily implement Gaussian process regression by specifying a kernel (e.g., RBF). The kernel hyperparameters are automatically optimized by maximizing the log marginal likelihood (use `n_restarts_optimizer` to guard against local optima)

■ SMT (Surrogate Modeling Toolbox) is a Python-only library that bundles surrogate-model methods such as kriging and radial basis functions along with sampling methods, featuring gradient (derivative) support

■ Both can be installed via `pip install`. The Session 2 exercise first covers the scikit-learn implementation

Source: scikit-learn, 'Gaussian Processes' https://scikit-learn.org/stable/modules/gaussian_process.html ; SMT: Surrogate Modeling Toolbox documentation https://smt.readthedocs.io/en/latest/

B10: Choosing Between RSM and Kriging, and Their Limits

■ For small-data, low-dimensional (few-factor) problems, polynomial regression and RSM are easy to interpret and practical

■ For systems with strong nonlinearity and complex factor interactions, kriging tends to be more accurate, but its computational cost (inverting the covariance matrix) grows with the number of factors and data points

■ For both, note that accuracy drops greatly outside the training-data range (the extrapolation region)

Source: ResearchGate, 'Response Surface Methodology Approach for History Matching and Uncertainty Assessment of Reservoir Simulation Models'
https://www.researchgate.net/publication/254527939_Response_Surface_Methodology_Approach_for_History_Matching_and_Uncertainty_Assessment_of_Reservoir_Simulation_Models

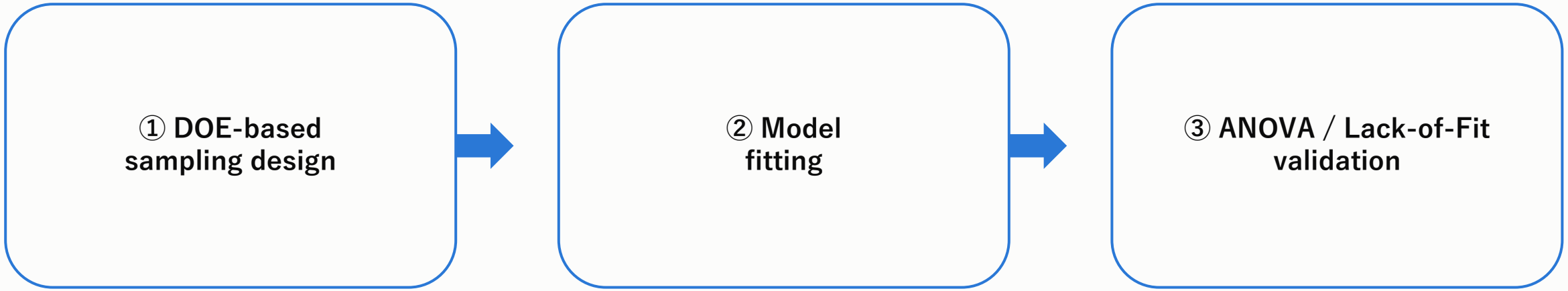
B11: Part B Summary

■ RSM, regression, and kriging are built in the flow: sampling design via DOE → model fitting → validation via ANOVA/lack-of-fit, etc.

■ The combination of pyDOE2 (sampling) and scikit-learn/SMT (fitting) is the basic form of implementation in Python

■ The next part covers machine-learning methods (random forest, neural network) that are stronger with nonlinearity and larger data

The Three Steps of RSM Construction



Part C

Building Models with Machine Learning (Random Forest, Neural Network)

› Aim of this part: understand concrete implementation steps using scikit-learn, PyTorch, etc., and the principles of hyperparameter tuning

› Compared with Part B's statistical methods, these are more accurate for strongly nonlinear systems and larger data, but require countermeasures against overfitting

› We cover two methods: random forest (a decision-tree ensemble) and neural network (MLP)

C1: Overall Design of Surrogate-Model Building with Machine Learning

■ The steps are largely common: data collection/preprocessing → split into train/validation/test → model training → hyperparameter tuning → accuracy validation (detailed in Part E)

■ Normalization/standardization of inputs (properties, operating conditions, etc.)—directly tied to training stability, especially for neural networks

■ Feature engineering (adding interaction terms, removing irrelevant features, etc.) contributes to improved accuracy

Source: ResearchGate, 'Advancements in Machine Learning-Driven Proxy Modelling Techniques for Reservoir Engineering: A Systematic Review'
https://www.researchgate.net/publication/399879336_Advancements_in_Machine_Learning-Driven_Proxy_Modelling_Techniques_for_Reservoir_Engineering_A_Systematic_Review

C2: Random Forest Implementation (1): Basic Form

■ In scikit-learn you can implement it in a few lines: after `from sklearn.ensemble import RandomForestRegressor`, train with `.fit(X_train, y_train)` and predict with `.predict(X_test)`

■ It is ensemble learning combining many decision trees (`n_estimators`); each tree is trained on a random subsample of features and data (bagging)

■ Multiple applications to reservoir production forecasting and proxy-model construction have been reported

Source: SBC, 'Predicting oil field production using the Random Forest algorithm' https://sol.sbc.org.br/index.php/sibgrapi_estendido/article/download/23277/23104/

C3: Random Forest Implementation (2): Feature Importance and OOB Evaluation

■ The `feature_importances_` attribute lets you quantitatively grasp which input parameters affect the prediction (RF's high interpretability is an advantage)

■ The out-of-bag (OOB) score (`oob_score=True`) gives an estimate of model accuracy without preparing separate validation data

■ An implementation of random forest regression has also been reported for predicting well-log values using the Volve dataset

Source: Towards Data Science, 'Random Forest Regression for Continuous Well Log Prediction' https://towardsdatascience.com/random-forest-regression-for-continuous-well-log-prediction-61d3ec1c683a/

C4: Principles of Hyperparameter Tuning (1): Key Parameters

■ `n_estimators` (number of trees): more is more stable but increases computational cost

■ `max_depth` / `max_leaf_nodes` (tree depth): too large leads to overfitting, too small to underfitting

■ `min_samples_leaf` / `min_samples_split`: control the minimum number of samples at leaf nodes to suppress overfitting

Source: scikit-learn, 'Tuning the hyper-parameters of an estimator' https://scikit-learn.org/stable/modules/grid_search.html

C5: Principles of Hyperparameter Tuning (2): Search Methods

■ Grid search (`GridSearchCV`): exhaustively evaluate all combinations of candidate parameters. Comprehensive, but costly when the number of combinations is large

■ Random search (`RandomizedSearchCV`): randomly sample from the parameter space and evaluate. Efficient in high dimensions

■ Bayesian optimization (Optuna, etc.): intelligently choose the next parameters to evaluate based on past results, reaching good solutions with fewer trials

Source: Akiba, T. et al., 'Optuna: A Next-generation Hyperparameter Optimization Framework', KDD 2019, arXiv:1907.10902 https://arxiv.org/abs/1907.10902

Evolution of Hyperparameter Search Methods

C6: Neural Network Implementation (1): Model Construction

■ In PyTorch, define an MLP by stacking fully connected layers (`nn.Linear`) and activation functions (ReLU, etc.) in a class inheriting `torch.nn.Module`

■ Match the input-layer node count to the number of features and the output-layer node count to the number of response variables to predict (production, pressure, etc.)

■ Adjust the number of hidden layers and nodes according to the data volume and problem complexity (detailed in Part C8)

Source: PyTorch official documentation https://pytorch.org/docs/stable/index.html

C7: Neural Network Implementation (2): Training Loop

■ Define a loss function (mean squared error MSE, mean absolute error MAE, etc.) and update the weights with an optimizer such as Adam

■ Repeat forward pass → loss computation → backward pass → parameter update for the number of epochs

■ Using mini-batches (`DataLoader`) enables stable training even with large data

Source: PyTorch official documentation https://pytorch.org/docs/stable/index.html

C8: Principles of Hyperparameter Tuning (NN)

■ Main targets: number of layers and nodes per layer, learning rate, batch size, number of epochs

■ Too large a learning rate diverges; too small slows convergence. Using a learning-rate scheduler is also effective

■ Cases combining a framework like Optuna with PyTorch to search these automatically have been reported

Source: Towards Data Science, 'Hyperparameter Tuning of Neural Networks with Optuna and PyTorch' https://towardsdatascience.com/hyperparameter-tuning-of-neural-networks-with-optuna-and-pytorch-22e179efc837/

C9: Countermeasures for Overfitting and Insufficient Data

■ Prevent the model from overfitting the training data with regularization (L2 regularization, weight decay) and dropout (Dropout layers)

■ Early stopping (halting training once the validation loss stops improving) is another widely used countermeasure

■ When data are scarce, switching to Part B (statistical methods) or considering data augmentation and transfer learning are also options (placeholder: specific data-augmentation methods to be finalized during material development)

C10: Choosing Between Random Forest and Neural Network

■ Random forest: relatively easy to implement and tune, high interpretability via feature importance, robust on medium-sized data

■ Neural network: strong on large data, high-dimensional inputs, and complex nonlinear relationships, but demands more data and compute and requires more hyperparameter-tuning effort

■ In practice, a realistic approach is to first establish a baseline accuracy with random forest and then try to improve accuracy with a neural network as needed

Source: ResearchGate, 'Advancements in Machine Learning-Driven Proxy Modelling Techniques for Reservoir Engineering: A Systematic Review'
https://www.researchgate.net/publication/399879336_Advancements_in_Machine_Learning-Driven_Proxy_Modelling_Techniques_for_Reservoir_Engineering_A_Systematic_Review

Random Forest vs. Neural Network

Random forest		Neural network
Easy to implement and tune	Choosing between them	Strong on large, high-dimensional data
Easy to interpret via feature importance		Requires more tuning effort

C11: Part C Summary

■ Both random forest and neural network can have their basic form implemented in a few dozen lines using scikit-learn/PyTorch

■ Hyperparameter tuning (grid search → random search → Bayesian optimization) and overfitting countermeasures (regularization, early stopping) determine accuracy

■ The next part covers advanced methods (ROM, FNO) effective when data and computational-cost constraints are even tighter

Part D

Overview and Implementation Points of Advanced Methods (ROM, Fourier Neural Operator, etc.)

› Aim of this part: organize implementation-side considerations for the Reduced-Order Model (ROM) and Fourier Neural Operator (FNO), etc., introduced in Session 1

› This course stays at an introductory overview and does not cover detailed mathematical derivations (following the policy from the Session 1 draft)

› Note that the hurdles of data volume, computational resources, and expertise are higher than for statistical and machine-learning methods

D1: The Role of Advanced Methods (Review)

■ A ROM (Reduced-Order Model) compresses high-dimensional simulation results into a few representative variables (a low-dimensional space) for computation

■ The Fourier Neural Operator (FNO) is a type of 'neural operator' that learns a mapping (operator) from function space to function space

■ Both are better suited than statistical methods and conventional machine learning for handling large, high-dimensional simulation output (spatial distributions, entire time series)

Source (internal): Research/Deliverables 'DataDrivenSimulator_S1_Detailed.pptx', B9-B10

D2: ROM Implementation Steps (1): Snapshot Collection and SVD

■ Step 1: Run the physics-based simulator under multiple input conditions (designed via DOE, etc.) and collect the state variables (pressure, saturation distributions) at each time and condition as 'snapshots'

■ Step 2: Apply singular value decomposition (SVD) to the snapshot matrix to obtain the Proper Orthogonal Decomposition (POD) basis (principal modes)

■ Decide the number of basis vectors (the degree of dimensionality reduction) by looking at how the singular values decay

Source: Cardoso, M.A., 'Development and Application of Reduced-Order Modeling for the Solution of Oil Reservoir Simulation Problems', Stanford University PhD Dissertation(SUPRI-B) https://pangea.stanford.edu/ERE/pdf/pereports/PhD/Cardoso09.pdf

D3: ROM Implementation Steps (2): Time Evolution and Reconstruction in the Low-Dimensional Space

■ Project the governing equations (conservation laws for pressure and saturation) onto the obtained low-dimensional basis (Galerkin projection) and compute the time evolution in the low-dimensional space

■ Examples combined with additional methods for handling nonlinearity, such as Trajectory Piecewise Linearization (TPWL), have been reported

■ Reconstruct the low-dimensional solution back into the original high-dimensional space (the full grid) using the basis to obtain the pressure and saturation distributions

Source: Stanford SUETRI-B, 'Reduced-Order Modeling for Compositional Simulation by Use of Trajectory Piecewise Linearization' https://suetri-b.stanford.edu/publications/journal-articles/reduced-order-modeling-compositional-simulation-use-trajectory

D4: ROM Implementation Considerations and Python Tools

■ POD itself can be implemented with just the SVD functionality of numpy/scipy, but dedicated Python ROM libraries such as EZyRB also exist

■ The need to integrate with the simulator itself (an environment for multiple runs to obtain snapshots) is an implementation hurdle that differs from statistical and machine-learning methods

■ (Placeholder: the specific ROM implementation library and exercise data used in this course to be finalized during material development)

Source: GitHub, vassilikitsios/snapshot_pod_rom_py, 'Python code to calculate proper orthogonal decomposition modes'
https://github.com/vassilikitsios/snapshot_pod_rom_py

D5: FNO Implementation Steps (1): Data Format and Preprocessing

■ FNO treats inputs and outputs as functions on a regular grid. Prepare spatial data such as permeability fields and pressure distributions as tensors (multidimensional arrays)

■ Resolution independence (inference possible on grids of a different resolution than during training) is considered a theoretical feature

■ Applications as a proxy model for cases with varying well conditions and permeability fields have been reported

Source: Li, Z. et al., 'Fourier Neural Operator for Parametric Partial Differential Equations', arXiv:2010.08895(2020) https://arxiv.org/abs/2010.08895

D6: FNO Implementation Steps (2): Building with the neuraloperator Library

■ Using the official implementation library `neuraloperator` (part of the PyTorch ecosystem), you can define a model in a few lines, e.g., `from neuralop.models import FNO`

■ By parameterizing the integral kernel in Fourier space, it is said to learn long-range spatial correlations more efficiently than convolutional neural networks

■ For training, use a PyTorch training loop (loss function, Adam, etc.) as with an ordinary neural network

Source: neuraloperator(GitHub official), 'Learning in infinite dimension with neural operators' https://github.com/neuraloperator/neuraloperator ; neuraloperator documentation, 'Fourier Neural Operators' https://neuraloperator.github.io/dev/theory_guide/fno.html

D7: Hurdles to Adopting Advanced Methods

■ Data volume: both ROM and FNO presuppose a sufficient number of simulation runs (snapshots, training samples) and often require more training data than statistical methods

■ Computational resources: for deep-learning-based methods (FNO, etc.), training in a GPU environment is often the realistic option

■ Expertise: implementation and tuning can require background knowledge such as partial differential equations and signal processing (Fourier transforms)

Source (internal): Research/References/NewCourseProposal_DataDrivenSimulator.md (Chapter 1)

D8: Part D Summary

■ ROM (POD) approximates high-dimensional, large-scale simulation via dimensionality reduction by SVD plus time evolution in a low-dimensional space, while FNO does so via operator learning in Fourier space—each a different approach

■ For both, the implementation hurdles are higher than for statistical and machine-learning methods, so this course limits itself to an overview and the conceptual approach to implementation

■ The next part organizes the principles of 'accuracy validation' common to all the methods covered so far

ROM vs Fourier Neural Operator

ROM(POD)		Fourier Neural Operator
Dimensionality reduction by SVD	Advanced methods	Operator learning in Fourier space
Time evolution in a low-dimensional space		Resolution-independent

Part E

Principles of Accuracy Validation

› Aim of this part: organize the principles of surrogate-model accuracy validation common to any method (RSM, machine learning, advanced methods)

› We cover two basic validation approaches—cross-validation and the hold-out method—and representative evaluation metrics (RMSE, R2, MAE)

› In the Part F exercise (Ex.2) we actually compute these

E1: The Hold-out Method (Train/Validation/Test Split)

■ Split the data into training, validation, and test sets; train the model on the training set, tune hyperparameters on the validation set, and evaluate final accuracy on the test set

■ You can easily split with scikit-learn's `train_test_split` function (specify the ratio with the `test_size` argument)

■ Note that with little data, a single simple split makes the evaluation result sensitive to how the split is made

Source: scikit-learn, 'Cross-validation: evaluating estimator performance' https://scikit-learn.org/stable/modules/cross_validation.html

E2: Cross-Validation (k-fold CV)

■ Split the data into k groups (folds), use one as validation and the rest for training, repeat k times, and average the evaluations

■ In practice k=5–10 is often chosen. It is considered an especially effective validation means for surrogate-model building with little data

■ It can be implemented with scikit-learn's `cross_val_score`, `KFold`, and `cross_validate` functions

Source: scikit-learn, 'Cross-validation: evaluating estimator performance' https://scikit-learn.org/stable/modules/cross_validation.html

E3: Evaluation Metrics (1): RMSE and MAE

■ RMSE (Root Mean Squared Error): squares the differences between predicted and actual values, averages them, and takes the square root. It strongly penalizes large errors

■ MAE (Mean Absolute Error): the mean of the absolute differences between predicted and actual values. Less affected by outliers

■ For both, smaller values indicate higher accuracy, but note that the units are the same as the original response variable (production, etc.)

Source: MachineLearningMastery.com, 'A Gentle Introduction to k-fold Cross-Validation' (referenced as a general explanation of evaluation metrics)
https://machinelearningmastery.com/k-fold-cross-validation/

E4: Evaluation Metrics (2): Coefficient of Determination R2 and When to Use Which

■ R2 (coefficient of determination): indicates how much of the response's variance the model explains (closer to 1 is a better fit)

■ Since RMSE/MAE indicate 'the size of the error' and R2 indicates 'explanatory power,' it is common to evaluate using both together

■ When comparing multiple methods (RSM, random forest, neural network, etc.), it is important to align on the same evaluation metric and the same test data

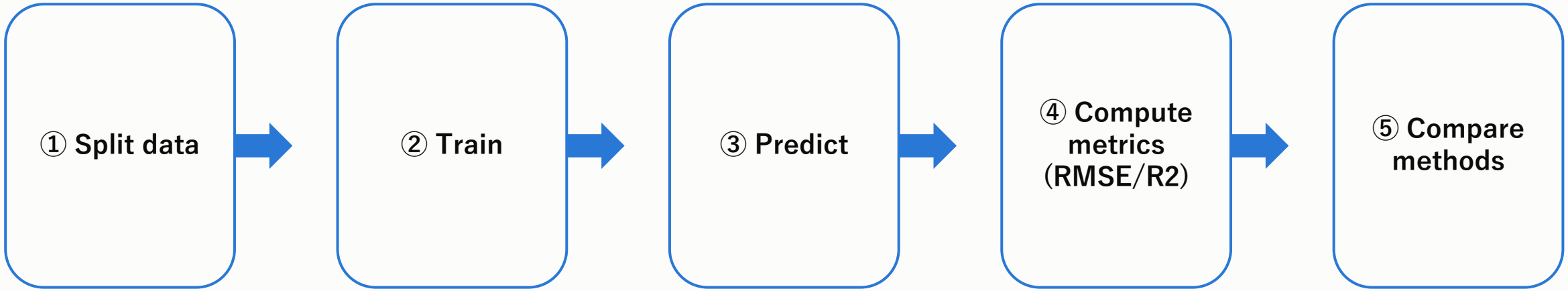
Source: scikit-learn, 'Cross-validation: evaluating estimator performance' https://scikit-learn.org/stable/modules/cross_validation.html

E5: Part E Summary — The Accuracy-Validation Workflow

■ Steps: split the data (hold-out or k-fold) → train → predict → compute evaluation metrics (RMSE, MAE, R2) → compare across methods

■ It is also important to state the applicable range (the range of input parameters covered by the training data) and to caution against applying it outside that range (extrapolation)

The Accuracy-Validation Workflow



List of References (Sources) (1/4)

■ Li, Z. et al., 'Fourier Neural Operator for Parametric Partial Differential Equations'(arXiv:2010.08895, 2020) https://arxiv.org/abs/2010.08895

■ ResearchGate, 'Advancements in Machine Learning-Driven Proxy Modelling Techniques for Reservoir Engineering: A Systematic Review' https://www.researchgate.net/publication/399879336_Advancements_in_Machine_Learning-Driven_Proxy_Modelling_Techniques_for_Reservoir_Engineering_A_Systematic_Review

■ ResearchGate, 'Response Surface Methodology Approach for History Matching and Uncertainty Assessment of Reservoir Simulation Models' https://www.researchgate.net/publication/254527939_Response_Surface_Methodology_Approach_for_History_Matching_and_Uncertainty_Assessment_of_Reservoir_Simulation_Models

■ Academia.edu, 'Development of proxy models for petroleum reservoir simulation: a systematic literature review and state-of-the-art' https://www.academia.edu/44346638/Development_of_proxy_models_for_petroleum_reservoir_simulation_a_systematic_literature_review_and_state-of-the-art

■ SBC, 'Predicting oil field production using the Random Forest algorithm' https://sol.sbc.org.br/index.php/sibgrapi_estendido/article/download/23277/23104/

List of References (Sources) (2/4)

■ GitHub, vassilikitsios/snapshot_pod_rom_py, 'Python code to calculate proper orthogonal decomposition modes' https://github.com/vassilikitsios/snapshot_pod_rom_py

■ ScienceDirect, 'Neural operator-based proxy for reservoir simulations considering varying well settings, locations, and permeability fields' https://www.sciencedirect.com/science/article/abs/pii/S0098300424003091

■ neuraloperator(GitHub official), 'Learning in infinite dimension with neural operators' https://github.com/neuraloperator/neuraloperator

■ neuraloperator documentation, 'Fourier Neural Operators' https://neuraloperator.github.io/dev/theory_guide/fno.html

■ Viana, F.A.C., 'A Tutorial on Latin Hypercube Design of Experiments', Quality and Reliability Engineering International, Vol.32, No.5(2016) https://onlinelibrary.wiley.com/doi/10.1002/qre.1924

■ GitHub, clicumu/pyDOE2, 'Design of experiments for Python' https://github.com/clicumu/pyDOE2

■ SPE Annual Technical Conference and Exhibition, 'Experimental Design or Monte Carlo Simulation? Strategies for Building Robust Surrogate Models' https://onepetro.org/SPEATCE/proceedings-abstract/15ATCE/15ATCE/D031S030R005/180367

List of References (Sources) (3/4)

- scikit-learn, 'Gaussian Processes' https://scikit-learn.org/stable/modules/gaussian_process.html
- scikit-learn, 'Cross-validation: evaluating estimator performance' https://scikit-learn.org/stable/modules/cross_validation.html
- scikit-learn, 'Tuning the hyper-parameters of an estimator' https://scikit-learn.org/stable/modules/grid_search.html
- MDPI, 'Importance of Using Modern Regression Analysis for Response Surface Models in Science and Technology' https://www.mdpi.com/2076-3417/15/13/7206
- Penn State University, STAT 503, 'Lesson 11: Response Surface Methods and Designs' https://online.stat.psu.edu/stat503/lesson/11
- Akiba, T. et al., 'Optuna: A Next-generation Hyperparameter Optimization Framework', KDD 2019, arXiv:1907.10902 https://arxiv.org/abs/1907.10902
- Towards Data Science, 'Hyperparameter Tuning of Neural Networks with Optuna and PyTorch' https://towardsdatascience.com/hyperparameter-tuning-of-neural-networks-with-optuna-and-pytorch-22e179efc837/

List of References (Sources) (4/4)

■ MachineLearningMastery.com, 'A Gentle Introduction to k-fold Cross-Validation' https://machinelearningmastery.com/k-fold-cross-validation/

■ Equinor, 'Volve field data set' https://www.equinor.com/energy/volve-data-sharing

■ SINTEF, 'spe10: Access to the SPE10 benchmark case' https://www.sintef.no/contentassets/2551f5f85547478590ceca14bc13ad51/spe10.html

■ Myers, R.H., Montgomery, D.C., Anderson-Cook, C.M., 'Response Surface Methodology: Process and Product Optimization Using Designed Experiments', 4th ed., Wiley (2016) (book, not held in-house)

■ Sacks, J., Welch, W.J., Mitchell, T.J., Wynn, H.P., 'Design and Analysis of Computer Experiments', Statistical Science, Vol.4, No.4 (1989) (paper, not held in-house)

■ Mohaghegh, S.D., 'Data-Driven Reservoir Modeling', SPE/PennWell(2017, ISBN 9781613995600) https://store.spe.org/Data-Driven-Reservoir-Modeling-P1054.aspx

■ Planning Department (internal document) 'NewCourseProposal_DataDrivenSimulator.md' (July 2026)

List of References That Must Be Purchased

■ Myers, R.H., Montgomery, D.C., Anderson-Cook, C.M., 'Response Surface Methodology: Process and Product Optimization Using Designed Experiments' (4th ed., Wiley, 2016; book) — Needed for: primary-source support for the mathematical details and specific testing procedures in Part B 'B6: Fitting Procedure for Polynomial Regression' and 'B7: Validating the RSM Model (ANOVA, Lack-of-Fit Test)'

■ Sacks, J., Welch, W.J., Mitchell, T.J., Wynn, H.P., 'Design and Analysis of Computer Experiments', Statistical Science, Vol.4, No.4 (1989) (paper, paywalled) — Needed for: citing the theoretical foundations (equations for the covariance function and prediction variance) in Part B 'B8: Basics of Kriging (Gaussian Process Regression)'

■ Viana, F.A.C., 'A Tutorial on Latin Hypercube Design of Experiments', Quality and Reliability Engineering International (Wiley, 2016) (paper, subscription required) — Needed for: citing the algorithm details and concrete examples in Part B 'B4: Concrete Steps (2): Latin Hypercube Sampling (LHS)' (currently limited to an abstract-level introduction)

■ Mohaghegh, S.D., 'Data-Driven Reservoir Modeling' (SPE/PennWell, 2017, ISBN 9781613995600; book) — Needed for: support when concretizing the overall machine-learning model-building workflow and exercise design of Parts C and F against a systematic practitioner's