GeospaNN:

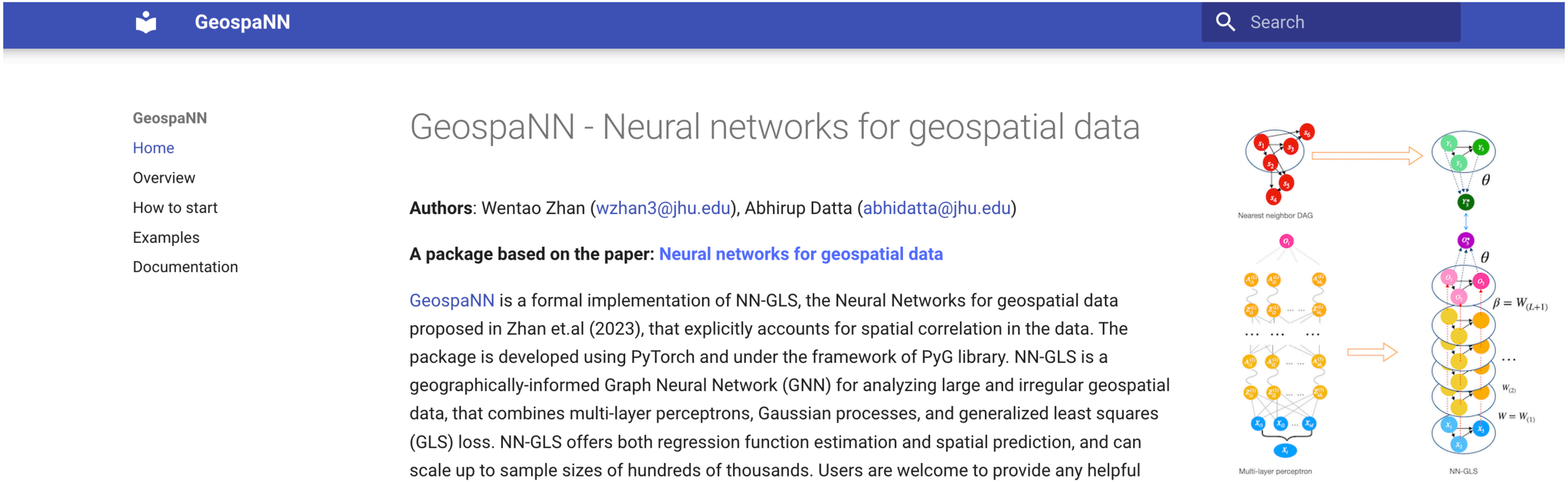
Neural Networks for Geospatial data

With running examples

Wentao Zhan

GeospaNN

Stands for: Geospatial Neural Networks.



Pypi: https://pypi.org/project/geospaNN/

Outline

1. **Basic features**
2. Simulation
3. Real data example

Start point: what we have?

Point process:

- Data: $(Y_i, X_i, s_i) \ : i = 1, \ldots, n$
  - Y_i : scalar response
  - X_i : d-dimensional covariate
  - s_i : location

precipitation	temperature	air pressure	relative humidity	U-wind	V-wind	PM 2.5	longitude
0.008044	0.362296	0.887664	0.774197	0.868530	0.781498	5.020834	0.980311
0.005516	0.355305	0.882153	0.751742	0.864206	0.770715	3.837500	0.983093
0.000000	0.335323	0.928359	0.714189	0.697080	0.813224	2.041666	0.974238
0.000000	0.338579	0.954218	0.690767	0.625266	0.868161	3.669444	0.976951
0.002528	0.293827	0.893599	0.685830	0.688808	0.842395	1.020833	0.945193
...	...	...	...	...	...	...	...
0.393932	0.901079	0.972022	0.910279	0.642436	0.469410	5.168750	0.469337
0.000689	0.810329	0.897414	0.539392	0.553265	0.464926	6.041666	0.436702

Simulate data: input

```
def f1(X): return 10 * np.sin(np.pi * 2 * X)

p = 1;
funXY = f1

n = 1000
nn = 20
range = [0,1]

X, Y, coord, cov, corerr = geospaNN.Simulation(n, p, nn, funXY, theta, range=range)
```

$$Y_i(s) = f\big(X_i(s)\big) + w(s) + \epsilon(s)$$

- "p": Dimension of the input.
- "funXY": 1-D function $f : X \in R^p \rightarrow Y \in R$
  - $Y = f(X) = 10\sin(2\pi X),\ X \in R$
  - $Y = f(X) = \frac{1}{6}\left(10\sin(\pi X_1 X_2) + 20(X_3 - 0.5)^2 + 20X_4 + 5X_5\right),\ X \in [0,1]^5$

Simulate data: geospaNN.Simulation

```
def f1(X): return 10 * np.sin(np.pi * 2 * X)

p = 1;
funXY = f1

n = 1000
nn = 20
range = [0,1]

+

sigma = 1
phi = 0.3
tau = 0.01
theta = torch.tensor([sigma, phi / np.sqrt(2), tau])

X, Y, coord, cov, corerr = geospaNN.Simulation(n, p, nn, funXY, theta, range=range)
```

$$Y_i(s) = X_i(s)\beta + w(s) + \epsilon(s)$$

- “n”: Number of spatial locations.
- “nn”: Number of nearest neighbors used for NNGP approximation. 20 recommended.
- “theta”: Spatial parameters in $Cov(s,t) = \sigma^2(\exp(-\phi\,|\,s-t\,|\,) + \tau I(s=t))$
- “range”: Spatial coordinates are sampled randomly from range x range.

Simulate data: geospaNN.Simulation

```
def f1(X): return 10 * np.sin(np.pi * 2 * X)

p = 1;
funXY = f1

n = 1000                    sigma = 1
nn = 20                     phi = 0.3
range = [0,1]               tau = 0.01
                            theta = torch.tensor([sigma, phi / np.sqrt(2), tau])
X, Y, coord, cov, corerr = geospaNN.Simulation(n, p, nn, funXY, theta, range=range)
```

$$Y_i(s) = f\big(X_i(s)\big) + w(s) + \epsilon(s)$$

"X": X(s), non-spatial covariates sampled from [0,1]

"Y": Y(s), response

"coord": s, spatial coordinates

"cov": covariance matrix

"corerr": $w(s) + \epsilon(s)$, correlated effect (error) term

Simulate data: geospaNN.Simulation

What if we want $X(s)$ to also have spatial distribution?

1: Simulate $X(s)$ as a spatial term.

```
torch.manual_seed(2025)
_, _, _, _, X = geospaNN.Simulation(n, p, nn, funXY, torch.tensor([1, 5, 0.01]), range=[0, 1])
X = X.reshape(-1,p)
X = (X - X.min())/(X.max() - X.min())
```

2: Simulate spatial coordinates and true spatial effect.

```
torch.manual_seed(2025)
_, _, coord, cov, corerr = geospaNN.Simulation(n, p, nn, funXY, theta, range=[0, 1])
```

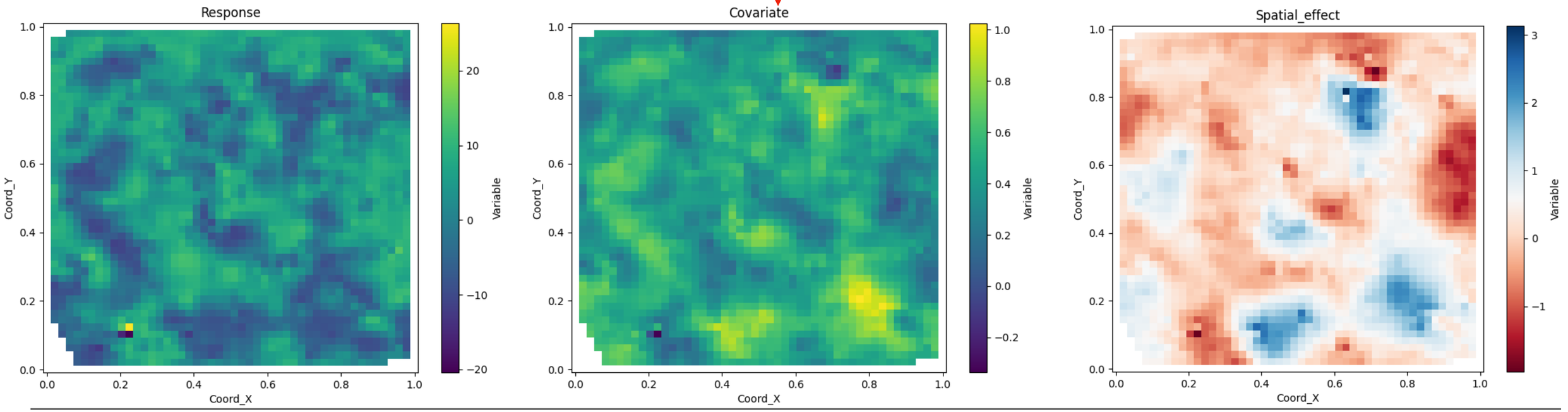
3: Compose the response.

```
Y = funXY(X).reshape(-1) + corerr
```

Visualize data: geospaNN.spatial_plot_surface

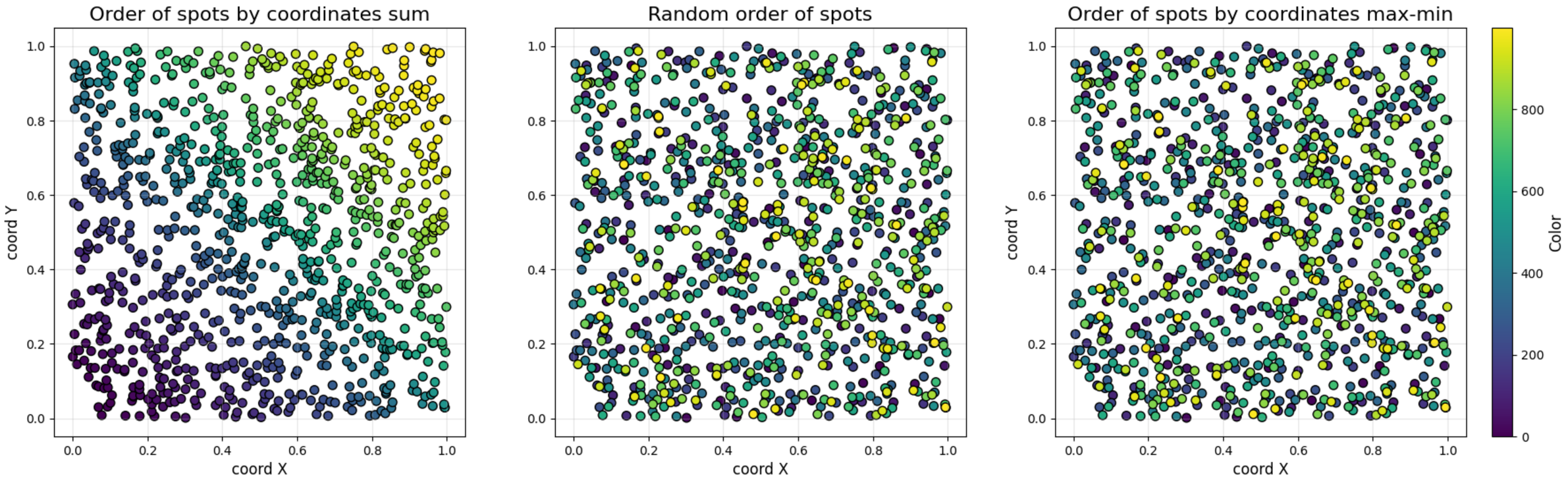
```
dict = {"Covariate": X, "Response": Y, "Spatial_effect":corerr}

for index, (name, variable) in enumerate(dict.items()):
    geospaNN.spatial_plot_surface(variable.detach().numpy().reshape(-1), coord.detach().numpy(),
                                  grid_resolution = 50, method = "CloughTocher",
                                  title = name, save_path = path, file_name = name + ".png")
```

$$Y(s) = f(X(s)) + w(s) + \epsilon(s)$$



Spatial ordering

```
X, Y, coord, _ = geospaNN.spatial_order(X, Y, coord, method='max-min')
```



Spatial data loader: geospaNN.make_graph

```
data = geospaNN.make_graph(X, Y, coord, nn, Ind_list = None)
```

Data loader provides an efficient way to iterate over a dataset.

geospaNN.make_graph generates data loader object including:

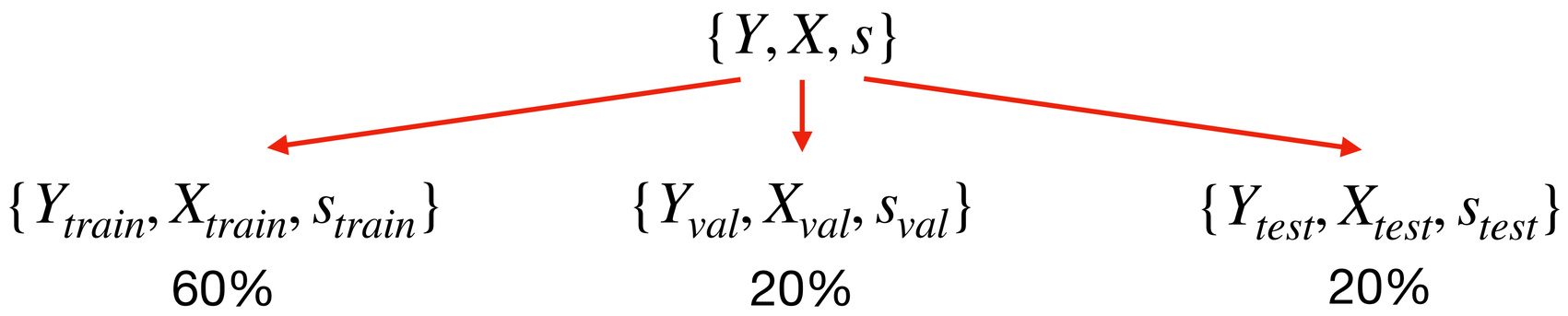
- X, Y, s in spatial modeling.
- Edge index and attributions connecting nearest neighbors.
- Allow users to pass predefined $n \times k$, k neighbor index matrix.
- Objects can be called by data.x, data.y, data.pos, ……
- Key function.

Training-testing split: geospaNN.split_data

```
torch.manual_seed(2024)
np.random.seed(0)
data_train, data_val, data_test = geospaNN.split_data(X, Y, coord, neighbor_size=nn, test_proportion=0.2)
```

geospaNN.split_data generates three data objects.

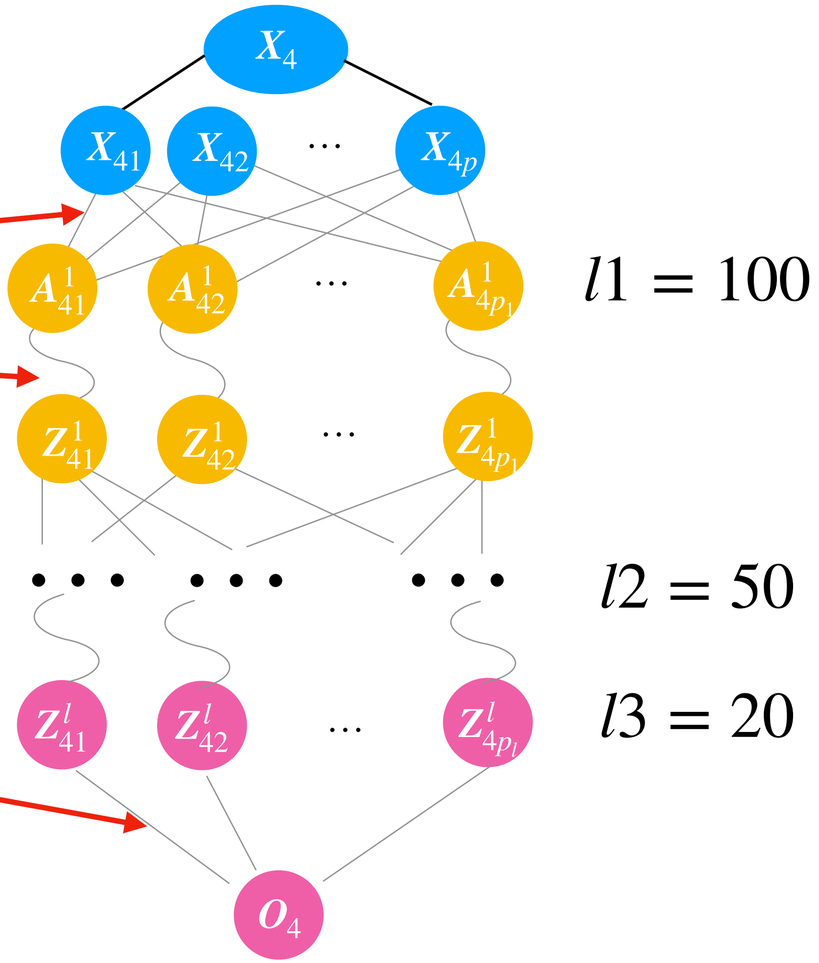
It's a wrapper for:




```
data_train = make_graph(X_train, Y_train, coord_train, neighbor_size)
data_val = make_graph(X_val, Y_val, coord_val, neighbor_size)
data_test = make_graph(X_test, Y_test, coord_test, neighbor_size)
```

Model training: ordinary neural networks

```
mlp_nn = torch.nn.Sequential(
    torch.nn.Linear(p, 100),
    torch.nn.ReLU(),
    torch.nn.Linear(100, 50),
    torch.nn.ReLU(),
    torch.nn.Linear(50, 20),
    torch.nn.ReLU(),
    torch.nn.Linear(20, 1),
)
```



Model training: ordinary neural networks

```
mlp_nn = torch.nn.Sequential(
    torch.nn.Linear(p, 100),
    torch.nn.ReLU(),
    torch.nn.Linear(100, 50),
    torch.nn.ReLU(),
    torch.nn.Linear(50, 20),
    torch.nn.ReLU(),
    torch.nn.Linear(20, 1),
)
trainer_nn = geospaNN.nn_train(mlp_nn, lr=0.01, min_delta=0.001)
training_log = trainer_nn.train(data_train, data_val, data_test, seed = 2025)
```

- “mlp_nn”: multi-layer perceptron (mlp) architecture
- “nn_model”: object for training process and hyper parameters.
  - “lr”: learning rate.
  - “min_delta”: cutoff for “significant update”.
- “nn_model.train”: A wrapper for the common training loop.

Model training: NN-GLS

```
mlp_nngls = torch.nn.Sequential(
    torch.nn.Linear(p, 100),
    torch.nn.ReLU(),
    torch.nn.Linear(100, 50),
    torch.nn.ReLU(),
    torch.nn.Linear(50, 20),
    torch.nn.ReLU(),
    torch.nn.Linear(20, 1),
)
model = geospaNN.nngls(p=p, neighbor_size=nn, coord_dimensions=2, mlp=mlp_nngls,
                       theta=torch.tensor(theta0))
trainer_nngls = geospaNN.nngls_train(model, lr=0.1, min_delta=0.001)
training_log = trainer_nngls.train(data_train, data_val, data_test, epoch_num= 200,
                                   Update_init=10, Update_step=2, seed = 2025)
```

$$Cov\big(w(s_1), w(s_2)\big) = C(s_1, s_2 \mid \theta) = \sigma^2\big(\exp(-\phi \mid s_1 - s_2 \mid) + \tau^2 I(s_1 = s_2)\big)$$

- Spatial parameters are initialized and updated in training.
  - “Update_init” is the initial epoch starting updating.
  - “Update_step” is the gap of epochs between updates.
- Everything else than θ are the same to NN.

How θ get updated?

```
torch.manual_seed(2025)
_, _, coord_simp, _, corerr_simp = geospaNN.Simulation(n, p, nn, funXY,
                                                       torch.tensor([1,1.5,0.01]), range=[0, 10])
theta_hat = geospaNN.theta_update(torch.tensor([2, 3, 0.05]), corerr_simp, coord_simp, neighbor_size=20)
print(theta_hat)
```

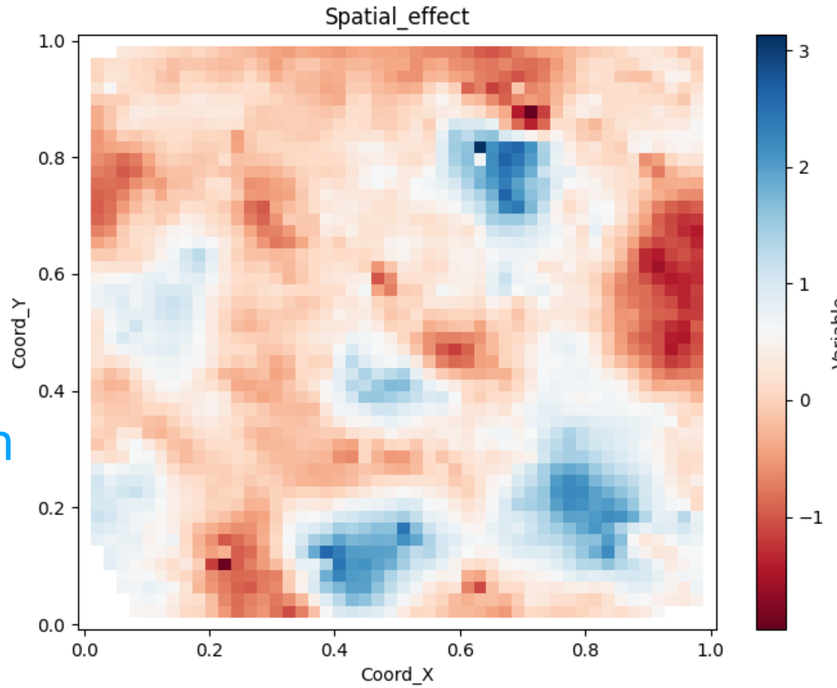
```
Theta updated from
[2.   3.   0.05]


[0.88942914 1.74742522 0.01030131]
```

- geospaNN.theta_update currently call the **BRISC** R-package (Saha, & Datta, 2018) for likelihood-based parameter estimation.
- geospaNN.linear_GLS, as wrapper of BRISC_estimation(), can be called to solve β and θ in SPLMM:

$$Y_i(s) = X_i(s)\beta + w(s) + \epsilon(s), w(s) \sim C(\cdot, \cdot | \theta)$$

How θ get updated?

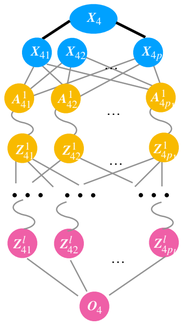
θ → Simulation → geospaNN.Simulation



Maximum likelihood → $\hat{\theta}$ geospaNN.theta_update

How θ get updated in NN-GLS?

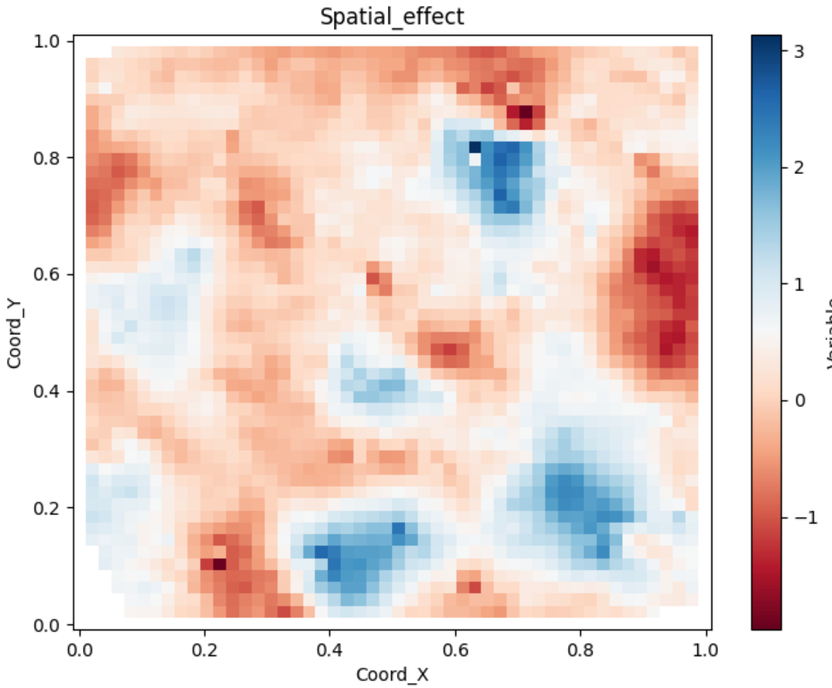
```
theta0 = geospaNN.theta_update(mlp_nn(data_train.x).squeeze() - data_train.y,
                               data_train.pos, neighbor_size=20)
```



Get Residual

$\hat{f}$

$\hat{f}(X) - Y$



Maximum likelihood

$\hat{\theta}$

geospaNN.theta_update

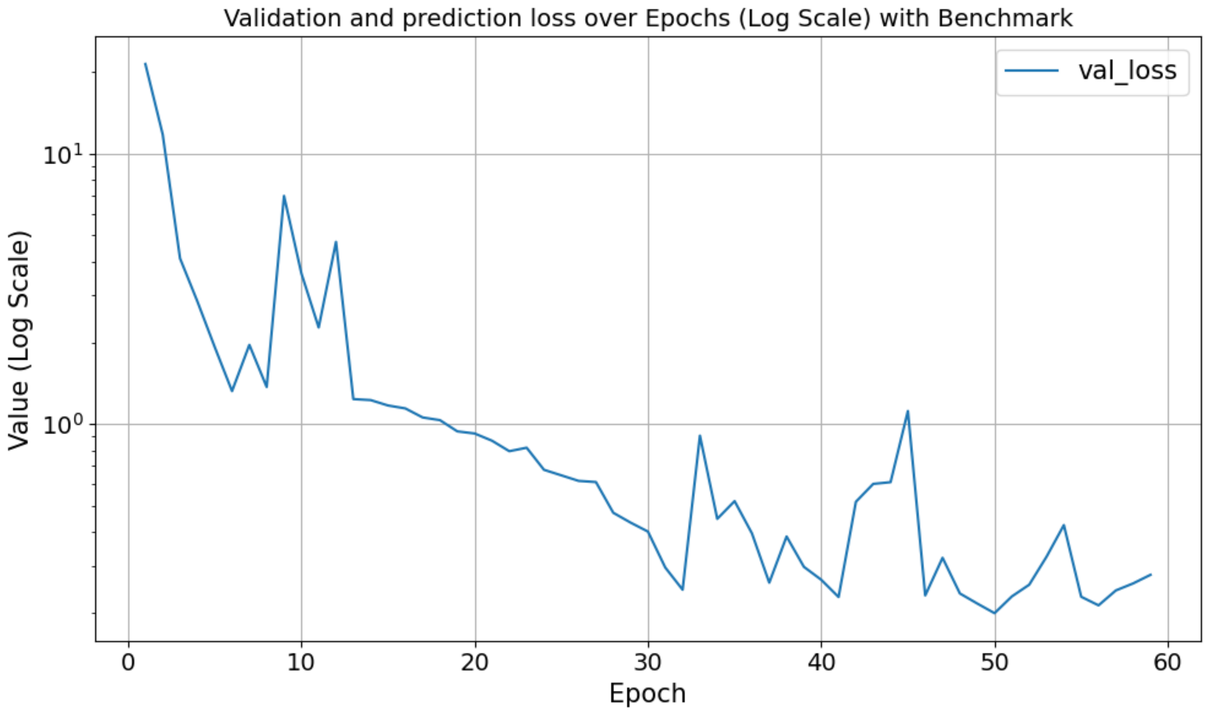
Back-propagation

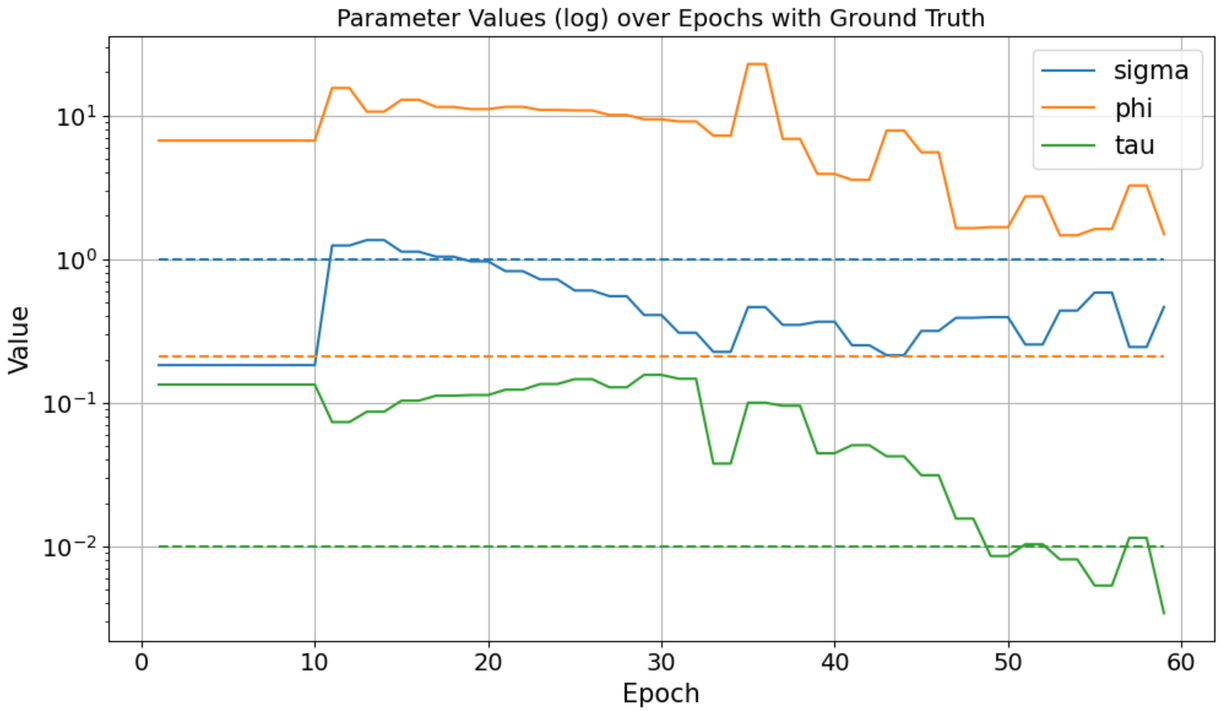
geospaNN.nngls_train.train

Training output

Training curves of validation loss and spatial parameters.

```
geospaNN.plot_log(training_log, theta, path)
```



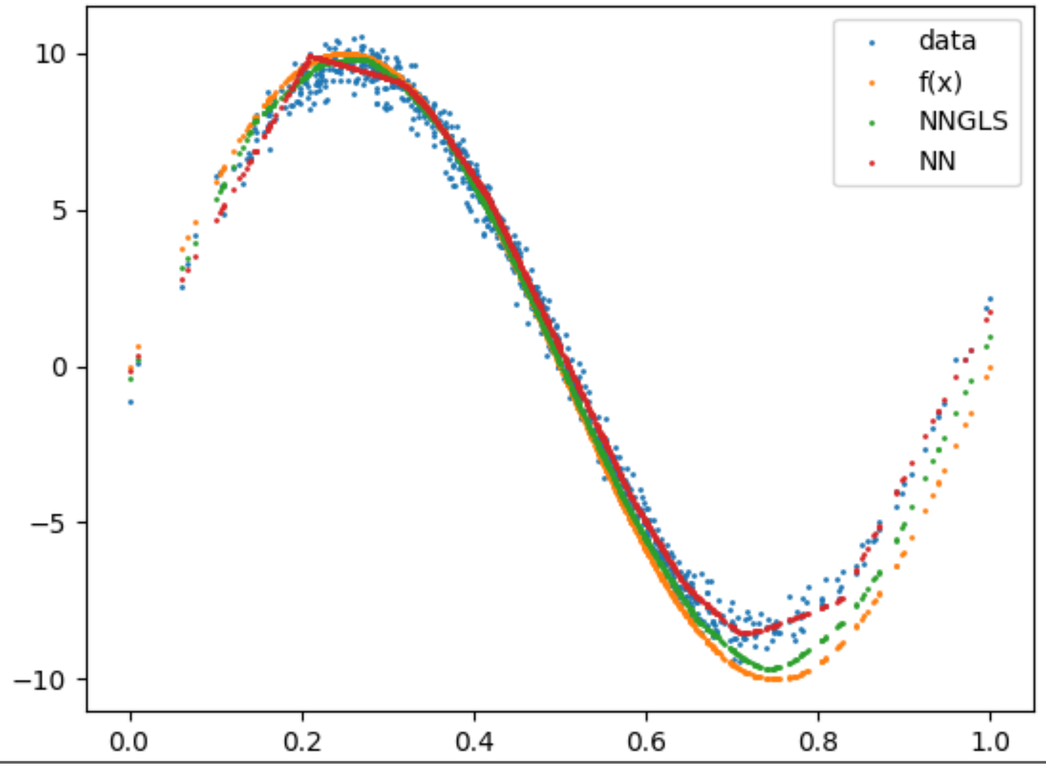


Estimation: geospaNN.nngls.estimate

```
model = geospaNN.nngls(p=p, neighbor_size=nn, coord_dimensions=2, mlp=mlp_nngls,
                       theta=torch.tensor(theta0))
```

Equivalent to “model.mlp(X)”.

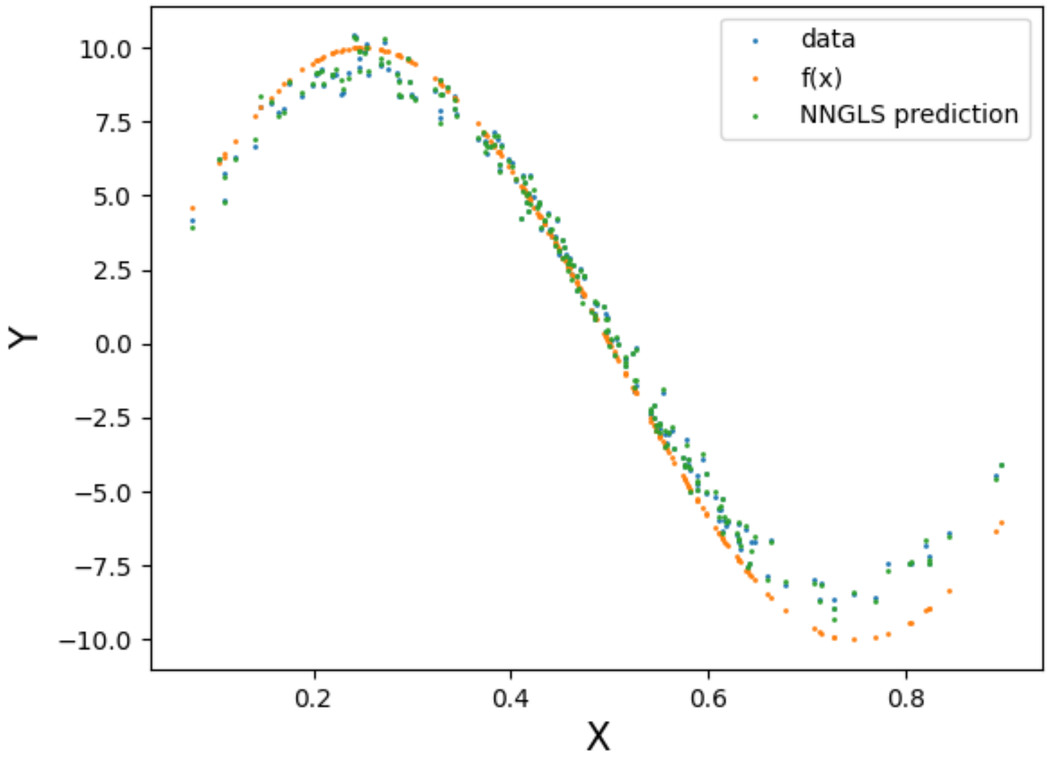
```
estimate = model.estimate(X)
```

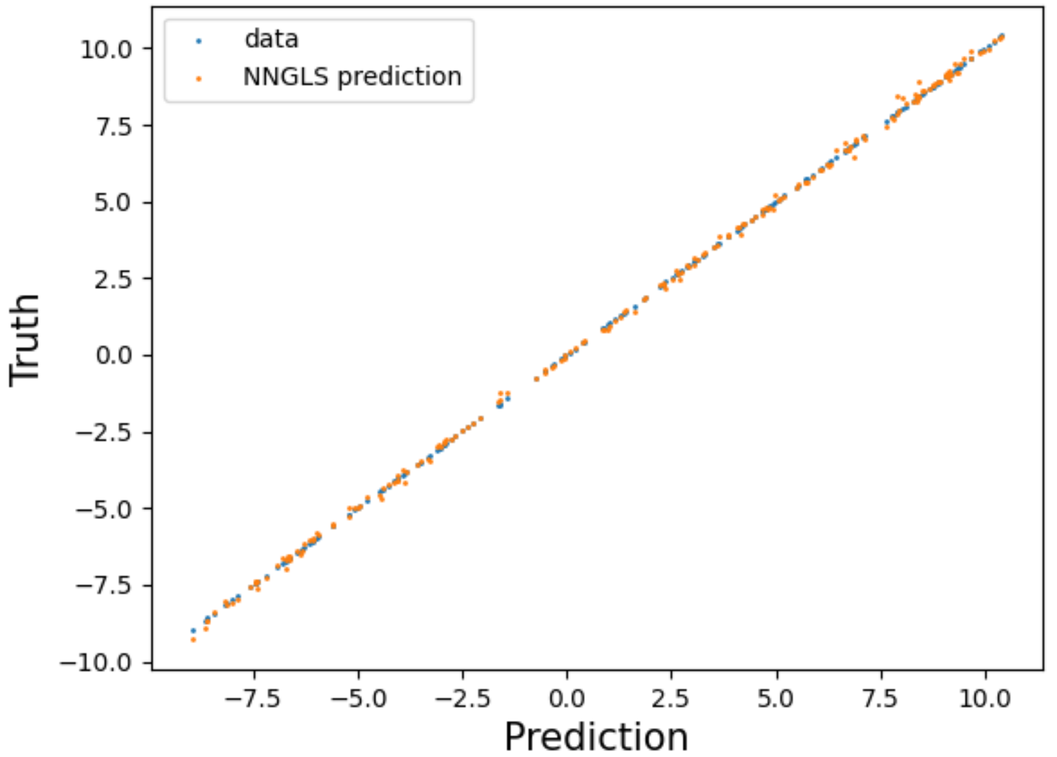


Prediction: geospaNN.nngls.predict

Efficient prediction through nearest neighbor kriging:

```
test_predict = model.predict(data_train, data_test)
```

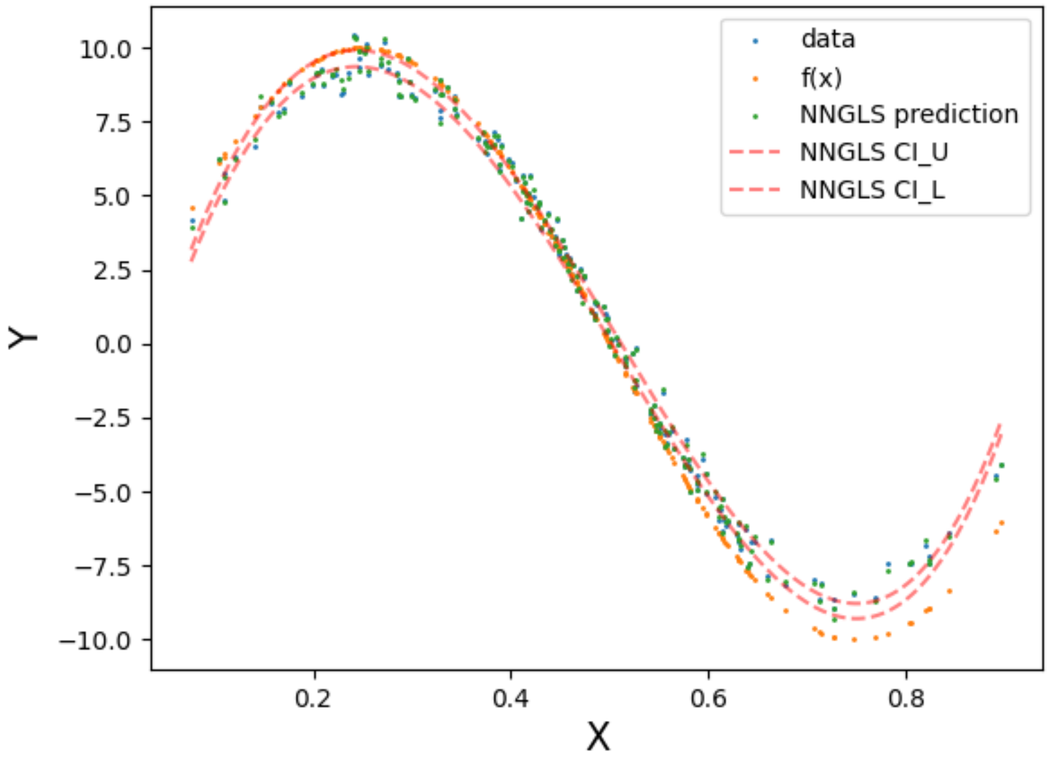


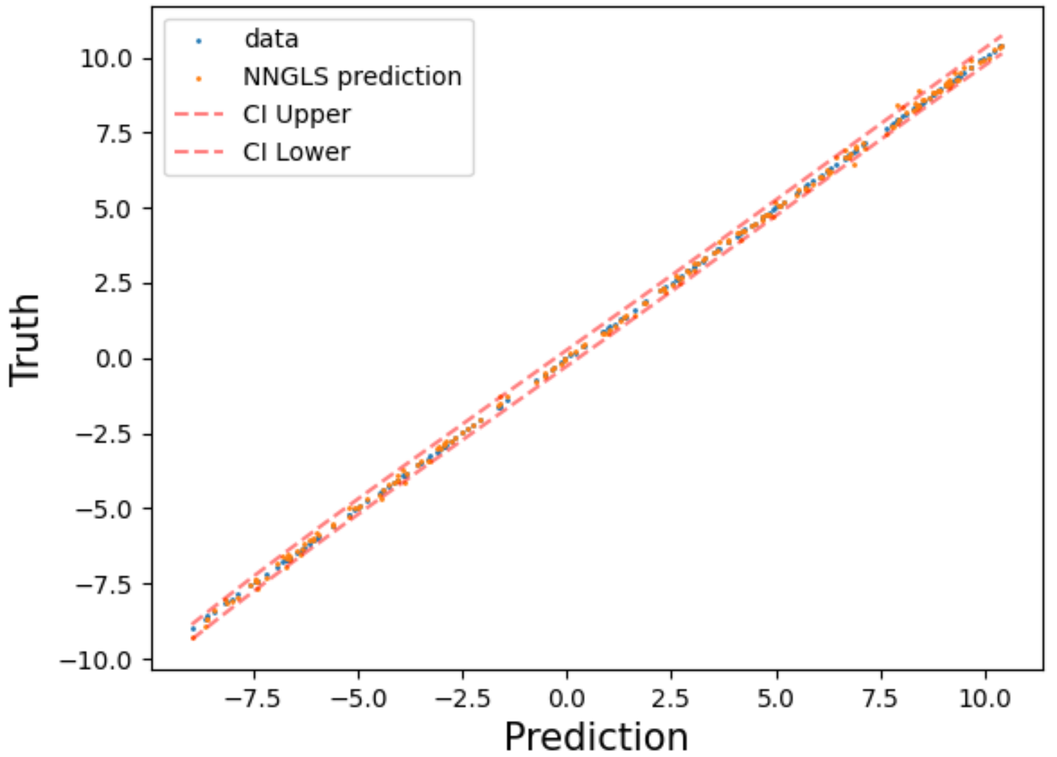


Prediction: geospaNN.nngls.predict

Efficient prediction through nearest neighbor kriging:

```
[test_predict, test_CI_U, test_CI_L] = model.predict(data_train, data_test, CI = True)
```

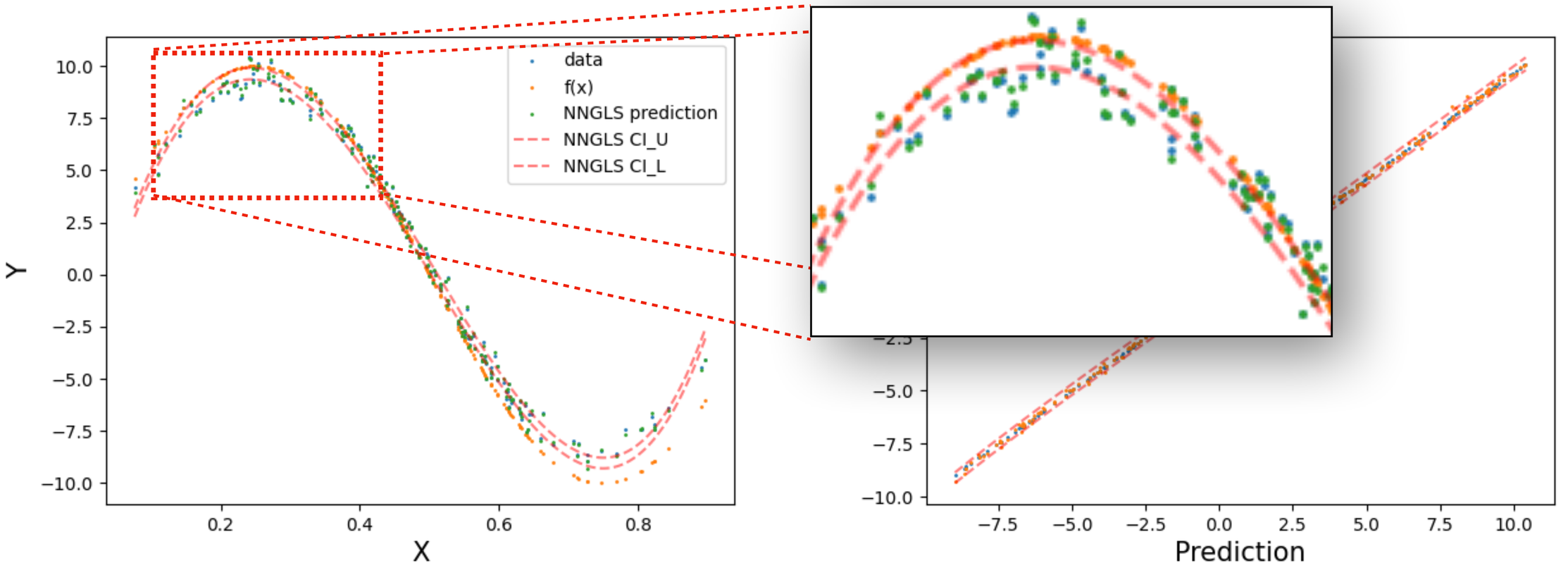




Prediction: geospaNN.nngls.predict

Efficient prediction through nearest neighbor kriging:

```
[test_predict, test_CI_U, test_CI_L] = model.predict(data_train, data_test, CI = True)
```



PDP (Partial Dependency Plot)

Partial Dependence plot shows the dependence between the target function $f(X_1, \cdots, X_p)$ and a set of individual features X_i.

$$PD(f, X_i) = \int f(X_1, \cdots, X_p) P(X_{\_i}) dX_{\_i}$$

Example: Friedman's function:

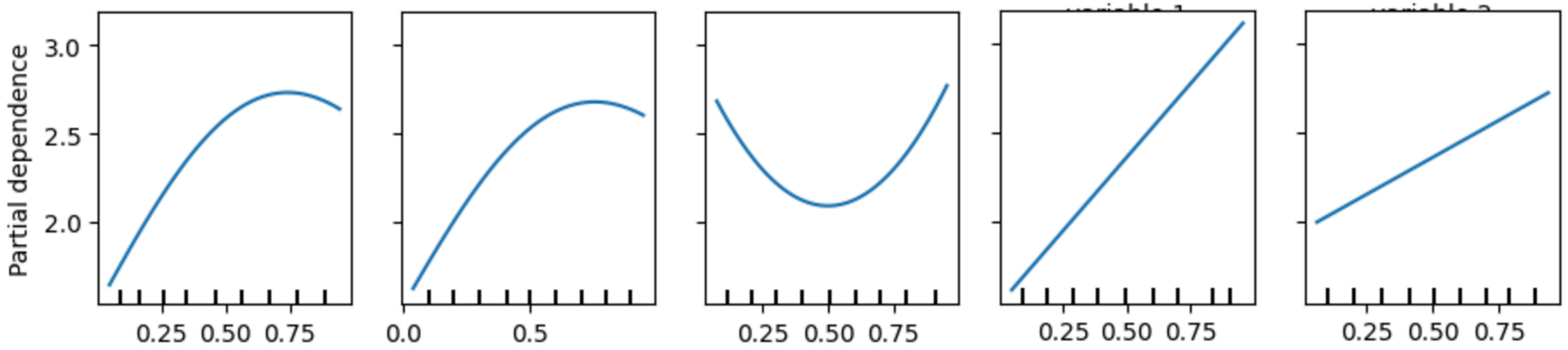
$$Y(X) = \left(10 \sin(\pi X_1 X_2) + 20(X_3 - 0.5)^2 + 10X_4 + 5X_5\right)/6$$

PDP: geospaNN.plot_PDP

Example: Friedman's function:

$$Y(X) = \left(10\sin(\pi X_1 X_2) + 20(X_3 - 0.5)^2 + 10X_4 + 5X_5\right)/6$$

```
def f5(X): return (10 * np.sin(np.pi * X[:, 0] * X[:, 1]) + 20 * (X[:, 2] - 0.5) ** 2 +
                   10 * X[:, 3] + 5 * X[:, 4]) / 6

PDP_truth = geospaNN.visualize.plot_PDP(funXY, X, names = ["PDP"], save_path = path, save = True)
```

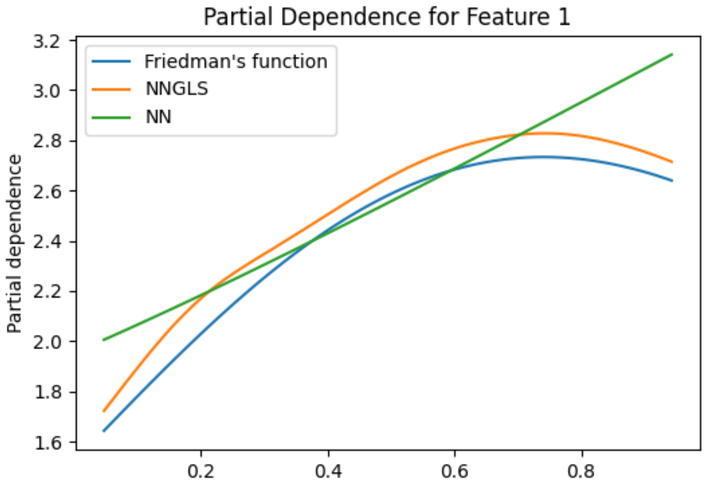


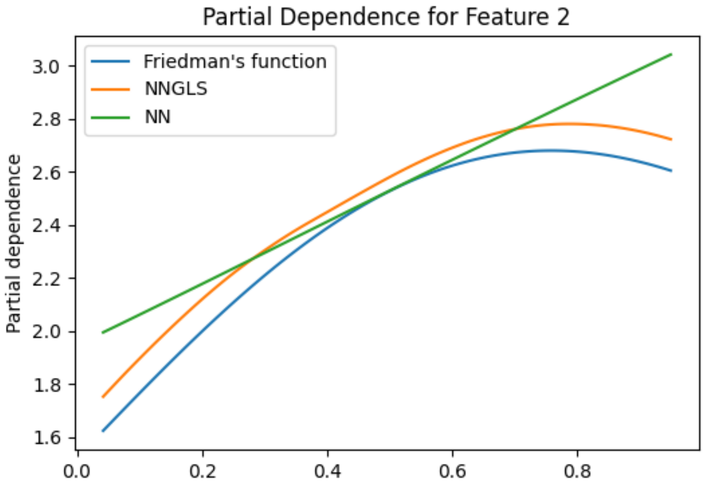
PDP: geospaNN.plot_PDP_list

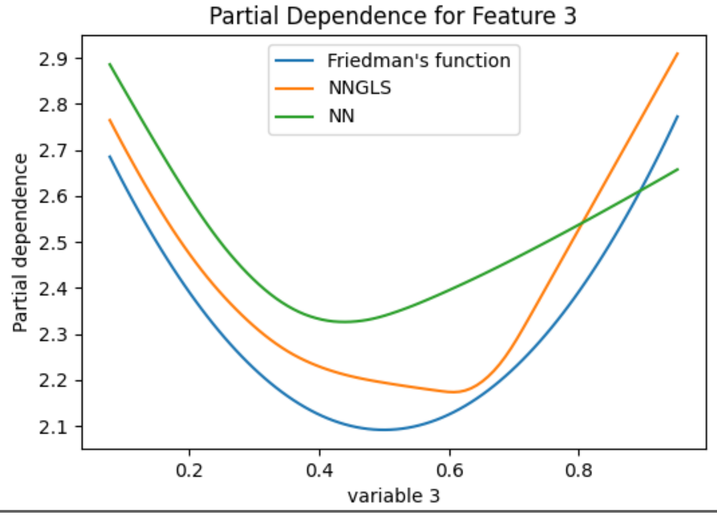
```
geospaNN.visualize.plot_PDP_list([funXY, mlp_nngls, mlp_nn], ['Friedmans function', 'NNGLS', 'NN'], X, split = True)
```

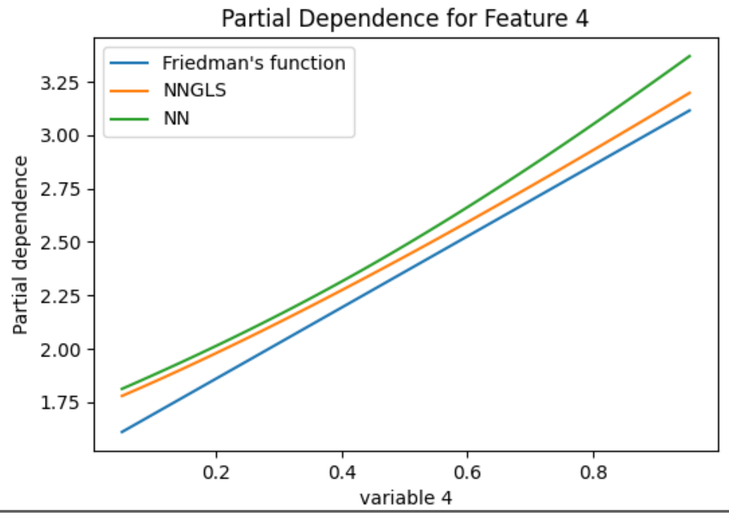
Friedman's function:

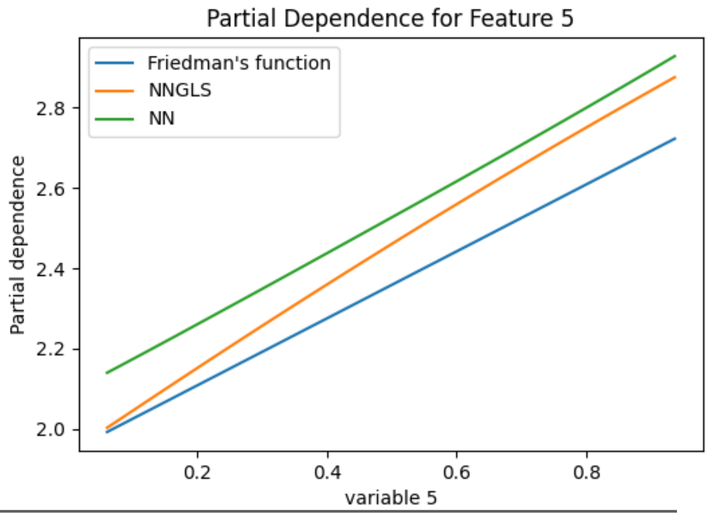
$$Y(X) = \big(10\sin(\pi X_1 X_2) + 20(X_3 - 0.5)^2 + 10X_4 + 5X_5\big)/6$$











Outline

1. Basic functions
2. **Simulation examples**
    A. **General Architecture design**
    B. NNGLS handles complex interaction
    C. NNGLS vs add-covariate approaches
3. Real data example

Architecture design

```
mlp_nngls = torch.nn.Sequential(
    torch.nn.Linear(p, 100),
    torch.nn.ReLU(),
    torch.nn.Linear(100, 50),
    torch.nn.ReLU(),
    torch.nn.Linear(50, 20),
    torch.nn.ReLU(),
    torch.nn.Linear(20, 1),
)
```

```
mlp_nngls = torch.nn.Sequential(
    torch.nn.Linear(p, 5),
    torch.nn.ReLU(),
    torch.nn.Linear(5, 1)
)
```

```
mlp_nngls = torch.nn.Sequential(
    torch.nn.Linear(p, 20),
    torch.nn.ReLU(),
    torch.nn.Linear(20, 1)
)
```

```
mlp_nngls = torch.nn.Sequential(
    torch.nn.Linear(p, 5),
    torch.nn.ReLU(),
    torch.nn.Linear(5, 2),
    torch.nn.ReLU(),
    torch.nn.Linear(2, 1)
)
```

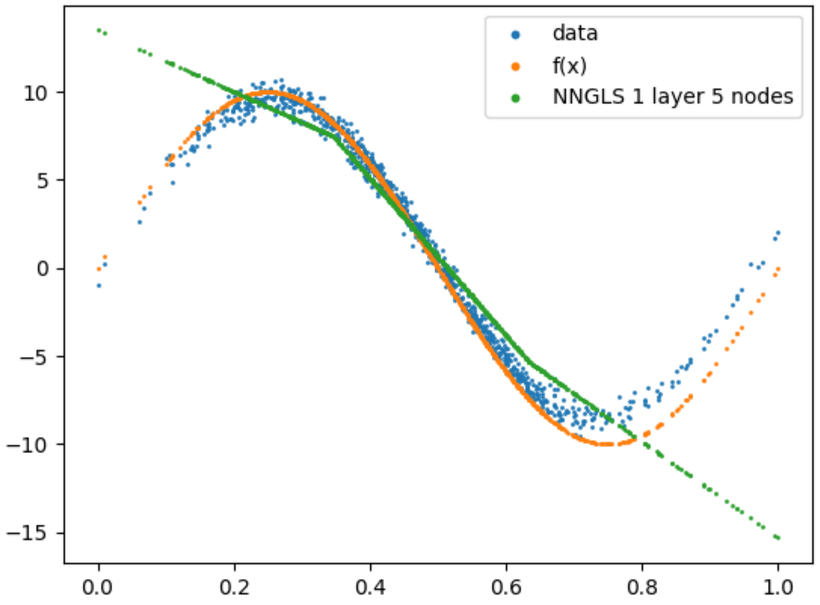
```
mlp_nngls = torch.nn.Sequential(
    torch.nn.Linear(p, 50),
    torch.nn.ReLU(),
    torch.nn.Linear(50, 20),
    torch.nn.ReLU(),
    torch.nn.Linear(20, 1)
)
```

```
model = geospaNN.nngls(p=p, neighbor_s                    lp_nn
                       theta=torch.ten
nngls_model = geospaNN.nngls_train(mode
training_log = nngls_model.train(data_train, data_val, data_test, Update_ini
```

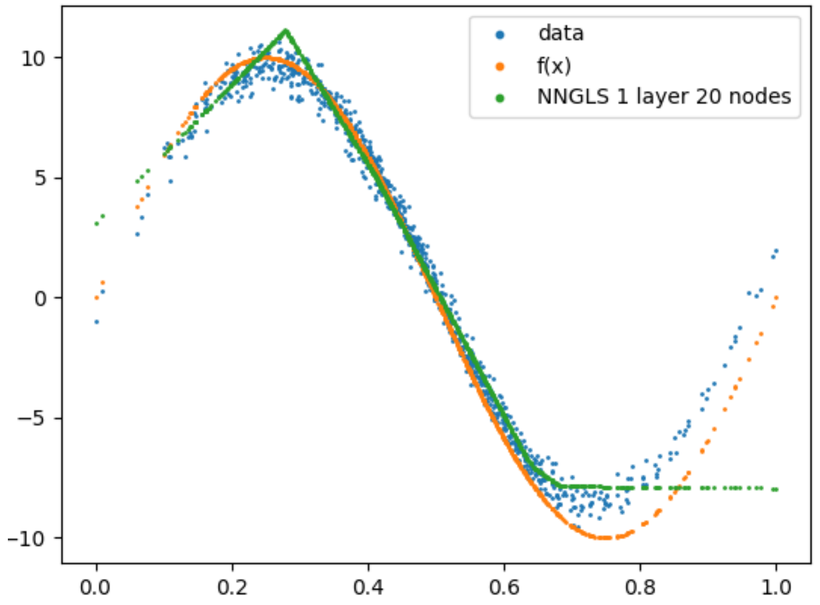
How to choose among architectures?

Architecture design: width

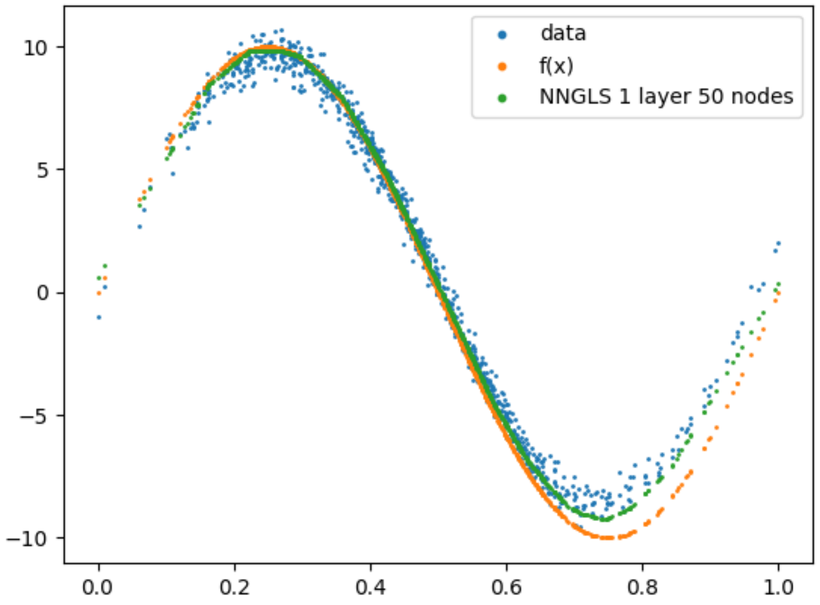
K = 5



K = 20



K = 50



Width of a layer

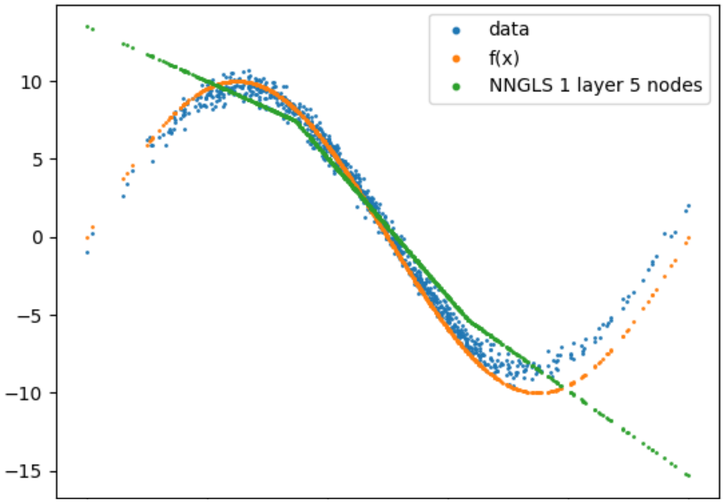
Architecture design: width & depth

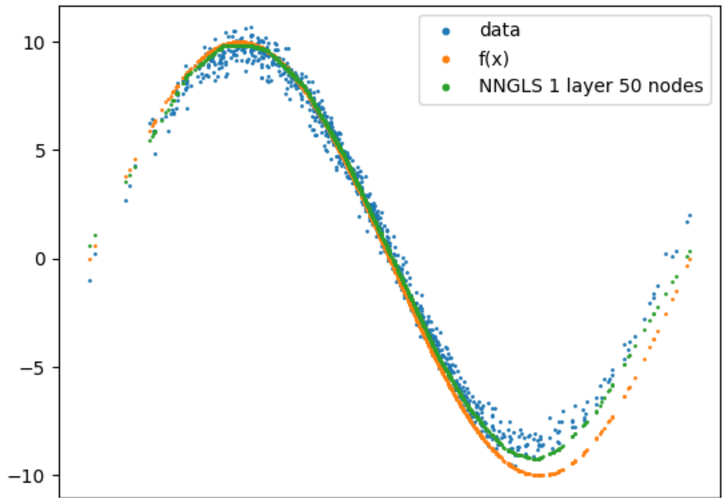
K = 5

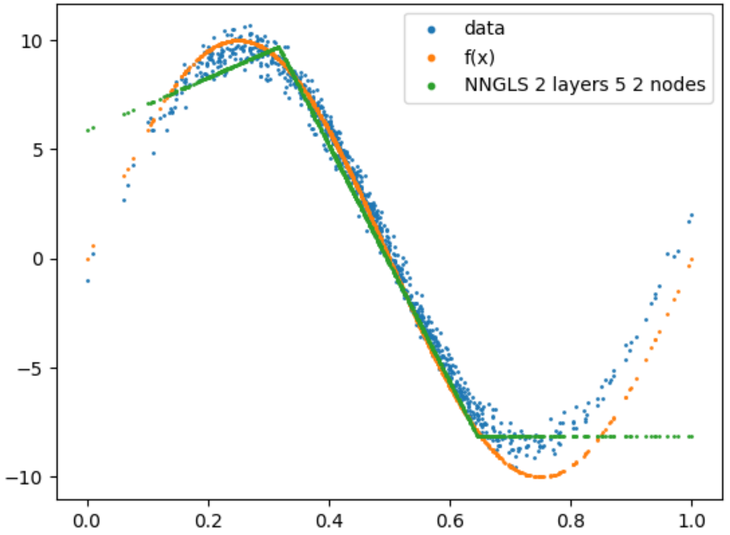
K = 50

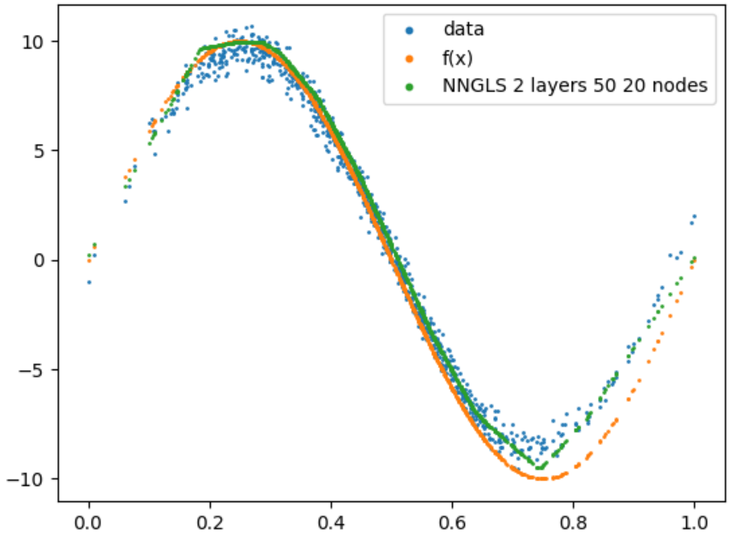
l = 1

l = 2





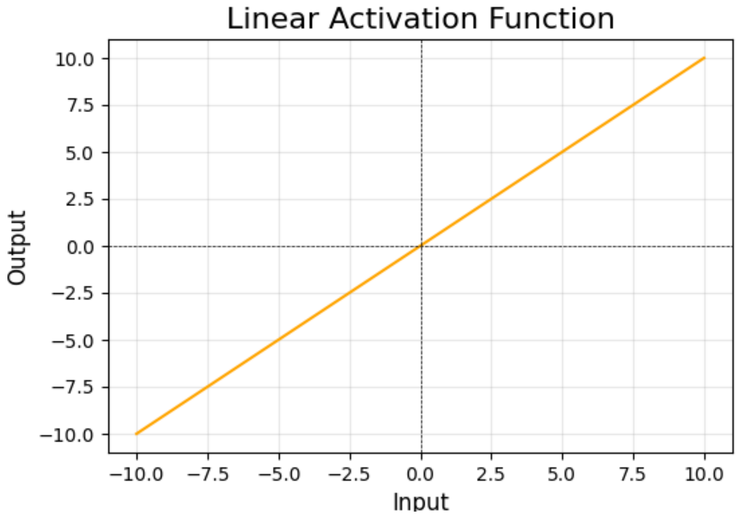




Architecture design: Activation functions

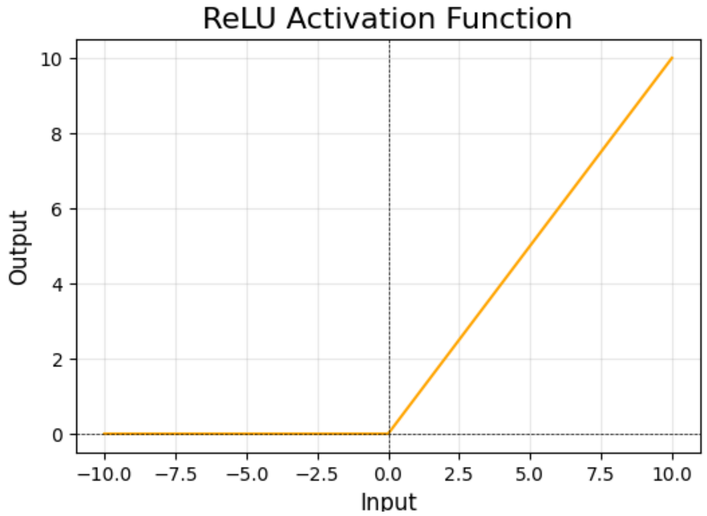
Linear:

$$Y = X$$



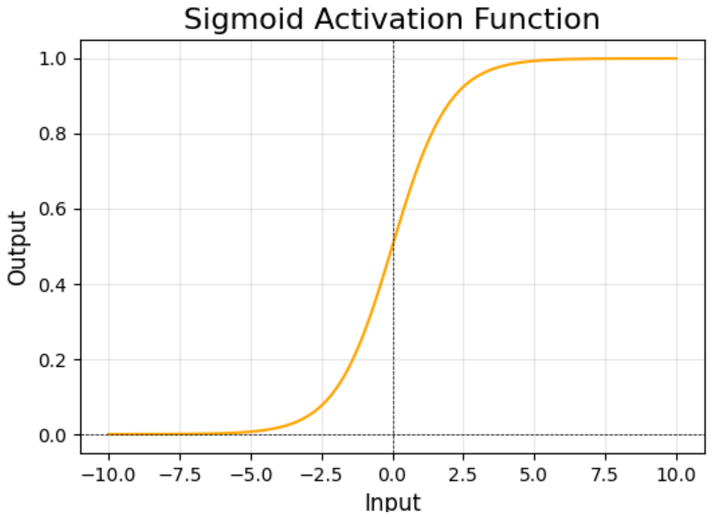
ReLu:

$$Y = X \cdot I(X > 0)$$



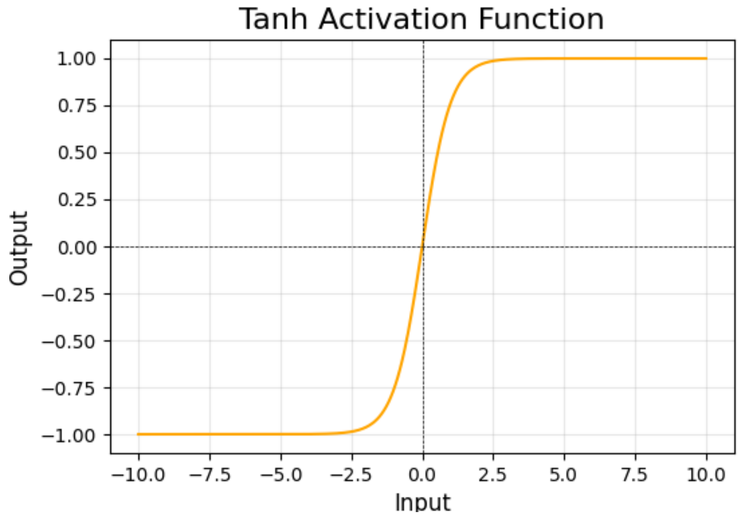
Sigmoid:

$$Y = \frac{1}{1 + e^{-x}}$$

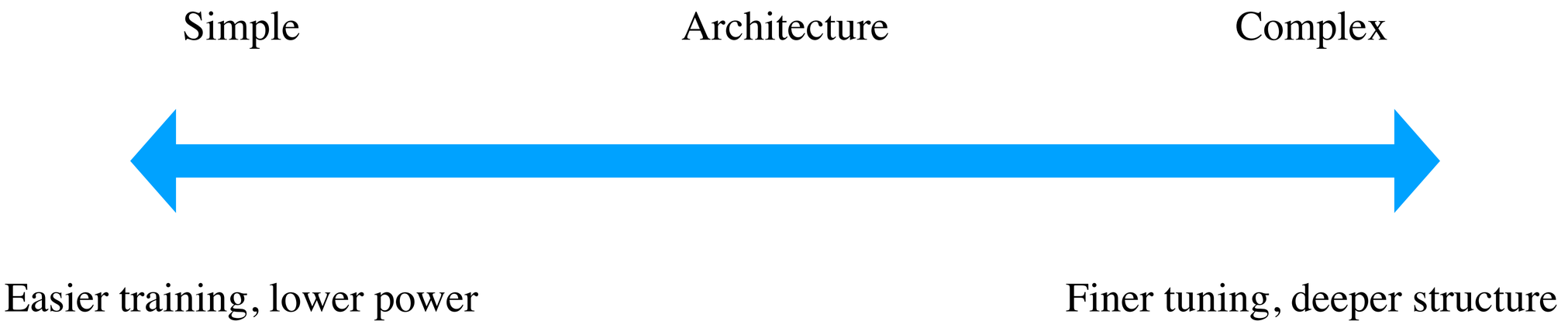


Tanh:

$$Y = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$



Architecture design



- Get a rough sense of the target function.
- Increase the number of width of the layers gradually.
- Choose non-linear activation functions properly (ReLU recommended)
- Try until no significant improvement is gained from increasing complexity.

Outline

1. Basic functions
2. **Simulation examples**
    A. General Architecture design
    B. NNGLS handles complex interaction
    C. NNGLS vs add-covariate approaches
3. Real data example

Compare with GAM

GAM (generalized additive models) is a common non-linear estimator.

$$f(X) = m_0 + \sum_{k=1}^{p} m_k(X_k)$$

Where $m_k()$'s are usually basis functions (for example B splines).

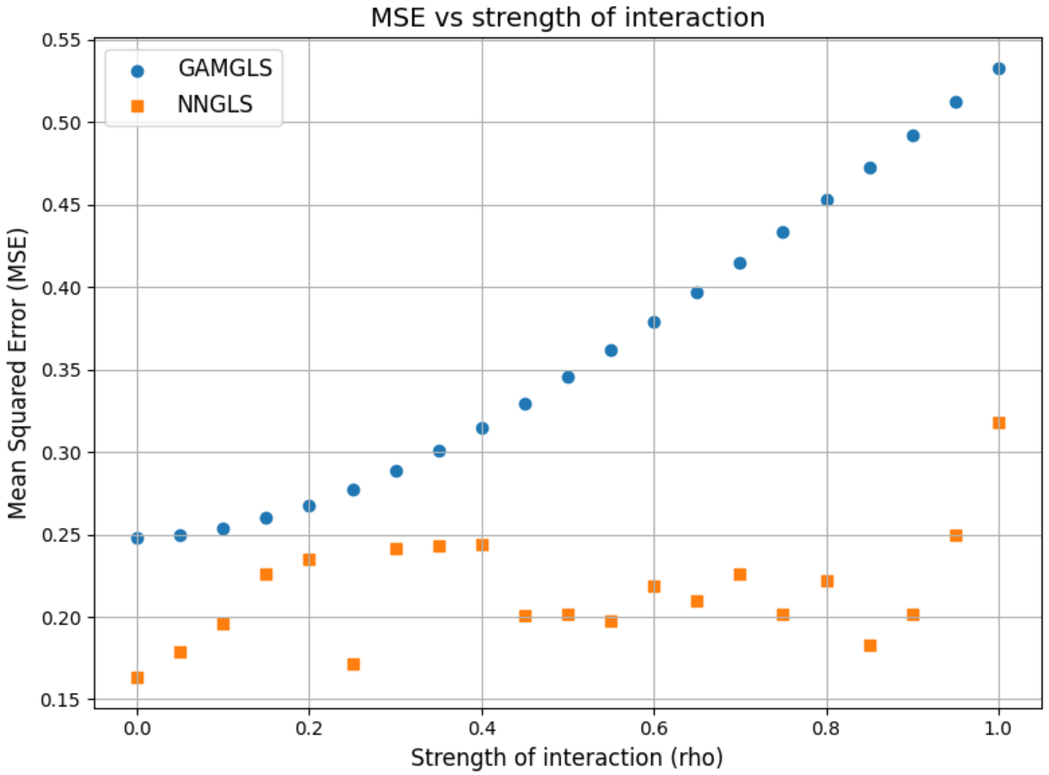
GAM assumes additive effects from $X_1, \cdots, X_p$.

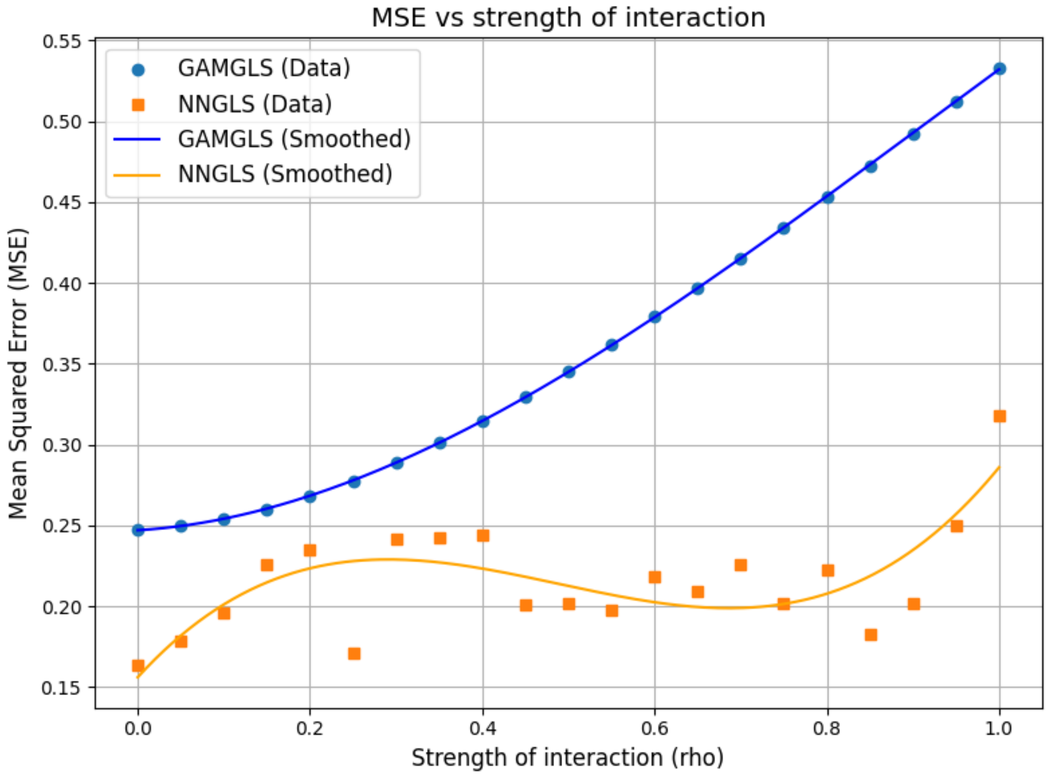
NN (NN-GLS) should outperform GAM (GLS version of GAM) by considering interaction terms.

GAM and the interaction term

$$Y(X) = \rho \frac{10 \sin(\pi X_1 X_2)}{3} + (1-\rho)\frac{20(X_3 - 0.5)^2 + 10X_4 + 5X_5)}{3}$$

Interaction





Outline

1. Basic functions
2. **Simulation examples**
    A. General Architecture design
    B. NNGLS handles complex interaction
    C. NNGLS vs added-spatial-features approaches
3. Real data example

Added-spatial-features approaches:

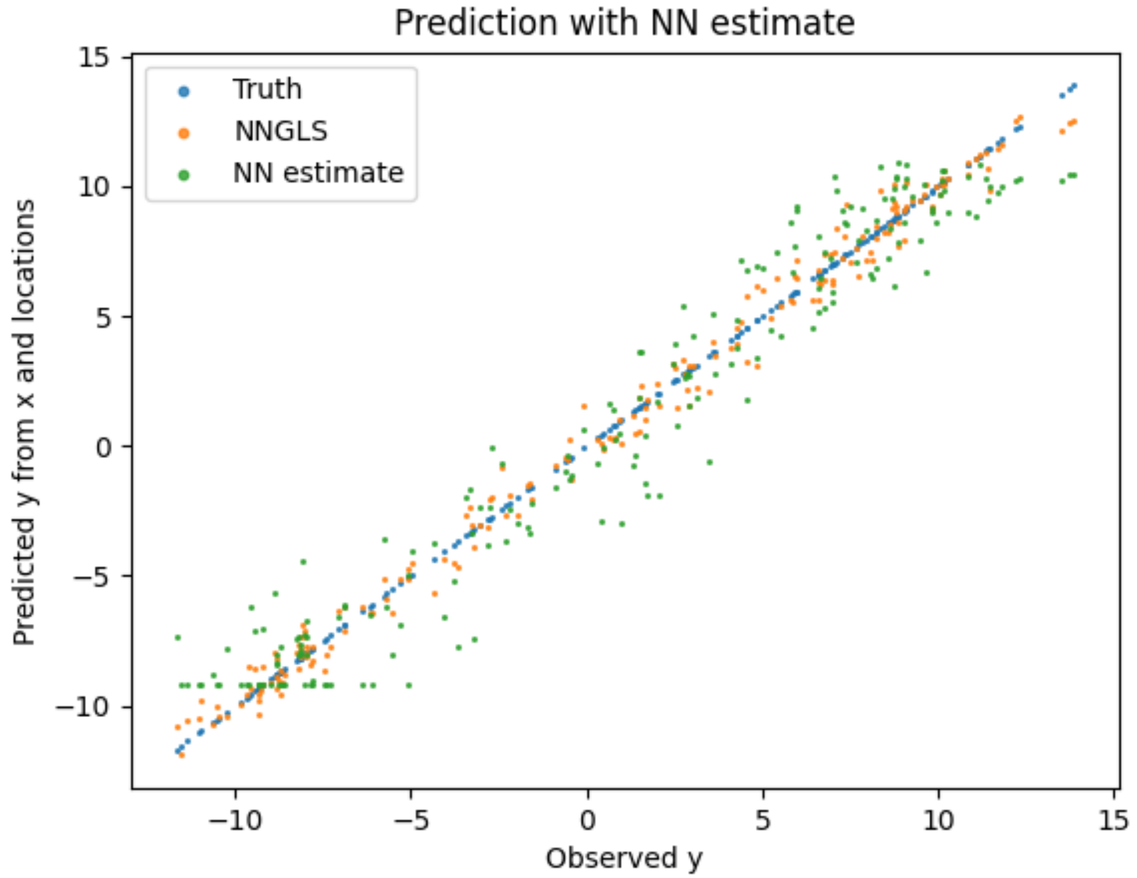
Model the spatial response $Y(X(s))$ as a fixed function of (X, s)

$$Y_i(s) = f\Big(X_i(s)\Big) + w(s) + \epsilon(s) = \tilde{f}(X_i, m(s)) + \tilde{\epsilon}(s)$$

Where $m(s)$ can be location, distance, or splines purely from s;

- $\tilde{f}(\,\cdot\,,\cdot\,)$ is used to predict at new location, but not able to separate fixed effect and spatial effect.
- Chen et.al. (2024) shows spline expansion is asymptotically equivalent to kriging (DeepKriging).

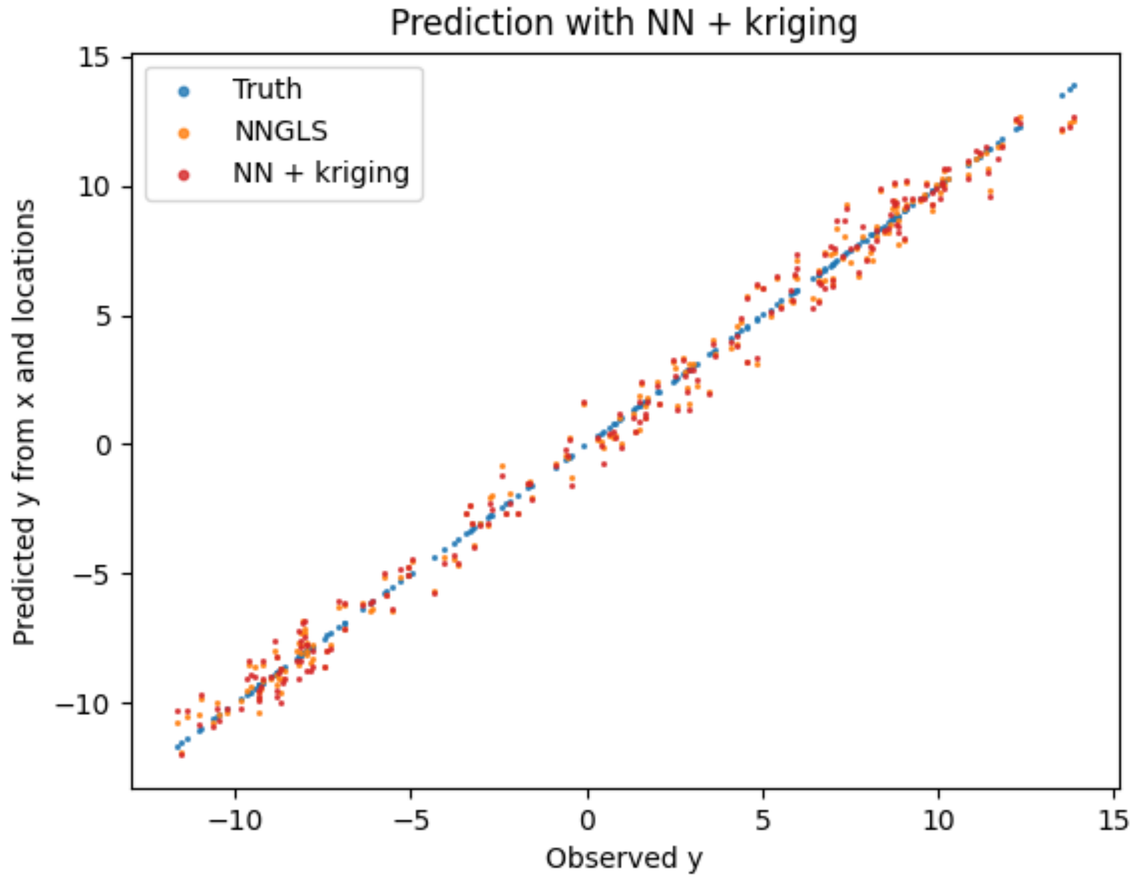
Versus added-spatial-features approaches:



Prediction vs Truth

```
RMSE nn-estimate:  2.98
RMSE nngls:  0.45
RMSE nn+kriging: 0.52
RMSE nn-add-coordinates: 2.84
RMSE nn-Deepkrig: 1.59
```

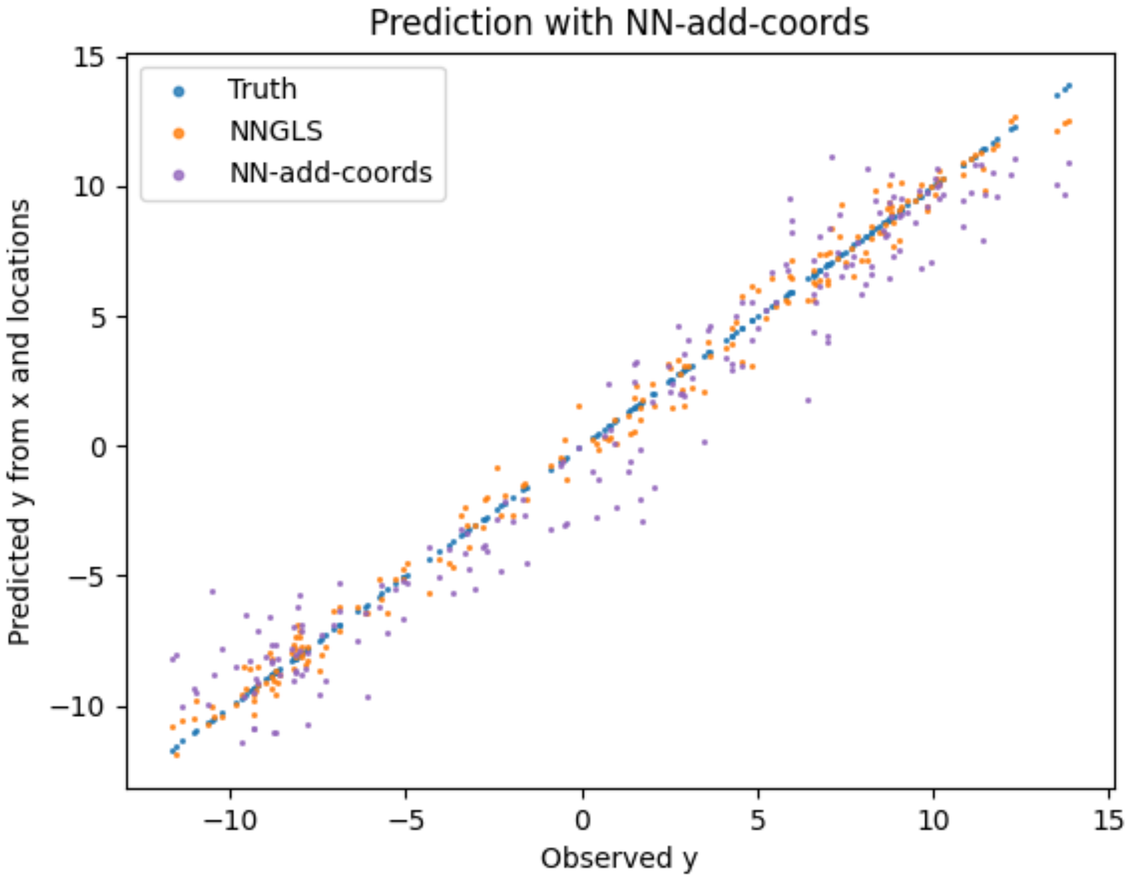
Versus added-spatial-features approaches:



Prediction vs Truth

```
RMSE nn-estimate:  2.98
RMSE nngls:  0.45
RMSE nn+kriging: 0.52
RMSE nn-add-coordinates: 2.84
RMSE nn-Deepkrig: 1.59
```

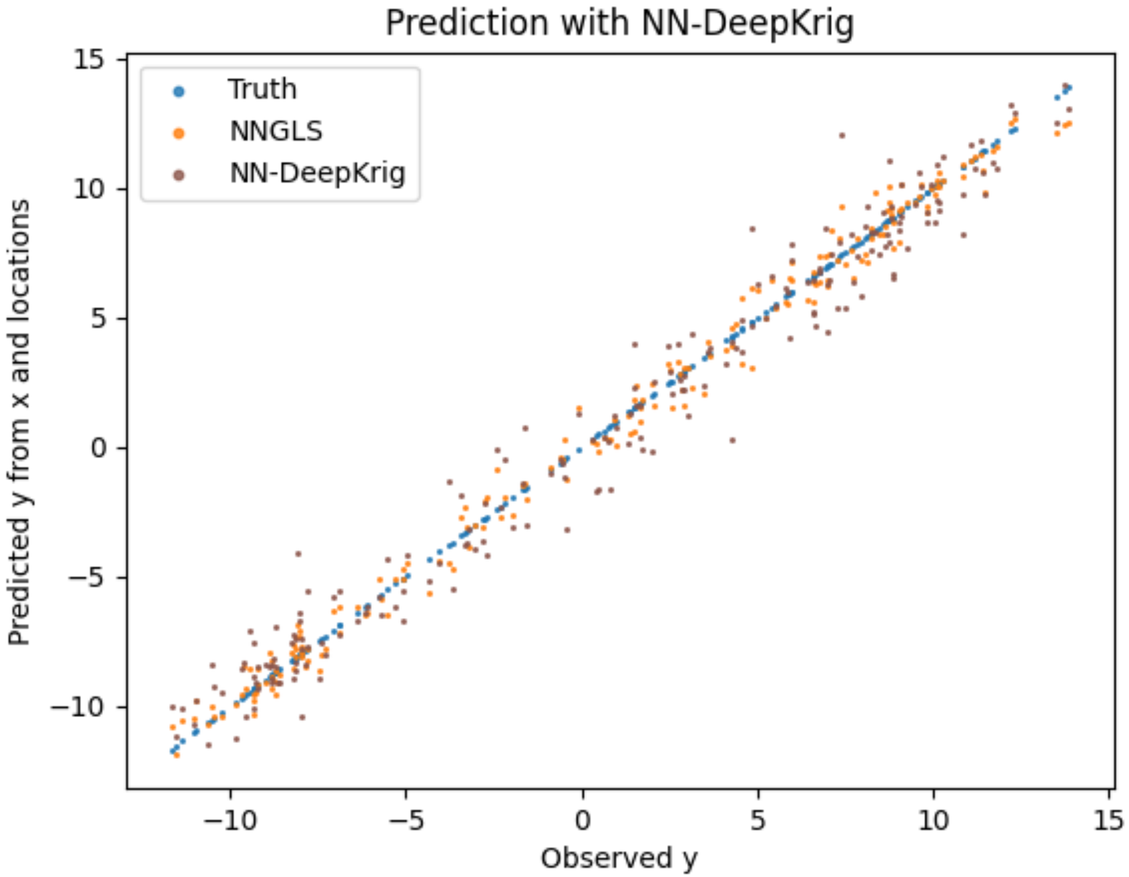
Versus added-spatial-features approaches:



Prediction vs Truth

```
RMSE nn-estimate:  2.98
RMSE nngls:  0.45
RMSE nn+kriging: 0.52
RMSE nn-add-coordinates: 2.84
RMSE nn-Deepkrig: 1.59
```

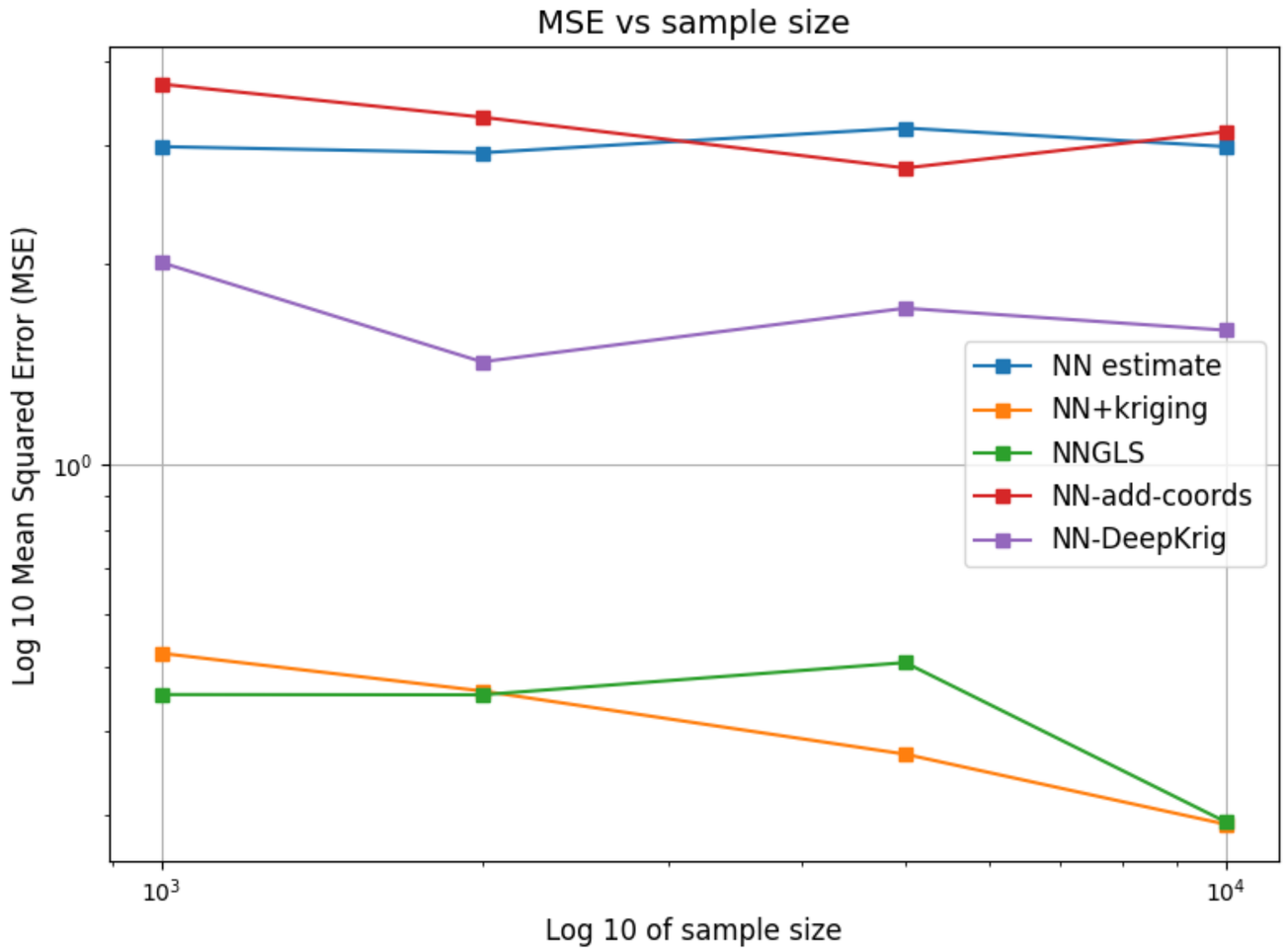
Versus added-spatial-features approaches:



Prediction vs Truth

```
RMSE nn-estimate:  2.98
RMSE nngls:  0.45
RMSE nn+kriging: 0.52
RMSE nn-add-coordinates: 2.84
RMSE nn-Deepkrig: 1.59
```

Versus added-spatial-features approaches:

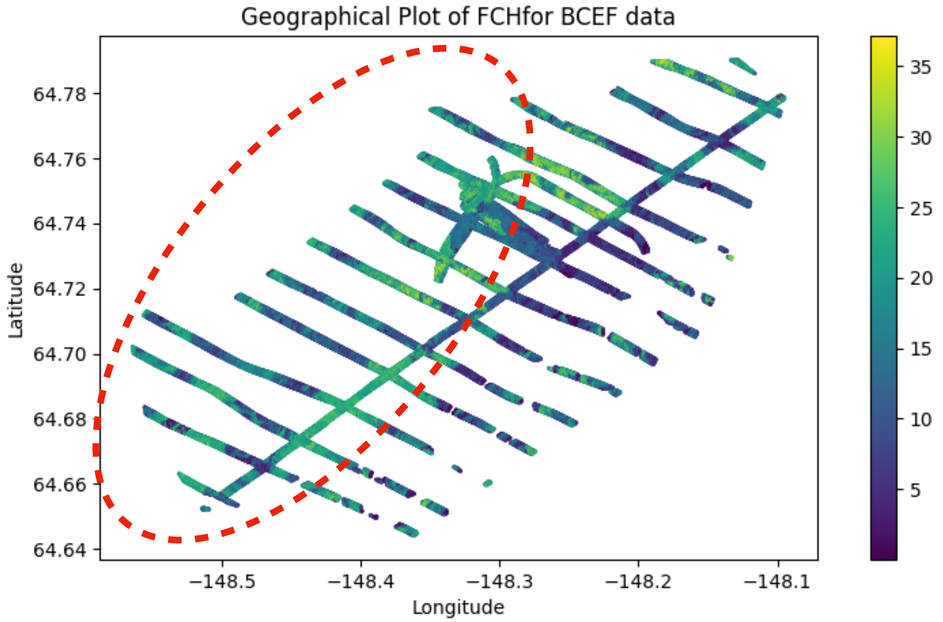


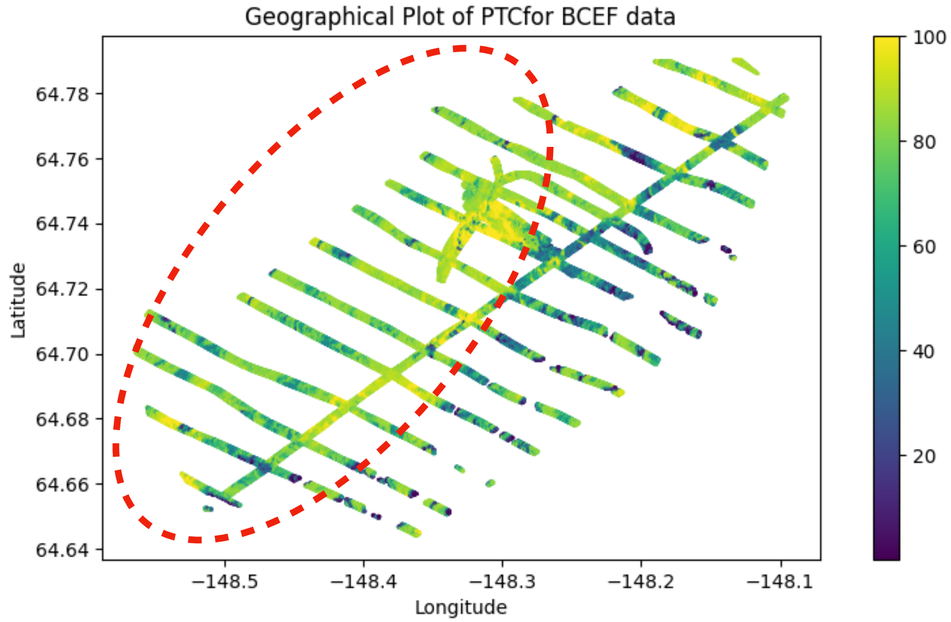
Prediction performance agains sample size

Outline

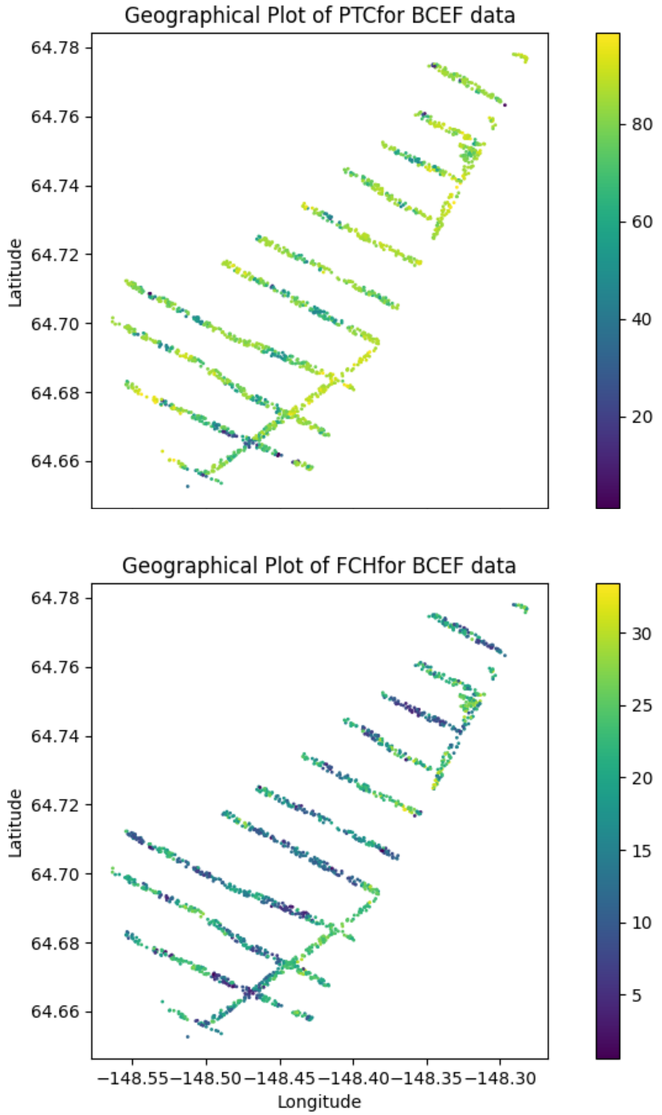
1. Basic functions
2. Simulation
3. **Real data example**

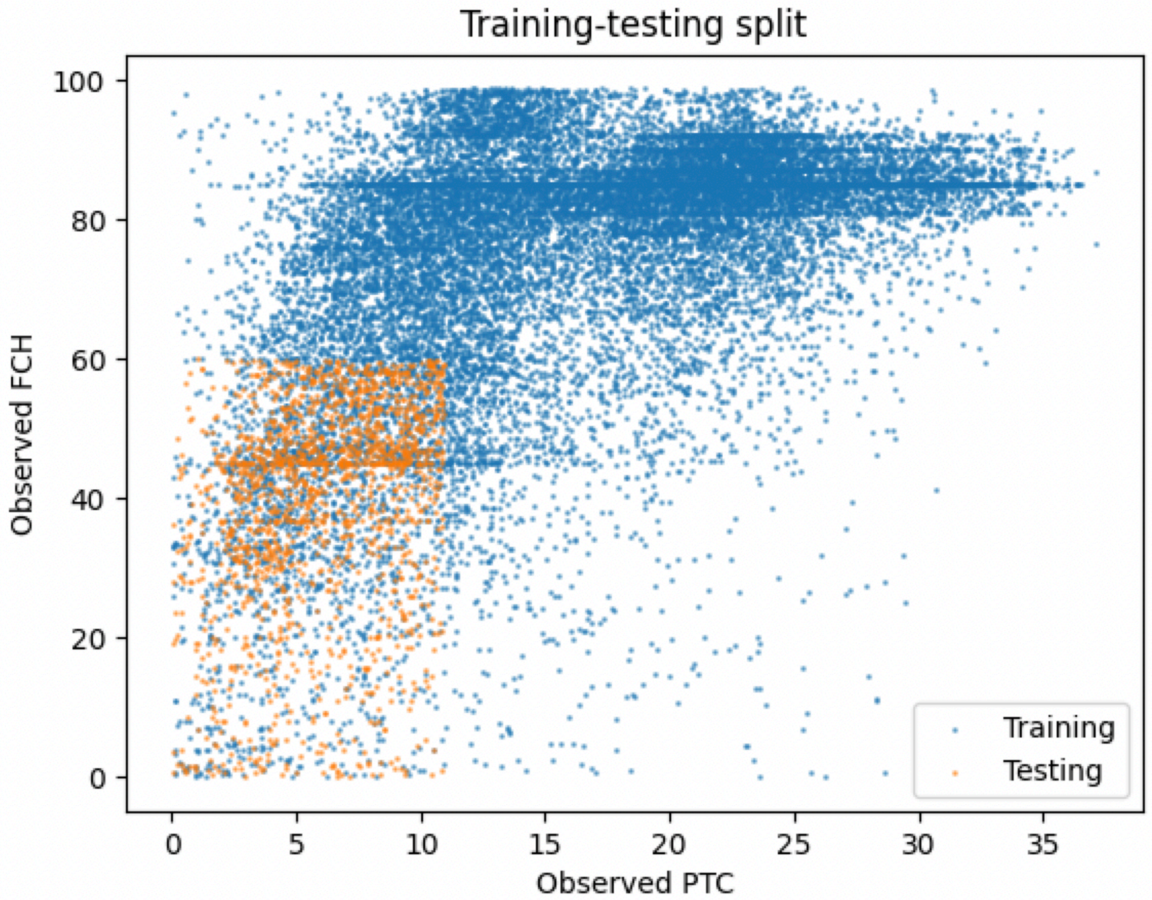
BCEF: choice of “testing area”





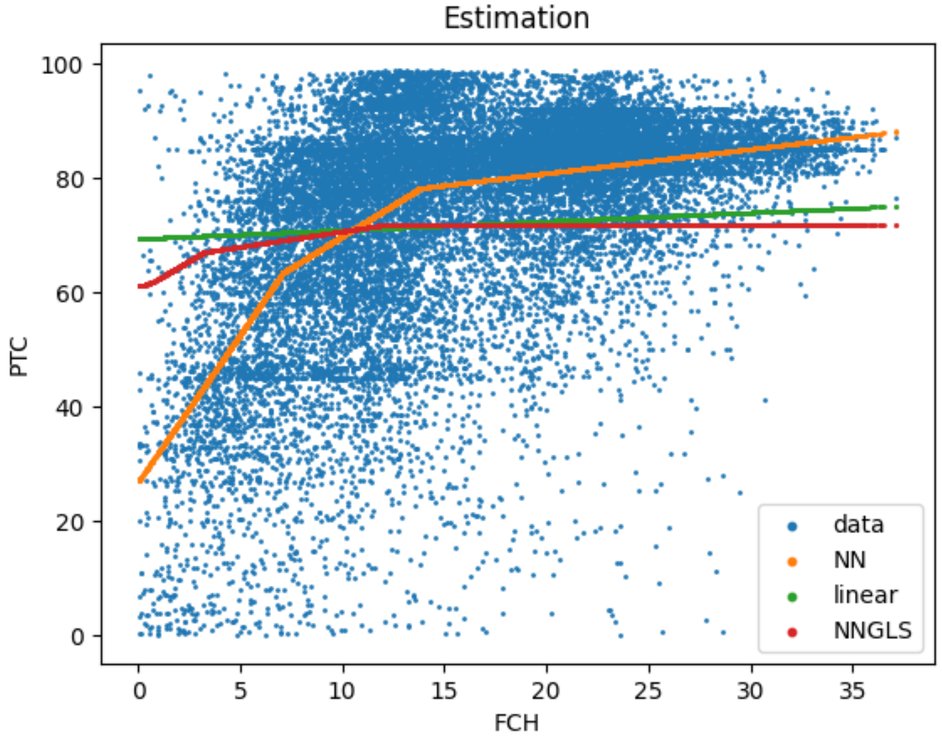
Training testing splitting:

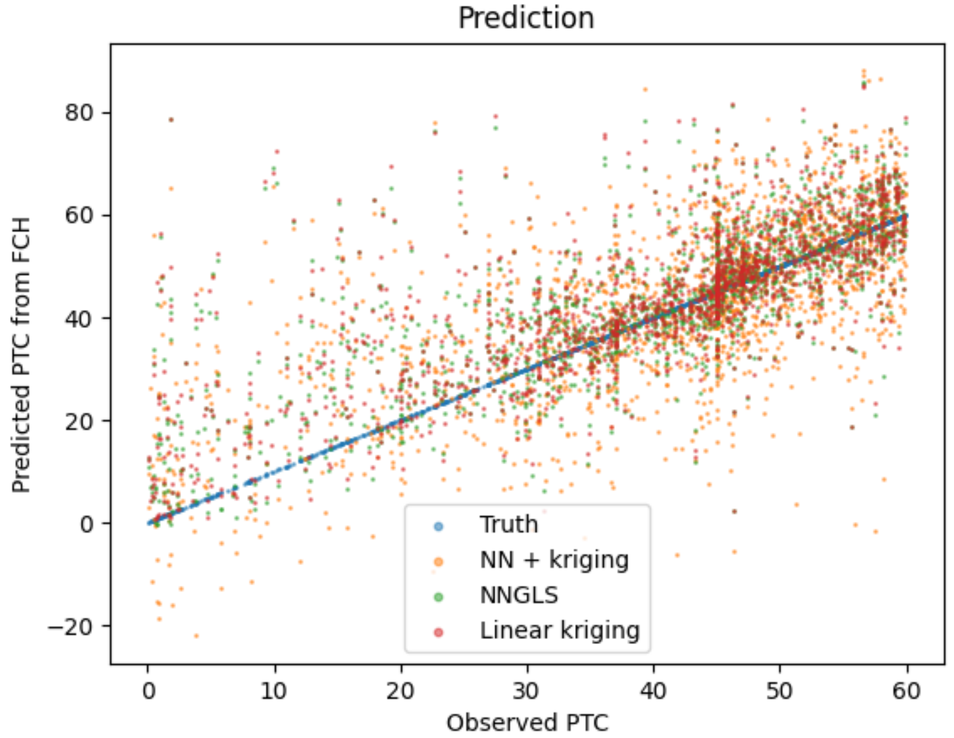




BCEF: special random split

PTC < 60%, FCH < quantile(FCH, 0.3), restricted in the testing area





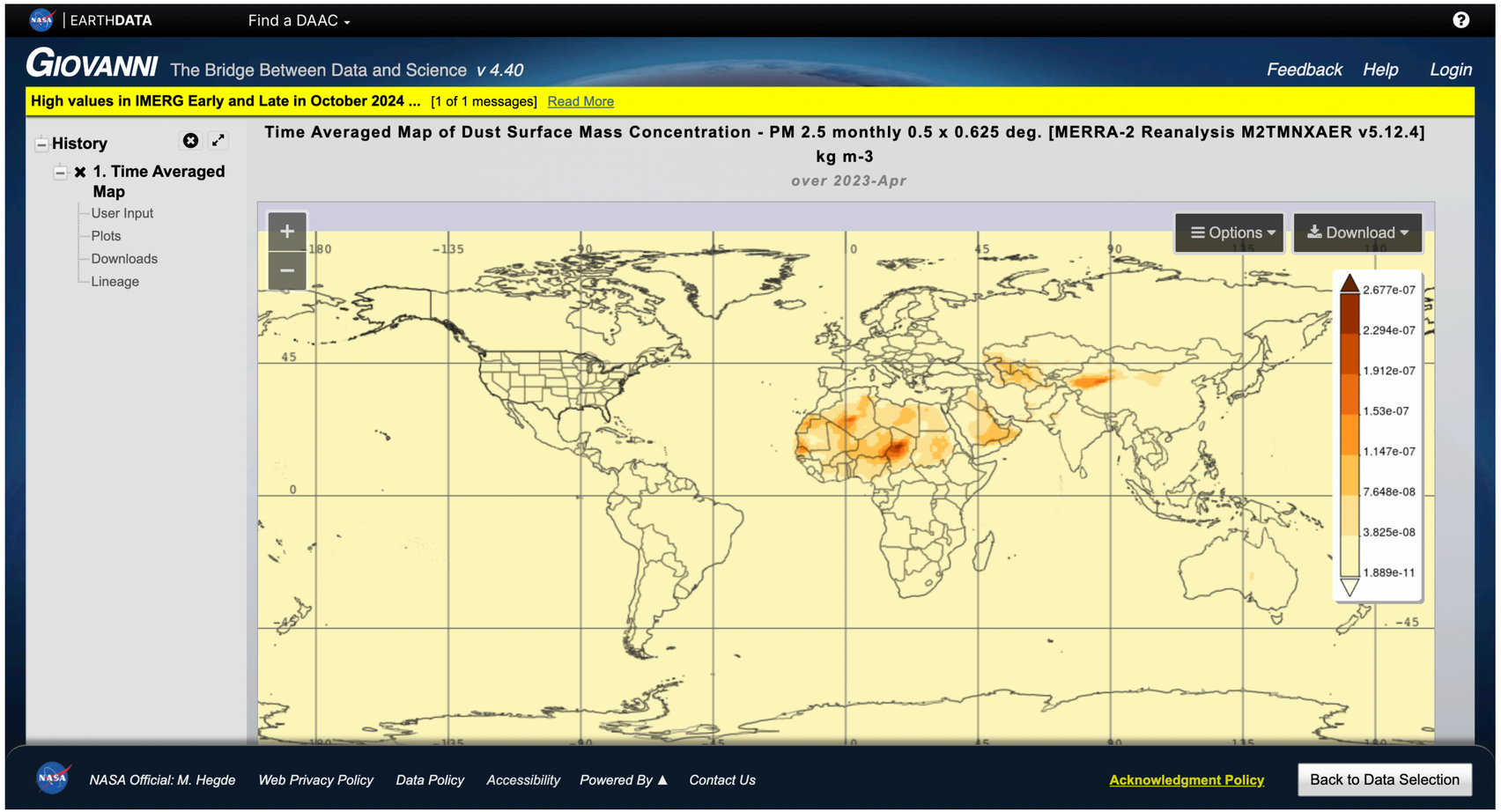
BRISC (σ^2, ϕ, τ) = (461, 66.2, 0.001)

NN-GLS (σ^2, ϕ, τ) = (431, 70.3, 0.001)

MSE:
BRISC 131.13
NNGLS 126.64
NN+kriging 177.97

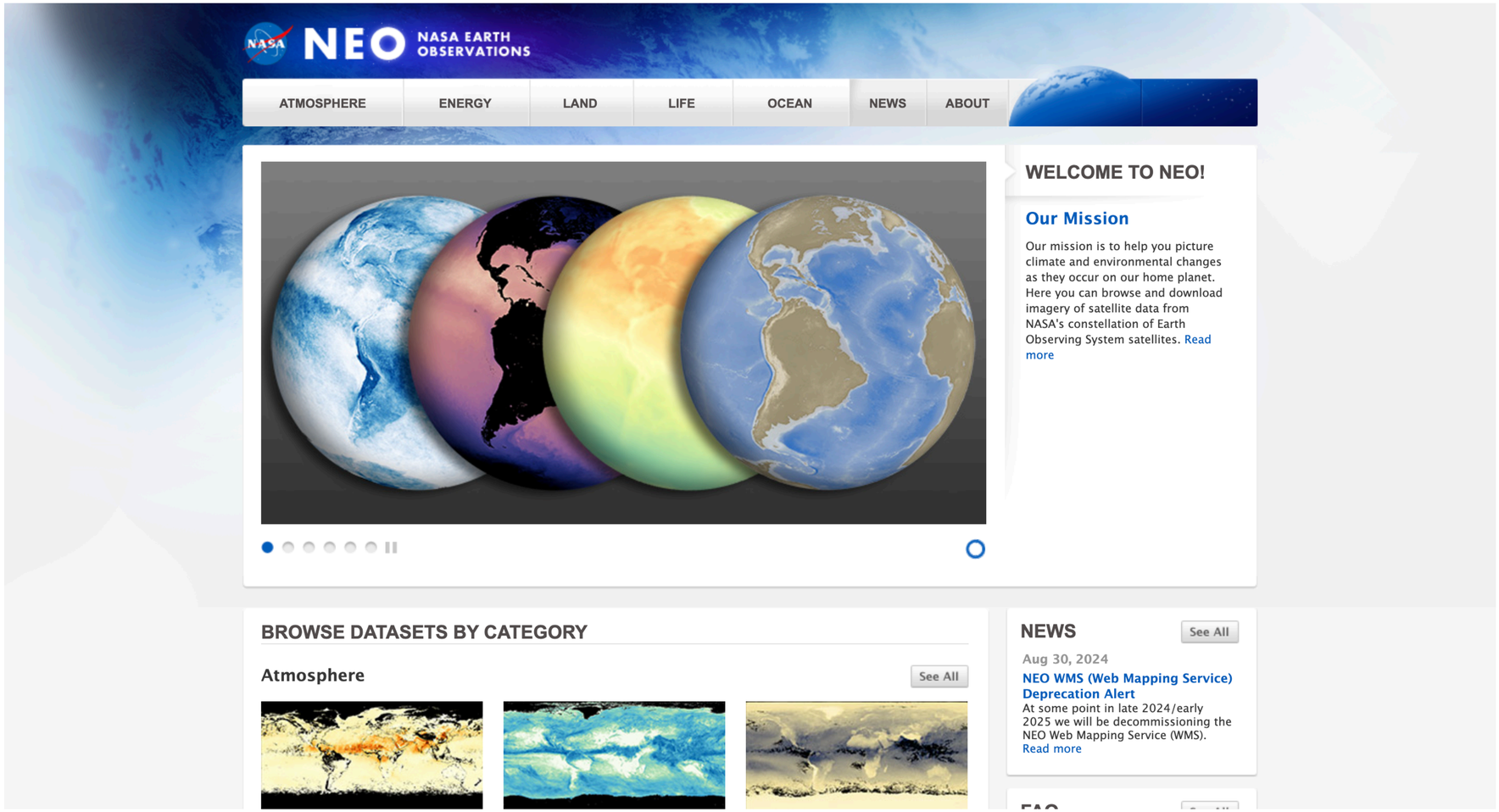
Useful resources of geospatial data

NASA Giovanni earth data: https://giovanni.gsfc.nasa.gov/giovanni



Useful resources of geospatial data

NASA earth observations: https://neo.gsfc.nasa.gov/



Useful resources of geospatial data

National Centers for Environmental Prediction's (NCEP) NARR

An official website of the United States government Here's how you know

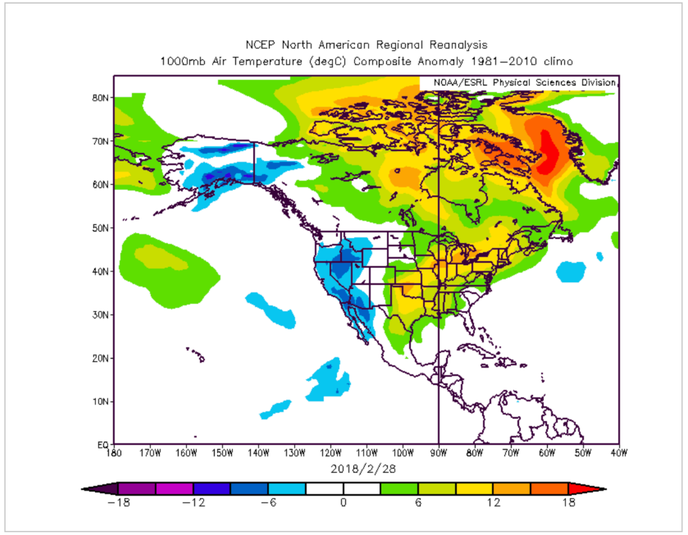
Physical Sciences Laboratory

About People Research Data Products News | Events Learn

Home » Data » Gridded Climate » NCEP North American Regional Reanalysis (NARR)

NCEP North American Regional Reanalysis (NARR)

NCEP's high resolution combined model and assimilated dataset. From 1979 to near present 8-times daily, daily and monthly data is output on a Northern Hemisphere Lambert Conformal Conic grid. See **PSL's NARR project page**.



SPECIFICATIONS

Temporal Coverage:

- 8-times, Daily and Monthly means for 1979/01/01 to 2024/10/31
- Long Term Daily, Monthly means for years 1981-2010

Spatial Coverage:

- The native model grid is converted to a Northern Lambert Conformal Conic grid which is what we archive. Corners of this grid are
  1.000001N, 145.5W; 0.897945N, 68.32005W; 46.3544N, 2.569891W; 46.63433N, 148.6418E
  The grid resolution is 349x277 which is approximately 0.3 degrees (32km) resolution at the lowest latitude. A page describing the coverage along with information on reading the projection is available.

Levels:

- 29 pressure levels (hPa):

Download and Plot Data Data Help

Useful resources of geospatial data

U.S. Environmental Protection Agency: https://www.epa.gov/

An official website of the United States government Here's how you know

EPA United States Environmental Protection Agency

Search EPA.gov

Environmental Topics | Laws & Regulations | Report a Violation | About EPA

Home / Outdoor Air Quality Data

Outdoor Air Quality Data

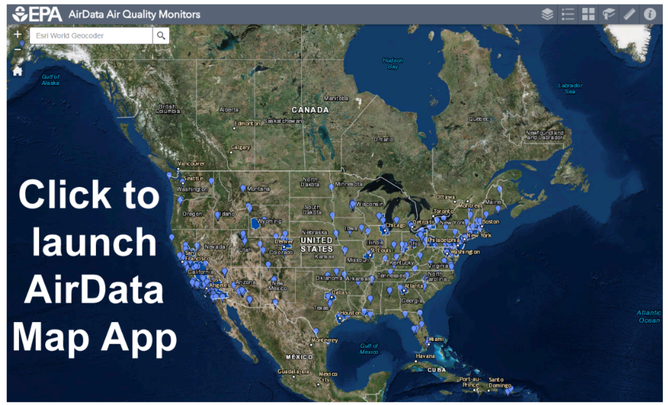
- Frequent Questions about AirData
- Learn about Air Data
- **Interactive Map**
- Pre-generated Data Files
- Download Daily Data
- Download Raw Data (API)
- Air Quality Index Report
- Air Quality Statistics Report
- Monitor Values Report
- Monitor Values Report - Hazardous Air Pollutants
- Air Quality Index Daily Values Report
- Daily Air Quality Tracker

Interactive Map of Air Quality Monitors

The AirData Air Quality Monitors app is a mapping application available on the web and on mobile devices that displays monitor locations and monitor-specific information. It also allows the querying and downloading of data daily and annual summary data.

Map layers include:

- Monitors for all criteria pollutants (CO, Pb, NO2, Ozone, PM10, PM2.5, and SO2)
- PM2.5 Chemical Speciation Network monitors
- IMPROVE (Interagency Monitoring of PROtected Visual Environments) monitors
- NATTS (National Air Toxics Trends Stations)
- NCORE (Multipollutant Monitoring Network)
- PAMS (Photochemical Assessment Monitoring Stations)
- Near road monitors
- Nonattainment areas for all criteria pollutants
- Tribal areas



Thanks!