GPtestScript

July 11, 2023

Python 3.10.11 | packaged by Anaconda, Inc. | (main, Apr 20 2023, 18:56:50) [MSC v.1916 64 bit (AMD64)]
Type 'copyright', 'credits' or 'license' for more information
IPython 8.13.2 – An enhanced Interactive Python. Type '?' for help.

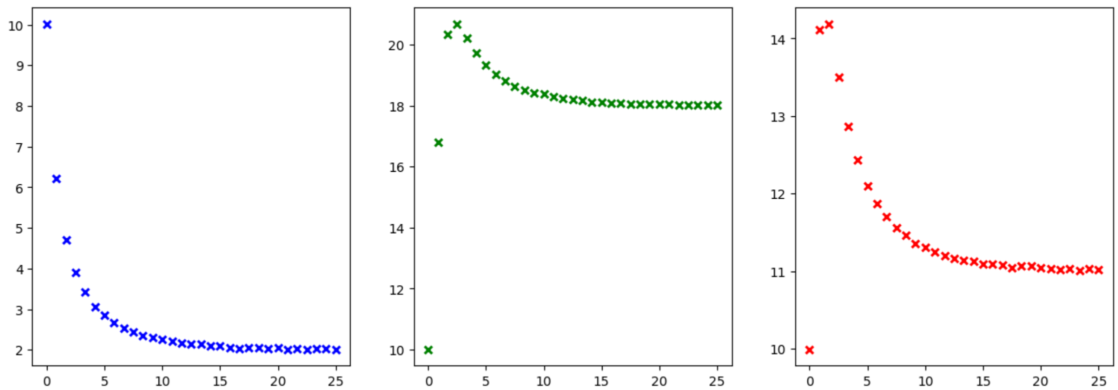
[ ]:
```python
import numpy as np
import math
from numpy.linalg import inv
import matplotlib.pyplot as plt
from numpy.linalg import cholesky, det
from scipy.linalg import solve_triangular
from scipy.optimize import minimize
from scipy.integrate import solve_ivp

import pretty_errors

import gpflow as gpf
from gpflow.utilities import print_summary
from gpflow.utilities import parameter_dict
from gpflow.ci_utils import reduce_in_tests

import tensorflow as tf

gpf.config.set_default_float(np.float64)
gpf.config.set_default_summary_fmt("notebook")
np.random.seed(0)

MAXITER = reduce_in_tests(5000)
```

[ ]:
```python
# The idea is that we simulate from a lotka volterra model with three species.␣
↪This model has three growth rates, mu, plus an interaction matrix, M
# We then fit Gaussian processes to the time courses and use model selection to␣
↪determine the best combination of kernels and mean functions to model the␣
↪data
# What we ultimately want to know is how the original parameters of the LV␣
↪model correspond to the best fitting GPs
# This will enable us to work out what information is contained in the GPs
```

```
[ ]: # This function is the Lotka-Volterra predator-prey model
     # It takes two arguments: t, the time, and y, a vector of the current␣
      ↪population sizes
     # It returns a list of the time derivatives of the populations, in the same␣
      ↪order as the input

     def lotka_volterra(t, y):
         mu = [0.2, 0.7, 0.9]
         M = np.array([[-0.1, 0.0, 0.0], [0.0, -0.1, 0.1], [0.1, 0.0, -0.1]])

         y1 = y[0]
         y2 = y[1]
         y3 = y[2]

         dy1 = y1*mu[0] + y1*(M[0, 0]*y1 + M[0, 1]*y2 + M[0, 2]*y3)
         dy2 = y2*mu[1] + y2*(M[1, 0]*y1 + M[1, 1]*y2 + M[1, 2]*y3)
         dy3 = y3*mu[2] + y3*(M[2, 0]*y1 + M[2, 1]*y2 + M[2, 2]*y3)

         return [dy1, dy2, dy3]


     def simulate(y0, t):
         return solve_ivp(fun=lotka_volterra, t_span=[min(t), max(t)], y0=y0,␣
      ↪t_eval=t, method='LSODA')



     nps = 31
     t = np.linspace(0, 25, nps)
     y0 = [10.0, 10.0, 10.0]
     sol = simulate(y0, t)

     # sample data points
     # s_idx = np.random.choice(len(t), size = 101, replace=False)
     # s_idx.sort()
     s_idx = np.arange(nps)
     ts = sol.t[s_idx]
     ys = sol.y[:, s_idx]

     # add noise to growth data
     y_hat = np.maximum(ys + np.random.normal(scale=0.01, size=ys.shape), 0)

     print(y_hat.shape)

     fig, ax = plt.subplots(figsize=(15, 5), ncols=3, nrows=1)
     ax[0].plot(ts, y_hat[0, :], "bx", mew=2)
     ax[1].plot(ts, y_hat[1, :], "gx", mew=2)
     ax[2].plot(ts, y_hat[2, :], "rx", mew=2)
```

```
(3, 31)
```

[ ]:
```
[<matplotlib.lines.Line2D at 0x1da30008fd0>]
```



[ ]:
```python
# Fit whole system using various multi-ouput kernels and VGP


def plot_gp_d(x, mu, var, color, label, ax):
    ax.plot(x, mu, color=color, lw=2, label=label)
    ax.fill_between(
        x[:, 0],
        (mu - 2 * np.sqrt(var))[:, 0],
        (mu + 2 * np.sqrt(var))[:, 0],
        color=color,
        alpha=0.4,
    )
    ax.set_xlabel("X")
    ax.set_ylabel("y")


def plot_model(m, X, P, L, K_L, M_F, BIC):
    fig, ax = plt.subplots(figsize=(15, 5), ncols=3, nrows=1)
    ax[0].plot(X[:, 0], Y[:, 0], "bx", mew=2)
    ax[1].plot(X[:, 0], Y[:, 1], "gx", mew=2)
    ax[2].plot(X[:, 0], Y[:, 2], "rx", mew=2)

    # just use the GP to predict at same timepoints
    mu1, var1 = m.predict_y(np.hstack((X, np.zeros_like(X))))
    plot_gp_d(X, mu1, var1, "b", "Y1", ax[0])

    mu2, var2 = m.predict_y(np.hstack((X, np.ones_like(X))))
    plot_gp_d(X, mu2, var2, "g", "Y2", ax[1])
```

```python
    mu3, var3 = m.predict_y(np.hstack((X, 2*np.ones_like(X))))
    plot_gp_d(X, mu3, var3, "r", "Y3", ax[2])

    fig.suptitle('species= ' + str(P) + ', latent_processes= ' + str(L) + ', kernel= ' +
                 str(K_L.__name__) + ', mean= ' + str(M_F.__class__.__name__) + ', BIC =' + str(BIC))


def optimize_model_with_scipy(model):
    optimizer = gpf.optimizers.Scipy()
    res = optimizer.minimize(
        model.training_loss_closure((X, Y)),
        variables=model.trainable_variables,
        method="l-bfgs-b",
        # options={"disp": 50, "maxiter": MAXITER},
        options={"maxiter": MAXITER},
    )
    return res


def count_params(m):
    p_dict = parameter_dict(m.trainable_parameters)
    # p_dict = parameter_dict(m)
    p_count = 0
    for val in p_dict.values():
        # print(val.shape)
        if len(val.shape) == 0:
            p_count = p_count + 1
        else:
            p_count = p_count + math.prod(val.shape)

    return p_count

# This is for model selection: the lower the BIC the better the model


def get_BIC(m, F, n):
    # Assumes F = -lnL
    # QUESTION: is this correct? Are we sure it is model parameters and not number of kernels parameters?
    k = count_params(m)
    return -2 * F + k * np.log(n)
    #return (-1/2)*k*np.log(n) + F
    # return k*np.log(n) + 2*F
```

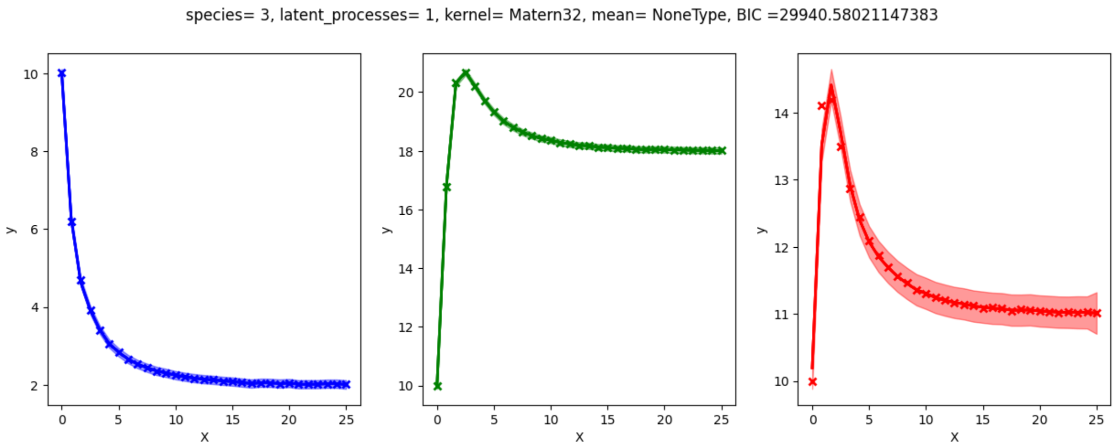
[ ]:
```python
# Here do coregionalization to estimate f(x) = W g(x)
# https://gpflow.github.io/GPflow/2.8.0/notebooks/advanced/multioutput.html
# https://gpflow.github.io/GPflow/develop/notebooks/getting_started/
↪mean_functions.html
# https://towardsdatascience.com/
↪sparse-and-variational-gaussian-process-what-to-do-when-data-is-large-2d3959f430e7
# https://gpflow.readthedocs.io/en/v1.5.1-docs/notebooks/advanced/
↪coregionalisation.html
# https://gpflow.github.io/GPflow/2.4.0/notebooks/advanced/coregionalisation.
↪html
# This uses VGP

X = ts.reshape(-1, 1)
Y = y_hat.T

print(X.shape)
print(Y.shape)

# Augment the input with ones or zeros to indicate the required output dimension
X_aug = np.vstack(
    (np.hstack((X, np.zeros_like(X))),
     np.hstack((X, np.ones_like(X))),
     np.hstack((X, 2*np.ones_like(X)))
     )
)

# Augment the Y data with ones or zeros that specify a likelihood from the list␣
↪of likelihoods
Y1 = Y[:, 0].reshape(-1, 1)
Y2 = Y[:, 1].reshape(-1, 1)
Y3 = Y[:, 2].reshape(-1, 1)

Y_aug = np.vstack(
    (np.hstack((Y1, np.zeros_like(Y1))),
     np.hstack((Y2, np.ones_like(Y2))),
     np.hstack((Y3, 2*np.ones_like(Y3)))
     )
)

# print(X)
# print(X_aug)
# print(Y_aug)
```

```
(31, 1)
(31, 3)

(31, 3)
```

[ ]:
```python
L = 1  # latent processes, g in R^L
P = 3  # observed processes, f in R^P

# Base kernel
k = gpf.kernels.Matern32(active_dims=[0])
# k = gpf.kernels.SquaredExponential(active_dims=[0])

# Coregion kernel
coreg = gpf.kernels.Coregion(
    output_dim=P,
    rank=L,
    active_dims=[1]
)

kern = k * coreg

# This likelihood switches between Gaussian noise with different variances for␣
↪each f_i:
lik = gpf.likelihoods.SwitchedLikelihood(
    [gpf.likelihoods.Gaussian(), gpf.likelihoods.Gaussian(),
     gpf.likelihoods.Gaussian()]
)

# now build the GP model as normal
m = gpf.models.VGP((X_aug, Y_aug), kernel=kern, likelihood=lik)

# fit the covariance function parameters
maxiter = reduce_in_tests(10000)
res = gpf.optimizers.Scipy().minimize(
    m.training_loss,
    m.trainable_variables,
    options=dict(maxiter=maxiter),
    method="L-BFGS-B",
)
# # get the maximum likelihood estimate of model hyperparameters
# m.kernel.kernels[0].lengthscales.numpy()
# m.kernel.kernels[0].variance.numpy()
# m.kernel.kernels[1].W.numpy()
# m.likelihood.variance.numpy()


print_summary(m)
# plot_model(m)
BIC = get_BIC(m, res.fun, X.shape[0])
print(BIC)
plot_model(m, X, P, L, gpf.kernels.Matern32, M_F=None, BIC=BIC)
```

```
<IPython.core.display.HTML object>
```

29940.58021147383




```
[ ]: # Wrap above code into a funtion
     # P is the number of outputs (three in this case for the three species)
     # L is the number of latent processes
     # K_L is the kernel for the latent processes
     # M_F is the mean function applied to latent processes


     def fit_model(X_aug, Y_aug, P, L, K_L=gpf.kernels.SquaredExponential, M_F=None):

         # Base kernel for leatent processes
         # k = gpf.kernels.Matern32(active_dims=[0])
         # k = gpf.kernels.SquaredExponential(active_dims=[0])

         k = K_L(active_dims=[0])

         # Coregion kernel
         coreg = gpf.kernels.Coregion(
             output_dim=P,
             rank=L,
             active_dims=[1]
         )

         kern = k * coreg

         # This likelihood switches between Gaussian noise with different variances␣
      ↪for each f_i:
         lik = gpf.likelihoods.SwitchedLikelihood(
             [gpf.likelihoods.Gaussian() for _ in range(P)]
         )
```

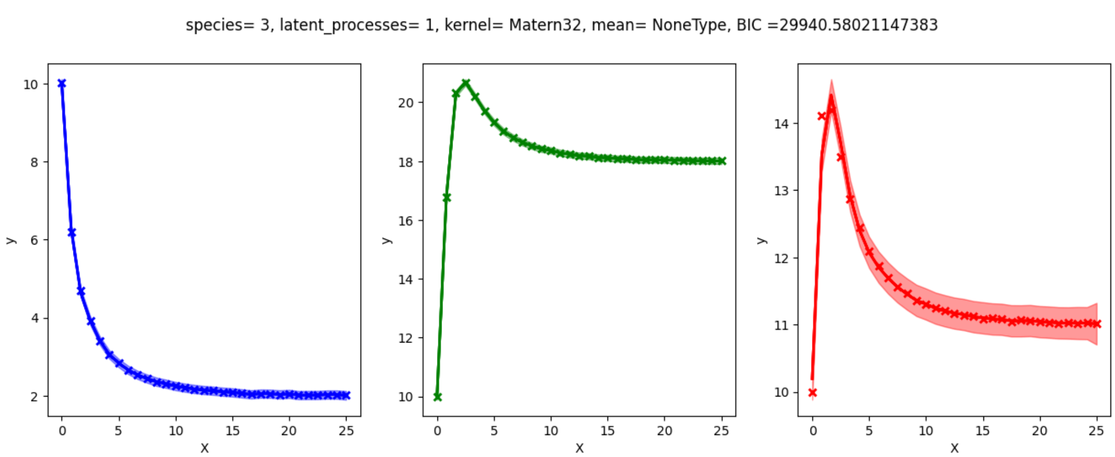
```
    # now build the GP model as normal
    m = gpf.models.VGP((X_aug, Y_aug), kernel=kern,
                       likelihood=lik, mean_function=M_F)

    # fit the covariance function parameters
    maxiter = reduce_in_tests(10000)
    res = gpf.optimizers.Scipy().minimize(
        m.training_loss,
        m.trainable_variables,
        options=dict(maxiter=maxiter),
        method="L-BFGS-B",
    )

    print_summary(m)
    BIC = get_BIC(m, res.fun, X.shape[0])
    print("BIC:", BIC)
    plot_model(m, X, P, L, K_L, M_F, BIC)

    return m, BIC


# Try different kernels
# k = gpf.kernels.Matern32(active_dims=[0])
# k = gpf.kernels.SquaredExponential(active_dims=[0])
# k = gpf.kernels.RationalQuadratic(active_dims=[0])
# k = gpf.kernels.Exponential(active_dims=[0])
# k = gpf.kernels.Linear(active_dims=[0])
# k = gpf.kernels.Cosine(active_dims=[0])
# k = gpf.kernels.Periodic(active_dims=[0])
# k = gpf.kernels.Polynomial(active_dims=[0])
# k = gpf.kernels.Matern12(active_dims=[0])
# k = gpf.kernels.Matern52(active_dims=[0])
# k = gpf.kernels.Brownian(active_dims=[0])
# k = gpf.kernels.White(active_dims=[0])
# k = gpf.kernels.Constant(active_dims=[0])
# k = gpf.kernels.Coregion(active_dims=[0])
# k = gpf.kernels.ChangePoints(active_dims=[0])
# k = gpf.kernels.LinearCoregionalization(active_dims=[0])
```

[ ]:
```
fit_model(X_aug, Y_aug, 3, 1, gpf.kernels.Matern32)
```

```
<IPython.core.display.HTML object>

BIC: 29940.58021147383
```

[ ]:
```
(<gpflow.models.vgp.VGP object at 0x000001DA31777FA0>
 name                                          class      transform          prior
```

```
trainable    shape        dtype    value
-------------------------------------  ---------  ----------------  -------
-----------  -----------  -------  ----------------------------------------
VGP.kernel.kernels[0].variance          Parameter  Softplus
True         ()           float64  8.35352
VGP.kernel.kernels[0].lengthscales      Parameter  Softplus
True         ()           float64  19.86844
VGP.kernel.kernels[1].W                 Parameter  Identity
True         (3, 1)       float64  [[-4.58565]
 [ 8.64141]
 [ 5.41759]]
VGP.kernel.kernels[1].kappa             Parameter  Softplus
True         (3,)         float64  [4.49827 4.80225 2.77682]
VGP.likelihood.likelihoods[0].variance  Parameter  Softplus + Shift
True         ()           float64  0.00158
VGP.likelihood.likelihoods[1].variance  Parameter  Softplus + Shift
True         ()           float64  0.00126
VGP.likelihood.likelihoods[2].variance  Parameter  Softplus + Shift
True         ()           float64  0.02874
VGP.num_data                            Parameter  Identity
False        ()           int32    93
VGP.q_mu                                Parameter  Identity
True         (93, 1)      float64  [[0.68452…
VGP.q_sqrt                              Parameter  FillTriangular
True         (1, 93, 93)  float64  [[[2.6700e-03, 0.0000e+00, 0.0000e+00…,
 29940.58021147383)
```

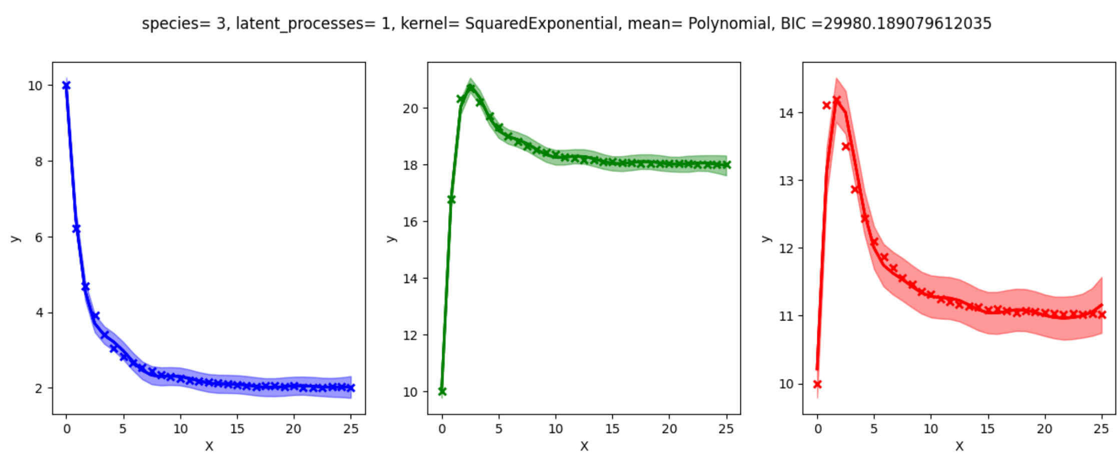



```
[ ]: fit_model(X_aug, Y_aug, 3, 1, gpf.kernels.SquaredExponential,
               M_F=gpf.functions.Polynomial(2))
```

<IPython.core.display.HTML object>

```
BIC: 29980.189079612035
```

[ ]:
```
(<gpflow.models.vgp.VGP object at 0x000001DA33B58EE0>
name                                     class      transform         prior
trainable    shape        dtype    value
---------------------------------------  ---------  ----------------  -------
-----------  -----------  -------  -------------------------------------------
VGP.mean_function.w                      Parameter  Identity
True         (1, 3)       float64  [[ 9.38712e+00 -4.02740e-01  5.53000e-03]]
VGP.kernel.kernels[0].variance           Parameter  Softplus
True         ()           float64  4.40948
VGP.kernel.kernels[0].lengthscales       Parameter  Softplus
True         ()           float64  4.29087
VGP.kernel.kernels[1].W                  Parameter  Identity
True         (3, 1)       float64  [[-4.14666]
 [ 8.85292]
 [ 4.79001]]
VGP.kernel.kernels[1].kappa              Parameter  Softplus
True         (3,)         float64  [0.08466 0.21568 0.0995 ]
VGP.likelihood.likelihoods[0].variance  Parameter  Softplus + Shift
True         ()           float64  0.01107
VGP.likelihood.likelihoods[1].variance  Parameter  Softplus + Shift
True         ()           float64  0.01519
VGP.likelihood.likelihoods[2].variance  Parameter  Softplus + Shift
True         ()           float64  0.06589
VGP.num_data                             Parameter  Identity
False        ()           int32    93
VGP.q_mu                                 Parameter  Identity
True         (93, 1)      float64  [[6.13100e-02…
VGP.q_sqrt                               Parameter  FillTriangular
True         (1, 93, 93)  float64  [[[1.00600e-02, 0.00000e+00, 0.00000e+00…,
 29980.189079612035)
```



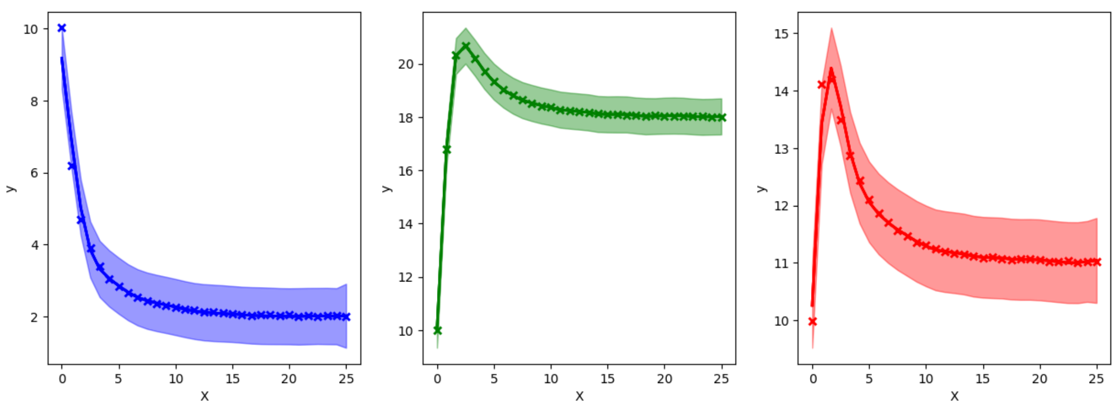
[ ]:
```python
# Try different numbers of latent processes, L, with the same kernel, K_L = matern32 and mean function, M_F = polynomial
fit_model(X_aug, Y_aug, 3, 1, gpf.kernels.Matern32,
          M_F=gpf.functions.Polynomial(2))
fit_model(X_aug, Y_aug, 3, 2, gpf.kernels.Matern32,
          M_F=gpf.functions.Polynomial(2))
fit_model(X_aug, Y_aug, 3, 3, gpf.kernels.Matern32,
          M_F=gpf.functions.Polynomial(2))

# Try different numbers of latent processes, L, with the same kernel, K_L = squared exponential and mean function, M_F = polynomial
fit_model(X_aug, Y_aug, 3, 1, gpf.kernels.SquaredExponential,
          M_F=gpf.functions.Polynomial(2))
fit_model(X_aug, Y_aug, 3, 2, gpf.kernels.SquaredExponential,
          M_F=gpf.functions.Polynomial(2))
fit_model(X_aug, Y_aug, 3, 3, gpf.kernels.SquaredExponential,
          M_F=gpf.functions.Polynomial(2))
```

```
<IPython.core.display.HTML object>

BIC: 29884.630996191674

<IPython.core.display.HTML object>

BIC: 29970.216530885846

<IPython.core.display.HTML object>

BIC: 29993.865315112755

<IPython.core.display.HTML object>

BIC: 29980.189079612035

<IPython.core.display.HTML object>

BIC: 29975.125836376308

<IPython.core.display.HTML object>

BIC: 29646.615139964797
```

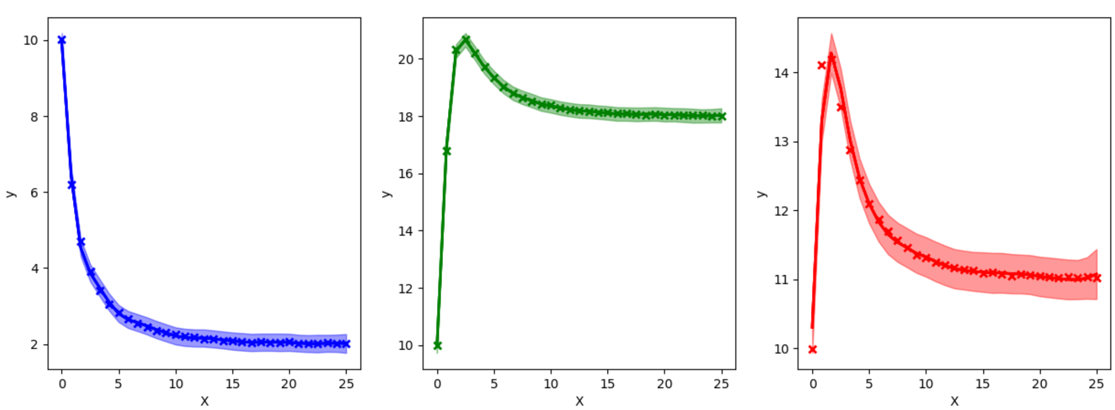
[ ]:
```
(<gpflow.models.vgp.VGP object at 0x000001DA3517D870>
name                                      class      transform         prior
trainable     shape         dtype    value
--------------------------------------  ---------  ----------------  -------
-----------  -----------  -------  ---------------------------------
VGP.mean_function.w                       Parameter  Identity
True          (1, 3)        float64  [[ 4.61277  0.7335  -0.01315]]
VGP.kernel.kernels[0].variance            Parameter  Softplus
True          ()            float64  3.78979
VGP.kernel.kernels[0].lengthscales        Parameter  Softplus
True          ()            float64  3.14788
```

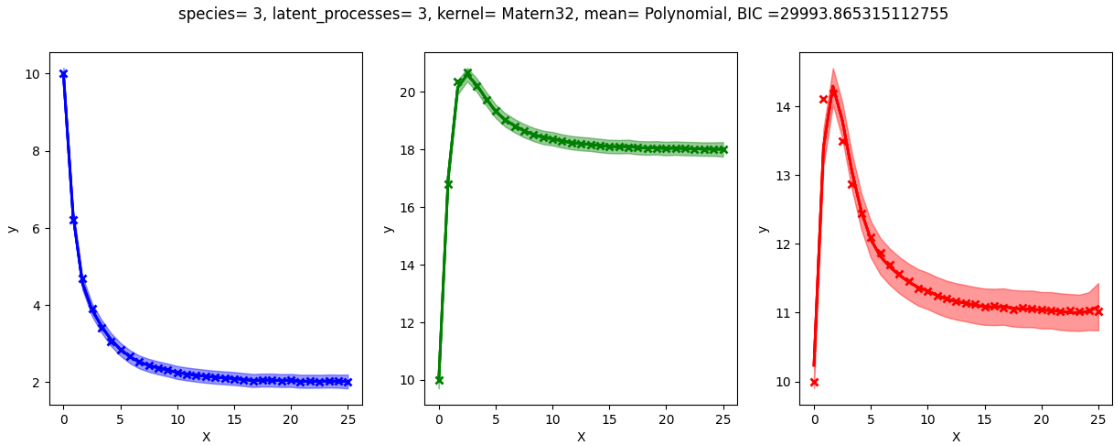
```
VGP.kernel.kernels[1].W                      Parameter  Identity
True          (3, 3)       float64  [[-0.26855, -0.26855, -0.26855…
VGP.kernel.kernels[1].kappa                  Parameter  Softplus
True          (3,)         float64  [0.7139   3.00268 1.38842]
VGP.likelihood.likelihoods[0].variance  Parameter  Softplus + Shift
True          ()           float64  4.00782
VGP.likelihood.likelihoods[1].variance  Parameter  Softplus + Shift
True          ()           float64  14.9812
VGP.likelihood.likelihoods[2].variance  Parameter  Softplus + Shift
True          ()           float64  8.42401
VGP.num_data                                 Parameter  Identity
False         ()           int32    93
VGP.q_mu                                     Parameter  Identity
True          (93, 1)      float64  [[-4.52000e-03…
VGP.q_sqrt                                   Parameter  FillTriangular
True          (1, 93, 93)  float64  [[[0.59626, 0., 0…,
 29646.615139964797)
```

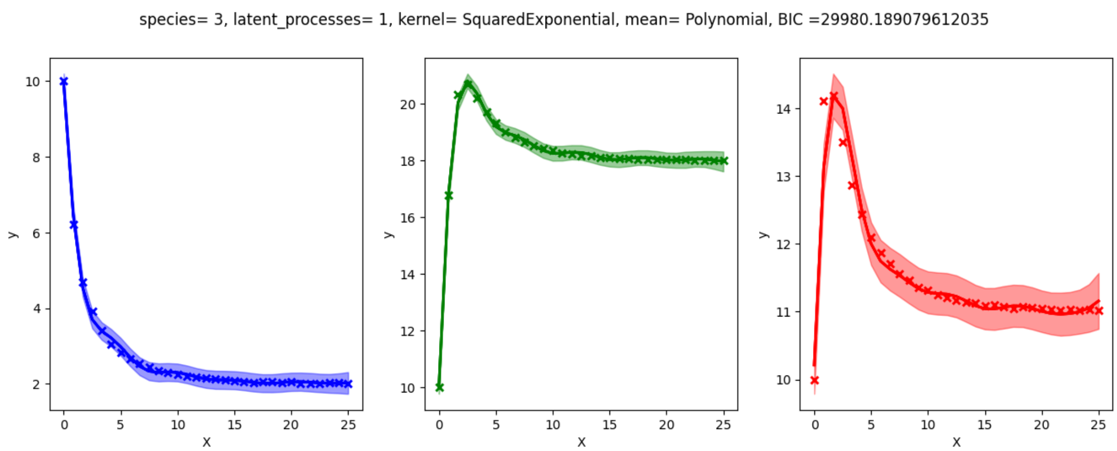
species= 3, latent_processes= 1, kernel= Matern32, mean= Polynomial, BIC =29884.630996191674

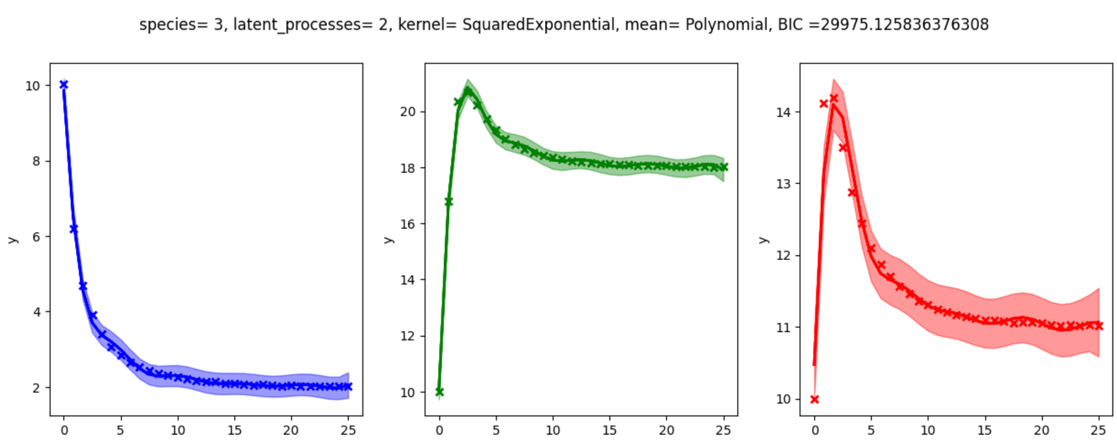


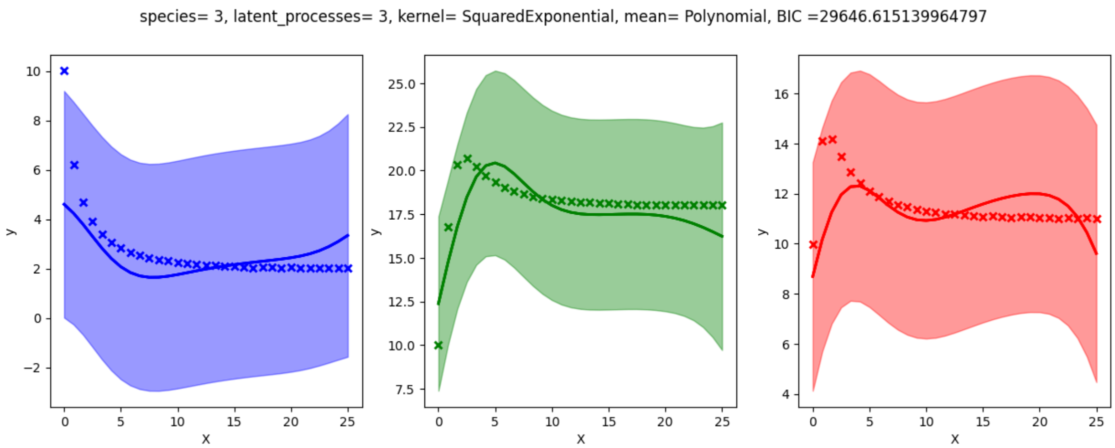
species= 3, latent_processes= 2, kernel= Matern32, mean= Polynomial, BIC =29970.216530885846




species= 3, latent_processes= 3, kernel= Matern32, mean= Polynomial, BIC =29993.865315112755
y
X


species= 3, latent_processes= 1, kernel= SquaredExponential, mean= Polynomial, BIC =29980.189079612035
y
X


species= 3, latent_processes= 2, kernel= SquaredExponential, mean= Polynomial, BIC =29975.125836376308
y
X




```
[ ]: # Identifying the best kernel, mean function and latent process dimensionality␣
     ↪for the data set using BIC score as the metric for comparison of models

     best_BIC = 0
     kernels = [gpf.kernels.SquaredExponential, gpf.kernels.Matern32, gpf.kernels.
      ↪RationalQuadratic, gpf.kernels.Exponential, gpf.kernels.Linear,
                 gpf.kernels.Cosine, gpf.kernels.Periodic, gpf.kernels.Polynomial,␣
      ↪gpf.kernels.Matern12, gpf.kernels.Matern52, gpf.kernels.White]
     reduced_kernels = [gpf.kernels.SquaredExponential, gpf.kernels.Matern32,
                        gpf.kernels.RationalQuadratic, gpf.kernels.Exponential, gpf.
      ↪kernels.Linear, gpf.kernels.Polynomial]
     for L in range(1, 4):
         for K_L in reduced_kernels:
             for M_F in [None, gpf.functions.Polynomial(2)]:
                 m, BIC = fit_model(X_aug, Y_aug, P=3, L=L, K_L=K_L, M_F=M_F)
                 if BIC > best_BIC:
                     best_model = m
                     best_BIC = BIC
                     best_L = L
                     best_K_L = K_L
                     best_M_F = M_F
```

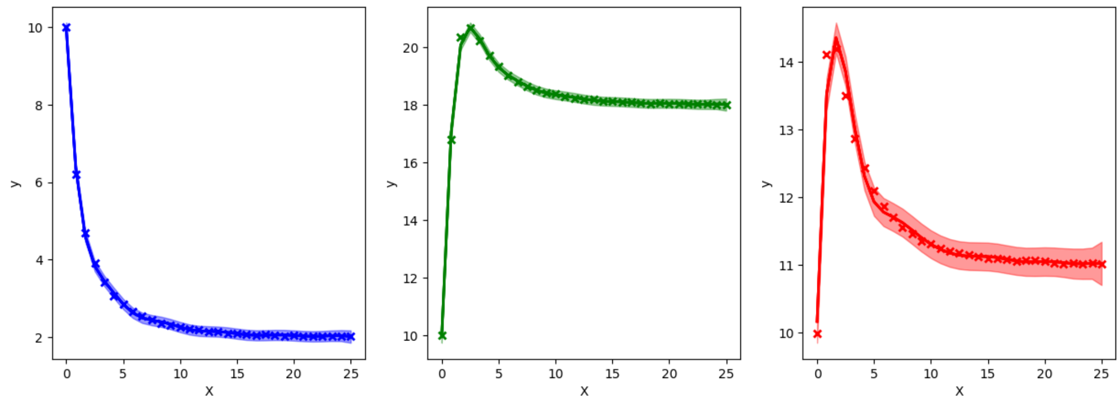
```
<IPython.core.display.HTML object>

BIC: 29961.571591261807

<IPython.core.display.HTML object>

BIC: 29980.189079612035

<IPython.core.display.HTML object>

BIC: 29940.58021147383

<IPython.core.display.HTML object>
```

```
BIC: 29884.630996191674
<IPython.core.display.HTML object>
BIC: 29967.270130849698
<IPython.core.display.HTML object>
BIC: 29967.16004327142
<IPython.core.display.HTML object>
BIC: 29866.96863441727
<IPython.core.display.HTML object>
BIC: 29929.597901621182
<IPython.core.display.HTML object>
BIC: 29469.795305646727
<IPython.core.display.HTML object>
BIC: 29632.52825181503
<IPython.core.display.HTML object>
BIC: 29742.92206499892
<IPython.core.display.HTML object>
BIC: 29752.272365914007
<IPython.core.display.HTML object>
BIC: 29971.611270027985
<IPython.core.display.HTML object>
BIC: 29975.125836376308
<IPython.core.display.HTML object>
BIC: 29945.63888629788
<IPython.core.display.HTML object>
BIC: 29970.216530885846
<IPython.core.display.HTML object>
BIC: 29983.89676161529
<IPython.core.display.HTML object>
BIC: 29979.3937193398
<IPython.core.display.HTML object>
BIC: 29872.217551092643
<IPython.core.display.HTML object>
```

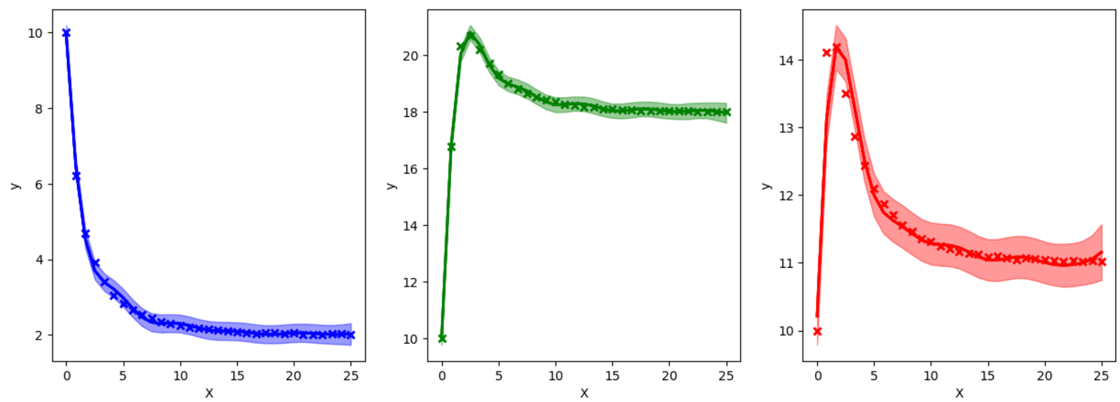
```
BIC: 29924.987014069815

<IPython.core.display.HTML object>

BIC: 29480.100007819696

<ipython-input-4-8a4850ed32c1>:19: RuntimeWarning: More than 20 figures have
been opened. Figures created through the pyplot interface
(`matplotlib.pyplot.figure`) are retained until explicitly closed and may
consume too much memory. (To control this warning, see the rcParam
`figure.max_open_warning`). Consider using `matplotlib.pyplot.close()`.
  fig, ax = plt.subplots(figsize=(15, 5), ncols=3, nrows=1)

<IPython.core.display.HTML object>

BIC: 29636.495799474375

<IPython.core.display.HTML object>

BIC: 29753.226075587925

<IPython.core.display.HTML object>

BIC: 29812.144650106566

<IPython.core.display.HTML object>

BIC: 29982.596613862595

<IPython.core.display.HTML object>

BIC: 29646.615139964797

<IPython.core.display.HTML object>

BIC: 29956.01627627161

<IPython.core.display.HTML object>

BIC: 29993.865315112755

<IPython.core.display.HTML object>

BIC: 29993.09293011421

<IPython.core.display.HTML object>

BIC: 29823.98372678672

<IPython.core.display.HTML object>

BIC: 29876.762686167884

<IPython.core.display.HTML object>

BIC: 29948.634605385152

<IPython.core.display.HTML object>

BIC: 29490.40079635991

<IPython.core.display.HTML object>
```

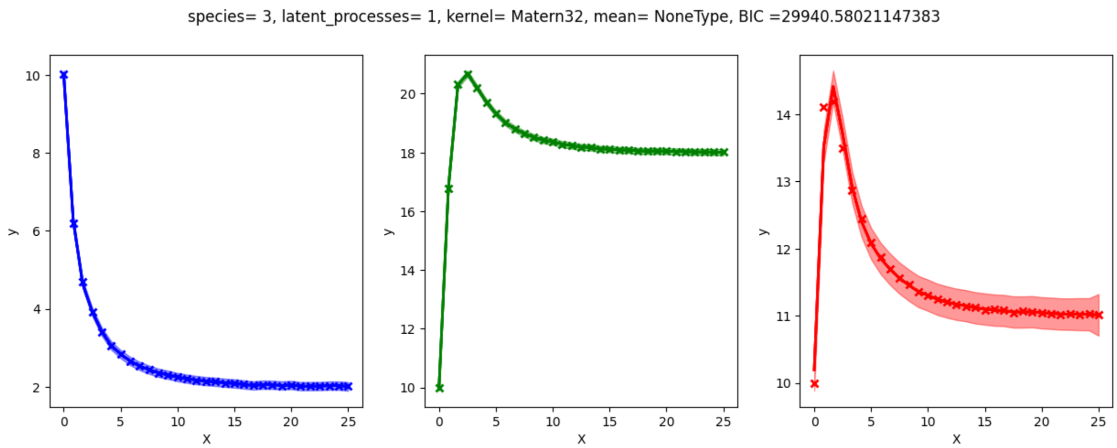
```
BIC: 29646.221211592572

&lt;IPython.core.display.HTML object&gt;

BIC: 29763.52493790427

&lt;IPython.core.display.HTML object&gt;

BIC: 29575.235785273857
```

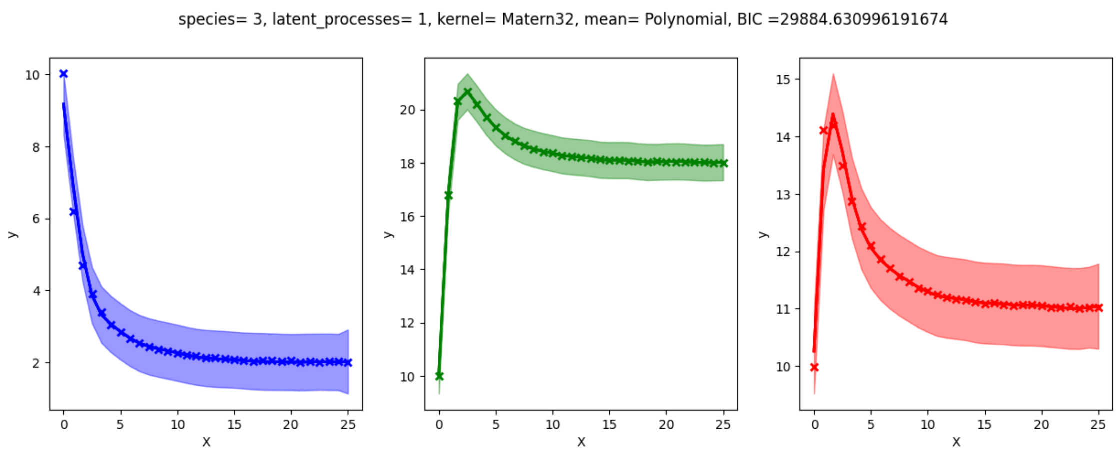
species= 3, latent_processes= 1, kernel= SquaredExponential, mean= NoneType, BIC =29961.571591261807

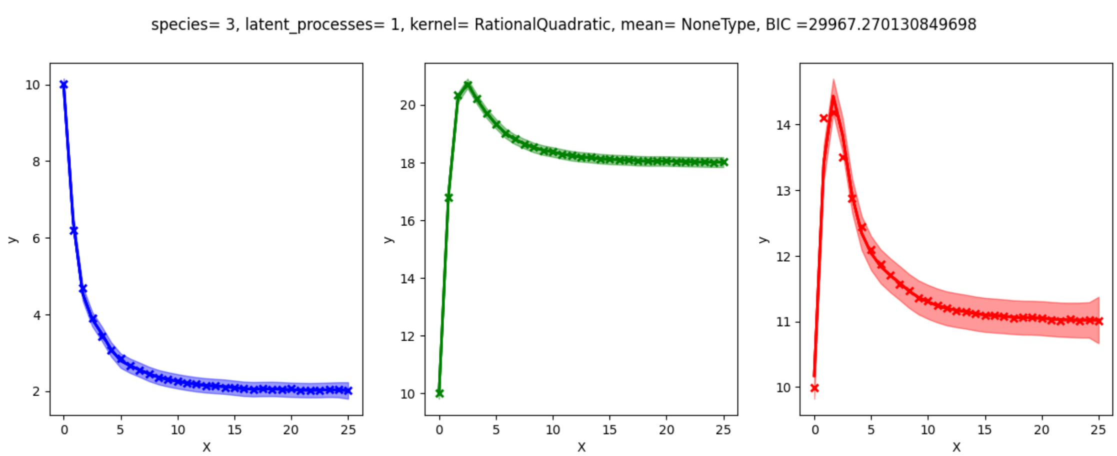


species= 3, latent_processes= 1, kernel= SquaredExponential, mean= Polynomial, BIC =29980.189079612035

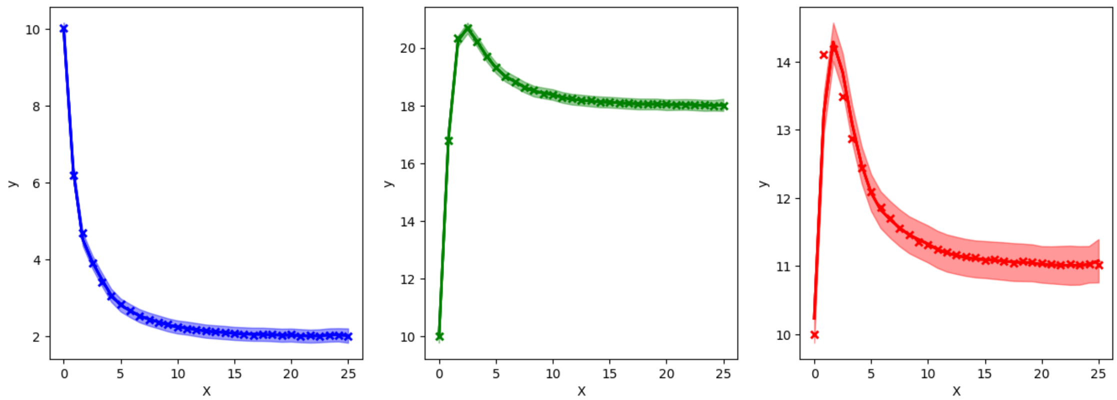



species= 3, latent_processes= 1, kernel= Matern32, mean= NoneType, BIC =29940.58021147383
x
y

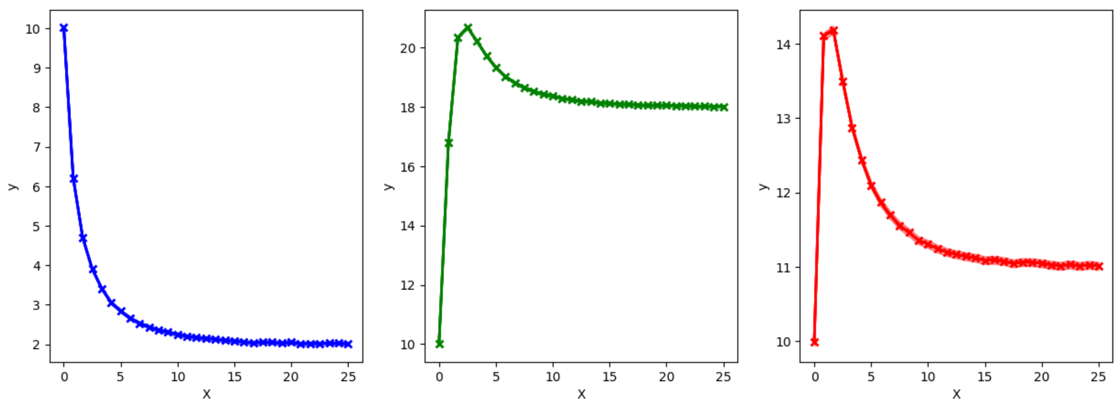

species= 3, latent_processes= 1, kernel= Matern32, mean= Polynomial, BIC =29884.630996191674
x
y


species= 3, latent_processes= 1, kernel= RationalQuadratic, mean= NoneType, BIC =29967.270130849698
x
y

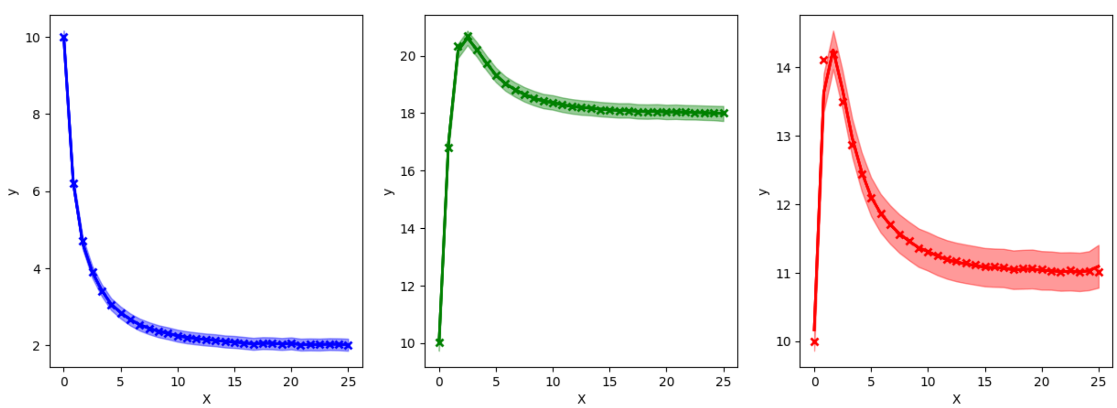
species= 3, latent_processes= 1, kernel= RationalQuadratic, mean= Polynomial, BIC =29967.16004327142



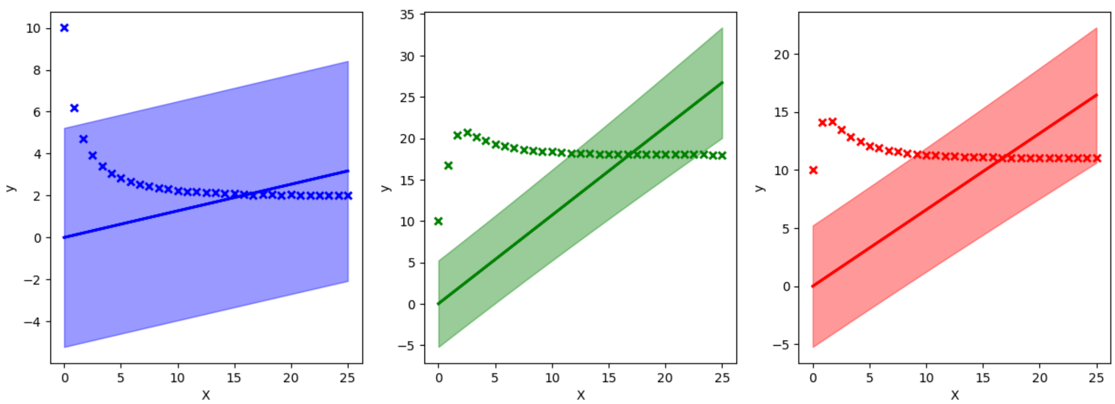
species= 3, latent_processes= 1, kernel= Exponential, mean= NoneType, BIC =29866.96863441727



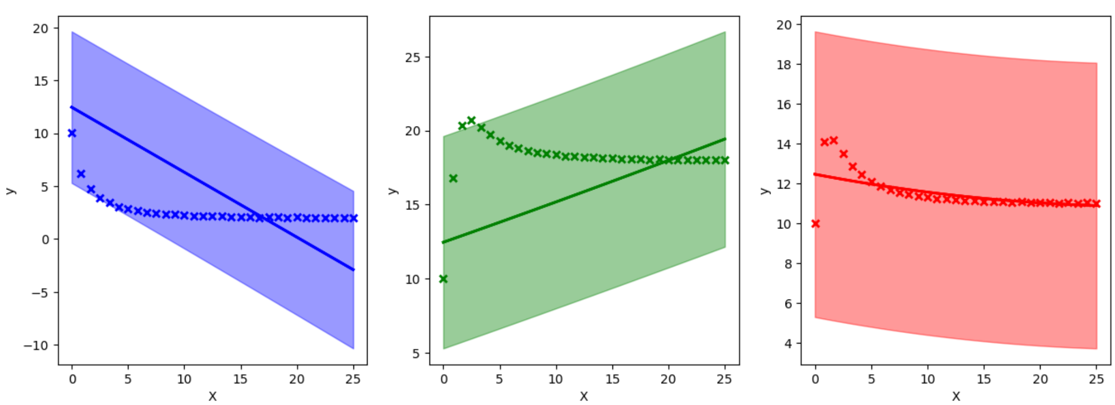
species= 3, latent_processes= 1, kernel= Exponential, mean= Polynomial, BIC =29929.597901621182



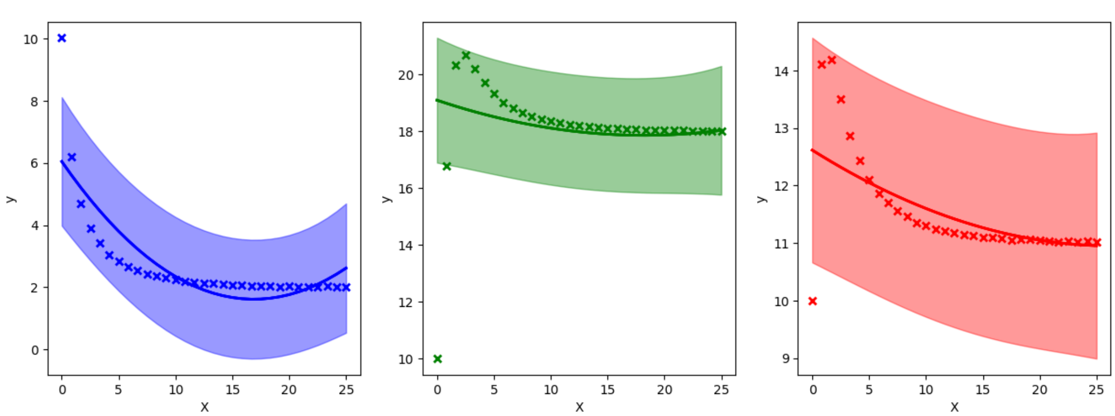
species= 3, latent_processes= 1, kernel= Linear, mean= NoneType, BIC =29469.795305646727

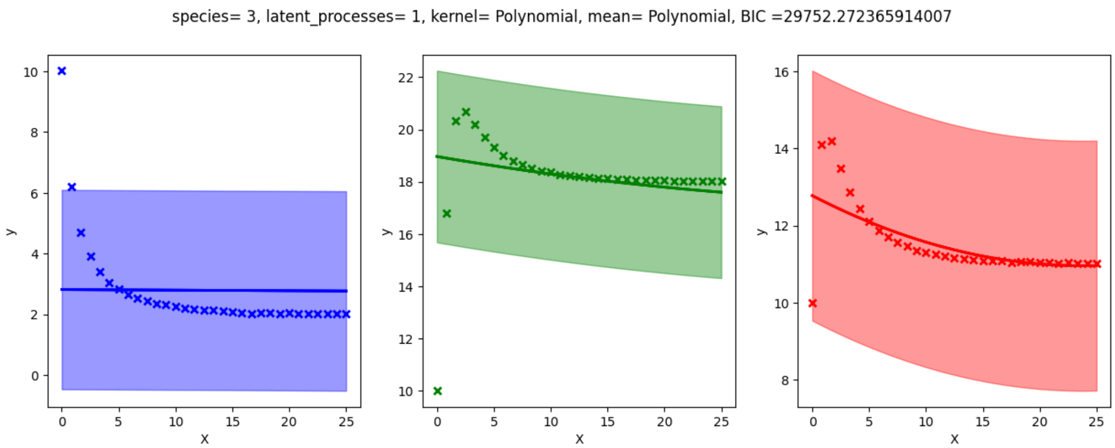


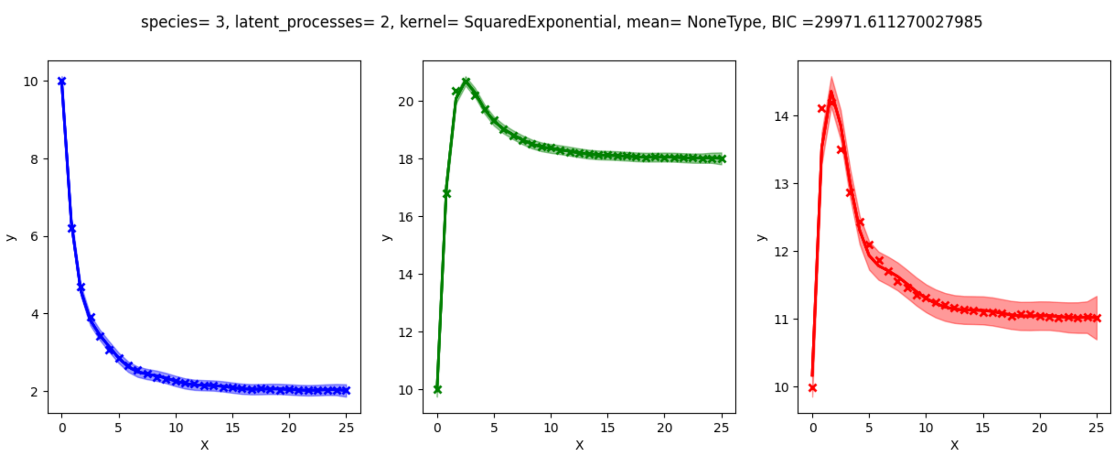
species= 3, latent_processes= 1, kernel= Linear, mean= Polynomial, BIC =29632.52825181503

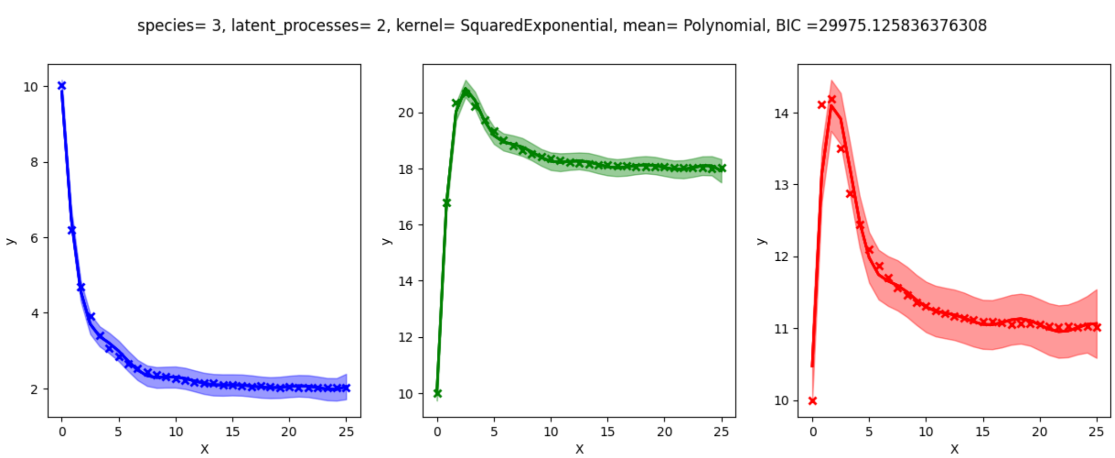


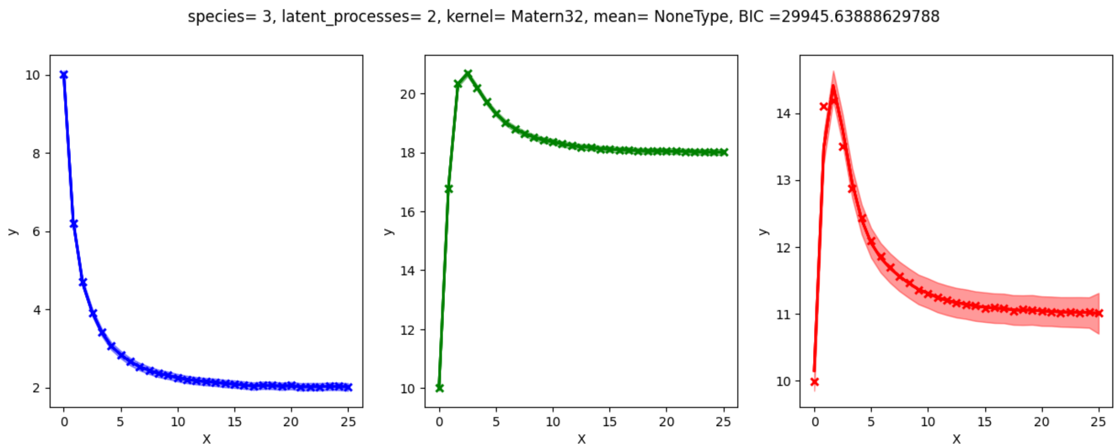
species= 3, latent_processes= 1, kernel= Polynomial, mean= NoneType, BIC =29742.92206499892

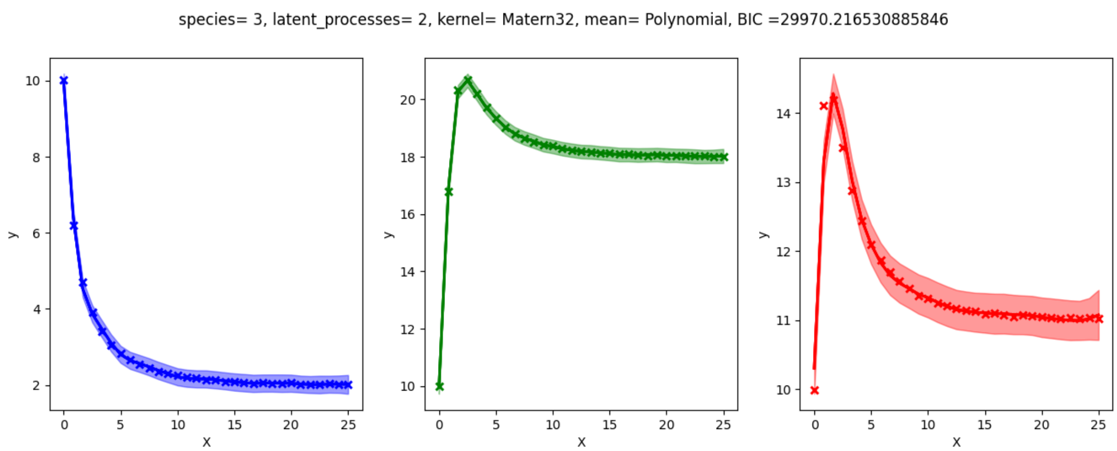


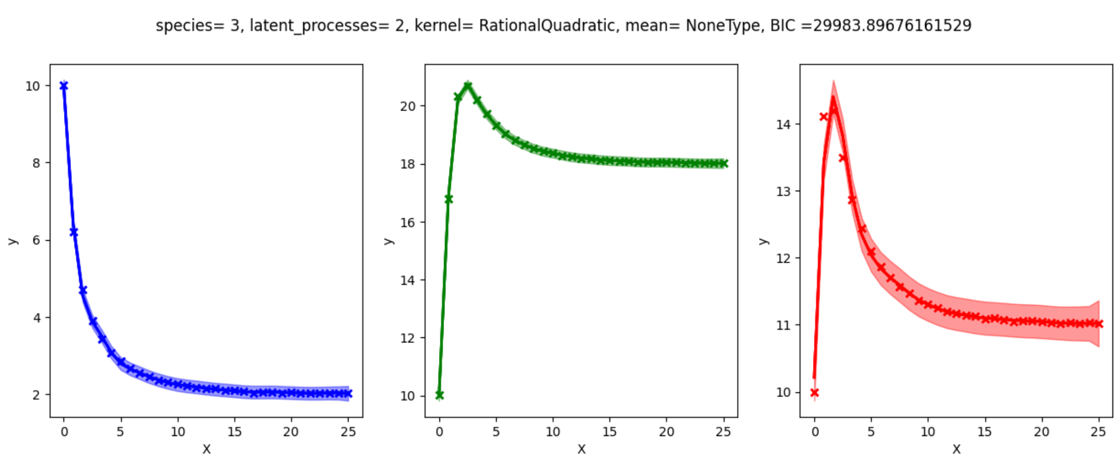

species= 3, latent_processes= 1, kernel= Polynomial, mean= Polynomial, BIC =29752.272365914007

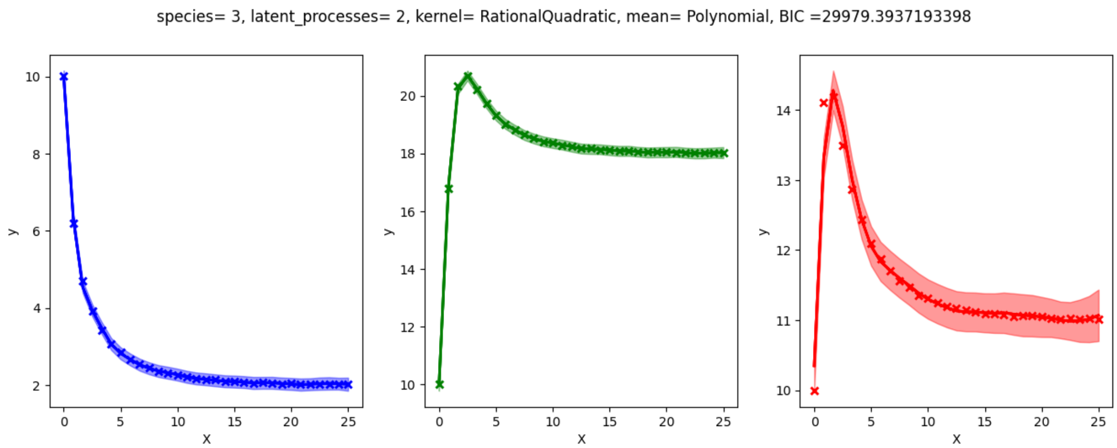

species= 3, latent_processes= 2, kernel= SquaredExponential, mean= NoneType, BIC =29971.611270027985

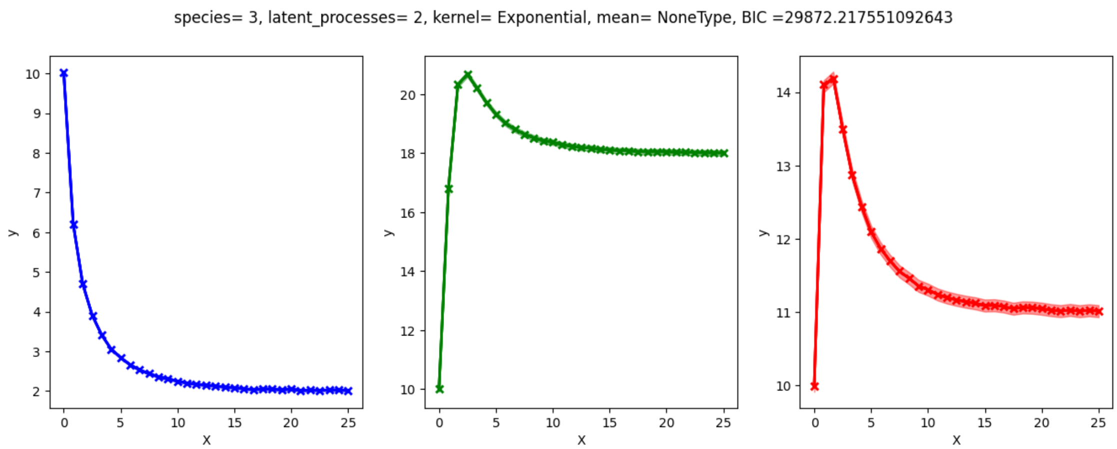

species= 3, latent_processes= 2, kernel= SquaredExponential, mean= Polynomial, BIC =29975.125836376308

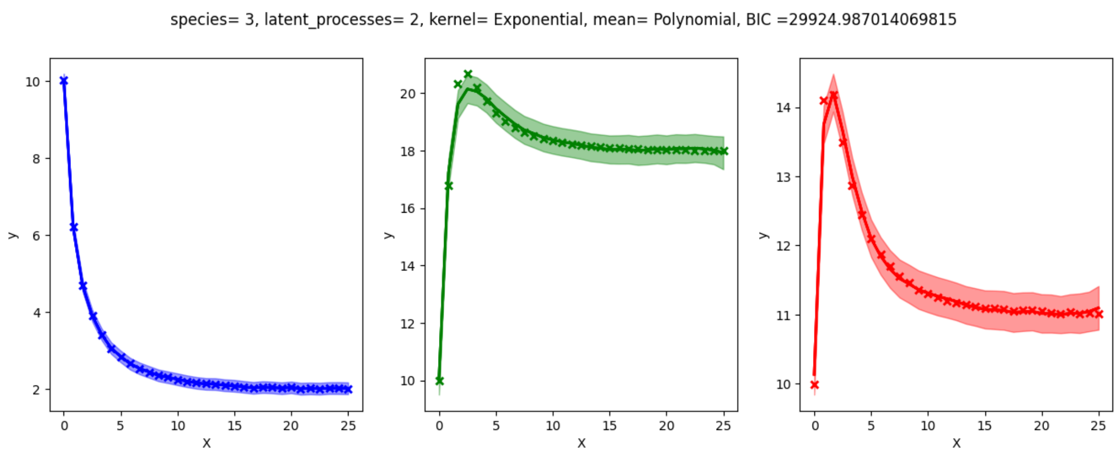

species= 3, latent_processes= 2, kernel= Matern32, mean= NoneType, BIC =29945.63888629788
y
X


species= 3, latent_processes= 2, kernel= Matern32, mean= Polynomial, BIC =29970.216530885846
y
X

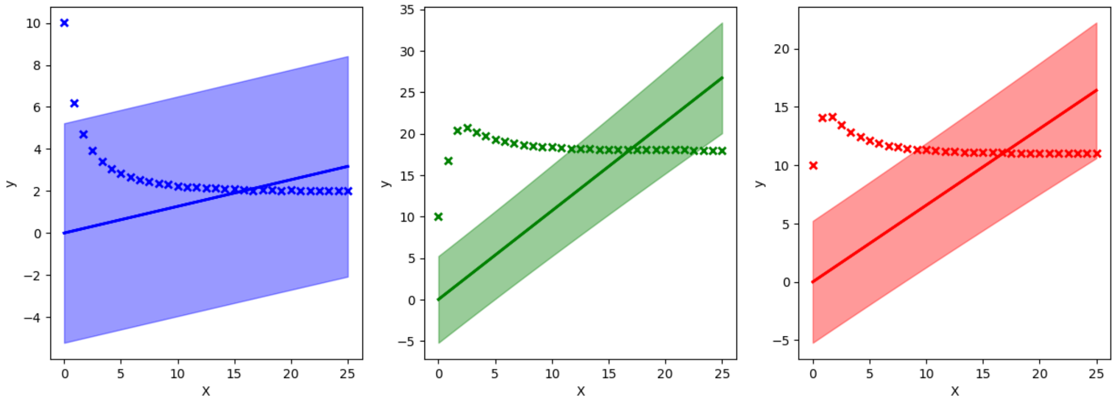

species= 3, latent_processes= 2, kernel= RationalQuadratic, mean= NoneType, BIC =29983.89676161529
y
X


species= 3, latent_processes= 2, kernel= RationalQuadratic, mean= Polynomial, BIC =29979.3937193398

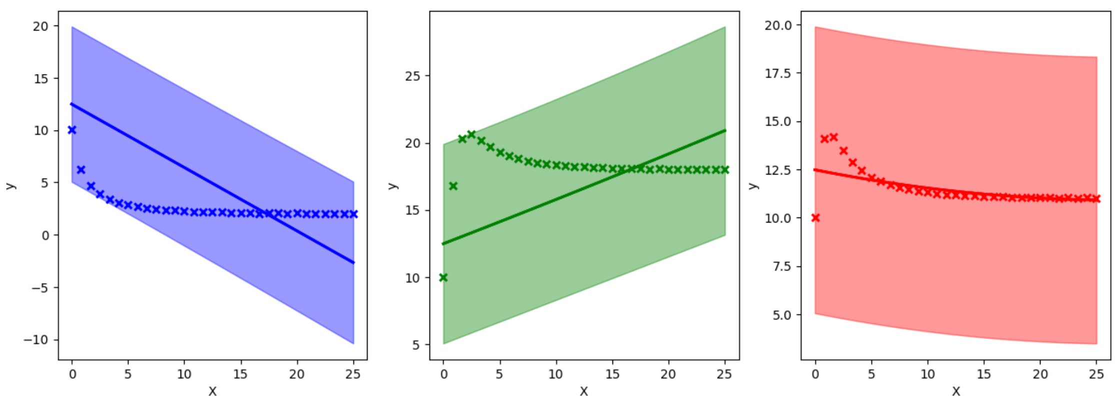

species= 3, latent_processes= 2, kernel= Exponential, mean= NoneType, BIC =29872.217551092643


species= 3, latent_processes= 2, kernel= Exponential, mean= Polynomial, BIC =29924.987014069815

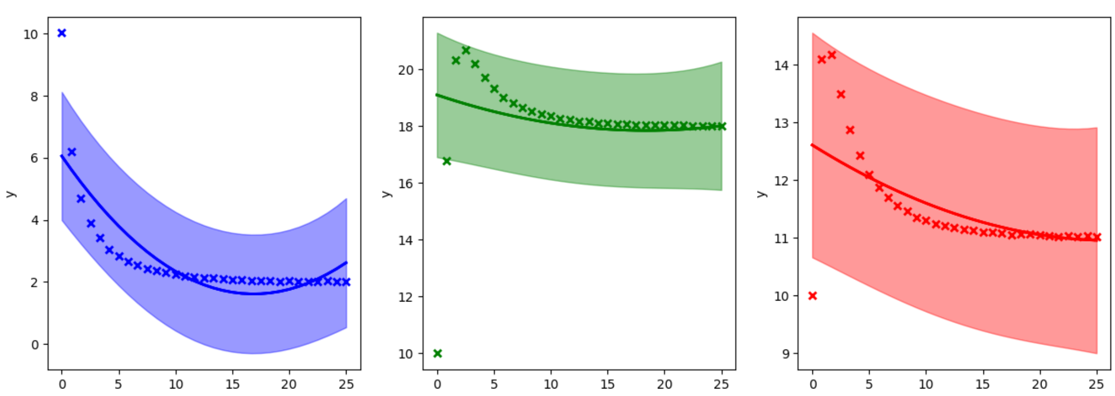
species= 3, latent_processes= 2, kernel= Linear, mean= NoneType, BIC =29480.100007819696

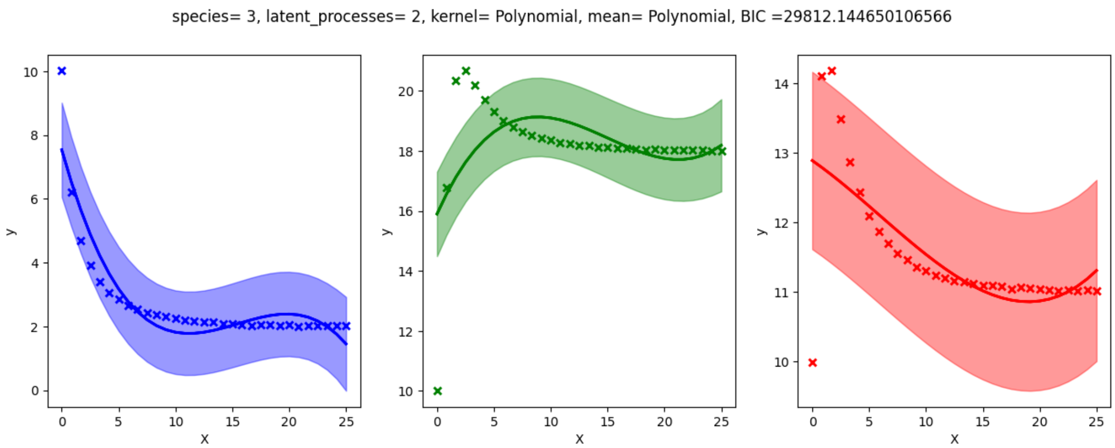


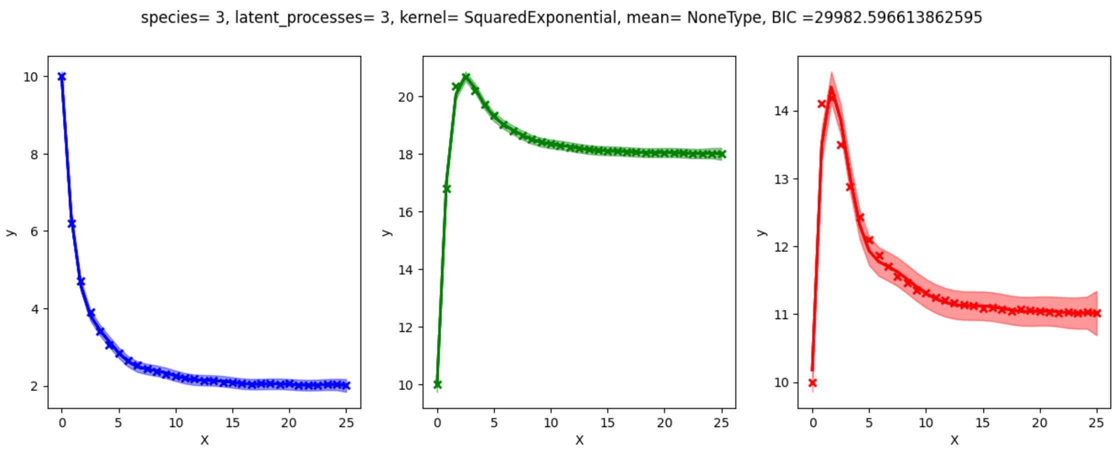
species= 3, latent_processes= 2, kernel= Linear, mean= Polynomial, BIC =29636.495799474375

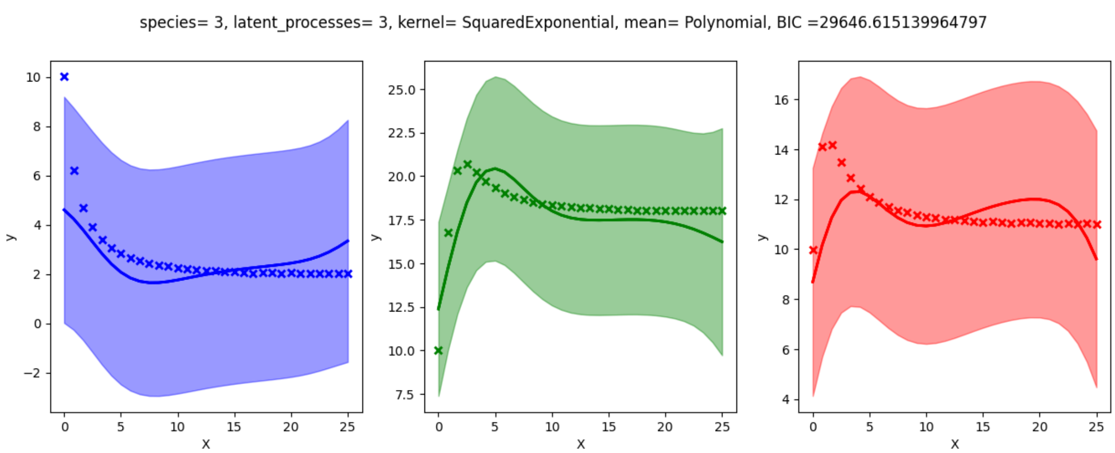


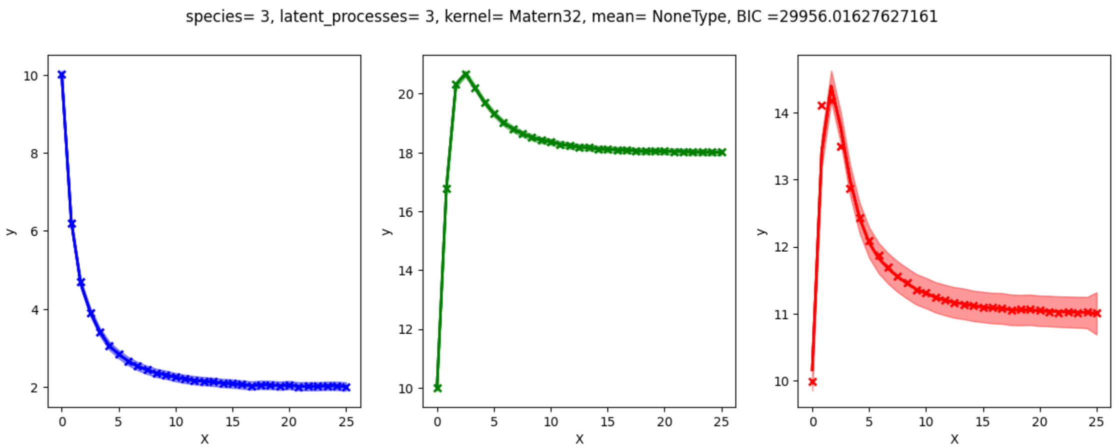
species= 3, latent_processes= 2, kernel= Polynomial, mean= NoneType, BIC =29753.226075587925

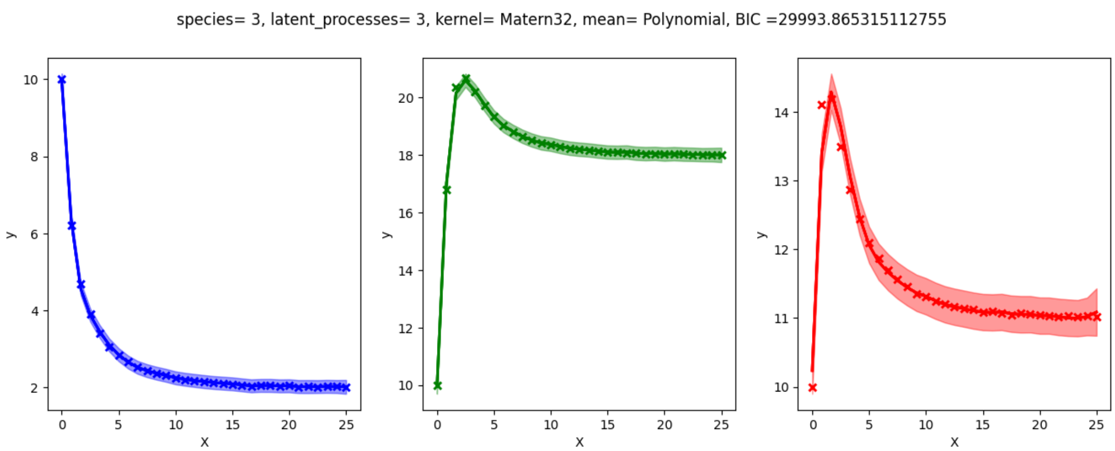


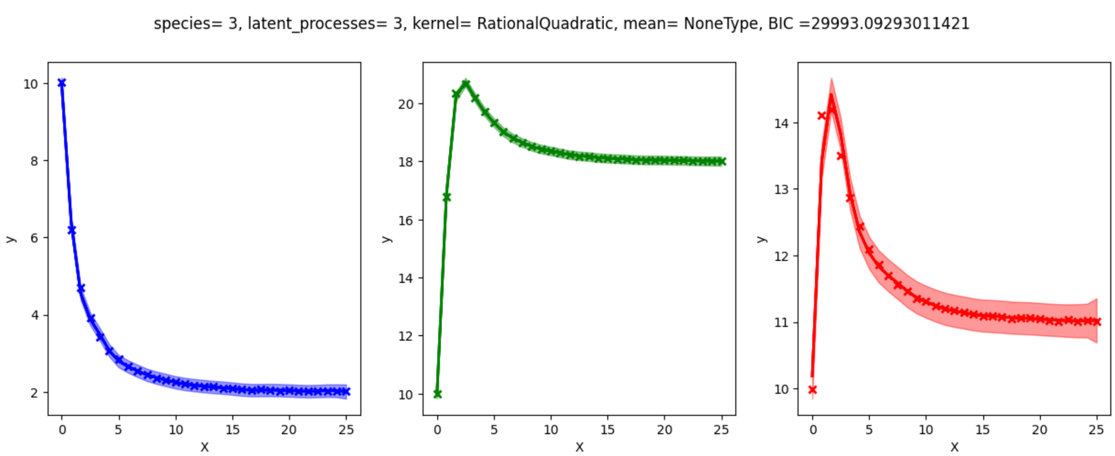

species= 3, latent_processes= 2, kernel= Polynomial, mean= Polynomial, BIC =29812.144650106566
y
x

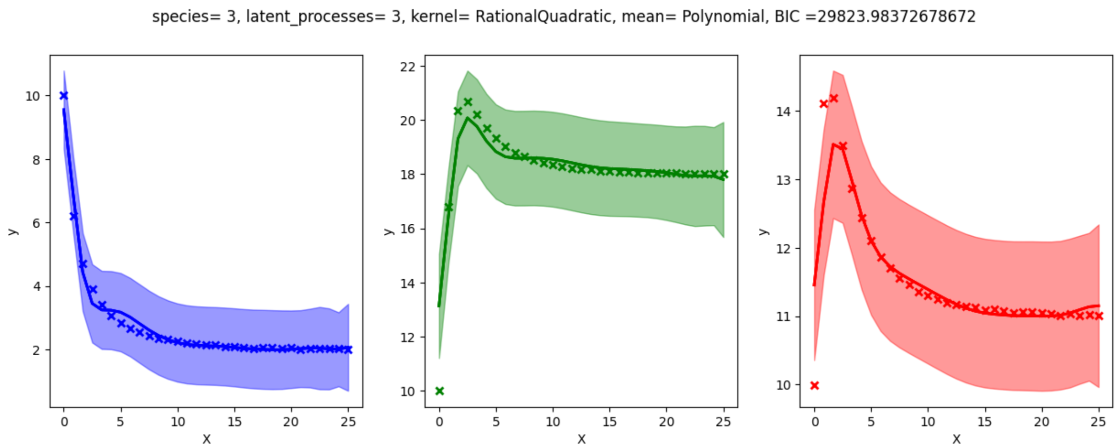

species= 3, latent_processes= 3, kernel= SquaredExponential, mean= NoneType, BIC =29982.596613862595
y
x

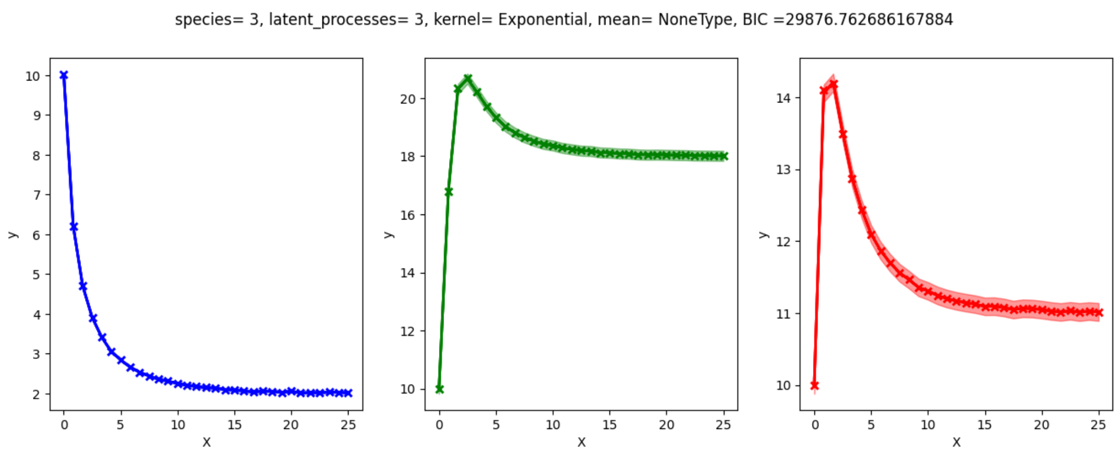

species= 3, latent_processes= 3, kernel= SquaredExponential, mean= Polynomial, BIC =29646.615139964797
y
x

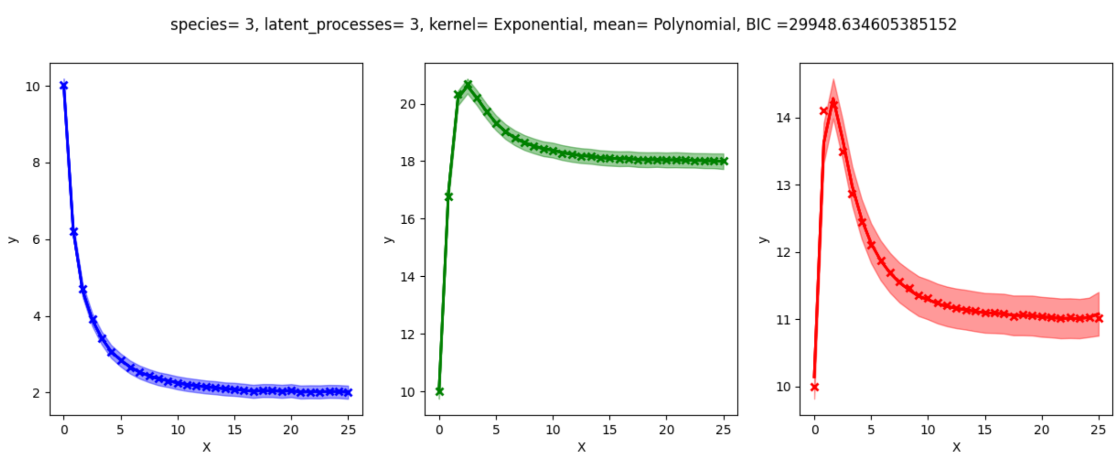

species= 3, latent_processes= 3, kernel= Matern32, mean= NoneType, BIC =29956.01627627161
y
x

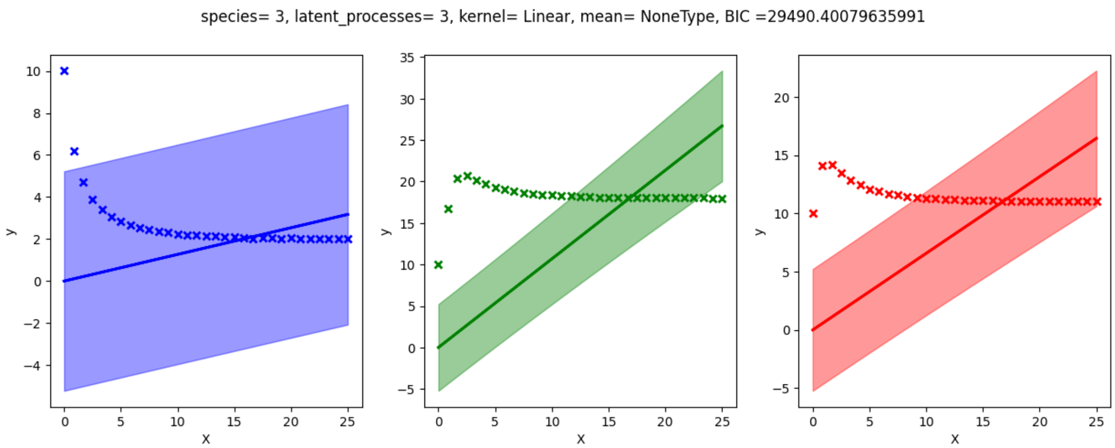

species= 3, latent_processes= 3, kernel= Matern32, mean= Polynomial, BIC =29993.865315112755
y
x

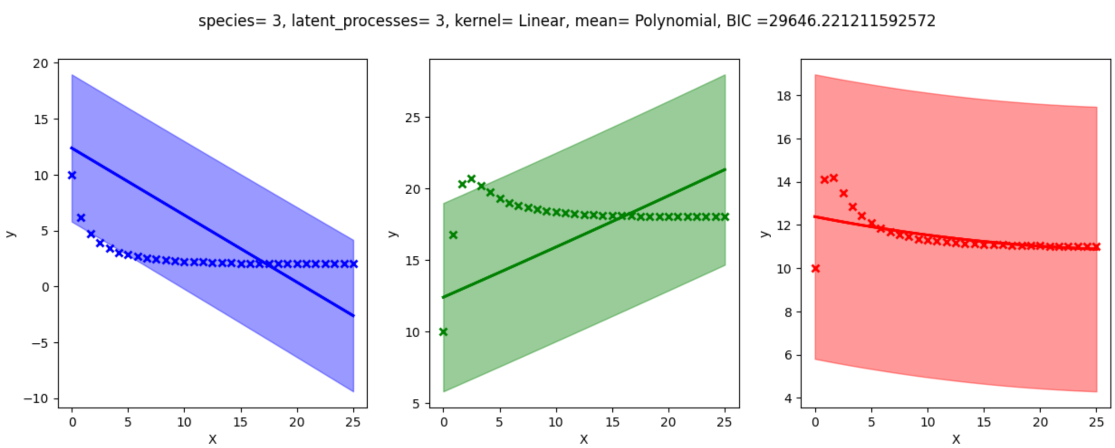

species= 3, latent_processes= 3, kernel= RationalQuadratic, mean= NoneType, BIC =29993.09293011421
y
x

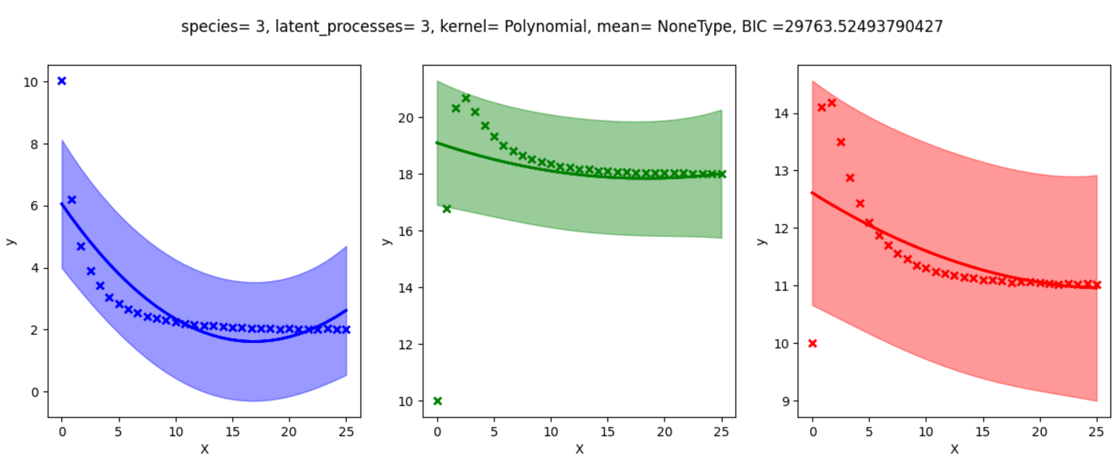

species= 3, latent_processes= 3, kernel= RationalQuadratic, mean= Polynomial, BIC =29823.98372678672

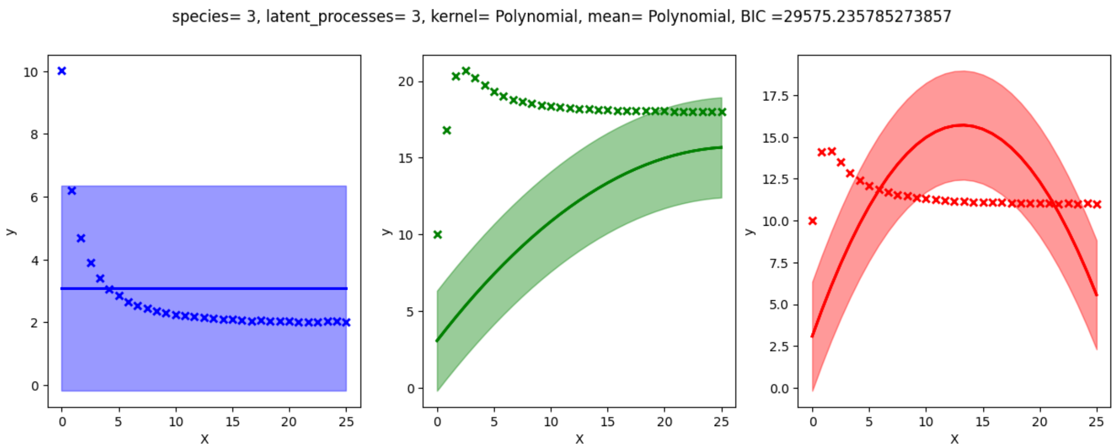

species= 3, latent_processes= 3, kernel= Exponential, mean= NoneType, BIC =29876.762686167884


species= 3, latent_processes= 3, kernel= Exponential, mean= Polynomial, BIC =29948.634605385152


species= 3, latent_processes= 3, kernel= Linear, mean= NoneType, BIC =29490.40079635991
y
x


species= 3, latent_processes= 3, kernel= Linear, mean= Polynomial, BIC =29646.221211592572
y
x


species= 3, latent_processes= 3, kernel= Polynomial, mean= NoneType, BIC =29763.52493790427
y
x




```
[ ]: print("best BIC: " + str(best_BIC))
     if 'best_L' in locals():
         print("N# latent processes: " + str(best_L))
     else:
         print("best_L is not defined")
     print("Kernel: " + str(best_K_L.__name__))
     print("Mean Function: " + str(best_M_F.__class__.__name__))
     print_summary(best_model)
```

```
best BIC: 29993.865315112755
N# latent processes: 3
Kernel: Matern32
Mean Function: Polynomial
```

```
<IPython.core.display.HTML object>
```