Gaussian process in a fully Bayesian mode

Enable brov
Settings to
executions

OK

This notebook demonstrates an application of the Gaussian process class in GPax package to a 1D problem.

Prepared by Maxim Ziatdinov (2024)

Edited by Josiah Park for the present purposes (2024)

Gaussian process (GP) is a powerful tool for reconstructing with quantified uncertainty an unknown ("black-box") function over a low-dimensional parameter space from sparse measurements. Formally, given a set of observed data points $(x_i, y_i)_{i=1,...,N}$ and assuming normally distributed observation noise ε, the GP aims to reconstruct the function $f(x)$ such as $y_i = f(x_i) + \varepsilon_i$, with f sampled from a multivariate normal prior distribution, $f \sim MVN(0; K(x_i, x_j))$. The functional form of the kernel K is chosen prior to the experiment, and its hyperparameters are inferred from the observations using either Markov chain Monte Carlo methods or stochastic variational inference.

Install & Import

Install GPax package:

```
!pip install -q gpax
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 103.4/103.4 kB 2.9 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 371.7/371.7 kB 6.3 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 351.1/351.1 kB 6.0 MB/s eta 0:00:00
```

```
!pip3 install openmeteo-requests
!pip3 install requests-cache retry-requests numpy pandas
import openmeteo_requests

import requests_cache
import pandas as pd
from retry_requests import retry

# Setup the Open-Meteo API client with cache and retry on error
cache_session = requests_cache.CachedSession('.cache', expire_after = 3600)
retry_session = retry(cache_session, retries = 5, backoff_factor = 0.2)
openmeteo = openmeteo_requests.Client(session = retry_session)

# Make sure all required weather variables are listed here
# The order of variables in hourly or daily is important to assign them correctly below
url = "https://api.open-meteo.com/v1/forecast"
params = {
    "latitude": 37.8715,
    "longitude": 122.273,
    "current": "surface_pressure",
    "hourly": "surface_pressure",
    "start_date": "2024-07-22",
    "end_date": "2024-10-14"
}
responses = openmeteo.weather_api(url, params=params)

# Process first location. Add a for-loop for multiple locations or weather models
response = responses[0]
print(f"Coordinates {response.Latitude()}°N {response.Longitude()}°E")
print(f"Elevation {response.Elevation()} m asl")
print(f"Timezone {response.Timezone()} {response.TimezoneAbbreviation()}")
print(f"Timezone difference to GMT+0 {response.UtcOffsetSeconds()} s")

# Current values. The order of variables needs to be the same as requested.
current = response.Current()
current_surface_pressure = current.Variables(0).Value()

print(f"Current time {current.Time()}")
print(f"Current surface_pressure {current_surface_pressure}")

# Process hourly data. The order of variables needs to be the same as requested.
hourly = response.Hourly()
hourly_surface_pressure = hourly.Variables(0).ValuesAsNumpy()
```

```
hourly_data = {"date": pd.date_range(
    start = pd.to_datetime(hourly.Time(), unit = "s", utc = True),
    end = pd.to_datetime(hourly.TimeEnd(), unit = "s", utc = True),
    freq = pd.Timedelta(seconds = hourly.Interval()),
    inclusive = "left"
)}
hourly_data["surface_pressure"] = hourly_surface_pressure

hourly_dataframe = pd.DataFrame(data = hourly_data)
print(hourly_dataframe)
hourly_dataframe.shape
```

Enable brov
Settings to
executions

```
Requirement already satisfied: openmeteo-requests in /usr/local/lib/python3.10/dist-packages (1.3.0)
Requirement already satisfied: openmeteo-sdk>=1.4.0 in /usr/local/lib/python3.10/dist-packages (from openmeteo-requests) (1.17.0)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from openmeteo-requests) (2.32.3)
Requirement already satisfied: flatbuffers>=24.0.0 in /usr/local/lib/python3.10/dist-packages (from openmeteo-sdk>=1.4.0->openmeteo-requ
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->openmeteo-requests) (
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->openmeteo-requests) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->openmeteo-requests) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->openmeteo-requests) (2024.8
Requirement already satisfied: requests-cache in /usr/local/lib/python3.10/dist-packages (1.2.1)
Requirement already satisfied: retry-requests in /usr/local/lib/python3.10/dist-packages (2.0.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (1.26.4)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (2.2.2)
Requirement already satisfied: attrs>=21.2 in /usr/local/lib/python3.10/dist-packages (from requests-cache) (24.2.0)
Requirement already satisfied: cattrs>=22.2 in /usr/local/lib/python3.10/dist-packages (from requests-cache) (24.1.2)
Requirement already satisfied: platformdirs>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests-cache) (4.3.6)
Requirement already satisfied: requests>=2.22 in /usr/local/lib/python3.10/dist-packages (from requests-cache) (2.32.3)
Requirement already satisfied: url-normalize>=1.4 in /usr/local/lib/python3.10/dist-packages (from requests-cache) (1.4.3)
Requirement already satisfied: urllib3>=1.25.5 in /usr/local/lib/python3.10/dist-packages (from requests-cache) (2.2.3)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages (from pandas) (2024.2)
Requirement already satisfied: exceptiongroup>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from cattrs>=22.2->requests-cache) (1.2
Requirement already satisfied: typing-extensions!=4.6.3,>=4.1.0 in /usr/local/lib/python3.10/dist-packages (from cattrs>=22.2->requests-
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.2->pandas) (1.16.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests>=2.22->requests-cache)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests>=2.22->requests-cache) (3.10)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests>=2.22->requests-cache) (2024
Coordinates 37.875°N 122.25°E
Elevation 0.0 m asl
Timezone None None
Timezone difference to GMT+0 0 s
Current time 1728955800
Current surface_pressure 1019.2000122070312
                          date  surface_pressure
0    2024-07-22 00:00:00+00:00       1001.200012
1    2024-07-22 01:00:00+00:00       1001.400024
2    2024-07-22 02:00:00+00:00       1000.599976
3    2024-07-22 03:00:00+00:00       1000.000000
4    2024-07-22 04:00:00+00:00        999.599976
...                        ...               ...
2035 2024-10-14 19:00:00+00:00       1016.099976
2036 2024-10-14 20:00:00+00:00       1016.099976
2037 2024-10-14 21:00:00+00:00       1016.400024
2038 2024-10-14 22:00:00+00:00       1017.000000
2039 2024-10-14 23:00:00+00:00       1018.000000

[2040 rows x 2 columns]
(2040, 2)
```

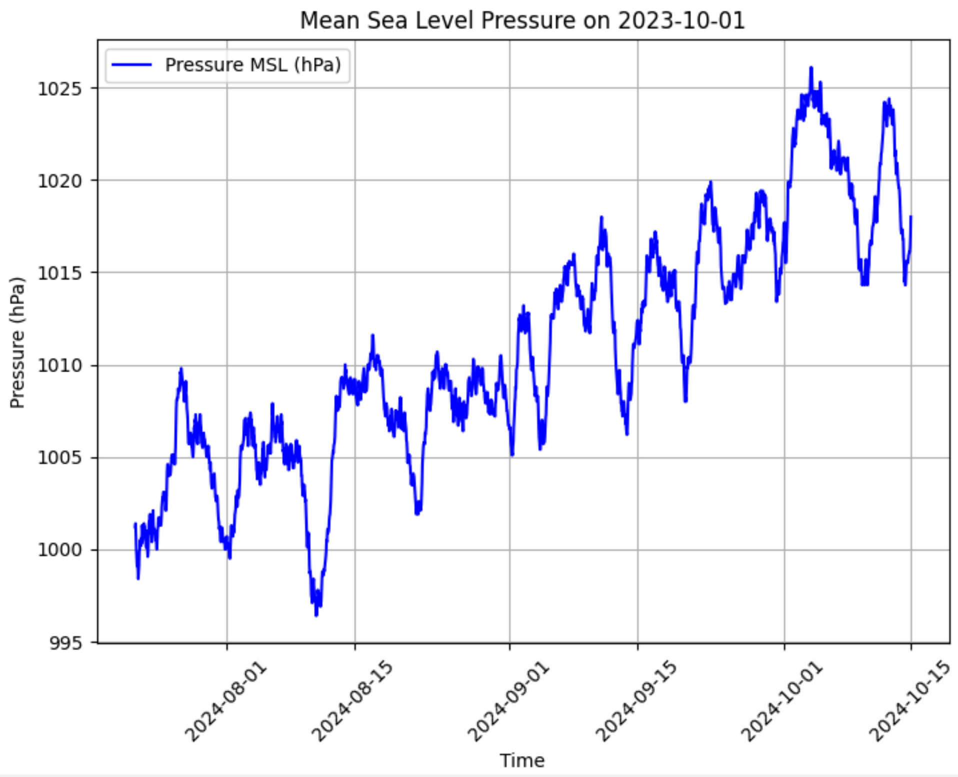
Import needed packages:

```
!pip install matplotlib
import matplotlib.pyplot as plt

# Plotting the data
plt.figure(figsize=(14, 6))

# Plotting Mean Sea Level Pressure
plt.subplot(1, 2, 1)
plt.plot(hourly_dataframe['date'], hourly_dataframe['surface_pressure'], color='blue', label='Pressure MSL (hPa)')
plt.title('Mean Sea Level Pressure on 2023-10-01')
plt.xlabel('Time')
plt.ylabel('Pressure (hPa)')
plt.xticks(rotation=45)
plt.grid()
plt.legend()
```

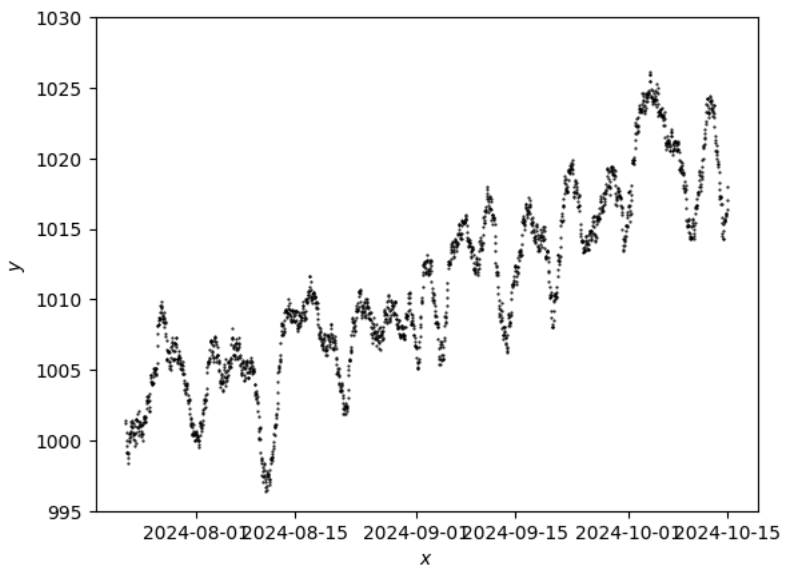
```
# Adjust layout
plt.tight_layout()
plt.show()
```

Enable brow
Settings to
executions

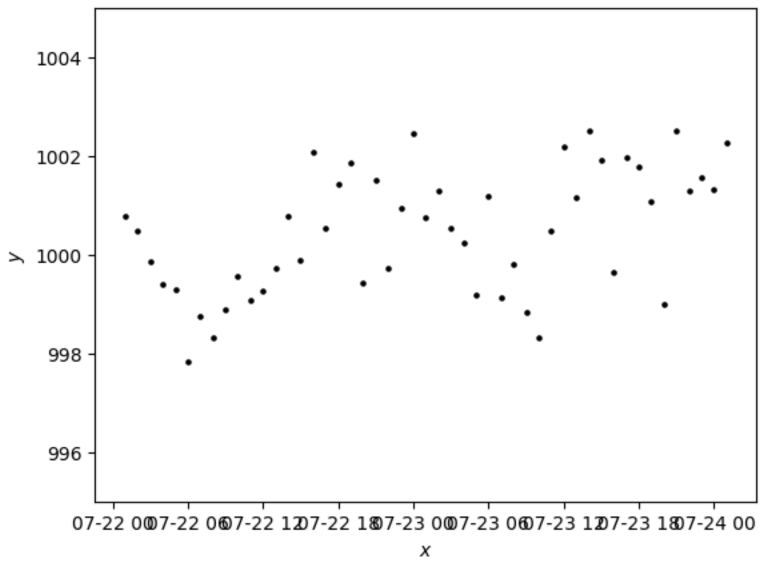
```
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (3.7.1)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (1.3.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (4.54.1)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (1.4.7)
Requirement already satisfied: numpy>=1.20 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (1.26.4)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (24.1)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (10.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (3.1.4)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplotlib) (1.16.0)
```



```
import numpy as np
NOISE_LEVEL = 1.0   # Example noise level
NUM_INIT_POINTS = 2040  # Example number of initial points

# Generate random samples from a normal distribution
random_values = np.random.normal(0., NOISE_LEVEL, NUM_INIT_POINTS)

# Create a Pandas DataFrame
df = pd.DataFrame(random_values, columns=['surface_pressure'])
x1=pd.DataFrame(hourly_dataframe['date'])
y=pd.DataFrame(hourly_dataframe['surface_pressure'])
yy=df+y


plt.figure(dpi=100)
plt.xlabel("$x$")
plt.ylabel("$y$")
plt.scatter(x1, y, marker='o', c='k', zorder=1,  s=.25,  label='No noise')
plt.ylim(995, 1030);
```



```
plt.figure(dpi=100)
plt.xlabel("$x$")
plt.ylabel("$y$")
plt.scatter(x1[1:50], yy[1:50], marker='o', c='k', zorder=1,  s=5,  label='Noisy observations')
plt.ylim(995, 1005);
```



```
import gpax
import numpy as np
import matplotlib.pyplot as plt

gpax.utils.enable_x64()  # enable double precision
```

```
yy=yy.to_numpy()
```

```
x1 = pd.DataFrame(hourly_dataframe['date'])
x1['unix_time'] = hourly_dataframe['date'].astype('int64') // 10**9  # Convert to seconds
x1['unix_time'] = x1['unix_time'] / 10**9
```

```
x1=x1.to_numpy()
```

```
x1=x1[:,1]
```

```
xx1=jnp.array(x1.astype(np.float32))
```

```
print(xx1)
```

```
[1.7216064 1.72161   1.7216136 ... 1.7289397 1.7289432 1.7289468]
```

Enable brov
Settings to
executions

Standard GP

Next, we initialize and train a GP model. We are going to use an RBF kernel, $k_{RBF} = \sigma exp(-\frac{||x_i - x_j||^2}{2l^2})$, which is a "go-to" kernel functions in GP.

```
import jax.numpy as jnp
# Get random number generator keys for training and prediction
key1, key2 = gpax.utils.get_keys()
```

```
gp_model.fit(key1, jnp.array(xx1[::2]), jnp.array(yy[::2]), num_chains=1, num_warmup=20, num_samples=20)
```

```
sample: 100%|██████████| 40/40 [02:15<00:00,  3.38s/it, 7 steps of size 6.81e-02. acc. prob=0.87]
                 mean       std    median      5.0%     95.0%     n_eff     r_hat
k_length[0]      0.07      0.09      0.00      0.00      0.22      3.68      1.61
    k_scale  77646.89  23059.94  76252.73  37432.96 102534.61     12.70      0.99
      noise      9.75      0.78      9.45      8.66     11.03      3.77      1.72


---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-97-de6fb658a02e> in <cell line: 8>()
      6 gp_model.fit(key1, jnp.array(xx1[::2]), jnp.array(yy[::2]), num_chains=1, num_warmup=20, num_samples=20)
      7
----> 8 current_loss = gp_model.loss()  # Replace with your method to compute loss

AttributeError: 'ExactGP' object has no attribute 'loss'
```

Next steps: Explain error

"Now let's use our trained model to obtain a probabilistic prediction of function values on new ("test") data. In the fully Bayesian mode, we get a pair of predictive mean and covariance, μ^i_* and Σ^i_*, for each i-th HMC sample with kernel parameters θ,

$$\mu^i_* = K(X_*, X|\theta^i)K(X, X|\theta^i)^{-1}y$$
$$\Sigma^i_* = K(X_*, X_*|\theta^i) - K(X_*, X|\theta^i)K(X, X|\theta^i)^{-1}K(X, X_*|\theta^i)$$

The `.predict()` method returns the center of the mass of all the predictive means,

$$\mu^{post}_* = \frac{1}{L}\sum_{i=1}^{L} \mu^i_*,$$

which corresponds to the `posterior_mean` in the code cell below, and samples

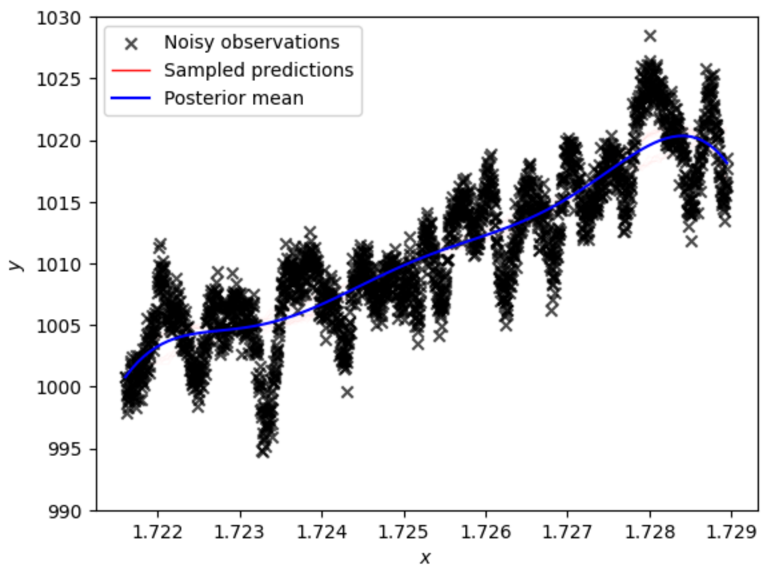
$$f^i_* \sim MVNormal(\mu^i_*, \Sigma^i_*)$$

from multivariate normal distributions for all the pairs of predictive means and covariances (`f_samples` in the code cell below). Note that model noise is absorbed into the kernel computation function."

```
# Prepare test inputs
X_test = np.linspace(1.7216064, 1.7289468, 100)
# Get the GP prediction. Here n stands for the number of samples from each MVNormal distribution
# (the total number of MVNormal distributions is equal to the number of HMC samples)
posterior_mean, f_samples = gp_model.predict(key2, X_test, n=200)
```

Plot the obtained results:

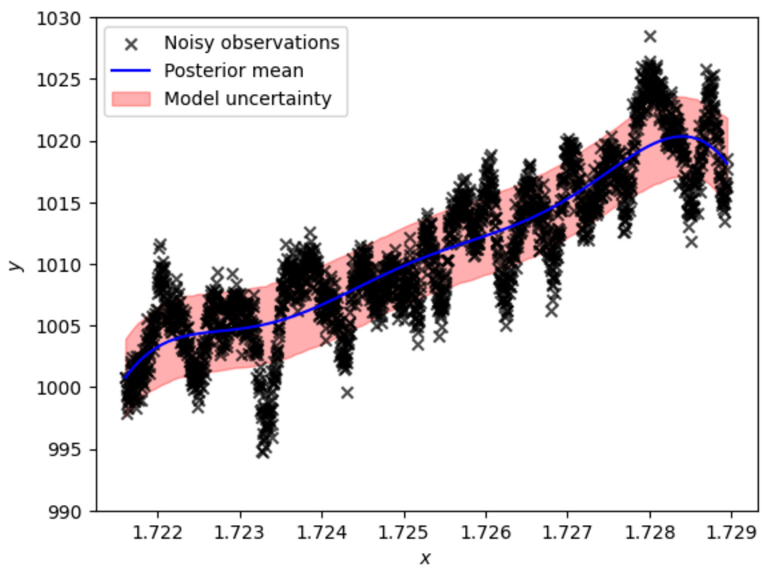
```
_, ax = plt.subplots(dpi=100)
ax.set_xlabel("$x$")
ax.set_ylabel("$y$")
ax.scatter(jnp.array(xx1), jnp.array(yy), marker='x', c='k', zorder=1, label="Noisy observations", alpha=0.7)
for y1 in f_samples:
    ax.plot(X_test, y1.mean(0), lw=.1, zorder=0, c='r', alpha=.1)
l, = ax.plot(X_test, f_samples[0].mean(0), lw=1, c='r', alpha=1, label="Sampled predictions")
ax.plot(X_test, posterior_mean, lw=1.5, zorder=1, c='b', label='Posterior mean')
ax.legend(loc='upper left')
l.set_alpha(0)
ax.set_ylim(990, 1030);
```



Enable brow
Settings to
executions

We can see there is a relatively large uncertainty - defined by the dispersion in sampled predictions - in GP prediction between -0.75 and -0.25 where no measurements are available. It is also common in the GP literature to draw the GP uncertainty as a 2σ shaded region around the GP posterior mean:

```
_, ax = plt.subplots(dpi=100)
ax.set_xlabel("$x$")
ax.set_ylabel("$y$")
ax.scatter(jnp.array(xx1), jnp.array(yy), marker='x', c='k', zorder=2, label="Noisy observations", alpha=0.7)
ax.plot(X_test, posterior_mean, lw=1.5, zorder=2, c='b', label='Posterior mean')
ax.fill_between(X_test,
                posterior_mean - f_samples.std(axis=(0,1)),
                posterior_mean + f_samples.std(axis=(0,1)),
                color='r', alpha=0.3, label="Model uncertainty")
ax.legend(loc='upper left')
ax.set_ylim(990, 1030);
```


Noisy observations
Posterior mean
Model uncertainty
y
x