More PyMC

Lecture 20

Dr. Colin Rundel

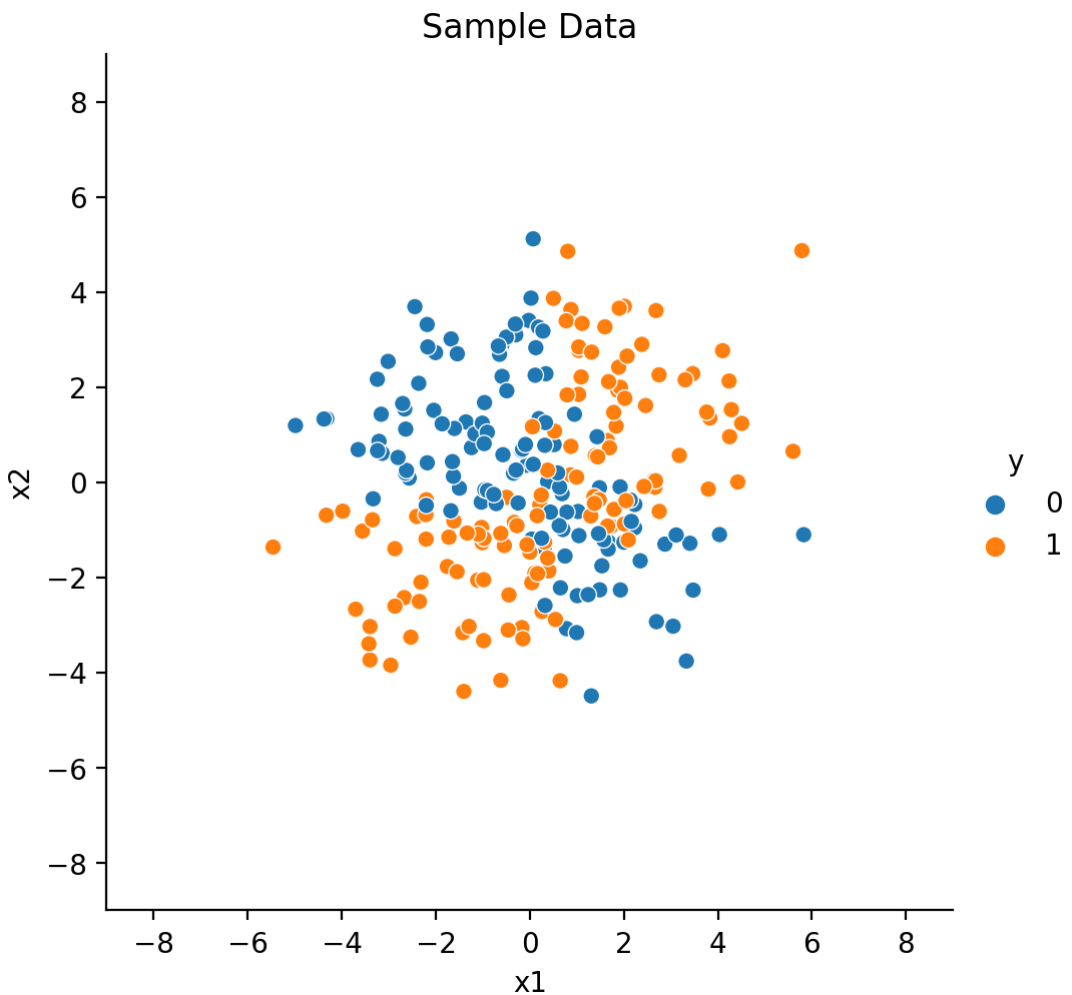
Demo 1 - Logistic Regression

Based on PyMC Out-Of-Sample Predictions example

Data

	x1	x2	y
0	-3.207674	0.859021	0
1	0.128200	2.827588	0
2	1.481783	-0.116956	0
3	0.305238	-1.378604	0
4	1.727488	-0.926357	1
..	...	...	..
245	-2.182813	3.314672	0
246	-2.362568	2.078652	0
247	0.114571	2.249021	0
248	2.093975	-1.212528	1
249	1.241667	-2.363412	0

[250 rows x 3 columns]



Test-train split

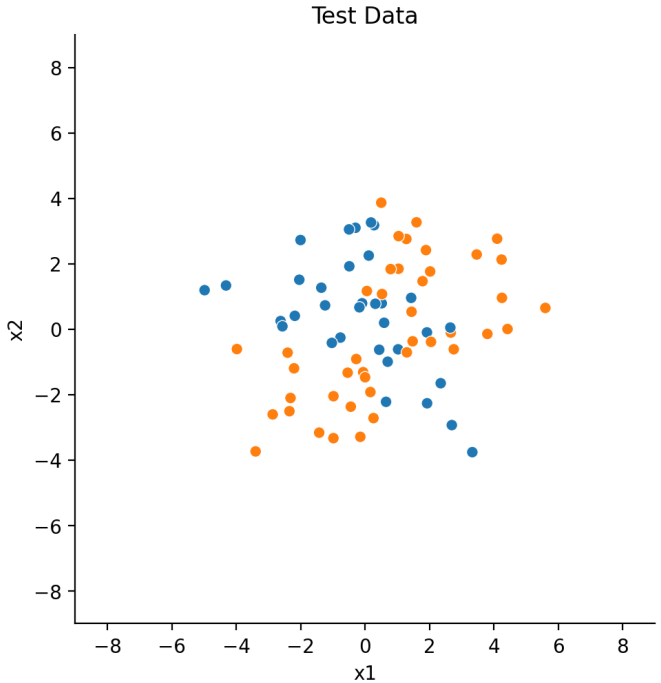
```
from sklearn.model_selection import train_test_split

y, X = patsy.dmatrices("y ~ x1 * x2", data=df)

X_lab = X.design_info.column_names
y_lab = y.design_info.column_names
y = np.asarray(y).flatten()
X = np.asarray(X)

X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7)
```


Training Data
x2
x1
8
6
4
2
0
−2
−4
−6
−8
−8
−6
−4
−2
0
2
4
6
8

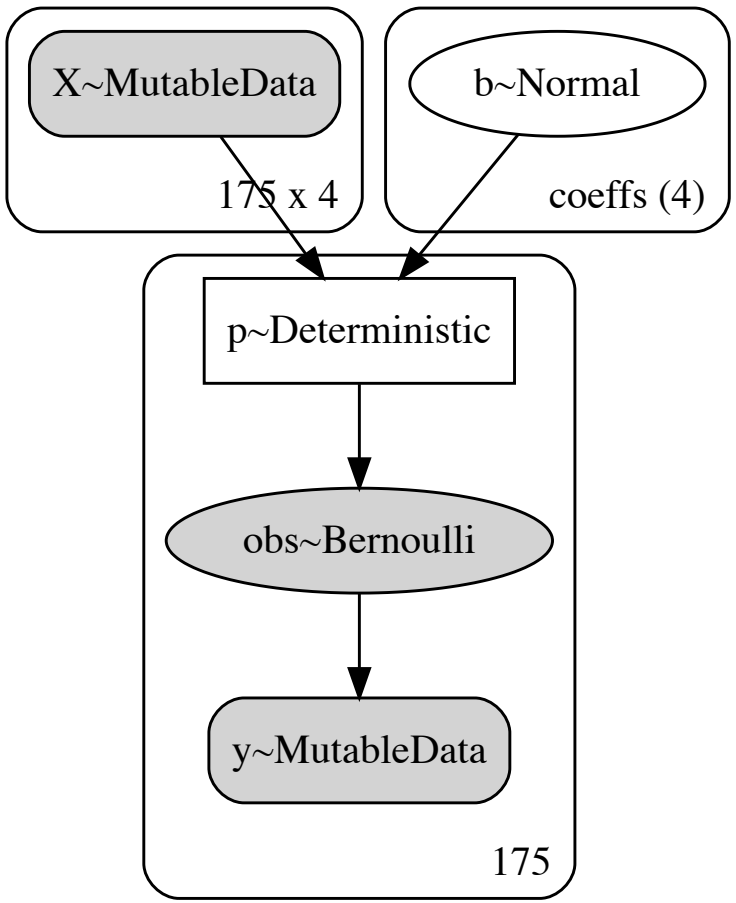

Test Data
x2
x1
8
6
4
2
0
−2
−4
−6
−8
−8
−6
−4
−2
0
2
4
6
8

Model

```
with pm.Model(coords = {"coeffs": X_lab}) as model:
    # data containers
    X = pm.MutableData("X", X_train)
    y = pm.MutableData("y", y_train)
    # priors
    b = pm.Normal("b", mu=0, sigma=3, dims="coeffs")
    # linear model
    mu = X @ b
    # link function
    p = pm.Deterministic("p", pm.math.invlogit(mu))
    # likelihood
    obs = pm.Bernoulli("obs", p=p, observed=y)
```

Visualizing models

```
1  pm.model_to_graphviz(model)
```



Fitting

```
with model:
    post = pm.sample(progressbar=False, random_seed=1234)
```

```
Auto-assigning NUTS sampler...
Initializing NUTS using jitter+adapt_diag...
Multiprocess sampling (4 chains in 4 jobs)
NUTS: [b]
Sampling 4 chains for 1_000 tune and 1_000 draw iterations (4_000 + 4_000 draws total) took 1 seconds.
```

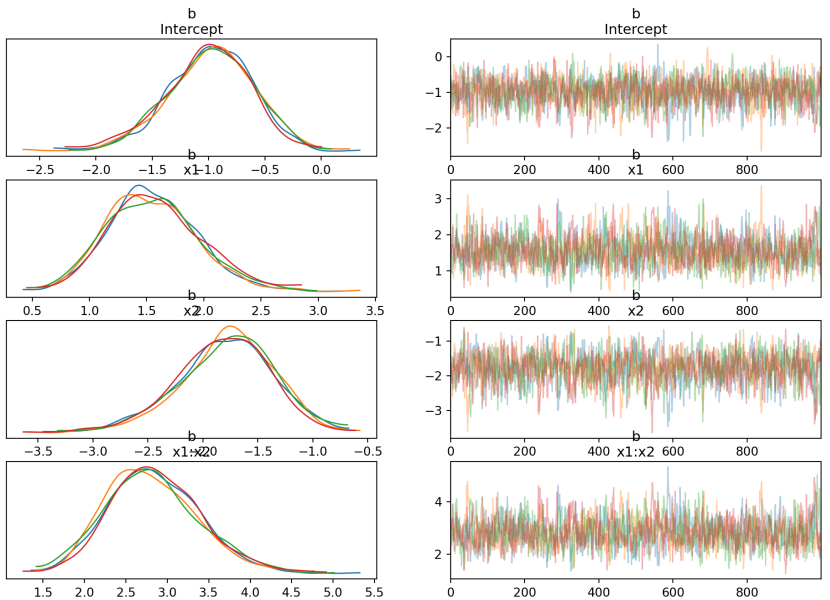
```
az.summary(post)
```

	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
b[Intercept]	-0.989	0.380	-1.682	-0.255	0.011	0.008	1277.0	1474.0	1.0
b[x1]	1.561	0.408	0.789	2.303	0.013	0.009	1066.0	1027.0	1.0
b[x2]	-1.794	0.432	-2.645	-1.014	0.014	0.010	1013.0	1331.0	1.0
b[x1:x2]	2.833	0.555	1.841	3.911	0.018	0.013	952.0	1057.0	1.0
p[0]	0.053	0.034	0.006	0.120	0.001	0.001	1632.0	1897.0	1.0
...	...	...	...	...	...	...	...	...	...
p[170]	0.000	0.001	0.000	0.001	0.000	0.000	895.0	1025.0	1.0
p[171]	0.999	0.003	0.995	1.000	0.000	0.000	1213.0	1520.0	1.0
p[172]	0.002	0.004	0.000	0.008	0.000	0.000	883.0	964.0	1.0
p[173]	1.000	0.000	1.000	1.000	0.000	0.000	1036.0	1352.0	1.0
p[174]	0.840	0.135	0.572	0.997	0.002	0.001	4483.0	3159.0	1.0

[179 rows x 9 columns]

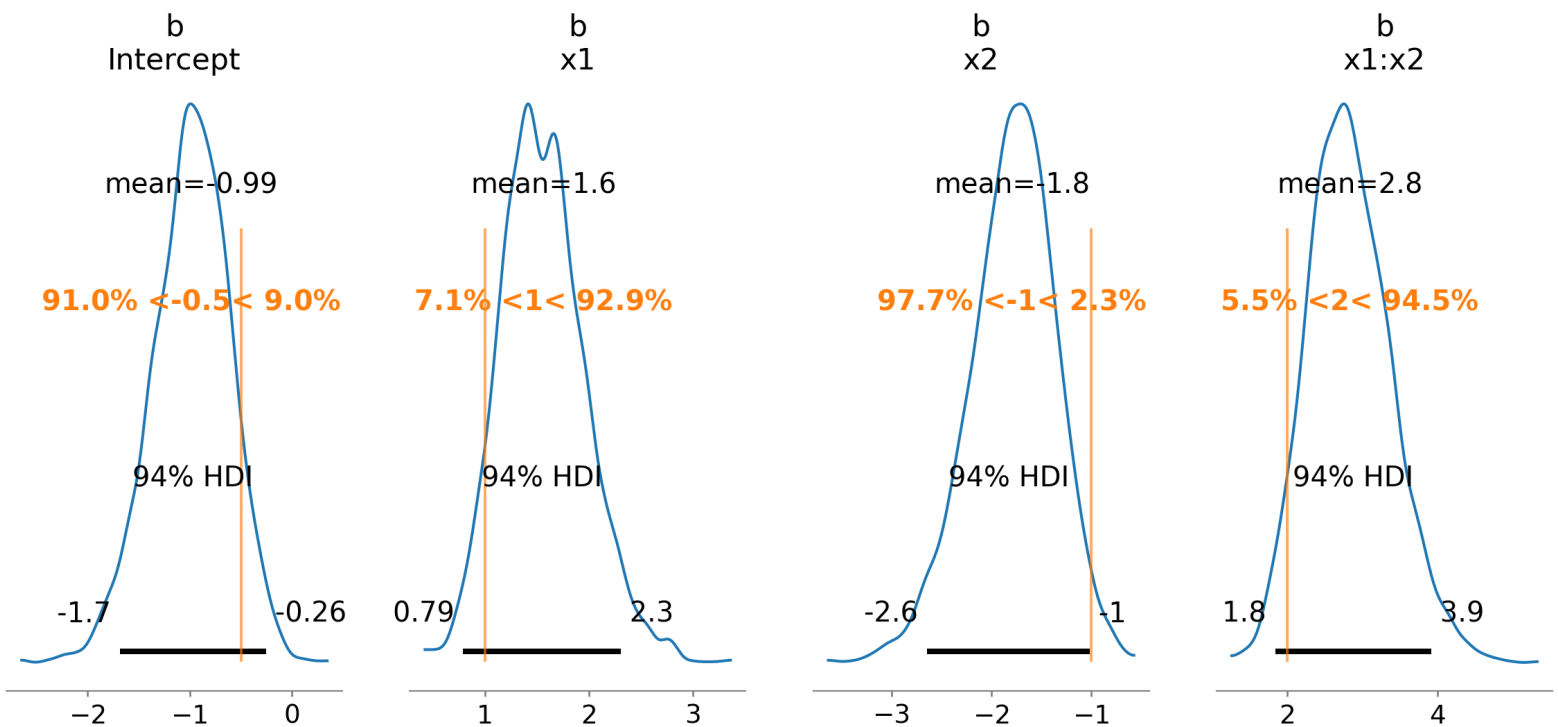
Trace plots

```
1  ax = az.plot_trace(post, var_names="b", compact=False)
2  plt.show()
```



Posterior plots

```
ax = az.plot_posterior(
    post, var_names=["b"], ref_val=[intercept, beta_x1, beta_x2, beta_interaction], figsize=(15, 6)
)
plt.show()
```



Out-of-sample predictions

```
1 post
```

arviz.InferenceData

- posterior
- sample_stats
- observed_data
- constant_data

```
1 with model:
2   pm.set_data({"X": X_test, "y": y_test})
3   post = pm.sample_posterior_predictive(
4     post, progressbar=False, var_names=["obs", "
5     extend_inferencedata = True
6   )
```

```
Sampling: [obs]
```

```
1 post
```

arviz.InferenceData

- posterior
- posterior_predictive
- sample_stats
- observed_data
- constant_data

Posterior predictive summary

```
az.summary(
  post.posterior_predictive
)
```

	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
obs[0]	0.906	0.292	0.000	1.000	0.005	0.003	3992.0	3992.0	1.0
obs[1]	1.000	0.000	1.000	1.000	0.000	0.000	4000.0	4000.0	NaN
obs[2]	0.994	0.079	1.000	1.000	0.001	0.001	4061.0	4000.0	1.0
obs[3]	0.000	0.000	0.000	0.000	0.000	0.000	4000.0	4000.0	NaN
obs[4]	0.992	0.086	1.000	1.000	0.001	0.001	3609.0	4000.0	1.0
...	...	...	...	...	...	...	...	...	...
p[70]	0.649	0.110	0.437	0.843	0.002	0.002	2629.0	2327.0	1.0
p[71]	1.000	0.000	1.000	1.000	0.000	0.000	1000.0	1016.0	1.0
p[72]	0.000	0.000	0.000	0.000	0.000	0.000	904.0	1071.0	1.0
p[73]	1.000	0.001	0.999	1.000	0.000	0.000	1140.0	1444.0	1.0
p[74]	0.386	0.110	0.203	0.612	0.002	0.002	1973.0	2341.0	1.0

[150 rows x 9 columns]

```
/opt/homebrew/lib/python3.10/site-packages/arviz/stats/diagnostics.py:592: RuntimeWarning: invalid value enc
   double_scalars
  (between_chain_variance / within_chain_variance + num_samples - 1) / (num_samples)
/opt/homebrew/lib/python3.10/site-packages/arviz/stats/diagnostics.py:592: RuntimeWarning: invalid value enc
   double_scalars
  (between_chain_variance / within_chain_variance + num_samples - 1) / (num_samples)
/opt/homebrew/lib/python3.10/site-packages/arviz/stats/diagnostics.py:592: RuntimeWarning: invalid value enc
   double_scalars
```

Evaluation

```
post.posterior["p"].shape
```

```
(4, 1000, 175)
```

```
post.posterior_predictive["p"].shape
```

```
(4, 1000, 75)
```

```
p_train = post.posterior["p"].mean(dim=["chain", "draw"])
p_test  = post.posterior_predictive["p"].mean(dim=["chain", "draw"])
```

```
1 p_train
```

```
<xarray.DataArray 'p' (p_dim_0: 175)>
array([0.05309, 0.00413, 0.40297, 0.99875,
       0.08667, 0.96987, 0.58873, 0.     ,
       0.61224, 0.95717, 0.64985, 1.     ,
       0.97592, 0.     , 0.32721, 0.00001,
       0.34297, 0.57578, 1.     , 0.96606,
       0.93605, 0.0522 , 0.     , 0.07021,
       0.15963, 0.00161, 0.07183, 0.27359,
       0.85473, 0.47879, 1.     , 0.65964,
       0.     , 0.99933, 0.00038, 0.99874,
Coordinates:
  * p_dim_0  (p_dim_0) int64 0 1 2 3 4 5 6
```

xarray.DataArray 'p' (**p_dim_0**: 175)

array([0.05309, 0.00413, 0.40297, 0.998
0.08667, 0.96987, 0.58873, 0.
0.61224, 0.95717, 0.64985, 1.
0.97592, 0. , 0.32721, 0.00(
0.34297, 0.57578, 1. , 0.96(
0.93605, 0.0522 , 0. , 0.07(
0.15963, 0.00161, 0.07183, 0.27
0.85473, 0.47879, 1. , 0.65
0. , 0.99933, 0.00038, 0.998

▾ Coordinates:

p_dim_0
(p_dim_0)

0 1 2 3 4 5 … 170 171 172 173 174

```
1 p_test
```

```
<xarray.DataArray 'p' (p_dim_2: 75)>
array([0.90924, 1.     , 0.99468, 0.00001,
       0.80827, 0.99265, 0.00132, 0.99018,
       0.07043, 0.35611, 0.00001, 0.78791,
       0.00374, 0.24545, 0.47969, 0.99949,
Coordinates:
  * p_dim_2  (p_dim_2) int64 0 1 2 3 4 5 6
```

xarray.DataArray 'p' (**p_dim_2**: 75)

array([0.90924, 1. , 0.99468, 0.00(
0.80827, 0.99265, 0.00132, 0.99(
0.07043, 0.35611, 0.00001, 0.78
0.00374, 0.24545, 0.47969, 0.99

▾ Coordinates:

p_dim_2
(p_dim_2)

0 1 2 3 4 5 6 … 69 70 71 72 73 74

▸ Indexes: (1)

▸ Attributes: (0)

ROC & AUC

```
from sklearn.metrics import RocCurveDisplay, accuracy_score, auc, roc_curve

# Test data
fpr_test, tpr_test, thd_test = roc_curve(y_true=y_test, y_score=p_test)
auc_test = auc(fpr_test, tpr_test); auc_test
```

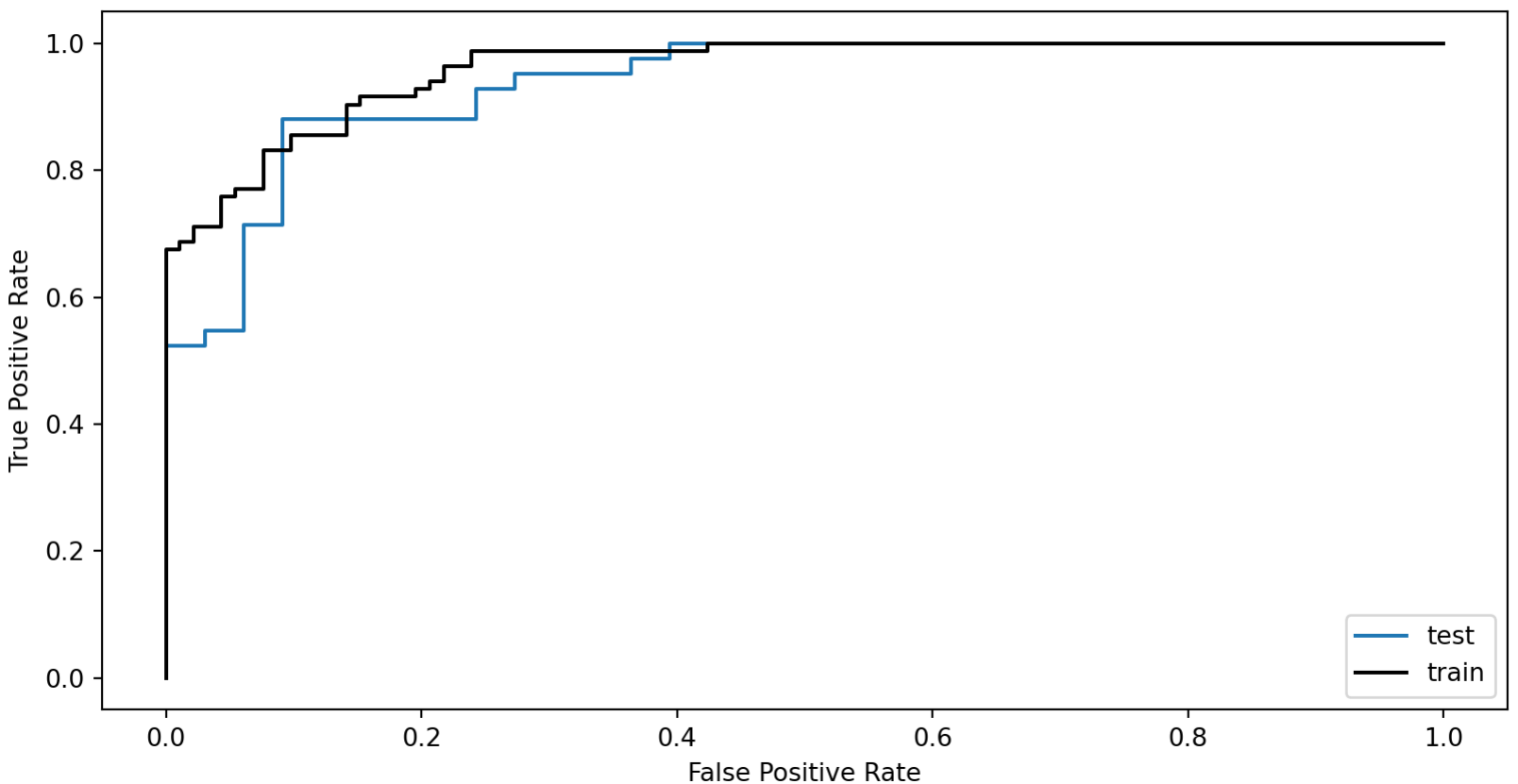
```
0.937950937950938
```

```
# Training data
fpr_train, tpr_train, thd_train = roc_curve(y_true=y_train, y_score=p_train)
auc_train = auc(fpr_train, tpr_train); auc_train
```

```
0.9600576217915139
```

ROC Curves

```
fig, ax = plt.subplots()
roc = RocCurveDisplay(fpr=fpr_test, tpr=tpr_test).plot(ax=ax, label="test")
roc = RocCurveDisplay(fpr=fpr_train, tpr=tpr_train).plot(ax=ax, color="k", label="train")
plt.show()
```

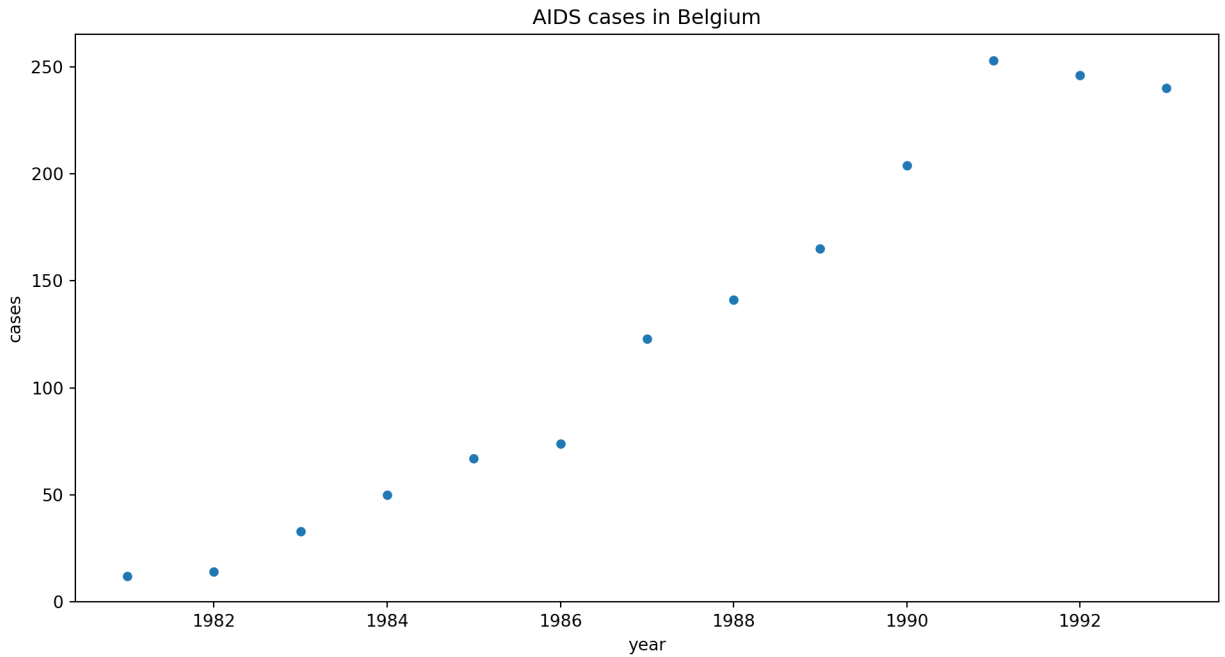


Demo 2 - Poisson Regression

Data

```
1 aids
```

	year	cases
0	1981	12
1	1982	14
2	1983	33
3	1984	50
4	1985	67
5	1986	74
6	1987	123
7	1988	141
8	1989	165
9	1990	204
10	1991	253
11	1992	246
12	1993	240



Model

```
y, X = patsy.dmatrices("cases ~ year", aids)

X_lab = X.design_info.column_names
y = np.asarray(y).flatten()
X = np.asarray(X)

with pm.Model(coords = {"coeffs": X_lab}) as model:
    b = pm.Cauchy("b", alpha=0, beta=1, dims="coeffs")
    η = X @ b
    λ = pm.Deterministic("λ", np.exp(η))

    y_ = pm.Poisson("y", mu=λ, observed=y)

    post = pm.sample(random_seed=1234, progressbar=False)
```

```
Auto-assigning NUTS sampler...
Initializing NUTS using jitter+adapt_diag...
Multiprocess sampling (4 chains in 4 jobs)
NUTS: [b]
Sampling 4 chains for 1_000 tune and 1_000 draw iterations (4_000 + 4_000 draws total) took 29 seconds.
The effective sample size per chain is smaller than 100 for some parameters.  A higher number is needed for
   rhat and ess computation. See https://arxiv.org/abs/1903.08008 for details
Chain 0 reached the maximum tree depth. Increase `max_treedepth`, increase `target_accept` or reparameterize
Chain 1 reached the maximum tree depth. Increase `max_treedepth`, increase `target_accept` or reparameterize
Chain 2 reached the maximum tree depth. Increase `max_treedepth`, increase `target_accept` or reparameterize
Chain 3 reached the maximum tree depth. Increase `max_treedepth`, increase `target_accept` or reparameterize
```

Adjusting the sampler

```
with model:
  post = pm.sample(
    random_seed=1234, progressbar=False,
    step = pm.NUTS(max_treedepth=20)
  )
```

```
Multiprocess sampling (4 chains in 4 jobs)
NUTS: [b]
Sampling 4 chains for 1_000 tune and 1_000 draw iterations (4_000 + 4_000 draws total) took 44 seconds.
```

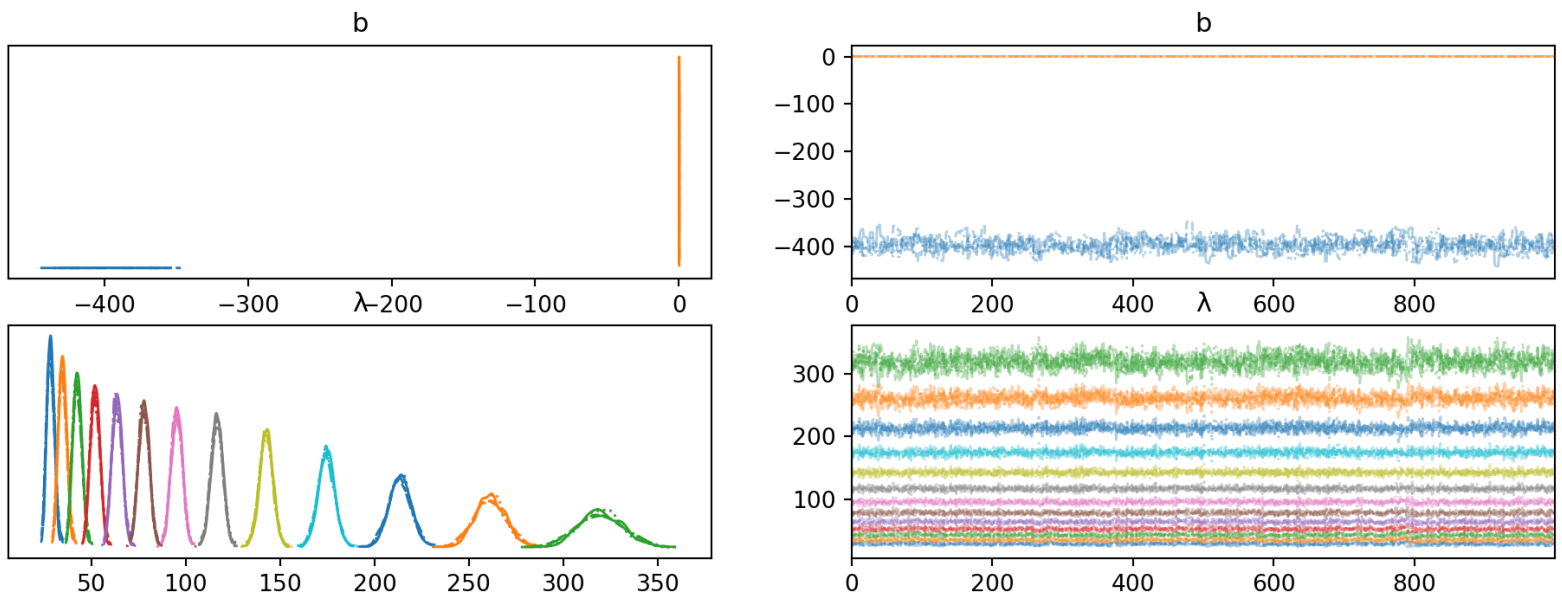
Summary

```
az.summary(post)
```

	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
b[Intercept]	-396.376	15.147	-425.313	-369.376	0.675	0.477	504.0	400.0	1.01
b[year]	0.202	0.008	0.188	0.216	0.000	0.000	504.0	400.0	1.01
λ[0]	28.436	1.969	24.883	32.126	0.082	0.058	577.0	689.0	1.00
λ[1]	34.778	2.161	31.112	39.095	0.089	0.063	597.0	784.0	1.00
λ[2]	42.536	2.348	37.908	46.623	0.094	0.067	626.0	816.0	1.00
λ[3]	52.029	2.523	47.287	56.656	0.098	0.069	671.0	933.0	1.00
λ[4]	63.643	2.680	58.568	68.554	0.098	0.070	748.0	1216.0	1.00
λ[5]	77.854	2.821	72.555	83.007	0.094	0.067	893.0	1631.0	1.00
λ[6]	95.243	2.965	89.826	100.765	0.086	0.061	1182.0	2007.0	1.00
λ[7]	116.524	3.176	110.429	122.255	0.075	0.053	1801.0	2476.0	1.00
λ[8]	142.568	3.586	135.697	149.166	0.066	0.047	2939.0	2704.0	1.00
λ[9]	174.444	4.408	165.582	182.323	0.081	0.058	2941.0	2659.0	1.00
λ[10]	213.458	5.886	202.814	224.964	0.140	0.099	1786.0	2198.0	1.00
λ[11]	261.213	8.249	245.893	276.948	0.253	0.179	1063.0	1870.0	1.00
λ[12]	319.670	11.744	297.363	341.648	0.411	0.291	815.0	1335.0	1.00

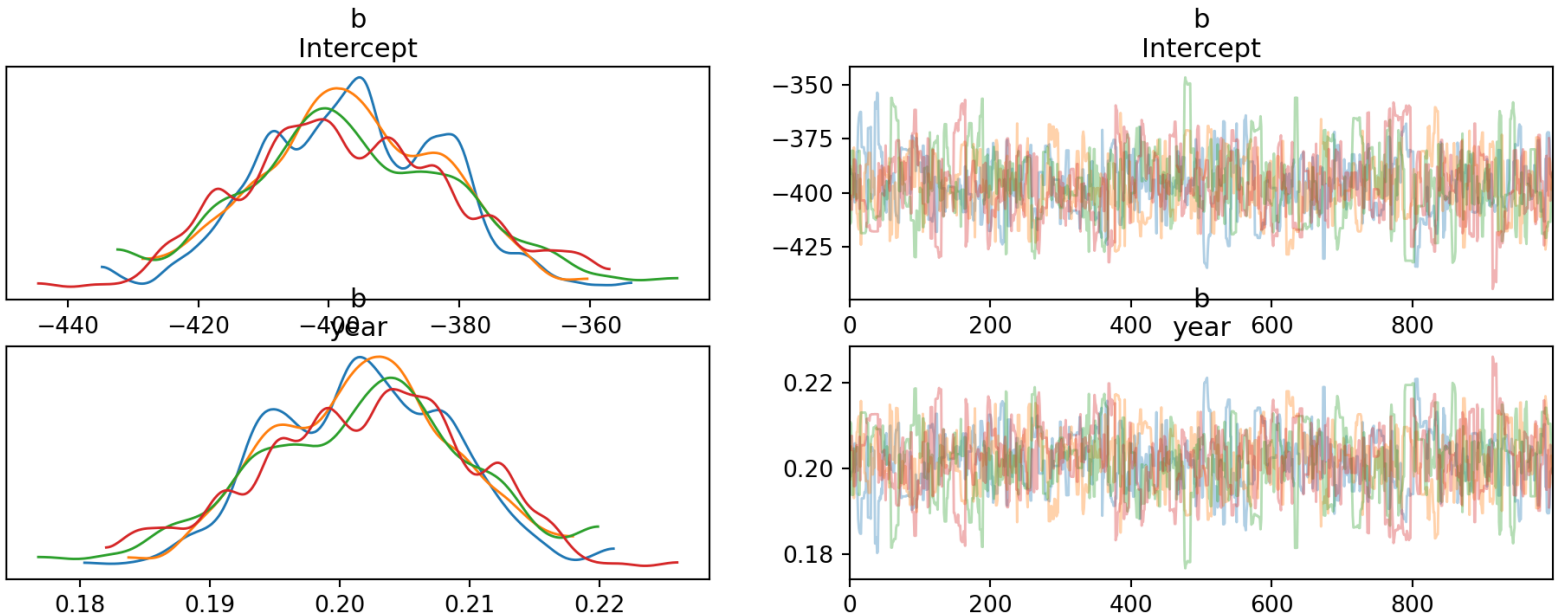
Trace plots

```
1  ax = az.plot_trace(post)
2  plt.show()
```



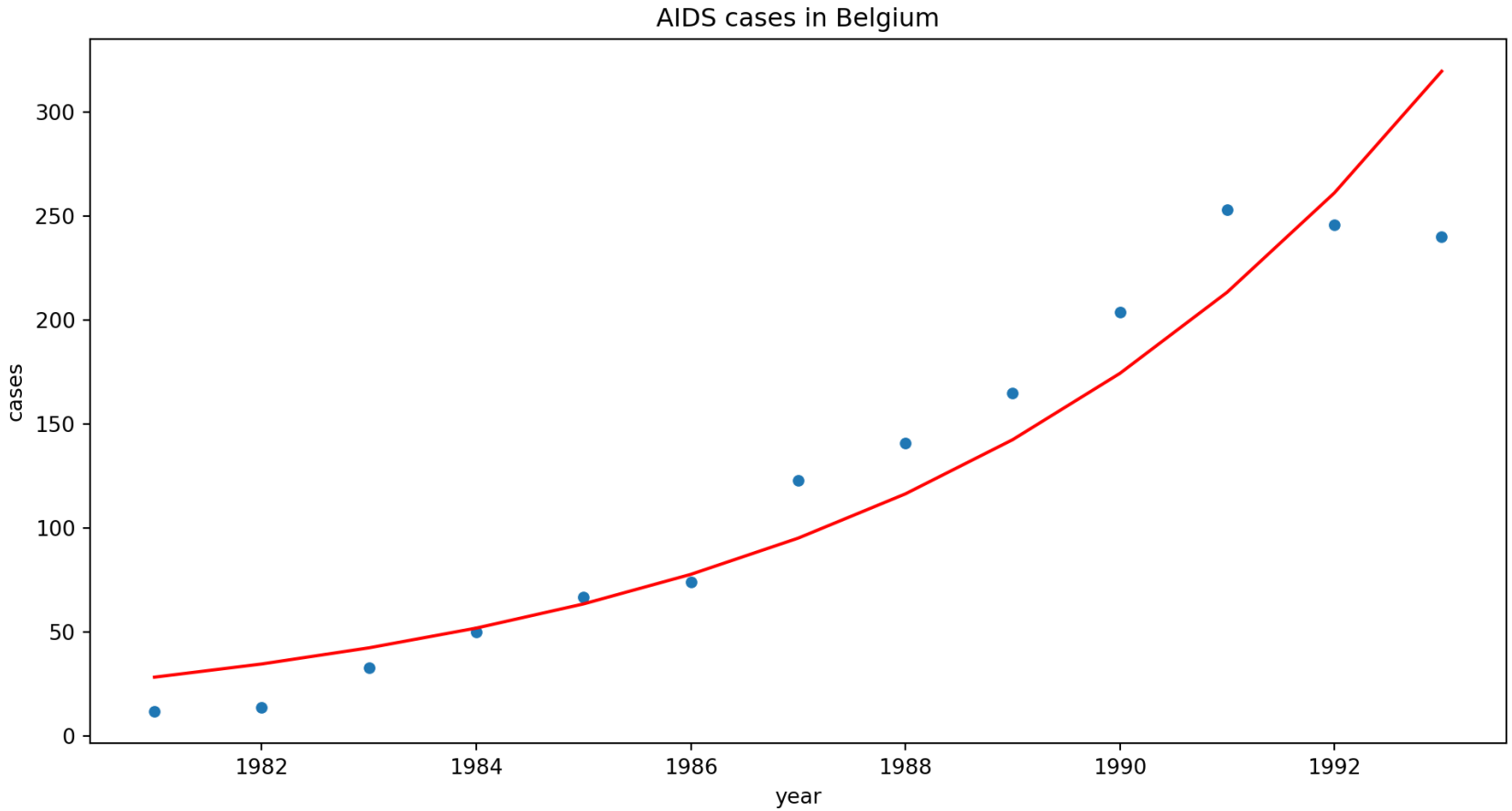
Trace plots (again)

```
1  ax = az.plot_trace(post.posterior["b"], compact=False)
2  plt.show()
```



Predictions (λ)

```
plt.figure(figsize=(12,6))
sns.scatterplot(x="year", y="cases", data=aids)
sns.lineplot(x="year", y=post.posterior["λ"].mean(dim=["chain", "draw"]),
             data=aids, color='red')
plt.title("AIDS cases in Belgium")
plt.show()
```


AIDS cases in Belgium
cases
year
0
50
100
150
200
250
300
1982
1984
1986
1988
1990
1992

Revised model

```
y, X = patsy.dmatrices(
  "cases ~ year_min + np.power(year_min,2)",
  aids.assign(year_min = lambda x: x.year-np.min(x.year))
)

X_lab = X.design_info.column_names
y = np.asarray(y).flatten()
X = np.asarray(X)

with pm.Model(coords = {"coeffs": X_lab}) as model:
    b = pm.Cauchy("b", alpha=0, beta=1, dims="coeffs")
    η = X @ b
    λ = pm.Deterministic("λ", np.exp(η))

    y_ = pm.Poisson("y", mu=λ, observed=y)

    post = pm.sample(random_seed=1234, progressbar=False)
```

```
Auto-assigning NUTS sampler...
Initializing NUTS using jitter+adapt_diag...
Multiprocess sampling (4 chains in 4 jobs)
NUTS: [b]
Sampling 4 chains for 1_000 tune and 1_000 draw iterations (4_000 + 4_000 draws total) took 2 seconds.
```

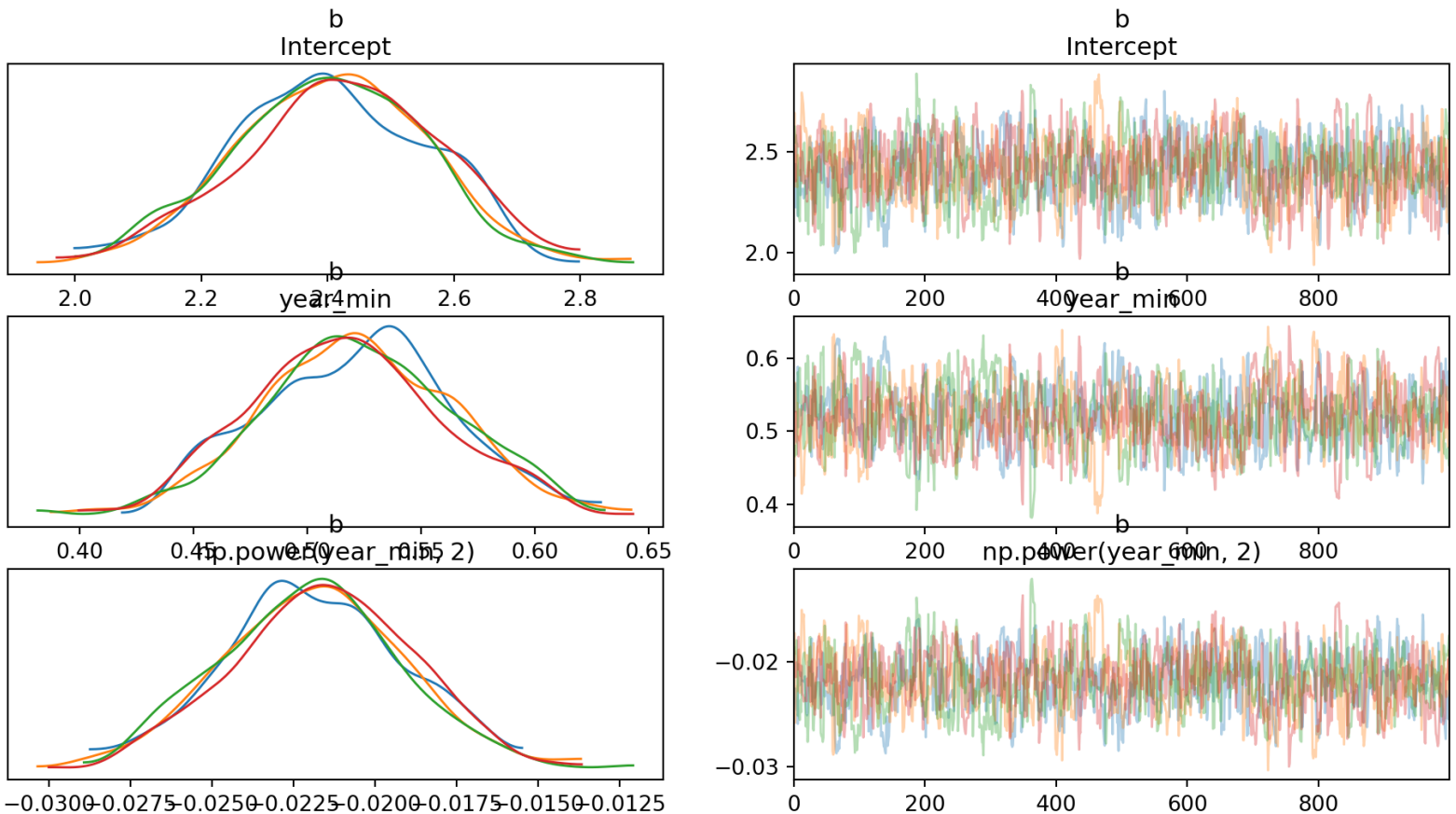
Summary

```
1 az.summary(post)
```

	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
b[Intercept]	2.408	0.152	2.108	2.671	0.006	0.004	662.0	950.0	1.01
b[year_min]	0.521	0.042	0.444	0.600	0.002	0.001	558.0	812.0	1.01
b[np.power(year_min, 2)]	-0.022	0.003	-0.027	-0.017	0.000	0.000	557.0	911.0	1.01
λ[0]	11.243	1.707	8.233	14.456	0.066	0.047	662.0	950.0	1.01
λ[1]	18.431	2.120	14.436	22.235	0.079	0.056	708.0	1036.0	1.01
λ[2]	28.977	2.463	24.495	33.640	0.086	0.061	812.0	1287.0	1.00
λ[3]	43.669	2.720	38.843	48.915	0.083	0.059	1065.0	1682.0	1.00
λ[4]	63.054	2.998	57.354	68.647	0.071	0.050	1791.0	2154.0	1.00
λ[5]	87.199	3.531	80.718	93.834	0.067	0.048	2771.0	2248.0	1.00
λ[6]	115.466	4.430	106.793	123.138	0.093	0.066	2270.0	2314.0	1.00
λ[7]	146.371	5.471	136.326	156.475	0.143	0.101	1455.0	2206.0	1.00
λ[8]	177.615	6.244	166.215	189.250	0.180	0.128	1173.0	2103.0	1.00
λ[9]	206.313	6.496	194.285	218.714	0.166	0.118	1501.0	2371.0	1.00
λ[10]	229.422	6.688	217.410	242.475	0.127	0.090	2752.0	2580.0	1.00
λ[11]	244.281	8.398	227.029	258.838	0.140	0.099	3615.0	3192.0	1.00
λ[12]	249.120	12.570	224.280	271.257	0.334	0.237	1415.0	2628.0	1.00

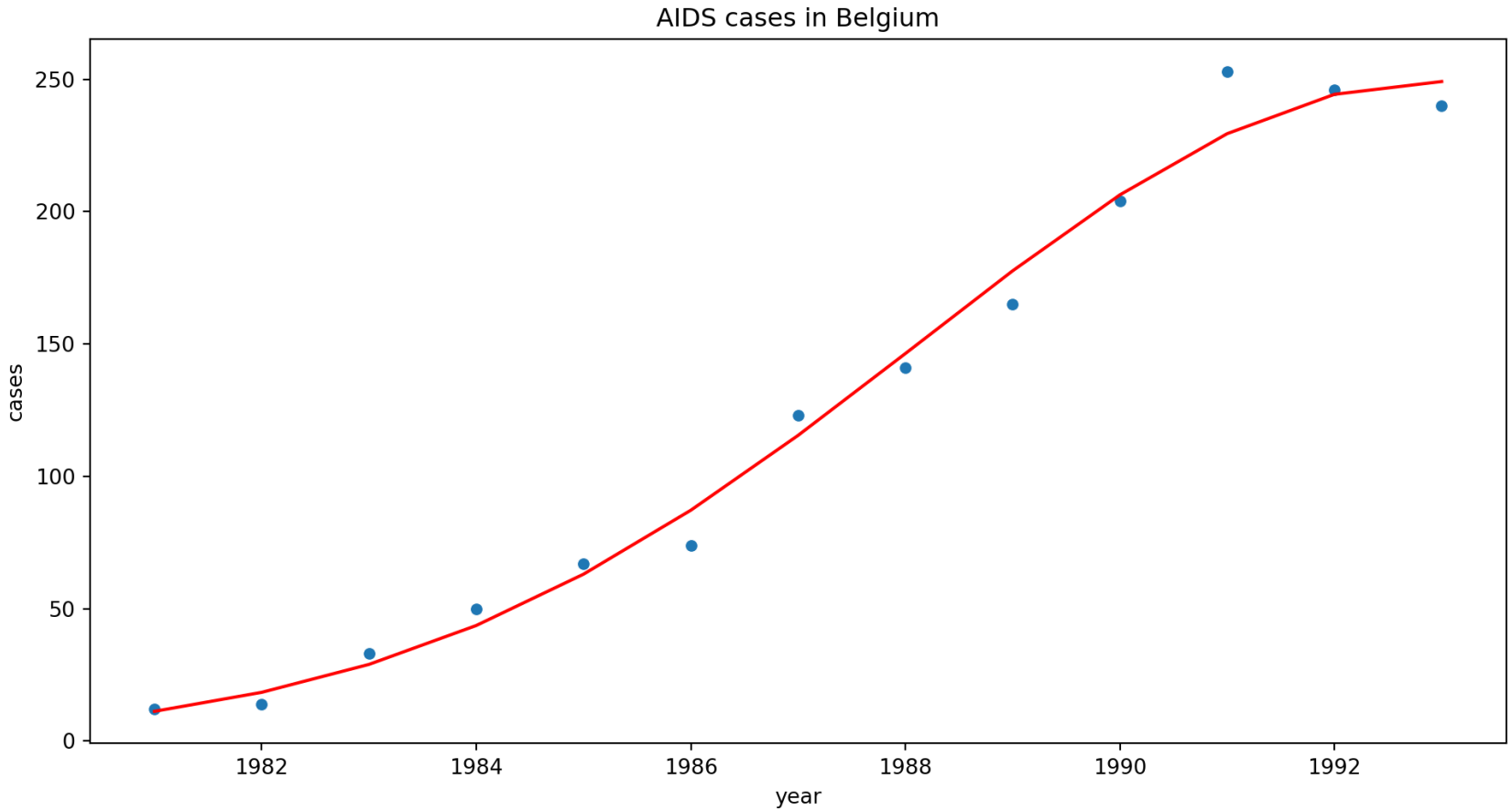
Trace plots

```
1  ax = az.plot_trace(post.posterior["b"], compact=False)
2  plt.show()
```



Predictions (λ)

```
plt.figure(figsize=(12,6))
sns.scatterplot(x="year", y="cases", data=aids)
sns.lineplot(x="year", y=post.posterior["λ"].mean(dim=["chain", "draw"]),
             data=aids, color='red')
plt.title("AIDS cases in Belgium")
plt.show()
```


AIDS cases in Belgium
cases
year
0
50
100
150
200
250
1982
1984
1986
1988
1990
1992

Demo 3 - Gaussian Process

Data

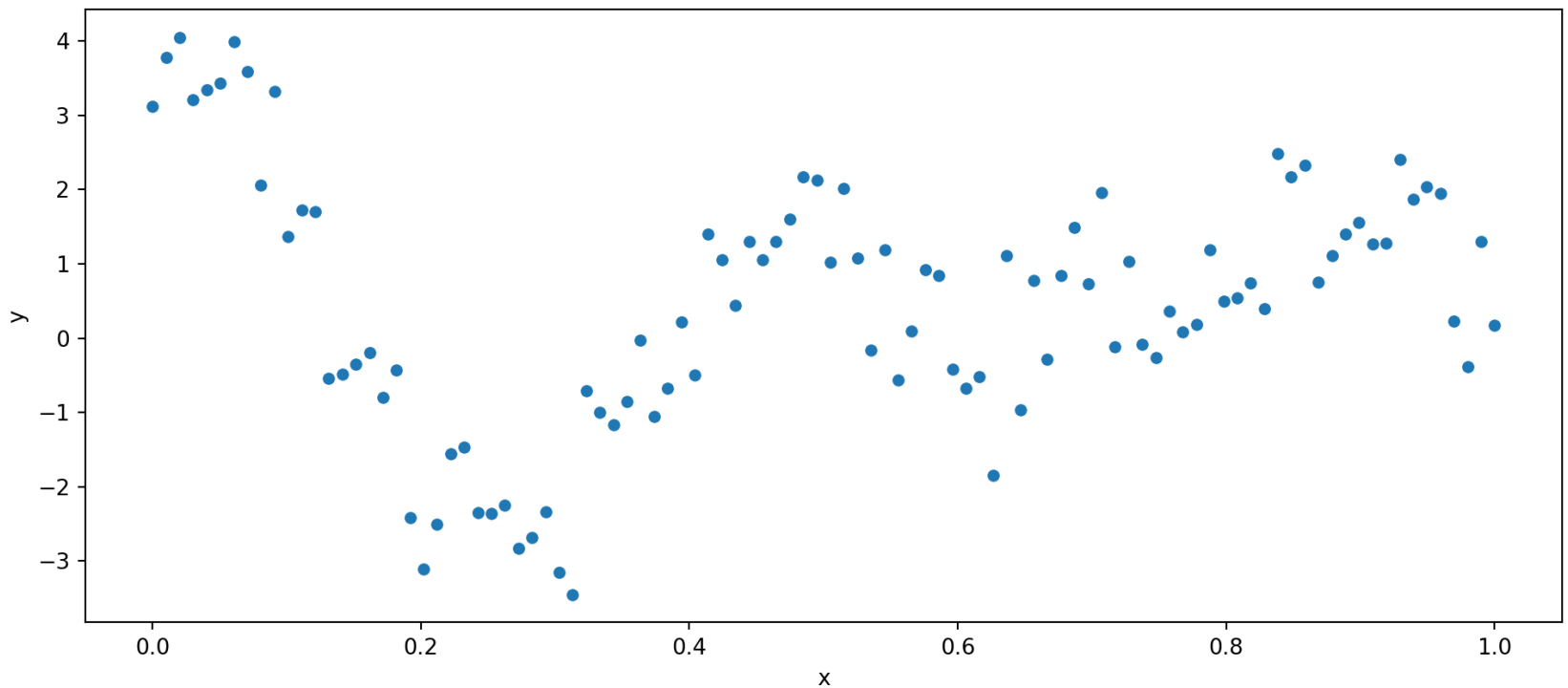
```
d = pd.read_csv("data/Lec20/gp.csv")
d
```

```
          x         y
0   0.000000  3.113179
1   0.010101  3.774512
2   0.020202  4.045562
3   0.030303  3.207971
4   0.040404  3.336638
..       ...       ...
95  0.959596  1.951793
96  0.969697  0.224769
97  0.979798 -0.387220
98  0.989899  1.304032
99  1.000000  0.174600

[100 rows x 2 columns]
```

```
n = d.shape[0]
D = np.array([ np.abs(xi - d.x) for xi in d.x])
I = np.eye(n)
```

```
fig = plt.figure(figsize=(12, 5))
ax = sns.scatterplot(x="x", y="y", data=d)
plt.show()
```



GP model

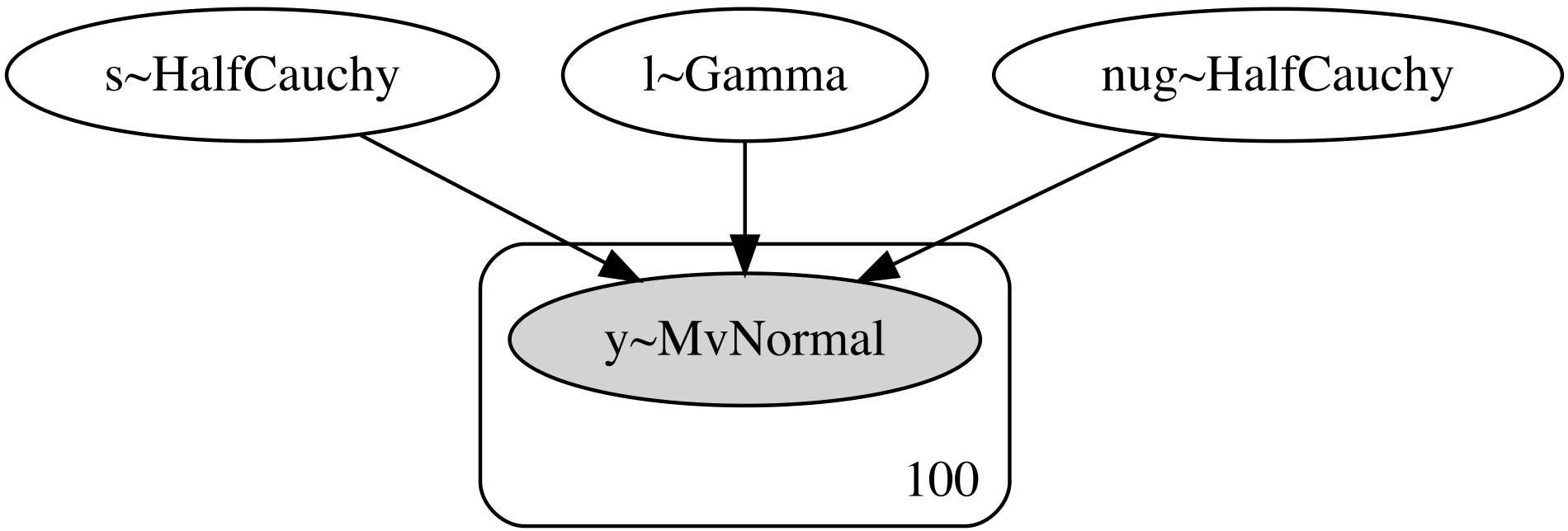
```
with pm.Model() as model:
  l = pm.Gamma("l", alpha=2, beta=1)
  s = pm.HalfCauchy("s", beta=5)
  nug = pm.HalfCauchy("nug", beta=5)

  cov = s**2 * pm.gp.cov.ExpQuad(1, l)
  gp = pm.gp.Marginal(cov_func=cov)

  y_ = gp.marginal_likelihood(
    "y",
    X=d.x.to_numpy().reshape(-1,1),
    y=d.y.to_numpy(),
    sigma=nug
  )
```

Model visualization

```
1  pm.model_to_graphviz(model)
```



MAP estimates

```
with model:
  gp_map = pm.find_MAP()
```

```
|████████████████████████████████████| 100.00% [22/22 00:00<00:00 logp = -134.97, ||grad|| = 0.0022539]
```

```
gp_map
```

```
{'l_log__': array(-2.35319), 's_log__': array(0.54918), 'nug_log__': array(-0.33237), 'l': array(0.09507), '
   array(1.73184), 'nug': array(0.71722)}
```

Sampling

```
with model:
  post_nuts = pm.sample(
    chains=2, cores=1,
    progressbar = False
  )
```

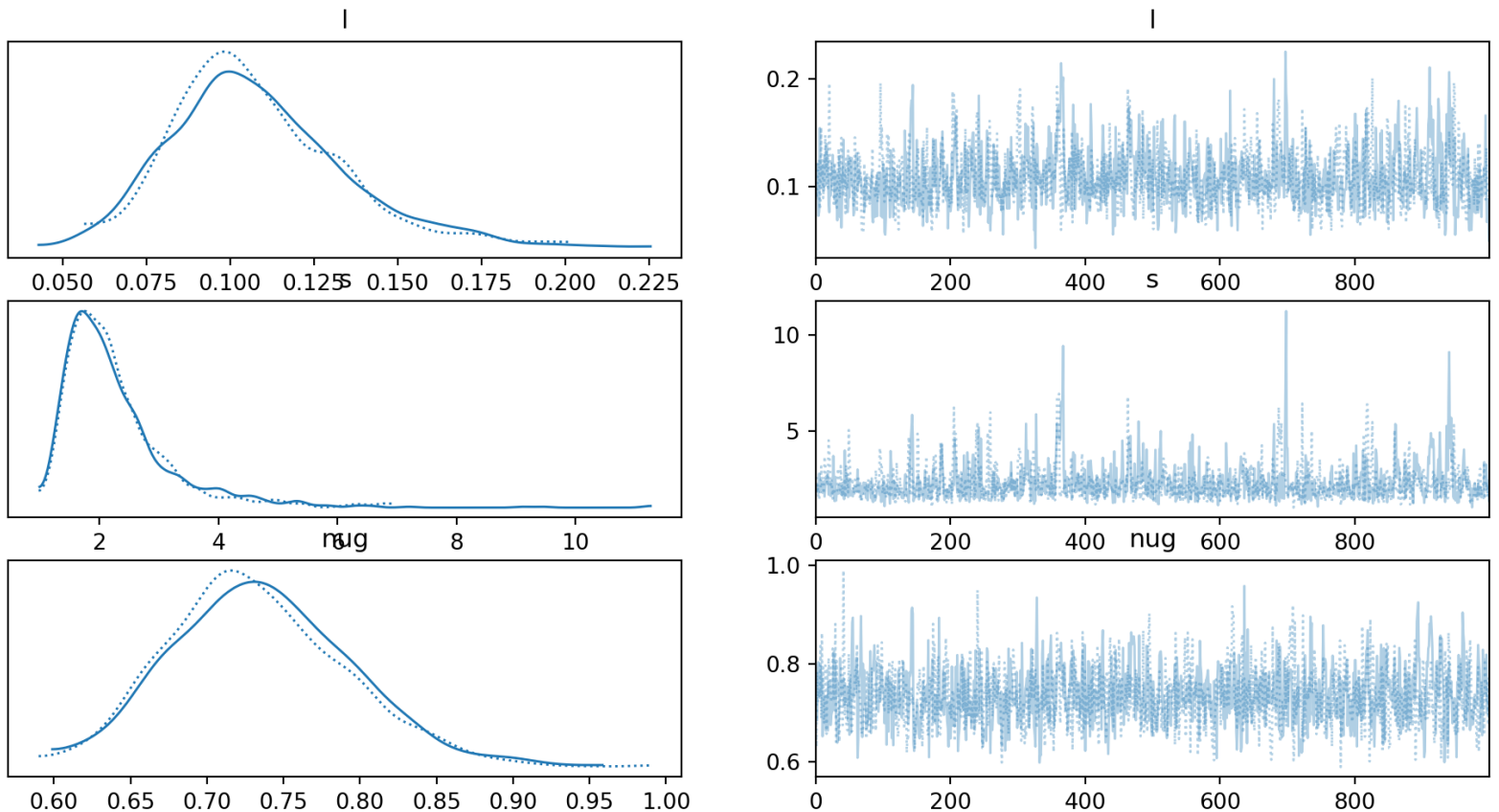
```
Auto-assigning NUTS sampler...
Initializing NUTS using jitter+adapt_diag...
Sequential sampling (2 chains in 1 job)
NUTS: [l, s, nug]
Sampling 2 chains for 1_000 tune and 1_000 draw iterations (2_000 + 2_000 draws total) took 25 seconds.
We recommend running at least 4 chains for robust computation of convergence diagnostics
```

```
az.summary(post_nuts)
```

	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
l	0.108	0.026	0.059	0.157	0.001	0.001	922.0	752.0	1.0
s	2.270	0.935	1.073	3.951	0.037	0.027	924.0	540.0	1.0
nug	0.735	0.058	0.628	0.844	0.002	0.001	1130.0	1160.0	1.0

Trace plots

```
1  ax = az.plot_trace(post_nuts)
2  plt.show()
```



slice sampler

```
with model:
    post_slice = pm.sample(
        chains = 2, cores = 1,
        step = pm.Slice([l,s,nug]),
        progressbar = False
    )
```

```
Sequential sampling (2 chains in 1 job)
CompoundStep
>Slice: [l]
>Slice: [s]
>Slice: [nug]
Sampling 2 chains for 1_000 tune and 1_000 draw iterations (2_000 + 2_000 draws total) took 30 seconds.
We recommend running at least 4 chains for robust computation of convergence diagnostics
```

```
az.summary(post_slice)
```

	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
l	0.107	0.024	0.068	0.159	0.001	0.001	798.0	1102.0	1.0
s	2.181	0.722	1.104	3.425	0.023	0.016	990.0	953.0	1.0
nug	0.736	0.060	0.628	0.849	0.001	0.001	1835.0	1338.0	1.0

MH sampler

```
with model:
    post_mh = pm.sample(
        chains = 2, cores = 1,
        step = pm.Metropolis([l,s,nug]),
        progressbar = False
    )
```

```
Sequential sampling (2 chains in 1 job)
CompoundStep
>Metropolis: [l]
>Metropolis: [s]
>Metropolis: [nug]
Sampling 2 chains for 1_000 tune and 1_000 draw iterations (2_000 + 2_000 draws total) took 9 seconds.
We recommend running at least 4 chains for robust computation of convergence diagnostics
```

```
az.summary(post_mh)
```

	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
l	0.110	0.025	0.070	0.162	0.002	0.002	143.0	234.0	1.02
s	2.308	0.784	1.204	3.954	0.060	0.043	157.0	277.0	1.01
nug	0.731	0.056	0.636	0.824	0.005	0.004	126.0	282.0	1.02

Mixing and matching

```
with model:
    post_mix = pm.sample(
        chains = 2, cores = 1,
        step = [
          pm.Metropolis([l]),
          pm.Slice([s])
        ],
        progressbar = False
    )
```

```
Sequential sampling (2 chains in 1 job)
CompoundStep
>Metropolis: [l]
>Slice: [s]
>NUTS: [nug]
Sampling 2 chains for 1_000 tune and 1_000 draw iterations (2_000 + 2_000 draws total) took 23 seconds.
We recommend running at least 4 chains for robust computation of convergence diagnostics
```

```
az.summary(post_mix)
```

	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
l	0.105	0.024	0.065	0.147	0.002	0.001	147.0	341.0	1.02
s	2.193	0.850	1.131	3.542	0.047	0.033	312.0	616.0	1.00
nug	0.736	0.059	0.625	0.842	0.002	0.001	770.0	1259.0	1.00

NUTS sampler (JAX)

```
from pymc.sampling import jax

with model:
    post_jax = jax.sample_blackjax_nuts(
        chains = 2, cores = 1
    )
```

```
Compiling...
Compilation time =  0:00:00.801486
Sampling...
Sampling time =  0:00:02.695987
Transforming variables...
Transformation time =  0:00:26.120903
```

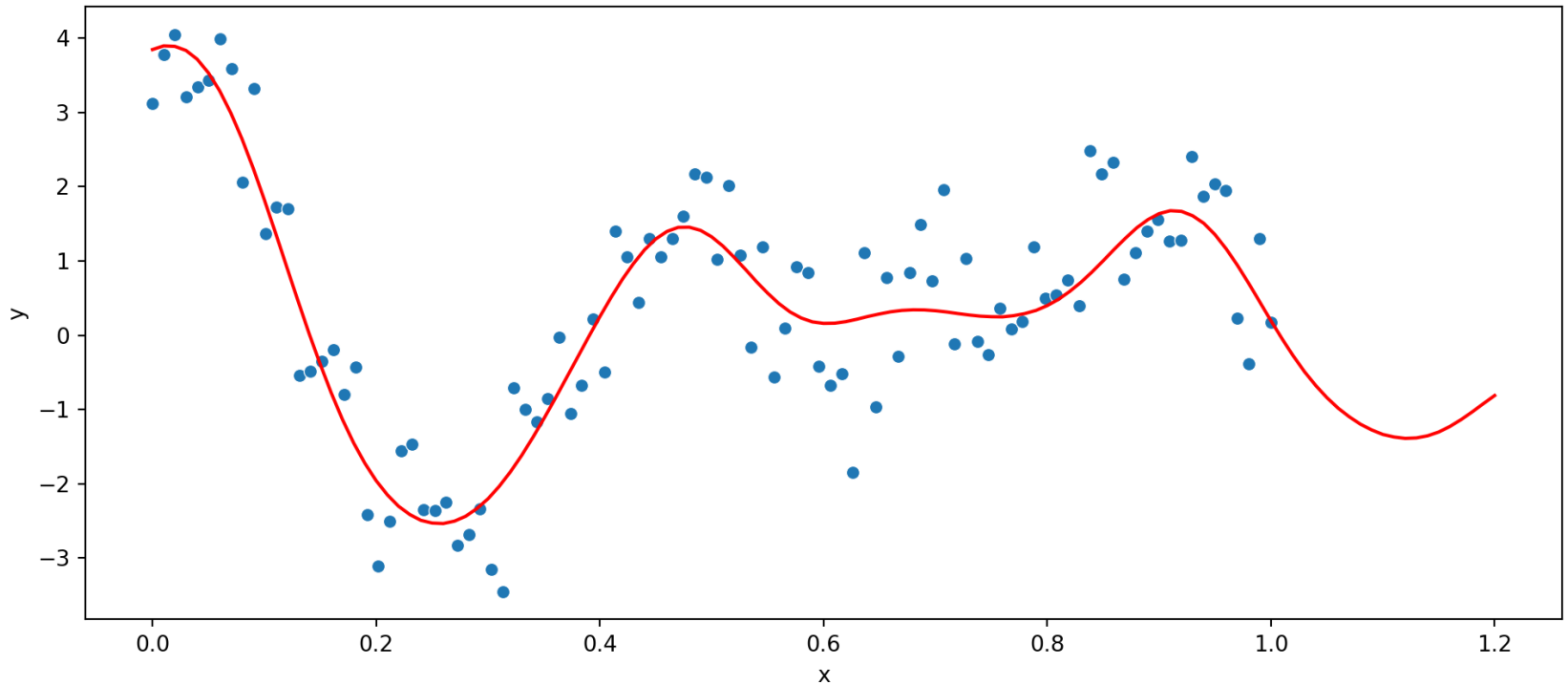
```
az.summary(post_jax)
```

	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
l	0.107	0.026	0.064	0.161	0.001	0.001	1259.0	1187.0	1.0
s	2.234	0.842	1.080	3.697	0.027	0.020	1264.0	1080.0	1.0
nug	0.734	0.059	0.625	0.839	0.002	0.001	1433.0	1142.0	1.0

Conditional Predictions (MAP)

```
with model:
  X_new = np.linspace(0, 1.2, 121).reshape(-1, 1)
  y_pred = gp.conditional("y_pred", X_new)
  pred_map = pm.sample_posterior_predictive(
    [gp_map], var_names=["y_pred"], progressbar = False
  )
```

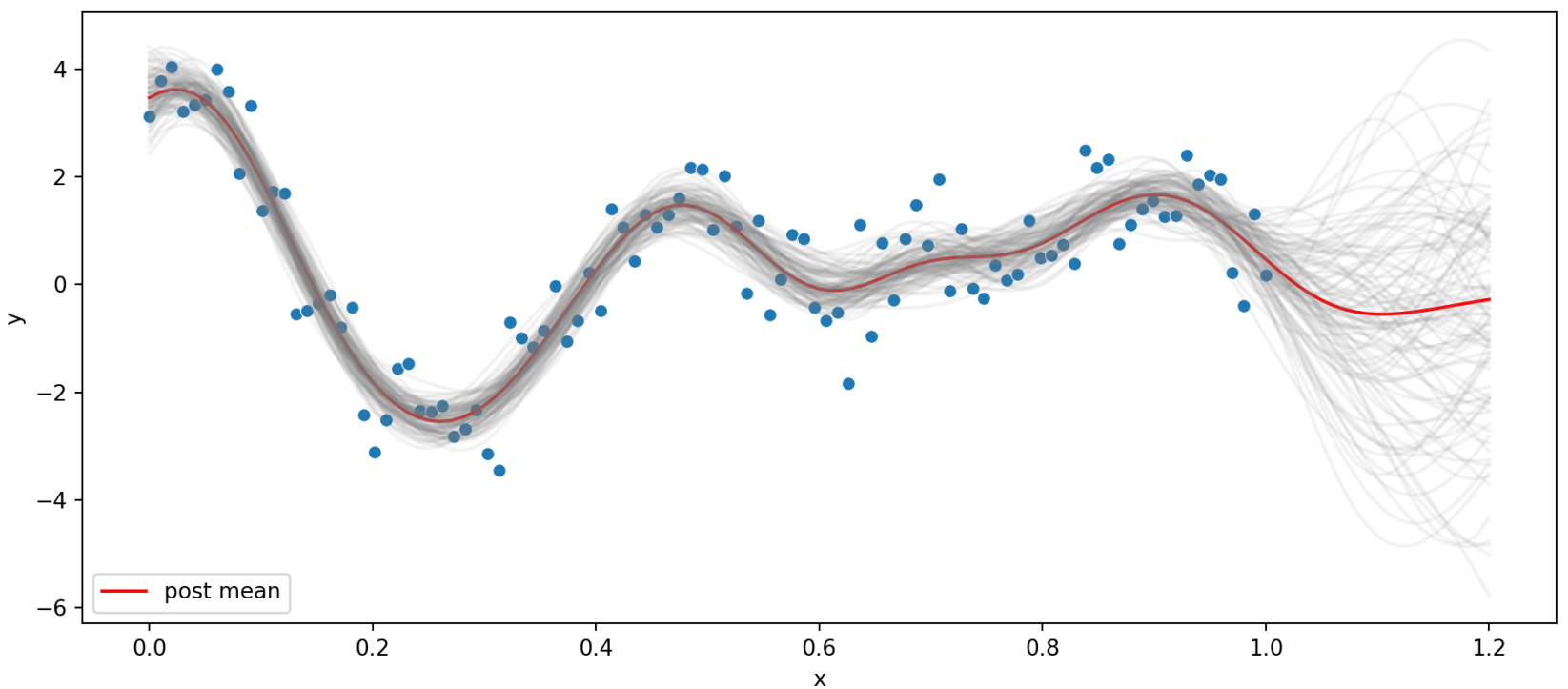
```
Sampling: [y_pred]----------------------| 0.00% [0/1 00:00<?]
 |████████████████████████████████████| 100.00% [1/1 00:00<00:00]
```


4
3
2
1
0
−1
−2
−3
y
0.0
0.2
0.4
0.6
0.8
1.0
1.2
x

Conditional Predictions (thinned posterior)

```
with model:
  pred_post = pm.sample_posterior_predictive(
    post_nuts.sel(draw=slice(None,None,10)), var_names=["y_pred"]
  )
```

```
Sampling: [y_pred]
|████████████████████████████████████████| 100.00% [400/400 03:40<00:00]
```



Conditional Predictions w/ nugget

```
1  with model:
2    y_star = gp.conditional("y_star", X_new, pred_noise=True)
3    predn_post = pm.sample_posterior_predictive(
4      post_nuts.sel(draw=slice(None,None,10)), var_names=["y_star"]
5    )
```

```
Sampling: [y_star]
|████████████████████████████████████████| 100.00% [200/200 01:51<00:00]
```