MCMC - Performance & Stan

Lecture 25

Dr. Colin Rundel

Example - Gaussian Process

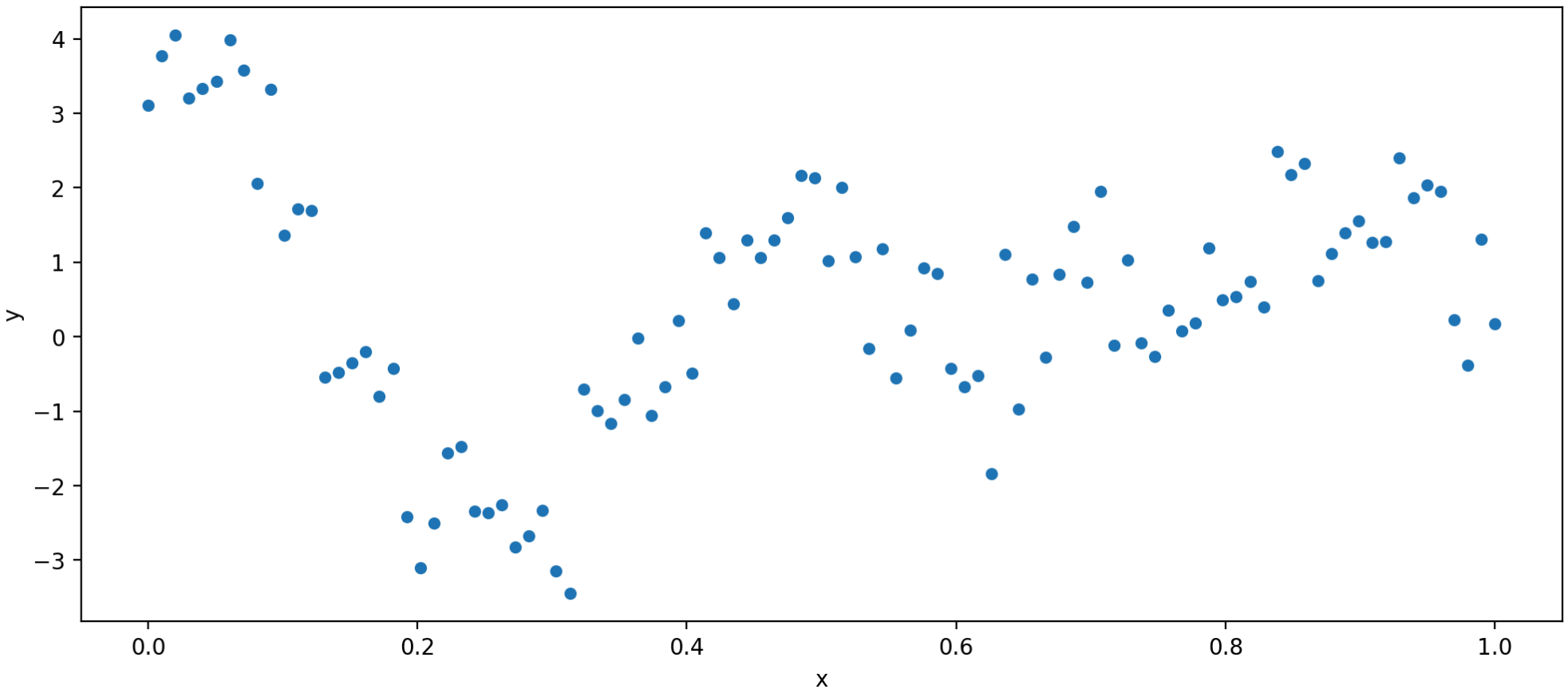
Data

```
1 d = pd.read_csv("data/gp2.csv")
2 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 × 2 columns

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

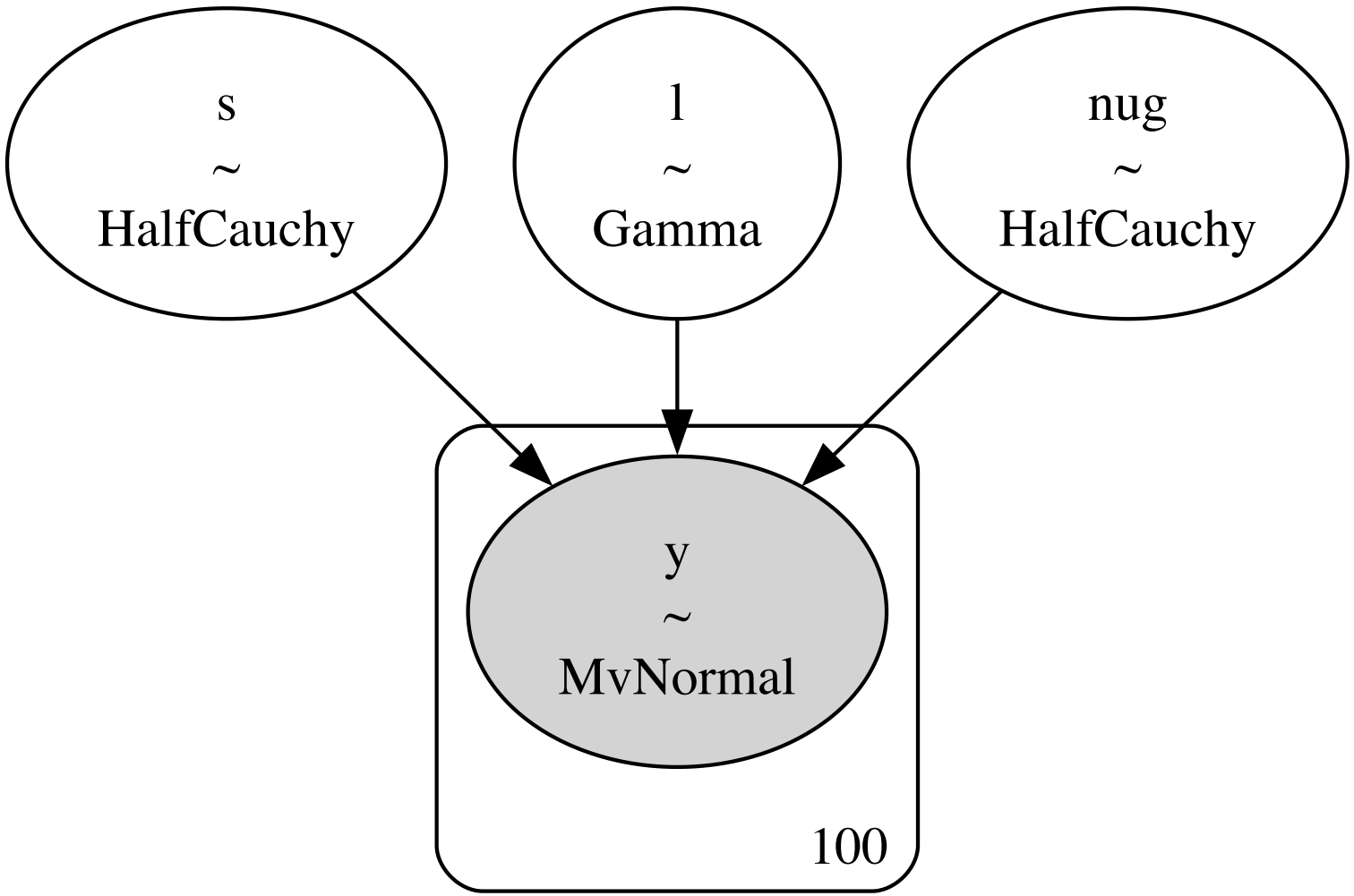


GP model

```
X = d.x.to_numpy().reshape(-1,1)
y = d.y.to_numpy()

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(input_dim=1, ls=l)
  gp = pm.gp.Marginal(cov_func=cov)

  y_ = gp.marginal_likelihood(
    "y", X=X, y=y, sigma=nug
  )
```

Model visualization

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



MAP estimates

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

```
MAP ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━  0% 0:00:04 logp = -134.97, ||grad|| = 0.0022532
```

```
1 gp_map
```

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

Full Posterior Sampling

```
with model:
  post_nuts = pm.sample()
```

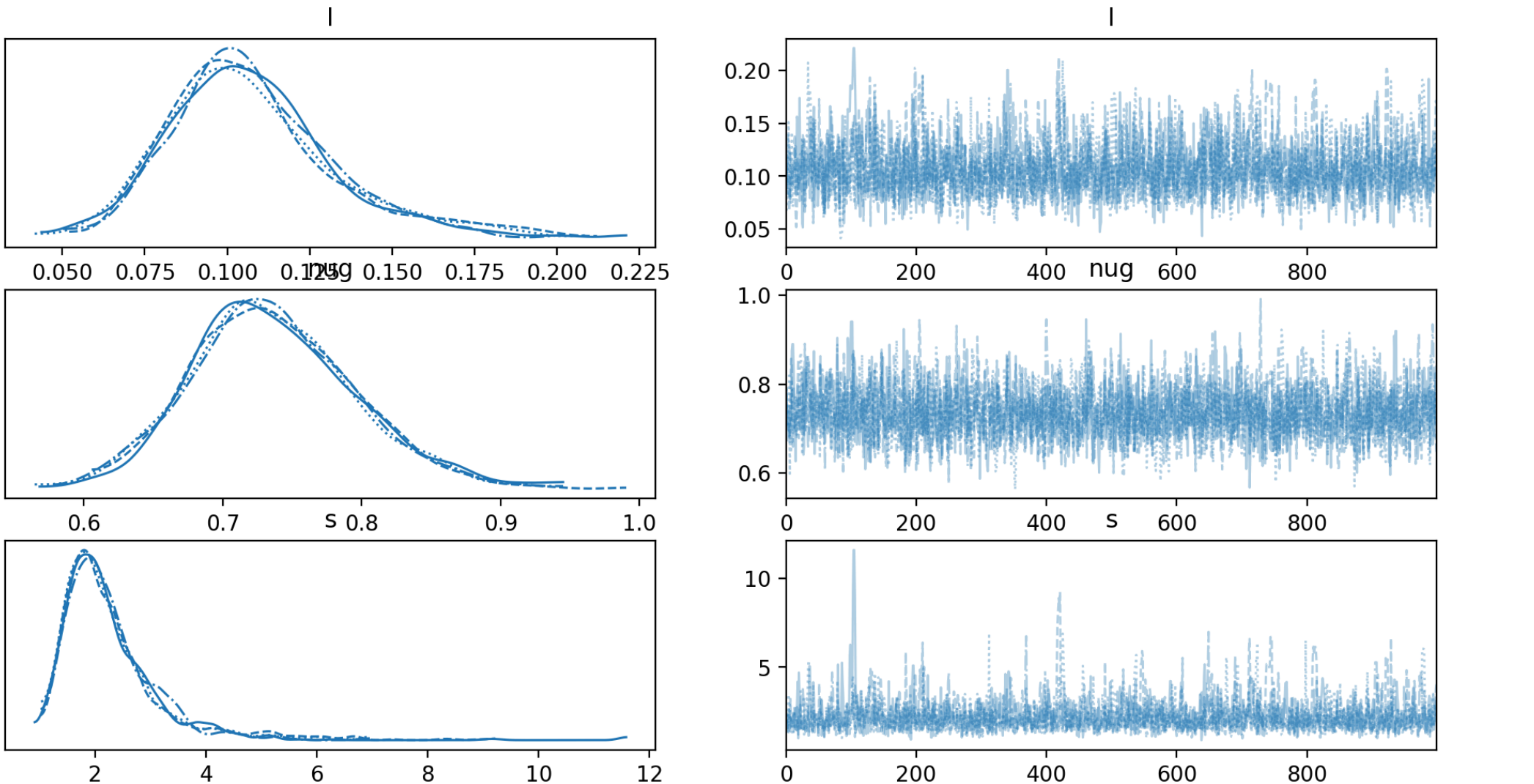
Progress	Draws	Divergences	Step size	Grad evals	Sampling Speed	Elap
————————————	2000	0	0.78	7	396.88 draws/s	0:00
————————————	2000	0	0.77	7	385.53 draws/s	0:00
————————————	2000	0	0.65	7	401.67 draws/s	0:00
————————————	2000	0	0.89	7	394.68 draws/s	0:00

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

	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
l	0.107	0.026	0.065	0.162	0.001	0.001	1835.0	1253.0	1.0
nug	0.736	0.059	0.625	0.846	0.001	0.001	2241.0	2191.0	1.0
s	2.252	0.902	1.049	3.778	0.027	0.057	1620.0	1347.0	1.0

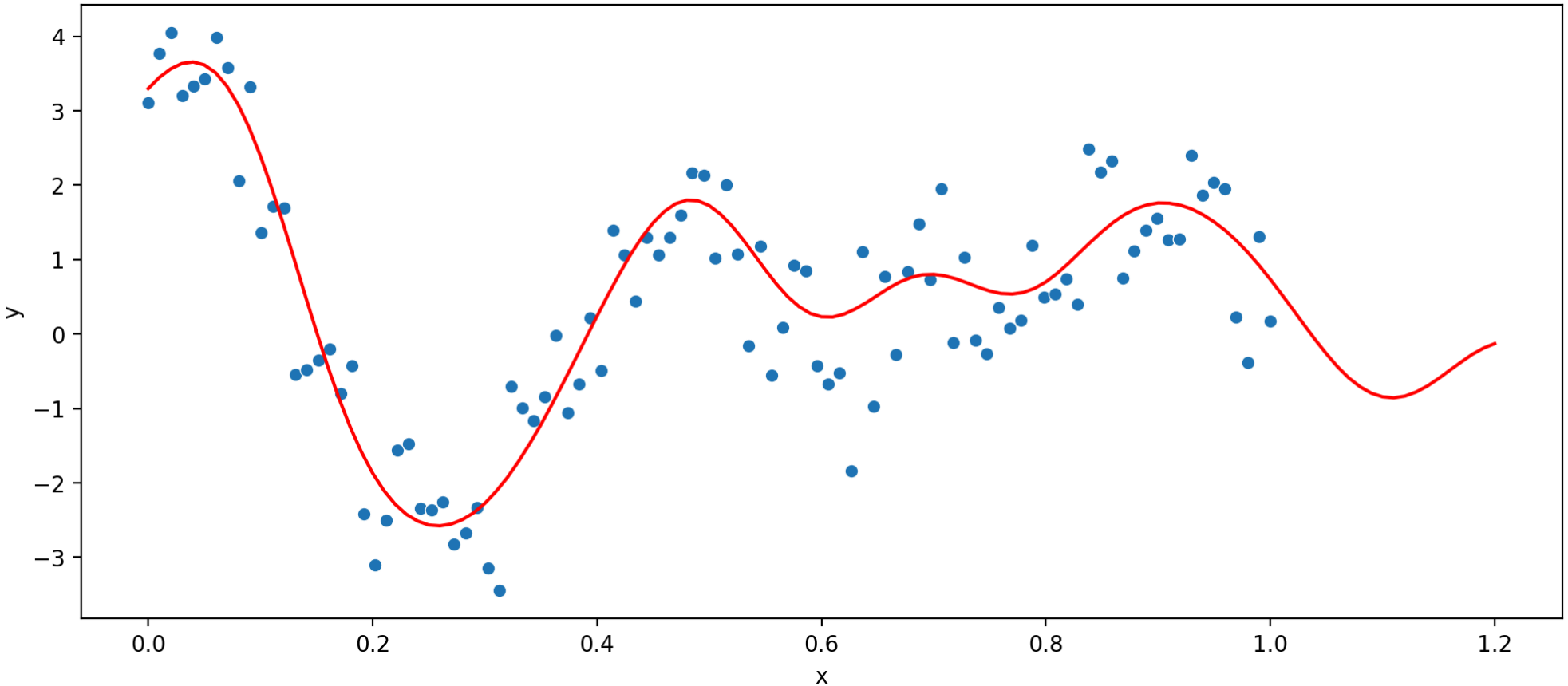
Trace plots

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



Conditional Predictions (MAP)

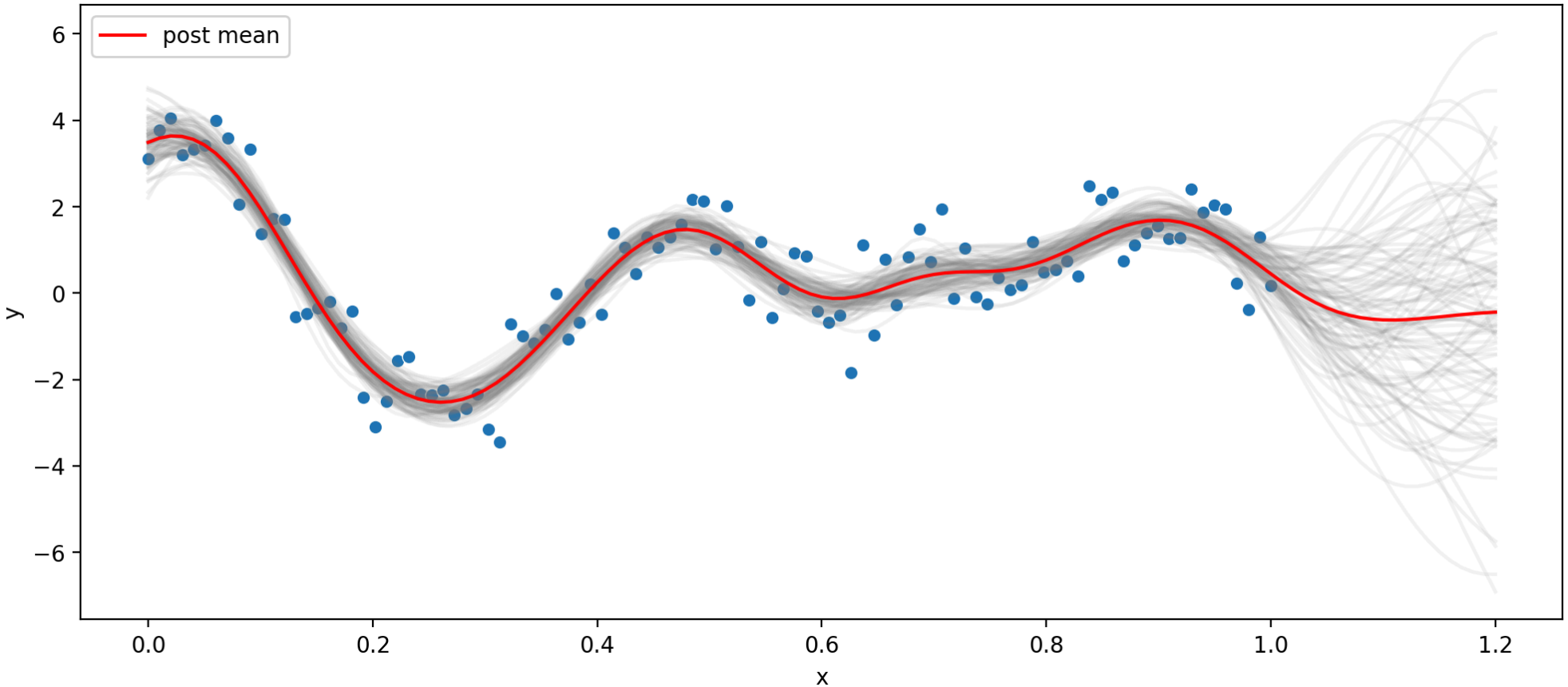
```
X_new = np.linspace(0, 1.2, 121).reshape(-1, 1)

with model:
  y_pred = gp.conditional("y_pred", X_new)
  pred_map = pm.sample_posterior_predictive(
    [gp_map], var_names=["y_pred"], progressbar = False
  )
```


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)

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

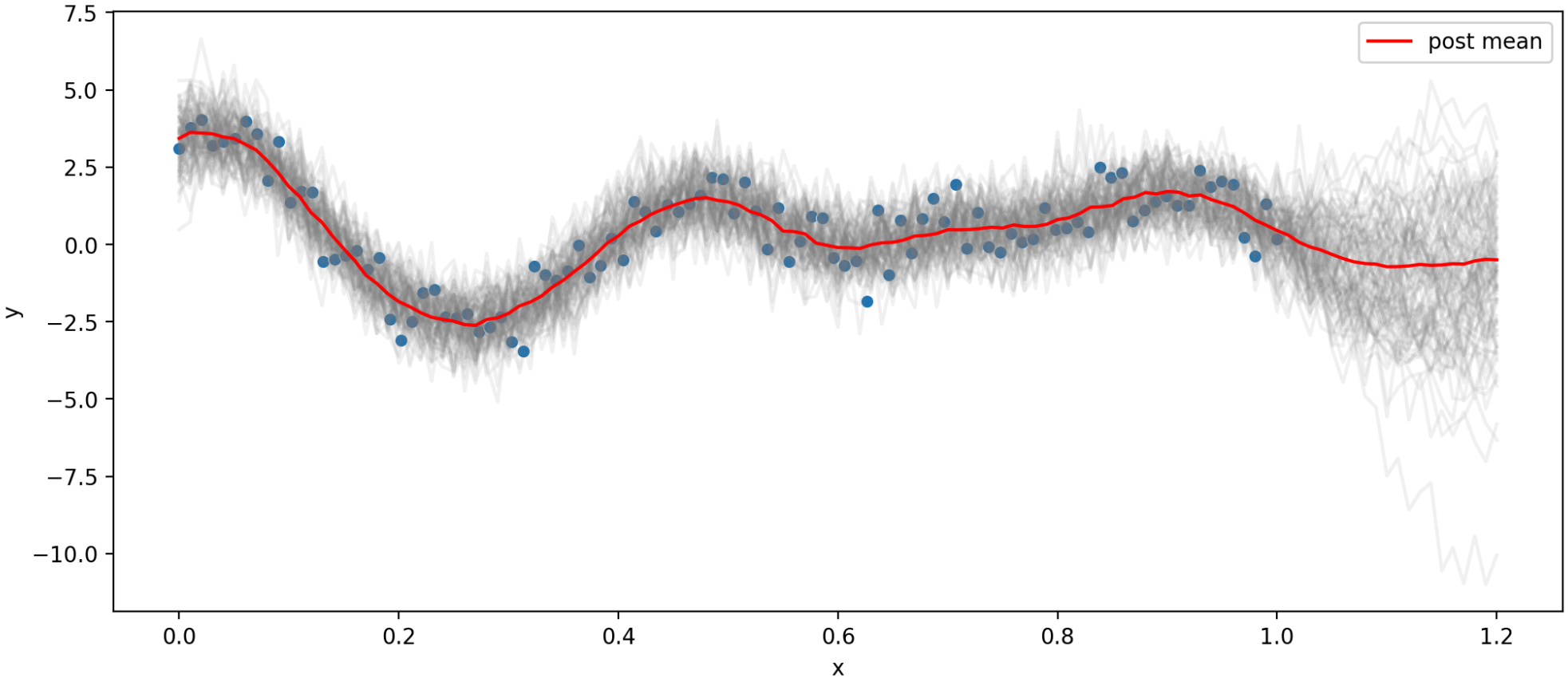
```
Sampling ... ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 100% 0:00:00 / 0:00:00
```


post mean
6
4
2
0
−2
−4
−6
y
0.0
0.2
0.4
0.6
0.8
1.0
1.2
x

Conditional Predictions w/ nugget

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

```
Sampling ... ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 100% 0:00:00 / 0:00:00
```


post mean
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
y
0.0
0.2
0.4
0.6
0.8
1.0
1.2
x

Alternative NUTS samplers

Beyond the ability of PyMC to use different sampling steps - it can also use different sampler algorithm implementations to run your model.

These can be changed via the `nuts_sampler` argument which currently supports:

- `pymc` - standard NUTS sampler using pymc's C backend
- `blackjax` - uses the blackjax library which is a collection of samplers written for JAX
- `numpyro` - probabilistic programming library for pyro built using JAX
- `nutpie` - provides a wrapper to the `nuts-rs` Rust library (slight variation on NUTS implementation)

Performance

```
%%timeit -r 3
with model:
  post_nuts = pm.sample(nuts_sampler="pymc", chains=4, progressbar=False)
```

```
6.11 s ± 35.2 ms per loop (mean ± std. dev. of 3 runs, 1 loop each)
```

```
%%timeit -r 3
with model:
    post_jax = pm.sample(nuts_sampler="blackjax", chains=4, progressbar=False)
```

```
3.96 s ± 21.3 ms per loop (mean ± std. dev. of 3 runs, 1 loop each)
```

```
%%timeit -r 3
with model:
    post_numpyro = pm.sample(nuts_sampler="numpyro", chains=4, progressbar=False)
```

```
3.44 s ± 35.4 ms per loop (mean ± std. dev. of 3 runs, 1 loop each)
```

```
%%timeit -r 3
with model:
    post_nutpie = pm.sample(nuts_sampler="nutpie", chains=4, progressbar=False)
```

```
9.62 s ± 60.8 ms per loop (mean ± std. dev. of 3 runs, 1 loop each)
```

Nutpie and compilation

```
1 import nutpie
2 compiled = nutpie.compile_pymc_model(model)
```

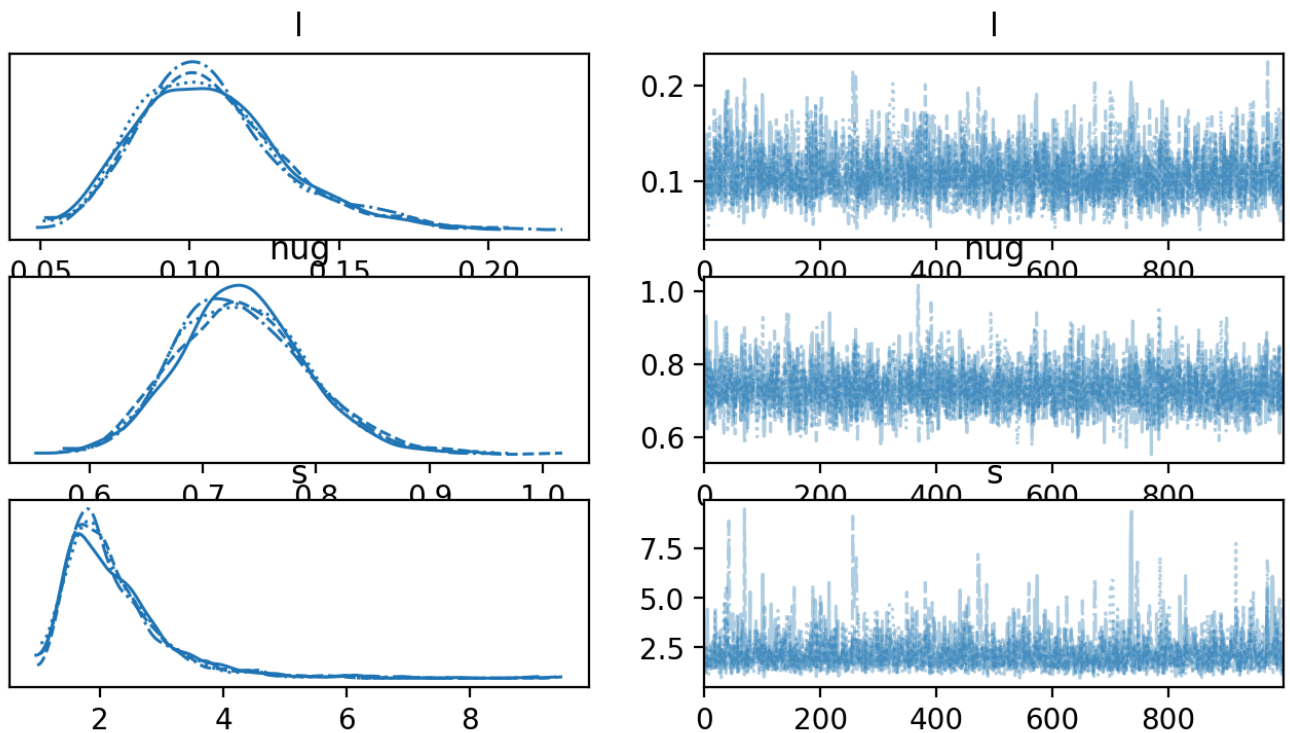
```
1 %%timeit -r 3
2 post_nutpie = nutpie.sample(compiled,chains=4, progress_bar=False)
```

```
2.79 s ± 25.5 ms per loop (mean ± std. dev. of 3 runs, 1 loop each)
```

JAX

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

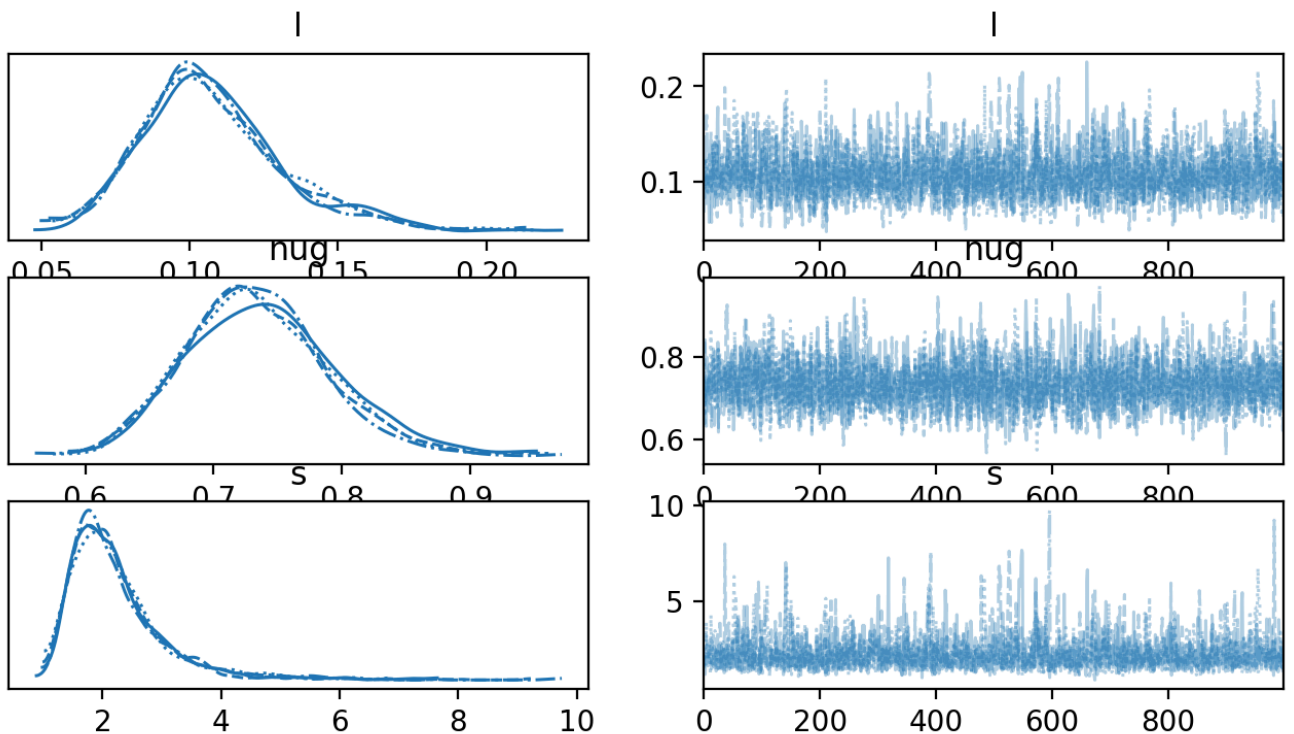
	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
l	0.107	0.025	0.061	0.156	0.001	0.001	2085.0	1709.0	1.0
nug	0.735	0.058	0.633	0.848	0.001	0.001	2551.0	2398.0	1.0
s	2.227	0.865	1.122	3.751	0.022	0.037	2163.0	1717.0	1.0



Numpyro NUTS sampler

```
1 az.summary(post_numpyro)
```

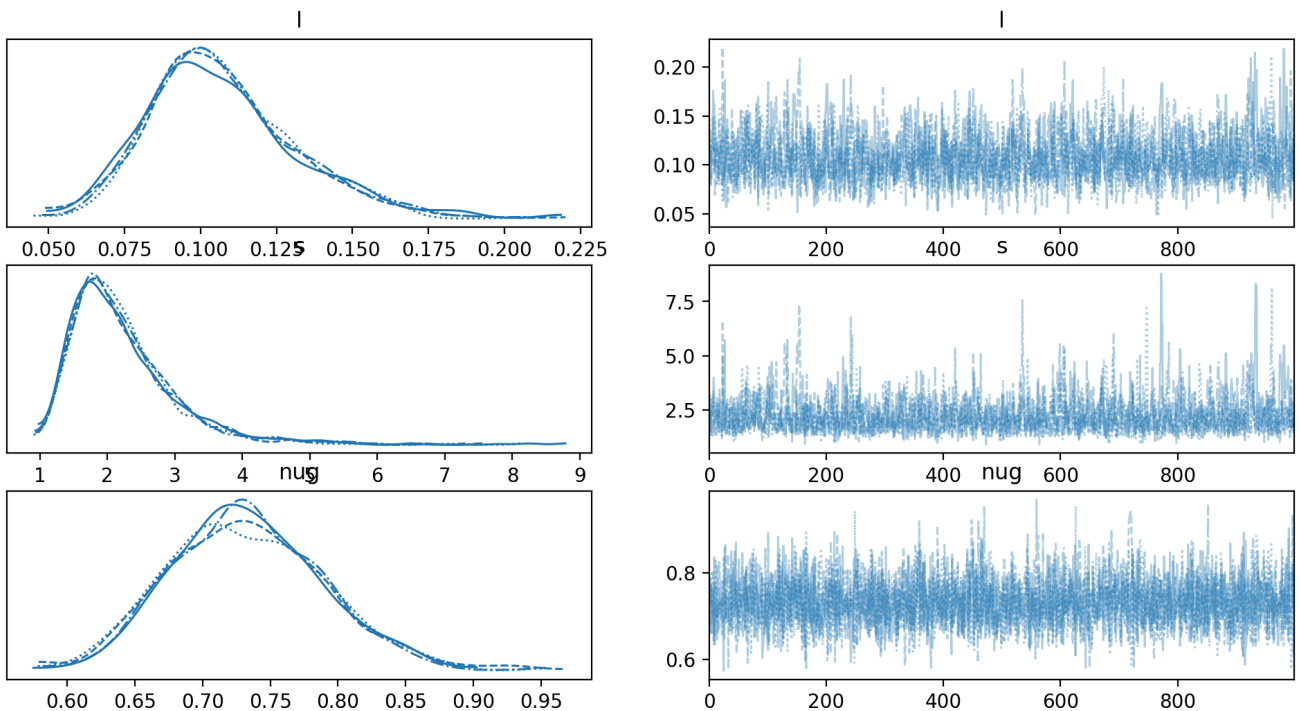
	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
l	0.108	0.025	0.065	0.157	0.001	0.001	2085.0	1921.0	1.0
nug	0.734	0.056	0.634	0.840	0.001	0.001	2717.0	2494.0	1.0
s	2.235	0.869	1.108	3.721	0.023	0.040	1991.0	1503.0	1.0



nutpie sampler

```
1 az.summary(post_nutpie.posterior[["l","s","nug"]])
```

	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
l	0.107	0.024	0.064	0.154	0.001	0.001	1641.0	1757.0	1.0
s	2.227	0.813	1.125	3.674	0.023	0.036	1631.0	1252.0	1.0
nug	0.733	0.057	0.631	0.841	0.001	0.001	3425.0	2555.0	1.0



Stan

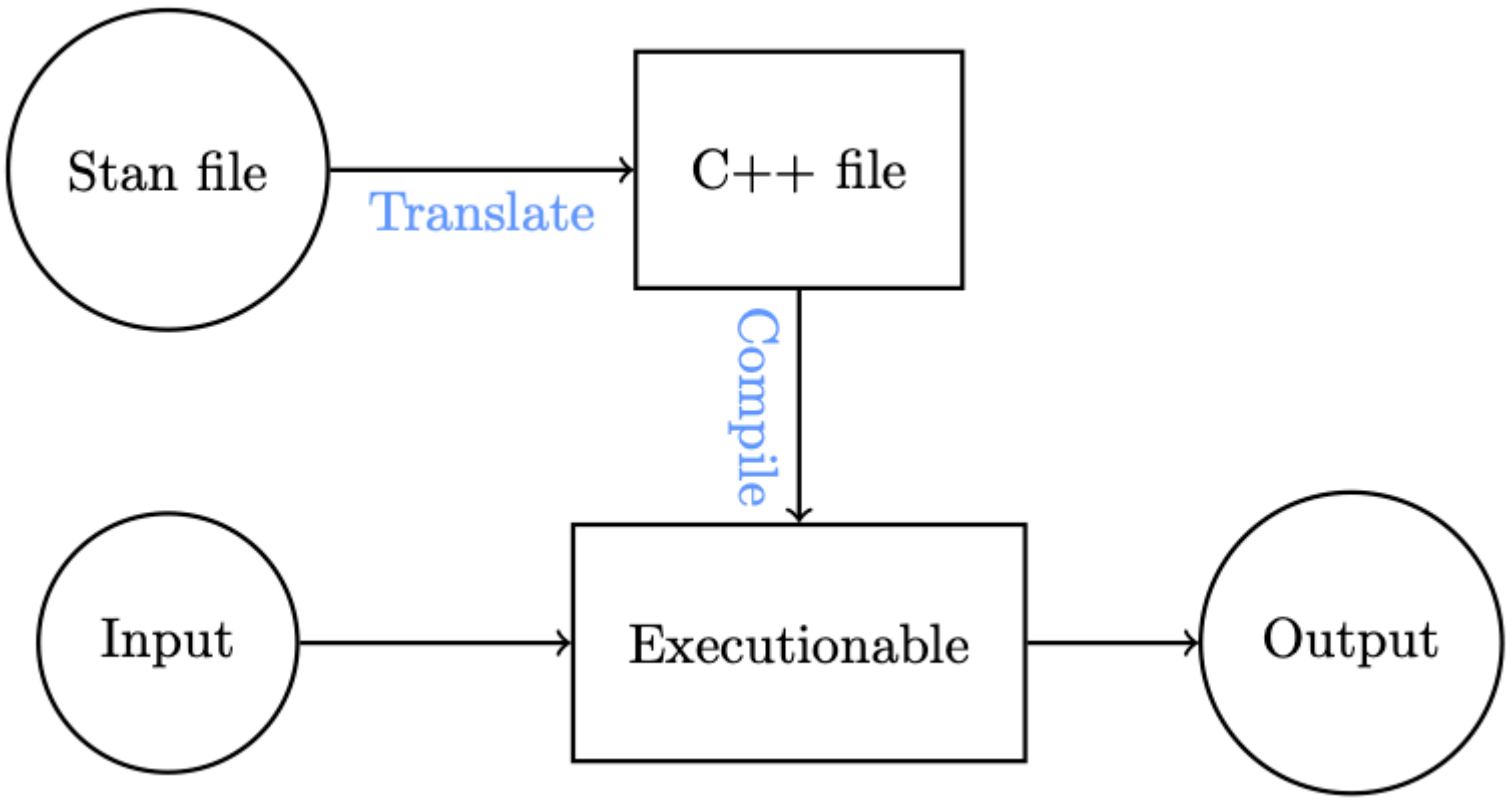
Stan in Python & R

At the moment both Python & R offer two variants of Stan:

- `pystan` & `RStan` - native language interface to the underlying Stan C++ libraries
  - Former does not play nicely with Jupyter (or quarto or positron) - see here for a fix
- `CmdStanPy` & `CmdStanR` - are wrappers around the `CmdStan` command line interface
  - Interface is through files (e.g. `./model.stan`)

Any of the above tools will require a modern C++ toolchain (C++17 support required).

Stan process



Stan file basics

Stan code is divided up into specific blocks depending on usage - all of the following blocks are optional but the ordering has to match what is given below.

```
functions {
  // user-defined functions
}
data {
  // declares the required data for the model
}
transformed data {
   // allows the definition of constants and transformations of the data
}
parameters {
   // declares the model's parameters
}
transformed parameters {
   // variables defined in terms of data and parameters
}
model {
   // defines the log probability function
}
generated quantities {
   // derived quantities based on parameters, data, and random number generation
}
```

A basic example

Lec25/bernoulli.stan

```
data {
  int<lower=0> N;
  array[N] int<lower=0, upper=1> y;
}
parameters {
  real<lower=0, upper=1> theta;
}
model {
  theta ~ beta(1, 1); // uniform prior on interval 0,1
  y ~ bernoulli(theta);
}
```

Lec25/bernoulli.json

```
{
    "N" : 10,
    "y" : [0,1,0,0,0,0,0,0,0,1]
}
```

Build & fit the model

```
1 from cmdstanpy import CmdStanModel
2 model = CmdStanModel(stan_file='Lec25/bernoulli.stan')
```

```
1 fit = model.sample(data='Lec25/bernoulli.json', show_progress=False)
```

```
1 type(fit)
```

```
cmdstanpy.stanfit.mcmc.CmdStanMCMC
```

```
1 fit
```

```
CmdStanMCMC: model=bernoulli chains=4['method=sample', 'algorithm=hmc', 'adapt', 'engaged=1']
 csv_files:
    /var/folders/v7/wrxd7cdj6l5gzr0191__m9lr0000gr/T/tmpmrtup1np/bernoulli_umwgr17/bernoulli-20250
    /var/folders/v7/wrxd7cdj6l5gzr0191__m9lr0000gr/T/tmpmrtup1np/bernoulli_umwgr17/bernoulli-20250
    /var/folders/v7/wrxd7cdj6l5gzr0191__m9lr0000gr/T/tmpmrtup1np/bernoulli_umwgr17/bernoulli-20250
    /var/folders/v7/wrxd7cdj6l5gzr0191__m9lr0000gr/T/tmpmrtup1np/bernoulli_umwgr17/bernoulli-20250
 output_files:
    /var/folders/v7/wrxd7cdj6l5gzr0191__m9lr0000gr/T/tmpmrtup1np/bernoulli_umwgr17/bernoulli-20250
    /var/folders/v7/wrxd7cdj6l5gzr0191__m9lr0000gr/T/tmpmrtup1np/bernoulli_umwgr17/bernoulli-20250
    /var/folders/v7/wrxd7cdj6l5gzr0191__m9lr0000gr/T/tmpmrtup1np/bernoulli_umwgr17/bernoulli-20250
    /var/folders/v7/wrxd7cdj6l5gzr0191__m9lr0000gr/T/tmpmrtup1np/bernoulli_umwgr17/bernoulli-20250
```

Posterior samples

```
1 fit.stan_variables()
```

```
{'theta': array([0.16229, 0.17837, 0.1677 , 0.3568 , 0.14067, 0.37961, 0.20291, 0.4442
       0.3893 , 0.26481, 0.39696, 0.5198 , 0.47839, 0.08574, 0.27821, 0.21389, 0.2266
       0.19007, 0.11608, 0.43204, 0.35169, 0.30381, 0.30381, 0.12406, 0.12406, 0.1798
```

```
1 np.mean( fit.stan_variables()["theta"] )
```

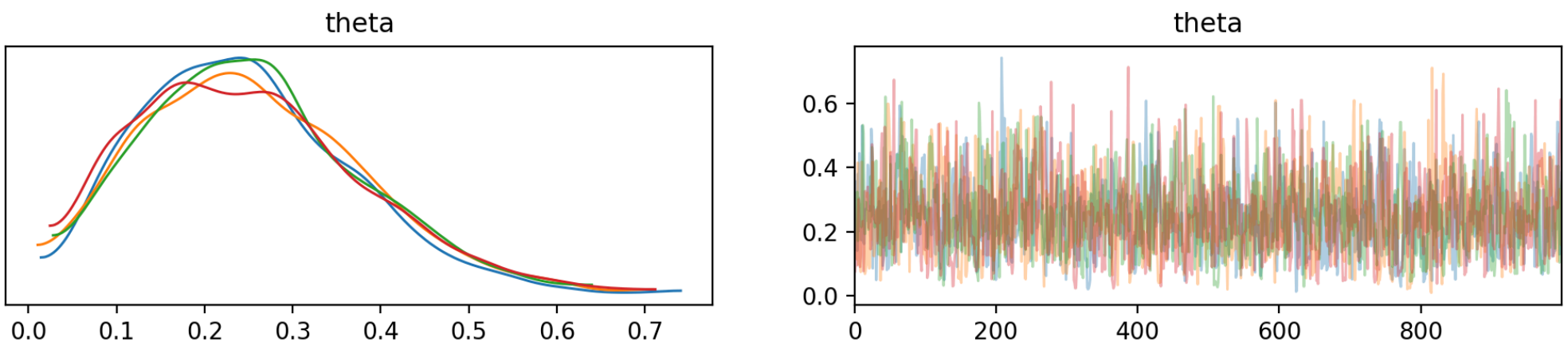
```
np.float64(0.2516746197475)
```

Summary & trace plots

```
1 fit.summary()
```

	Mean	MCSE	StdDev	MAD	5%	50%	95%	ESS_bulk	ESS_tail
lp__	-7.292610	0.020716	0.769255	0.334853	-8.851070	-6.990810	-6.749810	1611.41	1664.41
theta	0.251675	0.003140	0.121115	0.124615	0.076027	0.241017	0.471323	1413.27	1554.67

```
1 ax = az.plot_trace(fit, compact=False)
2 plt.show()
```



Diagnostics

```
1 fit.divergences
```

```
array([0, 0, 0, 0])
```

```
1 fit.max_treedepths
```

```
array([0, 0, 0, 0])
```

```
1 fit.method_variables().keys()
```

```
dict_keys(['lp__', 'accept_stat__', 'stepsize__', 'treedepth__', 'n_leapfrog__', 'divergent__', 'e
```

```
1 print(fit.diagnose())
```

```
Checking sampler transitions treedepth.
Treedepth satisfactory for all transitions.

Checking sampler transitions for divergences.
No divergent transitions found.

Checking E-BFMI - sampler transitions HMC potential energy.
E-BFMI satisfactory.

Rank-normalized split effective sample size satisfactory for all parameters.

Rank-normalized split R-hat values satisfactory for all parameters.

Processing complete, no problems detected.
```

Gaussian process Example

GP model

Lec25/gp.stan

```
data {
  int<lower=1> N;
  array[N] real x;
  vector[N] y;
}
parameters {
  real<lower=0> l;
  real<lower=0> s;
  real<lower=0> nug;
}
model {
  // Covariance
  matrix[N, N] K = gp_exp_quad_cov(x, s, l);
  K = add_diag(K, nug^2);
  matrix[N, N] L = cholesky_decompose(K);

  // priors
  l ~ gamma(2, 1);
  s ~ cauchy(0, 5);
  nug ~ cauchy(0, 1);

  // model
  y ~ multi_normal_cholesky(rep_vector(0, N), L);
```

Fit

```
d = pd.read_csv("data/gp2.csv").to_dict('list')
d["N"] = len(d["x"])
```

```
gp = CmdStanModel(stan_file='Lec25/gp.stan')
gp_fit = gp.sample(data=d, show_progress=False)
```

```
09:17:14 - cmdstanpy - INFO - CmdStan start processing
09:17:14 - cmdstanpy - INFO - Chain [1] start processing
09:17:14 - cmdstanpy - INFO - Chain [2] start processing
09:17:14 - cmdstanpy - INFO - Chain [3] start processing
09:17:14 - cmdstanpy - INFO - Chain [4] start processing
09:17:17 - cmdstanpy - INFO - Chain [4] done processing
09:17:17 - cmdstanpy - INFO - Chain [1] done processing
09:17:17 - cmdstanpy - INFO - Chain [3] done processing
09:17:17 - cmdstanpy - INFO - Chain [2] done processing
09:17:17 - cmdstanpy - WARNING - Non-fatal error during sampling:
Exception: cholesky_decompose: A is not symmetric. A[1,2] = nan, but A[2,1] = nan (in 'gp.stan', l
    Exception: cholesky_decompose: A is not symmetric. A[1,2] = nan, but A[2,1] = nan (in 'gp.star
    Exception: gp_exp_quad_cov: length_scale is 0, but must be positive! (in 'gp.stan', line 13, c
Exception: gp_exp_quad_cov: length_scale is 0, but must be positive! (in 'gp.stan', line 13, colu
Exception: cholesky_decompose: Matrix m is not positive definite (in 'gp.stan', line 15, column 2
    Exception: cholesky_decompose: Matrix m is not positive definite (in 'gp.stan', line 15, colu
    Exception: cholesky_decompose: Matrix m is not positive definite (in 'gp.stan', line 15, colu
Consider re-running with show_console=True if the above output is unclear!
```

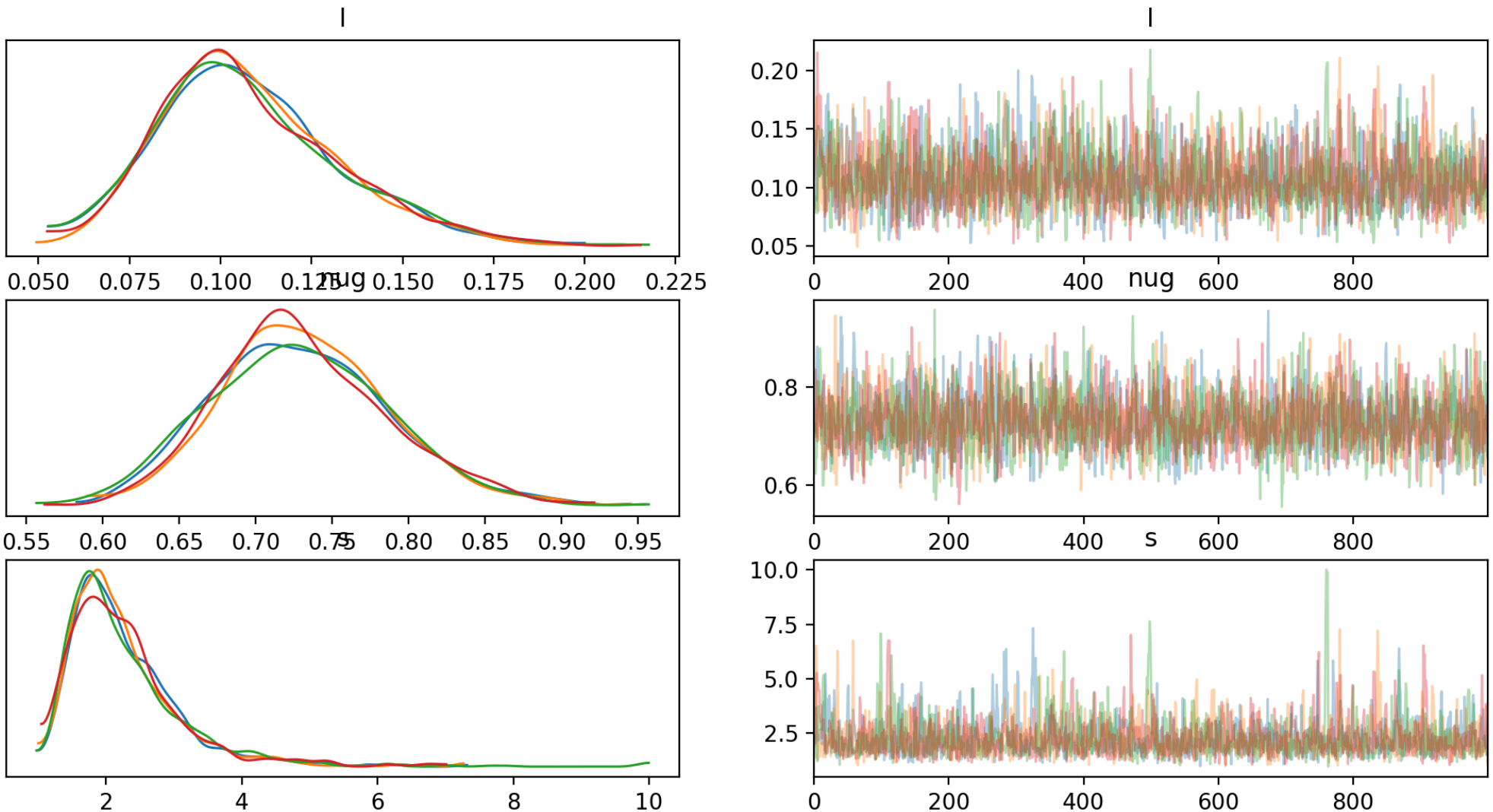
Summary

```
1 gp_fit.summary()
```

	Mean	MCSE	StdDev	MAD	5%	50%	95%	ESS_bulk	ESS_
lp__	-43.020600	0.032322	1.262810	1.073850	-45.525500	-42.711700	-41.604900	1701.51	1828
l	0.108023	0.000618	0.025276	0.023351	0.072006	0.104541	0.154795	1893.82	1925
s	2.263910	0.022461	0.862234	0.633634	1.338890	2.064410	3.828720	1856.83	1641
nug	0.730725	0.001202	0.057528	0.056061	0.641708	0.727218	0.830269	2308.52	2305

Trace plots

```
1  ax = az.plot_trace(gp_fit, compact=False)
2  plt.show()
```



Diagnostics

```
1 gp_fit.divergences
```

```
array([0, 0, 0, 0])
```

```
1 gp_fit.max_treedepths
```

```
array([0, 0, 0, 0])
```

```
1 gp_fit.method_variables().keys()
```

```
dict_keys(['lp__', 'accept_stat__', 'stepsize__', 'treedepth__', 'n_leapfrog__', 'divergent__', 'e
```

```
1 print(gp_fit.diagnose())
```

```
Checking sampler transitions treedepth.
Treedepth satisfactory for all transitions.

Checking sampler transitions for divergences.
No divergent transitions found.

Checking E-BFMI - sampler transitions HMC potential energy.
E-BFMI satisfactory.

Rank-normalized split effective sample size satisfactory for all parameters.

Rank-normalized split R-hat values satisfactory for all parameters.

Processing complete, no problems detected.
```

nutpie & stan

The `nutpie` package can also be used to compile and run stan models, it uses a package called `bridgestan` to interface with stan.

```
1 import nutpie
2 m = nutpie.compile_stan_model(filename="Lec25/gp.stan")
3 m = m.with_data(x=d["x"],y=d["y"],N=len(d["x"]))
4 gp_fit_nutpie = nutpie.sample(m, chains=4)
```

Sampler Progress

Total Chains: 4

Active Chains: 0

Finished Chains: 4

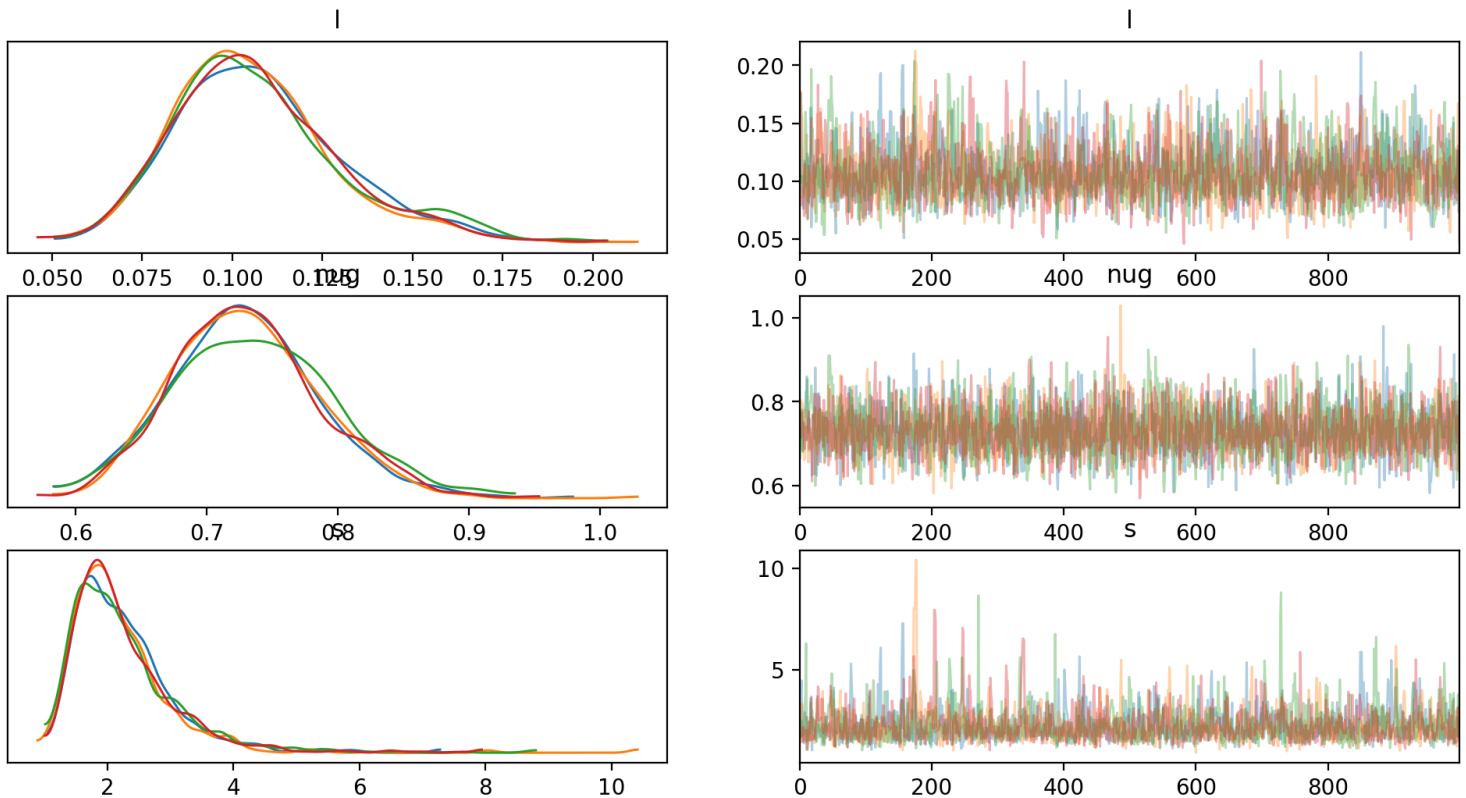
Sampling for now

Estimated Time to Completion: now

Progress	Draws	Divergences	Step Size	Gradients/Draw
	1400	0	0.83	3
	1400	0	0.82	3
	1400	0	0.80	7
	1400	0	0.80	7

```
1 az.summary(gp_fit_nutpie)
```

	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
l	0.107	0.024	0.068	0.158	0.001	0.000	1757.0	2101.0	1.0
nug	0.731	0.057	0.621	0.834	0.001	0.001	3027.0	2450.0	1.0
s	2.240	0.847	1.127	3.672	0.023	0.045	1714.0	1906.0	1.0



Performance

```
%%timeit -r 3
gp_fit = gp.sample(data=d, show_progress=False)
```

```
2.7 s ± 33.8 ms per loop (mean ± std. dev. of 3 runs, 1 loop each)
```

```
%%timeit -r 3
gp_fit_nutpie = nutpie.sample(m, chains=4, progress_bar=False)
```

```
1.2 s ± 17.8 ms per loop (mean ± std. dev. of 3 runs, 1 loop each)
```

Posterior predictive model

Lec25/gp2.stan

```
functions {
  // From https://mc-stan.org/docs/stan-users-guide/gaussian-processes.html#predictive-inference-with-a
  vector gp_pred_rng(
    array[] real x2,
    vector y1,
    array[] real x1,
    real alpha,
    real rho,
    real sigma,
    real delta
  ) {
    int N1 = rows(y1);
    int N2 = size(x2);
    vector[N2] f2;
    {
      matrix[N1, N1] L_K;
      vector[N1] K_div_y1;
      matrix[N1, N2] k_x1_x2;
      matrix[N1, N2] v_pred;
      vector[N2] f2_mu;
      matrix[N2, N2] cov_f2;
      matrix[N2, N2] diag_delta;
      matrix[N1, N1] K;
      K = gp_exp_quad_cov(x1, alpha, rho);
      for (n in 1:N1) {
```

Posterior predictive fit

```
Np = 121
d2 = pd.read_csv("data/gp2.csv").to_dict('list')
d2["N"] = len(d2["x"])
d2["xp"] = np.linspace(0, 1.2, Np)
d2["Np"] = Np
```

```
gp2 = CmdStanModel(stan_file='Lec25/gp2.stan')
gp2_fit = gp2.sample(data=d2, show_progress=False)
```

```
09:17:43 - cmdstanpy - INFO - CmdStan start processing
09:17:43 - cmdstanpy - INFO - Chain [1] start processing
09:17:43 - cmdstanpy - INFO - Chain [2] start processing
09:17:43 - cmdstanpy - INFO - Chain [3] start processing
09:17:43 - cmdstanpy - INFO - Chain [4] start processing
09:17:46 - cmdstanpy - INFO - Chain [4] done processing
09:17:46 - cmdstanpy - INFO - Chain [1] done processing
09:17:46 - cmdstanpy - INFO - Chain [2] done processing
09:17:46 - cmdstanpy - INFO - Chain [3] done processing
09:17:46 - cmdstanpy - WARNING - Non-fatal error during sampling:
Exception: cholesky_decompose: Matrix m is not positive definite (in 'gp2.stan', line 61, column 2
    Exception: cholesky_decompose: Matrix m is not positive definite (in 'gp2.stan', line 61, colu
Exception: gp_exp_quad_cov: sigma is 0, but must be positive! (in 'gp2.stan', line 59, column 2 to
Exception: cholesky_decompose: Matrix m is not positive definite (in 'gp2.stan', line 61, column 2
    Exception: cholesky_decompose: Matrix m is not positive definite (in 'gp2.stan', line 61, colu
    Exception: cholesky_decompose: Matrix m is not positive definite (in 'gp2.stan', line 61, colu
    Exception: cholesky_decompose: Matrix m is not positive definite (in 'gp2.stan', line 61, colu
Exception: cholesky_decompose: Matrix m is not positive definite (in 'gp2.stan', line 61, column 2
Consider re-running with show_console=True if the above output is unclear!
```

Summary

```
1 gp2_fit.summary()
```

	Mean	MCSE	StdDev	MAD	5%	50%	95%	ESS_bulk	ESS
lp__	-42.978800	0.029290	1.230960	1.058280	-45.340000	-42.688600	-41.599400	1835.49	224
l	0.106693	0.000558	0.024443	0.021555	0.071577	0.103845	0.152534	2073.21	180
s	2.217030	0.020762	0.829421	0.598059	1.316430	2.025060	3.716110	2131.79	192
nug	0.730309	0.001118	0.056770	0.055789	0.644002	0.726151	0.828365	2617.77	251
f[1]	3.474290	0.007158	0.436905	0.430087	2.749300	3.469260	4.189870	3727.72	381
...	...	...	...	...	...	...	...	...	...
f[117]	-0.687289	0.033885	2.052400	1.841560	-4.102700	-0.637946	2.538720	3738.04	359
f[118]	-0.653030	0.034672	2.101530	1.872460	-4.197940	-0.578823	2.647560	3747.05	356
f[119]	-0.616564	0.035333	2.143370	1.913370	-4.244260	-0.547837	2.696430	3752.84	356
f[120]	-0.578995	0.035891	2.179080	1.926490	-4.269130	-0.501916	2.816850	3764.33	347
f[121]	-0.541126	0.036365	2.209780	1.911760	-4.319800	-0.484247	2.923980	3768.46	351

125 rows × 10 columns

Draws

```
1 gp2_fit.stan_variable("f").shape
```

```
(4000, 121)
```

```
1 np.mean(gp2_fit.stan_variable("f"), axis=0)
```

```
array([ 3.47429,  3.57525,  3.62487,  3.61779,  3.55041,  3.42122,  3.23093,  2.98257,  2.68135,
       -1.56547, -1.8266 , -2.04792, -2.22885, -2.36928, -2.4694 , -2.52945, -2.54968, -2.53042, -
       -0.30175, -0.0207 ,  0.25158,  0.50953,  0.74746,  0.95963,  1.14046,  1.2848 ,  1.38822,
        0.21679,  0.0817 , -0.02153, -0.08898, -0.11938, -0.11416, -0.07734, -0.01514,  0.06461,
        0.53852,  0.56833,  0.61563,  0.68204,  0.76708,  0.86828,  0.98154,  1.10165,  1.22279,
        1.33125,  1.18103,  1.0112 ,  0.8271 ,  0.63462,  0.43976,  0.24831,  0.06554, -0.10406, -
       -0.74328, -0.71794, -0.68729, -0.65303, -0.61656, -0.579  , -0.54113])
```

Plot