MCMC - Performance

Lecture 25

Dr. Colin Rundel

Stan

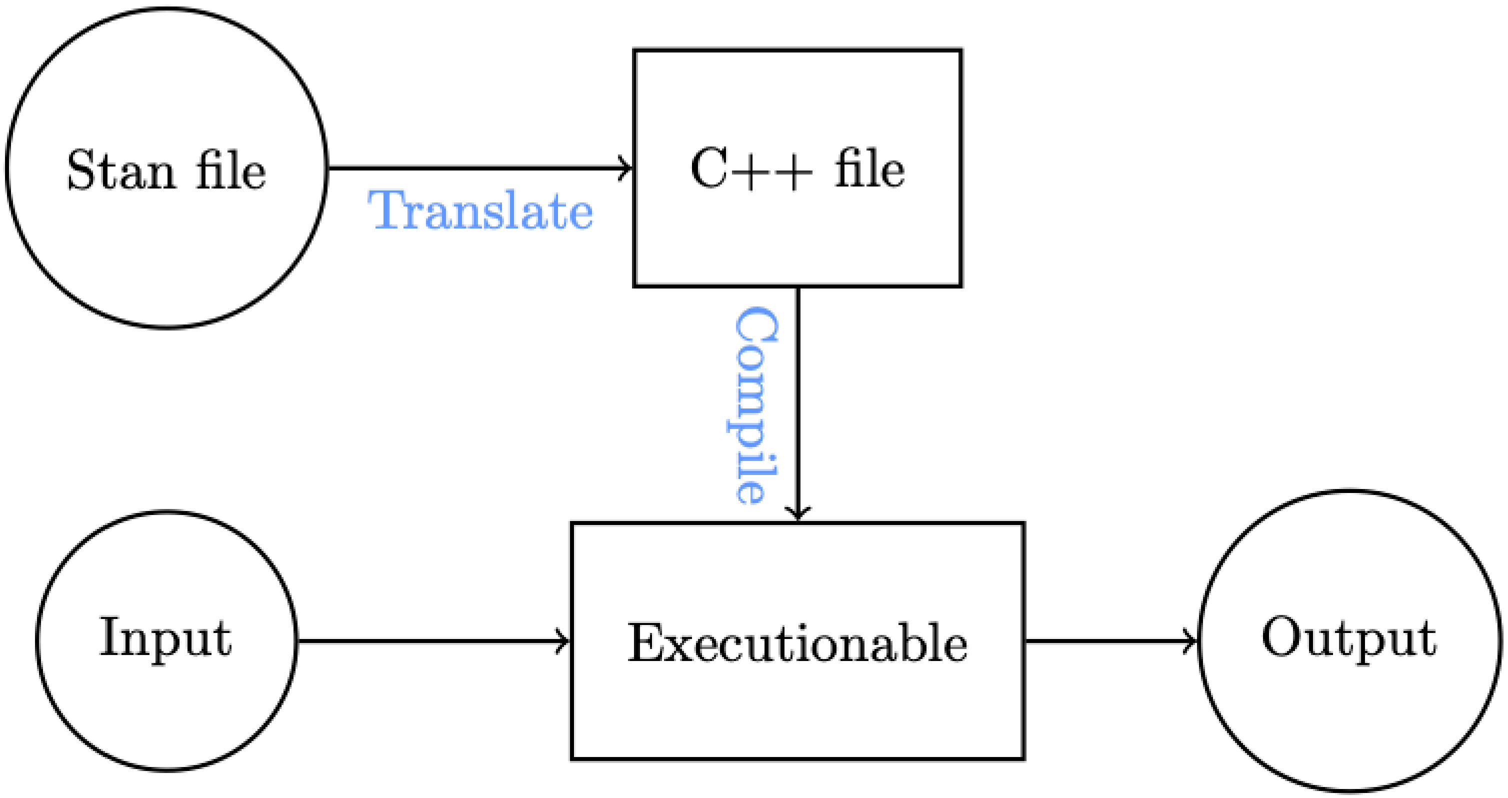
Stan in Python & R

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 transforms of the data
}
parameters {
   // declares the model's parameters
}
transformed parameters {
   // allows variables to be defined in terms of data and parameters
}
model {
   // defines the log probability function
}
generated quantities {
   // allows 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

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

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

```python
type(fit)
```

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

```python
fit
```

```
CmdStanMCMC: model=bernoulli chains=4['method=sample', 'algorithm=hmc', 'adapt', 'engaged=1']
 csv_files:
    /var/folders/v7/wrxd7cdj6l5gzr0191__m9lr0000gr/T/tmp17qeq763/bernoullixy6zw4di/bernoulli-2025
    /var/folders/v7/wrxd7cdj6l5gzr0191__m9lr0000gr/T/tmp17qeq763/bernoullixy6zw4di/bernoulli-2025
    /var/folders/v7/wrxd7cdj6l5gzr0191__m9lr0000gr/T/tmp17qeq763/bernoullixy6zw4di/bernoulli-2025
    /var/folders/v7/wrxd7cdj6l5gzr0191__m9lr0000gr/T/tmp17qeq763/bernoullixy6zw4di/bernoulli-2025
 output_files:
    /var/folders/v7/wrxd7cdj6l5gzr0191__m9lr0000gr/T/tmp17qeq763/bernoullixy6zw4di/bernoulli-2025
    /var/folders/v7/wrxd7cdj6l5gzr0191__m9lr0000gr/T/tmp17qeq763/bernoullixy6zw4di/bernoulli-2025
    /var/folders/v7/wrxd7cdj6l5gzr0191__m9lr0000gr/T/tmp17qeq763/bernoullixy6zw4di/bernoulli-2025
    /var/folders/v7/wrxd7cdj6l5gzr0191__m9lr0000gr/T/tmp17qeq763/bernoullixy6zw4di/bernoulli-2025
```

Posterior samples

```
fit.stan_variables()
```

```
{'theta': array([0.23173, 0.22212, 0.20124, 0.19168, 0.23014, 0.18065, 0.20952, 0.4903
        0.67739, 0.10433, 0.30922, 0.13858, 0.13503, 0.09625, 0.17158, 0.18875, 0.2777
        0.22989, 0.09783, 0.22426, 0.28606, 0.24765, 0.16021, 0.13769, 0.2667 , 0.2396
```

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

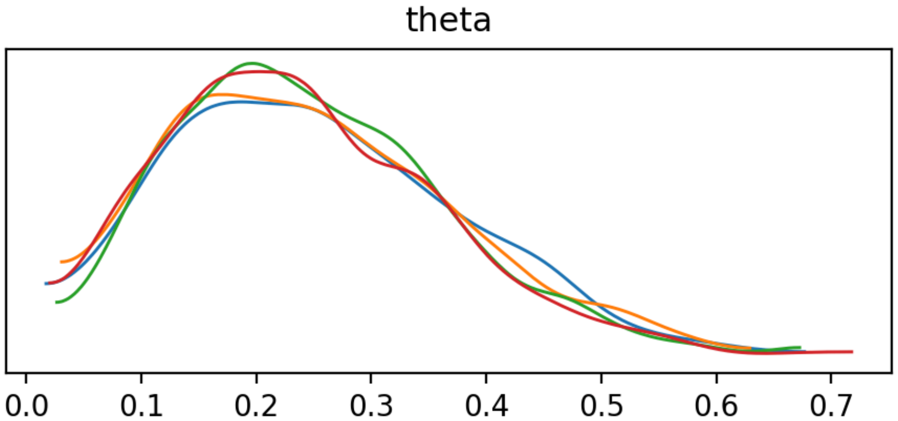
```
np.float64(0.249208613775)
```

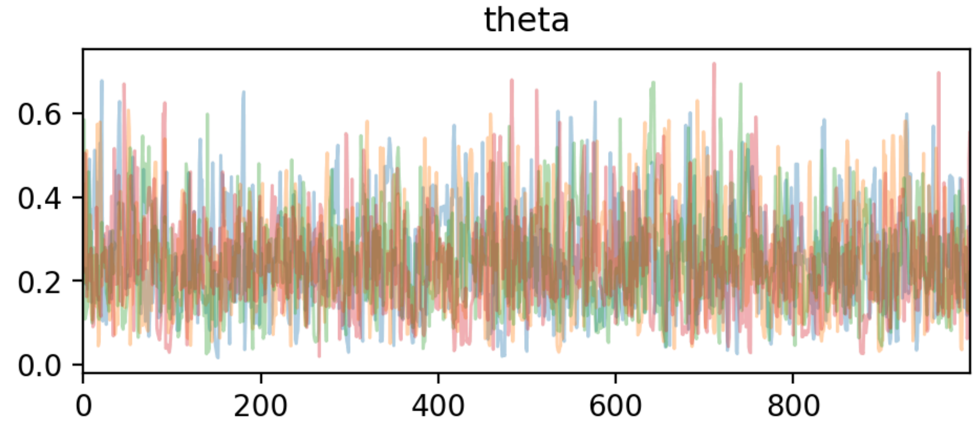
Summary & trace plots

```
fit.summary()
```

	Mean	MCSE	StdDev	MAD	5%	50%	95%	ESS_bulk	ESS_tail
lp__	-7.297180	0.021493	0.756099	0.343696	-8.887550	-6.999090	-6.749900	1461.40	1516.84
theta	0.249209	0.003271	0.121830	0.126722	0.076239	0.235845	0.472208	1350.19	1565.43

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





Diagnostics

```
fit.divergences
```

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

```
fit.max_treedepths
```

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

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

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

```
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;
}
transformed data {
  array[N] real xn = to_array_1d(x);
  vector[N] zeros = rep_vector(0, N);
}
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);
  matrix[N, N] L = cholesky_decompose(add_diag(K, nug^2));
  // priors
  l ~ gamma(2, 1);
  s ~ cauchy(0, 5);
  nug ~ cauchy(0, 1);
  // model
```

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)
```

```
12:26:11 - cmdstanpy - INFO - CmdStan start processing
12:26:11 - cmdstanpy - INFO - Chain [1] start processing
12:26:11 - cmdstanpy - INFO - Chain [2] start processing
12:26:11 - cmdstanpy - INFO - Chain [3] start processing
12:26:11 - cmdstanpy - INFO - Chain [4] start processing
12:26:14 - cmdstanpy - INFO - Chain [1] done processing
12:26:14 - cmdstanpy - INFO - Chain [2] done processing
12:26:14 - cmdstanpy - INFO - Chain [3] done processing
12:26:14 - cmdstanpy - INFO - Chain [4] done processing
12:26:14 - cmdstanpy - WARNING - Non-fatal error during sampling:
Exception: cholesky_decompose: Matrix m is not positive definite (in 'gp.stan', line 18, column 2
Exception: cholesky_decompose: A is not symmetric. A[1,2] = inf, but A[2,1] = inf (in 'gp.stan', l
Consider re-running with show_console=True if the above output is unclear!
```

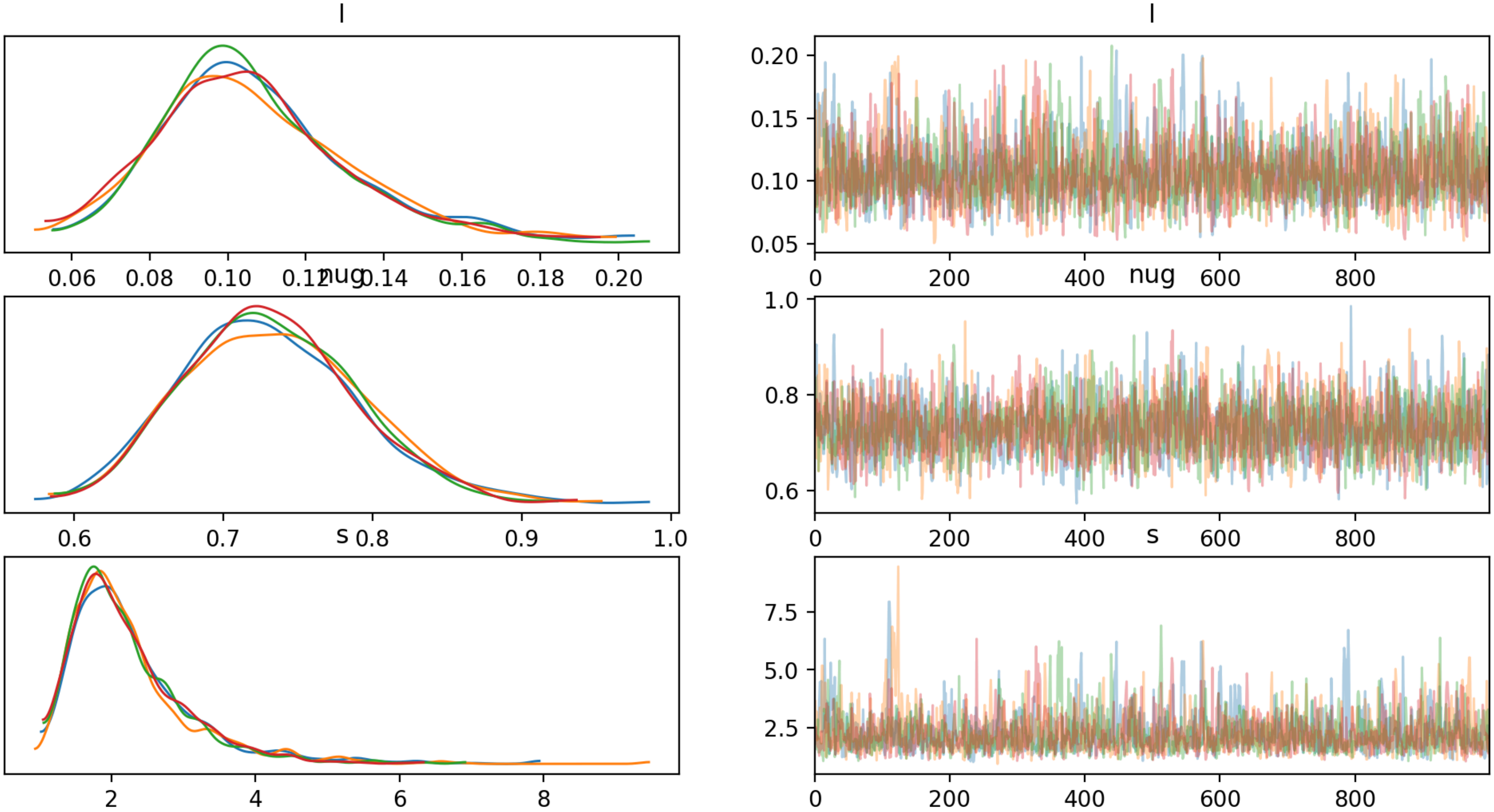
Summary

```
gp_fit.summary()
```

	Mean	MCSE	StdDev	MAD	5%	50%	95%	ESS_bulk	ESS_
lp__	-43.053700	0.030153	1.245150	1.093420	-45.492900	-42.773500	-41.615000	1766.64	2202
l	0.107124	0.000626	0.025045	0.022222	0.071451	0.103657	0.155039	1849.74	1485
s	2.251420	0.023971	0.846239	0.621499	1.325730	2.044350	3.884280	1737.87	1696
nug	0.731742	0.001231	0.057939	0.058689	0.642944	0.728222	0.832110	2228.94	2104

Trace plots

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



Diagnostics

```
gp_fit.divergences
```

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

```
gp_fit.max_treedepths
```

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

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

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

```
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.

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

Sampler Progress

Total Chains: 4

Active Chains: 0

Finished Chains: 4

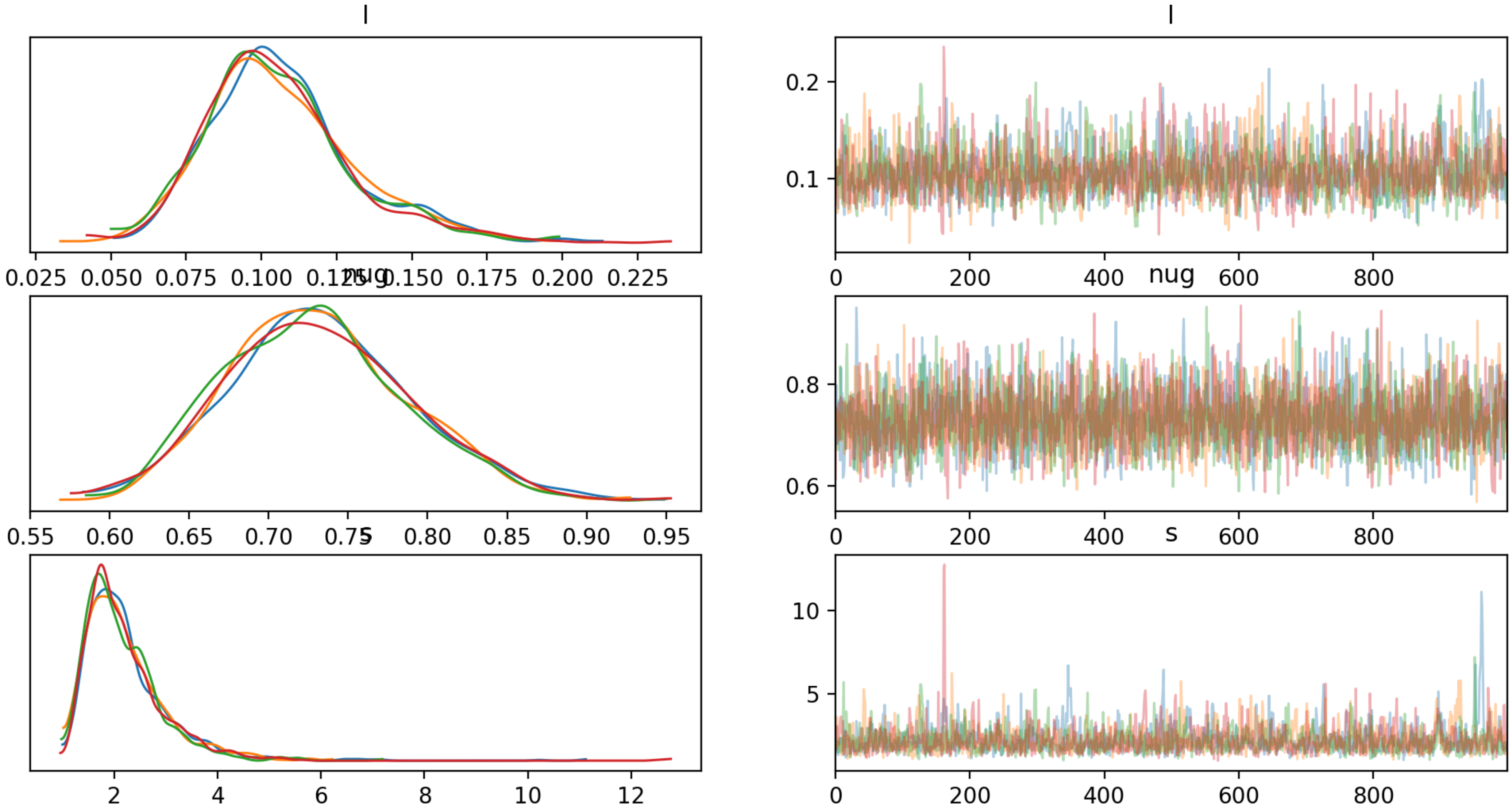
Sampling for now

Estimated Time to Completion: now

Progress	Draws	Divergences	Step Size	Gradients/Draw
	1400	0	0.77	3
	1400	0	0.83	7
	1400	0	0.83	3
	1400	0	0.78	1

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

	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
l	0.106	0.025	0.065	0.156	0.001	0.001	1614.0	1502.0	1.0
nug	0.732	0.058	0.631	0.841	0.001	0.001	3179.0	2751.0	1.0
s	2.199	0.819	1.108	3.612	0.023	0.052	1715.0	1637.0	1.0


l
l
0.025
0.050
0.075
0.100
0.125
0.150
0.175
0.200
0.225
0.2
0.1
0
200
400
600
800
nug
nug
0.55
0.60
0.65
0.70
0.75
0.80
0.85
0.90
0.95
0.8
0.6
s
s
2
4
6
8
10
12
10
5

Performance

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

```
2.66 s ± 39.5 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 ± 36.2 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) {
        K[n, n] = K[n, n] + square(sigma);
      }
```

Posterior predictive fit

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

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

```
12:26:38 - cmdstanpy - INFO - CmdStan start processing
12:26:38 - cmdstanpy - INFO - Chain [1] start processing
12:26:38 - cmdstanpy - INFO - Chain [2] start processing
12:26:38 - cmdstanpy - INFO - Chain [3] start processing
12:26:38 - cmdstanpy - INFO - Chain [4] start processing
12:26:41 - cmdstanpy - INFO - Chain [4] done processing
12:26:41 - cmdstanpy - INFO - Chain [1] done processing
12:26:41 - cmdstanpy - INFO - Chain [2] done processing
12:26:41 - cmdstanpy - INFO - Chain [3] done processing
12:26:41 - cmdstanpy - WARNING - Non-fatal error during sampling:
Exception: gp_exp_quad_cov: sigma is 0, but must be positive! (in 'gp2.stan', line 57, column 2 tc
    Exception: cholesky_decompose: Matrix m is not positive definite (in 'gp2.stan', line 58, colu
Exception: cholesky_decompose: Matrix m is not positive definite (in 'gp2.stan', line 58, column 2
    Exception: cholesky_decompose: Matrix m is not positive definite (in 'gp2.stan', line 58, colu
    Exception: cholesky_decompose: Matrix m is not positive definite (in 'gp2.stan', line 58, colu
Exception: cholesky_decompose: A is not symmetric. A[1,2] = nan, but A[2,1] = nan (in 'gp2.stan',
Exception: cholesky_decompose: Matrix m is not positive definite (in 'gp2.stan', line 58, column 2
    Exception: cholesky_decompose: Matrix m is not positive definite (in 'gp2.stan', line 58, colu
Consider re-running with show_console=True if the above output is unclear!
```

Summary

```
gp2_fit.summary()
```

	Mean	MCSE	StdDev	MAD	5%	50%	95%	ESS_bulk	ESS
lp__	-42.990000	0.029041	1.211980	1.056870	-45.319200	-42.713200	-41.602200	1760.07	247
l	0.106265	0.000531	0.024428	0.021857	0.071393	0.103104	0.154018	2194.15	203
s	2.186620	0.017279	0.788980	0.611246	1.306630	2.006460	3.686130	2369.93	231
nug	0.733819	0.001182	0.057063	0.056098	0.647260	0.730583	0.833775	2394.42	235
f[1]	3.462920	0.007542	0.444534	0.440503	2.735510	3.458080	4.194200	3495.86	377
...	...	...	...	...	...	...	...	...	...
f[117]	-0.615250	0.034740	2.064320	1.883500	-4.093470	-0.520666	2.522990	3585.17	342
f[118]	-0.576913	0.035629	2.116880	1.945550	-4.133100	-0.489661	2.617080	3583.75	358
f[119]	-0.536599	0.036372	2.162950	1.961060	-4.132490	-0.457685	2.758790	3593.85	348
f[120]	-0.495595	0.036972	2.202820	2.009690	-4.165190	-0.406080	2.837920	3606.30	345
f[121]	-0.454846	0.037435	2.236710	2.003730	-4.173390	-0.374889	2.936080	3623.48	348

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([ 3.46292,  3.56565,  3.61681,  3.6111 ,  3.54499,  3.41699,  3.22782,  2.98046,  2.68006,
       -1.56017, -1.82008, -2.04044, -2.22082, -2.36131, -2.46217, -2.52372, -2.54615, -2.52965, -
       -0.30188, -0.0181 ,  0.25633,  0.51576,  0.75446,  0.96675,  1.14713,  1.29061,  1.39297,
        0.22625,  0.09066, -0.0135 , -0.0821 , -0.11361, -0.10933, -0.07319, -0.01143,  0.06801,
        0.53952,  0.5692 ,  0.61627,  0.68238,  0.76707,  0.86797,  0.98107,  1.1012 ,  1.22252,
        1.32888,  1.18025,  1.01261,  0.83127,  0.64207,  0.45098,  0.26377,  0.08558, -0.07925, -
       -0.67927, -0.65   , -0.61525, -0.57691, -0.5366 , -0.4956 , -0.45485])
```

Plot