Lecture 21

Poisson GLMs to Gaussian processes

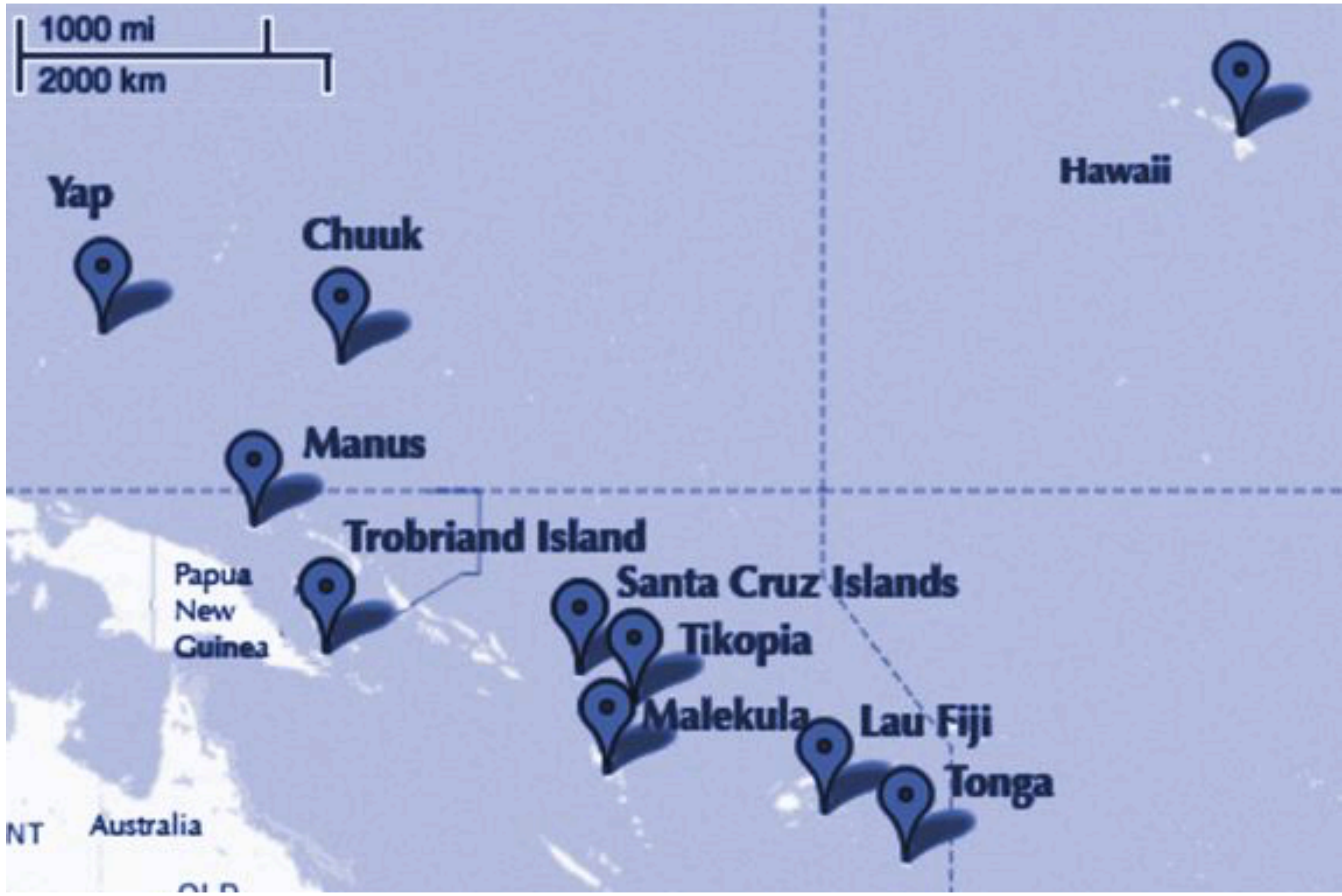
Important Dates

- Project Due Tue Dec 11th 11.59PM
- Strongly encourage you to finish by Sat Dec 8th or Sun Dec 9th by 11.59PM
- …or you might hurt your final…
- Final will be released Thu Dec 6th 11.59PM
- Final will be due Monday Dec 17th 11.59PM

Oceanic Tools

From Mcelreath:

The island societies of Oceania provide a natural experiment in technological evolution. Different historical island populations possessed tool kits of different size. These kits include fish hooks, axes, boats, hand plows, and many other types of tools. A number of theories predict that larger populations will both develop and sustain more complex tool kits. So the natural variation in population size induced by natural variation in island size in Oceania provides a natural experiment to test these ideas. It's also suggested that contact rates among populations effectively increase population size, as it's relevant to technological evolution. So variation in contact rates among Oceanic societies is also relevant. (McElreath 313)

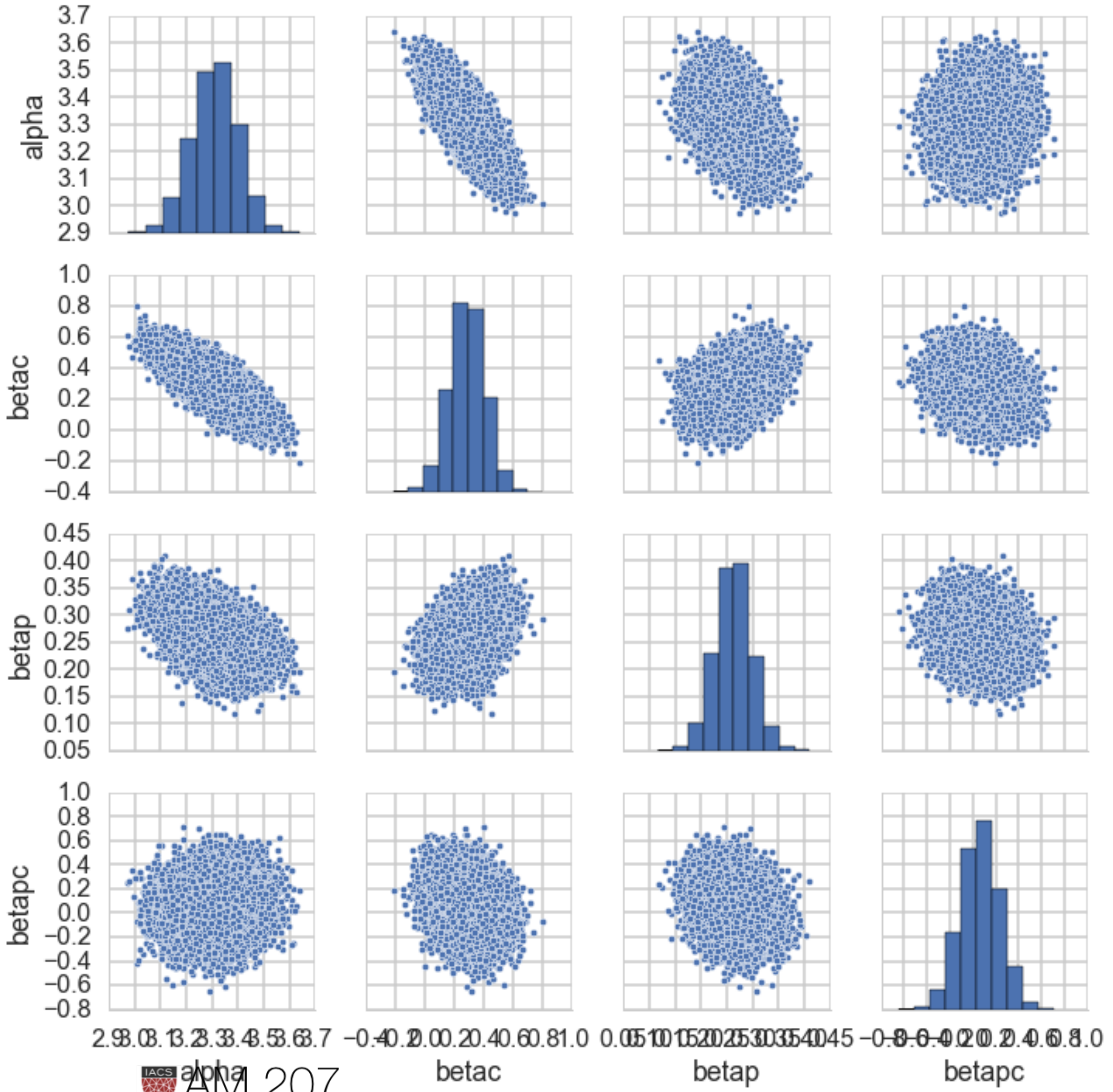


Model M1

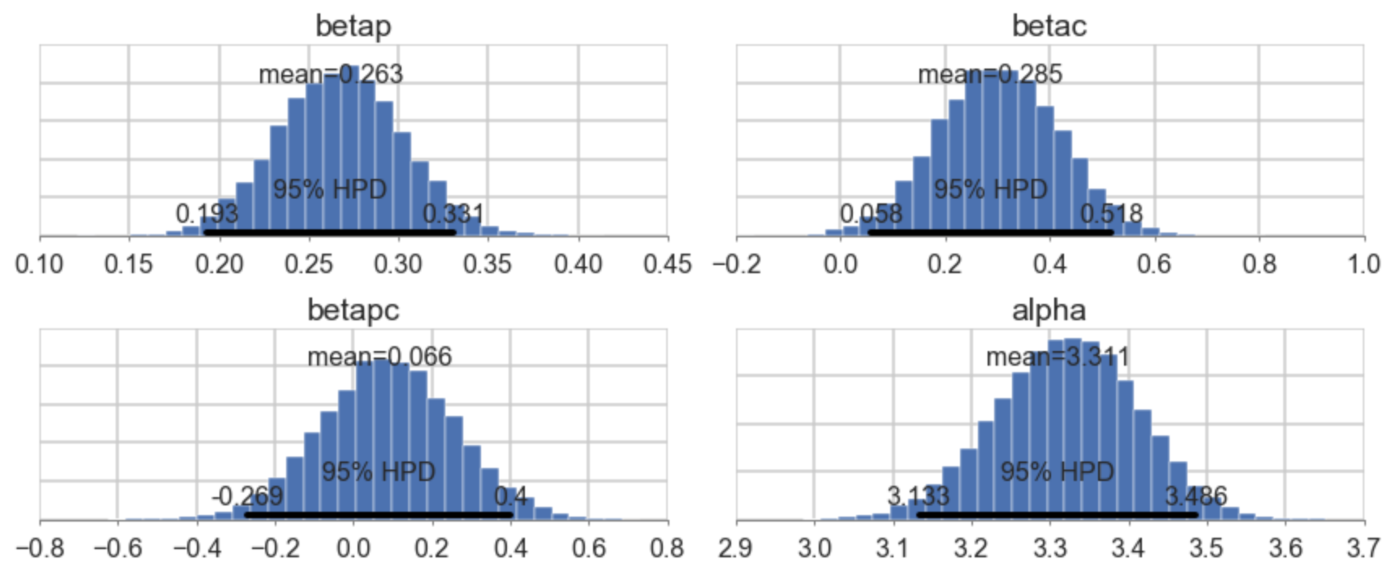
$$\begin{aligned} T_i &\sim Poisson(\lambda_i) \\ log(\lambda_i) &= \alpha + \beta_P log(P(c)_i) + \beta_C C_i + \beta_{PC} C_i log(P(c)_i) \\ \alpha &\sim N(0, 100) \\ \beta_P &\sim N(0, 1) \\ \beta_C &\sim N(0, 1) \\ \beta_{PC} &\sim N(0, 1) \end{aligned}$$

```python
df.logpop_c = df.logpop - df.logpop.mean()
with pm.Model() as m1c:
    betap = pm.Normal("betap", 0, 1)
    betac = pm.Normal("betac", 0, 1)
    betapc = pm.Normal("betapc", 0, 1)
    alpha = pm.Normal("alpha", 0, 100)
    loglam = alpha + betap*df.logpop_c + betac*df.clevel + betapc*df.clevel*df.logpop_c
    y = pm.Poisson("ntools", mu=t.exp(loglam), observed=df.total_tools)
```

```
{'alpha': 7978.0, 'betac': 7898.0, 'betap': 13621.0, 'betapc': 17703.0}
```



- better constrained, less correlated, sampling faster and better
- clear effect of contact, effect of interaction not clear yet
- will use model comparison next time for this!

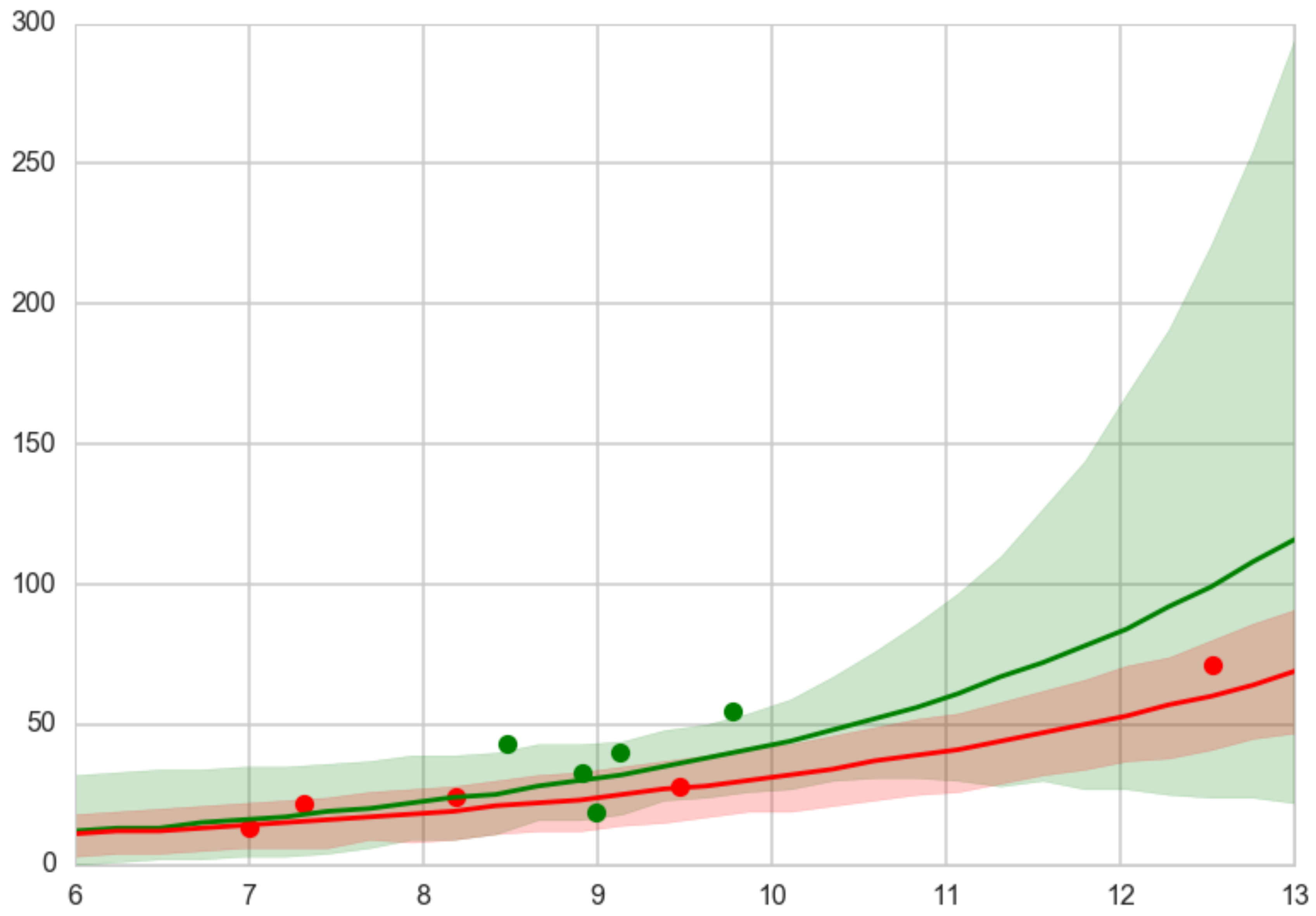


Computing a Posterior Predictive

```
def trace_or_zero(trace, name):
    if name in trace.varnames:
        return trace[name]
    else:
        return np.zeros(2*len(trace))

def compute_pp(lpgrid, trace, contact=0):
    alphatrace = trace_or_zero(trace, 'alpha')
    betaptrace = trace_or_zero(trace, 'betap')
    betactrace = trace_or_zero(trace, 'betac')
    betapctrace = trace_or_zero(trace, 'betapc')
    tl=2*len(trace)
    gl=lpgrid.shape[0]
    lam = np.empty((gl, tl))
    lpgrid_c = lpgrid - lpgrid.mean()
    for i, v in enumerate(lpgrid):
        temp = alphatrace + betaptrace*lpgrid_c[i] + betactrace*contact + betapctrace*contact*lpgrid_c[i]
        lam[i,:] = poisson.rvs(np.exp(temp))
    return lam
```

Counterfactual Posterior predictive

```
lpgrid = np.linspace(6,13,30)
pplow = compute_pp(lpgrid, trace1c)
pphigh = compute_pp(lpgrid, trace1c, contact=1)
We compute the medians and the hpds, and plot these against the data

pplowmed = np.median(pplow, axis=1)
pplowhpd = pm.stats.hpd(pplow.T)
pphighmed = np.median(pphigh, axis=1)
pphighhpd = pm.stats.hpd(pphigh.T)
```



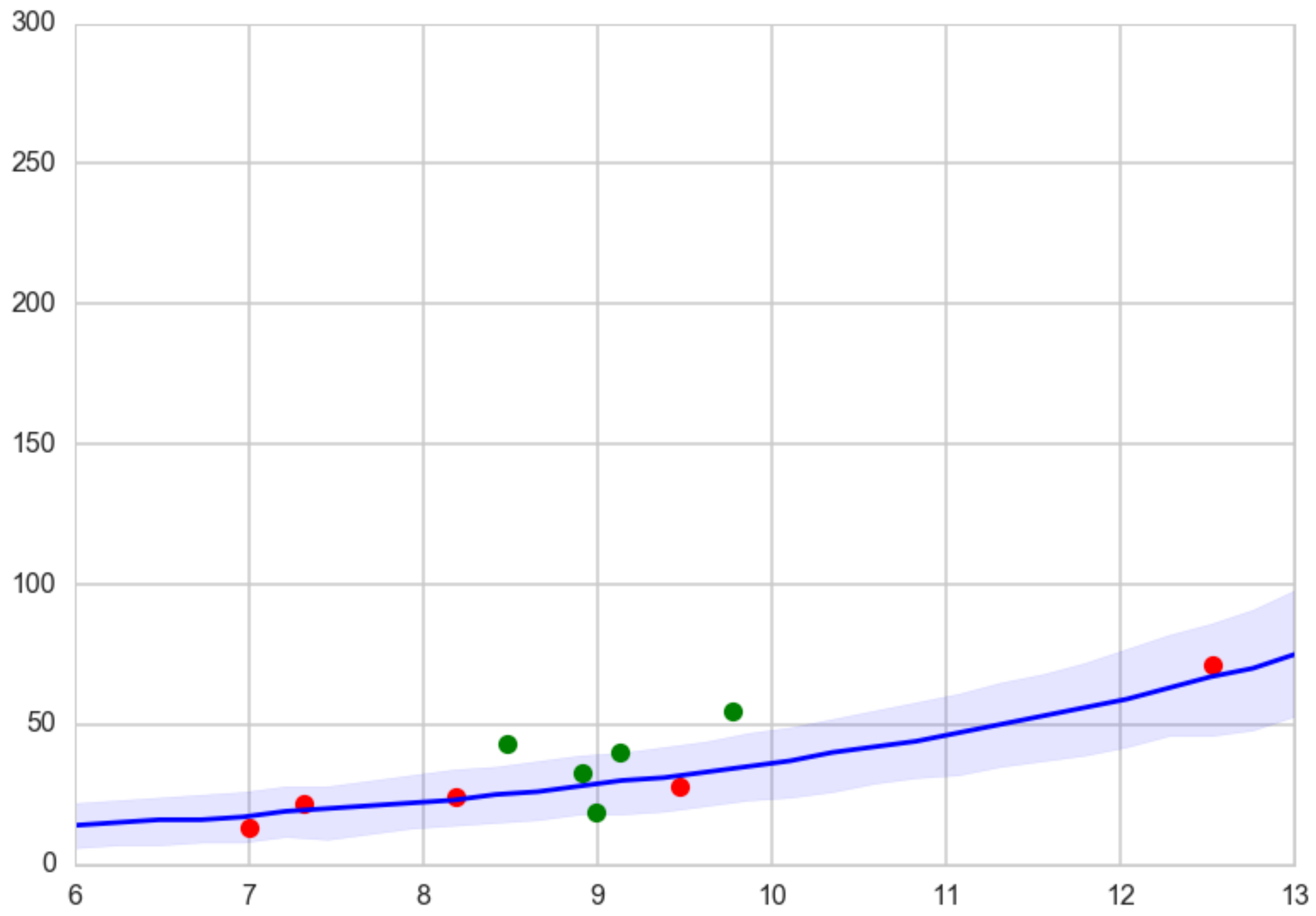
Overdispersion for only p: Model M2

$$T_i \sim Poisson(\lambda_i)$$

$$log(\lambda_i) = \alpha + \beta_P(log(P(c)_i)$$

```
m2c_onlyp: loglam = alpha + betap*df.logpop_c
```

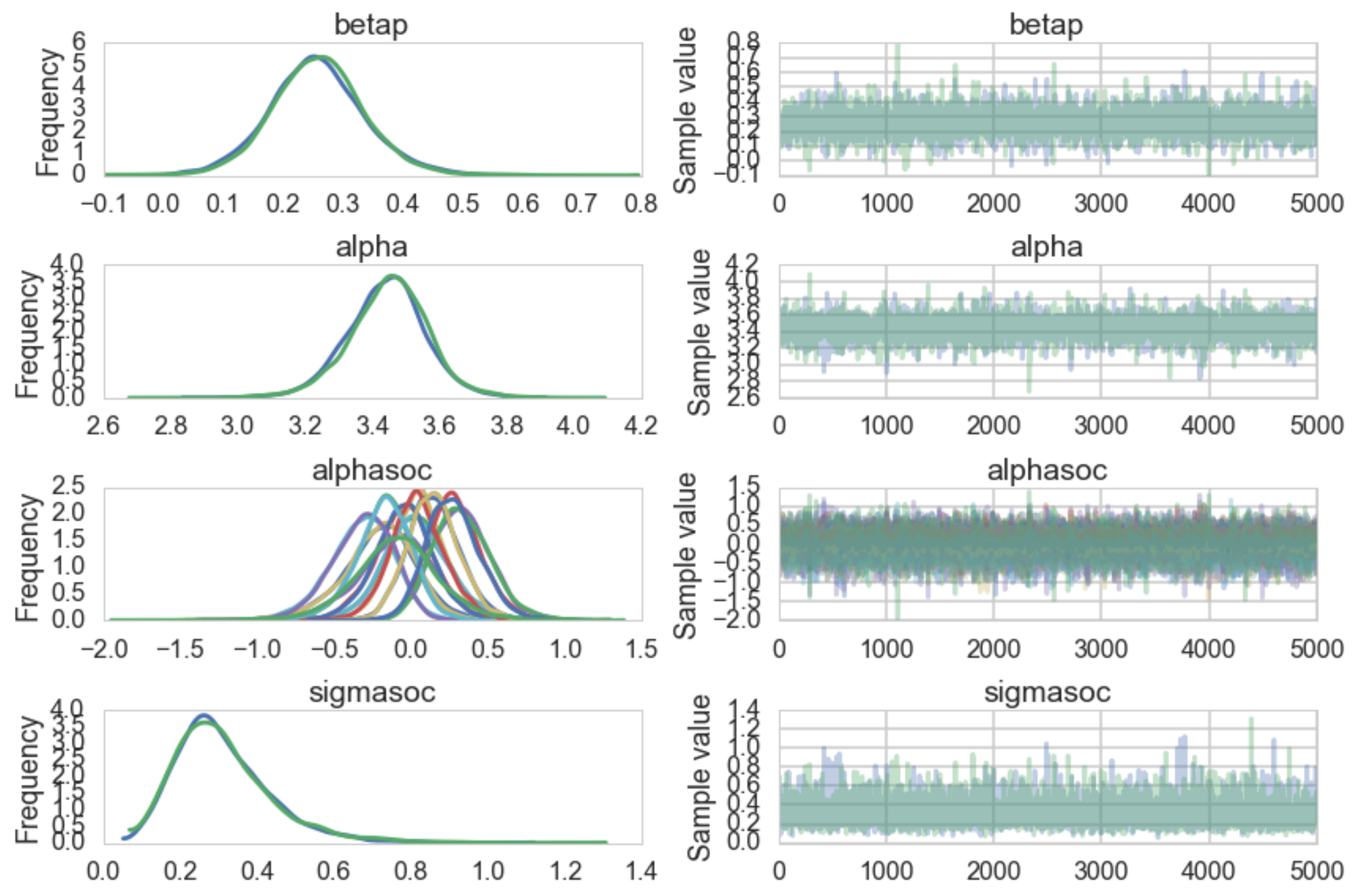
```
ppvar/ppmean
array([ 1.30421519,  1.26489919,  1.2693647 ,  1.20461164,  1.25536688,
        1.19957498,  1.1793642 ,  1.17456651,  1.14728935,  1.15605154,
        1.09427345,  1.12326509,  1.13490696,  1.09674559,  1.12324651,
        1.10038695,  1.11064864,  1.11575808,  1.11499395,  1.14792792,
        1.15350445,  1.18526221,  1.22732124,  1.29480132,  1.30994366,
        1.41243637,  1.48317552,  1.58211591,  1.67981142,  1.79674707])
```

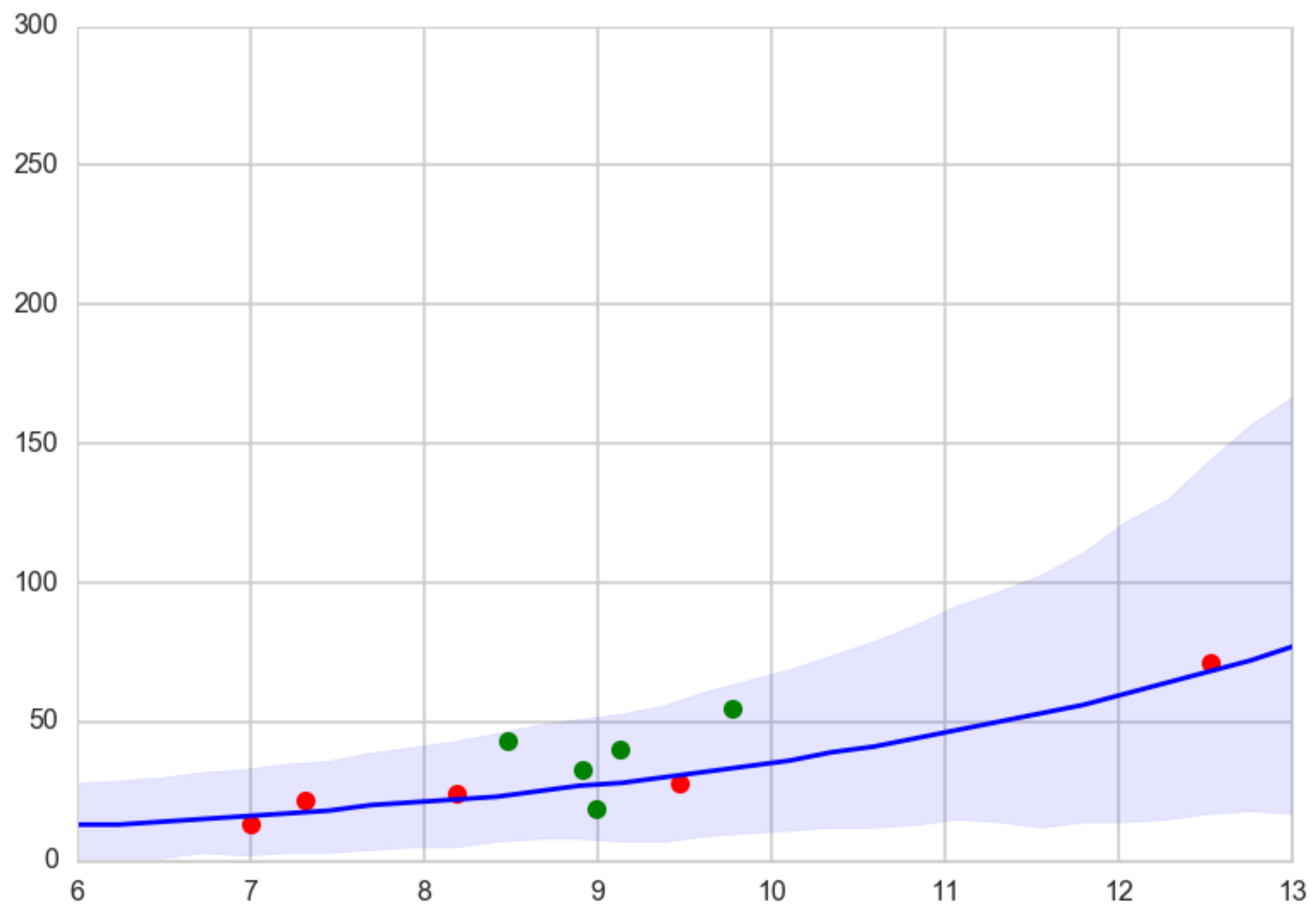


Varying hierarchical intercepts model

```
with pm.Model() as m3c:
    betap = pm.Normal("betap", 0, 1)
    alpha = pm.Normal("alpha", 0, 100)
    sigmasoc = pm.HalfCauchy("sigmasoc", 1)
    alphasoc = pm.Normal("alphasoc", 0, sigmasoc, shape=df.shape[0])
    loglam = alpha + alphasoc + betap*df.logpop_c
    y = pm.Poisson("ntools", mu=t.exp(loglam), observed=df.total_tools)
    trace3 = pm.sample(6000, tune=1000, nuts_kwargs=dict(target_accept=.95))
```

Hierarchical Model Posterior predictive





much wider, includes data areas

Trace Summary

```
                  mean        sd          mc_error     hpd_2.5     hpd_97.5    n_eff     Rhat

       betap     0.258089    0.082296    0.001338     0.096656    0.424239    4166.0    0.999973
       alpha     3.446465    0.120746    0.002027     3.199103    3.687472    3653.0    0.999921
 alphasoc__0    -0.209619    0.247940    0.003688    -0.718741    0.259506    4968.0    1.000043
 alphasoc__1     0.038430    0.219664    0.002941    -0.404914    0.487200    5961.0    0.999917
 alphasoc__2    -0.050901    0.195434    0.002468    -0.447657    0.339753    5818.0    0.999921
 alphasoc__3     0.324157    0.189557    0.002763    -0.031798    0.699002    4321.0    0.999929
 alphasoc__4     0.039406    0.175986    0.002227    -0.301135    0.401451    6167.0    1.000062
 alphasoc__5    -0.320429    0.208348    0.003087    -0.733230    0.055638    4967.0    0.999927
 alphasoc__6     0.144230    0.172236    0.002496    -0.168542    0.513625    5458.0    0.999972
 alphasoc__7    -0.174227    0.184070    0.002252    -0.568739    0.162993    6696.0    0.999919
 alphasoc__8     0.273610    0.174185    0.002854    -0.050347    0.627762    4248.0    1.000032
 alphasoc__9    -0.088533    0.291865    0.004870    -0.679972    0.487844    4385.0    0.999929
    sigmasoc     0.312019    0.129527    0.002224     0.098244    0.588338    2907.0    0.999981
```

Model Correlations

What if we model the correlation between societies based on the distance between them?

How?

Replace independent intercepts by correlated ones.

Draw from a Multivariate Normal with a modeled covariance matrix.

We can model society specific intercepts for oceanic tools as draws from a 0 mean MVN.

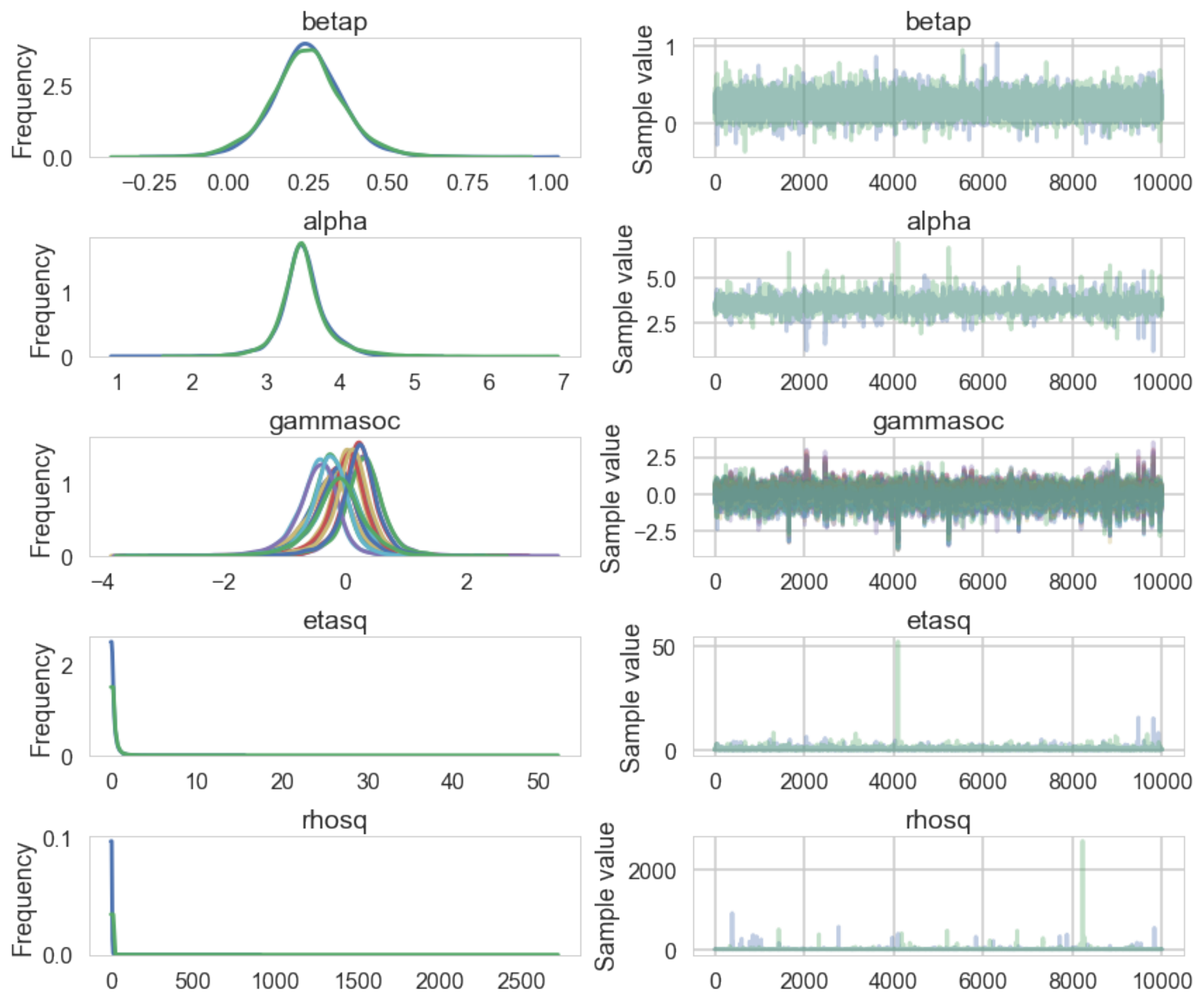
```
with pm.Model() as mgc:
    betap = pm.Normal("betap", 0, 1)
    alpha = pm.Normal("alpha", 0, 10)
    etasq = pm.HalfCauchy("etasq", 1)
    rhosq = pm.HalfCauchy("rhosq", 1)
    means=tt.stack([0.0]*10)
    sigma_matrix = tt.nlinalg.diag([0.01]*10)
    cov=tt.exp(-rhosq*dij*dij)*etasq + sigma_matrix
    gammasoc = pm.MvNormal("gammasoc", means,
        cov=cov, shape=df.shape[0])
    loglam = alpha + gammasoc + betap*df.logpop_c
    y = pm.Poisson("ntools", mu=t.exp(loglam),
        observed=df.total_tools)
    mgctrace = pm.sample(10000, tune=2000,
        nuts_kwargs=dict(target_accept=.95))
```

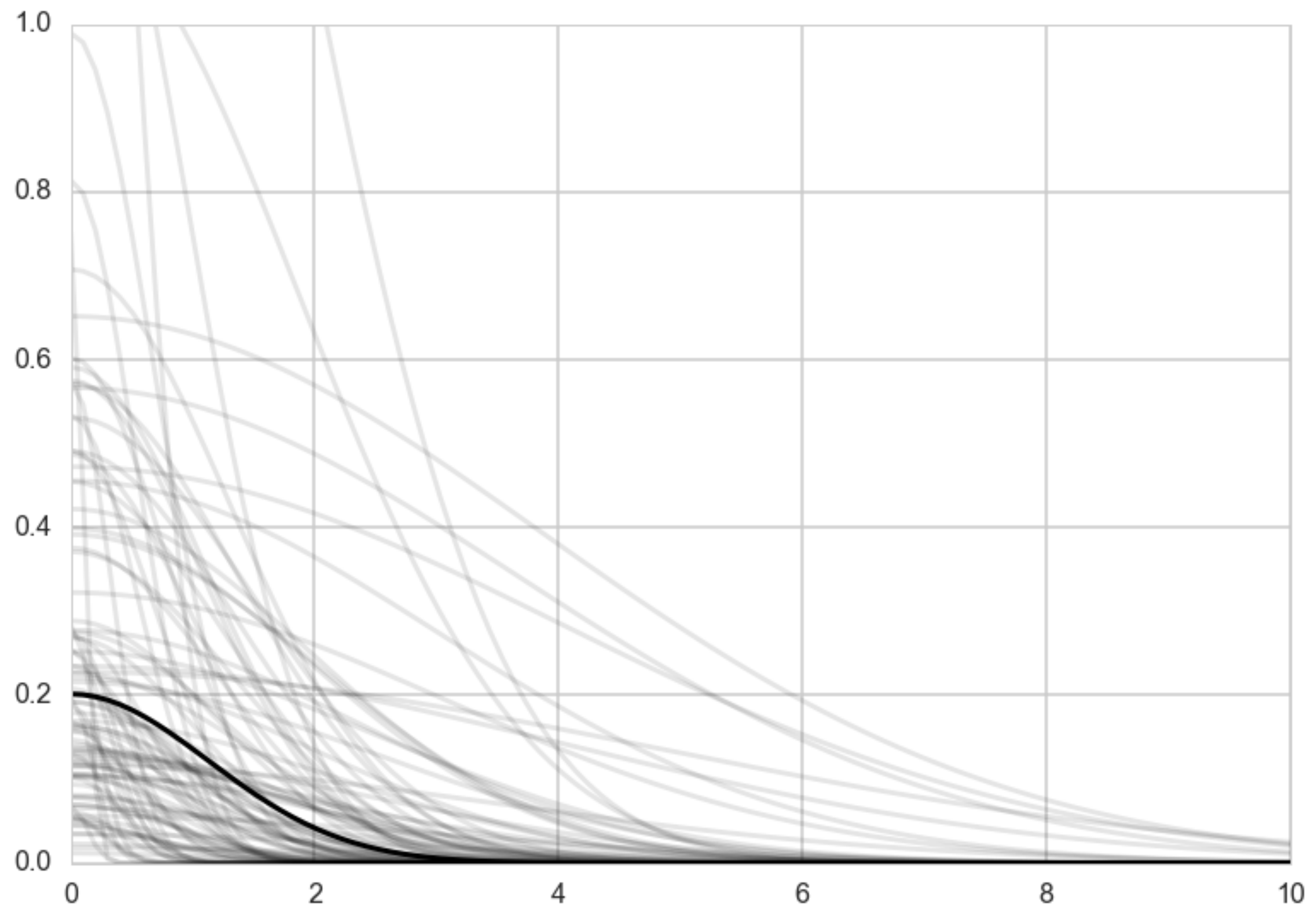
$$
\begin{aligned}
T_i &\sim \text{Poisson}(\lambda_i) \\
\log \lambda_i &= \alpha + \gamma_{\text{SOCIETY}[i]} + \beta_P \log P_i \\
\gamma &\sim \text{MVNormal}((0, \ldots, 0), \text{K}) \\
\text{K}_{ij} &= \eta^2 \exp(-\rho^2 D_{ij}^2) + \delta_{ij}(0.01) \\
\alpha &\sim \text{Normal}(0, 10) \\
\beta_P &\sim \text{Normal}(0, 1) \\
\eta^2 &\sim \text{HalfCauchy}(0, 1) \\
\rho^2 &\sim \text{HalfCauchy}(0, 1)
\end{aligned}
$$

Trace Summary

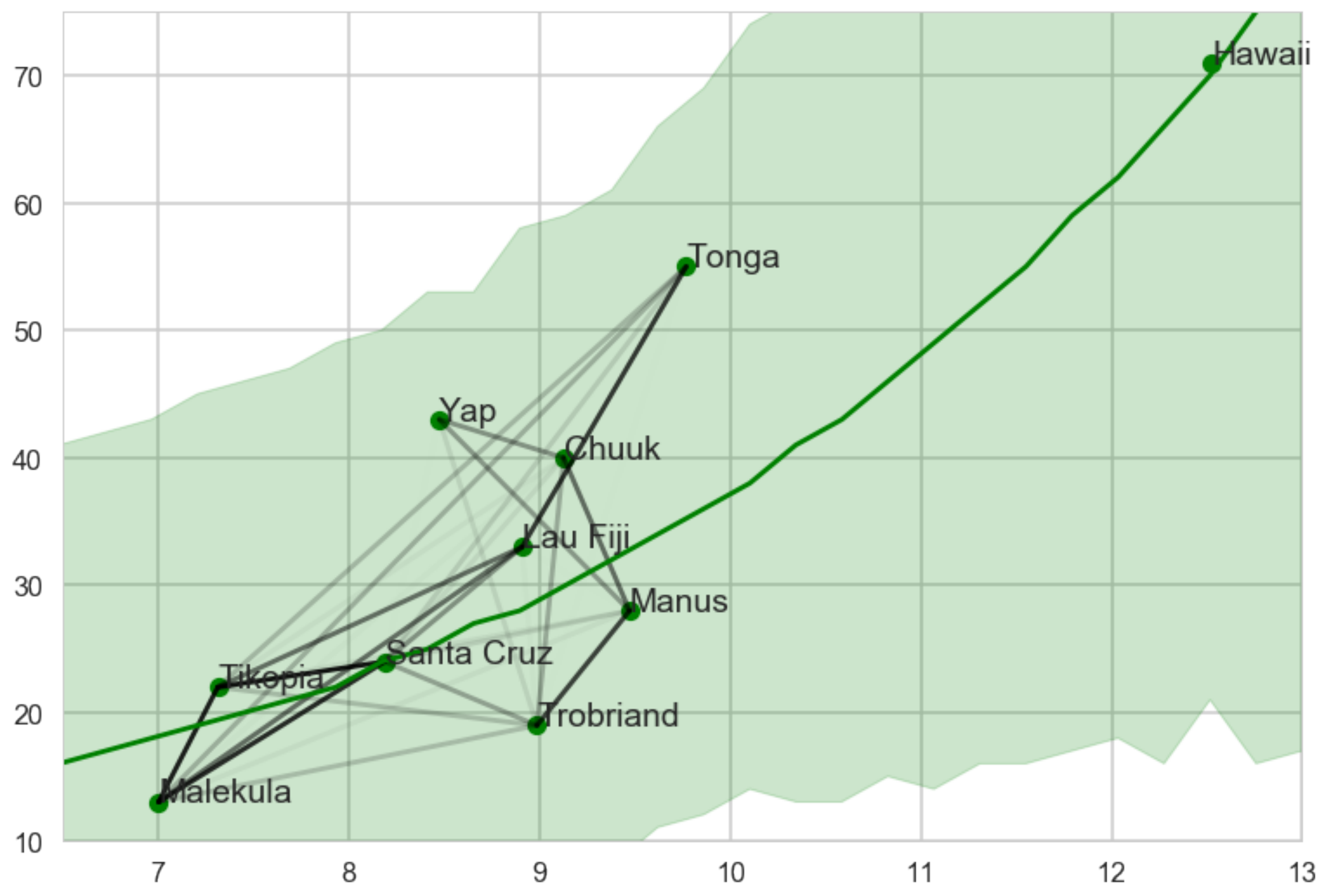
```
                  mean         sd           mc_error     hpd_2.5      hpd_97.5     n_eff    Rhat

       betap      0.247414     0.116188     0.001554      0.005191    0.474834     5471.0   1.000157
       alpha      3.512649     0.357406     0.007140      2.790415    4.239959     2111.0   1.001964
gammasoc__0      -0.269862     0.455632     0.008620     -1.239333    0.607492     2569.0   1.001484
gammasoc__1      -0.117755     0.445192     0.008520     -1.041095    0.757448     2439.0   1.001758
gammasoc__2      -0.165474     0.430544     0.008116     -1.042846    0.686170     2406.0   1.001881
gammasoc__3       0.299581     0.387140     0.007481     -0.481745    1.079855     2365.0   1.001936
gammasoc__4       0.026350     0.382587     0.007425     -0.763338    0.770842     2292.0   1.001728
gammasoc__5      -0.458827     0.389807     0.006976     -1.286453    0.231992     2481.0   1.001517
gammasoc__6       0.097538     0.377499     0.007064     -0.653992    0.840048     2382.0   1.001464
gammasoc__7      -0.263660     0.378417     0.006917     -1.077743    0.404521     2407.0   1.001890
gammasoc__8       0.233544     0.361616     0.006715     -0.510818    0.909164     2407.0   1.001721
gammasoc__9      -0.123068     0.473731     0.006671     -1.034810    0.850175     3985.0   1.000439
       etasq      0.354953     0.660893     0.009717      0.001437    1.114851     4904.0   1.000206
       rhosq      2.306880     30.113269    0.343453      0.000437    4.550517     8112.0   0.999955
```

Covariance posteriors:





Median Inter-Society Correlation



Lets INVERT the idea and start with the COVARIANCE

MVN Primer

$$p(x \mid \mu, \Sigma) = (2\pi)^{-n/2} |\Sigma|^{-1/2} \exp\left\{ -\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu) \right\}$$

JOINT: $p(x, y) = \mathcal{N}\left(\begin{bmatrix} \mu_x \\ \mu_y \end{bmatrix}, \begin{bmatrix} \Sigma_x & \Sigma_{xy} \\ \Sigma_{xy}^T & \Sigma_y \end{bmatrix} \right)$

MARGINAL: $p(x) = \int p(x, y) dy = \mathcal{N}(\mu_x, \Sigma_x)$

CONDITIONAL: $p(x \mid y) = \mathcal{N}(\mu_x + \Sigma_{xy} \Sigma_y^{-1} (y - \mu_y), \Sigma_x - \Sigma_{xy} \Sigma_y^{-1} \Sigma_{xy}^T)$

The marginal for any sub-block is independent of the rest of the matrix in the MVN!

Modeling co-variance via correlation

General expectation of continuity as you move from one adjacent point to another. In the absence of significant noise, two adjacent points ought to have fairly similar f values.

$$k(x_i, x_j) = \sigma_f^2 exp(\frac{-(x_i - x_j)^2}{2l^2})$$

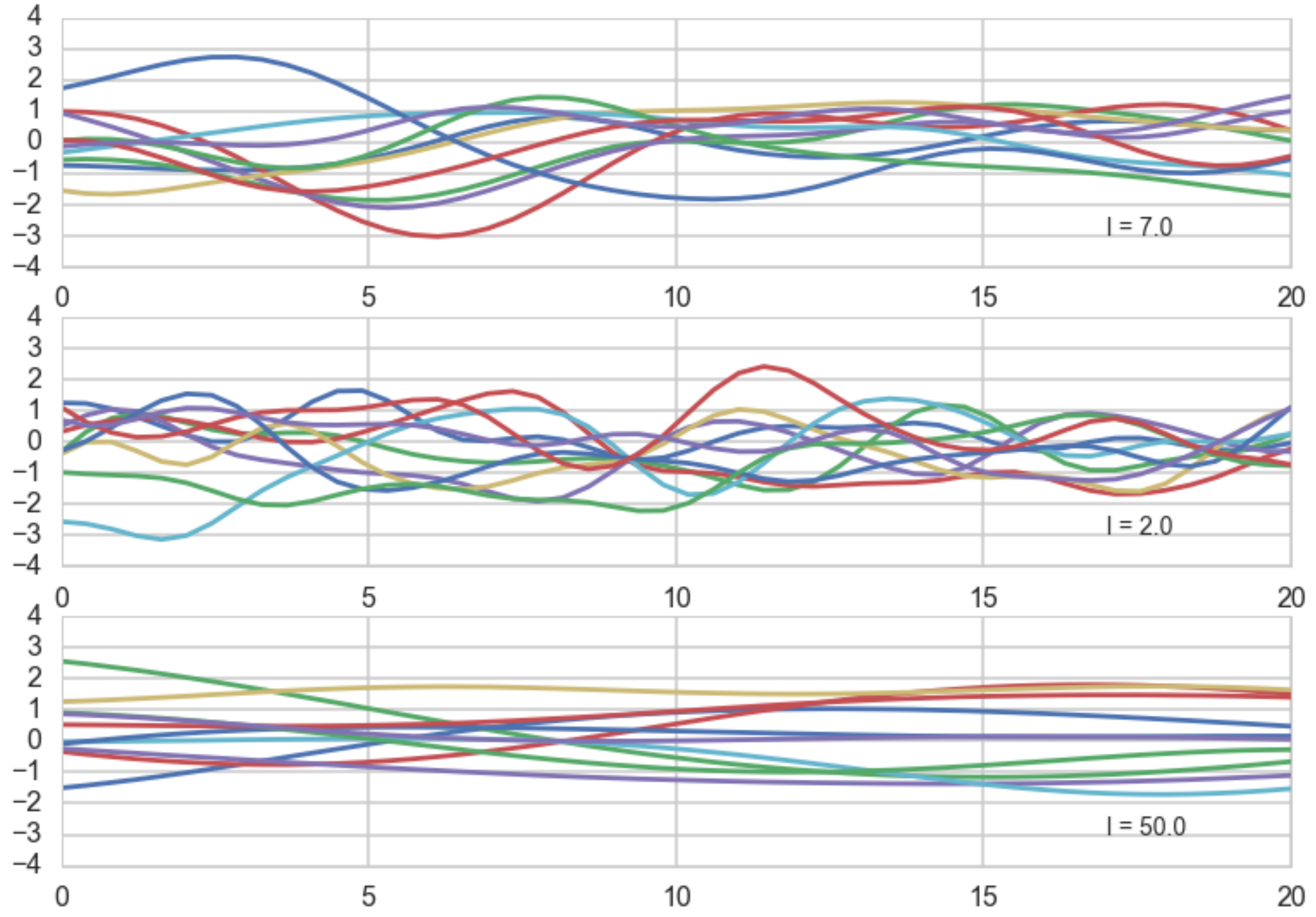
l is correlation length, σ_f^2 amplitude.

```
#Correlation Kernel
def exp_kernel(x1,x2, params):
    amplitude=params[0]
    scale=params[1]
    return amplitude * amplitude*np.exp(-((x1-x2)**2) / (2.0*scale))

#Covariance Matrix
covariance = lambda kernel, x1, x2, params: \
    np.array([[kernel(xi1, xi2, params) for xi1 in x1] for xi2 in x2])
```

Each curve in plots is generated as:

```
a = 1.0
nsamps = 50
xx = np.linspace(0,20,nsamps)

#Create Covariance Matrix
sigma = covariance(exp_kernel,xx,xx, [a,ell]) + np.eye(nsamps)*1e-06

#Draw samples from a 0-mean gaussian with cov=sigma
samples = np.random.multivariate_normal(np.zeros(nsamps), sigma)
```

The greater the correlation length, the smoother the curve.



What did we just do?

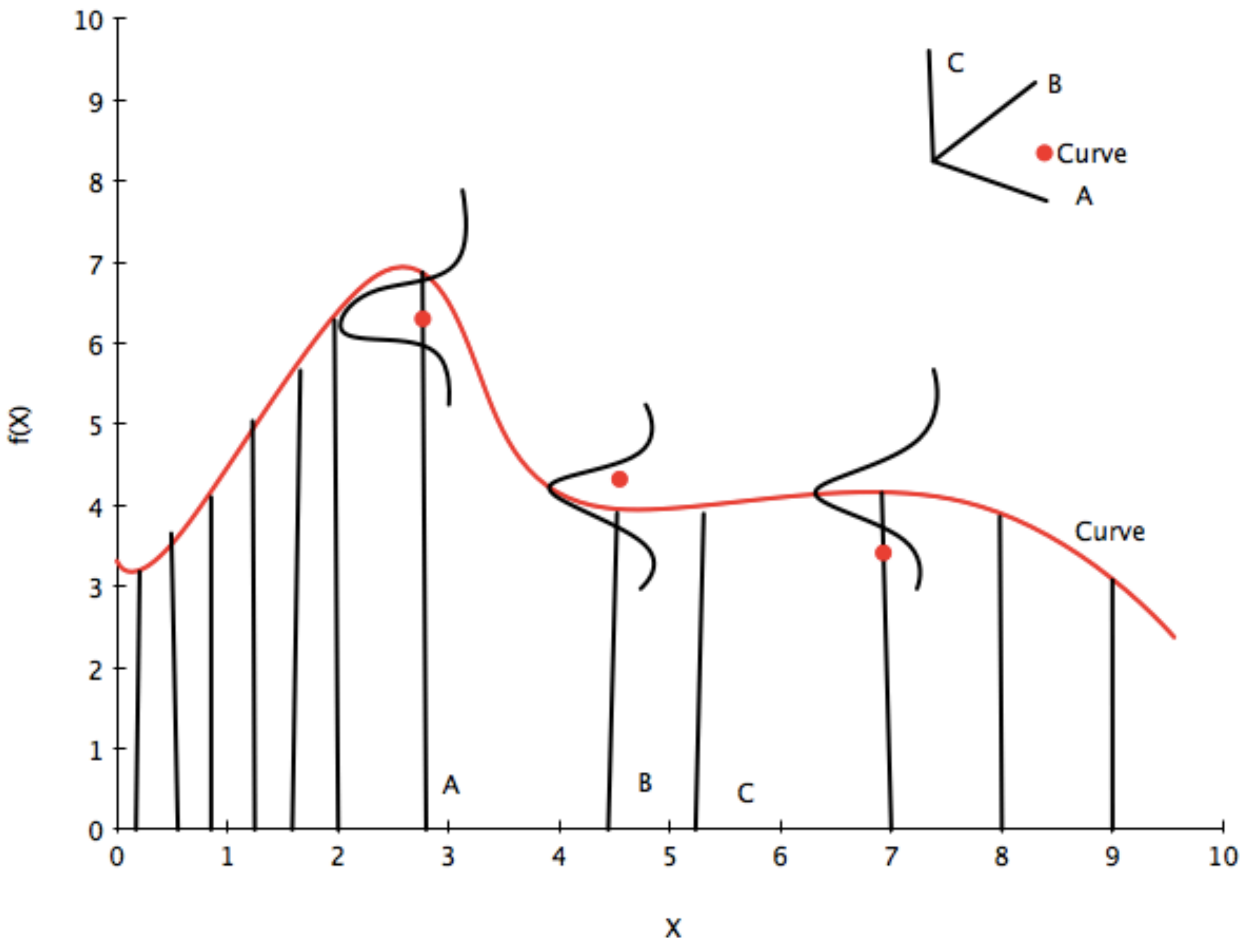
- we have not seen any data yet
- but we expect the function representing our data to have some level of continuity
- thus we considered different **PRIOR** functions that might represent our data
- as having come from MVNs with a covariance matrix based on the correlation length we think we have

Now we see some data

The red curve represents one of these "prior" functions from the calculation above.

We have 3 red data points, so it would be seem to be a prior curves consistent with this data.

Consider the 3 red data points to have been generated IID from some regression function $f(x)$ (like $w \cdot x$) with some univariate gaussian noise σ^2 at each point, that is a gaussian likelihood.



Basic Idea

- The regression curve is one of the prior curves
- it must be consistent with the data.
- Consider the curve as a point in a multi-dimensional space, **a draw from a multivariate gaussian** with as many points as points on the curve.

The regression curve f^* and the data y are assumed to be from a joint multivariate gaussian with the same form (kernel) of covariance matrix

JOINT:

$$p(y, f^*) = \mathcal{N}\left(\begin{bmatrix}\mu_y \\ \mu_{f^*}\end{bmatrix}, \begin{bmatrix}\Sigma_{yy} & \Sigma_{yf^*} \\ \Sigma_{yf^*}^T & \Sigma_{f^*f^*}\end{bmatrix}\right) = \mathcal{N}\left(\begin{bmatrix}\mu \\ \mu_*\end{bmatrix}, \begin{bmatrix}K + \sigma^2 I & K_* \\ K_*^T & K_{**}\end{bmatrix}\right)$$

MARGINAL: $p(f^*) = \int p(f^*, y)dy = \mathcal{N}(\mu_*, K_{**})$

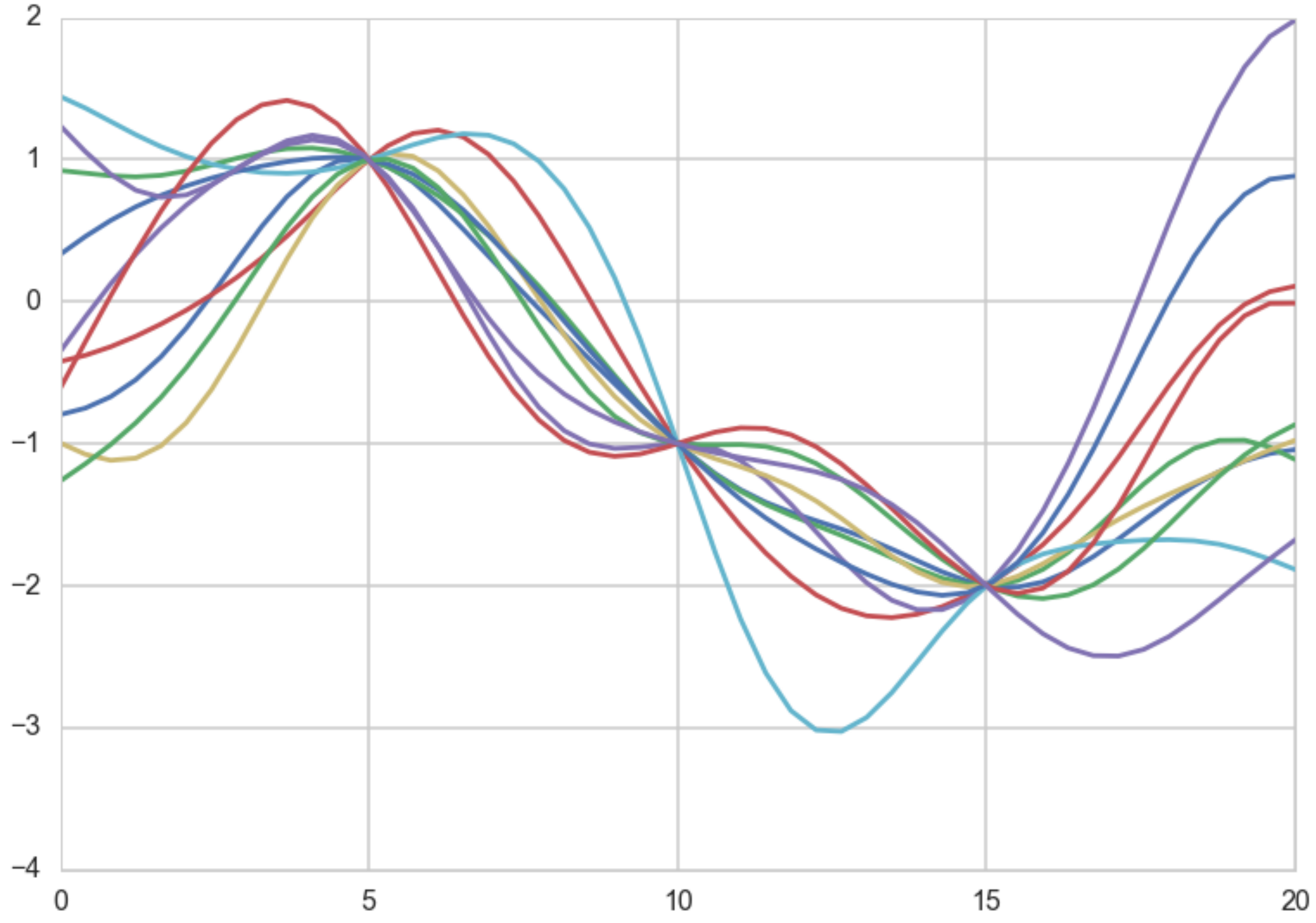
CONDITIONAL:

$$p(f^* \mid y) = \mathcal{N}\left(\mu_* + K_*(K + \sigma^2 I)^{-1}(y - \mu),\; K_{**} - K_*(K + \sigma^2 I)^{-1}K_*^T\right)$$

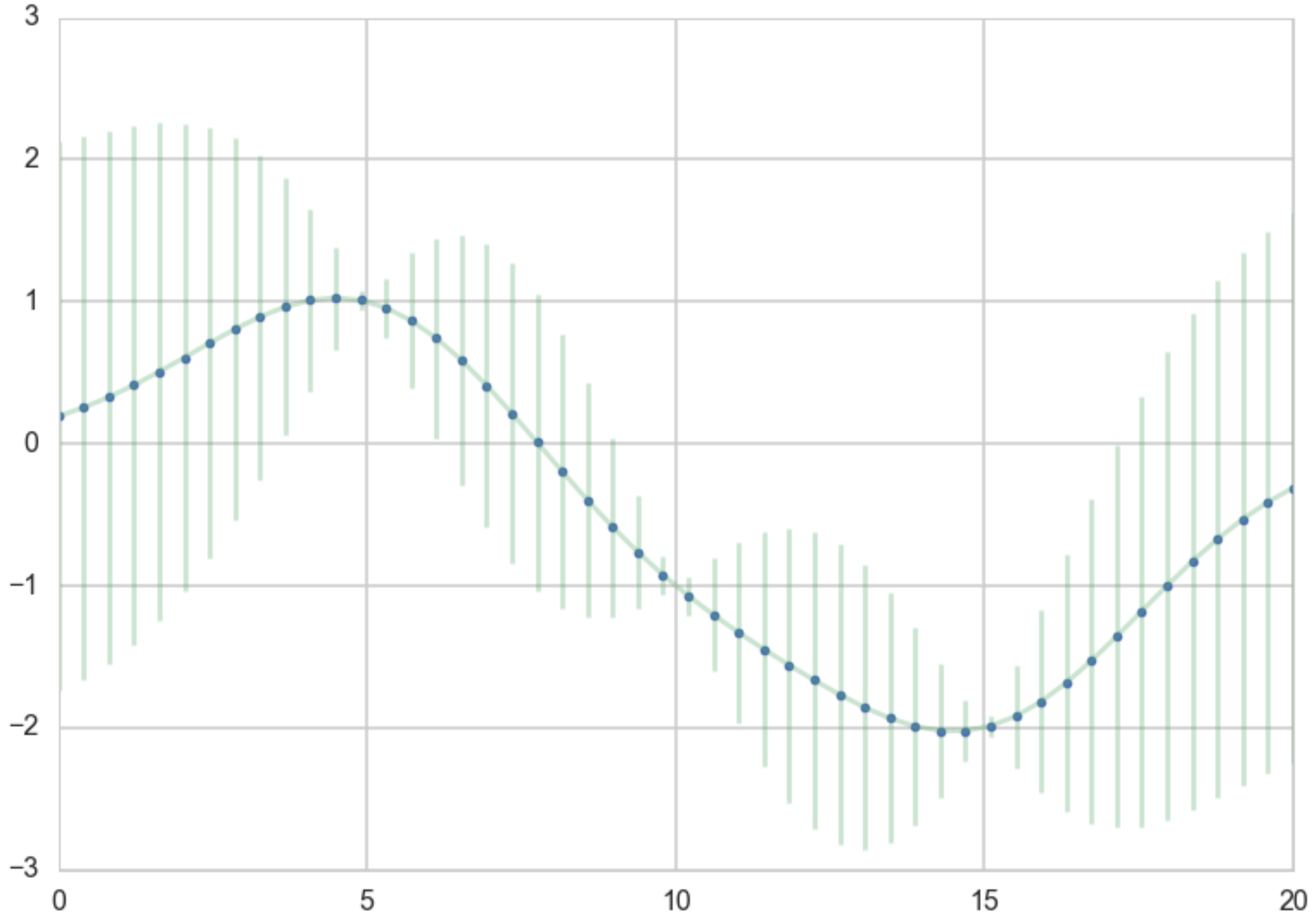
where: $K = K(x, x); K_* = K(x, x^*); K_{**} = K(x^*, x^*)$

$l = 7.$

```
#"test" data
x_star = np.linspace(0,20,nsamps) #50 grid points

# defining the training data
x = np.array([5.0, 10.0, 15.0]) # shape 3
f = np.array([1.0, -1.0, -2.0]).reshape(-1,1)


K = covariance(exp_kernel, x,x,[a,ell])
#shape 3,3

K_star = covariance(exp_kernel,x,x_star,[a,ell])
#shape 50, 3

K_star_star = covariance(exp_kernel, x_star, x_star, [a,ell])
#shape 50,50

K_inv = np.linalg.inv(K)
#shape 3,3

mu_star = np.dot(np.dot(K_star, K_inv),f)
#shape 50

sigma_star = K_star_star  - np.dot(np.dot(K_star, K_inv),K_star.T)
#shape 50, 50

for i in range(10):
    samples = np.random.multivariate_normal(mu_star.flatten(), sigma_star)
    plt.plot(x_star, samples)
```



Predictive 1



- we are left with a smaller infinity of prior functions consistent with the data
- these are our possible regression functions
- here we show the area these regression functions cover

```
plt.plot(x_star, mu_star, '.')
plt.errorbar(x_star, mu_star,
    yerr=1.96*np.sqrt(sigma_star.diagonal()), alpha=0.3);
```

2 main concepts:

conditional equals predictive

marginal block is independent

Conditional

$$p(f^* \mid y) = \mathcal{N}\left(\mu_* + K_*(K + \sigma^2 I)^{-1}(y - \mu),\ K_{**} - K_*(K + \sigma^2 I)^{-1} K_*^T\right)$$

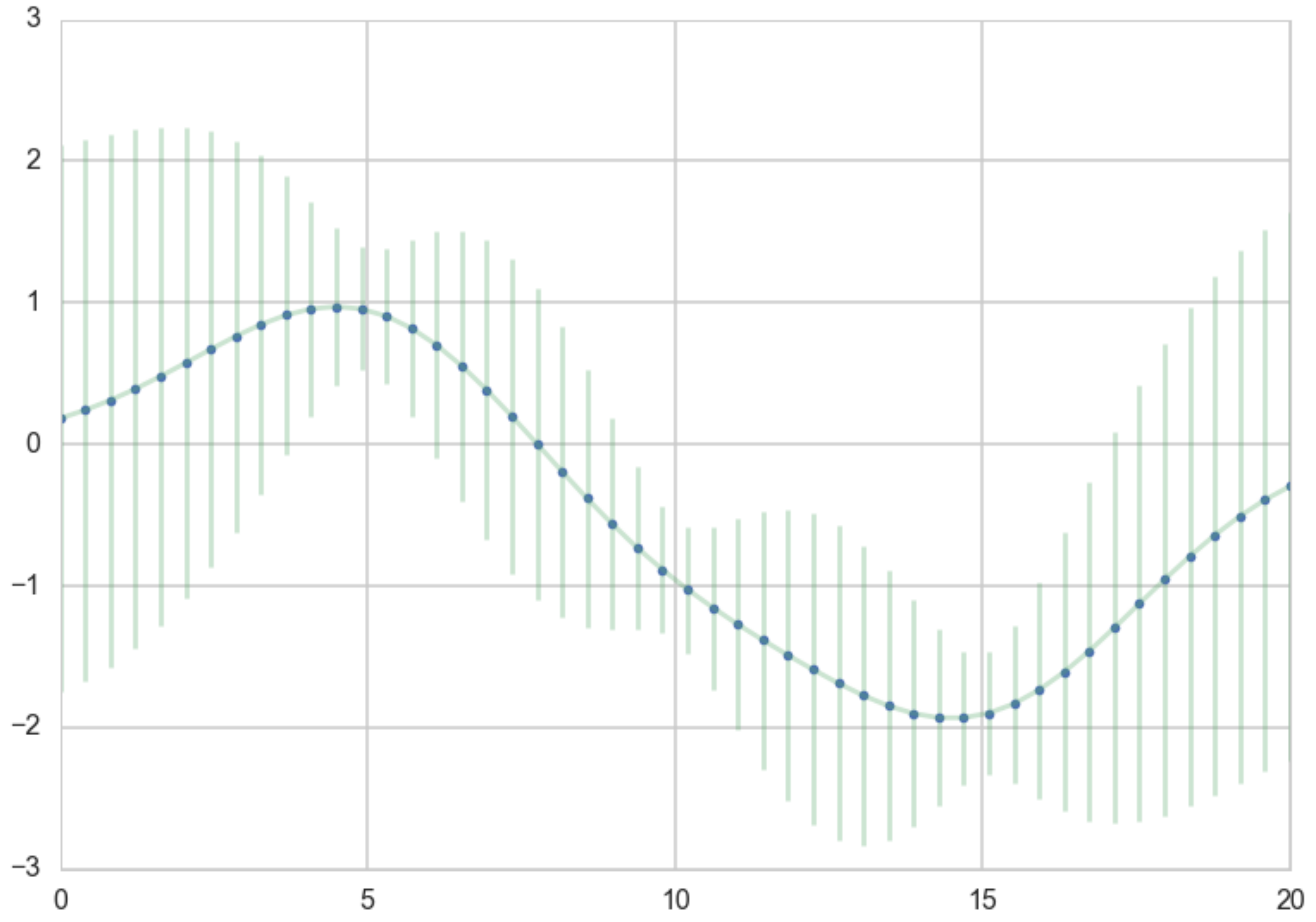
EQUALS Predictive

Predictive 2: add noise for $p(y^* \mid y)$

Now let's add some noise to our scenario.

We assume $y = f(x) + \epsilon$, $\epsilon \sim N(0, \sigma^2)$

```
sigma_epsilon_sq = 0.05

K_noise = K + np.diag([sigma_epsilon_sq] * K.shape[0])
K_noise_inv = np.linalg.inv(K_noise)

mu_star_noise = np.dot(np.dot(K_star, K_noise_inv),f)

sigma_star_noise = K_star_star  -
    np.dot(np.dot(K_star, K_noise_inv),K_star.T)

plt.plot(x_star, mu_star_noise, '.')
plt.errorbar(x_star, mu_star_noise,
    yerr=1.96*np.sqrt(sigma_star_noise.diagonal()), alpha=0.3);
```



So far

1. We built a covariance matrix from a kernel function
2. Use the covariance matrix to generate a "curve" as a point in a multi-dimensional space from a MVN
3. multiple such curves serve as prior fits for our data
4. now we bring in the data and condition on it (with noise added if needed) using normal distribution formulae
5. the conditional has the form of a predictive and we are done
6. Also notice that the marginal only has quantities from the predictive block. This means that we dont care about the size of the original block in calculating the marginal.

These observations are the building blocks of the GP.

KEY CONCEPT 2:

MARGINAL IS DECOUPLED

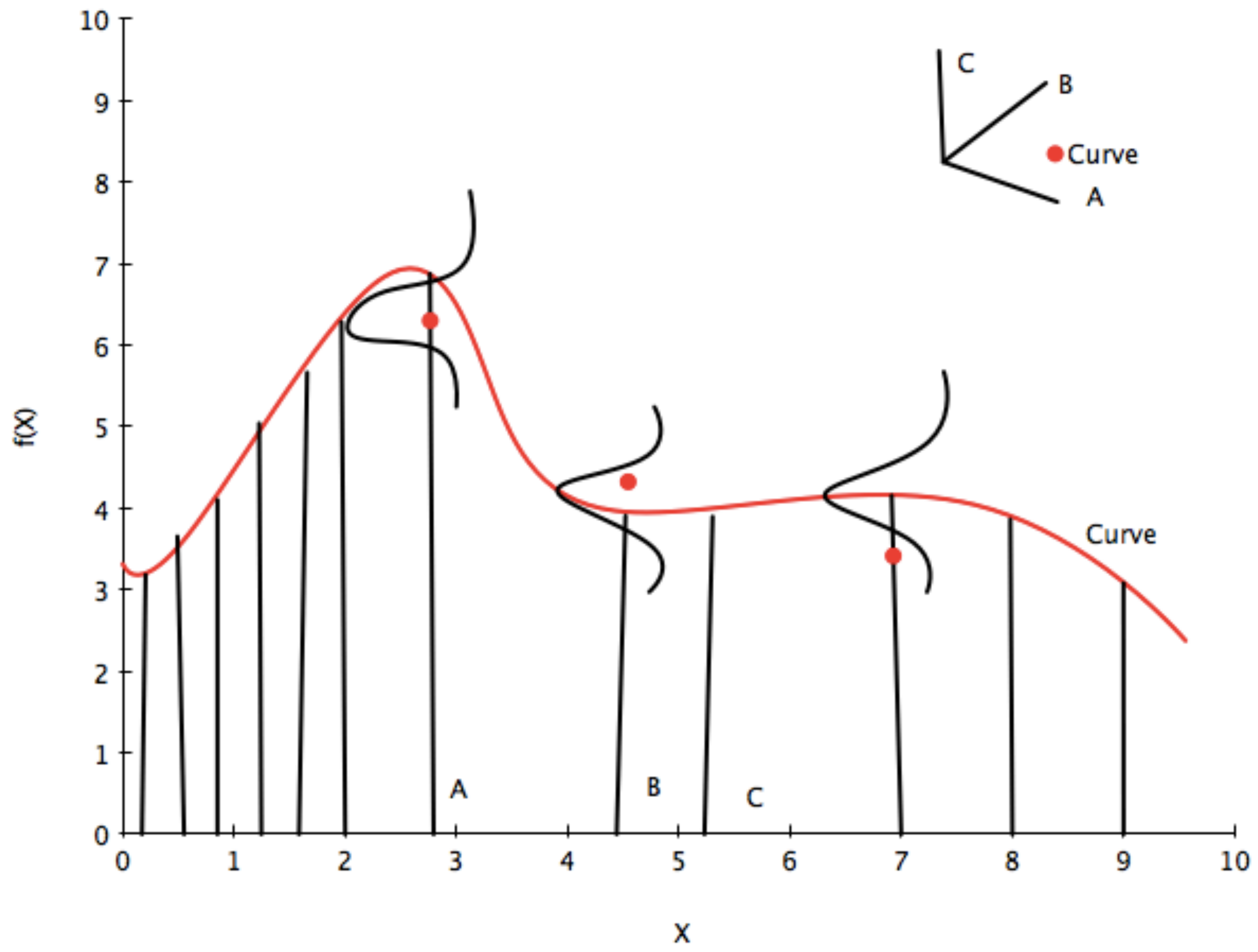
...for the marginal of a gaussian, only the covariance of the block of the matrix involving the unmarginalized dimensions matters! Thus "if you ask only for the properties of the function (you are fitting to the data) at a finite number of points, then inference in the Gaussian process will give you the same answer if you ignore the infinitely many other points, as if you would have taken them all into account!"

-Rasmunnsen

Use infinite gaussians!

- think of the function as an infinite vector.
- Draw $\bar{f}$ from some 'infinite' gaussian distribution with some mean and some kernel.

This then is the Gaussian Process, which we use to set a prior on the space of functions.



Back to our formulae...

JOINT:

$$p(f, f^{\infty}) = \mathcal{N}\left(\begin{bmatrix} \mu_f \\ \mu_{f^{\infty}} \end{bmatrix}, \begin{bmatrix} \Sigma_{ff} & \Sigma_{ff^{\infty}} \\ \Sigma_{ff^{\infty}}^T & \Sigma_{f^{\infty} f^{\infty}} \end{bmatrix}\right) = \mathcal{N}\left(\begin{bmatrix} \mu \\ \mu_{\infty} \end{bmatrix}, \begin{bmatrix} K & K_{\infty} \\ K_{\infty}^T & K_{\infty\infty} \end{bmatrix}\right)$$

MARGINAL:

$$p(f) = \int p(f, f^{\infty}) df^{\infty} = \mathcal{N}(\mu_f, K)$$

where: $K = K(x, x); K_{\infty} = K(x, x^{\infty}); K_{\infty\infty} = K(x^{\infty}, x^{\infty})$

Definition of Gaussian Process

Assume we have this function vector
$f = (f(x_1), \ldots f(x_n))$. If, for ANY choice of input points, $(x_1, \ldots, x_n)$, the marginal distribution over f:

$$P(F) = \int_{f \notin F} P(f) df$$

is multi-variate Gaussian, then the distribution $P(f)$ over the function f is said to be a Gaussian Process.

We write a Gaussian Process thus:

$$f(x) \sim \mathcal{GP}(m(x), k(x, x\prime))$$

where the mean and covariance functions can be thought of as the infinite dimensional mean vector and covariance matrix respectively.

a Gaussian Process defines a prior distribution over functions!

Once we have seen some data, this prior can be converted to a posterior over functions, thus restricting the set of functions that we can use based on the data.

So: consider every possible function and associate a prior probability with this function. e.g. assign smoother functions higher prior probability. But how are we possibly going to calculate over an uncountably infinite set of functions in finite time?

Size of the other block DOES NOT MATTER

Since the size of the "other" block of the matrix does not matter, we can do inference from a finite set of points.

Any m observations in an arbitrary data set, $y = y_1, \ldots, y_n = m$ can always be represented as a single point sampled from some m-variate Gaussian distribution. Thus, we can work backwards to 'partner' a GP with a data set, by marginalizing over the infinitely-many variables that we are not interested in, or have not observed.

GP regression

Using a Gaussian process as a prior for our model, and a Gaussian as our data likelihood, then we can construct a Gaussian process posterior.

Likelihood: $y|f(x), x \sim \mathcal{N}(f(x), \sigma^2 I)$ where the infinite $f(x)$ takes the place of the parameters.

Prior: $f(x) \sim \mathcal{GP}(m(x) = 0, k(x, x\prime))$

Infinite normal posterior process: $f(x)|y \sim \mathcal{GP}(m_{post}, \kappa_{post}(x, x\prime))$.
(**Kinda like a conjugate prior!**)

GP regression posterior

The posterior distribution for f is:

$$\begin{aligned} m_{post} &= k(x\prime, x)[k(x,x) + \sigma^2 I]^{-1} y \\ k_{post}(x, x\prime) &= k(x\prime, x\prime) - k(x\prime, x)[k(x,x) + \sigma^2 I]^{-1} k(x, x\prime) \end{aligned}$$

Posterior predictive distribution for $f(x_*)$ for a test vector input x_*, given a training set X with values y for the GP is:

$$\begin{aligned} m_* &= k(x_*, X)[k(X^T, X) + \sigma^2 I]^{-1} y \\ k_* &= k(x_*, x_*) - k(x_*, X^T)[k(X^T, X) + \sigma^2 I]^{-1} k(X^T, x_*) \end{aligned}$$

The predictive distribution of test targets y$_*$: add $\sigma^2 I$ to k_*.

What did we do

- usually in a parametric model we had some m (small) number of parameters
- but here our covariance functions are NxN !
- no free lunch: calculation involves inverting a NxN matrix as in the kernel space representation of regression.
- cannot thus handle large data if no approximations are used

How do we represent priors?

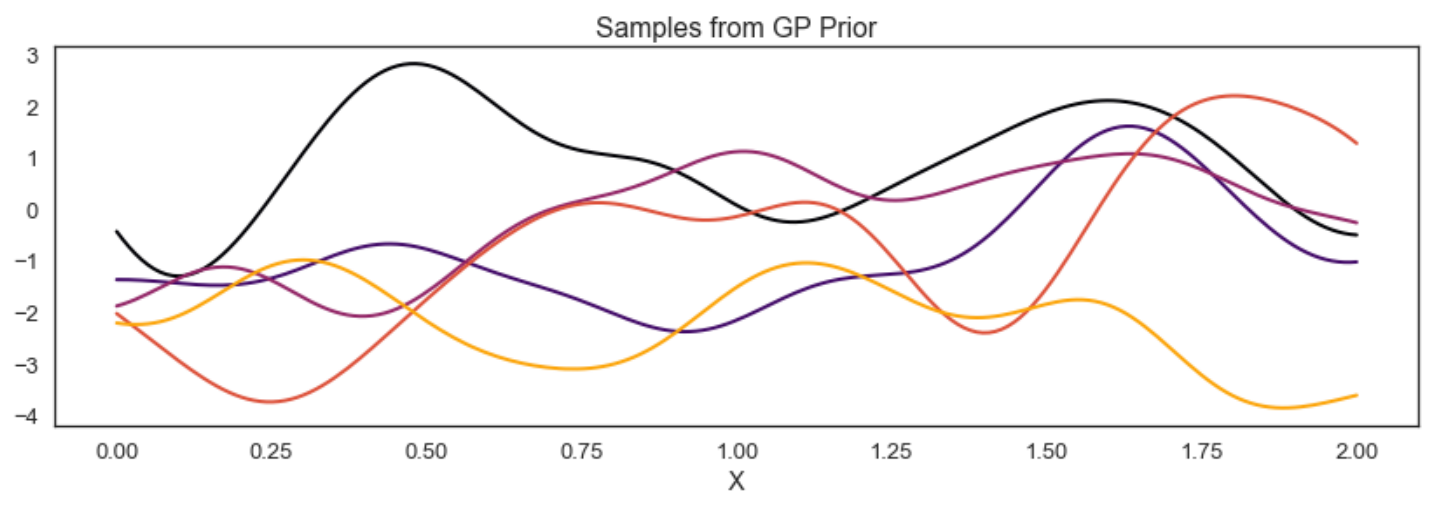
KERNELS for covariance matrices!

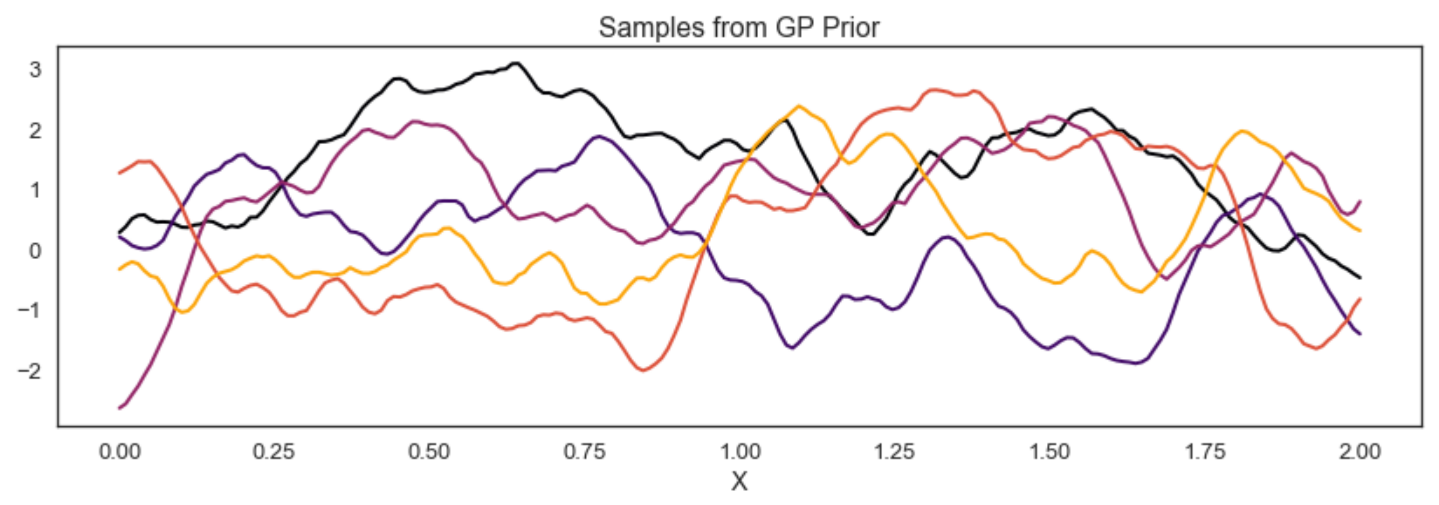
Can compose kernels.

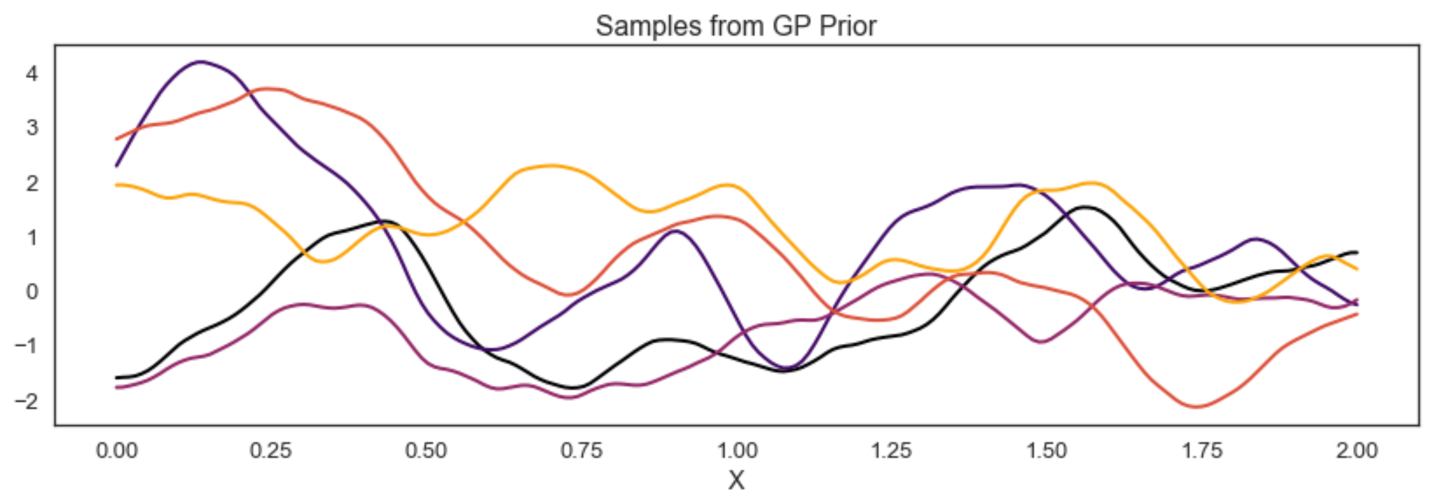
Examples: exponential, cosine, matern, etc

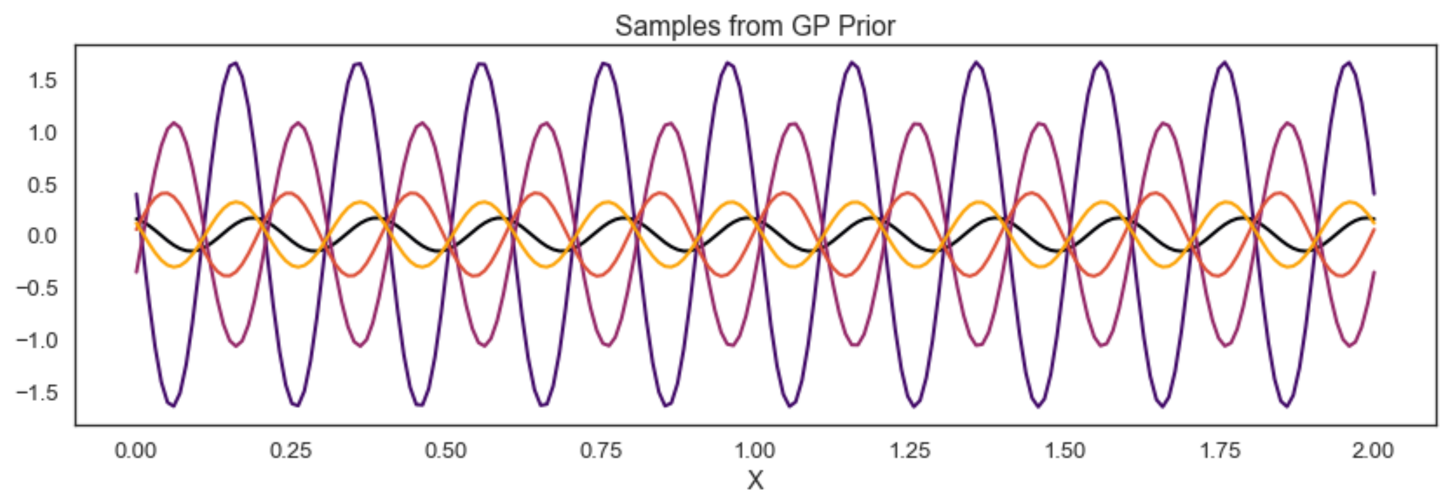
Then the infinite functions are again parametrized by finite number of kernel parameters.

Covariance Kernels (**row1**: exp, m32 **row2**: m52, cos)









Setting up the model

```python
with pm.Model() as model:
    # priors on the covariance function hyperparameters
    l = pm.Uniform('l', 0, 10)
    # uninformative prior on the function variance
    s2_f = pm.HalfCauchy('s2_f', beta=10)
    # uninformative prior on the noise variance
    s2_n = pm.HalfCauchy('s2_n', beta=5)
    # covariance functions for the function f and the noise
    f_cov = s2_f**2 * pm.gp.cov.ExpQuad(1, l)
    mgp = pm.gp.Marginal(cov_func=f_cov)
    y_obs = mgp.marginal_likelihood('y_obs',
        X=xtrain.reshape(-1,1), y=ytrain, noise=s2_n,
        is_observed=True)
```

How do we sample? Use the Marginal!

Posterior-predictive distribution, as a function of hyper parameters η:

$$p(y^* | D, \eta) = \int d\theta \, p(y^* | \theta) \, p(\theta | D, \eta)$$

A likelihood with parameters η and simply use maximum-likelihood with respect to η to estimate these η using our "data" y^*

INFERENCE

Want the marginal likelihood: $p(y|X) = \int_f p(y|f, X)p(f|X)df$

The Marginal likelihood given a GP prior and a gaussian likelihood is just the marginal for a finite MVN!

$$\log p(y|X) = -\frac{n}{2}\log 2\pi - \frac{1}{2}\log|K + \sigma^2 I| - \frac{1}{2}y^T(K + \sigma^2 I)^{-1}y$$

MAP-2 Fitting in pymc3

```python
with model:
    marginal_post = pm.find_MAP()
```

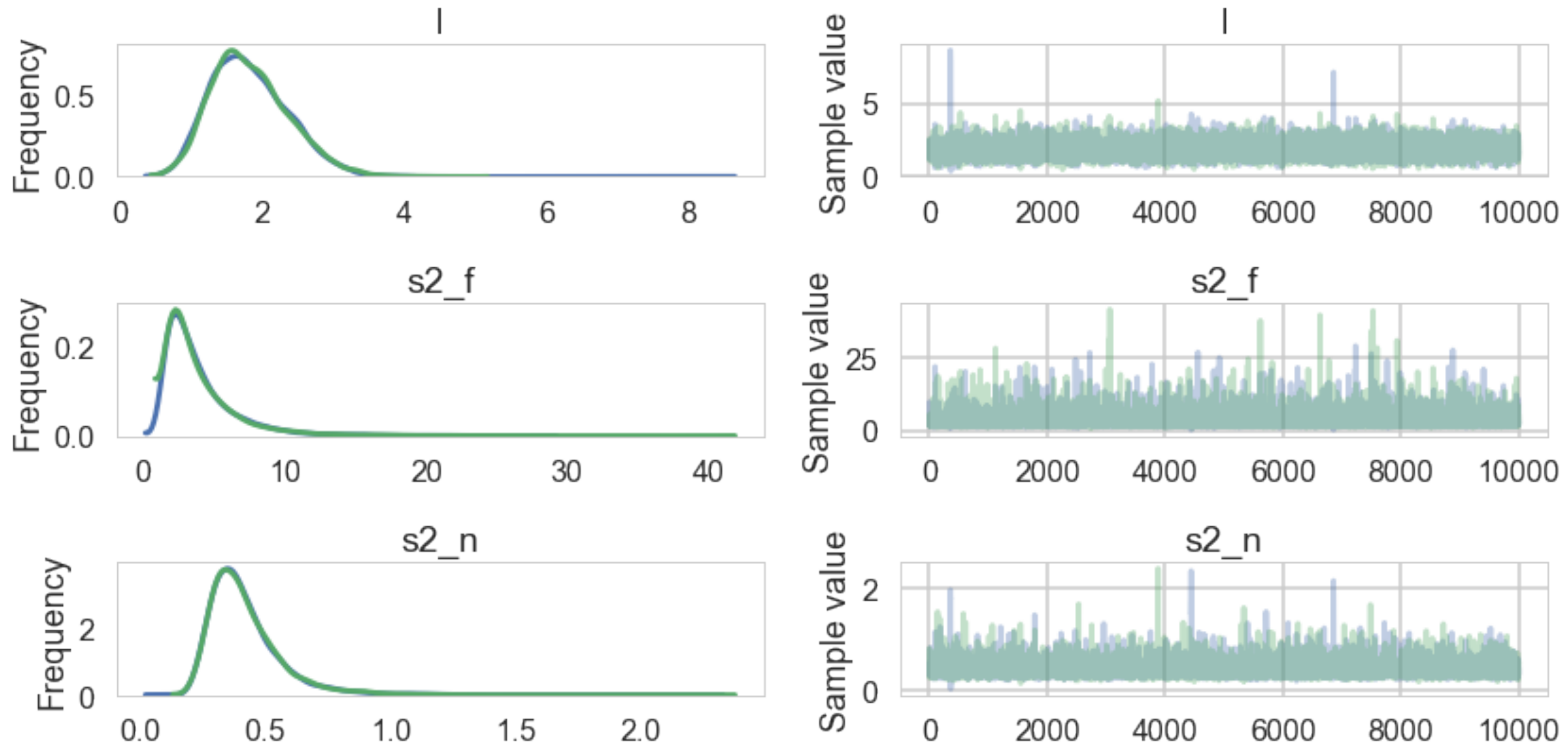
```
{'l': array(1.438132008790354),
 'l_interval__': array(-1.7839733342616466),
 's2_f': array(2.0475004392008898),
 's2_n': array(0.3465300514941838)}
```

How do we predict? Use the conditional!

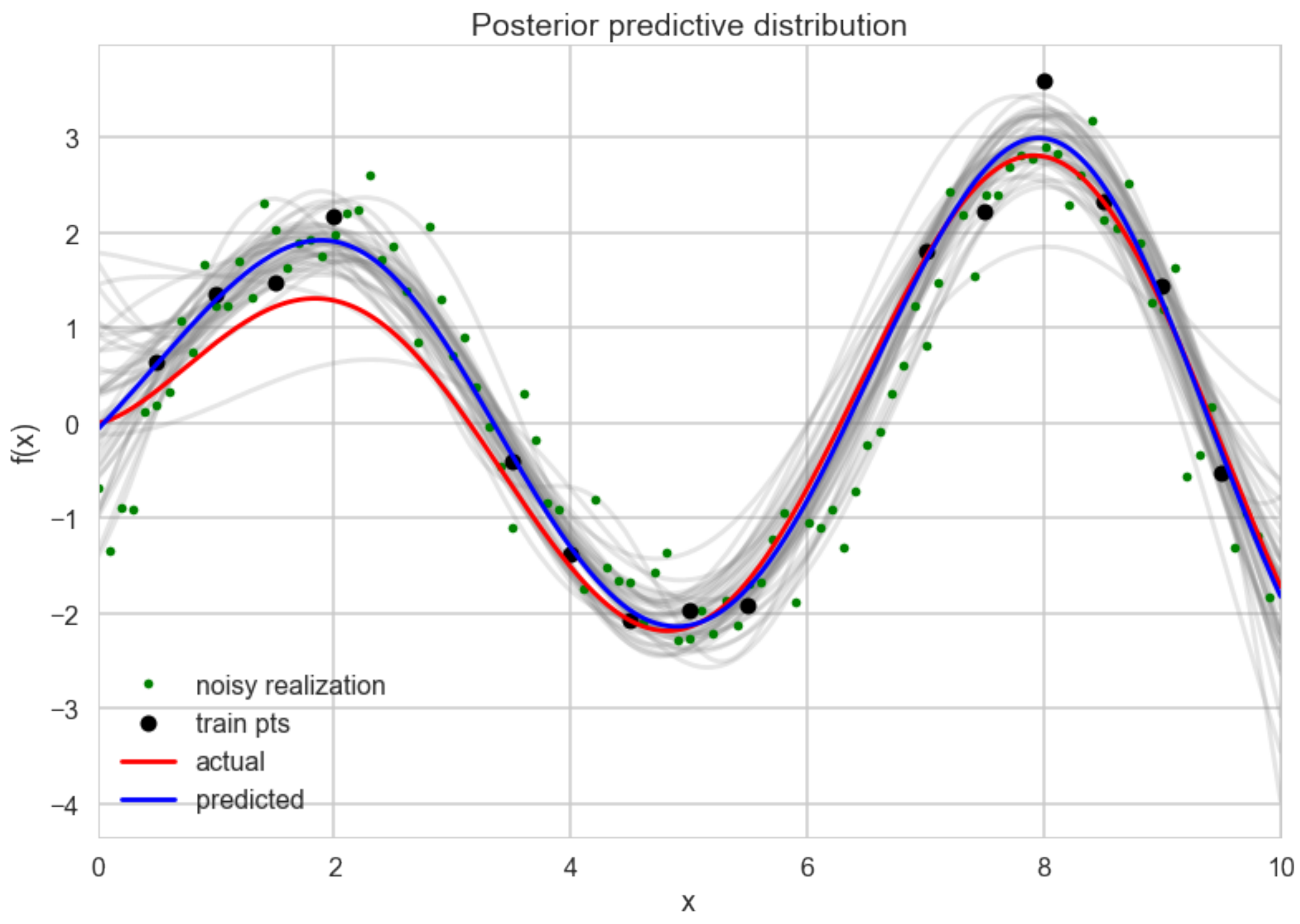
- we used the marginal (which was the marginal likelihood) as the likelihood in our model
- ultimately we want to predict
- once we have traces for the kernel parameters, we need to use each tracepoint
- to draw a sample from the conditional
- thats it!

MCMC

```
with model:
    trace = pm.sample(10000, tune=2000,
        nuts_kwargs={'target_accept':0.85})
 with model:
    fpred = mgp.conditional("fpred",
        Xnew = x_pred.reshape(-1,1),
        pred_noise=False)
    ypred = mgp.conditional("ypred",
        Xnew = x_pred.reshape(-1,1),
        pred_noise=True)
    gp_samples = pm.sample_ppc(trace,
         vars=[fpred, ypred],
         samples=200)
```



Posterior (predictive) curves



Where are GPs used?

- geostatistics with kriging, oil exploration
- spatial statistics
- as an interpolator (0 noise case) in weather simulations
- they are equivalent to many machine learning models such as kernelized regression, SVM and neural networks (some)
- ecology since model uncertainty is high
- they are the start of non-parametric regression
- time series analysis (see cover of BDA)
- because of the composability of kernels, in automates statistical analysis (see the automatic statistician)