FinalProject_Combined

May 12, 2020

1 Final Project

Please fill out the relevant cells below according to the instructions. When done, save the notebook and export it to PDF, upload both the `ipynb` *and the PDF file to Canvas.*

1.1 Group Members

Group submission is highly encouraged. If you submit as part of group, list all group members here. Groups can comprise up to 5 students.

- Aizhan Akhmetzhanova
- Tony Chen
- Lucas Makinen
- Emre Oezkan
- Sachin Smart

1.2 Sparse Interactions

1.2.1 Preparation (3pts)

Review the paper The Kernel Interaction Trick: Fast Bayesian Discovery of Pairwise Interactions in High Dimensions by Agrawal et al. (2019). Start with the general concepts and then go into the finer details.

When you feel comfortable with the content, answer the following questions:

1. Why does the Gaussian scale mixture prior promote sparsity of the regression coefficients θ?
2. What are the required properties of the model in Eq. (3) that allow it to be rewritten in the form of Eq. (6)?
3. What are the conceptual and practical limitation of the approach?

Hint: Some of the answers may require parsing the relevant references.

1.3 Preparation Response

1.3.1 Question 1:

Based on Griffin and Brown (2017), hierarchical priors in Bayesian regression are dependent on the size of the coefficients, where the hyperparameters control for shrinkage of coefficients similar to a penalty term. In particular, a small estimated variance in coefficients would enforce some shrinkage and likely yield a sparser model. The Gaussian scale mixture is just a special case of hierarchical priors, in which τ and Σ_τ control the shrinkage of θ. The SKIM model, in particular, has the form

$$\kappa \sim p(\kappa), \quad \eta \sim p(\eta), \quad c^2 \sim p(c^2)$$
$$\theta_{x_i}|\kappa, \nu \sim \mathcal{N}(0, \eta_1^2 \kappa_i^2) \quad \text{independent for} \quad i \in \{1, ..., p\},$$
$$\theta_{x_i x_j}|\kappa, \eta \sim \mathcal{N}(0, \eta_2^2 \kappa_i^2 k_j^2) \quad \text{independent for} \quad i, j \in \{1, ..., p\}, i \neq j,$$
$$\theta_{x_i^2}|\kappa, \eta \sim \mathcal{N}(0, \eta_3^2 \kappa_i^4) \quad \text{independent for} \quad i \in \{1, ..., p\},$$
$$\theta_0|c^2 \sim \mathcal{N}(0, c^2).$$

The pdfs $p(\kappa), p(\nu)$ and $p(c^2)$ are usually chosen to be heavy-tailed.

First, the model above promotes sparsity on the parameter θ_{x_i} for $i \in \{1, ..., p\}$. The parameter η_1 controls the overall sparsity level of the θ_{x_i}. If η_1 is small, then, $\eta_1^2 \kappa_i^2$ tends to be small for each $i \in \{1, ..., p\}$, leading to a very high probability mass of the normal density of θ_{x_i} around zero. Therefore, realizations of θ_{x_i} where $|\theta_{x_i}|$ is small tend to be more likely. The parameter κ_i, which is usually drawn from a heavy-tail distribution, controls for the local sparsity and can then overcome the global shrinkage induced on θ_{x_i} by η_1 if the realization of κ_i is large enough. If the realization of κ_i is large enough, the variance $\eta_1^2 \kappa_i^2$ of θ_{x_i} can become non-negligible, leading to a higher likelihood of non-zero values for $|\theta_{x_i}|$.

Secondly, the model promotes sparisty by its hirarchy property. In this context, hirarchy means that an interaction terms $\theta_{x_i x_j}$ can only be relevant if both θ_{x_i} and θ_{x_j} are relevant. Thus, if at least one of the parameter θ_{x_i} or θ_{x_j} are shrunk to zero, the interaction term $\theta_{x_i x_j}$ is shrunk to zero too. This can be seen by observing that the variance of the interaction term $\theta_{x_i x_j}$ is given by $\eta_2^2 \kappa_i^2 \kappa_j^2$. If at least one of the local parameter κ_i or κ_j is small, then, the variance of $\eta_2^2 \kappa_i^2 \kappa_j^2$ will be small, leading to a high likelihood concentration of $\theta_{x_i x_j}$ around small values of $|\theta_{x_i x_j}|$. Here again, the parameter η_2 controls for the global sparsity of the interaction terms $\theta_{x_i x_j}$. We see the sparsity in θ_{x_i} are "inherited" via hirarchy to sparsity in $\theta_{x_i x_j}$.

The above applies similarly for the quadratic terms $\theta_{x_i}^2$. If the local shrinkage parameter κ_i tends to be small, then, the variance of $\theta_{x_i}^2$, which is given by $\eta_3^2 \kappa_i^4$ will be small. Again, the parameter η_3 controls for the global sparsity of the interaction terms.

1.3.2 Question 2:

This result is possible due to the Proposition 4.1 in Agrawal et al (2019), and the weight-space view of GP from Rasmussen and Williams (2006). The function Φ_2 is designed such that it can be rewritten as a GP, namely the GP $g = \theta^T \Phi_2$. Then, for any draw $g|\tau \sim N(0, k_\tau)$, there exists a parameter vector θ such that $g = \theta^T \Phi_2$. The model in Eq (3) is equivalent to the model in Eq (6) for the right choice of the kernel function k_τ for the Gaussian Process $g|\tau \sim \text{GP}(0, k_\tau)$. The choice of the kernel function k_τ is

$$k_\tau(x, x') = \Phi_2(x)^t \Sigma_\tau \Phi_2(x').$$

For such a choice, the model defined in Eq (3) is always equivalent to the model defined in Eq (6). However, for the proposed algorithm to work in the proposed time efficiency, we need to be able to evaluate k_τ in $O(p)$ time, where p is the number of covariates. To do this, the papaer rewrites the kernel k_τ as a weighted sum of polynomial kernels. To do this, one one needs to require the variance-covariance matrix Σ_τ of the parameter θ to be diagonal.

1.3.3 Question 3:

We can identify the following limitations to this paper:

- The implementation of the SKIM model is dependent on the choice of several hyperparameters that need to be selected by the user. Some of these hyperparameter are non-trivial to choose, but at the same time very important. For example, the SKIM model requires the user to select a hyperparameter s which corresponds to the global sparsity of θ_{x_i}. Furthermore, the κ_i's are chosen from a truncated half-Cauchy distribution, in which one needs to specify a hyperparameter m that corresponds to the truncation cutoff of the half-Cauchy distribution. Therefore, m control indirectly for the heavy-tailness of $p(\kappa)$ and therefore, for the local sparsity on θ_{x_i}. Similar hyperparameter have to be chosen for the interaction terms $\theta_{x_i x_j}$ and the squared terms $\theta^2_{x_i}$. Overall, to sum it up, there are several hyperparameters to select and the choice is not trivial.

- The implementation of the SKIM model in the paper requires the variance-covariance matrix Σ_τ of the prior of the parameter θ to be diagonal. Therefore, it is impossible to incorporate correlations between different components of θ in the prior.

- The SKIM method implies hirarchy. This is, an interaction parameter $\theta_{x_i x_j}$ is always negligable if at least one of the parameter θ_{x_i} and θ_{x_j} is negligable. This is a desirable property in many applications. However, in certain applications, this property might not be desirable. Consider a situation in which both variables x_i and x_j do not directly influence the target variable y, however, their interaction term $x_i x_j$ has strong predictive power. This situation cannot be reflected by the SKIM method.

  - For a specific example, in the discussion of Priors for Related Predictors in Chipman (1996), one practical instance in which the strong hierarchy property would fail is in atmospheric sciences. A key relation is $log(Y) = log(A) + BC$, in which the interaction term BC would not satisfy strong hierarchy, and thus could not be properly modeled using this approach.

- The SKIM method, as implemented in the paper, takes $O(pN^2 + N^3)$ time to compute, where N denotes the number of observations and p the number of covariates. It has the nice property of scaling linearly in p, which makes the method applicable to situations in which p is large. However, the method scales cubic in N. This limits the applicability of the method to situations in which N is large. By comparison, the naive method requires $O(p^6 + p^4 N)$ time, which scales linearly in N (but obviously, terrible in p). Furthermore, memory is quadratic with respect to N, which is only as good or even worse than the other sampling methods compared (NAIVE, WOODBURY, and FULL).

- The method is based on sampling $\tau_1, ..., \tau_T \sim p(\tau|D)$ iid. To do so, MCMC methods are used. Therefore, the drawback that are specific to MCMC methods apply to this method too. Particularly, these are deciding when convergence of the Monte Carlo chain underlying the MCMC process is reached, how to sparse out the MCMC chain to obtain approximately independent draws and deciding on hyperparameters and initial choices.

- Another possible limitation is the reliance/assumption of pairwise interactions. On one hand, the dimension of the parameter space is increased from p to $\frac{p(p+1)}{2}$, which could be extremely computationally expensive when p is not even that large; on the other hand, there may be instances that involve higher-order polynomial interactions, and thus this model of at-most pairwise interactions could be too limited in its scope.

1.3.4 Code adaptation (3pts)

The method SKIM from chapter 6 has been implemented in jax/Numpyro here. Review the code and recognize how the theoretical concepts of the Kernel Interaction Trick and the specific features of SKIM have been implemented. Then copy the code to this notebook and modify it so that you can execute the provided test example inline. Confirm that you get a result comparable to theirs.

The last step of their example analysis (sampling from the posterior with the method `sample_theta_space`) often returns `nans`. It also reports the posterior for all θ (active and inactive ones), and only for one sample at a time. That's really clunky. Modify this function to produce flat posteriors samples from the MCMC (with an arbitrary length of samples) but only for the active direct and pairwise interaction terms. Visualize the posterior from the example with `corner`.

1.4 Code Adaptation Response

1.4.1 1. Transfer the code and check for comparable result

First, we transfer the code to the notebook

[1]:
```
import argparse
import itertools
import os
import time

import numpy as onp

import jax
from jax import vmap
import jax.numpy as np
import jax.random as random

import numpyro
import numpyro.distributions as dist
from numpyro.infer import MCMC, NUTS
import matplotlib.pyplot as plt
import corner

def dot(X, Z):
    return np.dot(X, Z[..., None])[..., 0]


# The kernel that corresponds to our quadratic regressor. (According to prop 6.
↪1)
```

```python
def kernel(X, Z, eta1, eta2, c, jitter=1.0e-6):
    eta1sq = np.square(eta1)
    eta2sq = np.square(eta2)
    k1 = 0.5 * eta2sq * np.square(1.0 + dot(X, Z))
    k2 = -0.5 * eta2sq * dot(np.square(X), np.square(Z))
    k3 = (eta1sq - eta2sq) * dot(X, Z)
    k4 = np.square(c) - 0.5 * eta2sq
    if X.shape == Z.shape:
        k4 += jitter * np.eye(X.shape[0])
    return k1 + k2 + k3 + k4


# Most of the model code is concerned with constructing the sparsity inducing␣
↪prior.
def model(X, Y, hypers):
    # Here X is the design matrix with N x p dimensions
    # read off dimensions P and N
    # S -  sparsity coeff
    S, P, N = hypers['expected_sparsity'], X.shape[1], X.shape[0]

    # sample variables from p. 18
    sigma = numpyro.sample("sigma", dist.HalfNormal(hypers['alpha3']))
    phi = sigma * (S / np.sqrt(N)) / (P - S)
    eta1 = numpyro.sample("eta1", dist.HalfCauchy(phi))

    msq = numpyro.sample("msq", dist.InverseGamma(hypers['alpha1'],␣
↪hypers['beta1']))
    xisq = numpyro.sample("xisq", dist.InverseGamma(hypers['alpha2'],␣
↪hypers['beta2']))

    eta2 = np.square(eta1) * np.sqrt(xisq) / msq

    lam = numpyro.sample("lambda", dist.HalfCauchy(np.ones(P)))
    kappa = np.sqrt(msq) * lam / np.sqrt(msq + np.square(eta1 * lam))

    # sample observation noise
    var_obs = numpyro.sample("var_obs", dist.InverseGamma(hypers['alpha_obs'],␣
↪hypers['beta_obs']))

    # compute kernel (as in proposition 6.1)
    kX = kappa * X
    k = kernel(kX, kX, eta1, eta2, hypers['c']) + var_obs * np.eye(N)
    assert k.shape == (N, N)

    # sample Y according to the standard gaussian process formula
    numpyro.sample("Y", dist.MultivariateNormal(loc=np.zeros(X.shape[0]),␣
↪covariance_matrix=k),
```

```
                    obs=Y)




# Compute the mean and variance of coefficient theta_i (where i = dimension)␣
↪for a
# MCMC sample of the kernel hyperparameters (eta1, xisq, ...).
# Compare to theorem 5.1 in reference [1].
def compute_singleton_mean_variance(X, Y, dimension, msq, lam, eta1, xisq, c,␣
↪var_obs):
    P, N = X.shape[1], X.shape[0]

    probe = np.zeros((2, P))
    probe = jax.ops.index_update(probe, jax.ops.index[:, dimension], np.
↪array([1.0, -1.0]))

    eta2 = np.square(eta1) * np.sqrt(xisq) / msq
    kappa = np.sqrt(msq) * lam / np.sqrt(msq + np.square(eta1 * lam))

    kX = kappa * X
    kprobe = kappa * probe

    k_xx = kernel(kX, kX, eta1, eta2, c) + var_obs * np.eye(N)
    k_xx_inv = np.linalg.inv(k_xx)
    k_probeX = kernel(kprobe, kX, eta1, eta2, c)
    k_prbprb = kernel(kprobe, kprobe, eta1, eta2, c)

    vec = np.array([0.50, -0.50]) ## a = (1/2, -1/2)
    mu = np.matmul(k_probeX, np.matmul(k_xx_inv, Y))
    mu = np.dot(mu, vec)

    var = k_prbprb - np.matmul(k_probeX, np.matmul(k_xx_inv, np.
↪transpose(k_probeX)))
    var = np.matmul(var, vec)
    var = np.dot(var, vec)

    return mu, var


# Compute the mean and variance of coefficient theta_ij for a MCMC sample of the
# kernel hyperparameters (eta1, xisq, ...). Compare to theorem 5.1 in reference␣
↪[1].
def compute_pairwise_mean_variance(X, Y, dim1, dim2, msq, lam, eta1, xisq, c,␣
↪var_obs):
    # Here X is the design matrix with N x p dimensions
```

```
    # read off dimensions P and N
    P, N = X.shape[1], X.shape[0]

    probe = np.zeros((4, P))
    probe = jax.ops.index_update(probe, jax.ops.index[:, dim1], np.array([1.0,␣
↪1.0, -1.0, -1.0]))
    probe = jax.ops.index_update(probe, jax.ops.index[:, dim2], np.array([1.0,␣
↪-1.0, 1.0, -1.0]))


    # compute eta2 and kappa from p. 18

    eta2 = np.square(eta1) * np.sqrt(xisq) / msq
    kappa = np.sqrt(msq) * lam / np.sqrt(msq + np.square(eta1 * lam))

    kX = kappa * X
    kprobe = kappa * probe

    # ?? compute a bunch of matrices w/ kernels ??
    k_xx = kernel(kX, kX, eta1, eta2, c) + var_obs * np.eye(N)
    k_xx_inv = np.linalg.inv(k_xx)
    k_probeX = kernel(kprobe, kX, eta1, eta2, c)
    k_prbprb = kernel(kprobe, kprobe, eta1, eta2, c)

    vec = np.array([0.25, -0.25, -0.25, 0.25]) ## ?? not sure why not (-1/2, 1/
↪2, -1, 1) ??
    mu = np.matmul(k_probeX, np.matmul(k_xx_inv, Y))
    mu = np.dot(mu, vec)

    var = k_prbprb - np.matmul(k_probeX, np.matmul(k_xx_inv, np.
↪transpose(k_probeX)))
    var = np.matmul(var, vec)
    var = np.dot(var, vec)

    return mu, var


# Sample coefficients theta from the posterior for a given MCMC sample.
# The first P returned values are {theta_1, theta_2, ...., theta_P}, while
# the remaining values are {theta_ij} for i,j in the list `active_dims`,
# sorted so that i < j.
def sample_theta_space(X, Y, active_dims, msq, lam, eta1, xisq, c, var_obs):␣
↪#(section B.5) ?
    # Here X is the design matrix with N x p dimensions
    # read off dimensions P and N
    # and number of active dimensions
    P, N, M = X.shape[1], X.shape[0], len(active_dims)
```

```
    # the total number of coefficients we return
    num_coefficients = P + M * (M - 1) // 2

    probe = np.zeros((2 * P + 2 * M * (M - 1), P))
    vec = np.zeros((num_coefficients, 2 * P + 2 * M * (M - 1)))
    start1 = 0
    start2 = 0

    for dim in range(P):
        probe = jax.ops.index_update(probe, jax.ops.index[start1:start1 + 2,␣
↪dim], np.array([1.0, -1.0]))
        vec = jax.ops.index_update(vec, jax.ops.index[start2, start1:start1 +␣
↪2], np.array([0.5, -0.5]))
        start1 += 2
        start2 += 1

    for dim1 in active_dims:
        for dim2 in active_dims:
            if dim1 >= dim2:
                continue
            probe = jax.ops.index_update(probe, jax.ops.index[start1:start1 +␣
↪4, dim1],
                                             np.array([1.0, 1.0, -1.0, -1.0]))
            probe = jax.ops.index_update(probe, jax.ops.index[start1:start1 +␣
↪4, dim2],
                                             np.array([1.0, -1.0, 1.0, -1.0]))
            vec = jax.ops.index_update(vec, jax.ops.index[start2, start1:start1␣
↪+ 4],
                                           np.array([0.25, -0.25, -0.25, 0.25]))
            start1 += 4
            start2 += 1

    eta2 = np.square(eta1) * np.sqrt(xisq) / msq
    kappa = np.sqrt(msq) * lam / np.sqrt(msq + np.square(eta1 * lam))

    kX = kappa * X
    kprobe = kappa * probe

    k_xx = kernel(kX, kX, eta1, eta2, c) + var_obs * np.eye(N)
    k_xx_inv = np.linalg.inv(k_xx)
    k_probeX = kernel(kprobe, kX, eta1, eta2, c)
    k_prbprb = kernel(kprobe, kprobe, eta1, eta2, c)

    mu = np.matmul(k_probeX, np.matmul(k_xx_inv, Y))
    mu = np.sum(mu * vec, axis=-1)
```

```
    covar = k_prbprb - np.matmul(k_probeX, np.matmul(k_xx_inv, np.
↪transpose(k_probeX)))
    covar = np.matmul(vec, np.matmul(covar, np.transpose(vec)))
    L = np.linalg.cholesky(covar)

    # sample from N(mu, covar)
    sample = mu + np.matmul(L, onp.random.randn(num_coefficients))

    return sample


# Helper function for doing HMC inference
def run_inference(model, args, rng_key, X, Y, hypers):
    start = time.time()
    kernel = NUTS(model)
    mcmc = MCMC(kernel, args.num_warmup, args.num_samples, num_chains=args.
↪num_chains,
                progress_bar=False if "NUMPYRO_SPHINXBUILD" in os.environ else␣
↪True)
    mcmc.run(rng_key, X, Y, hypers)
    mcmc.print_summary()
    print('\nMCMC elapsed time:', time.time() - start)
    return mcmc.get_samples()


# Get the mean and variance of a gaussian mixture
def gaussian_mixture_stats(mus, variances):
    mean_mu = np.mean(mus)
    mean_var = np.mean(variances) + np.mean(np.square(mus)) - np.square(mean_mu)
    return mean_mu, mean_var


# Create artificial regression dataset where only S out of P feature
# dimensions contain signal and where there is a single pairwise interaction
# between the first and second dimensions.
def get_data(N=20, S=2, P=10, sigma_obs=0.05):
    assert S < P and P > 1 and S > 0
    onp.random.seed(0)

    X = onp.random.randn(N, P)
    # generate S coefficients with non-negligible magnitude
    W = 0.5 + 2.5 * onp.random.rand(S)
    # generate data using the S coefficients and a single pairwise interaction
    Y = onp.sum(X[:, 0:S] * W, axis=-1) + X[:, 0] * X[:, 1] + sigma_obs * onp.
↪random.randn(N)
    Y -= np.mean(Y)
    Y_std = np.std(Y)
```

```
    assert X.shape == (N, P)
    assert Y.shape == (N,)

    return X, Y / Y_std, W / Y_std, 1.0 / Y_std


# Helper function for analyzing the posterior statistics for coefficient theta_i
def analyze_dimension(samples, X, Y, dimension, hypers):
    vmap_args = (samples['msq'], samples['lambda'], samples['eta1'],␣
↪samples['xisq'], samples['var_obs'])
    mus, variances = vmap(lambda msq, lam, eta1, xisq, var_obs:
                          compute_singleton_mean_variance(X, Y, dimension, msq,␣
↪lam,
                                                          eta1, xisq,␣
↪hypers['c'], var_obs))(*vmap_args)
    mean, variance = gaussian_mixture_stats(mus, variances)
    std = np.sqrt(variance)
    return mean, std


# Helper function for analyzing the posterior statistics for coefficient␣
↪theta_ij
def analyze_pair_of_dimensions(samples, X, Y, dim1, dim2, hypers):
    vmap_args = (samples['msq'], samples['lambda'], samples['eta1'],␣
↪samples['xisq'], samples['var_obs'])
    mus, variances = vmap(lambda msq, lam, eta1, xisq, var_obs:
                          compute_pairwise_mean_variance(X, Y, dim1, dim2, msq,␣
↪lam,
                                                         eta1, xisq,␣
↪hypers['c'], var_obs))(*vmap_args)
    mean, variance = gaussian_mixture_stats(mus, variances)
    std = np.sqrt(variance)
    return mean, std

def main(args):
    X, Y, expected_thetas, expected_pairwise = get_data(N=args.num_data, P=args.
↪num_dimensions,
                                                        S=args.
↪active_dimensions)

    # setup hyperparameters
    hypers = {'expected_sparsity': max(1.0, args.num_dimensions / 10),
              'alpha1': 3.0, 'beta1': 1.0,
              'alpha2': 3.0, 'beta2': 1.0,
              'alpha3': 1.0, 'c': 1.0,
```

```
              'alpha_obs': 3.0, 'beta_obs': 1.0}

    # do inference
    rng_key = random.PRNGKey(0)
    samples = run_inference(model, args, rng_key, X, Y, hypers)

    # compute the mean and square root variance of each coefficient theta_i
    means, stds = vmap(lambda dim: analyze_dimension(samples, X, Y, dim,␣
↪hypers))(np.arange(args.num_dimensions))

    print("Coefficients theta_1 to theta_%d used to generate the data:" % args.
↪active_dimensions, expected_thetas)
    print("The single quadratic coefficient theta_{1,2} used to generate the␣
↪data:", expected_pairwise)
    active_dimensions = []

    for dim, (mean, std) in enumerate(zip(means, stds)):
        # we mark the dimension as inactive if the interval [mean - 3 * std,␣
↪mean + 3 * std] contains zero
        lower, upper = mean - 3.0 * std, mean + 3.0 * std
        inactive = "inactive" if lower < 0.0 and upper > 0.0 else "active"
        if inactive == "active":
            active_dimensions.append(dim)
        print("[dimension %02d/%02d]  %s:\t%.2e +- %.2e" % (dim + 1, args.
↪num_dimensions, inactive, mean, std))

    print("Identified a total of %d active dimensions; expected %d." %␣
↪(len(active_dimensions),
                                                                        args.
↪active_dimensions))

    # Compute the mean and square root variance of coefficients theta_ij for␣
↪i,j active dimensions.
    # Note that the resulting numbers are only meaningful for i != j.
    if len(active_dimensions) > 0:
        dim_pairs = np.array(list(itertools.product(active_dimensions,␣
↪active_dimensions)))
        means, stds = vmap(lambda dim_pair: analyze_pair_of_dimensions(samples,␣
↪X, Y,
                                                                      ␣
↪dim_pair[0], dim_pair[1], hypers))(dim_pairs)
        for dim_pair, mean, std in zip(dim_pairs, means, stds):
            dim1, dim2 = dim_pair
            if dim1 >= dim2:
                continue
            lower, upper = mean - 3.0 * std, mean + 3.0 * std
```

```
            if not (lower < 0.0 and upper > 0.0):
                format_str = "Identified pairwise interaction between␣
↪dimensions %d and %d: %.2e +- %.2e"
                print(format_str % (dim1 + 1, dim2 + 1, mean, std))

        # Draw a single sample of coefficients theta from the posterior, where␣
↪we return all singleton
        # coefficients theta_i and pairwise coefficients theta_ij for i, j␣
↪active dimensions. We use the
        # final MCMC sample obtained from the HMC sampler.
        thetas = sample_theta_space(X, Y, active_dimensions,␣
↪samples['msq'][-1], samples['lambda'][-1],
                                    samples['eta1'][-1], samples['xisq'][-1],␣
↪hypers['c'], samples['var_obs'][-1])
        print("Single posterior sample theta:\n", thetas)
```

Next, we check that we can execute the example from the website and get a comparable result

[2]:
```
if __name__ == "__main__":
    assert numpyro.__version__.startswith('0.2.4')
    parser = argparse.ArgumentParser(description="Gaussian Process example")
    parser.add_argument("-n", "--num-samples", nargs="?", default=1000,␣
↪type=int)
    parser.add_argument("--num-warmup", nargs='?', default=500, type=int)
    parser.add_argument("--num-chains", nargs='?', default=1, type=int)
    parser.add_argument("--num-data", nargs='?', default=100, type=int)
    parser.add_argument("--num-dimensions", nargs='?', default=20, type=int)
    parser.add_argument("--active-dimensions", nargs='?', default=3, type=int)
    parser.add_argument("--device", default='cpu', type=str, help='use "cpu" or␣
↪"gpu".')
    args = parser.parse_args(args=[])

    numpyro.set_platform(args.device)
    numpyro.set_host_device_count(args.num_chains)

    main(args)
```

```
sample: 100%|      | 1500/1500 [00:34<00:00, 42.99it/s, 15 steps of size
2.18e-01. acc. prob=0.91]


                mean       std    median      5.0%     95.0%     n_eff     r_hat
      eta1      0.00      0.00      0.00      0.00      0.01    329.20      1.00
 lambda[0]    940.65   3861.83    255.35     21.13   1334.56    208.06      1.00
 lambda[1]   1375.68   6129.71    263.14     23.94   1810.23    373.75      1.00
 lambda[2]     98.68    296.77     49.96      8.81    164.59    359.24      1.00
 lambda[3]      1.19      2.10      0.71      0.01      2.49    690.21      1.00
 lambda[4]      1.27      1.66      0.75      0.00      2.83    794.74      1.00
```

```
 lambda[5]      1.09      1.53      0.67      0.00      2.31    781.26      1.00
 lambda[6]      1.27      1.69      0.74      0.00      2.95   1060.31      1.00
 lambda[7]      1.23      1.97      0.73      0.00      2.61    964.14      1.00
 lambda[8]      1.23      1.79      0.68      0.00      2.73    860.04      1.00
 lambda[9]      1.23      1.64      0.75      0.00      2.77    870.99      1.00
lambda[10]      1.16      1.57      0.74      0.00      2.50    826.66      1.00
lambda[11]      1.26      1.92      0.73      0.00      2.76   1060.91      1.00
lambda[12]      1.22      1.54      0.74      0.00      2.65    978.01      1.00
lambda[13]      1.30      1.91      0.76      0.00      3.06    669.13      1.00
lambda[14]      1.13      1.36      0.70      0.01      2.55    956.29      1.00
lambda[15]      1.32      1.91      0.71      0.00      3.04    891.08      1.00
lambda[16]      1.19      1.56      0.70      0.00      2.69    876.00      1.00
lambda[17]      1.29      2.04      0.72      0.00      2.77    878.08      1.00
lambda[18]      1.17      1.53      0.74      0.00      2.54   1050.62      1.00
lambda[19]      1.29      1.80      0.76      0.00      2.92    893.29      1.01
       msq      0.88      0.66      0.68      0.16      1.59    552.03      1.00
     sigma      0.71      0.52      0.59      0.03      1.44    912.89      1.00
   var_obs      0.02      0.00      0.02      0.02      0.03    900.03      1.00
      xisq      0.36      0.22      0.30      0.10      0.62    681.17      1.00

Number of divergences: 0

MCMC elapsed time: 42.58550786972046
Coefficients theta_1 to theta_3 used to generate the data: [0.69618857 0.7082517
0.35156995]
The single quadratic coefficient theta_{1,2} used to generate the data:
0.4637312
[dimension 01/20]  active:      6.87e-01 +- 1.51e-02
[dimension 02/20]  active:      7.05e-01 +- 1.75e-02
[dimension 03/20]  active:      3.50e-01 +- 1.72e-02
[dimension 04/20]  inactive:    6.44e-05 +- 5.06e-03
[dimension 05/20]  inactive:    -2.59e-04 +- 5.94e-03
[dimension 06/20]  inactive:    -1.59e-04 +- 5.14e-03
[dimension 07/20]  inactive:    7.09e-04 +- 6.00e-03
[dimension 08/20]  inactive:    -3.90e-04 +- 5.22e-03
[dimension 09/20]  inactive:    -3.58e-04 +- 5.29e-03
[dimension 10/20]  inactive:    -1.34e-04 +- 5.75e-03
[dimension 11/20]  inactive:    3.49e-04 +- 5.53e-03
[dimension 12/20]  inactive:    -6.93e-04 +- 5.35e-03
[dimension 13/20]  inactive:    4.80e-04 +- 5.64e-03
[dimension 14/20]  inactive:    1.88e-04 +- 5.78e-03
[dimension 15/20]  inactive:    1.31e-04 +- 5.22e-03
[dimension 16/20]  inactive:    1.82e-04 +- 5.95e-03
[dimension 17/20]  inactive:    -6.03e-04 +- 5.57e-03
[dimension 18/20]  inactive:    4.15e-06 +- 5.71e-03
[dimension 19/20]  inactive:    1.78e-04 +- 5.11e-03
[dimension 20/20]  inactive:    2.55e-04 +- 5.98e-03
Identified a total of 3 active dimensions; expected 3.
```

```
Identified pairwise interaction between dimensions 1 and 2: 4.53e-01 +- 2.15e-02
Single posterior sample theta:
 [nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan
 nan nan nan nan nan]
```

From the website, we see they ge the following result with the first 3 dimensions and a pairwise interaction between 1 and 2 identified as active.

```
Identified a total of 3 active dimensions; expected 3.
Identified pairwise interaction between dimensions 1 and 2: 4.53e-01 +- 2.15e-02
```

We also obtain these results as can be seen above. Thus, we can proceed to the next step. Note that we obtain nan values for the single posterior sample theta because the code is not yet adapted, which we will do now.

1.4.2 2. Modify `sample_theta_space()`

Next, we modify the method `sample_theta_space()` to return flat posterior samples from the MCMC only for the active direct and pairwise interaction terms.

[3]:
```
def sample_theta_space_modified(X, Y, active_dims, msq, lam, eta1, xisq, c,␣
 ↪var_obs, N_samps, dim_pair_arr):
    P, N, M = X.shape[1], X.shape[0], len(active_dims)

    num_coefficients = P + M * (M - 1) // 2

    probe = np.zeros((2 * P + 2 * M * (M - 1), P))
    vec = np.zeros((num_coefficients, 2 * P + 2 * M * (M - 1)))
    start1 = 0
    start2 = 0

    for dim in range(P):
        probe = jax.ops.index_update(probe, jax.ops.index[start1:start1 + 2,␣
 ↪dim], np.array([1.0, -1.0]))
        vec = jax.ops.index_update(vec, jax.ops.index[start2, start1:start1 +␣
 ↪2], np.array([0.5, -0.5]))
        start1 += 2
        start2 += 1

    for dim1 in active_dims:
        for dim2 in active_dims:
            if dim1 >= dim2:
                continue
            probe = jax.ops.index_update(probe, jax.ops.index[start1:start1 +␣
 ↪4, dim1],
                                          np.array([1.0, 1.0, -1.0, -1.0]))
            probe = jax.ops.index_update(probe, jax.ops.index[start1:start1 +␣
 ↪4, dim2],
                                          np.array([1.0, -1.0, 1.0, -1.0]))
```

```
            vec = jax.ops.index_update(vec, jax.ops.index[start2, start1:start1␣
↪+ 4],
                                            np.array([0.25, -0.25, -0.25, 0.25]))
            start1 += 4
            start2 += 1

    eta2 = np.square(eta1) * np.sqrt(xisq) / msq
    kappa = np.sqrt(msq) * lam / np.sqrt(msq + np.square(eta1 * lam))

    kX = kappa * X
    kprobe = kappa * probe

    k_xx = kernel(kX, kX, eta1, eta2, c) + var_obs * np.eye(N)
    k_xx_inv = np.linalg.inv(k_xx)
    k_probeX = kernel(kprobe, kX, eta1, eta2, c)
    k_prbprb = kernel(kprobe, kprobe, eta1, eta2, c)

    mu = np.matmul(k_probeX, np.matmul(k_xx_inv, Y))
    mu = np.sum(mu * vec, axis=-1)

    covar = k_prbprb - np.matmul(k_probeX, np.matmul(k_xx_inv, np.
↪transpose(k_probeX)))
    covar = np.matmul(vec, np.matmul(covar, np.transpose(vec)))
    L = np.linalg.cholesky(covar)

    #print("mu" + str(mu))
    #print("cov" + str(covar))

    # sample from N(mu, covar)
    sample = mu + np.matmul(L, onp.random.randn(num_coefficients))

    ####### ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~␣
↪##########
    ####### ~~~~~~~~~~~~~ CHANGES to the original method ~~~~~~~~~~~~~~~~~~~␣
↪#########

    # include active direct and pairwise interactions terms only
    all_active_dims = active_dims + dim_pair_arr
    mu_active = np.array([mu[i] for i in all_active_dims])

    cov_active = []
    for j in all_active_dims:
        cov_act_j = [covar[j][i] for i in all_active_dims]
        cov_active.append(cov_act_j)
    cov_active = onp.array(cov_active)

    # return posterior samples
```

```
    rng_key = random.PRNGKey(0)
    samps = numpyro.distributions.MultivariateNormal(loc = onp.array(mu_active),
                                                      covariance_matrix =␣
↪cov_active).sample(rng_key, sample_shape = (1, N_samps))

    samps_new = onp.reshape(samps, (N_samps, len(all_active_dims)))

    return samps_new
```

1.4.3 3. Modify `main()` method

Now, we modify the `main()` method to produce corner plots with posterior distributions.

[4]:
```
### X - parameters, Y - data points, {alpha_i, beta_i, c} - hyperparameters,
### N_samps - number of samples for visualization with corner
def main_modified(X, Y, args, sigma = 3.0, alpha1 = 3.0, beta1 = 1.0, alpha2 =␣
↪3.0, beta2 = 1.0,
                     alpha3 = 1.0, c = 1.0, alpha_obs = 3.0, beta_obs = 1.0,␣
↪N_samps = 1000, labels=None):


    if labels == None:
        labs = [str(_) for _ in range(X.shape[1])]
    else:
        labs = labels

    # setup hyperparameters
    hypers = {'expected_sparsity': max(1.0, X.shape[1]/2),
              'alpha1': alpha1, 'beta1': beta1,
              'alpha2': alpha2, 'beta2': beta2,
              'alpha3': alpha3, 'c': c,
              'alpha_obs': alpha_obs, 'beta_obs': beta_obs}

    # do inference
    rng_key = random.PRNGKey(0)
    samples = run_inference(model, args, rng_key, X, Y, hypers)

    # compute the mean and square root variance of each coefficient theta_i
    means, stds = vmap(lambda dim: analyze_dimension(samples, X, Y, dim,␣
↪hypers))(np.arange(X.shape[1]))
    num_dims = len(means)
    active_dimensions = []

    for dim, (mean, std) in enumerate(zip(means, stds)):
        # we mark the dimension as inactive if the interval [mean - 3 * std,␣
↪mean + 3 * std] contains zero
        lower, upper = mean - sigma * std, mean + sigma * std
```

```
        inactive = "inactive" if lower < 0.0 and upper > 0.0 else "active"
        if inactive == "active":
            active_dimensions.append(dim)
        print("[dimension %02d/%02d]  %s:\t%.2e +- %.2e" % (dim + 1, X.
↪shape[1], inactive, mean, std))

    print("Identified a total of %d active dimensions." %␣
↪(len(active_dimensions)))


    # Compute the mean and square root variance of coefficients theta_ij for␣
↪i,j active dimensions.
    # Note that the resulting numbers are only meaningful for i != j.
    if len(active_dimensions) > 0:

        dim_pairs = np.array(list(itertools.product(active_dimensions,␣
↪active_dimensions)))
        means, stds = vmap(lambda dim_pair: analyze_pair_of_dimensions(samples,␣
↪X, Y,
                                                                                 ␣
↪dim_pair[0], dim_pair[1], hypers))(dim_pairs)
        # print(dim_pairs)
        dim_pair_arr = []
        dim_pair_index = num_dims -1
        dim_pair_name = []
        pair_labs = []
        for dim_pair, mean, std in zip(dim_pairs, means, stds):
            dim1, dim2 = dim_pair
            if dim1 >= dim2:
                continue
            dim_pair_index += 1
            lower, upper = mean - sigma * std, mean + sigma * std
            if not (lower < 0.0 and upper > 0.0):
                dim_pair_arr.append(dim_pair_index)
                dim_pair_name.append('%d and %d'%(dim1 + 1, dim2 + 1))
                format_str = "Identified pairwise interaction between␣
↪dimensions %d and %d: %.2e +- %.2e"
                print(format_str % (dim1 + 1, dim2 + 1, mean, std))
                pair_labs.append(str(labs[dim1]) + ' and ' + str(labs[dim2]))

        # Draw a single sample of coefficients theta from the posterior, where␣
↪we return all singleton
        # coefficients theta_i and pairwise coefficients theta_ij for i, j␣
↪active dimensions. We use the
        # final MCMC sample obtained from the HMC sampler.
```

```
        #######␣
↪~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ ##########
        ####### ~~~~~~~~~~~~~ CHANGES to the original method␣
↪~~~~~~~~~~~~~~~~~~~ #########

        ## Get posterior samples from the sample_theta_space_modified() method
        thetas = sample_theta_space_modified(X, Y, active_dimensions,␣
↪samples['msq'][-1], samples['lambda'][-1],
                                    samples['eta1'][-1],␣
↪samples['xisq'][-1], hypers['c'],
                                    samples['var_obs'][-1], N_samps,␣
↪dim_pair_arr)
        print("Active dimensions: " + str(active_dimensions))

        ##  Visualize the posterior from the example with corner

        labels = ['dim '+str(i) for i in active_dimensions]
        active_dimensions = active_dimensions + dim_pair_arr
        if len(dim_pair_name) != 0:
            for n in range(len(dim_pair_name)):
                labels.append('dim ' + dim_pair_name[n])
        #fig = corner.corner(thetas, labels = labels);
        return active_dimensions, thetas, labels, pair_labs
    else:
        return active_dimensions, [], []
```

1.4.4 4. Check modified methods working properly

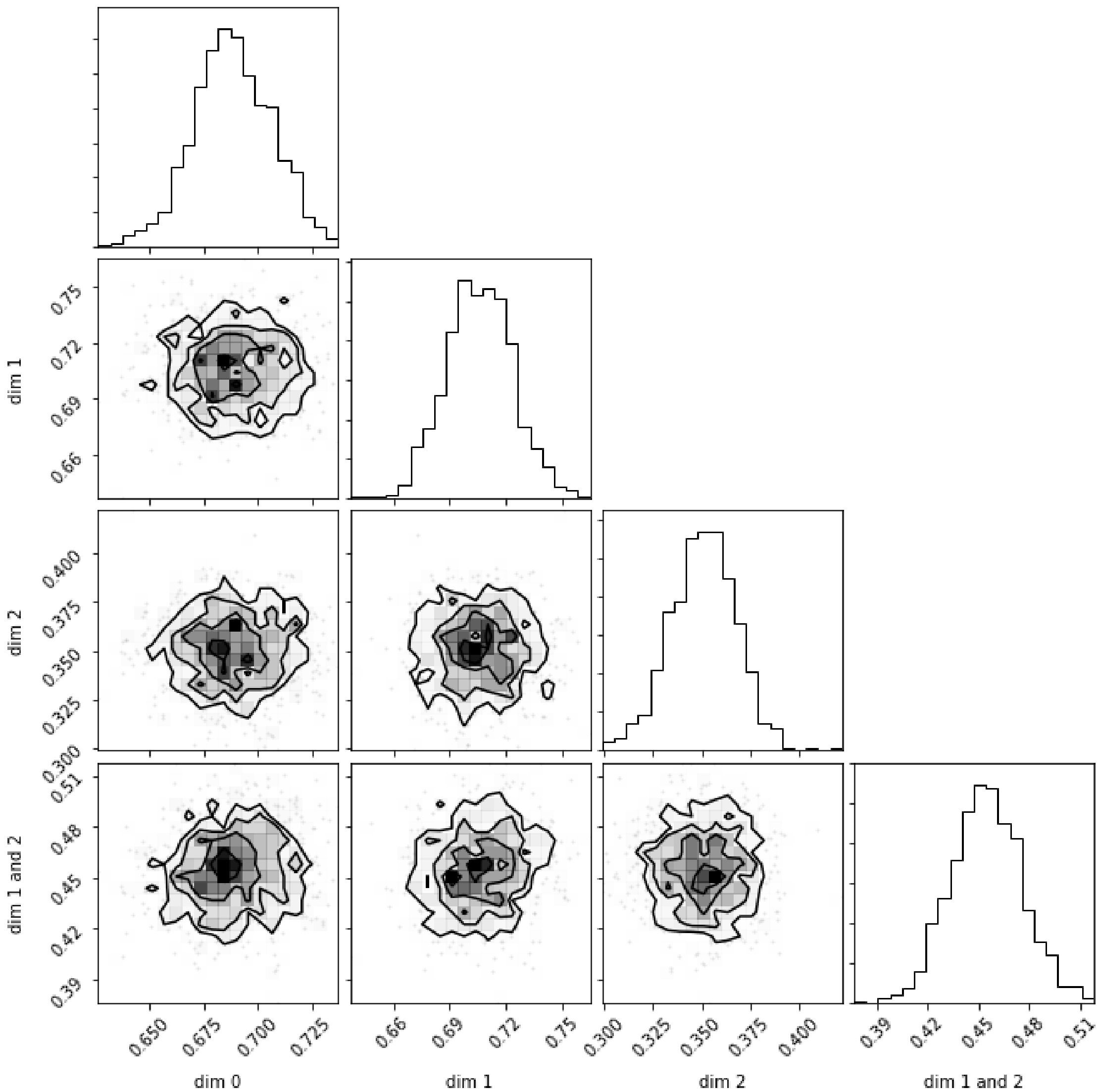
Finally, we check that the new `sample_theta_space_modified()` and `main_modified()` methods return expected results with the example from the website.

[5]:
```
if __name__ == "__main__":
    assert numpyro.__version__.startswith('0.2.4')
    parser = argparse.ArgumentParser(description="Gaussian Process example")
    parser.add_argument("-n", "--num-samples", nargs="?", default=1000,␣
↪type=int)
    parser.add_argument("--num-warmup", nargs='?', default=500, type=int)
    parser.add_argument("--num-chains", nargs='?', default=1, type=int)
    parser.add_argument("--num-data", nargs='?', default=100, type=int)
    parser.add_argument("--num-dimensions", nargs='?', default=20, type=int)
    parser.add_argument("--active-dimensions", nargs='?', default=3, type=int)
    parser.add_argument("--device", default='cpu', type=str, help='use "cpu" or␣
↪"gpu".')
    args = parser.parse_args(args=[])

    numpyro.set_platform(args.device)
```

```
    numpyro.set_host_device_count(args.num_chains)
    X, Y, expected_thetas, expected_pairwise = get_data(N=args.num_data, P=args.
↪num_dimensions,
                                                          S=args.
↪active_dimensions)


    all_active_dimensions, thetas, labels, pair_labs = main_modified(X, Y, args)
    fig = corner.corner(thetas, labels = labels)
```

```
sample: 100%|      | 1500/1500 [00:32<00:00, 46.81it/s, 31 steps of size
1.75e-01. acc. prob=0.91]


                mean       std    median      5.0%     95.0%     n_eff     r_hat
      eta1      0.01      0.00      0.00      0.00      0.01    522.79      1.00
 lambda[0]    810.60   4321.92    207.47     23.54   1055.89    764.45      1.00
 lambda[1]   1258.78   5891.21    214.57     24.09   1484.57    430.87      1.00
 lambda[2]     99.91    827.92     43.46      5.83    135.17    939.48      1.00
 lambda[3]      1.23      1.86      0.68      0.00      2.69    658.71      1.00
 lambda[4]      1.28      1.87      0.73      0.01      2.97    880.66      1.00
 lambda[5]      1.08      1.51      0.65      0.00      2.31    933.90      1.00
 lambda[6]      1.20      1.60      0.75      0.00      2.68   1036.53      1.00
 lambda[7]      1.12      1.48      0.66      0.00      2.50    998.68      1.00
 lambda[8]      1.12      1.41      0.69      0.00      2.56   1006.67      1.00
 lambda[9]      1.05      1.24      0.68      0.00      2.30   1017.29      1.00
lambda[10]      1.17      1.64      0.69      0.00      2.48    664.60      1.00
lambda[11]      1.18      1.93      0.69      0.00      2.50   1225.14      1.00
lambda[12]      1.20      1.78      0.69      0.00      2.64   1268.16      1.00
lambda[13]      1.19      1.82      0.64      0.00      2.51    625.67      1.00
lambda[14]      1.12      1.42      0.68      0.01      2.60    901.60      1.00
lambda[15]      1.27      1.75      0.73      0.01      2.87    893.50      1.00
lambda[16]      1.10      1.53      0.70      0.00      2.30   1314.54      1.00
lambda[17]      1.23      1.67      0.71      0.01      2.81    956.93      1.00
lambda[18]      1.13      1.79      0.67      0.00      2.66   1126.14      1.00
lambda[19]      1.19      1.88      0.72      0.00      2.60    839.72      1.00
       msq      0.87      0.61      0.71      0.23      1.54    681.22      1.00
     sigma      0.39      0.42      0.23      0.00      0.97   1025.90      1.00
   var_obs      0.02      0.00      0.02      0.02      0.03   1020.53      1.00
      xisq      0.36      0.24      0.29      0.11      0.62    685.86      1.01

Number of divergences: 0

MCMC elapsed time: 35.37909507751465
[dimension 01/20]  active:      6.87e-01 +- 1.54e-02
[dimension 02/20]  active:      7.05e-01 +- 1.76e-02
[dimension 03/20]  active:      3.50e-01 +- 1.73e-02
[dimension 04/20]  inactive:    8.12e-05 +- 5.55e-03
```

```
[dimension 05/20]  inactive:    -2.95e-04 +- 6.52e-03
[dimension 06/20]  inactive:    -1.94e-04 +- 5.76e-03
[dimension 07/20]  inactive:    7.68e-04 +- 6.28e-03
[dimension 08/20]  inactive:    -4.25e-04 +- 5.52e-03
[dimension 09/20]  inactive:    -4.24e-04 +- 5.71e-03
[dimension 10/20]  inactive:    -1.38e-04 +- 5.77e-03
[dimension 11/20]  inactive:    4.07e-04 +- 6.06e-03
[dimension 12/20]  inactive:    -7.62e-04 +- 5.66e-03
[dimension 13/20]  inactive:    5.33e-04 +- 6.00e-03
[dimension 14/20]  inactive:    2.03e-04 +- 5.95e-03
[dimension 15/20]  inactive:    1.51e-04 +- 5.54e-03
[dimension 16/20]  inactive:    2.03e-04 +- 6.39e-03
[dimension 17/20]  inactive:    -6.61e-04 +- 5.89e-03
[dimension 18/20]  inactive:    5.85e-06 +- 6.17e-03
[dimension 19/20]  inactive:    2.05e-04 +- 5.50e-03
[dimension 20/20]  inactive:    2.68e-04 +- 6.17e-03
Identified a total of 3 active dimensions.
Identified pairwise interaction between dimensions 1 and 2: 4.53e-01 +- 2.17e-02
Active dimensions: [0, 1, 2]
```



We again identify the same active dimensions and pairwise interactions. Additionally, the corner plots look sensible. With operational code, we can now proceed to the Application section.

1.4.5 Application (4pts)

COVID-19 dominates the news, with many countries still reporting rising case numbers. One surprising fact is that the mortality rate (i.e. the fraction of the infected who have died, also called **case-fatality ratio**) differs *a lot* between countries, from 15% to less than 1%. Find out which features (such as age distribution, population health indices, economic factors, COVID-specific factors ...) influences that rate.

This task has three parts:

- Think about what possible effects there could be.
- Find suitable data for as many countries as possible in public data archives. Combine them into a master data set.
- Perform the inference.

You will probably need to iterate and refine along the way. Explain your reasoning about the kinds of features you decided to include in your analysis. Then report the most important direct and pairwise interactions. Visualized the posterior samples with `corner`.

Note: This is an exploratory study. If your approach is sound, but the data don't show firm trends, partial points will be awarded. Include your final data compilation as a separate file with your submission.

Hints:

- Start here.
- Don't forget to standardize the data by subtracting the mean and dividing by the standard deviation.

1.5 Application Response

1.5.1 The Target Variable (y-variable)

The goal of this exercise is to identify explanatory variables that are expressive in the Covid-19 mortality rate across different countries. To do so, we consider a regression setup in which the Covid-19 mortatility rate is our target variable (y-variable). We interpret the realization of the target variable for any given country as an observation (or sample).

We source the Covid-19 mortality rate across countries from the John-Hopkins website. We have the death rate available for a total of 144 countries. We take a first look into the y-variable which is the Covid-19 mortality rate.

[8]:
```
import pandas as pd
import matplotlib.pyplot as plt
```

[9]:
```
y_var = pd.read_csv("JHU_recent_mortality_by_country.csv",
                    converters = {"Case-Fatality": lambda x: float(x.
 ↪strip("%"))/100})

(y_var.head())
```

[9]:
```
          Country  Case-Fatality
0              US          0.059
1  United Kingdom          0.150
2           Italy          0.138
3           Spain          0.117
4          France          0.150
```

We derive the basic sample statistics for our target variable.

[10]:
```
y_var.describe()
```

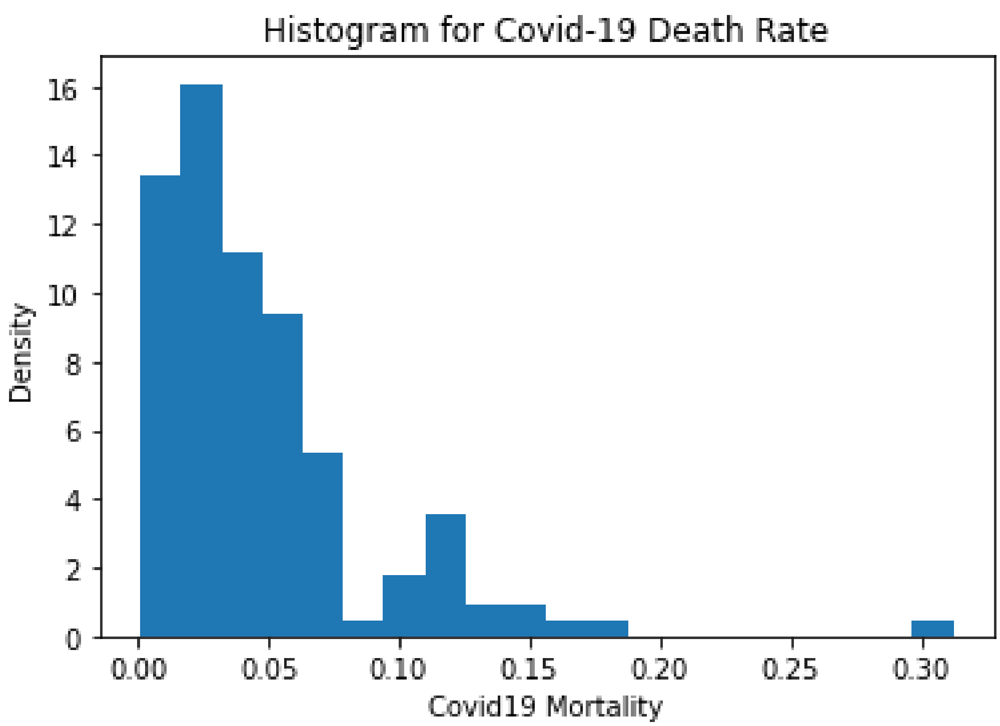
[10]:
```
         Case-Fatality
count       144.000000
mean          0.047083
std           0.043048
min           0.001000
25%           0.018000
50%           0.035500
75%           0.061000
max           0.312000
```

We make the following observation:

- The standard-deviation of the mortality is relatively high.
- The mortality ranges from ~0.1% to ~30%. This is a very wide range.

We plot the histogram for the mortality to get a sense for the distribution of our target variable.

[11]:
```
plt.hist(y_var.iloc[:,1], density = True, bins = 20)
plt.ylabel("Density")
plt.xlabel("Covid19 Mortality")
plt.title("Histogram for Covid-19 Death Rate");
```



1.5.2 The Explanatory Variables (X-data)

Our goal is to construct a regression model in which we predict the target variable. To do so, we need to come up with a set of explanatory variables that is expressive in predicting the Covid19 mortality rate. The data on the explanatory variables should be available for as many countries as possible, to create a data set of X-and y-data that is large enough.

Our approach is to come up with explanatory variables that we intuitively think are important to predict the Covid-19 rate. Then, we try to find data on these explanatory variables for as many countries as possible on the internet. We used the following sources for our data:

- World Health Organization (WHO),
- Wikipedia,
- worldindata.com,
- worldometer.com.

We source each explanatory variable from the internet in separate .csv files. Then, we use our data compilation code to combine all the different .csv files into one large .csv file, which we call "raw_data.csv". Please refer to the data compilation algorithm in the separate .ipynb file.

Load the .csv file containing the combined data sets.

[12]:
```
jhu = pd.read_csv("raw_data.csv")
```

These are the explanatory variables that we consider in order to predict the Covid19 mortality rate.

[13]:
```
pd.DataFrame(jhu.columns[2:], columns = ["Variables"])
```

[13]:
```
                                        Variables
0                   Human Development Index (HDI)
1                   Population median age (years)
2                             Adult mortality rate
3                Life expectancy at birth (years)
4    Mortality rate attributed to exposure to unsaf…
5                Population proportion over 60 (%)
6    Current health expenditure (CHE) per capita in…
7                 UHC index of service coverage (SCI)
8                          Cases per 1M population
9         Urban population (% of total population)
10                             diabetes_prevalence
11                           hospital_beds_per_100k
12                                  cvd_death_rate
13                                  GDP per capita
14                                   Tests/ 1M pop
```

[14]:
```
jhu.columns[2:].tolist()
```

[14]:
```
['Human Development Index (HDI)',
 'Population median age (years)',
 'Adult mortality rate',
```

```
'Life expectancy at birth (years)',
'Mortality rate attributed to exposure to unsafe WASH services',
'Population proportion over 60 (%)',
'Current health expenditure (CHE) per capita in US$',
'UHC index of service coverage (SCI)',
'Cases per 1M population',
'Urban population (% of total population)',
'diabetes_prevalence',
'hospital_beds_per_100k',
'cvd_death_rate',
'GDP per capita',
'Tests/ 1M pop']
```

The explanatory variables that we consider can be classified as follows:

- Demographic data (population median age, population proportion over 60%, urban population),
- Economic indicators (GDP per capita),
- Health care related indicators (adult mortaility rate, life expectancy at birth, mortality rate attributed to exposure to unsafe washing practices, health care expenditure per capita, diabetes prevelance, hospital beds per capita, CVD death rate),
- Covid-19 related indicators (Coronavirus cases per capita, Covid19 test rate),
- Indices that reflect a combination of economic, heatlh care and demographic information (Human Development Index, UHC index).

The justifications for these variables are as follows:

- HDI: More developed countries are better equipped to handle a large-scale health crisis, so may be better able to reduce the mortality rate.
- Median age: COVID-19 appears to be affecting the elderly more than the young, so an older population may be more likely to have a higher mortality rate.
- Mortality rate: Deaths from COVID-19 might be mistaken for other causes (or vice versa) in the early reporting so there could be relationship between overall mortality rate and COVID-19 mortality rate. Also countries like Belgium are including deaths of people who have not been tested but are suspected of having had COVID-19 in their COVID-1 death tallies.
- Mortality rate (unsafe WASH): A key part of the COVID-19 prevention guidelines is regular and thorough washing / cleansing; thus, if a country already has deaths from poor access to water, sanitation, and hygiene services, it is more likely to be difficult to 'flatten the curve' in that country.
- Population 60+: Similar to median age.
- Current health expenditure (CHE): Countries that spend more on healthcare are perhaps more likely to be able to handle a large-scale health crisis.
- UHC index: Similar to CHE – countries with better access to necessary health services are perhaps more likely to be able to handle a large-scale health crisis.
- Cases per 1M: This is (roughly) the denominator of the mortality rate so is closely linked. More cases may mean more moderate cases of COVID-19 have been found so the mortality rate may be lower. Alternatively, more cases may just mean the severity of the spread is higher and so the mortality rate could be higher because the pandemic is worse in that country.
- Urbanisation: More urban environments are more densely-packed and so contact / spread is

more likely: people use public transport, live in apartment buildings, eat out at restaurants more, go to the same parks, etc.
- Diabetes: COVID-19 does affect the more vulnerable portions of the population more, but this is also one (rudimentary) indicator of the pre-COVID-19 health of the population: more diabetics might indicate a generally less healthy population.
- Hospital beds: The issues of insufficient hospital beds and ventilators and PPE have been all over the news for weeks as hospitals are overrun with COVID-19 patients. More hospital beds may indicate a country is better able to handle the pandemic and maybe also just better prepared generally.
- COVID-19 death rate: This is of course linked to the COVID-19 mortality rate, so we feel it could be a good explanatory variable.
- GDP: Similar to HDI.
- Tests / 1M: The number of tests relates to the number of cases as more tests will lead to more positive results. Additionally, more tests may indicate a country's responsiveness to the crisis.

As described above, the explanatory data is obtained from different sources from the internet. Depending on the source, the information on any given explanatory variable is not always available across all countries. Furthermore, the data is sometimes not clean. Thus, we need to clean the data and deal with missing values. To do so, we have a data cleaning process in-place. The process follows the following logic:

- If any country has less than 90% of the data on the variables available, the country is dropped. This is to exclude countries that have too few information available.
- If a variable has less than 70% of the data across countries available, then, the variable is dropped.
- Countries belonging to the lower 40-percentile in terms of population size are dropped. This is because we deem small countries to not be representiative and the information too noisy.
- We fill any remaining NA using a data imputation algorithm. The data imputation algorithm is based on predicting missing data with a linear regression and pruning the prediction with the sample median and sample median absolute deviation.

We laod the clean data set and take a first look.

[15]:
```
jhu.head()
```

[15]:
```
          Country  Case-Fatality  Human Development Index (HDI)  \
0              US          0.059                          0.920
1  United Kingdom          0.150                          0.920
2           Italy          0.138                          0.883
3          France          0.150                          0.891
4         Belgium          0.159                          0.919

   Population median age (years)  Adult mortality rate  \
0                           37.4                 114.0
1                           40.2                  67.0
2                           44.3                  54.0
3                           40.6                  71.0
4                           41.6                  72.0
```

```
   Life expectancy at birth (years)  \
0                             78.5
1                             81.4
2                             82.8
3                             82.9
4                             81.2

   Mortality rate attributed to exposure to unsafe WASH services  \
0                                                0.2
1                                                0.2
2                                                0.1
3                                                0.3
4                                                0.3

   Population proportion over 60 (%)  \
0                             195.6
1                              45.0
2                              49.7
3                              47.4
4                               8.0

   Current health expenditure (CHE) per capita in US$  \
0                                           10246.1
1                                            3858.7
2                                            2840.1
3                                            4379.7
4                                            4507.4

   UHC index of service coverage (SCI)  Cases per 1M population  \
0                                 84.0                   3763.0
1                                 87.0                   3027.0
2                                 82.0                   3560.0
3                                 78.0                   1982.0
4                                 84.0                   4403.0

   Urban population (% of total population)  diabetes_prevalence  \
0                                    82.256                10.79
1                                    83.398                 4.28
2                                    70.438                 4.78
3                                    80.444                 4.77
4                                    98.001                 4.29

   hospital_beds_per_100k  cvd_death_rate  GDP per capita  Tests/ 1M pop
0                    2.77         151.089         59928.0        23522.0
1                    2.54         122.137         44920.0        20385.0
2                    3.18         113.151         40924.0        38221.0
```

```
3                    5.98         86.060         44033.0         16856.0
4                    5.64        114.898         49367.0         40914.0
```

We calculate the basic sample statistics on our data set.

```
[16]: jhu.describe()
```

```
[16]:        Case-Fatality  Human Development Index (HDI)  \
count        68.000000                      68.000000
mean          0.051574                       0.735721
std           0.039127                       0.154303
min           0.006000                       0.377000
25%           0.021750                       0.620750
50%           0.042000                       0.760000
75%           0.066750                       0.859250
max           0.159000                       0.946000

       Population median age (years)  Adult mortality rate  \
count                      68.000000             68.000000
mean                       29.580882            144.735294
std                         9.102997             74.459168
min                        15.000000             49.000000
25%                        22.125000             79.000000
50%                        27.750000            128.500000
75%                        38.725000            182.750000
max                        45.900000            334.000000

       Life expectancy at birth (years)  \
count                         68.000000
mean                          73.398529
std                            7.193602
min                           58.000000
25%                           68.300000
50%                           75.250000
75%                           77.975000
max                           84.200000

       Mortality rate attributed to exposure to unsafe WASH services  \
count                                          68.000000
mean                                           10.235294
std                                            17.544804
min                                             0.050000
25%                                             0.200000
50%                                             1.000000
75%                                            13.750000
max                                            70.800000
```

	Population proportion over 60 (%)
count	68.000000
mean	34.258824
std	90.781325
min	0.600000
25%	3.875000
50%	8.300000
75%	26.000000
max	646.000000

	Current health expenditure (CHE) per capita in US$
count	68.000000
mean	1403.435294
std	2248.267614
min	25.300000
25%	98.500000
50%	408.650000
75%	1363.250000
max	10246.100000

	UHC index of service coverage (SCI)	Cases per 1M population
count	68.000000	68.000000
mean	67.411765	786.823529
std	15.316789	1088.427922
min	37.000000	2.000000
25%	55.000000	56.000000
50%	74.000000	165.500000
75%	78.250000	1426.500000
max	89.000000	4403.000000

	Urban population (% of total population)	diabetes_prevalence
count	68.000000	68.000000
mean	62.559838	7.414853
std	21.869877	3.535779
min	16.425000	1.820000
25%	46.274250	5.235000
50%	67.650000	6.965000
75%	80.527500	9.020000
max	98.001000	17.720000

	hospital_beds_per_100k	cvd_death_rate	GDP per capita	Tests/ 1M pop
count	68.000000	68.000000	68.000000	68.000000
mean	2.830300	239.031368	21761.014706	11243.658348
std	2.554781	119.321789	19044.177430	18429.221893
min	0.100000	79.370000	1019.000000	50.000000
25%	1.100000	143.999250	5403.500000	913.500000
50%	1.950000	222.984000	15492.500000	3261.106915

```
75%                    3.660000     286.210750    30581.500000    14308.500000
max                   13.050000     597.029000    74035.000000   121330.000000
```

1.5.3 A First Look into Variable Selection: The Lasso

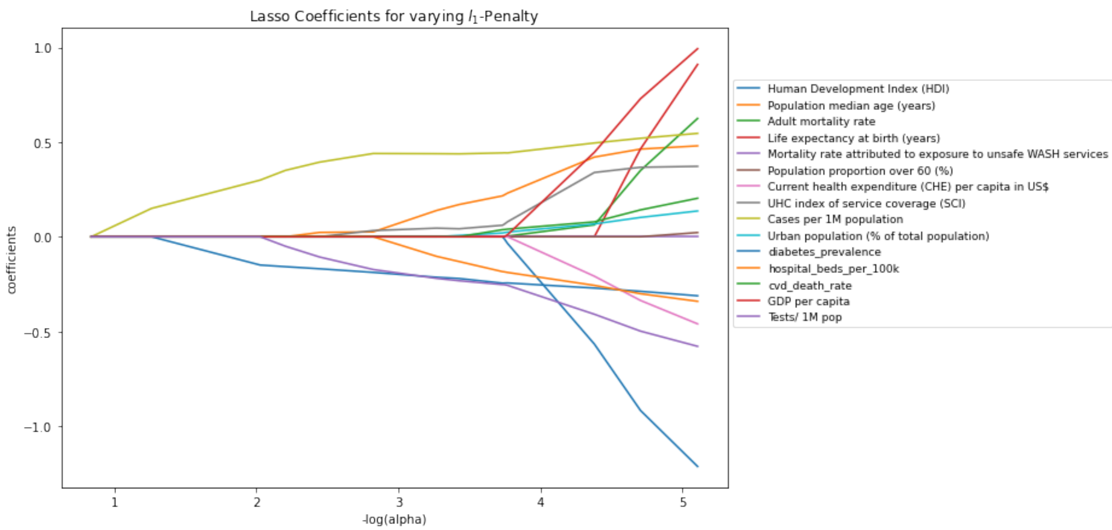
Before applying the SKIM method to our data set, we want to get a first impression on which of the explanatory variables might be important to predict our target variable. We apply Lasso to the data set to do so. Lasso is a regression method that adds a l_1-penalty to the sum-of-squares error to obtain a sparse coefficient vector. More specifically, the Lasso coefficients is the solution of

$$\min_{\beta \in \mathbb{R}^d} \frac{1}{2n} ||y - X\beta||_2^2 + \alpha ||\beta||_1.$$

Depending on the magnitude of the hyperparameter α, which controls for the l_1-penalty, we obtain a lasso coefficient vector $\hat{\beta}_{\text{Lasso}}$ with variying degree of sparsity. Now, we can solve this Lasso problem for varying levels of the α. In this way, we can begin our analysis with a completely sparse coefficient vector and reduce the sparsity in each step to include explanatory variables step-by-step. This is called the **Lasso path**. In the following, we calculate the Lasso path.

[17]:
```
##############################################################################
#Lasso heuristics
##############################################################################

#define X and y variable set
X = np.array(jhu.iloc[:,2:])
y = np.array(jhu.iloc[:,1])

#standardize data
from sklearn.preprocessing import StandardScaler
X = StandardScaler().fit(X).transform(X)
y = StandardScaler().fit(y.reshape(-1,1)).transform(y.reshape(-1,1)).ravel()

#apply Lasso path
from sklearn.linear_model import lars_path
alphas, active, coef_path_lars = lars_path(X, y, method='lasso')

#define pd data frame with active coefficients
var_sel = pd.DataFrame(coef_path_lars, index = jhu.columns[2:],
                       columns = onp.vectorize(lambda x: "alpha=␣
 ↪"+str(round(x,2)))(alphas))

import matplotlib.pyplot as plt
fig = plt.figure(figsize=(10,7))
ax = plt.subplot(111)

for i in range(var_sel.shape[0]):
    ax.plot(-np.log(alphas[:-1]), var_sel.iloc[i,:-1])

plt.title("Lasso Coefficients for varying $l_1$-Penalty")
```

```python
plt.xlabel("-log(alpha)")
plt.ylabel("coefficients")
# Shrink current axis by 20%
box = ax.get_position()
ax.set_position([box.x0, box.y0, box.width * 0.99, box.height])

# Put a legend to the right of the current axis
ax.legend(jhu.columns[2:], bbox_to_anchor=(1.0, 0.9),fontsize=9)
plt.show()
```



With varying degrees of l_1-penalty, the Lasso selects the following variables in the following order:

[18]:
```python
pd.DataFrame(jhu.columns[2:][active], columns = ["Lasso Selection"])
```

[18]:

	Lasso Selection
0	Cases per 1M population
1	diabetes_prevalence
2	Tests/ 1M pop
3	Population median age (years)
4	UHC index of service coverage (SCI)
5	hospital_beds_per_100k
6	Urban population (% of total population)
7	cvd_death_rate
8	Human Development Index (HDI)
9	Adult mortality rate
10	GDP per capita
11	Current health expenditure (CHE) per capita in…
12	Life expectancy at birth (years)
13	Population proportion over 60 (%)

Based on this Lasso-based variable selection process, has idenitfied the following variables as important:

- Number of Coronavirus cases per capita,
- The diabetes prevalance,
- Coronavirus testing rate.

Remarkably, the Lasso identified both Covid-19 related indicators. This makes intuitive sense. With a low Coronavirus testing rate, more severe cases are being tested, introducing a positive bias in the Covid19 mortatility.

Next, we will apply the SKIM method to our data set to obtain an alternative set of relevant variables.

We found that the out-of-the box hyperparameters given in the example code for the toy sparsity problem were unable to capture the massive variance in parameter prediction that we recovered after a couple of naive runs. The result was no reported active dimensions or pairwise interactions, even though the data possess strong qualitatively-correlated trends.

To mitigate this, we ran a Sequential Model-Based Optimization scheme on the Inverse Gamma hyperparameters using the package HyperOpt on an HPC cluster. We probed log-uniform hyperparameter space, making $-\exp(N_{\text{active}})$ our loss criterion, where

$$N_{\text{active}} = \text{num active dims} + \text{num pairwise active dims}$$

We also relax the built-in criterion for counting active dimensions to be a well-sampled distribution (no zero bins) within $\pm 1.5\sigma$ of the mean of the parameter contour estimate. The result of our optimization run was a set of rather unintuitive hyperparameters. Our Inverse Gamma location parameters α_1, α_2, and α_3 were found to be relatively small, while the parameter α_{obs} needed to be much much larger (~ 50) to recover any active dimensions over 540 optimization trials.

Below we run the optimized inference over COVID-19 data with different $\pm\sigma$ criterion. We recover the most intuitive set of Case Fatality-causing factors when using the bounds $\sigma = 1.5$.

We run SKIM for $\sigma = 3, 2, 1.5, 1$ below with 5000 samples.

[19]:
```
short_labels = ['HDI', 'PMA', 'AMR', 'LEB', 'wash', 'over-60', 'CHE', 'UHC',␣
 ↪'cases / 1M',
                'urban', 'diabetes', 'hbeds', 'cvd_dr', 'GDP', 'tests / 1M']

pd.DataFrame(zip(jhu.columns[2:].tolist(), short_labels), columns=['Variable',␣
 ↪'Short Label'])
```

[19]:

	Variable	Short Label
0	Human Development Index (HDI)	HDI
1	Population median age (years)	PMA
2	Adult mortality rate	AMR
3	Life expectancy at birth (years)	LEB

```
4    Mortality rate attributed to exposure to unsaf…        wash
5                       Population proportion over 60 (%)     over-60
6    Current health expenditure (CHE) per capita in…         CHE
7                    UHC index of service coverage (SCI)         UHC
8                                 Cases per 1M population  cases / 1M
9                Urban population (% of total population)       urban
10                                    diabetes_prevalence    diabetes
11                                  hospital_beds_per_100k       hbeds
12                                          cvd_death_rate      cvd_dr
13                                          GDP per capita         GDP
14                                           Tests/ 1M pop  tests / 1M
```

[20]:

```
# Optimised hyperparameters
hypers = {'alpha1': 0.26872577050471647,
          'alpha2': 4.818866884657901,
          'alpha3': 6.812836016637205,
          'alpha_obs': 49.093191654133754,   # this appears to be the key␣
 ↪difference
          'beta1': 1.2942745182532156,
          'beta2': 0.21267247881371867,
          'beta_obs': 3.3406960438157594,
          'c': 2.751017838090896}

# Numpyro args
n_data, n_dimensions = X.shape

parser = argparse.ArgumentParser(description="Gaussian Process example")
parser.add_argument("-n", "--num-samples", nargs="?", default=5000, type=int)
parser.add_argument("--num-warmup", nargs='?', default=500, type=int)
parser.add_argument("--num-chains", nargs='?', default=1, type=int)
parser.add_argument("--num-data", nargs='?', default=n_data, type=int)
parser.add_argument("--num-dimensions", nargs='?', default=n_dimensions,␣
 ↪type=int)
parser.add_argument("--active-dimensions", nargs='?', default=3, type=int)
parser.add_argument("--device", default='cpu', type=str, help='use "cpu" or␣
 ↪"gpu".')
args = parser.parse_args(args=[])

numpyro.set_platform(args.device)
numpyro.set_host_device_count(args.num_chains)
```

1.6 $\sigma = 3$

[21]:

```
all_active_dimensions, thetas, labels, pair_labs = main_modified(X=X, Y=y,␣
 ↪args=args, sigma=3.0, N_samps=5000,
```

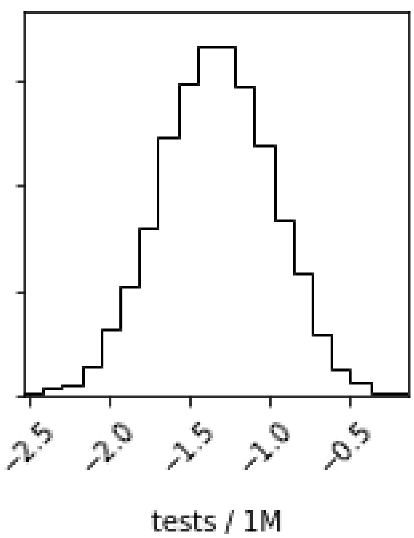
```
↪labels=short_labels, **hypers)
```

```
sample: 100%|      | 5500/5500 [00:40<00:00, 134.93it/s, 15 steps of size
2.41e-01. acc. prob=0.90]

                 mean       std    median      5.0%     95.0%     n_eff     r_hat
      eta1       3.51      4.42      2.08      0.20      7.32    747.07      1.00
 lambda[0]      23.25    452.41      1.78      0.01     10.85   1453.65      1.00
 lambda[1]       7.00     67.61      1.55      0.01      9.48   4483.16      1.00
 lambda[2]     121.94   5907.21      1.35      0.00      7.29   2611.16      1.00
 lambda[3]       4.48     27.25      1.19      0.00      6.83   3831.72      1.00
 lambda[4]       4.92    103.83      0.44      0.00      3.56   2887.68      1.00
 lambda[5]       3.98     41.79      0.63      0.00      4.64   3070.82      1.00
 lambda[6]       9.70     88.28      1.52      0.03     10.37   3008.38      1.00
 lambda[7]       4.71     27.18      1.39      0.04      7.62   3249.94      1.00
 lambda[8]       6.91     53.79      1.23      0.02      6.97   1416.95      1.00
 lambda[9]       2.79     63.51      0.13      0.00      1.34   4655.17      1.00
lambda[10]       2.34     18.14      0.41      0.00      2.86   2201.67      1.00
lambda[11]       9.30    177.00      1.39      0.00      7.89   4378.34      1.00
lambda[12]       5.62     84.31      1.21      0.00      6.78   2168.99      1.00
lambda[13]       6.82     64.57      1.51      0.01      8.48   3922.69      1.00
lambda[14]       7.85    186.08      1.40      0.04      8.57   4926.52      1.00
       msq       0.83      0.52      0.70      0.25      1.47   3279.58      1.00
     sigma       7.60      4.16      7.10      0.82     13.53   5977.26      1.00
   var_obs       0.13      0.03      0.13      0.09      0.17   2712.67      1.00
      xisq       0.09      0.06      0.08      0.02      0.16   3062.87      1.00

Number of divergences: 111

MCMC elapsed time: 48.55317974090576
[dimension 01/15]  inactive:    -9.49e-01 +- 5.04e-01
[dimension 02/15]  inactive:    4.41e-01 +- 2.48e-01
[dimension 03/15]  inactive:    3.05e-01 +- 3.40e-01
[dimension 04/15]  inactive:    6.90e-01 +- 4.74e-01
[dimension 05/15]  inactive:    -1.20e-01 +- 3.40e-01
[dimension 06/15]  inactive:    1.09e-01 +- 1.76e-01
[dimension 07/15]  inactive:    1.48e-01 +- 5.16e-01
[dimension 08/15]  inactive:    6.40e-01 +- 2.64e-01
[dimension 09/15]  inactive:    -1.75e-01 +- 3.08e-01
[dimension 10/15]  inactive:    9.11e-02 +- 1.33e-01
[dimension 11/15]  inactive:    -2.04e-01 +- 1.16e-01
[dimension 12/15]  inactive:    -3.05e-01 +- 2.54e-01
[dimension 13/15]  inactive:    2.72e-01 +- 1.98e-01
[dimension 14/15]  inactive:    7.54e-01 +- 4.50e-01
[dimension 15/15]  active:      -1.16e+00 +- 3.64e-01
Identified a total of 1 active dimensions.
```

```
Active dimensions: [14]
```

[22]:
```
labs = [short_labels[i] for i in all_active_dimensions if i <=␣
 ↪len(short_labels)] + pair_labs
fig = corner.corner(thetas, labels = labs)
fig.show()
```

```
/Users/sachinsmart/opt/anaconda3/lib/python3.7/site-
packages/ipykernel_launcher.py:3: UserWarning: Matplotlib is currently using
module://ipykernel.pylab.backend_inline, which is a non-GUI backend, so cannot
show the figure.
  This is separate from the ipykernel package so we can avoid doing imports
until
```



[23]:
```
print('Active dimensions:', labs)
```

```
Active dimensions: ['tests / 1M']
```

1.7 $\sigma = 2$

[24]:
```
all_active_dimensions, thetas, labels, pair_labs = main_modified(X=X, Y=y,␣
 ↪args=args, sigma=2.0, N_samps=5000,
                                                                            ␣
 ↪labels=short_labels, **hypers)
```

```
sample: 100%|      | 5500/5500 [00:41<00:00, 131.94it/s, 15 steps of size
2.41e-01. acc. prob=0.90]
```

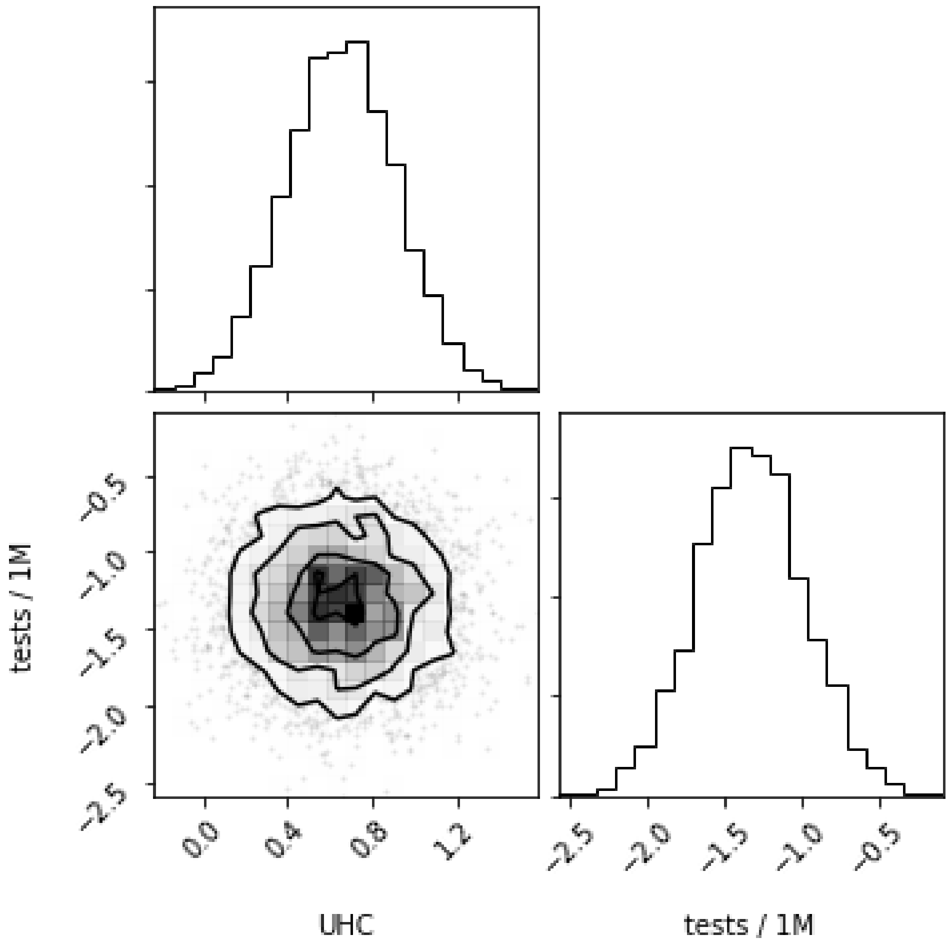
```
                 mean       std    median      5.0%     95.0%     n_eff     r_hat
```

```
     eta1      3.51      4.42      2.08      0.20      7.32    747.07      1.00
lambda[0]     23.25    452.41      1.78      0.01     10.85   1453.65      1.00
lambda[1]      7.00     67.61      1.55      0.01      9.48   4483.16      1.00
lambda[2]    121.94   5907.21      1.35      0.00      7.29   2611.16      1.00
lambda[3]      4.48     27.25      1.19      0.00      6.83   3831.72      1.00
lambda[4]      4.92    103.83      0.44      0.00      3.56   2887.68      1.00
lambda[5]      3.98     41.79      0.63      0.00      4.64   3070.82      1.00
lambda[6]      9.70     88.28      1.52      0.03     10.37   3008.38      1.00
lambda[7]      4.71     27.18      1.39      0.04      7.62   3249.94      1.00
lambda[8]      6.91     53.79      1.23      0.02      6.97   1416.95      1.00
lambda[9]      2.79     63.51      0.13      0.00      1.34   4655.17      1.00
lambda[10]     2.34     18.14      0.41      0.00      2.86   2201.67      1.00
lambda[11]     9.30    177.00      1.39      0.00      7.89   4378.34      1.00
lambda[12]     5.62     84.31      1.21      0.00      6.78   2168.99      1.00
lambda[13]     6.82     64.57      1.51      0.01      8.48   3922.69      1.00
lambda[14]     7.85    186.08      1.40      0.04      8.57   4926.52      1.00
      msq      0.83      0.52      0.70      0.25      1.47   3279.58      1.00
    sigma      7.60      4.16      7.10      0.82     13.53   5977.26      1.00
  var_obs      0.13      0.03      0.13      0.09      0.17   2712.67      1.00
     xisq      0.09      0.06      0.08      0.02      0.16   3062.87      1.00

Number of divergences: 111

MCMC elapsed time: 46.04230809211731
[dimension 01/15]  inactive:    -9.49e-01 +- 5.04e-01
[dimension 02/15]  inactive:    4.41e-01 +- 2.48e-01
[dimension 03/15]  inactive:    3.05e-01 +- 3.40e-01
[dimension 04/15]  inactive:    6.90e-01 +- 4.74e-01
[dimension 05/15]  inactive:    -1.20e-01 +- 3.40e-01
[dimension 06/15]  inactive:    1.09e-01 +- 1.76e-01
[dimension 07/15]  inactive:    1.48e-01 +- 5.16e-01
[dimension 08/15]  active:      6.40e-01 +- 2.64e-01
[dimension 09/15]  inactive:    -1.75e-01 +- 3.08e-01
[dimension 10/15]  inactive:    9.11e-02 +- 1.33e-01
[dimension 11/15]  inactive:    -2.04e-01 +- 1.16e-01
[dimension 12/15]  inactive:    -3.05e-01 +- 2.54e-01
[dimension 13/15]  inactive:    2.72e-01 +- 1.98e-01
[dimension 14/15]  inactive:    7.54e-01 +- 4.50e-01
[dimension 15/15]  active:      -1.16e+00 +- 3.64e-01
Identified a total of 2 active dimensions.
Active dimensions: [7, 14]
```

[25]:
```python
labs = [short_labels[i] for i in all_active_dimensions if i <=␣
 ↪len(short_labels)] + pair_labs
fig = corner.corner(thetas, labels = labs)
fig.show()
```

```
/Users/sachinsmart/opt/anaconda3/lib/python3.7/site-
```

packages/ipykernel_launcher.py:3: UserWarning: Matplotlib is currently using module://ipykernel.pylab.backend_inline, which is a non-GUI backend, so cannot show the figure.
  This is separate from the ipykernel package so we can avoid doing imports until



[26]:
```
print('Active dimensions:', labs)
```

```
Active dimensions: ['UHC', 'tests / 1M']
```

1.8 $\sigma = 1.5$

[27]:
```
all_active_dimensions, thetas, labels, pair_labs = main_modified(X=X, Y=y,␣
↪args=args, sigma=1.5, N_samps=5000,
                                                                     ␣
↪labels=short_labels, **hypers)
```

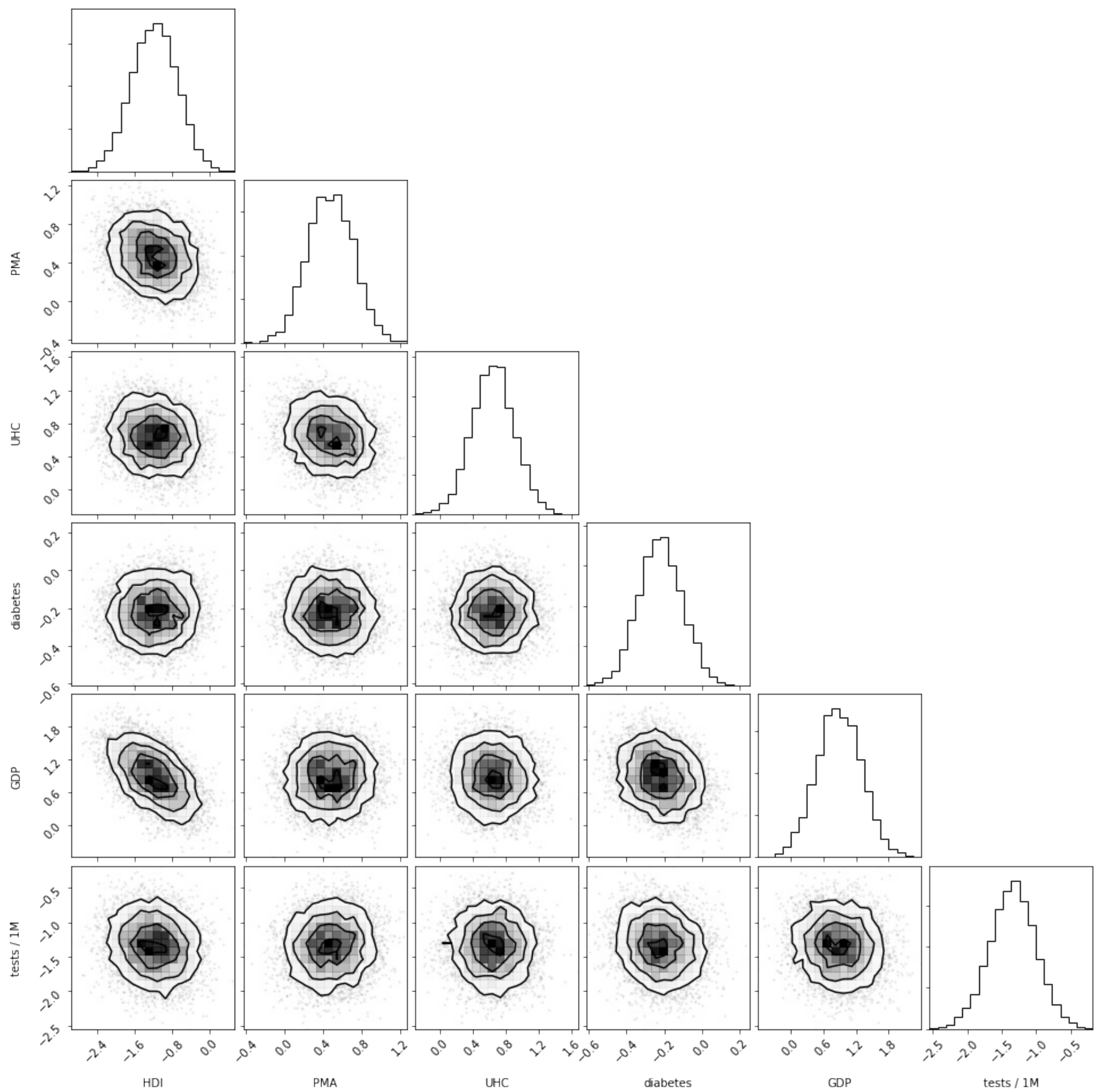
```
sample: 100%|      | 5500/5500 [00:33<00:00, 162.12it/s, 15 steps of size
```

```
2.41e-01. acc. prob=0.90]

                mean       std    median      5.0%     95.0%     n_eff     r_hat
      eta1      3.51      4.42      2.08      0.20      7.32    747.07      1.00
 lambda[0]     23.25    452.41      1.78      0.01     10.85   1453.65      1.00
 lambda[1]      7.00     67.61      1.55      0.01      9.48   4483.16      1.00
 lambda[2]    121.94   5907.21      1.35      0.00      7.29   2611.16      1.00
 lambda[3]      4.48     27.25      1.19      0.00      6.83   3831.72      1.00
 lambda[4]      4.92    103.83      0.44      0.00      3.56   2887.68      1.00
 lambda[5]      3.98     41.79      0.63      0.00      4.64   3070.82      1.00
 lambda[6]      9.70     88.28      1.52      0.03     10.37   3008.38      1.00
 lambda[7]      4.71     27.18      1.39      0.04      7.62   3249.94      1.00
 lambda[8]      6.91     53.79      1.23      0.02      6.97   1416.95      1.00
 lambda[9]      2.79     63.51      0.13      0.00      1.34   4655.17      1.00
lambda[10]      2.34     18.14      0.41      0.00      2.86   2201.67      1.00
lambda[11]      9.30    177.00      1.39      0.00      7.89   4378.34      1.00
lambda[12]      5.62     84.31      1.21      0.00      6.78   2168.99      1.00
lambda[13]      6.82     64.57      1.51      0.01      8.48   3922.69      1.00
lambda[14]      7.85    186.08      1.40      0.04      8.57   4926.52      1.00
       msq      0.83      0.52      0.70      0.25      1.47   3279.58      1.00
     sigma      7.60      4.16      7.10      0.82     13.53   5977.26      1.00
   var_obs      0.13      0.03      0.13      0.09      0.17   2712.67      1.00
      xisq      0.09      0.06      0.08      0.02      0.16   3062.87      1.00

Number of divergences: 111

MCMC elapsed time: 37.93953990936279
[dimension 01/15]  active:      -9.49e-01 +- 5.04e-01
[dimension 02/15]  active:      4.41e-01 +- 2.48e-01
[dimension 03/15]  inactive:    3.05e-01 +- 3.40e-01
[dimension 04/15]  inactive:    6.90e-01 +- 4.74e-01
[dimension 05/15]  inactive:    -1.20e-01 +- 3.40e-01
[dimension 06/15]  inactive:    1.09e-01 +- 1.76e-01
[dimension 07/15]  inactive:    1.48e-01 +- 5.16e-01
[dimension 08/15]  active:      6.40e-01 +- 2.64e-01
[dimension 09/15]  inactive:    -1.75e-01 +- 3.08e-01
[dimension 10/15]  inactive:    9.11e-02 +- 1.33e-01
[dimension 11/15]  active:      -2.04e-01 +- 1.16e-01
[dimension 12/15]  inactive:    -3.05e-01 +- 2.54e-01
[dimension 13/15]  inactive:    2.72e-01 +- 1.98e-01
[dimension 14/15]  active:      7.54e-01 +- 4.50e-01
[dimension 15/15]  active:      -1.16e+00 +- 3.64e-01
Identified a total of 6 active dimensions.
Active dimensions: [0, 1, 7, 10, 13, 14]
```

[28]:
```
labs = [short_labels[i] for i in all_active_dimensions if i <=␣
↪len(short_labels)] + pair_labs
```

```
fig = corner.corner(thetas, labels = labs)
fig.show()
```

```
/Users/sachinsmart/opt/anaconda3/lib/python3.7/site-
packages/ipykernel_launcher.py:3: UserWarning: Matplotlib is currently using
module://ipykernel.pylab.backend_inline, which is a non-GUI backend, so cannot
show the figure.
  This is separate from the ipykernel package so we can avoid doing imports
until
```



[29]:
```
print('Active dimensions:', labs)
```

```
Active dimensions: ['HDI', 'PMA', 'UHC', 'diabetes', 'GDP', 'tests / 1M']
```

1.9 $\sigma = 1.0$

[30]:
```
all_active_dimensions, thetas, labels, pair_labs = main_modified(X=X, Y=y,␣
↪args=args, sigma=1.0, N_samps=5000,                                                    ␣
↪labels=short_labels, **hypers)
```

```
sample: 100%|      | 5500/5500 [00:34<00:00, 160.12it/s, 15 steps of size
2.41e-01. acc. prob=0.90]
```

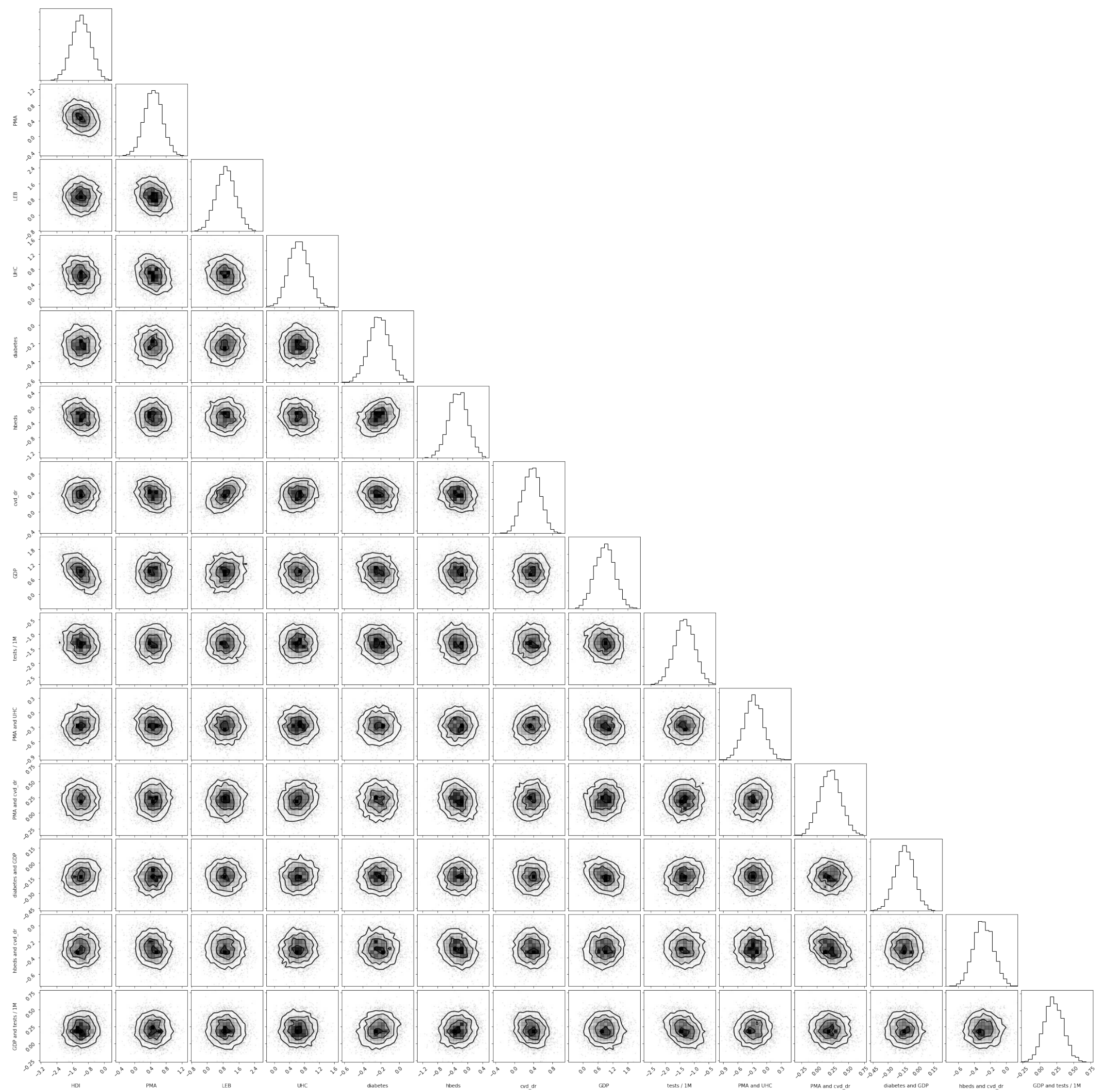
```
                 mean       std    median      5.0%     95.0%     n_eff     r_hat
       eta1      3.51      4.42      2.08      0.20      7.32    747.07      1.00
  lambda[0]     23.25    452.41      1.78      0.01     10.85   1453.65      1.00
  lambda[1]      7.00     67.61      1.55      0.01      9.48   4483.16      1.00
  lambda[2]    121.94   5907.21      1.35      0.00      7.29   2611.16      1.00
  lambda[3]      4.48     27.25      1.19      0.00      6.83   3831.72      1.00
  lambda[4]      4.92    103.83      0.44      0.00      3.56   2887.68      1.00
  lambda[5]      3.98     41.79      0.63      0.00      4.64   3070.82      1.00
  lambda[6]      9.70     88.28      1.52      0.03     10.37   3008.38      1.00
  lambda[7]      4.71     27.18      1.39      0.04      7.62   3249.94      1.00
  lambda[8]      6.91     53.79      1.23      0.02      6.97   1416.95      1.00
  lambda[9]      2.79     63.51      0.13      0.00      1.34   4655.17      1.00
 lambda[10]      2.34     18.14      0.41      0.00      2.86   2201.67      1.00
 lambda[11]      9.30    177.00      1.39      0.00      7.89   4378.34      1.00
 lambda[12]      5.62     84.31      1.21      0.00      6.78   2168.99      1.00
 lambda[13]      6.82     64.57      1.51      0.01      8.48   3922.69      1.00
 lambda[14]      7.85    186.08      1.40      0.04      8.57   4926.52      1.00
        msq      0.83      0.52      0.70      0.25      1.47   3279.58      1.00
      sigma      7.60      4.16      7.10      0.82     13.53   5977.26      1.00
    var_obs      0.13      0.03      0.13      0.09      0.17   2712.67      1.00
       xisq      0.09      0.06      0.08      0.02      0.16   3062.87      1.00

Number of divergences: 111

MCMC elapsed time: 38.34210777282715
[dimension 01/15]  active:      -9.49e-01 +- 5.04e-01
[dimension 02/15]  active:      4.41e-01 +- 2.48e-01
[dimension 03/15]  inactive:    3.05e-01 +- 3.40e-01
[dimension 04/15]  active:      6.90e-01 +- 4.74e-01
[dimension 05/15]  inactive:    -1.20e-01 +- 3.40e-01
[dimension 06/15]  inactive:    1.09e-01 +- 1.76e-01
[dimension 07/15]  inactive:    1.48e-01 +- 5.16e-01
[dimension 08/15]  active:      6.40e-01 +- 2.64e-01
[dimension 09/15]  inactive:    -1.75e-01 +- 3.08e-01
[dimension 10/15]  inactive:    9.11e-02 +- 1.33e-01
[dimension 11/15]  active:      -2.04e-01 +- 1.16e-01
[dimension 12/15]  active:      -3.05e-01 +- 2.54e-01
```

```
[dimension 13/15]  active:      2.72e-01 +- 1.98e-01
[dimension 14/15]  active:      7.54e-01 +- 4.50e-01
[dimension 15/15]  active:      -1.16e+00 +- 3.64e-01
Identified a total of 9 active dimensions.
Identified pairwise interaction between dimensions 2 and 8: -2.83e-01 +-
2.35e-01
Identified pairwise interaction between dimensions 2 and 13: 2.46e-01 +-
2.16e-01
Identified pairwise interaction between dimensions 11 and 14: -1.68e-01 +-
1.45e-01
Identified pairwise interaction between dimensions 12 and 13: -3.30e-01 +-
2.08e-01
Identified pairwise interaction between dimensions 14 and 15: 1.68e-01 +-
1.60e-01
Active dimensions: [0, 1, 3, 7, 10, 11, 12, 13, 14]
```

[31]:

```python
labs = [short_labels[i] for i in all_active_dimensions if i <=␣
 ↪len(short_labels)] + pair_labs
fig = corner.corner(thetas, labels = labs)
fig.show()
```

```
/Users/sachinsmart/opt/anaconda3/lib/python3.7/site-
packages/ipykernel_launcher.py:3: UserWarning: Matplotlib is currently using
module://ipykernel.pylab.backend_inline, which is a non-GUI backend, so cannot
show the figure.
  This is separate from the ipykernel package so we can avoid doing imports
until
```



[32]:
```
print('Active dimensions:', labs)
```

```
Active dimensions: ['HDI', 'PMA', 'LEB', 'UHC', 'diabetes', 'hbeds', 'cvd_dr',
'GDP', 'tests / 1M', 'PMA and UHC', 'PMA and cvd_dr', 'diabetes and GDP', 'hbeds
and cvd_dr', 'GDP and tests / 1M']
```

1.9.1 Conclusion

From our exploratory COVID-19 study, we were able to recover several active factors in our assembled dataset that help explain case fatality in different countries around the world. Making use of our $\pm 1.5\sigma$ bound in the last plot, we see that intuitive factors, such as Human Development Index (HDI) and diabetes prevalence contribute to higher fatality.

With the $\pm 1\sigma$ bound we identify 5 pairwise interactions and 9 singular interactions. What is interesting about this result is that we identify very frequent (2) interaction between the country's GDP per captia and other effects, hinting at an underlying explanation for otherwise poor health in those countries.

When we restrict our active dimension contour range to $\pm 2\sigma$, we recover just two active dimensions: the UHC health service coverage score and the availability of tests per 1 million people in the population. For a more comprehensive study, our data could be further weighted by the availability of tests, such that we do not recover biased fatality estimates from countries with more readily-available testing (unlike the US).

1.9.2 Bonus challenge (2pts extra):

Find another application (e.g. from your area of research) where the kernel-interaction method is directly applicable, or could be applied with some modification. Describe the application for a statistically knowledgeable but non-expert audience (think: your peers in SML 515). In particular, explain why the sparse interaction ansatz is justified. Then demonstrate the use with a suitable data set of your own choice.

Note: If your description is convincing, but you don't find any data or it doesn't lead to conclusive results, partial points will be awarded. Make sure that you have permission to use the data and include it as separate file in your submission.

1.10 Bonus Challenge Response

Another area where the SKIM is applicable is in volatility forecasting for financial time series. In finance, volatility is generally understood to mean a measure of the variation of the price of an asset through time. Higher variation means higher volatility. It is an important quantity for pricing models and risk management, so the ability to accurately forecast future volatility is a useful task.

To that end, volatilities of different assets can often be correlated and volatility time series themselves generally display high levels of autocorrelation. This last part means that the correlation is high between the current value in a volatility time series and past values. Taken together, these features mean that there is predictive power in the autoregressive lags and in the cross-section of volatilities of financial assets.

Thus, the task in this section is to forecast one-day-ahead volatility of the exchange rate between euro (EUR) and British pound sterling (GBP): EURGBP. The regressors for this problem will lags of realised volatility and implied volatility of 16 different currency pairs. Realised volatility is the actual volatility of the past days' time series of the exchange rates. Implied volatility is a value obtained from options markets. Without getting bogged down in the details, options are contracts that give the holders the right to buy or sell an asset at a specified price–they provide certainty. Their prices are related to expectations of future volatility by a famous relationship called the Black-Scholes equation, which is essentially a reformulation of the classical drift-diffusion PDEs in physics. For the purposes here, we can understand implied volatility to be the expected future volatility implied by option prices in the market. Thus, it is a useful regressor for our problem of forecasting future volatility.

We can see the target variable in the first column and the set of regressors in the rest of the columns of the below data frame.

[33]:
```
# Read in the data
fxdata = pd.read_csv('fxdata.csv', header=0)

# Subset the rows and columns
df = fxdata.copy()
df = df.iloc[-64:,:]
drop_cols = list(df.columns[df.columns.str.contains('Dummy')])
drop_cols = drop_cols + list(df.columns[df.columns.str.contains('LogReturn')])
drop_cols = drop_cols + list(df.columns[df.columns.str.contains('Spot')])
df.drop(drop_cols, axis=1, inplace=True)

# Set the target variable and shift it ahead by one day
target_col = 'EURGBP_log_RealVol'
df['Target_' + target_col] = df[target_col].shift(1).copy()
cols = df.columns.tolist()
cols.insert(0, cols.pop(cols.index('Target_' + target_col)))
df = df[cols].set_index('Date').dropna()

# Standardise each column
df[df.columns] = StandardScaler().fit_transform(df)
```

[34]:
```
df.head()
```

[34]:
```
            Target_EURGBP_log_RealVol  AUDJPY_log_ImplVol1d  \
Date
2019-12-31                  -0.382606             -1.113089
2020-01-02                  -0.210221             -0.984138
2020-01-03                  -0.445465             -1.366099
2020-01-06                  -0.530039             -0.567261
2020-01-07                  -0.830431             -0.748562

            AUDJPY_log_ImplVol1d_HAR22  AUDJPY_log_ImplVol1d_HAR5  \
Date
2019-12-31                   -0.857241                  -1.390873
2020-01-02                   -0.823987                  -1.193571
2020-01-03                   -0.866473                  -1.247583
2020-01-06                   -0.863042                  -0.984663
2020-01-07                   -0.869974                  -0.943429

            AUDJPY_log_ImplVol1m  AUDJPY_log_ImplVol1m_HAR22  \
Date
2019-12-31             -0.807958                   -0.843128
2020-01-02             -0.811705                   -0.825481
2020-01-03             -0.702246                   -0.804847
2020-01-06             -0.623899                   -0.788618
```

```
2020-01-07             -0.706986                   -0.782303

            AUDJPY_log_ImplVol1m_HAR5  AUDJPY_log_RealVol  \
Date
2019-12-31                  -0.941674           -1.359974
2020-01-02                  -0.879132           -1.123296
2020-01-03                  -0.800335           -0.443139
2020-01-06                  -0.720086           -0.718948
2020-01-07                  -0.682762           -0.624565

            AUDJPY_log_RealVol_HAR22  AUDJPY_log_RealVol_HAR5  …  \
Date                                                           …
2019-12-31                 -1.043436                -1.518471  …
2020-01-02                 -1.027466                -1.416277  …
2020-01-03                 -0.976796                -1.053005  …
2020-01-06                 -0.992457                -0.950720  …
2020-01-07                 -0.980799                -0.820193  …

            USDCHF_log_RealVol_HAR5  USDJPY_log_ImplVol1d  \
Date
2019-12-31                -0.972054             -0.897603
2020-01-02                -0.663547             -0.693473
2020-01-03                -0.422177             -0.933149
2020-01-06                -0.443892             -0.496679
2020-01-07                -0.460334             -0.699712

            USDJPY_log_ImplVol1d_HAR22  USDJPY_log_ImplVol1d_HAR5  \
Date
2019-12-31                   -0.796031                  -1.299435
2020-01-02                   -0.756117                  -1.053805
2020-01-03                   -0.769495                  -1.000837
2020-01-06                   -0.773760                  -0.769779
2020-01-07                   -0.773667                  -0.715646

            USDJPY_log_ImplVol1m  USDJPY_log_ImplVol1m_HAR22  \
Date
2019-12-31             -0.627827                   -0.758041
2020-01-02             -0.762690                   -0.745066
2020-01-03             -0.582873                   -0.729509
2020-01-06             -0.656583                   -0.735486
2020-01-07             -0.745540                   -0.742096

            USDJPY_log_ImplVol1m_HAR5  USDJPY_log_RealVol  \
Date
2019-12-31                  -0.916504           -1.135485
2020-01-02                  -0.825872           -0.748236
2020-01-03                  -0.714918           -0.071412
```

```
2020-01-06                    -0.646684          -0.657682
2020-01-07                    -0.630552          -0.902153

            USDJPY_log_RealVol_HAR22  USDJPY_log_RealVol_HAR5
Date
2019-12-31                   -0.941159                -1.437344
2020-01-02                   -0.907765                -1.182811
2020-01-03                   -0.863337                -0.786875
2020-01-06                   -0.868349                -0.656093
2020-01-07                   -0.892822                -0.649630

[5 rows x 145 columns]
```

Given there are 145 regressors here, it is unlikely that all are relevant for this forecasting problem. Furthermore, since all the observations are concurrent (each row corresponds to that regressors value at 17:00 ET on that date), there are clearly potential interactions between the regressors, because financial markets (especially currency markets) usually move together. If markets are generally volatile on a given day, then we expect all the regressors' values to be higher. Therefore, it seems reasonable to use SKIM with this dataset and see if we obtain any interesting results. This is, of course, not an out-of-sample testing procedure below, but it is still interesting to see if there are any insights about the most relevant regressors or interactions.

[35]:

```
X = df.iloc[:,1:].to_numpy(copy=True)
y = df.iloc[:,0].to_numpy(copy=True)
short_labels = df.columns.tolist()[1:]

# Optimised hyperparameters
hypers = {'alpha1': 3.122537300650637,
          'alpha2': 34.09492602758983,
          'alpha3': 18.300149226065756,
          'alpha_obs': 47.62800372787773,
          'beta1': 13.485881358212492,
          'beta2': 0.10054579153997745,
          'beta_obs': 0.16307865726277268,
          'c': 0.5174410338358931}

# Numpyro args
n_data, n_dimensions = X.shape

parser = argparse.ArgumentParser(description="Gaussian Process example")
parser.add_argument("-n", "--num-samples", nargs="?", default=5000, type=int)
parser.add_argument("--num-warmup", nargs='?', default=500, type=int)
parser.add_argument("--num-chains", nargs='?', default=1, type=int)
parser.add_argument("--num-data", nargs='?', default=n_data, type=int)
parser.add_argument("--num-dimensions", nargs='?', default=n_dimensions, type=int)
parser.add_argument("--active-dimensions", nargs='?', default=3, type=int)
```

```
parser.add_argument("--device", default='cpu', type=str, help='use "cpu" or␣
↪"gpu".')
args = parser.parse_args(args=[])

numpyro.set_platform(args.device)
numpyro.set_host_device_count(args.num_chains)
```

1.11 $\sigma = 2$

[36]:
```
all_active_dimensions, thetas, labels, pair_labs = main_modified(X=X, Y=y,␣
↪args=args, sigma=2.0, N_samps=5000,                                          ␣
↪labels=short_labels, **hypers)
```

```
sample: 100%|     | 5500/5500 [00:57<00:00, 95.80it/s, 31 steps of size
2.12e-01. acc. prob=0.90]


                 mean       std    median      5.0%     95.0%     n_eff
r_hat
       eta1      0.30      0.08      0.29      0.18      0.42   2621.14
1.00
  lambda[0]      6.04    104.00      1.02      0.00      5.63   2468.33
1.00
  lambda[1]      8.29    133.87      1.01      0.00      6.68   2471.60
1.00
  lambda[2]      4.86     28.23      1.05      0.00      6.69   3196.49
1.00
  lambda[3]    231.21  10434.69      8.06      0.00     48.10   4415.06
1.00
  lambda[4]      3.39     16.13      0.96      0.00      5.47   4469.30
1.00
  lambda[5]      4.86     54.22      0.92      0.00      4.90   3251.43
1.00
  lambda[6]     11.21    129.33      1.10      0.00     10.49   3385.64
1.00
  lambda[7]      7.77     55.05      1.29      0.00     11.73   3761.46
1.00
  lambda[8]      2.31     10.67      0.79      0.00      3.94   3106.93
1.00
  lambda[9]      9.27    118.05      1.31      0.00      6.32   1389.63
1.00
 lambda[10]      6.69     69.98      1.09      0.00      7.50   2451.16
1.00
 lambda[11]      4.66     39.83      0.93      0.00      5.21   2923.57
1.00
 lambda[12]     99.43   6702.91      0.83      0.00      4.07   4776.75
```

```
1.00
 lambda[13]     10.42    291.51       0.93       0.00       5.36   4626.91
1.00
 lambda[14]     12.16    324.05       0.92       0.00       5.15   2831.73
1.00
 lambda[15]     10.74     64.67       2.86       0.00      15.20   2781.14
1.00
 lambda[16]     15.64    713.70       1.01       0.00       6.46   4967.20
1.00
 lambda[17]      4.08    103.67       0.84       0.00       4.11   4627.15
1.00
 lambda[18]      5.25     82.81       0.70       0.00       3.70   2067.15
1.00
 lambda[19]      4.91     36.07       1.03       0.00       6.62   4503.21
1.00
 lambda[20]      2.96     17.76       0.85       0.00       4.17   3243.98
1.00
 lambda[21]      8.73     55.74       1.66       0.00      12.65   3958.22
1.00
 lambda[22]      6.60    107.94       1.02       0.00       6.84   2926.78
1.00
 lambda[23]      4.51     48.91       0.92       0.00       4.79   3339.81
1.00
 lambda[24]     14.52    272.58       0.68       0.00       4.22    845.54
1.00
 lambda[25]      5.98     51.46       0.96       0.00       5.92   2673.33
1.00
 lambda[26]      7.91     81.21       1.18       0.00       6.36   2133.23
1.00
 lambda[27]      5.18    115.94       0.84       0.00       5.09   4618.43
1.00
 lambda[28]      5.15     85.06       0.94       0.00       5.23   4947.28
1.00
 lambda[29]      5.58    101.19       0.86       0.00       4.69   2760.46
1.00
 lambda[30]      2.91     20.37       0.86       0.00       4.01   4009.23
1.00
 lambda[31]      4.75     59.01       0.89       0.00       5.35   2442.37
1.00
 lambda[32]      8.50    107.60       1.03       0.00       7.03   3147.09
1.00
 lambda[33]      3.06     16.79       0.79       0.00       4.37   3714.25
1.00
 lambda[34]      4.23     17.86       1.04       0.00       7.03   2961.12
1.00
 lambda[35]      7.63    197.73       0.90       0.00       4.67   2190.72
1.00
 lambda[36]      5.42     45.19       1.47       0.00       6.27   2238.18
```

```
1.00
 lambda[37]      8.78     94.22       1.25       0.00       9.81   3238.08
1.00
 lambda[38]     19.93    994.20       0.92       0.00       5.74   4193.64
1.00
 lambda[39]      8.88    107.36       1.43       0.00      10.26   2632.32
1.00
 lambda[40]     11.28    333.57       0.97       0.00       6.29   4508.75
1.00
 lambda[41]      5.02     49.31       0.93       0.00       5.88   3742.57
1.00
 lambda[42]      5.96     64.75       1.54       0.00       6.25   2264.62
1.00
 lambda[43]     10.73    374.18       0.98       0.00       6.12   3677.48
1.00
 lambda[44]      2.59     13.48       0.80       0.00       3.95   2539.23
1.00
 lambda[45]      3.06     13.54       1.08       0.00       4.45   2295.94
1.00
 lambda[46]      5.69     59.58       0.91       0.00       5.23   3444.22
1.00
 lambda[47]      9.18    294.29       0.97       0.00       5.49   3084.87
1.00
 lambda[48]      2.56     15.41       0.72       0.00       3.36   2199.28
1.00
 lambda[49]      3.77     24.32       0.92       0.00       5.39   4128.45
1.00
 lambda[50]      3.35     39.87       0.83       0.00       3.85   4634.50
1.00
 lambda[51]      3.19     82.98       0.52       0.00       2.20   4326.72
1.00
 lambda[52]      2.96     16.70       0.85       0.00       4.23   4136.74
1.00
 lambda[53]      3.51     24.61       0.93       0.00       4.51   2931.06
1.00
 lambda[54]      3.25     48.92       0.69       0.00       3.07   4085.80
1.00
 lambda[55]     10.94    211.25       1.09       0.00       7.04   2962.46
1.00
 lambda[56]      8.18    125.28       1.32       0.00       7.96   3666.57
1.00
 lambda[57]      4.06     60.78       0.78       0.00       3.66   2641.05
1.00
 lambda[58]      3.59     20.78       0.90       0.00       5.38   3200.75
1.00
 lambda[59]      6.38     95.45       1.14       0.00       7.53   3821.95
1.00
 lambda[60]      9.03     57.08       1.59       0.00      12.58   4054.11
```

```
1.00
 lambda[61]      6.50    159.52      0.88      0.00      4.69   3980.74
1.00
 lambda[62]      6.14     74.91      1.24      0.00      8.05   4812.84
1.00
 lambda[63]      5.60     43.87      1.34      0.00      7.31   4273.41
1.00
 lambda[64]      3.82     23.39      0.93      0.00      5.31   3763.84
1.00
 lambda[65]      8.03     45.30      1.66      0.00     12.48   3689.49
1.00
 lambda[66]     33.66    341.82      6.29      0.01     38.04   4171.97
1.00
 lambda[67]      6.56    130.46      0.98      0.00      5.91   2873.47
1.00
 lambda[68]      3.77     29.56      0.96      0.00      5.43   4319.67
1.00
 lambda[69]     14.79     92.03      2.98      0.00     21.94   3463.14
1.00
 lambda[70]      4.05     22.56      0.96      0.00      5.65   3255.57
1.00
 lambda[71]      5.20     52.94      1.00      0.00      5.52   4694.17
1.00
 lambda[72]      3.00     46.84      0.68      0.00      2.99   1695.25
1.00
 lambda[73]      4.65     47.20      0.97      0.00      6.06   2468.02
1.00
 lambda[74]      4.04     41.59      0.87      0.00      4.67   2901.75
1.00
 lambda[75]      5.36     53.58      1.25      0.00      8.24   4768.98
1.00
 lambda[76]    144.58   7044.54      0.93      0.00      5.52   2504.06
1.00
 lambda[77]      8.01    257.75      0.89      0.00      4.89   3324.47
1.00
 lambda[78]      3.86     33.64      1.04      0.00      4.44   2152.35
1.00
 lambda[79]      4.93     61.37      0.93      0.00      4.96   3959.20
1.00
 lambda[80]     12.71    564.33      0.89      0.00      5.16   4667.66
1.00
 lambda[81]      1.53     11.34      0.53      0.00      2.05   1904.02
1.00
 lambda[82]      4.20     25.02      0.99      0.00      5.83   1771.91
1.00
 lambda[83]      3.32     40.98      0.86      0.00      4.32   4704.95
1.00
 lambda[84]      2.89     20.36      0.81      0.00      3.81   3544.81
```

```
1.00
 lambda[85]      4.53      33.26        0.96        0.00        5.38   2383.15
1.00
 lambda[86]      6.89      96.38        0.89        0.00        4.86   2033.18
1.00
 lambda[87]      4.59      46.73        0.94        0.00        4.65   2995.20
1.00
 lambda[88]      2.33      11.67        0.84        0.00        3.84   3076.48
1.00
 lambda[89]     25.54     431.68        2.81        0.00       19.28   1720.86
1.00
 lambda[90]      9.13      52.84        1.94        0.00       11.89   3092.57
1.00
 lambda[91]      3.89      23.86        0.92        0.00        5.07   3762.97
1.00
 lambda[92]      3.63      30.01        0.91        0.00        4.67   4142.69
1.00
 lambda[93]      5.04      86.94        0.90        0.00        5.10   3366.06
1.00
 lambda[94]      5.22      55.11        0.89        0.00        5.10   2610.42
1.00
 lambda[95]      5.96      57.52        1.00        0.00        6.12   2812.28
1.00
 lambda[96]      3.42      28.93        0.80        0.00        4.61   4460.67
1.00
 lambda[97]      3.26      13.70        0.93        0.00        4.95   2928.97
1.00
 lambda[98]      3.64      27.93        0.87        0.00        4.27   3076.66
1.00
 lambda[99]      2.24       7.42        0.94        0.00        3.78   3448.23
1.00
lambda[100]      5.97      68.08        0.99        0.00        5.89   4376.18
1.00
lambda[101]      3.11      18.44        0.91        0.00        4.54   4078.13
1.00
lambda[102]      2.57      10.72        0.77        0.00        4.18   2362.98
1.00
lambda[103]     55.76    3563.30        0.92        0.00        4.95   4998.61
1.00
lambda[104]     11.76     273.28        1.00        0.00        5.72   2738.03
1.00
lambda[105]      3.34      56.02        0.69        0.00        2.92   2876.43
1.00
lambda[106]      2.95      19.60        0.83        0.00        3.92   1747.99
1.00
lambda[107]      4.97     138.61        0.87        0.00        4.40   4707.94
1.00
lambda[108]      2.36      20.99        0.76        0.00        3.12   2724.90
```

```
1.00
lambda[109]     11.95   388.38      0.91      0.00      4.94   3929.63
1.00
lambda[110]      7.62    64.73      1.17      0.00      8.42   4587.14
1.00
lambda[111]      5.98   115.92      0.89      0.00      4.94   2929.36
1.00
lambda[112]      2.69    10.49      0.93      0.00      4.67   2409.22
1.00
lambda[113]      3.85    22.44      0.89      0.00      4.77   2353.81
1.00
lambda[114]      4.08    43.68      0.92      0.00      4.86   2640.50
1.00
lambda[115]     40.69   709.33      3.13      0.00     26.82   1680.97
1.00
lambda[116]      7.02   164.14      1.04      0.00      5.72   4193.98
1.00
lambda[117]     28.22   830.43      1.70      0.00      7.24   2762.60
1.00
lambda[118]      5.94    45.77      1.17      0.00      8.14   3728.15
1.00
lambda[119]      3.49    46.25      0.84      0.00      4.03   4657.94
1.00
lambda[120]      4.21    69.82      0.79      0.00      4.04   4729.89
1.00
lambda[121]      5.01    39.45      0.96      0.00      5.48   3081.68
1.00
lambda[122]      4.65    58.70      0.92      0.00      4.82   3008.45
1.00
lambda[123]      1.27     4.87      0.55      0.00      2.22   3185.29
1.00
lambda[124]      6.98   178.54      0.92      0.00      5.34   2277.25
1.00
lambda[125]      5.01    72.84      0.85      0.00      4.20   3958.85
1.00
lambda[126]      3.73    26.25      1.21      0.00      4.90   3434.31
1.00
lambda[127]      6.17    54.81      1.07      0.00      6.96   2576.49
1.00
lambda[128]      8.40    63.40      1.61      0.00     10.40   3474.33
1.00
lambda[129]      5.21    70.17      0.82      0.00      4.63   3751.98
1.00
lambda[130]      7.88   158.37      0.98      0.00      5.88   2641.09
1.00
lambda[131]     11.16   159.62      1.20      0.00      7.98   3292.41
1.00
lambda[132]      2.29    28.85      0.65      0.00      2.79   4618.07
```

```
1.00
lambda[133]      6.00     86.21      0.93      0.00      5.01   4384.16
1.00
lambda[134]      6.08    139.63      0.79      0.00      4.24   2741.65
1.00
lambda[135]      3.54     15.18      1.21      0.00      5.69   3226.24
1.00
lambda[136]      4.21     38.12      0.96      0.00      6.08   4008.45
1.00
lambda[137]      3.19     40.60      0.83      0.00      4.23   4629.70
1.00
lambda[138]     14.81    489.74      1.01      0.00      7.49   3271.40
1.00
lambda[139]      3.40     14.52      0.93      0.00      5.61   2300.35
1.00
lambda[140]      5.56     67.99      0.91      0.00      5.23   1941.85
1.00
lambda[141]      2.29     26.19      0.59      0.00      2.83   4196.48
1.00
lambda[142]      9.93    136.14      1.26      0.00     10.35   3596.78
1.00
lambda[143]      3.51     32.35      0.80      0.00      3.86   2042.01
1.00
        msq      2.40      1.15      2.15      0.98      3.82   2021.55
1.00
      sigma      9.24      8.09      6.83      0.30     20.41   6204.54
1.00
    var_obs      0.00      0.00      0.00      0.00      0.00   5536.80
1.00
       xisq      0.00      0.00      0.00      0.00      0.00   4909.30
1.00

Number of divergences: 0

MCMC elapsed time: 64.59255814552307
[dimension 01/144]  inactive:   -2.07e-01 +- 3.10e-01
[dimension 02/144]  inactive:   -1.32e-01 +- 5.51e-01
[dimension 03/144]  inactive:   2.34e-01 +- 5.36e-01
[dimension 04/144]  active:     1.94e+00 +- 9.22e-01
[dimension 05/144]  inactive:   -1.09e-02 +- 5.29e-01
[dimension 06/144]  inactive:   -2.26e-02 +- 4.78e-01
[dimension 07/144]  inactive:   -1.58e-02 +- 2.29e-01
[dimension 08/144]  inactive:   4.04e-01 +- 7.74e-01
[dimension 09/144]  inactive:   -5.74e-02 +- 3.38e-01
[dimension 10/144]  inactive:   -3.11e-01 +- 3.10e-01
[dimension 11/144]  inactive:   -2.35e-01 +- 6.14e-01
[dimension 12/144]  inactive:   -1.44e-01 +- 4.56e-01
[dimension 13/144]  inactive:   -1.68e-02 +- 3.78e-01
```

```
[dimension 14/144]  inactive:   -2.88e-02 +- 5.07e-01
[dimension 15/144]  inactive:   -4.95e-02 +- 4.76e-01
[dimension 16/144]  inactive:   -5.90e-01 +- 3.23e-01
[dimension 17/144]  inactive:   1.15e-01 +- 5.68e-01
[dimension 18/144]  inactive:   -3.91e-02 +- 3.65e-01
[dimension 19/144]  inactive:   -5.05e-02 +- 2.08e-01
[dimension 20/144]  inactive:   -2.11e-01 +- 5.67e-01
[dimension 21/144]  inactive:   -8.56e-02 +- 3.54e-01
[dimension 22/144]  inactive:   -3.75e-01 +- 6.15e-01
[dimension 23/144]  inactive:   -1.46e-01 +- 6.16e-01
[dimension 24/144]  inactive:   -5.76e-02 +- 4.50e-01
[dimension 25/144]  inactive:   -5.20e-02 +- 1.62e-01
[dimension 26/144]  inactive:   1.44e-01 +- 5.32e-01
[dimension 27/144]  inactive:   -3.05e-01 +- 4.58e-01
[dimension 28/144]  inactive:   6.87e-02 +- 2.67e-01
[dimension 29/144]  inactive:   4.14e-02 +- 4.97e-01
[dimension 30/144]  inactive:   6.98e-02 +- 3.81e-01
[dimension 31/144]  inactive:   -2.90e-03 +- 3.97e-01
[dimension 32/144]  inactive:   -2.61e-02 +- 4.89e-01
[dimension 33/144]  inactive:   -1.89e-01 +- 5.71e-01
[dimension 34/144]  inactive:   -2.77e-02 +- 2.53e-01
[dimension 35/144]  inactive:   1.54e-01 +- 5.90e-01
[dimension 36/144]  inactive:   1.55e-01 +- 3.84e-01
[dimension 37/144]  inactive:   3.72e-01 +- 2.41e-01
[dimension 38/144]  inactive:   -3.90e-01 +- 7.54e-01
[dimension 39/144]  inactive:   -1.42e-01 +- 4.35e-01
[dimension 40/144]  inactive:   4.12e-01 +- 5.74e-01
[dimension 41/144]  inactive:   1.28e-01 +- 5.60e-01
[dimension 42/144]  inactive:   -1.56e-02 +- 4.75e-01
[dimension 43/144]  inactive:   -3.82e-01 +- 2.47e-01
[dimension 44/144]  inactive:   -1.31e-01 +- 5.37e-01
[dimension 45/144]  inactive:   2.86e-02 +- 3.09e-01
[dimension 46/144]  inactive:   2.33e-01 +- 1.68e-01
[dimension 47/144]  inactive:   -6.98e-02 +- 4.70e-01
[dimension 48/144]  inactive:   1.65e-01 +- 3.53e-01
[dimension 49/144]  inactive:   -1.11e-02 +- 2.39e-01
[dimension 50/144]  inactive:   -4.85e-02 +- 4.99e-01
[dimension 51/144]  inactive:   1.83e-02 +- 3.62e-01
[dimension 52/144]  inactive:   2.98e-02 +- 1.04e-01
[dimension 53/144]  inactive:   9.33e-02 +- 3.65e-01
[dimension 54/144]  inactive:   1.78e-01 +- 3.09e-01
[dimension 55/144]  inactive:   2.54e-02 +- 1.97e-01
[dimension 56/144]  inactive:   -2.49e-01 +- 4.81e-01
[dimension 57/144]  inactive:   3.51e-01 +- 5.19e-01
[dimension 58/144]  inactive:   8.60e-02 +- 3.16e-01
[dimension 59/144]  inactive:   6.46e-02 +- 4.37e-01
[dimension 60/144]  inactive:   -3.28e-01 +- 6.02e-01
[dimension 61/144]  inactive:   -3.21e-01 +- 3.09e-01
```

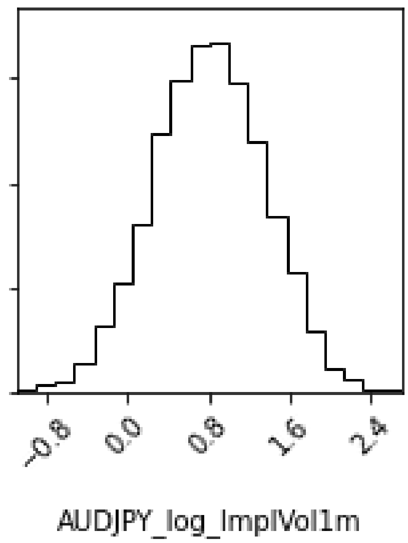
```
[dimension 62/144]  inactive:   -1.11e-01 +- 4.45e-01
[dimension 63/144]  inactive:   3.01e-01 +- 5.21e-01
[dimension 64/144]  inactive:   -3.52e-01 +- 3.91e-01
[dimension 65/144]  inactive:   -3.55e-02 +- 4.93e-01
[dimension 66/144]  inactive:   5.45e-01 +- 6.84e-01
[dimension 67/144]  inactive:   -1.47e+00 +- 9.23e-01
[dimension 68/144]  inactive:   -6.88e-02 +- 5.65e-01
[dimension 69/144]  inactive:   1.25e-01 +- 5.08e-01
[dimension 70/144]  inactive:   4.72e-01 +- 2.87e-01
[dimension 71/144]  inactive:   1.51e-01 +- 5.43e-01
[dimension 72/144]  inactive:   -2.17e-01 +- 4.20e-01
[dimension 73/144]  inactive:   -1.42e-02 +- 1.96e-01
[dimension 74/144]  inactive:   -1.25e-01 +- 5.23e-01
[dimension 75/144]  inactive:   -6.72e-02 +- 3.72e-01
[dimension 76/144]  inactive:   3.29e-01 +- 4.96e-01
[dimension 77/144]  inactive:   9.55e-02 +- 5.33e-01
[dimension 78/144]  inactive:   1.17e-02 +- 4.13e-01
[dimension 79/144]  inactive:   2.43e-01 +- 2.46e-01
[dimension 80/144]  inactive:   -8.94e-02 +- 4.93e-01
[dimension 81/144]  inactive:   -7.27e-02 +- 3.84e-01
[dimension 82/144]  inactive:   1.60e-02 +- 1.40e-01
[dimension 83/144]  inactive:   -1.96e-01 +- 4.15e-01
[dimension 84/144]  inactive:   -1.24e-01 +- 3.68e-01
[dimension 85/144]  inactive:   -3.06e-02 +- 2.85e-01
[dimension 86/144]  inactive:   1.48e-01 +- 4.58e-01
[dimension 87/144]  inactive:   -2.98e-02 +- 4.56e-01
[dimension 88/144]  inactive:   1.92e-01 +- 2.96e-01
[dimension 89/144]  inactive:   3.58e-02 +- 3.86e-01
[dimension 90/144]  inactive:   8.03e-01 +- 7.11e-01
[dimension 91/144]  inactive:   -3.73e-01 +- 2.54e-01
[dimension 92/144]  inactive:   -9.41e-02 +- 4.66e-01
[dimension 93/144]  inactive:   -1.33e-01 +- 4.41e-01
[dimension 94/144]  inactive:   -1.10e-04 +- 3.90e-01
[dimension 95/144]  inactive:   8.67e-02 +- 4.69e-01
[dimension 96/144]  inactive:   -1.59e-01 +- 5.60e-01
[dimension 97/144]  inactive:   -1.16e-01 +- 2.34e-01
[dimension 98/144]  inactive:   1.39e-01 +- 4.66e-01
[dimension 99/144]  inactive:   6.50e-02 +- 3.95e-01
[dimension 100/144]  inactive:  2.01e-01 +- 2.31e-01
[dimension 101/144]  inactive:  -1.63e-01 +- 4.36e-01
[dimension 102/144]  inactive:  1.06e-01 +- 4.01e-01
[dimension 103/144]  inactive:  -1.17e-02 +- 2.94e-01
[dimension 104/144]  inactive:  1.27e-01 +- 4.43e-01
[dimension 105/144]  inactive:  -2.06e-01 +- 5.11e-01
[dimension 106/144]  inactive:  9.90e-02 +- 1.90e-01
[dimension 107/144]  inactive:  -4.32e-02 +- 3.75e-01
[dimension 108/144]  inactive:  9.32e-02 +- 3.80e-01
[dimension 109/144]  inactive:  -1.22e-01 +- 2.00e-01
```

```
[dimension 110/144]  inactive:  -7.66e-02 +- 4.78e-01
[dimension 111/144]  inactive:  3.45e-01 +- 5.73e-01
[dimension 112/144]  inactive:  7.51e-02 +- 4.32e-01
[dimension 113/144]  inactive:  -4.14e-02 +- 4.91e-01
[dimension 114/144]  inactive:  -6.10e-02 +- 4.56e-01
[dimension 115/144]  inactive:  1.83e-01 +- 1.73e-01
[dimension 116/144]  inactive:  1.15e+00 +- 1.27e+00
[dimension 117/144]  inactive:  -2.53e-01 +- 4.23e-01
[dimension 118/144]  inactive:  4.12e-01 +- 2.37e-01
[dimension 119/144]  inactive:  -3.32e-01 +- 6.64e-01
[dimension 120/144]  inactive:  -8.77e-02 +- 3.47e-01
[dimension 121/144]  inactive:  5.06e-02 +- 2.89e-01
[dimension 122/144]  inactive:  9.52e-02 +- 5.16e-01
[dimension 123/144]  inactive:  -5.43e-02 +- 4.46e-01
[dimension 124/144]  inactive:  1.93e-02 +- 1.48e-01
[dimension 125/144]  inactive:  -5.40e-02 +- 4.69e-01
[dimension 126/144]  inactive:  1.56e-01 +- 3.19e-01
[dimension 127/144]  inactive:  -2.89e-01 +- 2.33e-01
[dimension 128/144]  inactive:  -2.20e-01 +- 6.02e-01
[dimension 129/144]  inactive:  -4.87e-01 +- 5.99e-01
[dimension 130/144]  inactive:  -3.89e-02 +- 3.32e-01
[dimension 131/144]  inactive:  1.71e-02 +- 5.49e-01
[dimension 132/144]  inactive:  -3.19e-01 +- 6.22e-01
[dimension 133/144]  inactive:  6.90e-02 +- 1.69e-01
[dimension 134/144]  inactive:  5.07e-02 +- 4.80e-01
[dimension 135/144]  inactive:  -7.53e-02 +- 3.29e-01
[dimension 136/144]  inactive:  2.75e-01 +- 2.66e-01
[dimension 137/144]  inactive:  -4.79e-02 +- 5.43e-01
[dimension 138/144]  inactive:  -2.04e-02 +- 3.72e-01
[dimension 139/144]  inactive:  2.28e-02 +- 4.40e-01
[dimension 140/144]  inactive:  3.23e-03 +- 5.45e-01
[dimension 141/144]  inactive:  7.42e-02 +- 4.70e-01
[dimension 142/144]  inactive:  -4.79e-03 +- 1.49e-01
[dimension 143/144]  inactive:  3.84e-01 +- 7.77e-01
[dimension 144/144]  inactive:  4.72e-03 +- 3.36e-01
Identified a total of 1 active dimensions.
Active dimensions: [3]
```

[37]:
```python
labs = [short_labels[i] for i in all_active_dimensions if i <=␣
↪len(short_labels)] + pair_labs
fig = corner.corner(thetas, labels = labs)
fig.show()
```

```
/Users/sachinsmart/opt/anaconda3/lib/python3.7/site-
packages/ipykernel_launcher.py:3: UserWarning: Matplotlib is currently using
module://ipykernel.pylab.backend_inline, which is a non-GUI backend, so cannot
show the figure.
  This is separate from the ipykernel package so we can avoid doing imports
```

until



[38]:
```
print('Active dimensions:', labs)
```

```
Active dimensions: ['AUDJPY_log_ImplVol1m']
```

1.12 $\sigma = 1.5$

[39]:
```
all_active_dimensions, thetas, labels, pair_labs = main_modified(X=X, Y=y,␣
↪args=args, sigma=1.5, N_samps=5000,                                        ␣
↪labels=short_labels, **hypers)
```

```
sample: 100%|      | 5500/5500 [00:52<00:00, 103.81it/s, 31 steps of size
2.12e-01. acc. prob=0.90]


                    mean       std    median      5.0%     95.0%     n_eff
r_hat
       eta1         0.30      0.08      0.29      0.18      0.42   2621.14
1.00
  lambda[0]         6.04    104.00      1.02      0.00      5.63   2468.33
1.00
  lambda[1]         8.29    133.87      1.01      0.00      6.68   2471.60
1.00
  lambda[2]         4.86     28.23      1.05      0.00      6.69   3196.49
1.00
  lambda[3]       231.21  10434.69      8.06      0.00     48.10   4415.06
1.00
  lambda[4]         3.39     16.13      0.96      0.00      5.47   4469.30
```

```
1.00
  lambda[5]      4.86     54.22       0.92      0.00       4.90   3251.43
1.00
  lambda[6]     11.21    129.33       1.10      0.00      10.49   3385.64
1.00
  lambda[7]      7.77     55.05       1.29      0.00      11.73   3761.46
1.00
  lambda[8]      2.31     10.67       0.79      0.00       3.94   3106.93
1.00
  lambda[9]      9.27    118.05       1.31      0.00       6.32   1389.63
1.00
 lambda[10]      6.69     69.98       1.09      0.00       7.50   2451.16
1.00
 lambda[11]      4.66     39.83       0.93      0.00       5.21   2923.57
1.00
 lambda[12]     99.43   6702.91       0.83      0.00       4.07   4776.75
1.00
 lambda[13]     10.42    291.51       0.93      0.00       5.36   4626.91
1.00
 lambda[14]     12.16    324.05       0.92      0.00       5.15   2831.73
1.00
 lambda[15]     10.74     64.67       2.86      0.00      15.20   2781.14
1.00
 lambda[16]     15.64    713.70       1.01      0.00       6.46   4967.20
1.00
 lambda[17]      4.08    103.67       0.84      0.00       4.11   4627.15
1.00
 lambda[18]      5.25     82.81       0.70      0.00       3.70   2067.15
1.00
 lambda[19]      4.91     36.07       1.03      0.00       6.62   4503.21
1.00
 lambda[20]      2.96     17.76       0.85      0.00       4.17   3243.98
1.00
 lambda[21]      8.73     55.74       1.66      0.00      12.65   3958.22
1.00
 lambda[22]      6.60    107.94       1.02      0.00       6.84   2926.78
1.00
 lambda[23]      4.51     48.91       0.92      0.00       4.79   3339.81
1.00
 lambda[24]     14.52    272.58       0.68      0.00       4.22    845.54
1.00
 lambda[25]      5.98     51.46       0.96      0.00       5.92   2673.33
1.00
 lambda[26]      7.91     81.21       1.18      0.00       6.36   2133.23
1.00
 lambda[27]      5.18    115.94       0.84      0.00       5.09   4618.43
1.00
 lambda[28]      5.15     85.06       0.94      0.00       5.23   4947.28
```

```
1.00
 lambda[29]      5.58    101.19       0.86       0.00       4.69   2760.46
1.00
 lambda[30]      2.91     20.37       0.86       0.00       4.01   4009.23
1.00
 lambda[31]      4.75     59.01       0.89       0.00       5.35   2442.37
1.00
 lambda[32]      8.50    107.60       1.03       0.00       7.03   3147.09
1.00
 lambda[33]      3.06     16.79       0.79       0.00       4.37   3714.25
1.00
 lambda[34]      4.23     17.86       1.04       0.00       7.03   2961.12
1.00
 lambda[35]      7.63    197.73       0.90       0.00       4.67   2190.72
1.00
 lambda[36]      5.42     45.19       1.47       0.00       6.27   2238.18
1.00
 lambda[37]      8.78     94.22       1.25       0.00       9.81   3238.08
1.00
 lambda[38]     19.93    994.20       0.92       0.00       5.74   4193.64
1.00
 lambda[39]      8.88    107.36       1.43       0.00      10.26   2632.32
1.00
 lambda[40]     11.28    333.57       0.97       0.00       6.29   4508.75
1.00
 lambda[41]      5.02     49.31       0.93       0.00       5.88   3742.57
1.00
 lambda[42]      5.96     64.75       1.54       0.00       6.25   2264.62
1.00
 lambda[43]     10.73    374.18       0.98       0.00       6.12   3677.48
1.00
 lambda[44]      2.59     13.48       0.80       0.00       3.95   2539.23
1.00
 lambda[45]      3.06     13.54       1.08       0.00       4.45   2295.94
1.00
 lambda[46]      5.69     59.58       0.91       0.00       5.23   3444.22
1.00
 lambda[47]      9.18    294.29       0.97       0.00       5.49   3084.87
1.00
 lambda[48]      2.56     15.41       0.72       0.00       3.36   2199.28
1.00
 lambda[49]      3.77     24.32       0.92       0.00       5.39   4128.45
1.00
 lambda[50]      3.35     39.87       0.83       0.00       3.85   4634.50
1.00
 lambda[51]      3.19     82.98       0.52       0.00       2.20   4326.72
1.00
 lambda[52]      2.96     16.70       0.85       0.00       4.23   4136.74
```

```
1.00
 lambda[53]      3.51     24.61       0.93       0.00       4.51   2931.06
1.00
 lambda[54]      3.25     48.92       0.69       0.00       3.07   4085.80
1.00
 lambda[55]     10.94    211.25       1.09       0.00       7.04   2962.46
1.00
 lambda[56]      8.18    125.28       1.32       0.00       7.96   3666.57
1.00
 lambda[57]      4.06     60.78       0.78       0.00       3.66   2641.05
1.00
 lambda[58]      3.59     20.78       0.90       0.00       5.38   3200.75
1.00
 lambda[59]      6.38     95.45       1.14       0.00       7.53   3821.95
1.00
 lambda[60]      9.03     57.08       1.59       0.00      12.58   4054.11
1.00
 lambda[61]      6.50    159.52       0.88       0.00       4.69   3980.74
1.00
 lambda[62]      6.14     74.91       1.24       0.00       8.05   4812.84
1.00
 lambda[63]      5.60     43.87       1.34       0.00       7.31   4273.41
1.00
 lambda[64]      3.82     23.39       0.93       0.00       5.31   3763.84
1.00
 lambda[65]      8.03     45.30       1.66       0.00      12.48   3689.49
1.00
 lambda[66]     33.66    341.82       6.29       0.01      38.04   4171.97
1.00
 lambda[67]      6.56    130.46       0.98       0.00       5.91   2873.47
1.00
 lambda[68]      3.77     29.56       0.96       0.00       5.43   4319.67
1.00
 lambda[69]     14.79     92.03       2.98       0.00      21.94   3463.14
1.00
 lambda[70]      4.05     22.56       0.96       0.00       5.65   3255.57
1.00
 lambda[71]      5.20     52.94       1.00       0.00       5.52   4694.17
1.00
 lambda[72]      3.00     46.84       0.68       0.00       2.99   1695.25
1.00
 lambda[73]      4.65     47.20       0.97       0.00       6.06   2468.02
1.00
 lambda[74]      4.04     41.59       0.87       0.00       4.67   2901.75
1.00
 lambda[75]      5.36     53.58       1.25       0.00       8.24   4768.98
1.00
 lambda[76]    144.58   7044.54       0.93       0.00       5.52   2504.06
```

```
1.00
 lambda[77]      8.01    257.75       0.89       0.00       4.89   3324.47
1.00
 lambda[78]      3.86     33.64       1.04       0.00       4.44   2152.35
1.00
 lambda[79]      4.93     61.37       0.93       0.00       4.96   3959.20
1.00
 lambda[80]     12.71    564.33       0.89       0.00       5.16   4667.66
1.00
 lambda[81]      1.53     11.34       0.53       0.00       2.05   1904.02
1.00
 lambda[82]      4.20     25.02       0.99       0.00       5.83   1771.91
1.00
 lambda[83]      3.32     40.98       0.86       0.00       4.32   4704.95
1.00
 lambda[84]      2.89     20.36       0.81       0.00       3.81   3544.81
1.00
 lambda[85]      4.53     33.26       0.96       0.00       5.38   2383.15
1.00
 lambda[86]      6.89     96.38       0.89       0.00       4.86   2033.18
1.00
 lambda[87]      4.59     46.73       0.94       0.00       4.65   2995.20
1.00
 lambda[88]      2.33     11.67       0.84       0.00       3.84   3076.48
1.00
 lambda[89]     25.54    431.68       2.81       0.00      19.28   1720.86
1.00
 lambda[90]      9.13     52.84       1.94       0.00      11.89   3092.57
1.00
 lambda[91]      3.89     23.86       0.92       0.00       5.07   3762.97
1.00
 lambda[92]      3.63     30.01       0.91       0.00       4.67   4142.69
1.00
 lambda[93]      5.04     86.94       0.90       0.00       5.10   3366.06
1.00
 lambda[94]      5.22     55.11       0.89       0.00       5.10   2610.42
1.00
 lambda[95]      5.96     57.52       1.00       0.00       6.12   2812.28
1.00
 lambda[96]      3.42     28.93       0.80       0.00       4.61   4460.67
1.00
 lambda[97]      3.26     13.70       0.93       0.00       4.95   2928.97
1.00
 lambda[98]      3.64     27.93       0.87       0.00       4.27   3076.66
1.00
 lambda[99]      2.24      7.42       0.94       0.00       3.78   3448.23
1.00
lambda[100]      5.97     68.08       0.99       0.00       5.89   4376.18
```

```
1.00
lambda[101]     3.11    18.44     0.91     0.00     4.54  4078.13
1.00
lambda[102]     2.57    10.72     0.77     0.00     4.18  2362.98
1.00
lambda[103]    55.76  3563.30     0.92     0.00     4.95  4998.61
1.00
lambda[104]    11.76   273.28     1.00     0.00     5.72  2738.03
1.00
lambda[105]     3.34    56.02     0.69     0.00     2.92  2876.43
1.00
lambda[106]     2.95    19.60     0.83     0.00     3.92  1747.99
1.00
lambda[107]     4.97   138.61     0.87     0.00     4.40  4707.94
1.00
lambda[108]     2.36    20.99     0.76     0.00     3.12  2724.90
1.00
lambda[109]    11.95   388.38     0.91     0.00     4.94  3929.63
1.00
lambda[110]     7.62    64.73     1.17     0.00     8.42  4587.14
1.00
lambda[111]     5.98   115.92     0.89     0.00     4.94  2929.36
1.00
lambda[112]     2.69    10.49     0.93     0.00     4.67  2409.22
1.00
lambda[113]     3.85    22.44     0.89     0.00     4.77  2353.81
1.00
lambda[114]     4.08    43.68     0.92     0.00     4.86  2640.50
1.00
lambda[115]    40.69   709.33     3.13     0.00    26.82  1680.97
1.00
lambda[116]     7.02   164.14     1.04     0.00     5.72  4193.98
1.00
lambda[117]    28.22   830.43     1.70     0.00     7.24  2762.60
1.00
lambda[118]     5.94    45.77     1.17     0.00     8.14  3728.15
1.00
lambda[119]     3.49    46.25     0.84     0.00     4.03  4657.94
1.00
lambda[120]     4.21    69.82     0.79     0.00     4.04  4729.89
1.00
lambda[121]     5.01    39.45     0.96     0.00     5.48  3081.68
1.00
lambda[122]     4.65    58.70     0.92     0.00     4.82  3008.45
1.00
lambda[123]     1.27     4.87     0.55     0.00     2.22  3185.29
1.00
lambda[124]     6.98   178.54     0.92     0.00     5.34  2277.25
```

```
1.00
lambda[125]     5.01    72.84     0.85     0.00     4.20   3958.85
1.00
lambda[126]     3.73    26.25     1.21     0.00     4.90   3434.31
1.00
lambda[127]     6.17    54.81     1.07     0.00     6.96   2576.49
1.00
lambda[128]     8.40    63.40     1.61     0.00    10.40   3474.33
1.00
lambda[129]     5.21    70.17     0.82     0.00     4.63   3751.98
1.00
lambda[130]     7.88   158.37     0.98     0.00     5.88   2641.09
1.00
lambda[131]    11.16   159.62     1.20     0.00     7.98   3292.41
1.00
lambda[132]     2.29    28.85     0.65     0.00     2.79   4618.07
1.00
lambda[133]     6.00    86.21     0.93     0.00     5.01   4384.16
1.00
lambda[134]     6.08   139.63     0.79     0.00     4.24   2741.65
1.00
lambda[135]     3.54    15.18     1.21     0.00     5.69   3226.24
1.00
lambda[136]     4.21    38.12     0.96     0.00     6.08   4008.45
1.00
lambda[137]     3.19    40.60     0.83     0.00     4.23   4629.70
1.00
lambda[138]    14.81   489.74     1.01     0.00     7.49   3271.40
1.00
lambda[139]     3.40    14.52     0.93     0.00     5.61   2300.35
1.00
lambda[140]     5.56    67.99     0.91     0.00     5.23   1941.85
1.00
lambda[141]     2.29    26.19     0.59     0.00     2.83   4196.48
1.00
lambda[142]     9.93   136.14     1.26     0.00    10.35   3596.78
1.00
lambda[143]     3.51    32.35     0.80     0.00     3.86   2042.01
1.00
        msq     2.40     1.15     2.15     0.98     3.82   2021.55
1.00
      sigma     9.24     8.09     6.83     0.30    20.41   6204.54
1.00
    var_obs     0.00     0.00     0.00     0.00     0.00   5536.80
1.00
       xisq     0.00     0.00     0.00     0.00     0.00   4909.30
1.00
```

```
Number of divergences: 0

MCMC elapsed time: 56.75512194633484
[dimension 01/144]  inactive:   -2.07e-01 +- 3.10e-01
[dimension 02/144]  inactive:   -1.32e-01 +- 5.51e-01
[dimension 03/144]  inactive:   2.34e-01 +- 5.36e-01
[dimension 04/144]  active:     1.94e+00 +- 9.22e-01
[dimension 05/144]  inactive:   -1.09e-02 +- 5.29e-01
[dimension 06/144]  inactive:   -2.26e-02 +- 4.78e-01
[dimension 07/144]  inactive:   -1.58e-02 +- 2.29e-01
[dimension 08/144]  inactive:   4.04e-01 +- 7.74e-01
[dimension 09/144]  inactive:   -5.74e-02 +- 3.38e-01
[dimension 10/144]  inactive:   -3.11e-01 +- 3.10e-01
[dimension 11/144]  inactive:   -2.35e-01 +- 6.14e-01
[dimension 12/144]  inactive:   -1.44e-01 +- 4.56e-01
[dimension 13/144]  inactive:   -1.68e-02 +- 3.78e-01
[dimension 14/144]  inactive:   -2.88e-02 +- 5.07e-01
[dimension 15/144]  inactive:   -4.95e-02 +- 4.76e-01
[dimension 16/144]  active:     -5.90e-01 +- 3.23e-01
[dimension 17/144]  inactive:   1.15e-01 +- 5.68e-01
[dimension 18/144]  inactive:   -3.91e-02 +- 3.65e-01
[dimension 19/144]  inactive:   -5.05e-02 +- 2.08e-01
[dimension 20/144]  inactive:   -2.11e-01 +- 5.67e-01
[dimension 21/144]  inactive:   -8.56e-02 +- 3.54e-01
[dimension 22/144]  inactive:   -3.75e-01 +- 6.15e-01
[dimension 23/144]  inactive:   -1.46e-01 +- 6.16e-01
[dimension 24/144]  inactive:   -5.76e-02 +- 4.50e-01
[dimension 25/144]  inactive:   -5.20e-02 +- 1.62e-01
[dimension 26/144]  inactive:   1.44e-01 +- 5.32e-01
[dimension 27/144]  inactive:   -3.05e-01 +- 4.58e-01
[dimension 28/144]  inactive:   6.87e-02 +- 2.67e-01
[dimension 29/144]  inactive:   4.14e-02 +- 4.97e-01
[dimension 30/144]  inactive:   6.98e-02 +- 3.81e-01
[dimension 31/144]  inactive:   -2.90e-03 +- 3.97e-01
[dimension 32/144]  inactive:   -2.61e-02 +- 4.89e-01
[dimension 33/144]  inactive:   -1.89e-01 +- 5.71e-01
[dimension 34/144]  inactive:   -2.77e-02 +- 2.53e-01
[dimension 35/144]  inactive:   1.54e-01 +- 5.90e-01
[dimension 36/144]  inactive:   1.55e-01 +- 3.84e-01
[dimension 37/144]  active:     3.72e-01 +- 2.41e-01
[dimension 38/144]  inactive:   -3.90e-01 +- 7.54e-01
[dimension 39/144]  inactive:   -1.42e-01 +- 4.35e-01
[dimension 40/144]  inactive:   4.12e-01 +- 5.74e-01
[dimension 41/144]  inactive:   1.28e-01 +- 5.60e-01
[dimension 42/144]  inactive:   -1.56e-02 +- 4.75e-01
[dimension 43/144]  active:     -3.82e-01 +- 2.47e-01
[dimension 44/144]  inactive:   -1.31e-01 +- 5.37e-01
[dimension 45/144]  inactive:   2.86e-02 +- 3.09e-01
```

```
[dimension 46/144]  inactive:   2.33e-01 +- 1.68e-01
[dimension 47/144]  inactive:   -6.98e-02 +- 4.70e-01
[dimension 48/144]  inactive:   1.65e-01 +- 3.53e-01
[dimension 49/144]  inactive:   -1.11e-02 +- 2.39e-01
[dimension 50/144]  inactive:   -4.85e-02 +- 4.99e-01
[dimension 51/144]  inactive:   1.83e-02 +- 3.62e-01
[dimension 52/144]  inactive:   2.98e-02 +- 1.04e-01
[dimension 53/144]  inactive:   9.33e-02 +- 3.65e-01
[dimension 54/144]  inactive:   1.78e-01 +- 3.09e-01
[dimension 55/144]  inactive:   2.54e-02 +- 1.97e-01
[dimension 56/144]  inactive:   -2.49e-01 +- 4.81e-01
[dimension 57/144]  inactive:   3.51e-01 +- 5.19e-01
[dimension 58/144]  inactive:   8.60e-02 +- 3.16e-01
[dimension 59/144]  inactive:   6.46e-02 +- 4.37e-01
[dimension 60/144]  inactive:   -3.28e-01 +- 6.02e-01
[dimension 61/144]  inactive:   -3.21e-01 +- 3.09e-01
[dimension 62/144]  inactive:   -1.11e-01 +- 4.45e-01
[dimension 63/144]  inactive:   3.01e-01 +- 5.21e-01
[dimension 64/144]  inactive:   -3.52e-01 +- 3.91e-01
[dimension 65/144]  inactive:   -3.55e-02 +- 4.93e-01
[dimension 66/144]  inactive:   5.45e-01 +- 6.84e-01
[dimension 67/144]  active:     -1.47e+00 +- 9.23e-01
[dimension 68/144]  inactive:   -6.88e-02 +- 5.65e-01
[dimension 69/144]  inactive:   1.25e-01 +- 5.08e-01
[dimension 70/144]  active:     4.72e-01 +- 2.87e-01
[dimension 71/144]  inactive:   1.51e-01 +- 5.43e-01
[dimension 72/144]  inactive:   -2.17e-01 +- 4.20e-01
[dimension 73/144]  inactive:   -1.42e-02 +- 1.96e-01
[dimension 74/144]  inactive:   -1.25e-01 +- 5.23e-01
[dimension 75/144]  inactive:   -6.72e-02 +- 3.72e-01
[dimension 76/144]  inactive:   3.29e-01 +- 4.96e-01
[dimension 77/144]  inactive:   9.55e-02 +- 5.33e-01
[dimension 78/144]  inactive:   1.17e-02 +- 4.13e-01
[dimension 79/144]  inactive:   2.43e-01 +- 2.46e-01
[dimension 80/144]  inactive:   -8.94e-02 +- 4.93e-01
[dimension 81/144]  inactive:   -7.27e-02 +- 3.84e-01
[dimension 82/144]  inactive:   1.60e-02 +- 1.40e-01
[dimension 83/144]  inactive:   -1.96e-01 +- 4.15e-01
[dimension 84/144]  inactive:   -1.24e-01 +- 3.68e-01
[dimension 85/144]  inactive:   -3.06e-02 +- 2.85e-01
[dimension 86/144]  inactive:   1.48e-01 +- 4.58e-01
[dimension 87/144]  inactive:   -2.98e-02 +- 4.56e-01
[dimension 88/144]  inactive:   1.92e-01 +- 2.96e-01
[dimension 89/144]  inactive:   3.58e-02 +- 3.86e-01
[dimension 90/144]  inactive:   8.03e-01 +- 7.11e-01
[dimension 91/144]  inactive:   -3.73e-01 +- 2.54e-01
[dimension 92/144]  inactive:   -9.41e-02 +- 4.66e-01
[dimension 93/144]  inactive:   -1.33e-01 +- 4.41e-01
```

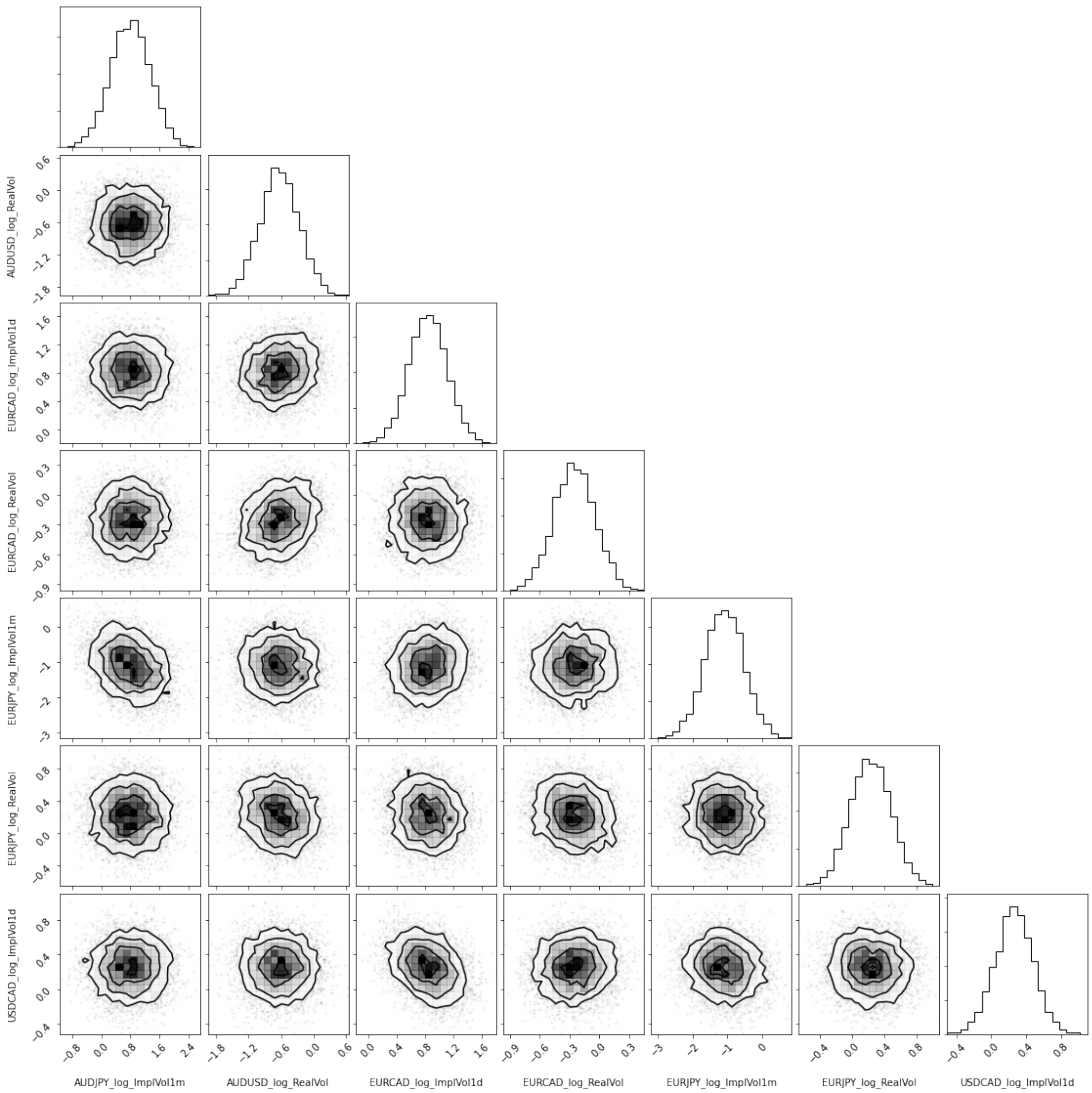
```
[dimension 94/144]  inactive:   -1.10e-04 +- 3.90e-01
[dimension 95/144]  inactive:   8.67e-02 +- 4.69e-01
[dimension 96/144]  inactive:   -1.59e-01 +- 5.60e-01
[dimension 97/144]  inactive:   -1.16e-01 +- 2.34e-01
[dimension 98/144]  inactive:   1.39e-01 +- 4.66e-01
[dimension 99/144]  inactive:   6.50e-02 +- 3.95e-01
[dimension 100/144]  inactive:  2.01e-01 +- 2.31e-01
[dimension 101/144]  inactive:  -1.63e-01 +- 4.36e-01
[dimension 102/144]  inactive:  1.06e-01 +- 4.01e-01
[dimension 103/144]  inactive:  -1.17e-02 +- 2.94e-01
[dimension 104/144]  inactive:  1.27e-01 +- 4.43e-01
[dimension 105/144]  inactive:  -2.06e-01 +- 5.11e-01
[dimension 106/144]  inactive:  9.90e-02 +- 1.90e-01
[dimension 107/144]  inactive:  -4.32e-02 +- 3.75e-01
[dimension 108/144]  inactive:  9.32e-02 +- 3.80e-01
[dimension 109/144]  inactive:  -1.22e-01 +- 2.00e-01
[dimension 110/144]  inactive:  -7.66e-02 +- 4.78e-01
[dimension 111/144]  inactive:  3.45e-01 +- 5.73e-01
[dimension 112/144]  inactive:  7.51e-02 +- 4.32e-01
[dimension 113/144]  inactive:  -4.14e-02 +- 4.91e-01
[dimension 114/144]  inactive:  -6.10e-02 +- 4.56e-01
[dimension 115/144]  inactive:  1.83e-01 +- 1.73e-01
[dimension 116/144]  inactive:  1.15e+00 +- 1.27e+00
[dimension 117/144]  inactive:  -2.53e-01 +- 4.23e-01
[dimension 118/144]  active:    4.12e-01 +- 2.37e-01
[dimension 119/144]  inactive:  -3.32e-01 +- 6.64e-01
[dimension 120/144]  inactive:  -8.77e-02 +- 3.47e-01
[dimension 121/144]  inactive:  5.06e-02 +- 2.89e-01
[dimension 122/144]  inactive:  9.52e-02 +- 5.16e-01
[dimension 123/144]  inactive:  -5.43e-02 +- 4.46e-01
[dimension 124/144]  inactive:  1.93e-02 +- 1.48e-01
[dimension 125/144]  inactive:  -5.40e-02 +- 4.69e-01
[dimension 126/144]  inactive:  1.56e-01 +- 3.19e-01
[dimension 127/144]  inactive:  -2.89e-01 +- 2.33e-01
[dimension 128/144]  inactive:  -2.20e-01 +- 6.02e-01
[dimension 129/144]  inactive:  -4.87e-01 +- 5.99e-01
[dimension 130/144]  inactive:  -3.89e-02 +- 3.32e-01
[dimension 131/144]  inactive:  1.71e-02 +- 5.49e-01
[dimension 132/144]  inactive:  -3.19e-01 +- 6.22e-01
[dimension 133/144]  inactive:  6.90e-02 +- 1.69e-01
[dimension 134/144]  inactive:  5.07e-02 +- 4.80e-01
[dimension 135/144]  inactive:  -7.53e-02 +- 3.29e-01
[dimension 136/144]  inactive:  2.75e-01 +- 2.66e-01
[dimension 137/144]  inactive:  -4.79e-02 +- 5.43e-01
[dimension 138/144]  inactive:  -2.04e-02 +- 3.72e-01
[dimension 139/144]  inactive:  2.28e-02 +- 4.40e-01
[dimension 140/144]  inactive:  3.23e-03 +- 5.45e-01
[dimension 141/144]  inactive:  7.42e-02 +- 4.70e-01
```

```
[dimension 142/144]  inactive:  -4.79e-03 +- 1.49e-01
[dimension 143/144]  inactive:  3.84e-01 +- 7.77e-01
[dimension 144/144]  inactive:  4.72e-03 +- 3.36e-01
Identified a total of 7 active dimensions.
Active dimensions: [3, 15, 36, 42, 66, 69, 117]
```

[40]:

```
labs = [short_labels[i] for i in all_active_dimensions if i <=␣
 ↪len(short_labels)] + pair_labs
fig = corner.corner(thetas, labels = labs)
fig.show()
```

```
/Users/sachinsmart/opt/anaconda3/lib/python3.7/site-
packages/ipykernel_launcher.py:3: UserWarning: Matplotlib is currently using
module://ipykernel.pylab.backend_inline, which is a non-GUI backend, so cannot
show the figure.
  This is separate from the ipykernel package so we can avoid doing imports
until
```



[41]:
```
print('Active dimensions:', labs)
```

```
Active dimensions: ['AUDJPY_log_ImplVol1m', 'AUDUSD_log_RealVol',
'EURCAD_log_ImplVol1d', 'EURCAD_log_RealVol', 'EURJPY_log_ImplVol1m',
'EURJPY_log_RealVol', 'USDCAD_log_ImplVol1d']
```

1.13 $\sigma = 1.0$

[42]:
```
all_active_dimensions, thetas, labels, pair_labs = main_modified(X=X, Y=y,␣
↪args=args, sigma=1.0, N_samps=5000,                                    ␣

↪labels=short_labels, **hypers)
```

```
sample: 100%|     | 5500/5500 [01:08<00:00, 80.81it/s, 31 steps of size
2.12e-01. acc. prob=0.90]


                mean       std    median      5.0%     95.0%     n_eff
r_hat
       eta1     0.30      0.08      0.29      0.18      0.42   2621.14
1.00
  lambda[0]     6.04    104.00      1.02      0.00      5.63   2468.33
1.00
  lambda[1]     8.29    133.87      1.01      0.00      6.68   2471.60
1.00
  lambda[2]     4.86     28.23      1.05      0.00      6.69   3196.49
1.00
  lambda[3]   231.21  10434.69      8.06      0.00     48.10   4415.06
1.00
  lambda[4]     3.39     16.13      0.96      0.00      5.47   4469.30
1.00
  lambda[5]     4.86     54.22      0.92      0.00      4.90   3251.43
1.00
  lambda[6]    11.21    129.33      1.10      0.00     10.49   3385.64
1.00
  lambda[7]     7.77     55.05      1.29      0.00     11.73   3761.46
1.00
  lambda[8]     2.31     10.67      0.79      0.00      3.94   3106.93
1.00
  lambda[9]     9.27    118.05      1.31      0.00      6.32   1389.63
1.00
 lambda[10]     6.69     69.98      1.09      0.00      7.50   2451.16
1.00
 lambda[11]     4.66     39.83      0.93      0.00      5.21   2923.57
1.00
 lambda[12]    99.43   6702.91      0.83      0.00      4.07   4776.75
1.00
 lambda[13]    10.42    291.51      0.93      0.00      5.36   4626.91
1.00
 lambda[14]    12.16    324.05      0.92      0.00      5.15   2831.73
1.00
 lambda[15]    10.74     64.67      2.86      0.00     15.20   2781.14
1.00
 lambda[16]    15.64    713.70      1.01      0.00      6.46   4967.20
1.00
 lambda[17]     4.08    103.67      0.84      0.00      4.11   4627.15
1.00
 lambda[18]     5.25     82.81      0.70      0.00      3.70   2067.15
1.00
 lambda[19]     4.91     36.07      1.03      0.00      6.62   4503.21
1.00
```

lambda[20]	2.96	17.76	0.85	0.00	4.17	3243.98	1.00
lambda[21]	8.73	55.74	1.66	0.00	12.65	3958.22	1.00
lambda[22]	6.60	107.94	1.02	0.00	6.84	2926.78	1.00
lambda[23]	4.51	48.91	0.92	0.00	4.79	3339.81	1.00
lambda[24]	14.52	272.58	0.68	0.00	4.22	845.54	1.00
lambda[25]	5.98	51.46	0.96	0.00	5.92	2673.33	1.00
lambda[26]	7.91	81.21	1.18	0.00	6.36	2133.23	1.00
lambda[27]	5.18	115.94	0.84	0.00	5.09	4618.43	1.00
lambda[28]	5.15	85.06	0.94	0.00	5.23	4947.28	1.00
lambda[29]	5.58	101.19	0.86	0.00	4.69	2760.46	1.00
lambda[30]	2.91	20.37	0.86	0.00	4.01	4009.23	1.00
lambda[31]	4.75	59.01	0.89	0.00	5.35	2442.37	1.00
lambda[32]	8.50	107.60	1.03	0.00	7.03	3147.09	1.00
lambda[33]	3.06	16.79	0.79	0.00	4.37	3714.25	1.00
lambda[34]	4.23	17.86	1.04	0.00	7.03	2961.12	1.00
lambda[35]	7.63	197.73	0.90	0.00	4.67	2190.72	1.00
lambda[36]	5.42	45.19	1.47	0.00	6.27	2238.18	1.00
lambda[37]	8.78	94.22	1.25	0.00	9.81	3238.08	1.00
lambda[38]	19.93	994.20	0.92	0.00	5.74	4193.64	1.00
lambda[39]	8.88	107.36	1.43	0.00	10.26	2632.32	1.00
lambda[40]	11.28	333.57	0.97	0.00	6.29	4508.75	1.00
lambda[41]	5.02	49.31	0.93	0.00	5.88	3742.57	1.00
lambda[42]	5.96	64.75	1.54	0.00	6.25	2264.62	1.00
lambda[43]	10.73	374.18	0.98	0.00	6.12	3677.48	1.00

lambda[44]	2.59	13.48	0.80	0.00	3.95	2539.23	1.00
lambda[45]	3.06	13.54	1.08	0.00	4.45	2295.94	1.00
lambda[46]	5.69	59.58	0.91	0.00	5.23	3444.22	1.00
lambda[47]	9.18	294.29	0.97	0.00	5.49	3084.87	1.00
lambda[48]	2.56	15.41	0.72	0.00	3.36	2199.28	1.00
lambda[49]	3.77	24.32	0.92	0.00	5.39	4128.45	1.00
lambda[50]	3.35	39.87	0.83	0.00	3.85	4634.50	1.00
lambda[51]	3.19	82.98	0.52	0.00	2.20	4326.72	1.00
lambda[52]	2.96	16.70	0.85	0.00	4.23	4136.74	1.00
lambda[53]	3.51	24.61	0.93	0.00	4.51	2931.06	1.00
lambda[54]	3.25	48.92	0.69	0.00	3.07	4085.80	1.00
lambda[55]	10.94	211.25	1.09	0.00	7.04	2962.46	1.00
lambda[56]	8.18	125.28	1.32	0.00	7.96	3666.57	1.00
lambda[57]	4.06	60.78	0.78	0.00	3.66	2641.05	1.00
lambda[58]	3.59	20.78	0.90	0.00	5.38	3200.75	1.00
lambda[59]	6.38	95.45	1.14	0.00	7.53	3821.95	1.00
lambda[60]	9.03	57.08	1.59	0.00	12.58	4054.11	1.00
lambda[61]	6.50	159.52	0.88	0.00	4.69	3980.74	1.00
lambda[62]	6.14	74.91	1.24	0.00	8.05	4812.84	1.00
lambda[63]	5.60	43.87	1.34	0.00	7.31	4273.41	1.00
lambda[64]	3.82	23.39	0.93	0.00	5.31	3763.84	1.00
lambda[65]	8.03	45.30	1.66	0.00	12.48	3689.49	1.00
lambda[66]	33.66	341.82	6.29	0.01	38.04	4171.97	1.00
lambda[67]	6.56	130.46	0.98	0.00	5.91	2873.47	1.00

lambda[68]	3.77	29.56	0.96	0.00	5.43	4319.67	1.00
lambda[69]	14.79	92.03	2.98	0.00	21.94	3463.14	1.00
lambda[70]	4.05	22.56	0.96	0.00	5.65	3255.57	1.00
lambda[71]	5.20	52.94	1.00	0.00	5.52	4694.17	1.00
lambda[72]	3.00	46.84	0.68	0.00	2.99	1695.25	1.00
lambda[73]	4.65	47.20	0.97	0.00	6.06	2468.02	1.00
lambda[74]	4.04	41.59	0.87	0.00	4.67	2901.75	1.00
lambda[75]	5.36	53.58	1.25	0.00	8.24	4768.98	1.00
lambda[76]	144.58	7044.54	0.93	0.00	5.52	2504.06	1.00
lambda[77]	8.01	257.75	0.89	0.00	4.89	3324.47	1.00
lambda[78]	3.86	33.64	1.04	0.00	4.44	2152.35	1.00
lambda[79]	4.93	61.37	0.93	0.00	4.96	3959.20	1.00
lambda[80]	12.71	564.33	0.89	0.00	5.16	4667.66	1.00
lambda[81]	1.53	11.34	0.53	0.00	2.05	1904.02	1.00
lambda[82]	4.20	25.02	0.99	0.00	5.83	1771.91	1.00
lambda[83]	3.32	40.98	0.86	0.00	4.32	4704.95	1.00
lambda[84]	2.89	20.36	0.81	0.00	3.81	3544.81	1.00
lambda[85]	4.53	33.26	0.96	0.00	5.38	2383.15	1.00
lambda[86]	6.89	96.38	0.89	0.00	4.86	2033.18	1.00
lambda[87]	4.59	46.73	0.94	0.00	4.65	2995.20	1.00
lambda[88]	2.33	11.67	0.84	0.00	3.84	3076.48	1.00
lambda[89]	25.54	431.68	2.81	0.00	19.28	1720.86	1.00
lambda[90]	9.13	52.84	1.94	0.00	11.89	3092.57	1.00
lambda[91]	3.89	23.86	0.92	0.00	5.07	3762.97	1.00

lambda[92]	3.63	30.01	0.91	0.00	4.67	4142.69	1.00
lambda[93]	5.04	86.94	0.90	0.00	5.10	3366.06	1.00
lambda[94]	5.22	55.11	0.89	0.00	5.10	2610.42	1.00
lambda[95]	5.96	57.52	1.00	0.00	6.12	2812.28	1.00
lambda[96]	3.42	28.93	0.80	0.00	4.61	4460.67	1.00
lambda[97]	3.26	13.70	0.93	0.00	4.95	2928.97	1.00
lambda[98]	3.64	27.93	0.87	0.00	4.27	3076.66	1.00
lambda[99]	2.24	7.42	0.94	0.00	3.78	3448.23	1.00
lambda[100]	5.97	68.08	0.99	0.00	5.89	4376.18	1.00
lambda[101]	3.11	18.44	0.91	0.00	4.54	4078.13	1.00
lambda[102]	2.57	10.72	0.77	0.00	4.18	2362.98	1.00
lambda[103]	55.76	3563.30	0.92	0.00	4.95	4998.61	1.00
lambda[104]	11.76	273.28	1.00	0.00	5.72	2738.03	1.00
lambda[105]	3.34	56.02	0.69	0.00	2.92	2876.43	1.00
lambda[106]	2.95	19.60	0.83	0.00	3.92	1747.99	1.00
lambda[107]	4.97	138.61	0.87	0.00	4.40	4707.94	1.00
lambda[108]	2.36	20.99	0.76	0.00	3.12	2724.90	1.00
lambda[109]	11.95	388.38	0.91	0.00	4.94	3929.63	1.00
lambda[110]	7.62	64.73	1.17	0.00	8.42	4587.14	1.00
lambda[111]	5.98	115.92	0.89	0.00	4.94	2929.36	1.00
lambda[112]	2.69	10.49	0.93	0.00	4.67	2409.22	1.00
lambda[113]	3.85	22.44	0.89	0.00	4.77	2353.81	1.00
lambda[114]	4.08	43.68	0.92	0.00	4.86	2640.50	1.00
lambda[115]	40.69	709.33	3.13	0.00	26.82	1680.97	1.00

lambda[116]	7.02	164.14	1.04	0.00	5.72	4193.98	1.00
lambda[117]	28.22	830.43	1.70	0.00	7.24	2762.60	1.00
lambda[118]	5.94	45.77	1.17	0.00	8.14	3728.15	1.00
lambda[119]	3.49	46.25	0.84	0.00	4.03	4657.94	1.00
lambda[120]	4.21	69.82	0.79	0.00	4.04	4729.89	1.00
lambda[121]	5.01	39.45	0.96	0.00	5.48	3081.68	1.00
lambda[122]	4.65	58.70	0.92	0.00	4.82	3008.45	1.00
lambda[123]	1.27	4.87	0.55	0.00	2.22	3185.29	1.00
lambda[124]	6.98	178.54	0.92	0.00	5.34	2277.25	1.00
lambda[125]	5.01	72.84	0.85	0.00	4.20	3958.85	1.00
lambda[126]	3.73	26.25	1.21	0.00	4.90	3434.31	1.00
lambda[127]	6.17	54.81	1.07	0.00	6.96	2576.49	1.00
lambda[128]	8.40	63.40	1.61	0.00	10.40	3474.33	1.00
lambda[129]	5.21	70.17	0.82	0.00	4.63	3751.98	1.00
lambda[130]	7.88	158.37	0.98	0.00	5.88	2641.09	1.00
lambda[131]	11.16	159.62	1.20	0.00	7.98	3292.41	1.00
lambda[132]	2.29	28.85	0.65	0.00	2.79	4618.07	1.00
lambda[133]	6.00	86.21	0.93	0.00	5.01	4384.16	1.00
lambda[134]	6.08	139.63	0.79	0.00	4.24	2741.65	1.00
lambda[135]	3.54	15.18	1.21	0.00	5.69	3226.24	1.00
lambda[136]	4.21	38.12	0.96	0.00	6.08	4008.45	1.00
lambda[137]	3.19	40.60	0.83	0.00	4.23	4629.70	1.00
lambda[138]	14.81	489.74	1.01	0.00	7.49	3271.40	1.00
lambda[139]	3.40	14.52	0.93	0.00	5.61	2300.35	1.00

```
lambda[140]      5.56     67.99      0.91      0.00      5.23   1941.85
1.00
lambda[141]      2.29     26.19      0.59      0.00      2.83   4196.48
1.00
lambda[142]      9.93    136.14      1.26      0.00     10.35   3596.78
1.00
lambda[143]      3.51     32.35      0.80      0.00      3.86   2042.01
1.00
        msq      2.40      1.15      2.15      0.98      3.82   2021.55
1.00
      sigma      9.24      8.09      6.83      0.30     20.41   6204.54
1.00
    var_obs      0.00      0.00      0.00      0.00      0.00   5536.80
1.00
       xisq      0.00      0.00      0.00      0.00      0.00   4909.30
1.00

Number of divergences: 0

MCMC elapsed time: 72.02126908302307
[dimension 01/144]  inactive:   -2.07e-01 +- 3.10e-01
[dimension 02/144]  inactive:   -1.32e-01 +- 5.51e-01
[dimension 03/144]  inactive:   2.34e-01 +- 5.36e-01
[dimension 04/144]  active:     1.94e+00 +- 9.22e-01
[dimension 05/144]  inactive:   -1.09e-02 +- 5.29e-01
[dimension 06/144]  inactive:   -2.26e-02 +- 4.78e-01
[dimension 07/144]  inactive:   -1.58e-02 +- 2.29e-01
[dimension 08/144]  inactive:   4.04e-01 +- 7.74e-01
[dimension 09/144]  inactive:   -5.74e-02 +- 3.38e-01
[dimension 10/144]  active:     -3.11e-01 +- 3.10e-01
[dimension 11/144]  inactive:   -2.35e-01 +- 6.14e-01
[dimension 12/144]  inactive:   -1.44e-01 +- 4.56e-01
[dimension 13/144]  inactive:   -1.68e-02 +- 3.78e-01
[dimension 14/144]  inactive:   -2.88e-02 +- 5.07e-01
[dimension 15/144]  inactive:   -4.95e-02 +- 4.76e-01
[dimension 16/144]  active:     -5.90e-01 +- 3.23e-01
[dimension 17/144]  inactive:   1.15e-01 +- 5.68e-01
[dimension 18/144]  inactive:   -3.91e-02 +- 3.65e-01
[dimension 19/144]  inactive:   -5.05e-02 +- 2.08e-01
[dimension 20/144]  inactive:   -2.11e-01 +- 5.67e-01
[dimension 21/144]  inactive:   -8.56e-02 +- 3.54e-01
[dimension 22/144]  inactive:   -3.75e-01 +- 6.15e-01
[dimension 23/144]  inactive:   -1.46e-01 +- 6.16e-01
[dimension 24/144]  inactive:   -5.76e-02 +- 4.50e-01
[dimension 25/144]  inactive:   -5.20e-02 +- 1.62e-01
[dimension 26/144]  inactive:   1.44e-01 +- 5.32e-01
[dimension 27/144]  inactive:   -3.05e-01 +- 4.58e-01
[dimension 28/144]  inactive:   6.87e-02 +- 2.67e-01
```

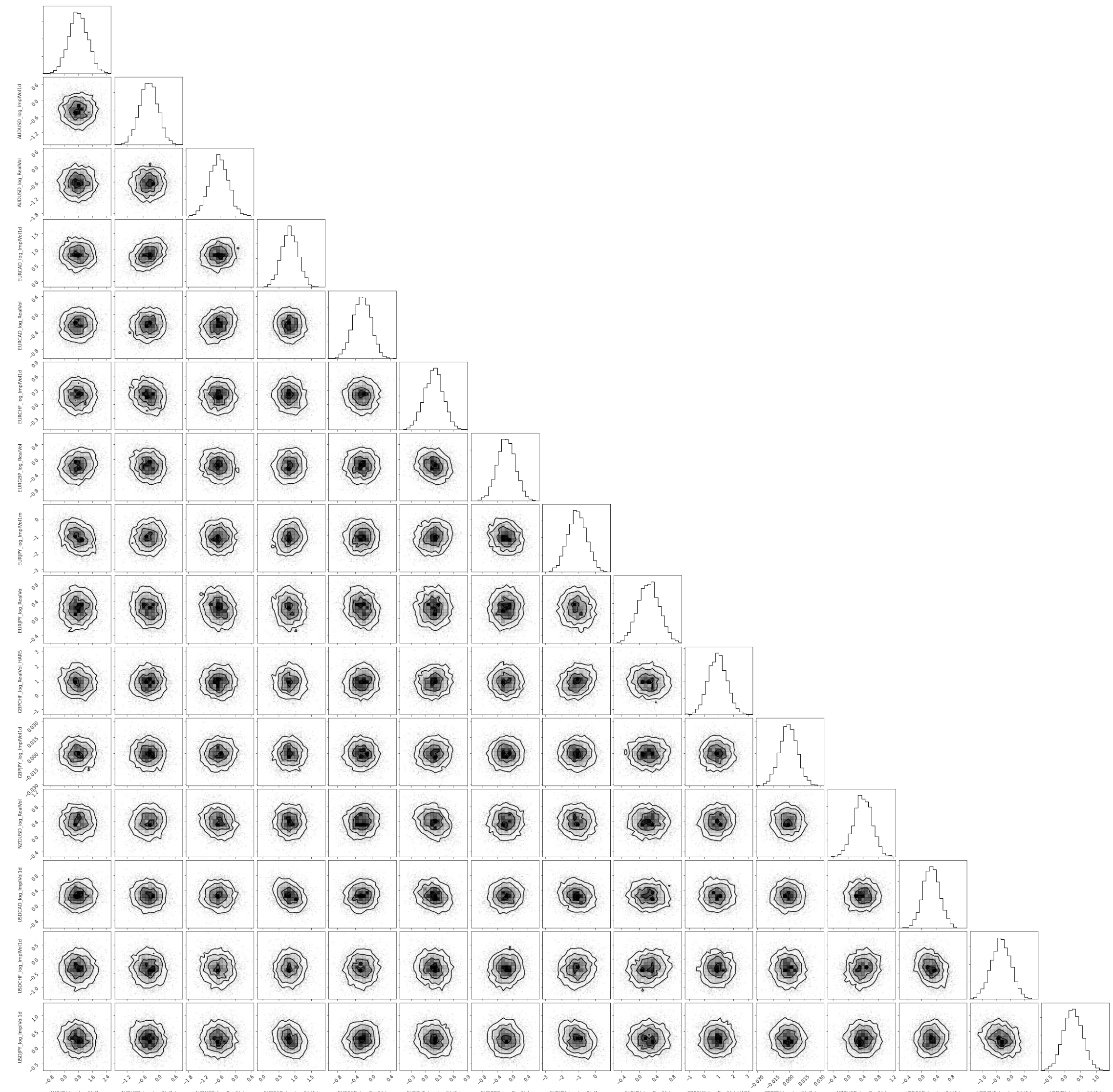
```
[dimension 29/144]  inactive:   4.14e-02 +- 4.97e-01
[dimension 30/144]  inactive:   6.98e-02 +- 3.81e-01
[dimension 31/144]  inactive:   -2.90e-03 +- 3.97e-01
[dimension 32/144]  inactive:   -2.61e-02 +- 4.89e-01
[dimension 33/144]  inactive:   -1.89e-01 +- 5.71e-01
[dimension 34/144]  inactive:   -2.77e-02 +- 2.53e-01
[dimension 35/144]  inactive:   1.54e-01 +- 5.90e-01
[dimension 36/144]  inactive:   1.55e-01 +- 3.84e-01
[dimension 37/144]  active:     3.72e-01 +- 2.41e-01
[dimension 38/144]  inactive:   -3.90e-01 +- 7.54e-01
[dimension 39/144]  inactive:   -1.42e-01 +- 4.35e-01
[dimension 40/144]  inactive:   4.12e-01 +- 5.74e-01
[dimension 41/144]  inactive:   1.28e-01 +- 5.60e-01
[dimension 42/144]  inactive:   -1.56e-02 +- 4.75e-01
[dimension 43/144]  active:     -3.82e-01 +- 2.47e-01
[dimension 44/144]  inactive:   -1.31e-01 +- 5.37e-01
[dimension 45/144]  inactive:   2.86e-02 +- 3.09e-01
[dimension 46/144]  active:     2.33e-01 +- 1.68e-01
[dimension 47/144]  inactive:   -6.98e-02 +- 4.70e-01
[dimension 48/144]  inactive:   1.65e-01 +- 3.53e-01
[dimension 49/144]  inactive:   -1.11e-02 +- 2.39e-01
[dimension 50/144]  inactive:   -4.85e-02 +- 4.99e-01
[dimension 51/144]  inactive:   1.83e-02 +- 3.62e-01
[dimension 52/144]  inactive:   2.98e-02 +- 1.04e-01
[dimension 53/144]  inactive:   9.33e-02 +- 3.65e-01
[dimension 54/144]  inactive:   1.78e-01 +- 3.09e-01
[dimension 55/144]  inactive:   2.54e-02 +- 1.97e-01
[dimension 56/144]  inactive:   -2.49e-01 +- 4.81e-01
[dimension 57/144]  inactive:   3.51e-01 +- 5.19e-01
[dimension 58/144]  inactive:   8.60e-02 +- 3.16e-01
[dimension 59/144]  inactive:   6.46e-02 +- 4.37e-01
[dimension 60/144]  inactive:   -3.28e-01 +- 6.02e-01
[dimension 61/144]  active:     -3.21e-01 +- 3.09e-01
[dimension 62/144]  inactive:   -1.11e-01 +- 4.45e-01
[dimension 63/144]  inactive:   3.01e-01 +- 5.21e-01
[dimension 64/144]  inactive:   -3.52e-01 +- 3.91e-01
[dimension 65/144]  inactive:   -3.55e-02 +- 4.93e-01
[dimension 66/144]  inactive:   5.45e-01 +- 6.84e-01
[dimension 67/144]  active:     -1.47e+00 +- 9.23e-01
[dimension 68/144]  inactive:   -6.88e-02 +- 5.65e-01
[dimension 69/144]  inactive:   1.25e-01 +- 5.08e-01
[dimension 70/144]  active:     4.72e-01 +- 2.87e-01
[dimension 71/144]  inactive:   1.51e-01 +- 5.43e-01
[dimension 72/144]  inactive:   -2.17e-01 +- 4.20e-01
[dimension 73/144]  inactive:   -1.42e-02 +- 1.96e-01
[dimension 74/144]  inactive:   -1.25e-01 +- 5.23e-01
[dimension 75/144]  inactive:   -6.72e-02 +- 3.72e-01
[dimension 76/144]  inactive:   3.29e-01 +- 4.96e-01
```

```
[dimension 77/144]  inactive:   9.55e-02 +- 5.33e-01
[dimension 78/144]  inactive:   1.17e-02 +- 4.13e-01
[dimension 79/144]  inactive:   2.43e-01 +- 2.46e-01
[dimension 80/144]  inactive:   -8.94e-02 +- 4.93e-01
[dimension 81/144]  inactive:   -7.27e-02 +- 3.84e-01
[dimension 82/144]  inactive:   1.60e-02 +- 1.40e-01
[dimension 83/144]  inactive:   -1.96e-01 +- 4.15e-01
[dimension 84/144]  inactive:   -1.24e-01 +- 3.68e-01
[dimension 85/144]  inactive:   -3.06e-02 +- 2.85e-01
[dimension 86/144]  inactive:   1.48e-01 +- 4.58e-01
[dimension 87/144]  inactive:   -2.98e-02 +- 4.56e-01
[dimension 88/144]  inactive:   1.92e-01 +- 2.96e-01
[dimension 89/144]  inactive:   3.58e-02 +- 3.86e-01
[dimension 90/144]  active:     8.03e-01 +- 7.11e-01
[dimension 91/144]  active:     -3.73e-01 +- 2.54e-01
[dimension 92/144]  inactive:   -9.41e-02 +- 4.66e-01
[dimension 93/144]  inactive:   -1.33e-01 +- 4.41e-01
[dimension 94/144]  inactive:   -1.10e-04 +- 3.90e-01
[dimension 95/144]  inactive:   8.67e-02 +- 4.69e-01
[dimension 96/144]  inactive:   -1.59e-01 +- 5.60e-01
[dimension 97/144]  inactive:   -1.16e-01 +- 2.34e-01
[dimension 98/144]  inactive:   1.39e-01 +- 4.66e-01
[dimension 99/144]  inactive:   6.50e-02 +- 3.95e-01
[dimension 100/144]  inactive:  2.01e-01 +- 2.31e-01
[dimension 101/144]  inactive:  -1.63e-01 +- 4.36e-01
[dimension 102/144]  inactive:  1.06e-01 +- 4.01e-01
[dimension 103/144]  inactive:  -1.17e-02 +- 2.94e-01
[dimension 104/144]  inactive:  1.27e-01 +- 4.43e-01
[dimension 105/144]  inactive:  -2.06e-01 +- 5.11e-01
[dimension 106/144]  inactive:  9.90e-02 +- 1.90e-01
[dimension 107/144]  inactive:  -4.32e-02 +- 3.75e-01
[dimension 108/144]  inactive:  9.32e-02 +- 3.80e-01
[dimension 109/144]  inactive:  -1.22e-01 +- 2.00e-01
[dimension 110/144]  inactive:  -7.66e-02 +- 4.78e-01
[dimension 111/144]  inactive:  3.45e-01 +- 5.73e-01
[dimension 112/144]  inactive:  7.51e-02 +- 4.32e-01
[dimension 113/144]  inactive:  -4.14e-02 +- 4.91e-01
[dimension 114/144]  inactive:  -6.10e-02 +- 4.56e-01
[dimension 115/144]  active:    1.83e-01 +- 1.73e-01
[dimension 116/144]  inactive:  1.15e+00 +- 1.27e+00
[dimension 117/144]  inactive:  -2.53e-01 +- 4.23e-01
[dimension 118/144]  active:    4.12e-01 +- 2.37e-01
[dimension 119/144]  inactive:  -3.32e-01 +- 6.64e-01
[dimension 120/144]  inactive:  -8.77e-02 +- 3.47e-01
[dimension 121/144]  inactive:  5.06e-02 +- 2.89e-01
[dimension 122/144]  inactive:  9.52e-02 +- 5.16e-01
[dimension 123/144]  inactive:  -5.43e-02 +- 4.46e-01
[dimension 124/144]  inactive:  1.93e-02 +- 1.48e-01
```

```
[dimension 125/144]  inactive:  -5.40e-02 +- 4.69e-01
[dimension 126/144]  inactive:  1.56e-01 +- 3.19e-01
[dimension 127/144]  active:    -2.89e-01 +- 2.33e-01
[dimension 128/144]  inactive:  -2.20e-01 +- 6.02e-01
[dimension 129/144]  inactive:  -4.87e-01 +- 5.99e-01
[dimension 130/144]  inactive:  -3.89e-02 +- 3.32e-01
[dimension 131/144]  inactive:  1.71e-02 +- 5.49e-01
[dimension 132/144]  inactive:  -3.19e-01 +- 6.22e-01
[dimension 133/144]  inactive:  6.90e-02 +- 1.69e-01
[dimension 134/144]  inactive:  5.07e-02 +- 4.80e-01
[dimension 135/144]  inactive:  -7.53e-02 +- 3.29e-01
[dimension 136/144]  active:    2.75e-01 +- 2.66e-01
[dimension 137/144]  inactive:  -4.79e-02 +- 5.43e-01
[dimension 138/144]  inactive:  -2.04e-02 +- 3.72e-01
[dimension 139/144]  inactive:  2.28e-02 +- 4.40e-01
[dimension 140/144]  inactive:  3.23e-03 +- 5.45e-01
[dimension 141/144]  inactive:  7.42e-02 +- 4.70e-01
[dimension 142/144]  inactive:  -4.79e-03 +- 1.49e-01
[dimension 143/144]  inactive:  3.84e-01 +- 7.77e-01
[dimension 144/144]  inactive:  4.72e-03 +- 3.36e-01
Identified a total of 15 active dimensions.
Active dimensions: [3, 9, 15, 36, 42, 45, 60, 66, 69, 89, 90, 114, 117, 126,
135]
```

[43]:
```python
labs = [short_labels[i] for i in all_active_dimensions if i <=␣
 ↪len(short_labels)] + pair_labs
fig = corner.corner(thetas, labels = labs)
fig.show()
```

```
/Users/sachinsmart/opt/anaconda3/lib/python3.7/site-
packages/ipykernel_launcher.py:3: UserWarning: Matplotlib is currently using
module://ipykernel.pylab.backend_inline, which is a non-GUI backend, so cannot
show the figure.
  This is separate from the ipykernel package so we can avoid doing imports
until
```



[44]:
```
print('Active dimensions:', labs)
```

```
Active dimensions: ['AUDJPY_log_ImplVol1m', 'AUDUSD_log_ImplVol1d',
'AUDUSD_log_RealVol', 'EURCAD_log_ImplVol1d', 'EURCAD_log_RealVol',
'EURCHF_log_ImplVol1d', 'EURGBP_log_RealVol', 'EURJPY_log_ImplVol1m',
'EURJPY_log_RealVol', 'GBPCHF_log_RealVol_HAR5', 'GBPJPY_log_ImplVol1d',
'NZDUSD_log_RealVol', 'USDCAD_log_ImplVol1d', 'USDCHF_log_ImplVol1d',
'USDJPY_log_ImplVol1d']
```

In this exchange rate volatility forecasting experiment, we manage to obtain one active dimension at $\pm 2\sigma$, and it is quite an unusual one as it is the 1-month implied volatility of another currency, AUDJPY. However, these variables are highly correlated as seen in the correlation matrix below. The first principal component of this data frame accounts for 70% of the total variance. Thus, it is possibly less surprising that this regressor was selected, as it could have been almost any of them!

With $\pm 1.5\sigma$ bounds, we are able to recover 7 active dimensions, and none of them are EURGBP values. This is quite surprising as they are regressors from a variety of currency pairs. This again demonstrates the how similar these regressors are.

With $\pm 1\sigma$ bounds, we are able to recover 15 active dimensions, and still there are no EURGBP values.

These results are quite different from those run with LASSO, which are produced below.

[45]:
```
df.corr()
```

[45]:
```
                          Target_EURGBP_log_RealVol  AUDJPY_log_ImplVol1d  \
Target_EURGBP_log_RealVol                  1.000000              0.856674
AUDJPY_log_ImplVol1d                       0.856674              1.000000
AUDJPY_log_ImplVol1d_HAR22                 0.904750              0.873535
AUDJPY_log_ImplVol1d_HAR5                  0.920098              0.935216
AUDJPY_log_ImplVol1m                       0.931120              0.945132
…                                                 …                     …
USDJPY_log_ImplVol1m_HAR22                 0.916727              0.872905
USDJPY_log_ImplVol1m_HAR5                  0.926819              0.931799
USDJPY_log_RealVol                         0.808212              0.851758
USDJPY_log_RealVol_HAR22                   0.930606              0.911251
USDJPY_log_RealVol_HAR5                    0.877524              0.914372

                          AUDJPY_log_ImplVol1d_HAR22  \
Target_EURGBP_log_RealVol                   0.904750
AUDJPY_log_ImplVol1d                        0.873535
AUDJPY_log_ImplVol1d_HAR22                  1.000000
AUDJPY_log_ImplVol1d_HAR5                   0.957390
AUDJPY_log_ImplVol1m                        0.920841
…                                                  …
USDJPY_log_ImplVol1m_HAR22                  0.988895
USDJPY_log_ImplVol1m_HAR5                   0.915543
USDJPY_log_RealVol                          0.779211
USDJPY_log_RealVol_HAR22                    0.985602
USDJPY_log_RealVol_HAR5                     0.860501

                          AUDJPY_log_ImplVol1d_HAR5  AUDJPY_log_ImplVol1m  \
Target_EURGBP_log_RealVol                  0.920098              0.931120
AUDJPY_log_ImplVol1d                       0.935216              0.945132
AUDJPY_log_ImplVol1d_HAR22                 0.957390              0.920841
AUDJPY_log_ImplVol1d_HAR5                  1.000000              0.974080
AUDJPY_log_ImplVol1m                       0.974080              1.000000
…                                                 …                     …
USDJPY_log_ImplVol1m_HAR22                 0.957673              0.931867
USDJPY_log_ImplVol1m_HAR5                  0.975895              0.987832
USDJPY_log_RealVol                         0.844033              0.893671
USDJPY_log_RealVol_HAR22                   0.982409              0.962075
```

```
USDJPY_log_RealVol_HAR5                      0.940708              0.951568

                            AUDJPY_log_ImplVol1m_HAR22  \
Target_EURGBP_log_RealVol                     0.898322
AUDJPY_log_ImplVol1d                          0.846166
AUDJPY_log_ImplVol1d_HAR22                    0.992699
AUDJPY_log_ImplVol1d_HAR5                     0.939743
AUDJPY_log_ImplVol1m                          0.901061
…                                                  …
USDJPY_log_ImplVol1m_HAR22                    0.994826
USDJPY_log_ImplVol1m_HAR5                     0.897337
USDJPY_log_RealVol                            0.740669
USDJPY_log_RealVol_HAR22                      0.977561
USDJPY_log_RealVol_HAR5                       0.827672

                            AUDJPY_log_ImplVol1m_HAR5  AUDJPY_log_RealVol  \
Target_EURGBP_log_RealVol                    0.944106            0.834454
AUDJPY_log_ImplVol1d                         0.915300            0.875247
AUDJPY_log_ImplVol1d_HAR22                   0.964634            0.814800
AUDJPY_log_ImplVol1d_HAR5                    0.988667            0.883166
AUDJPY_log_ImplVol1m                         0.975908            0.924791
…                                                 …                   …
USDJPY_log_ImplVol1m_HAR22                   0.974446            0.814008
USDJPY_log_ImplVol1m_HAR5                    0.976982            0.906374
USDJPY_log_RealVol                           0.823868            0.957348
USDJPY_log_RealVol_HAR22                     0.985036            0.871221
USDJPY_log_RealVol_HAR5                      0.922383            0.924949

                            AUDJPY_log_RealVol_HAR22  AUDJPY_log_RealVol_HAR5  \
Target_EURGBP_log_RealVol                   0.916295                 0.905238
AUDJPY_log_ImplVol1d                        0.896594                 0.930350
AUDJPY_log_ImplVol1d_HAR22                  0.995000                 0.912192
AUDJPY_log_ImplVol1d_HAR5                   0.975994                 0.981201
AUDJPY_log_ImplVol1m                        0.943369                 0.968132
…                                                …                        …
USDJPY_log_ImplVol1m_HAR22                  0.987210                 0.910504
USDJPY_log_ImplVol1m_HAR5                   0.940116                 0.977724
USDJPY_log_RealVol                          0.816859                 0.897022
USDJPY_log_RealVol_HAR22                    0.994988                 0.956685
USDJPY_log_RealVol_HAR5                     0.895008                 0.979772

                            …  USDCHF_log_RealVol_HAR5  \
Target_EURGBP_log_RealVol   …                 0.920832
AUDJPY_log_ImplVol1d        …                 0.913002
AUDJPY_log_ImplVol1d_HAR22  …                 0.889965
AUDJPY_log_ImplVol1d_HAR5   …                 0.954158
AUDJPY_log_ImplVol1m        …                 0.968428
```

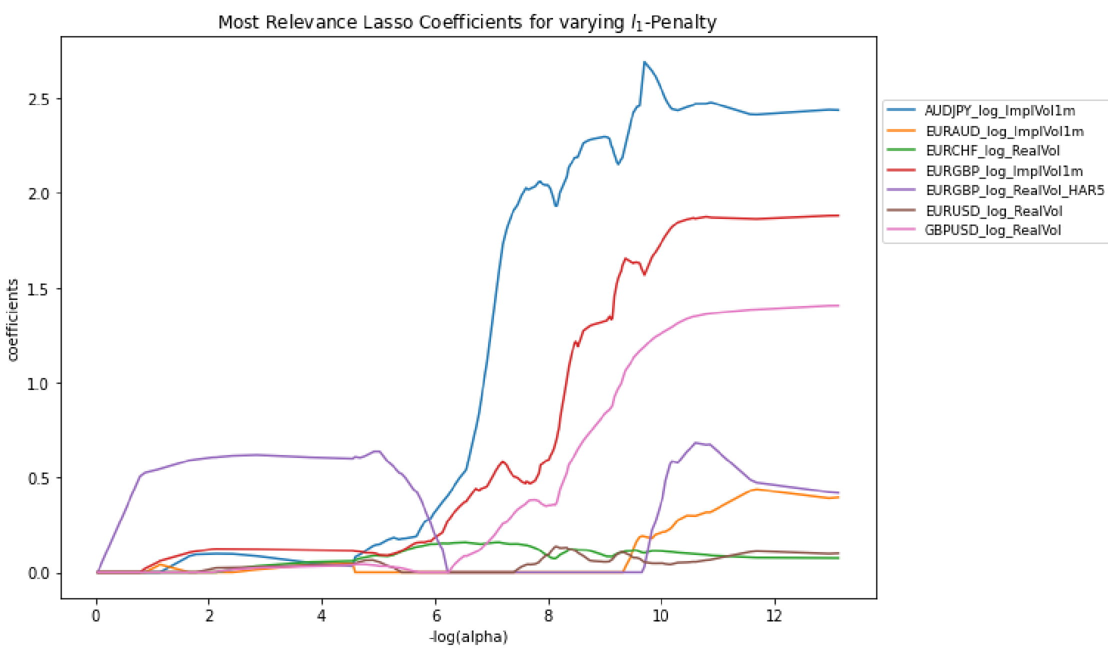
```
…                              …                      …
USDJPY_log_ImplVol1m_HAR22     …               0.909519
USDJPY_log_ImplVol1m_HAR5      …               0.987364
USDJPY_log_RealVol             …               0.893837
USDJPY_log_RealVol_HAR22       …               0.946453
USDJPY_log_RealVol_HAR5        …               0.969632

                           USDJPY_log_ImplVol1d  USDJPY_log_ImplVol1d_HAR22  \
Target_EURGBP_log_RealVol              0.837219                    0.926747
AUDJPY_log_ImplVol1d                   0.957982                    0.897391
AUDJPY_log_ImplVol1d_HAR22             0.821375                    0.992742
AUDJPY_log_ImplVol1d_HAR5              0.897375                    0.973904
AUDJPY_log_ImplVol1m                   0.932596                    0.951658
…                                           …                           …
USDJPY_log_ImplVol1m_HAR22             0.820749                    0.995304
USDJPY_log_ImplVol1m_HAR5              0.932594                    0.952215
USDJPY_log_RealVol                     0.929023                    0.812164
USDJPY_log_RealVol_HAR22               0.882446                    0.996511
USDJPY_log_RealVol_HAR5                0.952501                    0.896826

                           USDJPY_log_ImplVol1d_HAR5  USDJPY_log_ImplVol1m  \
Target_EURGBP_log_RealVol                   0.902636              0.880419
AUDJPY_log_ImplVol1d                        0.935423              0.928715
AUDJPY_log_ImplVol1d_HAR22                  0.897352              0.863306
AUDJPY_log_ImplVol1d_HAR5                   0.973757              0.929736
AUDJPY_log_ImplVol1m                        0.977429              0.972734
…                                                …                     …
USDJPY_log_ImplVol1m_HAR22                  0.903041              0.874222
USDJPY_log_ImplVol1m_HAR5                   0.991834              0.972729
USDJPY_log_RealVol                          0.904365              0.943009
USDJPY_log_RealVol_HAR22                    0.950907              0.925415
USDJPY_log_RealVol_HAR5                     0.986708              0.967700

                           USDJPY_log_ImplVol1m_HAR22  \
Target_EURGBP_log_RealVol                    0.916727
AUDJPY_log_ImplVol1d                         0.872905
AUDJPY_log_ImplVol1d_HAR22                   0.988895
AUDJPY_log_ImplVol1d_HAR5                    0.957673
AUDJPY_log_ImplVol1m                         0.931867
…                                                 …
USDJPY_log_ImplVol1m_HAR22                   1.000000
USDJPY_log_ImplVol1m_HAR5                    0.931436
USDJPY_log_RealVol                           0.772985
USDJPY_log_RealVol_HAR22                     0.988184
USDJPY_log_RealVol_HAR5                      0.860800

                           USDJPY_log_ImplVol1m_HAR5  USDJPY_log_RealVol  \
```

```
Target_EURGBP_log_RealVol              0.926819               0.808212
AUDJPY_log_ImplVol1d                   0.931799               0.851758
AUDJPY_log_ImplVol1d_HAR22             0.915543               0.779211
AUDJPY_log_ImplVol1d_HAR5              0.975895               0.844033
AUDJPY_log_ImplVol1m                   0.987832               0.893671
…                                         …                      …
USDJPY_log_ImplVol1m_HAR22             0.931436               0.772985
USDJPY_log_ImplVol1m_HAR5              1.000000               0.891416
USDJPY_log_RealVol                     0.891416               1.000000
USDJPY_log_RealVol_HAR22               0.965156               0.844413
USDJPY_log_RealVol_HAR5                0.971058               0.937203

                            USDJPY_log_RealVol_HAR22  USDJPY_log_RealVol_HAR5
Target_EURGBP_log_RealVol                   0.930606                 0.877524
AUDJPY_log_ImplVol1d                        0.911251                 0.914372
AUDJPY_log_ImplVol1d_HAR22                  0.985602                 0.860501
AUDJPY_log_ImplVol1d_HAR5                   0.982409                 0.940708
AUDJPY_log_ImplVol1m                        0.962075                 0.951568
…                                              …                        …
USDJPY_log_ImplVol1m_HAR22                  0.988184                 0.860800
USDJPY_log_ImplVol1m_HAR5                   0.965156                 0.971058
USDJPY_log_RealVol                          0.844413                 0.937203
USDJPY_log_RealVol_HAR22                    1.000000                 0.922185
USDJPY_log_RealVol_HAR5                     0.922185                 1.000000

[145 rows x 145 columns]
```

[46]:

```
X = df.iloc[:,1:].to_numpy(copy=True)
y = df.iloc[:,0].to_numpy(copy=True)

#apply Lasso path
alphas, active, coef_path_lars = lars_path(X, y, method='lasso')

#define pd data frame with active coefficients
var_sel = pd.DataFrame(coef_path_lars, index = df.columns[1:],
                       columns = onp.round(alphas,2)) #onp.vectorize(lambda x:␣
 ↪"alpha= "+str(round(x,2)))(alphas))

#keep only the variables which are nonzero when alpha=0.06
var_sel = var_sel.loc[var_sel[0.06] != 0,:]

import matplotlib.pyplot as plt
fig = plt.figure(figsize=(10,7))
ax = plt.subplot(111)

for i in range(var_sel.shape[0]):
    ax.plot(-np.log(alphas[:-1]), var_sel.iloc[i,:-1])
```

```python
plt.title("Most Relevance Lasso Coefficients for varying $l_1$-Penalty")
plt.xlabel("-log(alpha)")
plt.ylabel("coefficients")

box = ax.get_position()
ax.set_position([box.x0, box.y0, box.width * 0.99, box.height])

# Put a legend to the right of the current axis
ax.legend(var_sel.index, bbox_to_anchor=(1.0, 0.9),fontsize=9)
plt.show()
```



[47]:
```python
pd.DataFrame(var_sel.index, columns = ["Relevant Regressors"])
```

[47]:

	Relevant Regressors
0	AUDJPY_log_ImplVol1m
1	EURAUD_log_ImplVol1m
2	EURCHF_log_RealVol
3	EURGBP_log_ImplVol1m
4	EURGBP_log_RealVol_HAR5
5	EURUSD_log_RealVol
6	GBPUSD_log_RealVol

With LASSO, the AUDJPY_log_ImplVol1m is still important, which is comforting. However, this set of regressors makes more intuitive sense as they are mostly EUR and GBP currency pairs. Regardless, it was also disappointing to not get any interaction terms from the SKIM experiment.

Altogether, it seems like an area where SKIM could be used instead of including polynomial features or other indirect methods for including interaction terms.