Freni-Sterrantino et al 2017 - BYM2 connected, disconnected for Scotland Lip Cancer Dataset

Mitzi Morris

In A note on intrinsic Conditional Autoregressive models for disconnected graphs, Freni-Sterrantino et.al. show how to implement the BYM2 model for use with areal data where the graph structure of the map is not fully connected. In this notebook, we present that Stan implementation of this proposal.

The BYM2 model provides an intuitive parameterization for a GLM which has both an ICAR component ϕ which accounts for the spatial structure of the data, and a ordinaty random effects component θ for non-spatial heterogeneity. In addition, the BYM2 model has a single precision (scale) parameter σ on the combined components and a mixing parameter ρ for the amount of spatial/non-spatial variation. In order for σ to legitimately be the standard deviation of the combined components, it is critical that for each i, $\mathrm{Var}(\phi_i) \approx \mathrm{Var}(\theta_i) \approx 1$. This is done by adding a scaling factor τ to the model which scales the proportion of variance ρ. Riebler et al. recommend scaling the model so the geometric mean of these variances is 1. Because the scaling factor depends on the dataset, it comes into the model as data.

The Stan case study Spatial Models in Stan: Intrinsic Auto-Regressive Models for Areal Data for details on the ICAR, BYM, and BYM2 models.

Overview of BYM2 model for a fully connected spatial structure

When the areal map is a single, fully connected component, i.e., a graph where any node in the graph can be reached from any other node, the BYM2 model is implemented as follows.

The spatial structure and scaling factor are data inputs to the model:

```
data {
  int<lower = 0> I;  // number of nodes
  int<lower = 0> J;  // number of edges
  int<lower = 1, upper = I> edges[2, J];  // node[1, j] adjacent to
node[2, j]
  real tau; // scaling factor
```

The spatial and heterogeneous effects, combined variance, and proportion of spatial variance are model parameters:

```
parameters {
  real<lower=0, upper=1> rho; // proportion of spatial effect
that's spatially smoothed
  real<lower = 0> sigma;  // scale of spatial effects
  vector[I] theta;  // standardized heterogeneous spatial effects
  vector[I] phi;  // standardized spatially smoothed spatial
effects
```

The combined BYM2 component is computed in the `transformed parameters` block:

```
transformed parameters {
  // spatial effects (combine heterogeneous and spatially smoothed)
  vector[I] gamma = (sqrt(1 - rho) * theta + sqrt(rho / tau) * phi)
* sigma;
```

The ICAR component is implemented as log probability density function which computes the ICAR pairwise difference and imposes a soft sum-to-zero constraint:

```
real standard_icar_lpdf(vector phi, int[ , ] adjacency) {
    return 0.5 * dot_self(phi[adjacency[1,]] - phi[adjacency[2]])
      + normal_lpdf(sum(phi) | 0, 0.001 * rows(phi));
}
```

Freni-Sterrantino recommendations for a disconnected graph and Stan implementation

Freni-Sterrantino et al show how to adjust the scaling factors when the areal map is not fully connected but has at least one connected multi-node component.

1. Each connected component of size > 1 is scaled independently
2. Components of size 1 are scaled with a normal with precision K, where K is the number of components.

To extend the BYM2 model to these areal maps, we agument this model with a series of per-component masks into the node and edgelists and use Stan's multi-index operator and vectorized operations for efficient computation.

The spatial structure includes a set of arrays describing component-wise node, edgesets. The `_cts` arrays record the size of the node and edgelists for each component, the `_idx` arrays provide the indices of the members of each component.

```
 int<lower=0, upper=I> K;  // number of components in spatial graph
  int<lower=0, upper=I> K_node_cts[K];   // per-component nodes
  int<lower=0, upper=J> K_edge_cts[K];   // per-component edges
  int<lower=0, upper=I> K_node_idxs[K, I];  // rows contain per-
component node indexes
  int<lower=0, upper=J> K_edge_idxs[K, J];  // rows contain per-
component edge indexes

  vector[K] tau; // scaling factor
```

Per recommendataion 1, above, the combined spatial and random effects in the BYM2 model are computed component-wise, each with their own scaling factor. For singletons, the scaling factor `tau` is 1/K.

```
transformed parameters {
  vector[I] gamma;
```

```
    // each component scaled by tau[k]
    for (k in 1:K)
      gamma[K_node_idxs[k, 1:K_node_cts[k]]] =
              (sqrt(1 - rho) * theta[K_node_idxs[k, 1:K_node_cts[k]]]
               +
               sqrt(rho / tau) * phi[K_node_idxs[k,
1:K_node_cts[k]]])
              * sigma;
```

The ICAR `_lpdf` function puts a sum-to-zero constraint on each component. Singletons have normal spatial variance.

```
real standard_icar_disconnected_lpdf(vector phi,
                                      int[ , ] adjacency,
                                      int[ ] node_cts,
                                      int[ ] edge_cts,
                                      int[ , ] node_idxs,
                                      int[ , ] edge_idxs) {
    real total = 0;
    for (n in 1:size(node_cts)) {
      if (node_cts[n] > 1)
        total += -0.5 * dot_self(phi[adjacency[1, edge_idxs[n,
1:edge_cts[n]]]] -
                                 phi[adjacency[2, edge_idxs[n,
1:edge_cts[n]]]])
                 + normal_lpdf(sum(phi[node_idxs[n,
1:node_cts[n]]]) |
                                      0, 0.001 * node_cts[n]);
      else
          total += normal_lpdf(phi[n] | 0, 1);  // iid spatial
variance
    }
    return total;
}
```

Python packages used in this notebook, including CmdStanPy

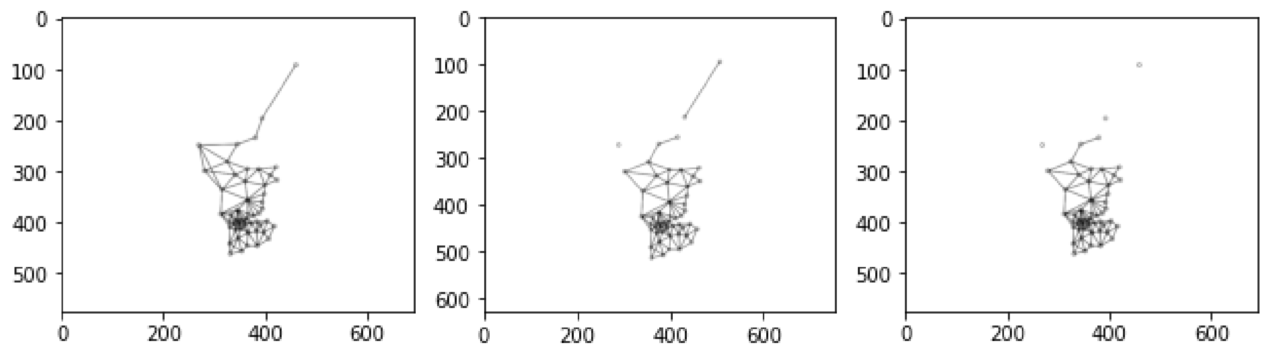
In [1]:
```
import json

import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from matplotlib import rcParams
%matplotlib inline

import numpy as np
import pandas as pd

from cmdstanpy import cmdstan_path, CmdStanModel, install_cmdstan
# install_cmdstan()  # as needed - will install latest release (as needed)
```

Areal data: the counties in Scotland, circa 1980

The canonical dataset used to test and compare different parameterizations of ICAR models is a study on the incidence of lip cancer in Scotland in the 1970s and 1980s. The data, including the names and coordinates for the counties of Scotland are available from R package SpatialEpi, dataset `scotland`.

3 of these counties are islands: the Outer Hebrides (western.isles), Shetland, and Orkney. In the canonical datasets, these islands are conntected to the mainland, so that the adjacency graph consists of a single, fully connected component. However, different maps are possible: a map with 4 components, the mainland and the 3 islands; or a map with 3 components: the mainland, a component consisting of Shetland and Orkney, and a singleton consisting of the Hebrides. The following plots demonstrate the differences:

In [2]:
```
# figure size in inches optional
rcParams['figure.figsize'] = 11 ,8
img_A = mpimg.imread('scot_connected.png')
img_B = mpimg.imread('scot_3_comp.png')
img_C = mpimg.imread('scot_islands.png')
# display images
fig, ax = plt.subplots(1,3)
ax[0].imshow(img_A);
ax[1].imshow(img_B);
ax[2].imshow(img_C);
```



Data prep: from spatial polygon to 2D array of edges

In the Stan implementation of the ICAR model, the edgelist is a 2D array of size $2 \times J$ where J is the number of edges in the graph. Each column entry in this array represents one undirected edge in the graph, where for each edge j, entries [j,1] and [j,2] index the nodes connected by that edge. Treating these are parallel arrays and using Stan's vectorized operations provides a transparent implementation of the pairwise difference formula used to compute the ICAR component.

The common format for the spatial structure of an areal dataset is as a set of shapefiles. The areal regions are described by a set of spatial polygons, i.e., a description of the shape of each region in terms of its lat,lon coordinates. The R package spdep extracts the adjacency relations as a `nb` object. We have written a set of helper functions which take the `nb` objects for each graph into the set of data structures needed by the Stan models, these are in file `bym2_helpers.R`. The helper function `nb_to_edge_array` takes the `nb` object and returns the $2 \times J$ edge array; the helper function `scaling_factor` uses the edge array to compute the geometric mean of the corresponding adjacency matrix.

The `scotland` dataset contains the shapefiles for the counties in Scotland. The fully connected graph corresponds to the data as distributed. By editing the `nb` objects, we have created the alternative maps above. The three versions of the Scotland spatial structure are in files `scotland_nbs.data.R`, `scotland_3_comp_nbs.data.R`, and `scotland_islands_nbs.data.R`. The file `munge_scotland.R` munges the data, and it has been saved as JSON data files.

Regression data: the Scotland cancer dataset

The cancer study data is:

- y : observed outcome - number of cases of lip cancer
- x : single predictor - percent of population working in agriculture, forestry, or fisheries.
- E : population

This dataset is available via several different R packages; often `x` is called `AFF`; sometime it is given as a percentage, i.e., scaled from 1 to 100; sometimes as a proportion, scaled from 0 to 1. The version of this dataset distributed as a BUGS example scales the percentage by 1/10, as does the INLA model. In order to compare the Stan results with INLA et al, in this dataset, `x`, the percenatage population, is also scaled by 1/10.

Fitting the BYM2_islands model to Scotland map as mainland component plus islands

For the Scotland map with 3 island (singleton) components, in file `scotland_islands_nbs.data.R`, we use function index_components and write_json produce the input data file `scotland_islands.data.json`. This map has the same nodes set as in the `scotland` dataset, but the neighbors object (file `scotland_nbs.data.R`) has been edited to remove all edges between islands and the mainland or each other.

In [3]:
```
with open('scotland_islands.data.json') as fd:
    islands_data = json.load(fd)

print('num nodes: {}, num edges: {}'.format(islands_data['I'], islands_data['J']
```

```
num nodes: 56, num edges: 126
```

The `index_components` function creates the per-component node and edge index arrays, and computes the vector of scaling factors `tau`:

In [4]:
```
print('num components: {}\nscaling factors: {}\nnodes per component: {}\nnode in
    islands_data['K'], islands_data['tau'], islands_data['K_node_cts'], islands_
```

```
num components: 4
scaling factors: [0.4504, 0.25, 0.25, 0.25]
nodes per component: [53, 1, 1, 1]
node indices: [[1, 2, 3, 4, 5, 7, 9, 10, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21,
22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41,
42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 0, 0, 0], [6, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0], [11, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0]]
```

We use CmdStanPy to compile and fit the model using

In [5]:
```
from cmdstanpy import cmdstan_path, CmdStanModel
bym2_islands_model = CmdStanModel(stan_file='bym2_islands.stan')
```

```
INFO:cmdstanpy:found newer exe file, not recompiling
INFO:cmdstanpy:compiled model file: /Users/mitzi/github/stan-dev/example-models/
knitr/car-iar-poisson/update_2021_02/bym2_islands
```

In [6]:
```
print(bym2_islands_model.code())
```

```
functions {
  /**
   * Return the log probability density of the specified vector of
   * coefficients under the ICAR model with unit variance, where
   * adjacency is determined by the adjacency array and the spatial
   * structure is a disconnected graph which has at least one
   * connected component.  Each connected component has a
   * soft sum-to-zero constraint.   Singleton nodes have
   * distribution normal(0, 1/sqrt(K))
   *
   * The spatial structure is described by a 2-D adjacency array
   * over the all edges in the areal map and a arrays of the
   * indices of per-component nodes and edges which are used as
   * masks into phi and the adjacency matrix.   Because the Stan
   * language lacks ragged arrays, these are all square matrices,
   * padded out with zeros; additional vectors record the number
   * of nodes and edges in each component.
   *
   * @param phi vector of varying effects
   * @param adjacency parallel arrays of indexes of adjacent elements of phi
   * @param node_cts array of sizes of per-component nodes
   * @param edge_cts array of sizes of per-component edges
   * @param node_idxs array of arrays of per_component node indexes.
   * @param edge_idxs array of arrays of per_component edge indexes.
   *
   * @return ICAR log probability density
   *
   * @reject if the the adjacency matrix does not have two rows
   * @reject if size mismatch between indexing arrays
   * @reject if size mismatch between phi and node indexes columns.
   */
  real standard_icar_disconnected_lpdf(vector phi,
                                       int[ , ] adjacency,
                                       int[ ] node_cts,
                                       int[ ] edge_cts,
                                       int[ , ] node_idxs,
                                       int[ , ] edge_idxs) {
    int num_nodes = size(phi);
    int num_edges = dims(adjacency)[2];
    int num_comps = size(edge_cts);
    if (size(adjacency) != 2)
      reject("require 2 rows for adjacency array;",
             " found rows = ", size(adjacency));
    if (!(num_nodes == dims(node_idxs)[2]
          && size(node_cts) == size(edge_cts)
          && size(node_cts) == size(node_idxs)
          && size(edge_cts) == size(edge_idxs)))
      reject("arguments have size mismatch, expecting ",
             num_comps,
             " rows for node_cts edge_cts, node_idxs, and edge_idxs,",
             num_nodes,
              " elements in phi and columns of node_idxs, and ",
              num_edges,
              " columns of edge_idxs.");

    real total = 0;
    for (n in 1:num_comps) {
      if (node_cts[n] > 1)
        total += -0.5 * dot_self(phi[adjacency[1, edge_idxs[n, 1:edge_cts[n]]]]
-
                                 phi[adjacency[2, edge_idxs[n, 1:edge_cts[n]]]])
          + normal_lpdf(sum(phi[node_idxs[n, 1:node_cts[n]]]) | 0, 0.001 * node_
cts[n]);
      else
        total += normal_lpdf(phi[node_idxs[n, 1]] | 0, 1);
    }
    return total;
  }
}
data {
  // spatial structure
  int<lower = 0> I;  // number of nodes
  int<lower = 0> J;  // number of edges
  int<lower = 1, upper = I> edges[2, J];  // node[1, j] adjacent to node[2, j]

  int<lower=0, upper=I> K;  // number of components in spatial graph
  int<lower=0, upper=I> K_node_cts[K];   // per-component nodes
  int<lower=0, upper=J> K_edge_cts[K];   // per-component edges
  int<lower=0, upper=I> K_node_idxs[K, I];  // rows contain per-component node i
ndexes
  int<lower=0, upper=J> K_edge_idxs[K, J];  // rows contain per-component edge i
ndexes

  vector[K] tau; // scaling factor

  int<lower=0> y[I];              // count outcomes
  vector<lower=0>[I] E;           // exposure
  vector[I] x;                 // predictor
}
transformed data {
  vector[I] log_E = log(E);
}
parameters {
  real alpha;            // intercept
  real beta;       // covariates

  // spatial effects
  real<lower=0, upper=1> rho; // proportion unstructured vs. spatially structure
d variance
  real<lower = 0> sigma;  // scale of spatial effects
  vector[I] theta;  // standardized heterogeneous spatial effects
  vector[I] phi;  // standardized spatially smoothed spatial effects
}
transformed parameters {
  vector[I] gamma;
  for (k in 1:K)
    gamma[K_node_idxs[k, 1:K_node_cts[k]]] =
      (sqrt(1 - rho) * theta[K_node_idxs[k, 1:K_node_cts[k]]]
       +
       sqrt(rho / tau[k]) * phi[K_node_idxs[k, 1:K_node_cts[k]]])
      * sigma;
}
model {
  y ~ poisson_log(log_E + alpha + x * beta + gamma * sigma);  // co-variates

  alpha ~ normal(0, 1);
  beta ~ normal(0, 1);

  // spatial hyperpriors and priors
  sigma ~ normal(0, 1);
  rho ~ normal(0, 1);
  theta ~ normal(0, 1);
  phi ~ standard_icar_disconnected(edges, K_node_cts, K_edge_cts, K_node_idxs, K
_edge_idxs);
}
generated quantities {
  // posterior predictive checks
  vector[I] eta = log_E + alpha + x * beta + gamma * sigma;
  vector[I] y_prime = exp(eta);
  //   int y_rep[I,10];
  //   for (j in 1:10) {
  //     if (max(eta) > 20) {
  //       // avoid overflow in poisson_log_rng
  //       print("max eta too big: ", max(eta));
  //       for (i in 1:I)
  //    y_rep[i,j] = -1;
  //     } else {
  //       for (i in 1:I)
  //         y_rep[i,j] = poisson_log_rng(eta[i]);
  //     }
  //   }
  real logit_rho = log(rho / (1.0 - rho));
}
```

We fit the model using the data dictionary `islands_data` , alternatively, one can specify file `scotland_islands.data.json` . For reproducibility, we specify the seed supplied to the Stan pseudo-random number generator used by the sampler.

In [7]:
```
bym2_islands_fit = bym2_islands_model.sample(data=islands_data, seed=12345)
```

```
INFO:cmdstanpy:start chain 1
INFO:cmdstanpy:start chain 2
INFO:cmdstanpy:start chain 3
INFO:cmdstanpy:start chain 4
INFO:cmdstanpy:finish chain 1
INFO:cmdstanpy:finish chain 4
INFO:cmdstanpy:finish chain 3
INFO:cmdstanpy:finish chain 2
```

The CmdStanMCMC method `summary` wraps the CmdStan `stansummary` utility; it returns a pandas DataFrame object, one row per summary output row for the joint log probability density `lp__` and for the values of all variables in the Stan program.

In [8]:
```
summary = bym2_islands_fit.summary()
summary
```

Out[8]:

name	Mean	MCSE	StdDev	5%	50%	95%	N_Eff	N_Eff/s	R_hat
lp__	750.00	0.3200	9.000	730.00	750.00	760.000	820.00	54.00	1.0
alpha	-0.30	0.0029	0.130	-0.50	-0.30	-0.086	1800.00	120.00	1.0
beta	0.42	0.0033	0.130	0.19	0.43	0.630	1700.00	110.00	1.0
rho	0.76	0.0060	0.170	0.44	0.79	0.980	820.00	55.00	1.0

name	Mean	MCSE	StdDev	5%	50%	95%	N_Eff	N_Eff/s	R_hat
sigma	0.70	0.0019	0.058	0.61	0.70	0.800	950.00	63.00	1.0
...	...	...	...	...	...	...	...	...	...
y_prime[53]	2.30	0.0094	0.840	1.20	2.20	3.900	8043.00	534.00	1.0
y_prime[54]	2.90	0.0120	0.990	1.50	2.70	4.600	6501.00	432.00	1.0
y_prime[55]	3.50	0.0180	1.200	1.80	3.40	5.700	4675.00	311.00	1.0
y_prime[56]	1.40	0.0064	0.490	0.72	1.30	2.300	5977.00	397.00	1.0
logit_rho	1.53	0.0500	1.310	-0.24	1.35	3.960	819.88	54.47	1.0

286 rows × 9 columns

The R_hat and N_Eff (number of effective samples) in the summary report indicate that the model has fit the data. To further check the fit, we run the `diagnose` method, which wraps CmdStan's `diagnose` utility.

In [9]:
```
bym2_islands_fit.diagnose()
```

```
INFO:cmdstanpy:Processing csv files: /var/folders/db/4jnggnf549s42z50bd61jskm000
0gq/T/tmpki148d8f/bym2_islands-202102231522-1-aleetrec.csv, /var/folders/db/4jng
gnf549s42z50bd61jskm0000gq/T/tmpki148d8f/bym2_islands-202102231522-2-iyn78g0k.cs
v, /var/folders/db/4jnggnf549s42z50bd61jskm0000gq/T/tmpki148d8f/bym2_islands-202
102231522-3-ade_yeag.csv, /var/folders/db/4jnggnf549s42z50bd61jskm0000gq/T/tmpki
148d8f/bym2_islands-202102231522-4-0m0l_i_o.csv

Checking sampler transitions treedepth.
Treedepth satisfactory for all transitions.

Checking sampler transitions for divergences.
No divergent transitions found.

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

Effective sample size satisfactory.

Split R-hat values satisfactory all parameters.

Processing complete, no problems detected.
```

Out[9]:
```
'Processing csv files: /var/folders/db/4jnggnf549s42z50bd61jskm0000gq/T/tmpki148
d8f/bym2_islands-202102231522-1-aleetrec.csv, /var/folders/db/4jnggnf549s42z50bd
61jskm0000gq/T/tmpki148d8f/bym2_islands-202102231522-2-iyn78g0k.csv, /var/folder
s/db/4jnggnf549s42z50bd61jskm0000gq/T/tmpki148d8f/bym2_islands-202102231522-3-ad
e_yeag.csv, /var/folders/db/4jnggnf549s42z50bd61jskm0000gq/T/tmpki148d8f/bym2_is
lands-202102231522-4-0m0l_i_o.csv\n\nChecking sampler transitions treedepth.\nTr
eedepth satisfactory for all transitions.\n\nChecking sampler transitions for di
vergences.\nNo divergent transitions found.\n\nChecking E-BFMI - sampler transit
ions HMC potential energy.\nE-BFMI satisfactory for all transitions.\n\nEffectiv
e sample size satisfactory.\n\nSplit R-hat values satisfactory all parameters.\n
\nProcessing complete, no problems detected.'
```

The diagnose command detects *potential* problems by examining the set of sampler diagnostic variables; these are the initial columns of the Stan CSV output file, which have column labels which end in `__`. For example, to further check the treedepth, which is an integer output between 1 and sampler configuration argument `max_treedepth` , default 10, we can access this column from the output and do a quick summary of the per-draw treedepth counts:

In [10]:
```
bym2_islands_draws = bym2_islands_fit.draws(concat_chains=True)
treedepth_idx = bym2_islands_fit.column_names.index('treedepth__')
treedepths = np.bincount(bym2_islands_draws[:, treedepth_idx].astype('int'))
for idx, val in enumerate(treedepths):
    if val > 0:
        print(idx, val)
```

```
7 7
8 3993
```

To get the summary statistics for a Stan program variable across all chains, we can access individual rows of the pandas DataFrame object using `iloc`

In [11]:
```
names = list(summary.index)
rows = [names.index(name) for name in names if name.startswith('y_prime[')]
summary.iloc[rows,:][5:11]
```

Out[11]:

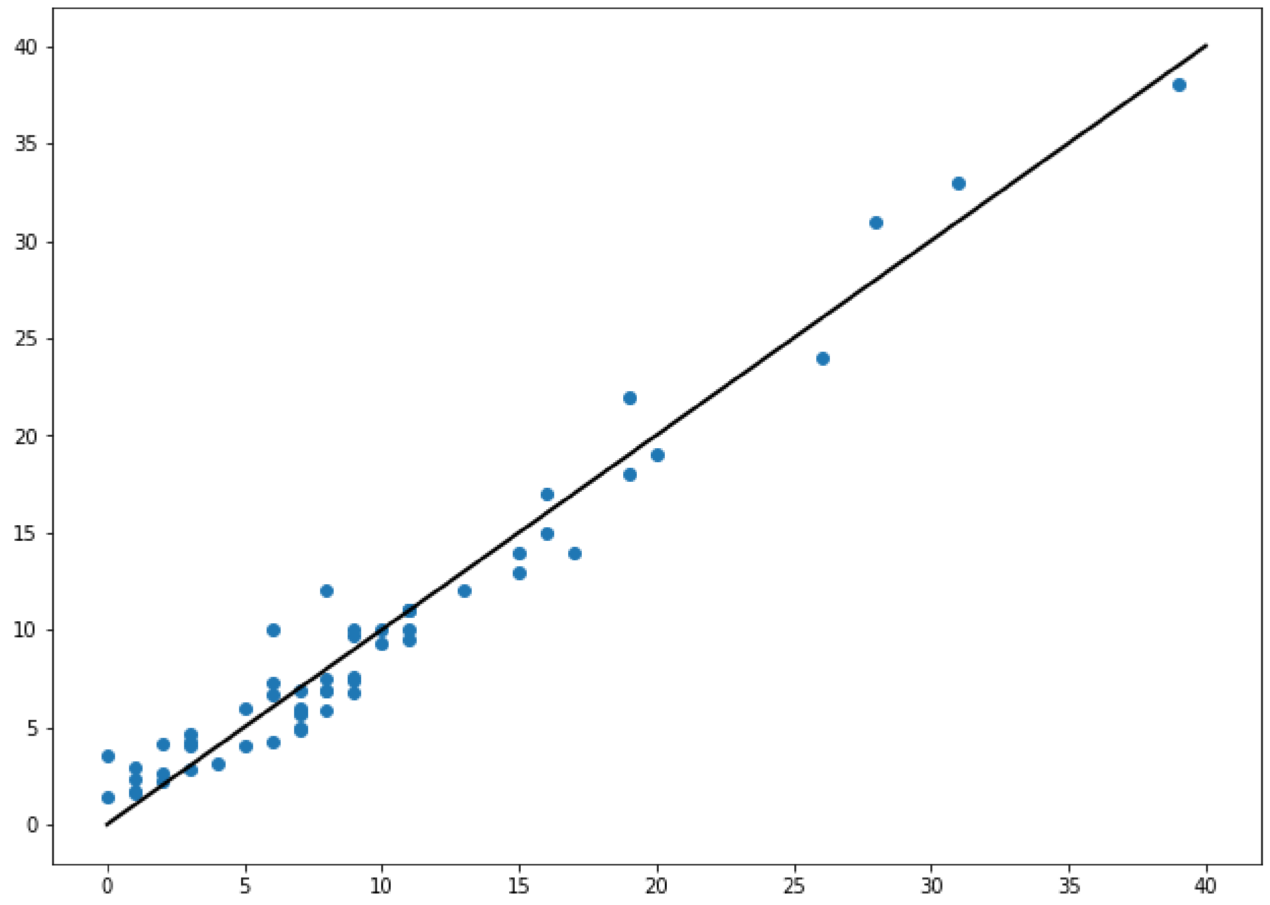
name	Mean	MCSE	StdDev	5%	50%	95%	N_Eff	N_Eff/s	R_hat
y_prime[6]	7.5	0.031	2.5	3.9	7.1	12.0	6520.0	433.0	1.0
y_prime[7]	24.0	0.067	4.5	17.0	24.0	32.0	4544.0	302.0	1.0
y_prime[8]	5.8	0.026	2.1	2.8	5.5	9.7	6397.0	425.0	1.0
y_prime[9]	4.3	0.016	1.4	2.4	4.1	6.8	7395.0	491.0	1.0
y_prime[10]	19.0	0.053	3.9	14.0	19.0	26.0	5233.0	348.0	1.0
y_prime[11]	12.0	0.046	3.3	6.8	11.0	18.0	5194.0	345.0	1.0

In [12]:
```
islands_data['y'][5:11]
```

Out[12]:
```
[8, 26, 7, 6, 20, 13]
```

In [13]:
```
# checking model estimates for y against input data
y_primes = summary.iloc[rows, 0]
plt.scatter(islands_data['y'], y_primes)
plt.plot([0, 40], [0, 40], color = 'black', linewidth = 2)
```

Out[13]:
```
[<matplotlib.lines.Line2D at 0x7ffe20d9f850>]
```



Fit connected graph on Scotland Lip cancer dataset with BYM2 model implemented in Stan.

In [14]:
```
from cmdstanpy import cmdstan_path, CmdStanModel, install_cmdstan
# install_cmdstan()  # as needed - will install latest release (as needed)
```

The dataset `scot_connected.data.json` contains the cancer dataset together with the spatial structure. The spatial structure is comprised of:

- I: `int<lower = 0> I;  // number of nodes`
- J: `int<lower = 0> J;  // number of edges`
- edges: `int<lower = 1, upper = I> edges[2, J];  // node[1, j] adjacent to node[2, j]`
- tau: `real tau; // scaling factor`

In [15]:
```
with open('scotland_connected.data.json') as fd:
    connected_data = json.load(fd)

print('num nodes: {}, num edges: {}'.format(connected_data['I'], connected_data[
print('scaling factor: {}\nedges: {}'.format(connected_data['tau'], connected_da
```

```
num nodes: 56, num edges: 132
scaling factor: 0.4853
edges: [[1, 1, 1, 1, 2, 2, 3, 3, 4, 4, 4, 5, 5, 5, 5, 6, 7, 7, 7, 7, 9, 9, 9, 9,
9, 10, 10, 11, 13, 13, 14, 14, 14, 15, 15, 15, 16, 16, 16, 16, 17, 17, 18, 18, 1
8, 18, 18, 20, 21, 21, 23, 23, 23, 23, 23, 24, 24, 24, 24, 24, 24, 24, 24, 25, 2
5, 26, 26, 26, 27, 27, 27, 28, 28, 29, 29, 29, 30, 30, 30, 30, 30, 31, 31, 31, 3
1, 32, 33, 33, 34, 34, 34, 34, 34, 34, 34, 35, 35, 36, 36, 36, 37, 37, 38, 38, 3
8, 38, 38, 39, 39, 40, 40, 40, 41, 41, 41, 42, 42, 44, 44, 45, 46, 46, 47, 47, 4
7, 48, 49, 49, 49, 51, 52, 55], [5, 9, 11, 19, 7, 10, 6, 12, 18, 20, 28, 11, 12,
13, 19, 8, 10, 13, 16, 17, 11, 17, 19, 23, 29, 16, 22, 12, 17, 19, 31, 32, 35, 2
5, 29, 50, 17, 21, 22, 29, 19, 29, 20, 28, 33, 55, 56, 55, 29, 50, 29, 34, 36, 3
7, 39, 27, 30, 31, 44, 47, 48, 55, 56, 26, 29, 29, 42, 43, 31, 32, 55, 33, 45, 3
4, 43, 50, 38, 42, 44, 45, 56, 32, 35, 46, 47, 35, 45, 56, 39, 40, 42, 43, 51, 5
2, 54, 37, 46, 37, 39, 41, 41, 46, 42, 44, 49, 51, 54, 40, 41, 41, 49, 52, 46, 4
9, 53, 43, 51, 48, 49, 56, 47, 53, 48, 49, 53, 49, 52, 53, 54, 54, 54, 56]]
```

In [16]:

```
bym2_model = CmdStanModel(stan_file='bym2.stan')
bym2_fit = bym2_model.sample(data=connected_data, seed=12345)
bym2_fit.summary()
```

```
INFO:cmdstanpy:found newer exe file, not recompiling
INFO:cmdstanpy:compiled model file: /Users/mitzi/github/stan-dev/example-models/
knitr/car-iar-poisson/update_2021_02/bym2
INFO:cmdstanpy:start chain 1
INFO:cmdstanpy:start chain 2
INFO:cmdstanpy:start chain 3
INFO:cmdstanpy:start chain 4
INFO:cmdstanpy:finish chain 3
INFO:cmdstanpy:finish chain 1
INFO:cmdstanpy:finish chain 2
INFO:cmdstanpy:finish chain 4
```

Out[16]:

	Mean	MCSE	StdDev	5%	50%	95%	N_Eff	N_Eff/s	R_hat
name									
lp__	750.00	0.3000	9.000	740.00	750.00	770.000	930.00	77.00	1.00
alpha	-0.23	0.0030	0.130	-0.45	-0.23	-0.014	1900.00	160.00	1.00
beta	0.38	0.0031	0.130	0.16	0.38	0.600	1800.00	150.00	1.00
rho	0.79	0.0061	0.160	0.48	0.82	0.980	680.00	57.00	1.00
sigma	0.71	0.0018	0.057	0.62	0.71	0.810	1100.00	90.00	1.00
...	...	...	...	...	...	...	...	...	...
y_prime[53]	2.30	0.0110	0.860	1.20	2.20	3.900	6035.00	505.00	1.00
y_prime[54]	2.90	0.0110	0.990	1.50	2.70	4.700	7452.00	623.00	1.00
y_prime[55]	3.40	0.0170	1.200	1.70	3.30	5.500	4697.00	393.00	1.00
y_prime[56]	1.40	0.0059	0.470	0.71	1.30	2.200	6320.00	529.00	1.00
logit_rho	1.68	0.0500	1.300	-0.10	1.52	4.000	675.61	56.52	1.01

286 rows × 9 columns

In [17]:

```
bym2_fit.diagnose()
```

```
INFO:cmdstanpy:Processing csv files: /var/folders/db/4jnggnf549s42z50bd61jskm000
0gq/T/tmpki148d8f/bym2-202102231522-1-udl_ikgj.csv, /var/folders/db/4jnggnf549s4
2z50bd61jskm0000gq/T/tmpki148d8f/bym2-202102231522-2-uzopf0ww.csv, /var/folders/
db/4jnggnf549s42z50bd61jskm0000gq/T/tmpki148d8f/bym2-202102231522-3-kdbvegtq.cs
v, /var/folders/db/4jnggnf549s42z50bd61jskm0000gq/T/tmpki148d8f/bym2-20210223152
2-4-glp67e71.csv

Checking sampler transitions treedepth.
Treedepth satisfactory for all transitions.

Checking sampler transitions for divergences.
```

```
No divergent transitions found.

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

Effective sample size satisfactory.

Split R-hat values satisfactory all parameters.

Processing complete, no problems detected.
```

```
Out[17]: 'Processing csv files: /var/folders/db/4jnggnf549s42z50bd61jskm0000gq/T/tmpki148
d8f/bym2-202102231522-1-udl_ikgj.csv, /var/folders/db/4jnggnf549s42z50bd61jskm00
00gq/T/tmpki148d8f/bym2-202102231522-2-uzopf0ww.csv, /var/folders/db/4jnggnf549s
42z50bd61jskm0000gq/T/tmpki148d8f/bym2-202102231522-3-kdbvegtq.csv, /var/folder
s/db/4jnggnf549s42z50bd61jskm0000gq/T/tmpki148d8f/bym2-202102231522-4-glp67e71.c
sv\n\nChecking sampler transitions treedepth.\nTreedepth satisfactory for all tr
ansitions.\n\nChecking sampler transitions for divergences.\nNo divergent transi
tions found.\n\nChecking E-BFMI - sampler transitions HMC potential energy.\nE-B
FMI satisfactory for all transitions.\n\nEffective sample size satisfactory.\n\n
Split R-hat values satisfactory all parameters.\n\nProcessing complete, no probl
ems detected.'
```

Mainland plus 1 singleton, and combined Shetland + Orkney component

For the Scotland map with 3 components, in file `scotland_3_comps_nbs.data.R` , the islands of Shetland and Orkney have been combined. This dataset exists to test additional possible kinds of maps, but provides no additional insights on the model or the domain data.

```
In [ ]:
```