

Appendix A1

Stan code to implement the Markov Chain Monte Carlo (MCMC) sampling for model 1

The function `sparse_iar_lpdf()` was devised by Joseph (2016)

```
functions {  
  /**  
   * Return the log probability of a proper intrinsic autoregressive (IAR) prior  
   * with a sparse representation for the adjacency matrix  
   *  
   * @param phi Vector containing the parameters with a IAR prior  
   * @param tau Precision parameter for the IAR prior (real)  
   * @param W_sparse Sparse representation of adjacency matrix (int array)  
   * @param n Length of phi (int)  
   * @param W_n Number of adjacent pairs (int)  
   * @param D_sparse Number of neighbors for each location (vector)  
   *  
   * @return Log probability density of IAR prior up to additive constant  
   */  
  real sparse_iar_lpdf(vector phi, real tau,  
    int[,] W_sparse, vector D_sparse, int n, int W_n) {  
    row_vector[n] phit_D; // phi' * D  
    row_vector[n] phit_W; // phi' * W  
    vector[n] ldet_terms;  
    phit_D = (phi .* D_sparse);  
    phit_W = rep_row_vector(0, n);  
    for (i in 1:W_n) {  
      phit_W[W_sparse[i, 1]] = phit_W[W_sparse[i, 1]] + phi[W_sparse[i, 2]];  
      phit_W[W_sparse[i, 2]] = phit_W[W_sparse[i, 2]] + phi[W_sparse[i, 1]];  
    }  
  }  
}
```

```

1      }
2      return 0.5 * ((n-1) * log(tau) - tau * (phit_D * phi - (phit_W * phi)));
3      }
4  }
5
6  data
7  {
8      int < lower = 1 > nData;
9      int < lower = 0 > damage[ nData ];
10     vector[ nData ] windSpeed;
11     int < lower = 1 > nForWindDir;
12     int < lower = 1, upper = nForWindDir > windDirIndex[ nData ];
13     matrix< lower = 0, upper = 1 >[ nForWindDir, nForWindDir ] WForWindDir; // adjacency
14 matrix
15     int < lower = 1 > W_nForWindDir;           // number of adjacent region pairs
16     int < lower = 1 > nForSeason;
17     int < lower = 1, upper = nForSeason > seasonIndex[ nData ];
18     matrix< lower = 0, upper = 1 >[ nForSeason, nForSeason ] WForSeason; // adjacency matrix
19     int < lower = 1 > W_nForSeason;           // number of adjacent region pairs
20     int < lower = 1 > nForDecade;
21     int < lower = 1, upper = nForDecade > decadeIndex[ nData ];
22     // real < lower = 0, upper = 100 > sigmaFirstRDecade;
23     int nObsMeanVol;
24     vector[ nObsMeanVol ] obsMeanVol;
25     int decadeWithMeanVol[ nObsMeanVol ];
26     int < lower = 0, upper = 100 > nWindSpeedForPred;
27     vector[ nWindSpeedForPred ] windSpeedForPred;
28     int < lower = 0, upper = 100 > nMeanVolForPred;
29     vector[ nMeanVolForPred ] meanVolForPred;
30 }

```

```

1
2 transformed data
3 {
4   int W_sparseForWindDir[ W_nForWindDir, 2 ]; // adjacency pairs
5   vector[ nForWindDir ] D_sparseForWindDir; // diagonal of D (number of neighbors for each
6 site)
7   int W_sparseForSeason[ W_nForSeason, 2 ]; // adjacency pairs
8   vector[ nForSeason ] D_sparseForSeason; // diagonal of D (number of neighbors for each site)
9   real mWindSpeedForPred;
10  real mMeanVolForPred;
11
12  mWindSpeedForPred = mean( windSpeedForPred );
13  mMeanVolForPred = mean( meanVolForPred );
14
15  { // generate sparse representation for W
16    int counter;
17    counter = 1;
18    // loop over upper triangular part of W to identify neighbor pairs
19    for ( i in 1:( nForWindDir - 1 ) )
20    {
21      for ( j in ( i + 1 ):nForWindDir )
22      {
23        if ( WForWindDir[i, j] == 1 )
24        {
25          W_sparseForWindDir[ counter, 1 ] = i;
26          W_sparseForWindDir[ counter, 2 ] = j;
27          counter = counter + 1;
28        }
29      }
30    }

```

```

1  }
2  for (i in 1:nForWindDir)
3    D_sparseForWindDir[ i ] = sum( WForWindDir[ i ] );
4  {
5    vector[ nForWindDir ] invsqrtDForWindDir;
6    for ( i in 1:nForWindDir )
7      {
8        invsqrtDForWindDir[ i ] = 1 / sqrt( D_sparseForWindDir[ i ] );
9      }
10 }
11
12 { // generate sparse representation for W
13   int counter;
14   counter = 1;
15   // loop over upper triangular part of W to identify neighbor pairs
16   for ( i in 1:( nForSeason - 1 ) )
17     {
18       for ( j in ( i + 1 ):nForSeason )
19         {
20           if ( WForSeason[i, j] == 1 )
21             {
22               W_sparseForSeason[ counter, 1 ] = i;
23               W_sparseForSeason[ counter, 2 ] = j;
24               counter = counter + 1;
25             }
26         }
27     }
28 }
29 for (i in 1:nForSeason)
30   D_sparseForSeason[ i ] = sum( WForSeason[ i ] );

```

```

1  {
2    vector[ nForSeason ] invsqrtDForSeason;
3    for ( i in 1:nForSeason )
4    {
5      invsqrtDForSeason[ i ] = 1 / sqrt( D_sparseForSeason[ i ] );
6    }
7  }
8  }
9
10 parameters
11 {
12   real < lower = -100, upper = 100 > b0;
13   real < lower = -100, upper = 100 > bForWindDir;
14   vector[ nForWindDir ] phi_unscaledForWindDir;
15   real < lower = 0, upper = 100 > sigmaForWindDir;
16   vector[ nForSeason ] phi_unscaledForSeason;
17   real < lower = 0, upper = 100 > sigmaForSeason;
18   real bForMeanVol;
19   vector < lower = 0, upper = 500 > [ nForDecade ] estMeanVol;
20   real < lower = 0, upper = 100 > sigmaForDecade;
21   real < lower = 0, upper = 100 > sigmaForObsMeanVol;
22 }
23
24 transformed parameters
25 {
26   vector[ nForWindDir ] phiForWindDir; // brute force centering
27   vector[ nForSeason ] phiForSeason; // brute force centering
28   // vector[ nData ] linearPredictor;
29   vector[ nData ] p;
30   real tauForWindDir;

```

```

1   real tauForSeason;
2
3   phiForWindDir = phi_unscaledForWindDir - mean( phi_unscaledForWindDir );
4   phiForSeason = phi_unscaledForSeason - mean( phi_unscaledForSeason );
5   // linearPredictor = b1 * windSpeed + phiForWindDir[ windDirIndex ] +
6   phiForSeason[ seasonIndex ] + rDecade[ decadeIndex ];
7   p = inv_logit( b0 + bForWindDir * windSpeed + bForMeanVol * estMeanVol[ decadeIndex ] +
8   phiForWindDir[ windDirIndex ] + phiForSeason[ seasonIndex ] );
9   tauForWindDir = 1 / ( sigmaForWindDir * sigmaForWindDir );
10  tauForSeason = 1 / ( sigmaForSeason * sigmaForSeason );
11  }
12
13  model
14  {
15    phi_unscaledForWindDir ~
16    sparse_iar( tauForWindDir, W_sparseForWindDir, D_sparseForWindDir, nForWindDir,
17  W_nForWindDir );
18    phi_unscaledForSeason ~
19    sparse_iar( tauForSeason, W_sparseForSeason, D_sparseForSeason, nForSeason,
20  W_nForSeason );
21
22    estMeanVol[ 1 ] ~ normal( 0, 100 );
23    estMeanVol[ 2:nForDecade ] ~ normal( estMeanVol[ 1:( nForDecade - 1 ) ], sigmaForDecade );
24    obsMeanVol ~ normal( estMeanVol[ decadeWithMeanVol ], sigmaForObsMeanVol );
25    damage ~ bernoulli( p );
26  }
27
28  generated quantities
29  {
30    vector [ nWindSpeedForPred ] predPForWindSpeed;

```

```

1  vector [ nForWindDir ] predPForWindDir;
2  vector [ nForSeason ] predPForSeason;
3  vector [ nMeanVolForPred ] predPForMeanVol;
4  vector [ nData ] log_lik;
5
6  for ( i in 1:nData )
7  {
8    log_lik[ i ] = bernoulli_lpmf( damage[ i ] | p[ i ] );
9  }
10
11  predPForWindSpeed = inv_logit( b0 + bForWindDir * windSpeedForPred + bForMeanVol *
12  mMeanVolForPred );
13  predPForWindDir = inv_logit( b0 + bForWindDir * mWindSpeedForPred + bForMeanVol *
14  mMeanVolForPred + phiForWindDir );
15  predPForSeason = inv_logit( b0 + bForWindDir * mWindSpeedForPred + bForMeanVol *
16  mMeanVolForPred + phiForSeason );
17  predPForMeanVol = inv_logit( b0 + bForWindDir * mWindSpeedForPred + bForMeanVol *
18  meanVolForPred );
19  }

```

21 **References**

22
23 Joseph M, 2016. Exact sparse CAR models in Stan. [https://mc-](https://mc-stan.org/users/documentation/case-studies/mbjoseph-CARStan.html)
24 [stan.org/users/documentation/case-studies/mbjoseph-CARStan.html](https://mc-stan.org/users/documentation/case-studies/mbjoseph-CARStan.html). [1 May 2019]

25
26

Appendix A2

Stan code to implement the Markov Chain Monte Carlo (MCMC) sampling for model 2

The function `sparse_iar_lpdf()` was devised by Joseph (2016)

```
functions {  
  /**  
   * Return the log probability of a proper intrinsic autoregressive (IAR) prior  
   * with a sparse representation for the adjacency matrix  
   *  
   * @param phi Vector containing the parameters with a IAR prior  
   * @param tau Precision parameter for the IAR prior (real)  
   * @param W_sparse Sparse representation of adjacency matrix (int array)  
   * @param n Length of phi (int)  
   * @param W_n Number of adjacent pairs (int)  
   * @param D_sparse Number of neighbors for each location (vector)  
   *  
   * @return Log probability density of IAR prior up to additive constant  
   */  
  real sparse_iar_lpdf(vector phi, real tau,  
    int[,] W_sparse, vector D_sparse, int n, int W_n) {  
    row_vector[n] phit_D; // phi' * D  
    row_vector[n] phit_W; // phi' * W  
    vector[n] ldet_terms;  
    phit_D = (phi .* D_sparse);  
    phit_W = rep_row_vector(0, n);  
    for (i in 1:W_n) {  
      phit_W[W_sparse[i, 1]] = phit_W[W_sparse[i, 1]] + phi[W_sparse[i, 2]];  
      phit_W[W_sparse[i, 2]] = phit_W[W_sparse[i, 2]] + phi[W_sparse[i, 1]];  
    }  
  }  
}
```

```

1      }
2      return 0.5 * ((n-1) * log(tau) - tau * (phit_D * phi - (phit_W * phi)));
3      }
4  }
5
6  data
7  {
8      int < lower = 1 > nData;
9      int < lower = 0 > damage[ nData ];
10     vector[ nData ] windSpeed;
11     int < lower = 1 > nForWindDir;
12     int < lower = 1, upper = nForWindDir > windDirIndex[ nData ];
13     matrix< lower = 0, upper = 1 >[ nForWindDir, nForWindDir ] WForWindDir; // adjacency
14 matrix
15     int < lower = 1 > W_nForWindDir;           // number of adjacent region pairs
16     int < lower = 1 > nForSeason;
17     int < lower = 1, upper = nForSeason > seasonIndex[ nData ];
18     matrix< lower = 0, upper = 1 >[ nForSeason, nForSeason ] WForSeason; // adjacency matrix
19     int < lower = 1 > W_nForSeason;           // number of adjacent region pairs
20     int < lower = 1 > nForDecade;
21     int < lower = 1, upper = nForDecade > decadeIndex[ nData ];
22     // real < lower = 0, upper = 100 > sigmaFirstRDecade;
23     int < lower = 0, upper = 100 > nWindSpeedForPred;
24     vector[ nWindSpeedForPred ] windSpeedForPred;
25 }
26
27 transformed data
28 {
29     int W_sparseForWindDir[ W_nForWindDir, 2 ]; // adjacency pairs

```

```

1  vector[ nForWindDir ] D_sparseForWindDir;  // diagonal of D (number of neighbors for each
2  site)
3  int W_sparseForSeason[ W_nForSeason, 2 ]; // adjacency pairs
4  vector[ nForSeason ] D_sparseForSeason;  // diagonal of D (number of neighbors for each site)
5  real mWindSpeedForPred;
6
7  mWindSpeedForPred = mean( windSpeedForPred );
8
9  { // generate sparse representation for W
10   int counter;
11   counter = 1;
12   // loop over upper triangular part of W to identify neighbor pairs
13   for ( i in 1:( nForWindDir - 1 ) )
14   {
15     for ( j in ( i + 1 ):nForWindDir )
16     {
17       if ( WForWindDir[i, j] == 1 )
18       {
19         W_sparseForWindDir[ counter, 1 ] = i;
20         W_sparseForWindDir[ counter, 2 ] = j;
21         counter = counter + 1;
22       }
23     }
24   }
25 }
26 for (i in 1:nForWindDir)
27   D_sparseForWindDir[ i ] = sum( WForWindDir[ i ] );
28 {
29   vector[ nForWindDir ] invsqrtDForWindDir;
30   for ( i in 1:nForWindDir )

```

```

1      {
2      invsqrtDForWindDir[ i ] = 1 / sqrt( D_sparseForWindDir[ i ] );
3      }
4  }
5
6  { // generate sparse representation for W
7      int counter;
8      counter = 1;
9      // loop over upper triangular part of W to identify neighbor pairs
10     for ( i in 1:( nForSeason - 1 ) )
11     {
12         for ( j in ( i + 1 ):nForSeason )
13         {
14             if ( WForSeason[i, j] == 1 )
15             {
16                 W_sparseForSeason[ counter, 1 ] = i;
17                 W_sparseForSeason[ counter, 2 ] = j;
18                 counter = counter + 1;
19             }
20         }
21     }
22 }
23 for (i in 1:nForSeason)
24     D_sparseForSeason[ i ] = sum( WForSeason[ i ] );
25 {
26     vector[ nForSeason ] invsqrtDForSeason;
27     for ( i in 1:nForSeason )
28     {
29         invsqrtDForSeason[ i ] = 1 / sqrt( D_sparseForSeason[ i ] );
30     }

```

```

1    }
2    }
3
4    parameters
5    {
6        real < lower = -100, upper = 100 > b1;
7        vector[ nForWindDir ] phi_unscaledForWindDir;
8        real < lower = 0, upper = 100 > sigmaForWindDir;
9        vector[ nForSeason ] phi_unscaledForSeason;
10       real < lower = 0, upper = 100 > sigmaForSeason;
11       vector [ nForDecade ] rDecade;
12       real < lower = 0, upper = 100 > sigmaForDecade;
13   }
14
15   transformed parameters
16   {
17       vector[ nForWindDir ] phiForWindDir; // brute force centering
18       vector[ nForSeason ] phiForSeason; // brute force centering
19       // vector[ nData ] linearPredictor;
20       vector[ nData ] p;
21       real tauForWindDir;
22       real tauForSeason;
23
24       phiForWindDir = phi_unscaledForWindDir - mean( phi_unscaledForWindDir );
25       phiForSeason = phi_unscaledForSeason - mean( phi_unscaledForSeason );
26       // linearPredictor = b1 * windSpeed + phiForWindDir[ windDirIndex ] +
27       phiForSeason[ seasonIndex ] + rDecade[ decadeIndex ];
28       p = inv_logit( b1 * windSpeed + phiForWindDir[ windDirIndex ] +
29       phiForSeason[ seasonIndex ] + rDecade[ decadeIndex ] );
30       tauForWindDir = 1 / ( sigmaForWindDir * sigmaForWindDir );

```

```

1    tauForSeason = 1 / ( sigmaForSeason * sigmaForSeason );
2  }
3
4  model
5  {
6    phi_unscaledForWindDir ~
7      sparse_iar( tauForWindDir, W_sparseForWindDir, D_sparseForWindDir, nForWindDir,
8    W_nForWindDir );
9    phi_unscaledForSeason ~
10     sparse_iar( tauForSeason, W_sparseForSeason, D_sparseForSeason, nForSeason,
11   W_nForSeason );
12
13    rDecade[ 1 ] ~ normal( 0, 100 );
14    rDecade[ 2:nForDecade ] ~ normal( rDecade[ 1:( nForDecade - 1 ) ], sigmaForDecade );
15    damage ~ bernoulli( p );
16    // target += bernoulli_logit_lpmf( damage | linearPredictor );
17  }
18
19  generated quantities
20  {
21    vector [ nWindSpeedForPred ] predPForWindSpeed;
22    vector [ nForWindDir ] predPForWindDir;
23    vector [ nForSeason ] predPForSeason;
24    vector [ nForDecade ] predPForDecade;
25    vector [ nData ] log_lik;
26
27    for ( i in 1:nData )
28    {
29      log_lik[ i ] = bernoulli_lpmf( damage[ i ] | p[ i ] );
30    }

```

```

1  predPForWindSpeed = inv_logit( b1 * windSpeedForPred + mean( rDecade ) );
2  predPForWindDir = inv_logit( b1 * mWindSpeedForPred + mean( rDecade ) +
3  phiForWindDir );
4  predPForSeason = inv_logit( b1 * mWindSpeedForPred + mean( rDecade ) + phiForSeason );
5  predPForDecade = inv_logit( b1 * mWindSpeedForPred + rDecade );
6  }

```

7

8 **References**

9

10 Joseph M, 2016. Exact sparse CAR models in Stan. [https://mc-](https://mc-stan.org/users/documentation/case-studies/mbjoseph-CARStan.html)
11 [stan.org/users/documentation/case-studies/mbjoseph-CARStan.html](https://mc-stan.org/users/documentation/case-studies/mbjoseph-CARStan.html). [1 May 2019]

12

13