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]];
```

Supplementary Appendices S1 and S2 to the article "A model for longitudinal data sets relating wind-damage probability to biotic and abiotic factors: a Bayesian approach", *by* Kiyoshi Umeki, Marc D. Abrams, Keisuke Toyama and Eri Nabeshima. *Forest Systems Vol. 28 No. 3, December 2019 (https://doi.org/10.5424/fs/2019283-15200)*

```
    }
    return 0.5 * ((n-1) * log(tau) - tau * (phit_D * phi - (phit_W * phi)));
    }
}

data
{
  int < lower = 1 > nData;
  int < lower = 0 > damage[ nData ];
  vector[ nData ] windSpeed;
  int < lower = 1 > nForWindDir;
  int < lower = 1, upper = nForWindDir > windDirIndex[ nData ];
  matrix< lower = 0, upper = 1 >[ nForWindDir, nForWindDir ] WForWindDir; // adjacency
matrix
  int < lower = 1 > W_nForWindDir;              // number of adjacent region pairs
  int < lower = 1 > nForSeason;
  int < lower = 1, upper = nForSeason > seasonIndex[ nData ];
  matrix< lower = 0, upper = 1 >[ nForSeason, nForSeason ] WForSeason; // adjacency matrix
  int < lower = 1 > W_nForSeason;              // number of adjacent region pairs
  int < lower = 1 > nForDecade;
  int < lower = 1, upper = nForDecade > decadeIndex[ nData ];
  // real < lower = 0, upper = 100 > sigmaFirstRDecade;
  int nObsMeanVol;
  vector[ nObsMeanVol ] obsMeanVol;
  int decadeWithMeanVol[ nObsMeanVol ];
  int < lower = 0, upper = 100 > nWindSpeedForPred;
  vector[ nWindSpeedForPred ] windSpeedForPred;
  int < lower = 0, upper = 100 > nMeanVolForPred;
  vector[ nMeanVolForPred ] meanVolForPred;
}
```

Supplementary Appendices S1 and S2 to the article "A model for longitudinal data sets relating wind-damage probability to biotic and abiotic factors: a Bayesian approach", *by* Kiyoshi Umeki, Marc D. Abrams, Keisuke Toyama and Eri Nabeshima. *Forest Systems Vol. 28 No. 3, December 2019 (https://doi.org/10.5424/fs/2019283-15200)*

```
transformed data
{
  int W_sparseForWindDir[ W_nForWindDir, 2 ];   // adjacency pairs
  vector[ nForWindDir ] D_sparseForWindDir;     // diagonal of D (number of neigbors for each site)
  int W_sparseForSeason[ W_nForSeason, 2 ];   // adjacency pairs
  vector[ nForSeason ] D_sparseForSeason;     // diagonal of D (number of neigbors for each site)
  real mWindSpeedForPred;
  real mMeanVolForPred;

  mWindSpeedForPred = mean( windSpeedForPred );
  mMeanVolForPred = mean( meanVolForPred );

  { // generate sparse representation for W
    int counter;
    counter = 1;
    // loop over upper triangular part of W to identify neighbor pairs
    for ( i in 1:( nForWindDir - 1 ) )
    {
      for ( j in ( i + 1 ):nForWindDir )
      {
        if ( WForWindDir[i, j] == 1 )
        {
          W_sparseForWindDir[ counter, 1 ] = i;
          W_sparseForWindDir[ counter, 2 ] = j;
          counter = counter + 1;
        }
      }
    }
```

```
}
for (i in 1:nForWindDir)
  D_sparseForWindDir[ i ] = sum( WForWindDir[ i ] );
{
  vector[ nForWindDir ] invsqrtDForWindDir;
  for ( i in 1:nForWindDir )
  {
    invsqrtDForWindDir[ i ] = 1 / sqrt( D_sparseForWindDir[ i ] );
  }
}

{ // generate sparse representation for W
  int counter;
  counter = 1;
  // loop over upper triangular part of W to identify neighbor pairs
  for ( i in 1:( nForSeason - 1 ) )
  {
    for ( j in ( i + 1 ):nForSeason )
    {
      if ( WForSeason[i, j] == 1 )
      {
        W_sparseForSeason[ counter, 1 ] = i;
        W_sparseForSeason[ counter, 2 ] = j;
        counter = counter + 1;
      }
    }
  }
}
for (i in 1:nForSeason)
  D_sparseForSeason[ i ] = sum( WForSeason[ i ] );
```

```
  {
   vector[ nForSeason ] invsqrtDForSeason;
   for ( i in 1:nForSeason )
   {
     invsqrtDForSeason[ i ] = 1 / sqrt( D_sparseForSeason[ i ] );
   }
  }
}

parameters
{
  real < lower = -100, upper = 100 > b0;
  real < lower = -100, upper = 100 > bForWindDir;
  vector[ nForWindDir ] phi_unscaledForWindDir;
  real < lower = 0, upper = 100 > sigmaForWindDir;
  vector[ nForSeason ] phi_unscaledForSeason;
  real < lower = 0, upper = 100 > sigmaForSeason;
  real bForMeanVol;
  vector < lower = 0, upper = 500 >  [ nForDecade ] estMeanVol;
  real < lower = 0, upper = 100 > sigmaForDecade;
  real < lower = 0, upper = 100 > sigmaForObsMeanVol;
}

transformed parameters
{
  vector[ nForWindDir ] phiForWindDir; // brute force centering
  vector[ nForSeason ] phiForSeason; // brute force centering
  // vector[ nData ] linearPredictor;
  vector[ nData ] p;
  real tauForWindDir;
```

```
  real tauForSeason;

  phiForWindDir = phi_unscaledForWindDir - mean( phi_unscaledForWindDir );
  phiForSeason = phi_unscaledForSeason - mean( phi_unscaledForSeason );
  // linearPredictor = b1 * windSpeed + phiForWindDir[ windDirIndex ] +
phiForSeason[ seasonIndex ] + rDecade[ decadeIndex ];
  p = inv_logit( b0 + bForWindDir * windSpeed + bForMeanVol * estMeanVol[ decadeIndex ] +
phiForWindDir[ windDirIndex ] + phiForSeason[ seasonIndex ] );
  tauForWindDir = 1 / ( sigmaForWindDir * sigmaForWindDir );
  tauForSeason = 1 / ( sigmaForSeason * sigmaForSeason );
}

model
{
  phi_unscaledForWindDir ~
    sparse_iar( tauForWindDir, W_sparseForWindDir, D_sparseForWindDir, nForWindDir,
W_nForWindDir );
  phi_unscaledForSeason ~
    sparse_iar( tauForSeason, W_sparseForSeason, D_sparseForSeason, nForSeason,
W_nForSeason );

  estMeanVol[ 1 ] ~ normal( 0, 100 );
  estMeanVol[ 2:nForDecade ] ~ normal( estMeanVol[ 1:( nForDecade - 1 ) ], sigmaForDecade );
  obsMeanVol ~ normal( estMeanVol[ decadeWithMeanVol ], sigmaForObsMeanVol );
  damage ~ bernoulli( p );
}

generated quantities
{
  vector [ nWindSpeedForPred ] predPForWindSpeed;
```

```
  vector [ nForWindDir ] predPForWindDir;
  vector [ nForSeason ] predPForSeason;
  vector [ nMeanVolForPred ] predPForMeanVol;
  vector [ nData ] log_lik;

  for ( i in 1:nData )
  {
    log_lik[ i ] = bernoulli_lpmf( damage[ i ] | p[ i ] );
  }

  predPForWindSpeed = inv_logit( b0 + bForWindDir * windSpeedForPred + bForMeanVol *
mMeanVolForPred );
  predPForWindDir = inv_logit( b0 + bForWindDir * mWindSpeedForPred + bForMeanVol *
mMeanVolForPred + phiForWindDir );
  predPForSeason = inv_logit( b0 + bForWindDir * mWindSpeedForPred + bForMeanVol *
mMeanVolForPred + phiForSeason );
  predPForMeanVol = inv_logit( b0 + bForWindDir * mWindSpeedForPred + bForMeanVol *
meanVolForPred );
}
```

References

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

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]];
```

```
    }
    return 0.5 * ((n-1) * log(tau) - tau * (phit_D * phi - (phit_W * phi)));
    }
}

data
{
  int < lower = 1 > nData;
  int < lower = 0 > damage[ nData ];
  vector[ nData ] windSpeed;
  int < lower = 1 > nForWindDir;
  int < lower = 1, upper = nForWindDir > windDirIndex[ nData ];
  matrix< lower = 0, upper = 1 >[ nForWindDir, nForWindDir ] WForWindDir; // adjacency matrix
  int < lower = 1 > W_nForWindDir;               // number of adjacent region pairs
  int < lower = 1 > nForSeason;
  int < lower = 1, upper = nForSeason > seasonIndex[ nData ];
  matrix< lower = 0, upper = 1 >[ nForSeason, nForSeason ] WForSeason; // adjacency matrix
  int < lower = 1 > W_nForSeason;                // number of adjacent region pairs
  int < lower = 1 > nForDecade;
  int < lower = 1, upper = nForDecade > decadeIndex[ nData ];
  // real < lower = 0, upper = 100 > sigmaFirstRDecade;
  int < lower = 0, upper = 100 > nWindSpeedForPred;
  vector[ nWindSpeedForPred ] windSpeedForPred;
}

transformed data
{
  int W_sparseForWindDir[ W_nForWindDir, 2 ];   // adjacency pairs
```

Supplementary Appendices S1 and S2 to the article "A model for longitudinal data sets relating wind-damage probability to biotic and abiotic factors: a Bayesian approach", *by* Kiyoshi Umeki, Marc D. Abrams, Keisuke Toyama and Eri Nabeshima. *Forest Systems Vol. 28 No. 3, December 2019 (https://doi.org/10.5424/fs/2019283-15200)*

```
  vector[ nForWindDir ] D_sparseForWindDir;     // diagonal of D (number of neigbors for each site)
  int W_sparseForSeason[ W_nForSeason, 2 ];   // adjacency pairs
  vector[ nForSeason ] D_sparseForSeason;     // diagonal of D (number of neigbors for each site)
  real mWindSpeedForPred;

  mWindSpeedForPred = mean( windSpeedForPred );

  { // generate sparse representation for W
    int counter;
    counter = 1;
    // loop over upper triangular part of W to identify neighbor pairs
    for ( i in 1:( nForWindDir - 1 ) )
    {
      for ( j in ( i + 1 ):nForWindDir )
      {
        if ( WForWindDir[i, j] == 1 )
        {
          W_sparseForWindDir[ counter, 1 ] = i;
          W_sparseForWindDir[ counter, 2 ] = j;
          counter = counter + 1;
        }
      }
    }
  }
  for (i in 1:nForWindDir)
    D_sparseForWindDir[ i ] = sum( WForWindDir[ i ] );
  {
    vector[ nForWindDir ] invsqrtDForWindDir;
    for ( i in 1:nForWindDir )
```

Supplementary Appendices S1 and S2 to the article "A model for longitudinal data sets relating wind-damage probability to biotic and abiotic factors: a Bayesian approach", *by* Kiyoshi Umeki, Marc D. Abrams, Keisuke Toyama and Eri Nabeshima. *Forest Systems Vol. 28 No. 3, December 2019 (https://doi.org/10.5424/fs/2019283-15200)*

```
    {
      invsqrtDForWindDir[ i ] = 1 / sqrt( D_sparseForWindDir[ i ] );
    }
  }

  { // generate sparse representation for W
    int counter;
    counter = 1;
    // loop over upper triangular part of W to identify neighbor pairs
    for ( i in 1:( nForSeason - 1 ) )
    {
      for ( j in ( i + 1 ):nForSeason )
      {
        if ( WForSeason[i, j] == 1 )
        {
          W_sparseForSeason[ counter, 1 ] = i;
          W_sparseForSeason[ counter, 2 ] = j;
          counter = counter + 1;
        }
      }
    }
  }
  for (i in 1:nForSeason)
    D_sparseForSeason[ i ] = sum( WForSeason[ i ] );
  {
    vector[ nForSeason ] invsqrtDForSeason;
    for ( i in 1:nForSeason )
    {
      invsqrtDForSeason[ i ] = 1 / sqrt( D_sparseForSeason[ i ] );
    }
```

```
  }
}

parameters
{
  real < lower = -100, upper = 100 > b1;
  vector[ nForWindDir ] phi_unscaledForWindDir;
  real < lower = 0, upper = 100 > sigmaForWindDir;
  vector[ nForSeason ] phi_unscaledForSeason;
  real < lower = 0, upper = 100 > sigmaForSeason;
  vector [ nForDecade ] rDecade;
  real < lower = 0, upper = 100 > sigmaForDecade;
}

transformed parameters
{
  vector[ nForWindDir ] phiForWindDir; // brute force centering
  vector[ nForSeason ] phiForSeason; // brute force centering
  // vector[ nData ] linearPredictor;
  vector[ nData ] p;
  real tauForWindDir;
  real tauForSeason;

  phiForWindDir = phi_unscaledForWindDir - mean( phi_unscaledForWindDir );
  phiForSeason = phi_unscaledForSeason - mean( phi_unscaledForSeason );
  // linearPredictor = b1 * windSpeed + phiForWindDir[ windDirIndex ] +
phiForSeason[ seasonIndex ] + rDecade[ decadeIndex ];
  p = inv_logit( b1 * windSpeed + phiForWindDir[ windDirIndex ] +
phiForSeason[ seasonIndex ] + rDecade[ decadeIndex ] );
  tauForWindDir = 1 / ( sigmaForWindDir * sigmaForWindDir );
```

```
  tauForSeason = 1 / ( sigmaForSeason * sigmaForSeason );
}

model
{
  phi_unscaledForWindDir ~
    sparse_iar( tauForWindDir, W_sparseForWindDir, D_sparseForWindDir, nForWindDir,
W_nForWindDir );
  phi_unscaledForSeason ~
    sparse_iar( tauForSeason, W_sparseForSeason, D_sparseForSeason, nForSeason,
W_nForSeason );

  rDecade[ 1 ] ~ normal( 0, 100 );
  rDecade[ 2:nForDecade ] ~ normal( rDecade[ 1:( nForDecade - 1 ) ], sigmaForDecade );
  damage ~ bernoulli( p );
  // target += bernoulli_logit_lpmf( damage | linearPredictor );
}

generated quantities
{
  vector [ nWindSpeedForPred ] predPForWindSpeed;
  vector [ nForWindDir ] predPForWindDir;
  vector [ nForSeason ] predPForSeason;
  vector [ nForDecade ] predPForDecade;
  vector [ nData ] log_lik;

  for ( i in 1:nData )
  {
    log_lik[ i ] = bernoulli_lpmf( damage[ i ] | p[ i ] );
  }
```

```
 predPForWindSpeed = inv_logit( b1 * windSpeedForPred + mean( rDecade ) );
 predPForWindDir = inv_logit( b1 * mWindSpeedForPred + mean( rDecade ) +
phiForWindDir );
 predPForSeason = inv_logit( b1 * mWindSpeedForPred + mean( rDecade ) + phiForSeason );
 predPForDecade = inv_logit( b1 * mWindSpeedForPred + rDecade );
}
```

References

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