# Appendix 5: Robustness Assessments

## Appendix 5

To assess model performance and robustness, we conducted both a model selection and model checking procedure. For model selection, we evaluated multiple functional forms for describing the overwinter mortality using Watanabe–Akaike information criterion (WAIC). To check the model, we calculated posterior predictive p-values.

### Appendix 5.1

The inter-annual population transitions (i.e., transition from year $i$ to year $i+1$) are largely described by density-dependent overwinter mortality. Since density dependence only enters the model during this process and is therefore likely influential for forecasting the stable size distribution, we compared multiple functional forms for size- and density-dependent overwinter mortality (Equations 15-16 in main text).

We fit four separate models that are identical, apart from the functional form of overwinter survival (Equations 15-16). These models varied by whether the relationship between size- and density-dependence is additive or interactive, as well as how rapidly mortality decreases with size.

**Model 1:** interactive density- and size-dependence, steeper mortality decrease with size

$$S_o(y, N^T_{t_{max},i}) = exp(-\frac{\alpha^o_i \times N^T_{t_{max},i}}{y^2})$$

$$\alpha^o_i \sim Gamma(\alpha^o_\alpha, \alpha^o_\theta)$$

**Model 2:** interactive density- and size-dependence, less steep mortality decrease with size

$$S_o(y, N^T_{t_{max},i}) = exp(-\frac{\alpha^o_i \times N^T_{t_{max},i}}{y})$$

$$\alpha^o_i \sim Gamma(\alpha^o_\alpha, \alpha^o_\theta)$$

**Model 3:** additive density- and size-dependence, steeper mortality decrease with size

$$S_o(y, N^T_{t_{max},i}) = exp(-(\frac{\alpha^o}{N^T_{t_{max},i}} + \frac{\psi}{y^2}))$$

**Model 4:** additive density- and size-dependence, less steep mortality decrease with size

$$S_o(y, N^T_{t_{max},i}) = exp(-(\frac{\alpha^o}{N^T_{t_{max},i}} + \frac{\psi}{y}))$$

WAIC was calculated using the combined posterior samples from four chains, after discarding 2000 burn-in samples (32000 total samples). Model 1, with a size- and density-dependent overwinter mortality interaction and a steeper mortality decrease with size had the lowest WAIC and was used in subsequent analyses.

| Model | WAIC | lppd | pWAIC | ΔWAIC |
|---|---|---|---|---|
| Model 1 | 6441.55 | -3169.32 | 51.46 | 0.00 |
| Model 2 | 6448.93 | -3171.91 | 52.56 | 7.38 |
| Model 3 | 6481.68 | -3187.81 | 53.03 | 40.13 |
| Model 4 | 6506.52 | -3198.85 | 54.41 | 64.97 |

## Appendix 5.2

Bayesian model checking was used to determine if the model is an adequate representation of the observed data. Fit was checked with an omnibus discrepancy function, deviance. Here, $y$ refers to the the count of removed crabs, $n^{C}_{t,j,i,y}$, the total number of removed crabs, $n^{R}_{t,i,y}$, and the number of recaptured marked crabs, $mc^{mc}_{y}$:

$$T(y, \theta) = -2log(y|\theta)$$

as well as a targeted discrepancy function to check zero inflation in the count data. Here, $y$ refers to the the count of removed crabs, $n^{C}_{t,j,i,y}$:

$$T(y) = \sum_{i} I(y_i = 0)$$

### Generating posterior predictive samples and calculating deviance

The posterior predictive samples (PPS) and deviance of PPS was calculated using the following nimble function:

```
ppSamplerNF <- nimbleFunction(
  setup = function(model, samples) {

    # theta
    topNodes <- model$getNodeNames(stochOnly = TRUE, topOnly = TRUE)

    # nodes to simulate
    simNodes <- model$getNodeNames()[!(model$getNodeNames() %in% topNodes)]

    # data nodes
    dataNodes <- model$getNodeNames(dataOnly = TRUE)

    n <- length(model$expandNodeNames(dataNodes, returnScalarComponents = TRUE))
    vars <- colnames(samples[, topNodes])

    # subset posterior samples to just theta (top nodes in graph)
    samples_sub <- samples[, topNodes]

  },
  run = function(samples = double(2)) {

    nSamp <- dim(samples)[1]
```

```
    ppSamples <- matrix(nrow = nSamp, ncol = n + 1)

    for(i in 1:nSamp) {

      # add theta to model
      values(model, vars) <<- samples_sub[i, ]

      # update nodes based on theta
      model$simulate(simNodes, includeData = TRUE)

      # save PPS
      ppSamples[i, 1:n] <- values(model, dataNodes)

      # store deviance of each sample
      ppSamples[i, n + 1] <- 2 * model$calculate(dataNodes)
    }
    returnType(double(2))
    return(ppSamples)
  })
```

The posterior predictive samples (PPS) and deviance were calculated and saved using the nimble function:

```
# generate posterior predictive sampler
ppSampler <- ppSamplerNF(
  myModel, # uncompiled model (see Appendix 3)
  samples # posterior samples
)
# compile posterior predictive samples
cppSampler <- compileNimble(
  ppSampler,
  project = CmyModel # compiled model (see Appendix 3)
)
# run compiled function
ppSamples_via_nf <- cppSampler$run(samples)

# get PPS and samples
n <- length(CmyModel$expandNodeNames(dataNodes, returnScalarComponents = TRUE))
ppSamples <- ppSamples_via_nf[, 1:n]
deviance <- ppSamples_via_nf[, n + 1]

# save
saveRDS(ppSamples, paste0("data/posterior_predictive_check/PPS.rds"))
saveRDS(deviance, "data/posterior_predictive_check/deviance_yrep.rds")
```

### Calculating deviance of data

Bayesian p-values are calculated by comparing the deviance of the posterior predictive samples and the deviance of the observed data. Deviance of the data was calculated using the following nimble function:

```
calc_deviance_y <- nimbleFunction(
  setup = function(model, samples) {

    # theta = top nodes and latent states
```

```
    theta <- colnames(samples)

    # calculate model graph dependencies of theta to update
    deps <- CmyModel$getDependencies(theta, self = TRUE)

    # data nodes
    dataNodes <- model$getNodeNames(dataOnly = TRUE)

  },
  run = function(samples = double(2)) {

    nSamp <- dim(samples)[1]
    deviance <- rep(NA, nSamp)

    for(i in 1:nSamp) {

      # add theta
      values(model, theta) <<- samples[i, ]

      # update dependencies
      CmyModel$calculate(deps)

      # calculate deviance
      deviance[i] <- 2 * model$calculate(dataNodes)
    }
    returnType(double(1))
    return(deviance)
  })
```

The deviance of the data was calculated and saved using the nimble function:

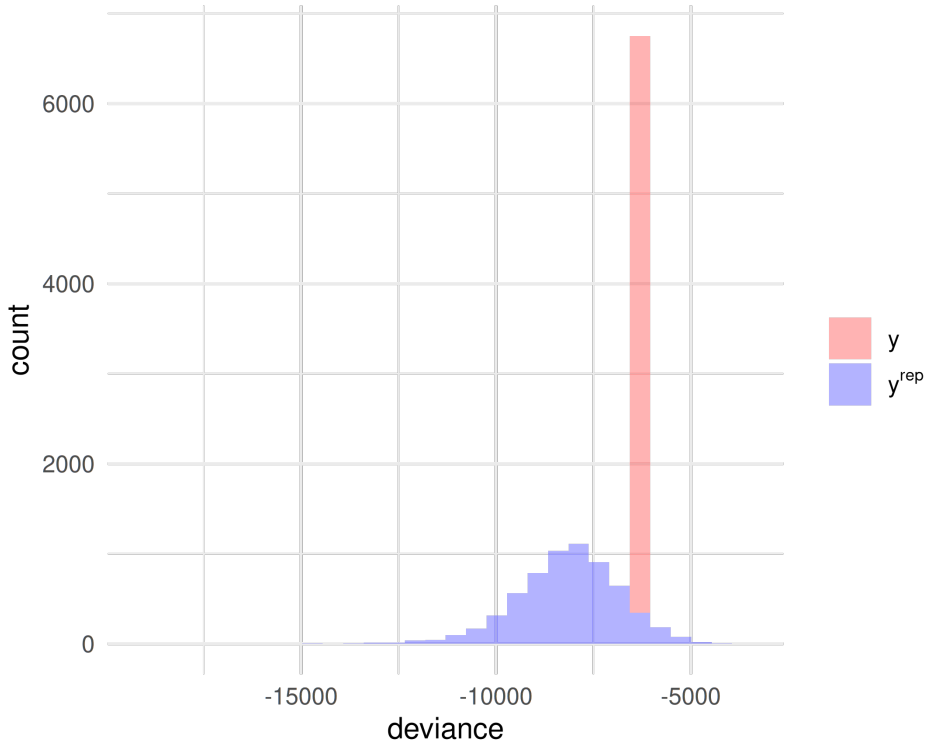
```
# generate deviance calculator
devianceCalculator <- calc_deviance_y(
  myModel, # uncompiled model, contains data (see Appendix 3)
  samples # posterior samples
)
# compile deviance calculator
CdevianceCalculator <- compileNimble(
  devianceCalculator,
  project = CmyModel # compiled model, contains data (see Appendix 3)
)
# run compiled function
data_deviance_via_nf <- CdevianceCalculator$run(samples)

# save
saveRDS(data_deviance_via_nf, "data/posterior_predictive_check/deviance_y.rds")
```

**Model checking with deviance**

The Bayesian p-value calculated using deviance as the proportion of $T(y_i) > T(y_i^{rep})$. The deviance p-value is 0.92.

**Figure A5.1:** Histogram of deviance generated for each posterior sample for $y$ (red) and $y^{rep}$ (blue).



**Model checked with zero inflation**

The Bayesian p-value using the targeted zero-inflation check was calculated as the proportion of $T(y_i) > T(y_i^{rep})$, where $y$ refers to the count data. The zero-inflation check p-value is 0.85.

**Figure A5.2:** Histogram of proportion of zeros in each set, $i$, of posterior predictive samples, $y^{rep}$. Red line indicates the proportion of zeros in the count data, $y$.