Supplementary material to *Hierarchical diffusion models for two-choice response times*: II. WinBUGS code for the example applications

Joachim Vandekerckhove, Francis Tuerlinckx
Department of Psychology, University of Leuven

Michael Lee
Department of Cognitive Sciences, University of California, Irvine, CA

Abstract

In this document, we provide the exact WinBUGS code we used for the two example applications. There are four models in total: BM1, BM2, and BM3 for the first application, and PHM for the second application.

```
Benchmark data, Model BM1 {
   # Prior for beta
   beta ~ dunif(0.1,0.9)

   # Insert a plate over instruction conditions
   for (i in 1:2)
   {
      # Prior for alphas
      alpha[i] ~ dunif(0.03,0.25)

      # The Wiener distribution code works with
      # the zeta_init parameter and not with
      # beta. So we have to rescale the entire
      # mixing distribution in a somewhat
```

This research was funded by grants GOA/00/02–ZKA4511, GOA/2005/04–ZKB3312, IUAP P5/24, and K.2.215.07.N.01. This paper is part of the doctoral project of JV. This research was conducted utilizing high performance computational resources provided by the University of Leuven, `http://ludit.kuleuven.be/hpc`. We are further indebted to Microsoft Corporation and Dell Inc. for generously providing us with additional computing resources. Correspondence concerning this article may be addressed to: Joachim Vandekerckhove, University of Leuven, Department of Psychology, Tiensestraat 102 B3713, B–3000 Leuven, Belgium; ph: +3216326118; fax: +3216325993; e: joachim.vandekerckhove@psy.kuleuven.be

```
    # complicated way.

    # The mean of the uniform mixing
    # distribution is alpha*beta
    z[i] <- alpha[i]*beta

    # Now we work on the range of the
    # uniform. The range is constrained
    # by twice the distance to the nearest
    # boundary. So we create a logical node
    # to store this distance (either
    # alpha-zeta_init or zeta_init)
    edges[i,1] <- z[i]
    edges[i,2] <- alpha[i]-z[i]
    szmax[i] <- 2*min(edges[i,1],edges[i,2])

    # Having the maximum of the uniform's
    # range stored, we now create a node
    # with prior U(0,szmax) by rescaling a
    # U(0,1) prior. Note that the Beta(1,1)
    # prior is the same as a U(0,1) prior.
    sztmp[i] ~ dbeta(1,1)
    sz[i] <- sztmp[i]*szmax[i]
    # sz now has the correct prior

    # To apply the mixing distribution later,
    # we store the lower bound of the
    # trial-to-trial mixing distribution.
    zlo[i] <- z[i]-sz[i]/2

    # Insert a plate over brightness
    # conditions and set a population
    # distribution for the condition-specific
    # drift rates.
    for (s in 1:nc)
    {
       # Mean of the population distribution
       # is determined by a single parameter
       # in this particular model.
       nu.hat[s,i] <- mu
       nu[s,i] ~ dnorm(nu.hat[s,i],prec)I(-.95,.95)
    }
}
```

```
# The mean has a prior
mu ~ dunif(-.7,.7)

# The standard deviation of the population
# distribution is sigma_epsilon but needs to
# be transformed into a precision (1/var).
sigma.epsilon ~ dunif(0.0001,0.6)
prec <- pow(sigma.epsilon,-2)

# The trial-to-trial mixing of nondecision
# time also needs a mean and a precision.
theta ~ dbeta(1,1)
chi ~ dbeta(1,1)
pt <- pow(chi,-2)

# The trial-to-trial mixing of drift rate is
# coded directly into the distribution file,
# so we don't need to transform the standard
# deviation eta. The mean of the mixing
# distribution is the nu parameter that was
# defined above.
eta ~ dunif(0.0001,.4)

# Now we 'loop' over data points. We have
# covariate vectors 'stim' and 'ins' that
# tell us in which stimulus/instruction
# condition trial j was (ins: 1 for speed,
# 2 for accuracy).
for (j in 1:N)
{
    # Trial-specific nondecision time tau.
    tau[j] ~ dnorm(theta,pt)I(0,2)

    # Trial-specific zeta_init is again a
    # rescaled U(0,1). The dependence on the
    # instruction condition is expressed by
    # using the 'ins' covariate as an index.
    ztemp[j] ~ dbeta(1,1)
    zinit[j] <- zlo[ins[j]]+sz[ins[j]]*ztemp[j]

    # Finally, the data are connected to the
    # Wiener distribution with the correct
    # parameters.
```

```
      t[j] ~ dwiener.eta(alpha[ins[j]],tau[j],
                         zinit[j],nu[stim[j],ins[j]],eta)
   }
}
```

```
Benchmark data, Model BM2 {
   beta ~ dunif(0.1,0.9)

   for (i in 1:2)
   {
      alpha[i] ~ dunif(0.03,0.25)
      z[i] <- alpha[i]*beta
      edges[i,1] <- z[i]
      edges[i,2] <- alpha[i]-z[i]
      szmax[i] <- 2*min(edges[i,1],edges[i,2])
      sztmp[i] ~ dbeta(1,1)
      sz[i] <- sztmp[i]*szmax[i]
      zlo[i] <- z[i]-sz[i]/2

      for (s in 1:nc)
      {
         # Mean of the population distribution
         # is now a nonlinear function of s,
         # the stimulus intensity condition.
         pmf[s,i] <- 1-exp(-pow(s/(33*nu.sc),nu.sh))
         nu.hat[s,i] <- nu.lo + (nu.hi-nu.lo) * pmf[s,i]

         nu[s,i] ~ dnorm(nu.hat[s,i],prec)I(-.95,.95)
      }
   }

   # The Weibull link parameters get priors.
   nu.lo ~ dunif(-0.95,-0.2)
   nu.hi ~ dunif(0.2,0.95)
   nu.sc ~ dunif(0.25,0.75)
   nu.sh ~ dunif(1,20)

   sigma.epsilon ~ dunif(0.0001,0.6)
   prec <- pow(sigma.epsilon,-2)
```

```
   theta ~ dbeta(1,1)
   chi ~ dbeta(1,1)
   pt <- pow(chi,-2)

   eta ~ dunif(0.0001,.4)

   for (j in 1:N)
   {
       tau[j] ~ dnorm(theta,pt)I(0,2)
       ztemp[j] ~ dbeta(1,1)
       zinit[j] <- zlo[ins[j]]+sz[ins[j]]*ztemp[j]
       t[j] ~ dwiener.eta(alpha[ins[j]],tau[j],
                          zinit[j], nu[stim[j],ins[j]],eta)
   }
}
```

```
Benchmark data, Model BM3 {
   beta ~ dunif(0.1,0.9)

   for (i in 1:2)
   {
      alpha[i] ~ dunif(0.03,0.25)
      z[i] <- alpha[i]*beta
      edges[i,1] <- z[i]
      edges[i,2] <- alpha[i]-z[i]
      szmax[i] <- 2*min(edges[i,1],edges[i,2])
      sztmp[i] ~ dbeta(1,1)
      sz[i] <- sztmp[i]*szmax[i]
      zlo[i] <- z[i]-sz[i]/2

      # In this model, the Weibull link
      # parameters are allowed to differ as a
      # function of the instruction condition.
      nu.lo[i] ~ dunif(-0.95,-0.2)
      nu.hi[i] ~ dunif(0.2,0.95)
      nu.sc[i] ~ dunif(0.25,0.75)
      nu.sh[i] ~ dunif(1,20)

      for (s in 1:nc)
      {
```

```
        pmf[s,i] <- 1-exp(-pow(s/(33*nu.sc[i]),nu.sh[i]))
        nu.hat[s,i] <- nu.lo[i] + (nu.hi[i]-nu.lo[i]) * pmf[s,i]

        nu[s,i] ~ dnorm(nu.hat[s,i],prec)I(-.95,.95)
      }
   }

   sigma.epsilon ~ dunif(0.0001,0.6)
   prec <- pow(sigma.epsilon,-2)

   theta ~ dbeta(1,1)
   chi ~ dbeta(1,1)
   pt <- pow(chi,-2)

   eta ~ dunif(0.0001,.4)

   for (j in 1:N)
   {
       tau[j] ~ dnorm(theta,pt)I(0,2)
       ztemp[j] ~ dbeta(1,1)
       zinit[j] <- zlo[ins[j]]+sz[ins[j]]*ztemp[j]
       t[j] ~ dwiener.eta(alpha[ins[j]],tau[j],
                          zinit[j],nu[stim[j],ins[j]],eta)
   }
}
```

```
This is the Population Hierarchical Model (PHM) {
    mu.alpha ~ dunif(0.02,0.30)
    sigma.alpha ~ dunif(0.0001,0.15)
    prec.alpha <- pow(sigma.alpha,-2)

    beta <- 0.5

    mu.theta ~ dunif(0.02,0.70)
    sigma.theta ~ dunif(0.0001,0.15)
    prec.theta <- pow(sigma.theta,-2)

    mu.chi ~ dunif(0.0001,0.15)
    sigma.chi ~ dunif(0.0001,0.10)
    prec.chi <- pow(sigma.chi,-2)
```

```
    mu.eta ~ dunif(0.001,0.45)
    sigma.eta ~ dunif(0.0001,0.15)
    prec.eta <- pow(sigma.eta,-2)

    for(i in 1:nc)
    {
        mu.nu[i] ~ dunif(-.5,.6)
        sigma.nu[i] ~ dunif(0,0.6)
        prec.nu[i] <- pow(sigma.nu[i],-2)
    }

    for(p in 1:np)
    {
        alpha[p] ~ dnorm(mu.alpha,prec.alpha)I(0.01,0.25)
        zeta_init[p] <- alpha[p]*beta
        theta[p] ~ dnorm(mu.theta,prec.theta)I(0.01,0.80)
        chi[p] ~ dnorm(mu.chi,prec.chi)I(0.00001,0.49)
        prec.tau[p] <- pow(chi[p],-2)
        eta[p] ~ dnorm(mu.eta,prec.eta)I(0.00001,0.49)
        for(i in 1:nc)
        {
            nu[p,i] ~ dnorm(mu.nu[i],prec.nu[i])I(-.7,.7)
        }
    }

    for(j in 1:N)
    {
        tau[j] ~ dnorm(theta[pnum[j]],prec.tau[pnum[j]])I(0,)
        t[j] ~ dwiener.eta(alpha[pnum[j]],tau[j],zeta_init[pnum[j]],
                    nu[pnum[j],cond[j]],eta[pnum[j]])
    }
}
```