```
## Warning:  package 'pracma' was built under R version 3.1.1
## Loading required package:  wavethresh
## Loading required package:  MASS
## WaveThresh:  R wavelet software, release 4.6.6, installed
##
## Copyright Guy Nason and others 1993-2013
##
## Note:  nlevels has been renamed to nlevelsWT
##
## Loading required package:  adlift
## Loading required package:  EbayesThresh
##
##  ********************************************
##  adlift:  a package to perform wavelet lifting schemes
##
##  --- Written by Matt Nunes and Marina Knight ---
##    Current package version:  1.3-2  ( 01/11/2012 )
##
##              -+ packaged by MAN +-
##  ********************************************
##
##  adlift 1.3-2 loaded
##
##
## Attaching package:  'adlift'
##
## The following object is masked from 'package:EbayesThresh':
##
##    postmean.cauchy
##
##
##  ********************************************
##  binhf:  Haar-Fisz functions for binomial data
##
##  --- Written by Matt Nunes ---
##    Current package version:  1.0-1  ( 24/04/2014 )
##
##
##  ********************************************
##
##  binhf 1.0-1 loaded
##
##
## Attaching package:  'binhf'
##
## The following objects are masked from 'package:EbayesThresh':
##
##    ebayesthresh.wavelet.wd, negloglik.laplace, wandafromx
##
## The following object is masked from 'package:wavethresh':
##
##    madmad
##
## The following object is masked from 'package:base':
##
##    norm
##
##
## Attaching package:  'deSolve'
```

```
##
## The following object is masked from 'package:pracma':
##
##     rk4
##
## Loading required package:  rootSolve
##
## Attaching package:  'rootSolve'
##
## The following objects are masked from 'package:pracma':
##
##     gradient, hessian
##
## Loading required package:  coda
## Loading required package:  lattice
##
## Attaching package:  'FME'
##
## The following object is masked from 'package:pracma':
##
##     Norm
##
## Loading required package:  timeDate
## Loading required package:  timeSeries
##
## Attaching package:  'fBasics'
##
## The following object is masked from 'package:deSolve':
##
##     rk
##
## The following object is masked from 'package:binhf':
##
##     norm
##
## The following objects are masked from 'package:pracma':
##
##     akimaInterp, inv, kron, pascal
##
## The following object is masked from 'package:base':
##
##     norm
## 14    SC5A     2     61.339  -9.078  -8.651  -8.970  -6.938 -6.8880
## 15    SC6A     2     67.914 -10.397  -9.534 -10.329  -6.971 -6.8860
## 16    SC7A     2     65.968  -6.645  -6.447  -6.536  -5.203 -5.1440
## 17    SC8A     2     60.767  -9.449  -7.590  -7.952  -5.188 -5.1480
## 18    SC9B     2     60.937  -9.384  -8.139  -9.311  -5.203 -5.1370
##    S.L.AVG S.D.AVG   E.V.MIN E.V.MAX  E.A.MIN E.A.MAX T.AVG FD.MIN FD.MAX
## 1   -3.256  0.4457 -0.174330       0 -0.08441       0 293.0 0.7018 0.8354
## 2   -1.014  3.9473 -0.017568       0 -0.15550       0 297.5 0.9258 0.9422
## 3   -4.999  4.0097 -0.068824       0 -0.09518       0 294.7 0.9170 0.9823
## 4   -1.008  4.0019 -0.033969       0 -0.15064       0 297.9 0.9073 0.9387
## 5   -2.000  3.9803 -0.060718       0 -0.14177       0 297.1 0.9092 0.9661
## 6   -3.003  3.9901 -0.061275       0 -0.13528       0 296.7 0.9312 0.9900
## 7   -4.000  3.9895 -0.069213       0 -0.14917       0 294.8 0.9043 0.9691
## 8   -1.003  4.9916 -0.006195       0 -0.01482       0 298.0 0.8373 0.8425
## 9   -1.025  4.9693 -0.021617       0 -0.04677       0 298.0 0.7813 0.7983
## 10  -3.439  0.6925 -0.243860       0 -0.09874       0 298.0 0.7286 0.9298
## 11  -3.459  1.3683 -0.225030       0 -0.11139       0 298.0 0.7445 0.9324
## 12  -3.454  2.0623 -0.241510       0 -0.13777       0 298.0 0.7291 0.9283
## 13  -6.894  0.6944 -0.222610       0 -0.09474       0 298.0 0.7439 0.9294
## 14  -6.916  2.0549 -0.190630       0 -0.09817       0 298.0 0.8167 0.9883
## 15  -6.912  3.4164 -0.188180       0 -0.12228       0 298.0 0.7874 0.9504
## 16  -5.173  1.3636 -0.227880       0 -0.10278       0 298.0 0.7706 0.9678
## 17  -5.167  2.7848 -0.216170       0 -0.13914       0 298.0 0.7461 0.9261
## 18  -5.165  4.1463 -0.212080       0 -0.16152       0 298.0 0.7791 0.9631
##    W.AVG   ER.V.MIN  ER.V.MAX   ER.A.MIN  ER.A.MAX
## 1   2.40 -1.064e-06 0.000e+00 -5.177e-07 0.000e+00
## 2   1.66 -1.208e-06 4.225e-09 -1.131e-05 0.000e+00
## 3   1.49 -4.494e-06 7.194e-09 -4.622e-06 7.000e-06
## 4   1.77 -7.282e-06 1.038e-09 -1.510e-05 0.000e+00
## 5   1.63 -2.617e-06 0.000e+00 -3.613e-06 0.000e+00
## 6   1.59 -1.827e-06 5.523e-08 -2.597e-06 0.000e+00
## 7   1.55 -2.395e-06 3.822e-09 -2.329e-06 0.000e+00
## 8   0.00 -1.986e-08 0.000e+00 -6.720e-08 0.000e+00
## 9   0.00 -4.492e-06 0.000e+00 -1.512e-05 0.000e+00
## 10  2.34 -1.487e-05 0.000e+00 -7.015e-06 0.000e+00
## 11  2.25 -1.119e-05 0.000e+00 -6.524e-06 0.000e+00
## 12  2.21 -1.406e-05 0.000e+00 -9.305e-06 0.000e+00
## 13  2.27 -2.178e-05 2.778e-09 -1.050e-05 1.389e-09
## 14  2.52 -1.354e-05 0.000e+00 -7.716e-06 0.000e+00
## 15  2.19 -1.220e-05 0.000e+00 -1.060e-05 0.000e+00
## 16  2.33 -1.509e-05 2.556e-08 -8.127e-06 0.000e+00
## 17  2.29 -1.355e-05 6.805e-09 -9.277e-06 0.000e+00
## 18  2.33 -1.649e-05 5.556e-09 -1.308e-05 0.000e+00
```

```r
###########################################################################
# SOLVE FOR THE STRAINS USING CALLAHAN'S MODEL
# TEST SC1B
###########################################################################

# ==== "events" function, for specifying step functions ====
# ---- SETS STRAINS AND RATES = 0 AT TIME == 0 ----
eventfun <- function(Time, State, Parm){
  with (as.list(State),{

    # ---- if time == 0, derivative = 0 ----
    DZ <- ifelse(c(Time == 0, Time == 0, Time == 0), {c(0, 0, 0)}, {DZ})

    return(DZ)
  })
}

# ==== DIFFERENTIAL EQUATION ====
STRAINS.02 <- function(Time, State, Parm){
  with(as.list(c(State, Parm)),{
    # function for calculating axial and lateral strain rates
    # Input must be in vector or matrix form, no data frames
    # Eqns. referenced from:  SAND97-2601
    # CPar: EAT0, ETA1, ETA2, NF, AA1, PP, NSP, R1, R3, R4, QSR
    # FPar: KAP0, KAP1, KAP2, NK, DDT
    # TestData:

    # ============= parameters hard coded into function directly ========
    # browser()
    KAP0 <- 10.119
    KAP1 <- 1.005
    DDT  <- 0.896
    NK   <- 1.331
    KAP2 <- 1

    ETA0  <- 0.102854        # -
    ETA1  <- 3.9387          # -
    ETA2   <- 1               # constant -
    NF          <- 3.5122          # -
    AA1         <- 0.3147          # -
    PP          <- 1.6332          # -
    NSP         <- 0.557621        # -
    R1          <- 1.041 * 10 ^ -6 # [K/(MPa-sec)]
    R3          <- 15.1281         # -
    R4          <- 0.1677765       # -
    QSR         <- 1077.46         # [K]

    # ---- Munson-Dawson Creep Parameters (17) ----
    A1          <- 8.386e22
    A2          <- 9.672e12
    Q1R         <- 12581
    Q2R         <- 5033
    N1          <- 5.5
    N2          <- 5.0
    B1          <- 6.0856e6
    B2          <- 3.034e-2
    Q           <- 5335
    S0          <- 20.57
    M           <- 3
```

```
K0          <- 6.275e5
C           <- 9.198e-3
ALPHA <- -17.37
BETA        <- -7.738
DELTA <- 0.58
MU          <- 12400

# ---- fitting assumptions ----
RHOIS <- 2160.0  # ASSUMED IN SITU SALT DENSITY

# ---- interpolated input variables ----
TIME <- time.interp(Time)
# browser()
TEMP <- temp.interp(Time)
AS   <- as.interp(Time)
LS   <- ls.interp(Time)
D    <- d.interp(Time)

# ---- calculate variables ----
MS  <- (2.0 * LS + AS) / 3  # MEAN STRESS
DS  <- LS - AS                                    # STRESS DIFFERENCE
#   ELC <- (EVC - EAC) / 2   # CREEP TRUE LATERAL STRAIN
D0  <- 1382.4 / RHOIS                         # EMPLACED FRACTIONAL DENSITY (0.64 FRAC DENSITY)
DI  <- RHOI / RHOIS                     # FRACTIONAL DENSITY at the start of creep

# ==== this portion has been moved to lambda <- function() =====
#WT1 <- DT / NTIME   # WEIGHTING FUNCTION FOR CREEP CONSOLIDATION PARAMETERS
#WT  <- 1   # WEIGHTING FUNCTION FOR FLOW PARAMETERS
# ==============================================================
# integral of Eqn 2-27, (initial values)

# ==== define the differential equation ====
# browser()
VOL         <- Z1 + 2*Z2                        # TRUE VOLUMETRIC STRAIN
VOLT        <- VOL + log(D0/DI)      # VOLUMETRIC STRAIN + INITIAL TRUE STRAIN ESTIMATE
DEN         <- DI/exp(VOL)                  # CURRENT FRACTIONAL DENSITY

#     ifelse(D >= 1,{
#       MD <- 0  # if fractional density is 1, disclocation creep = 0
#       SP <- 0},# if fractional density is 1, pressure solutioning = 0
#     {VAR <- ifelse(DEN <= DDT, DDT, DEN) # DEFINE DENSITY CEILING ISH

VAR <- ifelse(DEN <= DDT, DDT, DEN) # DEFINE DENSITY floor ISH
# ==== DEBUG ====
DEBUG.VAR <- ifelse(DEN <= DDT, 1, -1)

# ---- Equivalent Stress ----
OMEGAA      <- ((1 - DEN) * NF / (1 - (1 - DEN)^(1/NF))^NF)^(2/(NF + 1))
OMEGAK      <- ((1 - VAR) * NK / (1 - (1 - VAR)^(1/NK))^NK)^(2/(NK + 1))
ETA         <- ETA0 * OMEGAA^ETA1
KAP         <- KAP0 * OMEGAK^KAP1
TERMA       <- ((2 - DEN)/DEN)^((2 * NF)/(NF + 1))
TERMK       <- ((2 - DEN)/DEN)^((2 * NK)/(NK + 1))

# ---- Eqn. 2-3 (SAND97-2601) ----
# Equivalent stress measure for Disl. Creep and Press Sol'ing
SEQF        <- sqrt(ETA * MS^2 + ETA2 * TERMA * DS^2)
# Equivalent stress measure for Flow Potential
SEQ         <- sqrt(KAP * MS^2 + KAP2 * TERMK * DS^2)
# ---- Eqn. 2-17 (SAND97-2601) ----
ALPHA2      <- KAP * MS / 3
BETA2       <- KAP2 * TERMK * DS

# ---- Eqn. 2-20, WithOUT dislocation creep and pressure solutioning ----
F2A <-      (ALPHA2 - BETA2)/SEQ        # fit to axial strains
F2L <-      (ALPHA2 + 0.5 * BETA2)/SEQ  # fit to lateral strains
F2V <-  3 * ALPHA2 / SEQ            # fit to volumetric strains

# ==== START: equivalent inelastic strain rate form for dislocation creep ====

# ---- Steady State Strain Rate Calc ----
ES1 <- A1 * (SEQF / MU)^N1 * exp(-Q1R/TEMP) # Dislocation climb - Eqn. 2-30
ES2 <- A2 * (SEQF / MU)^N2 * exp(-Q2R/TEMP) # Undefined Mechanism - Eqn. 2-31

# Slip - Eqn. 2-32 (SAND98-2601)
H   <- SEQF - S0 # HEAVISIDE FUNCTION
ARG <- Q * (SEQF - S0) / MU
ES3 <- ifelse(H >= 0, 0.5 * (B1 * exp(-Q1R / TEMP) + B2 * exp(-Q2R / TEMP)) *
                (exp(ARG) - exp(-ARG)),0)
# ==== DEBUG ====
DEBUG.ES3 <- ifelse(H >=0, 1, -1)

ESS = ES1 + ES2 + ES3 # Steady-state strain rate, Eqn. 2-29 (SAND97-2601)

# ---- EVALUATE TRANSIENT FUNCTION, 3 branches: work hardening, equilibrium, recovery
EFT  <- K0 * exp(C * TEMP) * (SEQF / MU) ^ M  # Transient Strain Limit, Eqn. 2-28
BIGD <- ALPHA + BETA * log10(SEQF / MU)       # Work-Hardening parameter, Eqn 2-28

FU <- ifelse(Z3 == EFT, 1, ifelse(Z3 < EFT, exp(BIGD * (1 - Z3 / EFT) ^ 2),
                                  exp(-DELTA * (1 - Z3 / EFT) ^ 2)))

# ==== DEBUG ====
DEBUG.FU <- ifelse(Z3 == EFT, 0, ifelse(Z3 < EFT, -1, 1))

MD <- FU * ESS  # equivalent inelastic strain rate form for dislocation creep, Eqn 2-23

# ==== START: Equivalent Inelastic Strain Rate Form for Pressure Solutioning ====
# ---- Calculate initial volumetric strain - Based on spherical packing ----
CR <- abs(exp(VOLT) - 1) # USES THE DEFINITION OF ENGINEERING STRAIN

# ---- Determine functional form - either large or small strains, Eqn 2-34 ----
GAMMA <- ifelse(CR <= 0.15, 1, (abs((D0 - exp(VOLT)) / ((1 - D0) * exp(VOLT)))) ^ NSP)
# Small Strains (Vol Strain > - 15%)
# Large Strains (Vol Strain < - 15%)
# ==== DEBUG ====
DEBUG.GAMMA <- ifelse(CR <= 0.15, 1,-1)

# ---- component of eqn 2-35 ---
X3 <- exp((R3 - 1) * VOLT) / (abs(1 - exp(VOLT))) ^ R4

# ---- determine value of moisture function (w) ----
M2 <- ifelse (W == 0, 0, W ^ AA1)

# ---- Equivalent Inelastic Strain Rate Form for Pressure Solutioning, Eqn 2-35
G2 <- 1 / DD ^ PP # calculate grain size function
T2 <- exp(-QSR / TEMP) / TEMP
SP <- R1 * M2 * G2 * T2 * X3 * GAMMA * SEQF #})
```

```
    DZ1 <- (MD + SP) * F2A # derivative: axial strain rate
    DZ2 <- (MD + SP) * F2L # derivative: lateral strain rate
    DZ3 <- (FU - 1) * ESS  # derivative of internal variable "xi"
    #     browser()
    DZ <- list(c(DZ1, DZ2, DZ3), MD, FU, ESS, ES1, ES2, ES3, SP, DZ1, DZ2,
               DZ3, F2A, F2L, EFT, SEQ, SEQF, BIGD, DEBUG.GAMMA,
               DEBUG.ES3, AS, LS, OMEGAA, OMEGAK,VAR, DEN)

    return(DZ)
  })
}
```

```
PAR.TEST <- DATA.INP[which(DATA.INP$ITEST == "SC1B"),] # SUBSET OF DATA FOR ANALYSIS
# debug.out <- paste(CurrentDirectory,"debug_SC1B.csv",sep = "/")
# write.table(ODE.DT,file = debug.out, sep = ",")

# ---- linear interpolation functions to be called in "strain_Rates.01" ----
time.interp <- approxfun(x = PAR.TEST$TIME, y = PAR.TEST$TIME)
temp.interp <- approxfun(x = PAR.TEST$TIME, y = PAR.TEST$TEMP)
# as.interp   <- approxfun(x = PAR.TEST£TIME, y = PAR.TEST£AS)
# ls.interp   <- approxfun(x = PAR.TEST£TIME, y = PAR.TEST£LS)
as.interp   <- approxfun(x = DATA.BEAN$TIME, y = DATA.BEAN$STRESS_YY / 10^6)
ls.interp   <- approxfun(x = DATA.BEAN$TIME, y = DATA.BEAN$STRESS_XX / 10^6)

d.interp    <- approxfun(x = PAR.TEST$TIME, y = PAR.TEST$D)

RHOI   <- as.numeric(PAR.TEST$RHOI[1])  # DENSITY AT THE START OF CREEP
DD              <- as.numeric(PAR.TEST$DD[1])   # AVERAGE GRAIN SIZE [MM]
W               <- as.numeric(PAR.TEST$W[1])    # WATER CONENT BY PERCENT WEIGHT

PARM <- c(RHOI = RHOI, DD = DD, W = W) # CONSTANT TEST SPECIFIC PARAMETERS

# ---- intial values for state variables ----
Z1      <- 0 # Predicted axial strain (initial values)
Z2      <- 0 # Predicted lateral strain (initial values)
Z3      <- 0 # internal variable "xi" for the transient function (FU)
# integral of Eqn 2-27, (initial values)

IC <- (c(Z1 = Z1, Z2 = Z2, Z3 = Z3)) # array of initial values

TIME <- DATA.BEAN$TIME

# ---- function for Predicting the Creep Strain(E) Rates ----
P.CER <- ode(func = STRAINS.02, parms = PARM, y = IC,
             times = TIME, verbose = TRUE)#, events = list(func = eventfun, time = 0),

##
## --------------------
## Time settings
## --------------------
##
##   Normal computation of output values of y(t) at t = TOUT
##
## --------------------
## Integration settings
## --------------------
##
##   Model function an R-function:
##   Jacobian not specified
```

```
##
##
## --------------------
## lsoda return code
## --------------------
##
##   return code (idid) =  2
##   Integration was successful.
##
## --------------------
## INTEGER values
## --------------------
##
##   1 The return code : 2
##   2 The number of steps taken for the problem so far: 116
##   3 The number of function evaluations for the problem so far: 238
##   5 The method order last used (successfully): 4
##   6 The order of the method to be attempted on the next step: 4
##   7 If return flag =-4,-5: the largest component in error vector 0
##   8 The length of the real work array actually required: 68
##   9 The length of the integer work array actually required: 23
##  14 The number of Jacobian evaluations and LU decompositions so far: 0
##  15 The method indicator for the last succesful step,
##            1=adams (nonstiff), 2= bdf (stiff): 1
##  16 The current method indicator to be attempted on the next step,
##            1=adams (nonstiff), 2= bdf (stiff): 1
##
## --------------------
## RSTATE values
## --------------------
##
##   1 The step size in t last used (successfully): 252000
##   2 The step size to be attempted on the next step: 252000
##   3 The current value of the independent variable which the solver has reached: 5548000
##   4 Tolerance scale factor > 1.0 computed when requesting too much accuracy: 0
##   5 The value of t at the time of the last method switch, if any: 0
##
```