doi:10.1038/nature19798

Evolution of global temperature over the past two million years

Carolyn Snyder

Supplementary Methods

R code for interpolation and dating uncertainty estimates for sea-surface temperature (SST) reconstructions

As described in the Methods, this research uses a weighted average of the SST reconstruction from each time point on the 1kyr timescale ("time.new") to interpolate each SST reconstruction ("variable.new") to a common 1kyr timescale and to estimate dating uncertainty ("variable.new.sd") when comparing independent records using uncertain absolute timescales (R. Samsworth was instrumental in developing this method). The following R code can be used to implement the method.

```
# Step 1: Input data
        # For SST reconstruction define: 1) time ("time.old"); 2) uncertainty in time ("time.old.sd"); 3) SST ("variable.old"); and 4) uncertainty in SST ("variable.old.sd").
        # Define new timescale
                time.new <- (1:2000)*1000

# Step 2: Interpolate raw SST and SST-sd estimates to the new timescale
        time.old.sd.int <- spline (time.old, time.old.sd, xout = time.new)$y
        variable.old.sd.int <- spline (time.old, variable.old.sd, xout = time.new)$y

# Step 3: Estimate SST at new timescale using weighted averages
        variable.new <- rep(NA, length(time.new))
        for (i in 1:length(time.new)) { variable.new[i] <- ksmooth (time.old, variable.old, kernel =
                "normal", bandwidth = time.old.sd.int[i], x.points = time.new[i])[[2]] }

# Step 4:  Estimate new variance including the variance from the dating uncertainty plus from uncertainty in the variable
        Residuals <- (variable.old %o% rep(1,length(time.new)) - rep(1,length(time.old)) %o%
                variable.new)^2
        # estimating uncertainty from dating
                variable.new.var.dating <- rep(NA,length(time.new))
                for (j in 1:length(time.new)) {variable.new.var.dating[j] <- ksmooth (time.old,
                        Residuals[,j], kernel= "normal", bandwidth= time.old.sd.int[j], x.points =
                        time.new[j])[[2]] }
        # summing variances from both dating uncertainty and variable uncertainty
                variable.new.var.total <- (variable.new.var.dating + (variable.old.sd.int)^2
                variable.new.sd <- sqrt(variable.new.var.total)
```

R code for creating global average surface temperature (GAST) reconstruction

As described in the Methods, this research estimates GAST from local SST proxy-based reconstructions, and the following R code can be used to implement the method.

```
# Step 1: Input the SST reconstructions
    # Define "Temp.data" as a table of the SST estimates for each reconstruction every 1kyr using
        the method described above (2000 rows, 61 columns)
    # Define "Temp.SD.data" as a table of the SST uncertainty estimate for each reconstruction
        every 1kyr using the method described above (2000 rows, 61 columns)
    # Define v.latitude as the latitude for each SST reconstruction
    time <- (1:2000) * 1000; tt = 2000
    n.cores = 61
    n.bootstrap.sim <- 2000 # number of simulations per method, resampling records and adding
        measurement noise

# Step 2: Estimate average SST values for latitudinal zones using a variety of possible spatial weighting
schemes and combine to estimate average SST over 60ºN to 60ºS
    # Loop over zonal methods

    ## Not assuming North-South Symmetry and 4 zones
    # make vector of different zone definitions
    latitude.zones.N.S.4 <- rbind(
        c(-30, 0, 30), #equal degrees
        c(-25.7, 0, 25.7),  #equal areas
        c(-20, 0, 20),
        c(-35, 0, 35))
    n.methods.sim <- dim(latitude.zones.N.S.4)[1]  #simulations for each zonal boundary
        specifications
    Sim.zonal.N.S.4 <- matrix (nrow = tt, ncol = n.bootstrap.sim * n.methods.sim)

    #loop over lat zone definitions
    for (l in 1:n.methods.sim){
        lat.zones.d <- latitude.zones.N.S.4[l,]
        lat.zones.r <- lat.zones.d * pi/180
    #bootstrap for randomly selecting a record and random noise for each record ~(0,sd)
        for (i in 1:n.bootstrap.sim) {
            #creating subset
            ii <- sample (n.cores, replace = TRUE)
            Temp.ss <- Temp.data[,ii]
            Temp.SD.data.cut <- Temp.SD.data[,ii]
            v.latitude.ss <- v.latitude[ii]
            #add random noise to each core based on SD estimate per time point for each core
            Random.error <- matrix (0, nrow = tt, ncol = n.cores)
            for (j in 1:n.cores){ Random.error[,j] <- rnorm(tt, mean = 0,sd =
                Temp.SD.data.cut[,j]) }
            Temp.ss.rn <- Temp.ss[,] + Random.error

            #Calculate zonal averages with core subset and noise
```

```
                dd = 1; Z1 <- Temp.ss.rn[, (v.latitude.ss <= lat.zones.d[dd])]
                dd = 2; Z2 <- Temp.ss.rn[, (v.latitude.ss <= lat.zones.d[dd] & v.latitude.ss >
                        lat.zones.d[dd-1])]
                dd = 3; Z3 <- Temp.ss.rn[, (v.latitude.ss <= lat.zones.d[dd] & v.latitude.ss >
                        lat.zones.d[dd-1])]
                dd = 4; Z4 <- Temp.ss.rn[, (v.latitude.ss > lat.zones.d[dd-1])]
                nn <- matrix (NA, nrow=tt, ncol=2) #add extra NA in case the set is empty,
                        otherwise get errors
                Temp.Z1 <- rowMeans(cbind(Z1, nn), na.rm = T)
                Temp.Z2 <- rowMeans(cbind(Z2, nn), na.rm = T)
                Temp.Z3 <- rowMeans(cbind(Z3, nn), na.rm = T)
                Temp.Z4 <- rowMeans(cbind(Z4, nn), na.rm = T)

                #Spatially-weighted average, where sum of weights is used to normalize
                #Global.SST.N.S.4 record has NA for every point where at least one latitude zone
                        is empty
                Global.SST.N.S.4 <- (( Temp.Z1 * (sin(lat.zones.r[1]) - sin(-60 * pi/180)) +
                        Temp.Z2 * (sin(lat.zones.r[2]) - sin(lat.zones.r[1]))
                        +  Temp.Z3 * (sin(lat.zones.r[3]) - sin(lat.zones.r[2]))
                        +  Temp.Z4 * (sin(60 * pi/180) - sin(lat.zones.r[3])) )
                        / (2 * (sin(60 * pi/180))))
                Sim.zonal.N.S.4[,i + (l-1)*n.bootstrap.sim]<- Global.SST.N.S.4
                        }
                }

## Not assuming North-South Symmetry and 6 zones
# make vector of different zone definitions
latitude.zones.N.S.6 <- rbind(
        c(-40, -20, 0, 20, 40),  #equal degrees
        c(-35.3, -16.8, 0, 16.8, 35.3),  #equal areas
        c(-30, -15, 0, 15, 30),
        c(-40, -15, 0, 15, 40),
        c(-35, -20, 0, 20, 35))
n.methods.sim <- dim(latitude.zones.N.S.6)[1] #simulations for each zonal boundary
        specifications
Sim.zonal.N.S.6 <- matrix (nrow = tt, ncol = n.bootstrap.sim * n.methods.sim)

#loop over lat zone definitions
for (l in 1:n.methods.sim){
        lat.zones.d <- latitude.zones.N.S.6[l,]
        lat.zones.r <- lat.zones.d * pi/180

#bootstrap for randomly selecting a record and random noise for each record ~(0,sd)
        for (i in 1:n.bootstrap.sim) {
                #creating subset
                ii <- sample (n.cores, replace = TRUE)
                Temp.ss <- Temp.data[,ii]
                Temp.SD.data.cut <- Temp.SD.data[,ii]
                v.latitude.ss <- v.latitude[ii]
```

```
                #add random noise to each core based on SD estimate per time point for each core
                Random.error <- matrix (0, nrow = tt, ncol = n.cores)
                for (j in 1:n.cores){ Random.error[,j] <- rnorm(tt, mean = 0,sd =
                        Temp.SD.data.cut[,j]) }
                Temp.ss.rn <- Temp.ss[,] + Random.error

                #Calculate zonal averages with core subset and noise
                dd = 1; Z1 <- Temp.ss.rn[, (v.latitude.ss <= lat.zones.d[dd])]
                dd = 2; Z2 <- Temp.ss.rn[, (v.latitude.ss <= lat.zones.d[dd] & v.latitude.ss >
                        lat.zones.d[dd-1])]
                dd = 3; Z3 <- Temp.ss.rn[, (v.latitude.ss <= lat.zones.d[dd] & v.latitude.ss >
                        lat.zones.d[dd-1])]
                dd = 4; Z4 <- Temp.ss.rn[, (v.latitude.ss <= lat.zones.d[dd] & v.latitude.ss >
                        lat.zones.d[dd-1])]
                dd = 5; Z5 <- Temp.ss.rn[, (v.latitude.ss <= lat.zones.d[dd] & v.latitude.ss >
                        lat.zones.d[dd-1])]
                dd = 6; Z6 <- Temp.ss.rn[, (v.latitude.ss > lat.zones.d[dd-1])]
                nn <- matrix (NA, nrow=tt, ncol=2) #add extra NA in case the set is empty,
                        otherwise get errors
                Temp.Z1 <- rowMeans(cbind(Z1, nn), na.rm = T)
                Temp.Z2 <- rowMeans(cbind(Z2, nn), na.rm = T)
                Temp.Z3 <- rowMeans(cbind(Z3, nn), na.rm = T)
                Temp.Z4 <- rowMeans(cbind(Z4, nn), na.rm = T)
                Temp.Z5 <- rowMeans(cbind(Z5, nn), na.rm = T)
                Temp.Z6 <- rowMeans(cbind(Z6, nn), na.rm = T)

                #Spatially-weighted average, where sum of weights is used to normalize
                #Global.SST.N.S.6 record has NA for every point where at least one latitude zone
                        is empty
                Global.SST.N.S.6 <- (( Temp.Z1 * (sin(lat.zones.r[1]) - sin(-60 * pi/180))
                        +  Temp.Z2 * (sin(lat.zones.r[2]) - sin(lat.zones.r[1]))
                        + Temp.Z3 * (sin(lat.zones.r[3]) - sin(lat.zones.r[2]))
                        + Temp.Z4 * (sin(lat.zones.r[4]) - sin(lat.zones.r[3]))
                        + Temp.Z5 * (sin(lat.zones.r[5]) - sin(lat.zones.r[4]))
                        + Temp.Z6  * (sin(60 * pi/180) - sin(lat.zones.r[5])) )
                        / (2 * (sin(60 * pi/180))))
                Sim.zonal.N.S.6[,i + (l-1)*n.bootstrap.sim]<- Global.SST.N.S.6
                }
        }

## Make the representative SST 60ºN to 60ºS ensemble
#first remove the ones with all the NA entries
        NA.vector <- apply(Sim.zonal.N.S.4, 2, function(x)  sum(is.na(x)));
                Sim.zonal.N.S.4.NA1 <- Sim.zonal.N.S.4[ ,NA.vector != tt]
        NA.vector <- apply(Sim.zonal.N.S.6, 2, function(x)  sum(is.na(x)));
                Sim.zonal.N.S.6.NA1 <- Sim.zonal.N.S.6[ ,NA.vector != tt]
ZONAL.raw <- cbind(Sim.zonal.N.S.4.NA1,Sim.zonal.N.S.6.NA1)
ZONAL.final <- ZONAL.raw[,sample(dim(ZONAL.raw)[2],2000,replace = FALSE)]
```

```
# Step 3: Scale from latitudinal SST averages to GAST using scalars estimated from PMIP model
experiments (see Methods for details), using scalars of mean 1.9 and sd 0.2 (uniform distribution is
assumed)
        n.sim <- 5000
        OUTPUT.scaled <- matrix (NA, ncol = n.sim, nrow = tt)
        for (j in 1: n.sim){
                zonal.scalar.value <- rnorm(1, 1.9, 0.2)
                ii <- sample(1:dim(ZONAL.final)[2],1)
                OUTPUT.scaled[,j] <- (ZONAL.final [,ii] - mean(ZONAL.final [1:5, ii], na.rm = T)) *
                zonal.scalar.value
                }
```