Supplementary Material for 'Cenozoic pelagic accumulation rates and biased sampling of the deep sea record'

Johan Renaudie[1] David B. Lazarus[1]

[1]Museum für Naturkunde, Leibniz-Institut für Evolutions- und Biodiversitätsforschung, Invalidenstrasse 43, 10115 Berlin, Germany.

December 18, 2023

Contents of the Supplementary Material

- Supplementary Text 1: The Drilling Bias Model.
- Supplementary Figures 1 and 2: Results similar to Figures 2 and 3 in the main text but reporting mean rather than median value. Shown here only to allow direct comparison to published literature.
- Supplementary Code 1: Python code for the drilling bias model.
- Supplementary Data 1: The full Neptune Database Age model Library.
- Supplementary Data 2: Table of LSR and SAR implied by the Neptune Database Age Model Library.
- Supplementary Data 3: The output of the compilation and computations presented in the paper.

Supplementary Text 1: The Drilling Bias Model

Our sediment accumulation rate bias analysis was inspired by our experience of creating age models for the NSB database over three decades and thus what sedimentation history looks like in deep sea sections, and as well our awareness that the recovered geologic record has been shown in other areas of earth science research (e.g. paleobiology) to be strongly biased, both by the nature of geologic sedimentation, and by human biases in sampling it. We decided therefore to model how deep sea drilling sampling of a largely layer-cake sedimentary record would be biased by the sampling process, and by geologic factors e.g. hiatuses, affecting the sedimentation history of the record itself. The idea and initial version of the model was by dbl, and then slightly modified by jr, in particular to create the final runs which included variation analyses.

The model is conceived to create a simulated record of deep-sea drilled sections that share, as much as possible, the characteristics listed above: variation in sedimentation rate between sites; and (for simplicity) constant accumulation rate for any given site. Each site has a total depth of penetration drawn from the actual distribution of drilling depths achieved by the deep sea drilling programs, and an initial sedimentation rate chosen at random from the distribution of sedimentation rates seen in the NSB database for Pleistocene sediments (as these rates have not been subtantially altered by compaction, or hiatus development that increasingly affect older parts of sections). Calculating the distribution of sedimentation rates from NSB was chosen after a literature search failed to find a robust global estimate for pelagic sedimentation rates - all studies found either included non pelagic sedimentation, were only for smaller regions, or had other disqualifying issues. Use of NSB data also insured our model parameter would be fixed to a value that is appropriate for the sections used in the rest of the model. After creating initial versions of the code the actual distributions of site depth penetration and sedimentation rates were replaced by fitted functions to these data, to insure that minor variations/incompleteness in the actual NSB data did not create anomalies in the model output. For site depths the fitted function was a gamma distribution of shape $\kappa = 2.93$ and scale $\theta = 166.97$, and for sedimentation rates (a translated Weibull distribution of shape $\kappa = 0.95$, scale $\lambda = 3.34$ and location $\theta = 0.4$). Similarly, hiatuses are added to sections using a random generator based on the observed frequency and duration of hiatuses (in Myr) in the NSB age models dataset.

The code works as follows: a results array ('resar'), with the dimensions geologic age (e.g. 1 Myr bins, 0-65 my) by number of simulated sections (as chosen for the run) is created and initialized to null values. It is then filled, one section at a time, using the, for this section, randomly chosen sedimentation rate (in m/Myr), randomly chosen total penetration (in meters) and a globally averaged compaction vs depth curve - a simple linear fit derived from the IODP database - to calculate how many 1 Myr time bins will have been penetrated. The resar time bins for each section are filled incrementally, and each bin that is filled is assigned the adjusted sedimentation rate for this section and time interval (the original sedimentation rate, adjusted to lower values with increasing depth due to compaction). After filling the resar array with simulated sections and their adjusted sedimentation rate values per time bin, the vectors representing each section are conditionally adjusted by adding hiatuses (via the hiatus length random generator) that shift age bins downwards by the duration of the hiatus, if a random number call returns a value greater than the probability of hiatus initiation. The hiatus created gaps in sedimentation are filled with null values.

At the end of each run a vector of the average adjusted sedimentation rate per time interval across all simulated sections is calculated (ignoring null values) and saved. Other than this relatively

small decline in relative sedimentation rate due to compaction, the only other factor affecting the apparent sedimentation rate per time interval is the non-uniform ability of high vs low sedimentation rate sections to penetrate far enough back in time to create a filled resar time interval bin. Multiple runs with adjusted values for various parameters were carried out to insure that the model worked correctly and produced reasonable results. Final runs (by jr) added via additional loop variables a systematic exploration of how key parameters affected results, providing thus an estimate of the range of error in the simulated output. The mean and median values of relative sedimentation rate decline with increasing geologic age - due to the bias in sampling older time intervals - was used as a correction factor for the observed accumulation rate vs time in the actual NSB dataset.

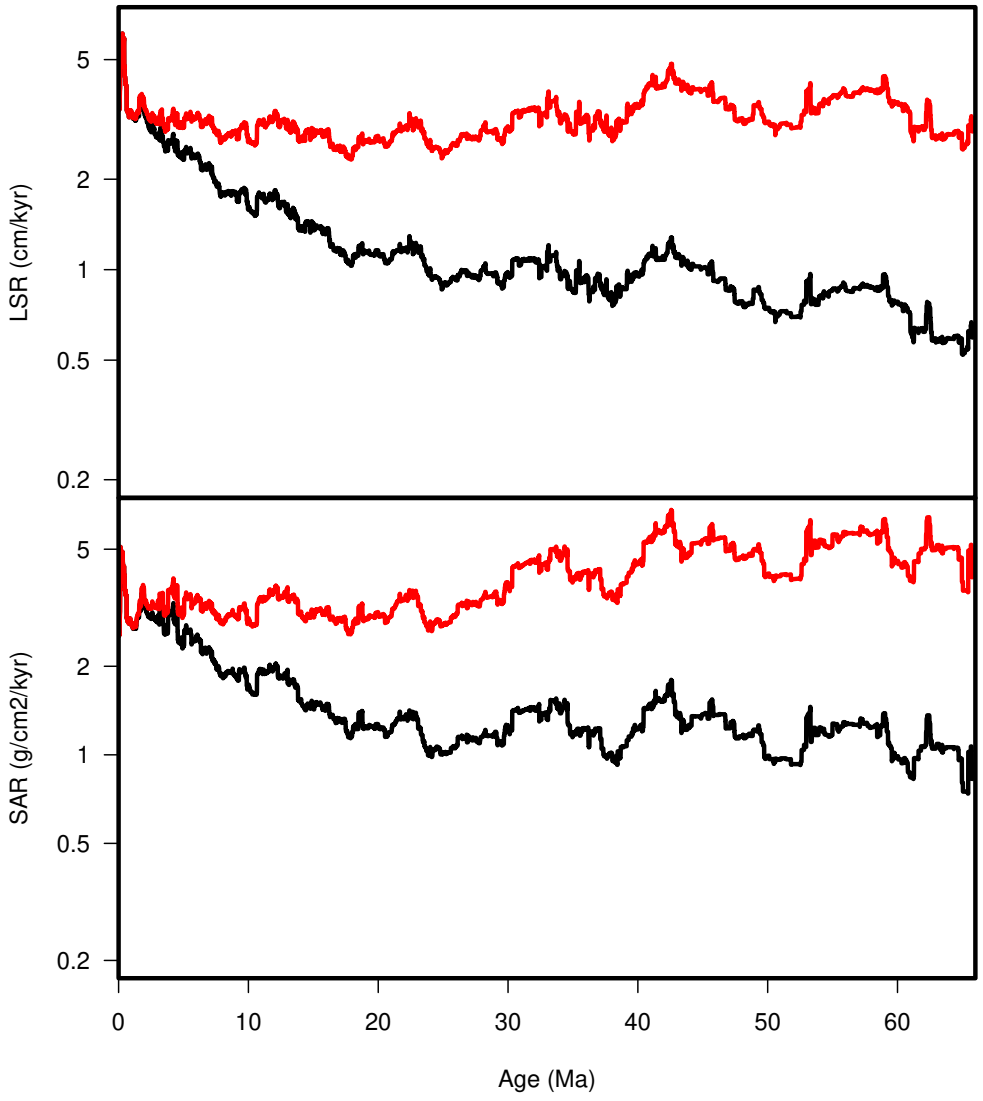
The code is included as part of this SOM. It is written in Python v3.9 (Van Rossum and Drake, 2009) using the numpy and scipy modules (Virtanen et al., 2020), and is annotated to the extent that it should be largely readable even to someone who does not know the Python language.

References

Van Rossum, G. and Drake, F. L. (2009). *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA.

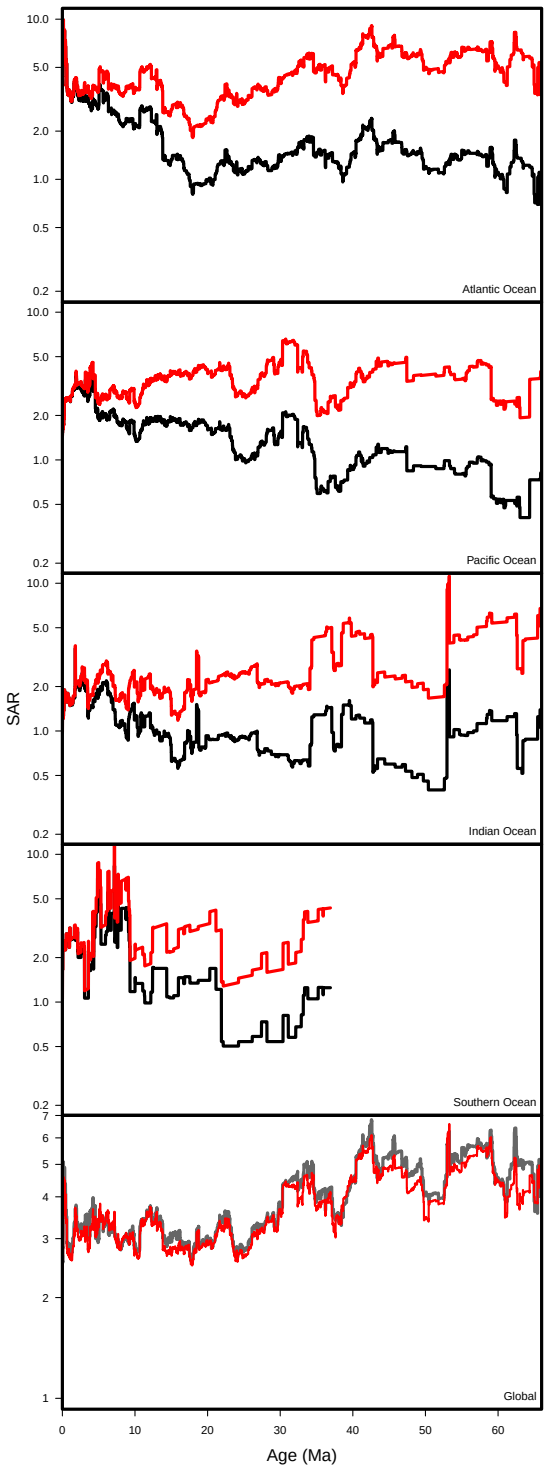
Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C. J., Polat, İ., Feng, Y., Moore, E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., and SciPy 1.0 Contributors (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272.

Supplementary Figure 1



Linear Sedimentation Rates (top; expressed in $cm.kyr^{-1}$) and Accumulation Rates (bottom panel; expressed in $g.cm^{-2}.kyr^{-1}$). Black: mean of the distribution as observed in our global dataset; Red: mean sedimentation and accumulation rates corrected for drilling bias using the result of the model). Rates are on a log-scale.

Supplementary Figure 2



Mean Sediment Accumulation Rates (expressed in $g.cm^{-2}.kyr^{-1}$) observed in each ocean basin (black: mean; red: mean corrected for drilling bias using the result of the model), shown here on a log-scale. Bottom panel show the weighted global composite based on each basin corrected mean value (in red) compared to the corrected but area-unweighted global mean SAR shown in Supplementary Figure 2.

Supplementary Code 1

```
# -*- coding: utf-8 -*-
# Sampling bias in estimating sedrates from deep sea or other drilled sections
# Uses distribution of maximum depths from compiled data in external file (sites_maxdepth.txt)
# Initial runs use all DSDP to IODP sites (other than basalt sites, logging or other technical
# sites, or sites with no age info at SEDIS)
# Sedrate is assumed constant over time at any site.
# Sedrate value for site chosen from plausible random distribution
# (here Weibull fit by JR to all NSB Pleistocene sedrates)

# Version 0.2 7.3.22        - first version that runs and creates apparently valid output
#         v 0.21 15.3     - changed Rayleigh spread to 1 in sedrate generator to fit NSB values
#         v 0.30 7.5        - resar fill section replaced with new compaction honoring code
#         v 0.40 25.5     - Rayleigh replaced with Weibull to generate sedrates
#                           (after JR test of best function fits),
#                           directory path now user input not hardwired, metadata and
#                           timestamp added to output file
#         v 0.41 27.5             - added loc=.4 for Weibull
#         v 0.50 15.6     - added hiatus code block
#         v 0.51 16.6     - another attempt to get hiatus indexing right
#         v 0.52 17.6     - corrected missing 10x factor on sedrate - was there earlier but got lost in other editing
#         v 0.53 19.6     - rewrote hiatus code block after examination of resar still shows errors. Now works
#         v 0.54 22.6     - cleaning up. Remove unused/unneeded imports (os, pandas). More comments
#         v 0.55 22.6     - added break to hiatus loop when base age w. sedrate reached: avoid unnecessary calcs
#         v 0.60 26.7     - added outer loop to repeat the simulation x times.
#                           added randomizer for maxdepth based on modelled distribution.
#                           made the final save file a json dump of everything. (JR)

# by Dave Lazarus

# Program logic: create 2D array resar: age bins x number of virtual sites to hold sedrates result
# For each virtual site:
# Pick a random sedrate and maximum drilled depth
# Fill array bins up to resultant max age reached with the sedrate value of each virtual site, adjusting for compaction
# Then when all virtual sites done:
# Insert hiatuses in sections
# Calculate average sedrate for each age bin across all virtual sites and output result

# -- load needed libraries

import numpy as np
from scipy.stats import weibull_min, gamma
import random as ran
from datetime import datetime
import orjson

ran.seed(20220726)
timestampstr=datetime.now().strftime("%Y-%m-%d %Hh%Mm%Ss")
output = 0 #0 saves parameters and averaged results; 1 saves full result (might take up to 1h30 to save the ca. 7Gb file)
# -- define constants

step=1 # binning interval, e.g. 1 my
maxtime=65  # for Cenozoic, other number for different time range
nbins=int(maxtime/step)
ntries=10000  # number of virtual holes in a simulation run
nsim=1000

# Bulk density increase as linear function of depth: bulk density(d)=bulk(0)+bulkdinc*d
#       slope and intercept values from IODP db compilation provided by JR
bulkd0=1.0996
#bulkdinc=0.00079851
bulkdinc=0
# Sedimentation rate simulated curve, from JR bestfit find of Weibull curve to Pleistocene sedrates in NSB
# Weibull curve parameters are shape, scale, location
wshape=.95
wscale=3.34
wloc=.4
# hiatus parameters. Currently based on all NSB locs w. quality G or better
hiprob=.02278
hishape=.95
hiscale=4
#Hole depths parameters. Gamma distribution.
dshape=2.93
dscale=166.67

#--def avesedrate, maxages vectors, set to NaN

avesedrate=np.empty((nbins,nsim))
avesedrate[:]=np.NaN
maxages=np.empty((ntries,nsim)) # maximum age penetration before hiatus additions
maxages[:]=np.NaN
holedepths=np.empty((ntries,nsim))

#--def resar age bin x n virtual holes array as float number with nbins, ntries as indices.
        # this is the result array of sedrates

resar=np.empty((nbins,ntries,nsim))
resar[:]=np.NaN
```

```
#--define sedratedist list, ntries long with mean and sd values
        # note can't have negative sedrates
        # in m/m.y. units
        # here use Weibull function as found by JR in tests to give best fit to actual Pleist.
    # ie non-compacted sedrate compilation from NSB
print('Step 1 - calculating sedimentation depths')
# -- fill resar :
for s in range(nsim):
    sedrate=10*weibull_min.rvs(wshape, scale=wscale, loc=wloc, size=ntries)
    #10* to give m/my, not cm/kyr as from weibull fn
    print(str(s), end="\r")
    for j in range(ntries):
        #-- pick random value for maxdepth for each virtual section from maxdepths
        #maxdepth=np.random.choice(maxdepths)
        maxdepth=gamma.rvs(dshape,scale=dscale)
        holedepths[j,s]=maxdepth

        # incrementally move downhole for hole j by time bin i, adding depth in decreasing increments
        #        due to increased density shortening of sedrate via compaction
        totdepth=0
        for i in range(nbins):
            # calc relative bulk density for this interval
            bulkdchange=(bulkd0 + bulkdinc*totdepth)/bulkd0

            # calc depth down for this interval i as
            depthdown=step*sedrate[j]/bulkdchange

            # add depthdown to total cumulative depth, exit loop if maxdepth hole reached
            totdepth=totdepth+depthdown
            if totdepth>maxdepth:
                totdepth=maxdepth
                break

            # assign apparent (compacted) sedrate to resar bin
            resar[i,j,s]=sedrate[j]/bulkdchange
            maxages[j,s]=i*step


#-- add hiatuses to resar
print('Step 2 - adding hiatus')
for sim in range(nsim):
    # for each column 'col' in resar
    print(str(sim), end="\r")
    for col in range(ntries):
        # iterate thru column by 'bin'.
        for bin in range(nbins):

            if (np.isnan(resar[bin,col,sim])==True): # reached maxage of hole penetration, exit to next hole
                break
            if(ran.random()<hiprob): #random number probability threshold for hiatus exceeded

                # get hiatus length from hiatus length generator function.
                hilenar=weibull_min.rvs(hishape, scale=hiscale, size=1)
                if hilenar[0]>step/2:
                    hlen=round(hilenar.item(0)) # convert 1 element numpy array to scalar

                    if(bin+hlen>=nbins-1): # hiatus extends below base hole, just set bins to NaN
                        for j in range(bin,nbins):
                            resar[j,col,sim]=np.NaN
                        bin=nbins
                    else:
                        sourcecol=np.copy(resar[:,col,sim]) #create separate old hole source for copy
                        movlen=nbins-(bin+hlen)
                        for i in range(0,movlen):  # move column below hiatus down
                            resar[bin+hlen+i,col,sim]=sourcecol[bin+i]
                        for j in range(bin,bin+hlen): # fill gap with NaN
                            resar[j,col,sim]=np.NaN
                        bin=bin+hlen # move bin position below gap
                else:
                    break

avesedrate = np.nanmedian(resar,axis=1).round(3)
globalavesedrate = np.nanmedian(avesedrate,axis=1)

#-- For mean output instead of median: uncomment the following two lines
# avesedrate = np.nanmean(resar,axis=1).round(3)
# globalavesedrate = np.nanmean(avesedrate,axis=1)

#-- Save to files
data = {'avesedrate': avesedrate.tolist(), 'globalavesedrate': globalavesedrate.tolist()}
params={'holedepths': holedepths.tolist(), 'maxages': maxages.tolist(),
        'params': {'bulkd0':bulkd0,
                   'bulkdinc':bulkdinc,
                   'wshape':wshape,
                   'wscale':wscale,
                   'wloc':wloc,
                   'maxtime':maxtime,
                   'steps':step,
                   'nsim':nsim,
```

```
                    'ntries':ntries,
                    }}

print("Output result to a file")
filename = 'result_%s.json' % timestampstr
with open(filename,'wb') as f:
    f.write(orjson.dumps(data))

print("Output parameters used to a file")
filename2 = 'params_%s.json' % timestampstr
with open(filename2,'wb') as f:
    f.write(orjson.dumps(params))

if output>0:
    print("Output full data used to a file")
    filename3 = 'resar_%s.json' % timestampstr
    with open(filename3,'wb') as f:
        f.write(orjson.dumps({'resar':resar.tolist()}))
```