VAPOR TRANSPORT MODELING OF CONTINENTAL WATER ISOTOPE GRADIENTS

A FINAL PROJECT SUBMITTED TO THE
DEPARTMENT OF EARTH SYSTEM SCIENCE
AT STANFORD UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
THE DEGREE OF MASTER OF SCIENCE

Andrea Jean Ritch
June 1, 2016

Abstract

Stable isotopes have been widely used to reconstruct past climatic conditions and topographic histories of mountain belts. However, many studies do not account for the influences of evapotranspiration and vapor recycling on downstream meteoric water isotopic compositions. Here I present a case study of the modern Sierra Nevada and Basin and Range to illustrate the value of using process-based models across larger spatial scales to reconstruct the conditions driving local- to regional-scale water isotopic compositions.

We use a one-dimensional reactive vapor transport model, driven by the National Centers for Environmental Prediction (NCEP) high-resolution North American Regional Reanalysis (NARR) dataset, to simulate the isotopic composition of modern meteoric waters (δ^{18}O and δD) along storm tracks across the Sierra Nevada and Basin and Range. Storm track pathways are generated using NOAA's Air Resources Laboratory's Hybrid Single Particle Lagrangian Integrated Trajectory (HYSPLIT) model. In addition, we couple the vapor transport model with a soil moisture model to simulate depth profiles of the oxygen isotopic composition of authigenic carbonate along the storm tracks. We show that, given reasonable estimates of the modern partitioning between evaporation and transpiration, the model output not in disagreement with modern isotopic data both from compilations of published meteoric water samples and from newly collected soil carbonate samples along the northern Sierra Nevada and Basin and Range (~38-42° N). These results demonstrate that our modeling approach can be used to analyze the relative contributions of climate and topography to observed isotopic gradients. Future studies can apply this modeling framework to isotopes preserved in the geologic record to provide a quantitative means of understanding the paleoclimatic influences on spatial isotopic distributions.

Acknowledgements

I am very fortunate to have been part of a very supportive group of intelligent and inspiring individuals during my time at Stanford. Professor Page Chamberlain, my advisor, and Jeremy Caves, Daniel Ibarra, and Matthew Winnick, my labmates, were essential contributors to the work described herein. I am also grateful for the support of Yadira Ibarra, Heidi Hirsh, Ellen Ward, and Dan Moyer.

TABLE OF CONTENTS

Vapor transport modeling of continental water isotope gradients

1. Introduction

The hydrogen and oxygen stable isotopes (δD and δ^{18}O) of precipitation serve as conservative tracers of the origin, phase transitions, and transport pathways of moisture, and are thus useful in understanding controls on the hydrologic cycle. Therefore, these isotopes in water-derived carbonates formed in paleosols and in lacustrine sediments serve as tracers of the interactions between moisture source, transport pathway, topography, and the precipitation-recycling ratio (Amundson et al., 1996; Hoke et al., 2009; Quade et al., 2011; Hoke et al., 2013; Rohrmann et al., 2014). As such, they have become a useful tool for studying past climatic, environmental, and topographic conditions.

It has been well established that the presence of high topography results in increased windward-side precipitation; this is known as orographic precipitation. Topography thus is also associated with decreased values of δ^{18}O and δD due to the amount effect (Dansgaard, 1964). This relationship has been studied extensively and has been widely used in reconstructing past topographic histories (*e.g.* Garzione et al., 2000; Mulch et al., 2006; Mulch et al., 2008; Mix et al., 2011; Quade et al., 2011; Chamberlain et al., 2012; Fan et al., 2014). However, many studies of paleoaltimetry rely on empirical relationships (Poage and Chamberlain, 2001), resulting in limitations due to the lack of a process-based approach to reconstruction (Mulch, 2016). More recently, a number of studies have also suggested the need to incorporate moisture source and trajectory into paleotopographic reconstruction (Lechler and Galewsky, 2013; Mix et al., 2015; Mulch, 2016; Wheeler et al., 2016). Incorporating moisture source and trajectories allows for the consideration of upstream vapor recycling and its effects on the composition of stable isotopes (Winnick et al., 2014).

This study attempts to qualify the necessity of using a vapor transport model to understand stable isotopic records. By considering upstream environmental and climatic conditions across larger spatial gradients, we argue that stable isotopic records can be reproduced without invoking an empirical relationship with topographic height.

2. Background

Never before has understanding the hydrologic response to future climate change been more critical. Currently, atmospheric pCO$_2$ is above 400 ppm (Keeling et al., 2001) and is projected to increase over the coming century (Hansen and Sato, 2004; Jones et al., 2013; Tjiputra et al., 2014). However, our understanding of the hydrologic response to the increased $p\mathrm{CO}_2$ over terrestrial systems, as derived from climate models, is subject to considerable uncertainty (Allen and Ingram, 2002; Gosling et al., 2011; Haddeland et al., 2011; Hagemann et al., 2013). Both models and observations have suggested that, under a warming climate, there will be a general intensification of current hydrological conditions (*i.e.* wet regions become wetter and dry regions become drier) (Held and Soden, 2006; Huntington, 2006; Durack et al., 2012). Historical records (1948-2005) imply that this is an oversimplification over land: only 10.8% of global land area has fallen under the trend of intensification, while 9.5% has followed the opposite pattern (Greve et al., 2014). It is still argued, though, that dry land areas will become drier due to an even greater increase in potential evapotranspiration than any increase in precipitation that may be associated with warmer temperatures (Feng and Fu, 2013; Fu and Feng, 2014; Sherwood and Fu, 2014).

In spite of the controversy surrounding global, terrestrial hydrologic trends, model ensembles statistically demonstrate that, in southwestern North America (SWNA), there will be increasingly arid conditions associated with warming global temperatures, though there is uncertainty as to the degree of these changes (Seager et al., 2007; Seager and Vecchi, 2010; Seager et al., 2012; Cook et al., 2014; Lau and Kim, 2015). These predictions result from (*1*) an increase in potential evapotranspiration due to increase saturation vapor pressure with higher temperatures (Held and Soden, 2006; Seager et al., 2010; Feng and Fu, 2013; Fu and Feng, 2014), and (*2*) a decrease in precipitation as a consequence of a poleward shift of the downwelling branch of the Hadley Circulation (Yin, 2005; Miller et al., 2006; Previdi and Liepert, 2007; Seager et al., 2010; Scheff and Frierson, 2012). Furthermore, in addition to projected background drying trends in SWNA, several studies have shown increased drought risk associated with warming temperatures through the 21st century (Woodhouse et al., 2010; Ault et al., 2014; Cook et al., 2015; Diffenbaugh et al., 2015; Lau and Kim, 2015).

In contrast, however, paleoclimate records suggest that, under high-temperature, high-$p\mathrm{CO}_2$ conditions, such as those projected within the next century, SWNA should become wetter, not drier. For example, during the Pliocene epoch (5.33-2.58 Ma), the interglacial climate system was characterized by near-modern atmospheric CO_2 concentrations (Raymo et al., 1996; Pagani et al., 2010; Bartoli et al., 2011), and average global temperatures were about 3 °C above modern

(Haywood et al., 2000; Haywood and Valdes, 2004; Dowsett et al., 2009; Lunt et al., 2012; Fedorov et al., 2013; Haywood et al., 2013). As such, the Pliocene interglacial periods represent an analogue of how SWNA hydrology may change in response to warm, high $p\mathrm{CO}_2$ conditions (Salzmann et al., 2011). In contrast to modern predictions, Pliocene SWNA experienced wetter-than-modern conditions (Thompson, 1991; Salzmann et al., 2008; Haywood et al., 2013; Winnick et al., 2013) that allowed for the sustained presence of multiple lake systems, which are currently absent, throughout the region (Reheis et al., 2002; Knott et al., 2008; Phillips, 2008; Pound et al., 2014). This highlights the need for an increased understanding of past environments and the role of the hydrologic cycle as we attempt to predict the future impacts of climate change.

This study argues that a vapor transport model can be used to understand the varying contributions of precipitation, evaporation, and transpiration to downstream isotopic records. This approach should help researchers understand the hydrologic features and mechanisms of past climate states and it will provide insight as to how the hydrologic cycle might respond to current and future climate change.

3. Methods

This project utilizes both a vapor transport model and a soil moisture model to predict the isotopic values of precipitation along a storm path and to generate hypothetical depth profiles of soil carbonate isotopic values along that transect. Storm pathways are generated using an atmospheric back-trajectory model.

3.1 Vapor Transport Model

Recent work by our group has been focused on how isotopic compositions are affected by moisture transport pathways (advective vs. eddy-diffusive) and vapor recycling using a one-dimensional vapor transport model along a given storm track (Mix et al., 2013; Chamberlain et al., 2014; Winnick et al., 2014; Caves et al., 2015; Winnick et al., 2015). This isotope-enabled model, based on the work of Hendricks et. al. (2000), tracks the isotopic composition of atmospheric water vapor and precipitation along a storm path. Atmospheric moisture balance is defined by

$$\frac{dw}{dt} = \nabla[K\nabla w] - v\nabla w + ET - P \qquad (1)$$

where w is specific humidity (precipitable water), K is the eddy diffusivity of water vapor, v is the advection velocity, ET is evapotranspiration, and P is precipitation (Hendricks et al., 2000; Winnick et al., 2014). By considering equation (*1*) for two isotopologues of water and by assuming isotopic steady state, it can be shown that the analytical solutions to this model have two endmembers for the scenarios of purely advective transport and purely eddy-diffusive transport, which are given respectively by:

$$\delta_{a(adv)} = (\delta_a^0 - \delta_a^\infty) exp\left[(\alpha + \alpha N_d - 1)\left(\frac{x}{w}\frac{dw}{dx}\right)\right] + \delta_a^\infty \quad (2)$$

$$\delta_{a(eddy)} = (\delta_a^0 - \delta_a^\infty) exp\left[\left(\sqrt{\alpha + \alpha N_d} - 1\right)\left(\frac{x}{w}\frac{dw}{dx}\right)\right] + \delta_a^\infty \quad (3)$$

where δ_a is the isotopic composition of atmospheric water vapor, α is the equilibrium fractionation factor of condensation, x is the distance along the storm track, N_d is the Damköhler number, δ_a^∞ and δ_a^0 are the isotopic compositions of atmospheric water vapor as x approaches ∞ and at x = 0, respectively, and

$$\delta_a^\infty = \frac{N_d \delta_{ET} - (1+N_d)(\alpha-1)10^3}{\alpha + \alpha N_d - 1} \quad (4)$$

where δ_{ET} is the isotopic composition of evapotranspiration (ET) (Hendricks et al., 2000; Winnick et al., 2014). The Damköhler number, N_d, is given by $N_d = -\frac{ET}{v^{dw}/_{dx}}$ and, in the advection-only case, reduces to $N_d = \frac{ET}{P-ET}$ and relates the amount of water entering the vapor to the amount exiting (via runoff). Higher Damköhler numbers emerge for areas with high rates of evapotranspiration relative to total precipitation (when ET approaches P). These areas are generally much more arid and are characterized by fairly large contributions of vapor recycling to total precipitation, such as in the southwestern United States, the Middle East, the Saharan Desert, and Central Asia (Winnick et al., 2014).

This vapor transport model also requires the partitioning of the evaporation and transpiration components of total evapotranspiration rates. As this partitioning is not well constrained, particularly across regional gradients, we run the model using a suite of evaporation/transpiration ratios.

We run this isotopically-enabled vapor transport model as a forward model, driven by the NARR dataset, to predict the isotopic composition of water along a given storm track. Both mean trajectories and individual trajectories are calculated; climatological inputs are described in Section 4.1.

3.2 Storm Path Generation

To generate storm track pathways, we use the Hybrid Single Particle Lagrangian Integrated Trajectory (HYSPLIT) model from NOAA's Air Resources Laboratory (Draxler and Hess, 1998; Stein et al., 2015). We run HYSPLIT using the National Centers for Environmental Prediction (NCEP) high-resolution North American Regional Reanalysis (NARR) dataset (Mesinger et al., 2004). Use of HYSPLIT has become increasingly common in studies of moisture source, particularly those which also incorporate the use of stable isotopes (Sjostrom and Welker, 2009; Bershaw et al., 2012; Oster et al., 2012; Lechler and Galewsky, 2013; Caves et al., 2014; Caves et al., 2015).

HYSPLIT model runs are seven-day back-trajectories initiated every six hours for each month during the years 1979-2014 at an elevation of 1000 m above ground level; 50,240 trajectories are thus calculated for each site. Storm paths are defined as the trajectories that generate precipitation within six hours of their endpoint. These trajectories are then combined into two-dimensional histogram maps which contour the regions of trajectory occurrence (Lechler and Galewsky, 2013; Caves et al., 2014; Caves et al., 2015).

3.3 Soil Moisture Model

The soil moisture model used in this project is a diffusion model in which evaporation at the soil surface causes isotopic fractionation and the upward diffusion of deeper moisture (Zimmermann et al., 1967; Chamberlain et al., 2014). Thus, at any given depth, *z*, in a soil profile, the isotopic composition of water at that depth ($\delta_{w,z}$) is:

$$\delta_{w,z} = (\delta_{w,0} - \delta_p) exp(-{}^{z}/_{z^*}) + \delta_p \qquad (5)$$

where δ_p is the isotopic composition of precipitation and

$$z^* = \frac{\rho\tau d}{E} \qquad (6)$$

where ρ is the free-air soil porosity, τ is the soil tortuosity, *d* is the self-diffusion coefficient of soil water, and *E* is the evaporation rate. Additionally, we define the isotopic composition of water at the surface (*z* = 0) to be:

$$\delta_{w,0} = \alpha_{eq}\left((1-h)\alpha_{kin}\delta_p + h\delta_{atm}\right) \qquad (7)$$

in which α_{eq} and α_{kin} are the equilibrium and kinetic fractionation factors, *h* is the relative humidity, and δ_{atm} is the isotopic composition of atmospheric water vapor, assumed to be in equilibrium with the precipitation.

This model is used with NARR data to calculate hypothetical water isotopic depth profiles in soils. These water isotope values are converted to carbonate isotopic profiles assuming carbonate species equilibrium and summer-autumn temperatures. Climatological inputs are described below in Section 4.1.

4. Data

Three varieties of data are used in this project: reanalysis data, modern meteoric water isotopic data, and modern soil carbonate data.

4.1 Reanalysis Data

Climatological data used in this project comes from the National Centers for Environmental Prediction (NCEP) high-resolution North American Regional Reanalysis (NARR) dataset (Mesinger et al., 2004). Though this reanalysis dataset has some notable imperfections (including instances of negative precipitation rates), we choose it for its high resolution (0.3°, ~32km) relative to other datasets. All NARR data is available as NetCDF files from http://www.esrl.noaa.gov/psd/data/gridded/data.narr.html.

For the vapor transport model, we use the following inputs:

-2 m air temperature (air.2m)

-precipitable water (pr_wtr)

-2 m relative humidity (rhum.2m)

-land mask data (land)

-Damköhler number, N_d, (evap/(prate-evap)

To calculate the Damköhler number, defined as the ratio of evapotranspiration to runoff, many possible calculation permutations were considered using various evaporation, precipitation, runoff, and latent heat measurements available in the NARR dataset. The 'evap' and 'prate' variables were chosen for use due to their improved stability and range over the other options. Unfortunately, ~75% of the N_d values are calculated to be negative, which is physically impossible, but these result from occurrences of negative precipitation and evaporation rates in the dataset. As the model does not function properly for $N_d < 0$, the use of longer-term mean values is preferred. Thus, for calculating mean trends along a mean storm track, we use the long-term (1979-2014) monthly-mean data from NARR, though HYSPLIT analyses and individual storm tracks utilize 8-times daily values.

For the soil moisture model, we also use the following long-term (1979-2014) mean data from NARR:

-mean annual precipitation (apcp)

-surface temperature (JJASO; air.sfc)

-soil temperature (JJASO; depths 0-800 cm; tsoil)

For surface and soil temperatures, summer-autumn (JJASO) values are used because it is during these months that the NARR dataset demonstrated minimized surface soil moisture and maximized surface temperature, conditions considered more conducive to seasonal carbonate formation (Breecker et al., 2009; Passey et al., 2010; Peters et al., 2013; Quade et al., 2013). Soil temperatures are provided by NARR at select depths; we use a linear interpolation for the in-between values.

4.2 Modern Water Isotope Data

Modern water isotopic data (δD and δ^{18}O) compilation was inspired by Ingraham and Taylor (Ingraham and Taylor, 1991), who collected transects of meteoric isotopic data moving inward from the coast during the winter months (when most precipitation is delivered). We primarily focus on their Traverse II, located at ~39 °N, though a far more extensive modern water dataset was compiled (see Appendix A).

4.3 Modern Soil Isotope Data

Soil samples were collected along a transect meant to represent the mean storm pathways across the Sierra Nevada and Basin and Range. Modern soil samples were collected and analyzed in the summer of 2014 by Jeremy Caves, Daniel Ibarra, and Sam Kramer. The samples were ground and analyzed at Stanford's Stable Isotope Biogeochemistry Laboratory using a Thermo Finnigan Delta Plus XL mass spectrometer. These data, along with other modern soil isotopic values, can be found in the data repository (see Appendix A). Additional samples collected in 2015 have not yet been analyzed.

5. Results and Discussion

Here we discuss the results of HYSPLIT storm track generation along with the results of both the vapor transport model and the soil moisture model. For the vapor transport and soil moisture models, we compare model results to compiled data.

5.1 HYSPLIT Storm Tracks

HYSPLIT results from one of our soil sample sites, Hay, are shown below in Figure 1. This figure demonstrates that, on an annual scale, moisture is sourced both from the west and from the south. In the winter (DJF) and spring (MAM) months, when the majority of precipitating trajectories occur, moisture seems predominantly westerly-sourced. Thus, westerly-sourced winter precipitation trajectories may be sufficient for driving our vapor transport model, though an improved approach would be to integrate the seasonality of westerly and southerly sourced moisture.

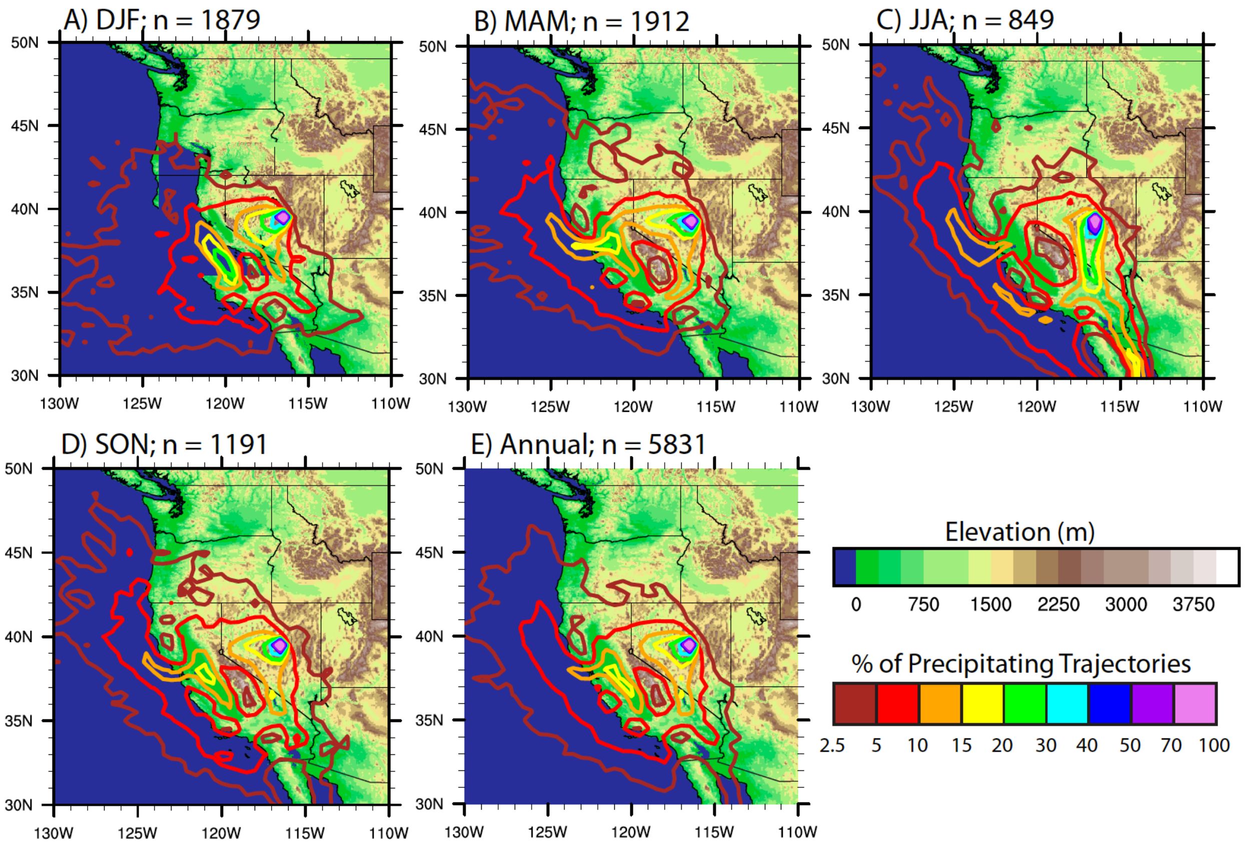

Figure 1: HYSPLIT seven-day back-trajectory results for our Hay soil sample (39.57°N; 116.20°W) site, overlain on topography as a contoured 2D histogram of trajectory occurrence. All trajectories shown are those which produce precipitation; non-precipitating trajectories are not included here. Values of n indicate the number of precipitating trajectories shown for each season. (A) December, January, February (DJF) trajectories; (B) March, April, May (MAM) trajectories; (C) June, July, August (JJA) trajectories; (D) September, October, November (SON) trajectories; (E) all trajectories.

5.2 Vapor Transport Model

Results from the vapor transport model along mean trajectories are plotted below in Figure 2 along with the isotopic values from Ingraham and Taylor (1991). We plot model runs for both advection-only and eddy diffusion-only cases for three different partitionings of evapotranspiration. In Figure 2, it is worth noting that the plotted groundwater data have consistently higher values of $\delta^{18}O$ relative to the precipitation samples, likely a result of evaporative enrichment. Additionally, the model does not seem to generate an initial decrease in isotopic values closest to the coast. The observed decrease in $\delta^{18}O$ is likely a result of increased orographic rainout along the California Coast Ranges that is not recorded by the relatively low-resolution precipitation rates of the NARR dataset. However, the model output $\delta^{18}O$ values do seem to level off at about 300 km from the coast into the Basin and Range. This is a result of higher Damköhler numbers inherent to the endorheic Basin and Range, in which the majority of precipitation is a result of vapor recycling processes (Winnick et al.,

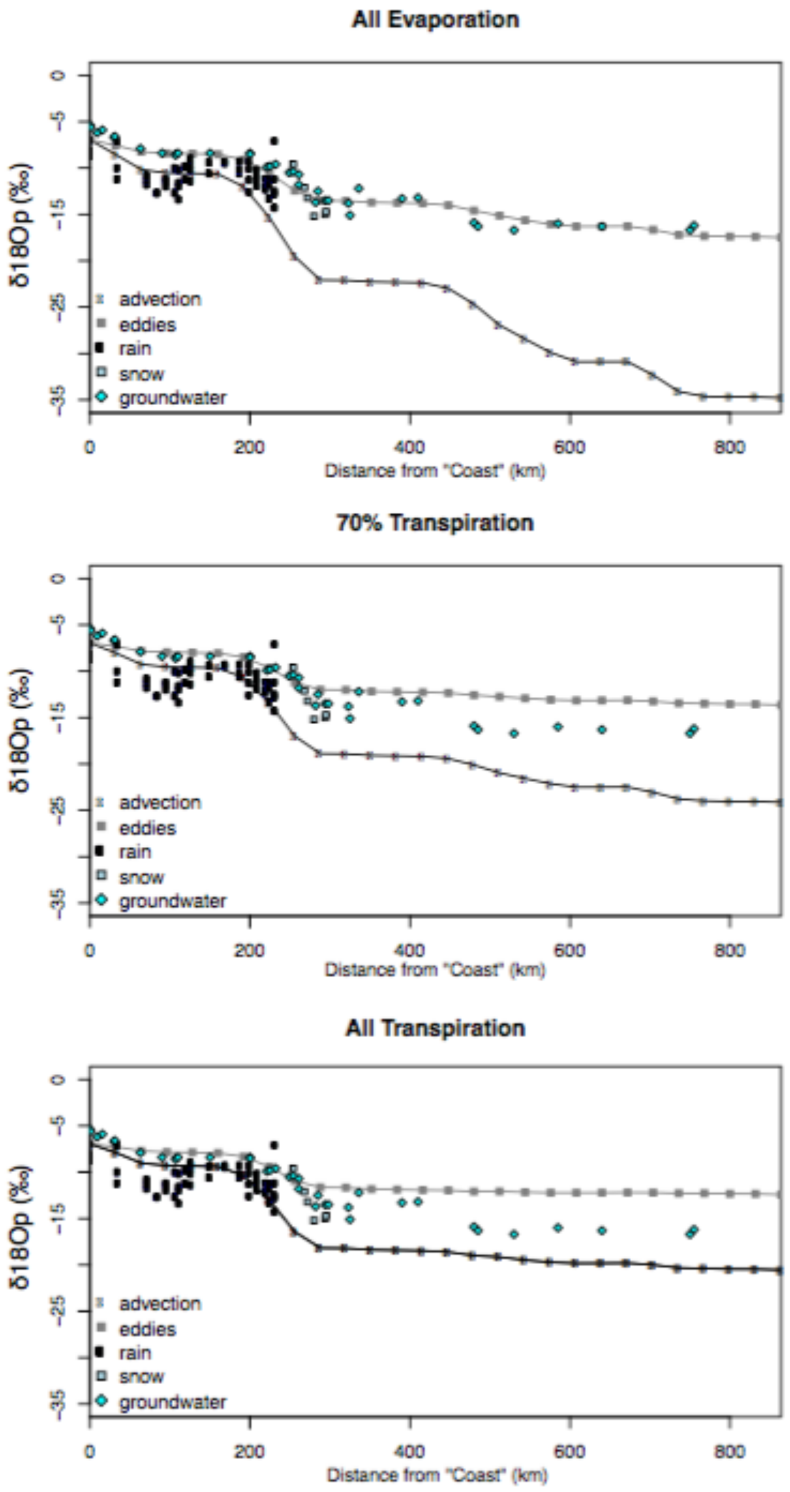

Figure 2: Vapor transport model results across the Sierra Nevada and Basin and Range for both eddies-only and advection-only cases using three values of evapotranspiration partitioning. Modern precipitation, snow, and groundwater $\delta^{18}O$ values are from Ingraham and Taylor (1991).

2014).

We can also see in Figure 2 that neither the advection-only nor eddies-only case is fully appropriate to characterize the observed δ^{18}O data (particularly when we consider transpiration contributions to vapor recycling); both likely play a role in determining downstream isotopic compositions. Regardless, in spite of the limitations of the observed dataset, we see that the model is able to reproduce changes in isotopic composition across the Sierra Nevada without invoking empirical relationships due to elevation, thus demonstrating that upstream environmental and climatic variables are important in considering downstream isotopic records.

Our model is also able to consider deuterium excess. Deuterium excess, often labeled as d-excess or *d*, is defined as the intercept for the linear relationship between δD and δ^{18}O in meteoric waters (Dansgaard, 1964):

$$d = \delta D - 8\times\delta^{18}O \qquad (8)$$

While the slope of this relationship is governed by mass-dependent equilibrium fractionation, the d-excess values are controlled by the non-equilibrium conditions during kinetic fractionation between condensed and vapor phases. Particularly, d-excess values can provide insight to evaporative conditions such as temperature and relative humidity (Merlivat and Jouzel, 1979; Jouzel and Merlivat, 1984; Johnsen et al., 1989).

More recently, some studies have been focusing on ^{17}O-excess as a tracer of evaporative conditions in the hydrologic cycle (Angert et al., 2004; Luz and Barkan, 2010; Risi et al., 2010). ^{17}O-excess, like d-excess, describes the kinetic fractionation affecting the relationship between δ^{17}O and δ^{18}O, in which low values of ^{17}O-excess are associated with highly evaporative conditions (Luz and Barkan, 2010):

$$^{17}O - excess = \ln(\delta^{17}O + 1) - 0.528\times\ln(\delta^{18}O + 1) \qquad (9)$$

^{17}O-excess is a powerful tool because, unlike d-excess, it is insensitive to changing temperatures, indicating that it is a direct measure of changing relative humidity (Angert et al., 2004; Risi et al., 2010). Thus, when used in combination, d-excess and ^{17}O-excess should be able to distinguish between the kinetic effects of temperature and relative humidity on isotopic compositions to better constrain evaporative conditions.

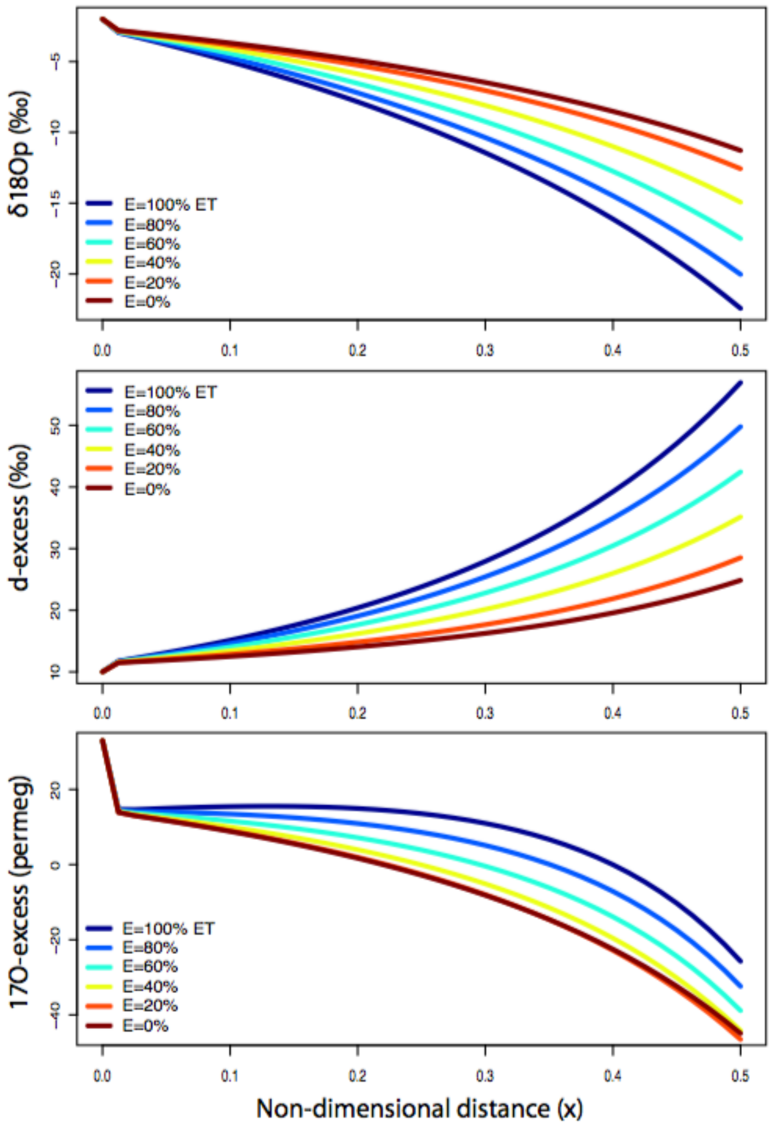

Figure 3: Model-predicted values of δ^{18}O, d-excess, and ^{17}O-excess of precipitation along a hypothetical storm track.

Though limited data are available for modern values of δD and δ^{17}O with which to calculate modern d-excess and ^{17}O-excess, we are still able to model their expected behaviors using our vapor transport model, as shown in Figure 3. From this, we see that, as a storm rains out as it moves inward across a continent, it progressively loses moisture, and the δ^{18}O values progressively decrease. Concurrently, the d-excess values increase and the ^{17}O-excess values decrease as a result of kinetic fractionation during vapor recycling processes.

5.3 Soil Moisture Model

By combining the vapor transport model with the soil moisture model, we are able to generate hypothetical soil carbonate δ^{18}O profiles along a given storm track, as shown in Figure 4. In Figure 4, we plot the δ^{18}O values of soil carbonate by depth, for six distances from the coast (at the approximate inward distances of our modern soil carbonate observations). A key observation from this figure is that, at increased distance along a storm track, the δ^{18}O values of carbonate at depth, which should closely record the δ^{18}O values of precipitation, decrease, as is consistent with the expected behavior of δ^{18}O of precipitation. This is particularly notable in the case of increased evaporation relative to transpiration, again consistent with the understanding that transpiration does not cause isotopic fractionation.

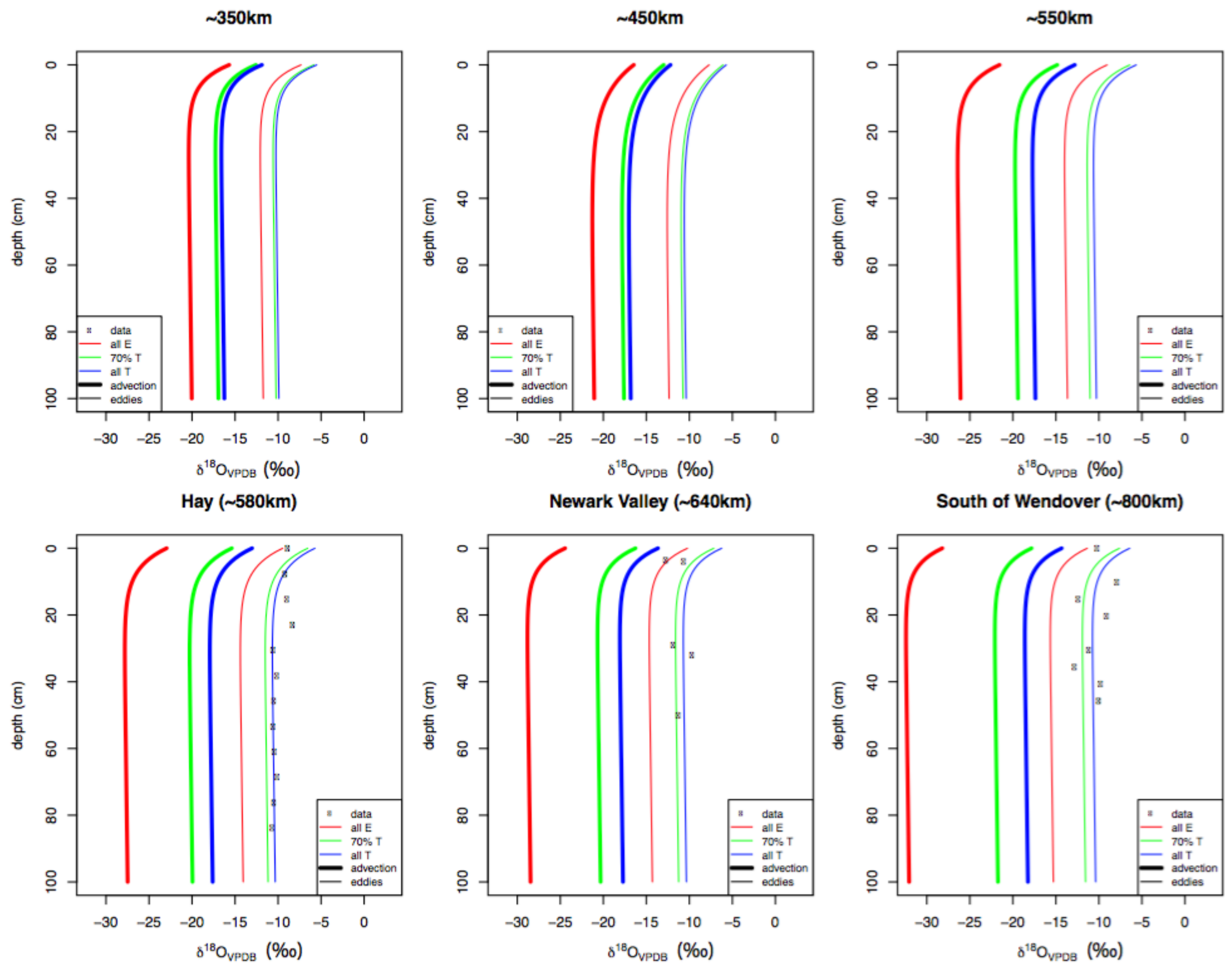

Figure 4: Model-output soil carbonate δ^{18}O profiles at six points along the vapor transport pathway for three evapotranspiration partitionings for both the advection-only and eddies-only cases. The Hay, Newark Valley, and South of Wendover sites also include modern soil carbonate δ^{18}O data.

Also shown in Figure 4 are modern soil carbonate δ^{18}O data. These data suggest that the conditions for moisture pathway included mostly eddy-diffusive processes and more transpiration relative to evaporation, but more data and continued analyses will be required to determine if this is an accurate assessment. Furthermore, because not all of the precipitation reaching these sites is sourced from westerly moisture (see Figure 1), an analysis that integrates other moisture trajectories would likely yield improved results.

5.4 Vapor Transport along Individual HYSPLIT Trajectories

A large portion of this project was devoted to integrating individual HYSPLIT output trajectories with the appropriate NARR data to run the vapor transport model along individual storm paths instead of over mean trajectories. Unfortunately, due to imperfections in the NARR dataset (including negative precipitation rates) and the difficulty in constraining initial isotopic compositions and evapotranspiration partitioning, further work will be needed to produce usable results. A more thorough discussion of the model/HYSPLIT integration and the associated complications is provided in Appendix C.

6. Conclusion

Our results demonstrate that we are able to use a vapor transport model to predict the oxygen isotopic composition of precipitation and soil carbonate, assuming reasonable estimates of evapotranspiration partitioning and initial isotopic compositions. Limited data exist to statistically validate our model, though the data that are available suggest that, at least for the atmospheric component, our model can successfully reproduce observations across spatial gradients without invoking an empirically based relationship between isotopic value and topographic height. This demonstrates that future work on isotopic records, both modern and in the geologic record, should make use of a model such as ours that considers the effects of upstream hydrologic processes on downstream isotopic compositions. The effects of evapotranspiration and vapor recycling across a continent have quantifiable impacts on stable isotope values, and these should not be ignored in future studies.

Future work on this project will require better constraints on evapotranspiration partitioning and the initial composition of the stable isotopes of meteoric water. In addition, much more data will need to be collected, both for atmospheric water and soil carbonate, to allow for statistical testing of our model. Once fully validated, our model can be used for any set of isotopic records that span a geographical gradient to better understand the contributions of precipitation, evapotranspiration, and vapor recycling to observed isotopic distributions.

7. References

Allen, M.R., and Ingram, W.J., 2002, Constraints on future changes in climate and the hydrologic cycle.: Nature, v. 419, no. 6903, p. 224–32, doi: 10.1038/nature01092.

Amundson, R., Chadwick, O., Kendall, C., and Wang, Y., 1996, Isotopic evidence for shifts in atmospheric circulation patterns during the late Quaternary in mid-North America: Geology, , no. 1969, p. 23–26, doi: 10.1130/0091-7613(1996).

Angert, A., Cappa, C.D., and DePaolo, D.J., 2004, Kinetic 17O effects in the hydrologic cycle: Indirect evidence and implications: Geochimica et Cosmochimica Acta, v. 68, no. 17, p. 3487–3495, doi: 10.1016/j.gca.2004.02.010.

Ault, T.R., Cole, J.E., Overpeck, J.T., Pederson, G.T., and Meko, D.M., 2014, Assessing the risk of persistent drought using climate model simulations and paleoclimate data: Journal of Climate, v. 27, no. 20, p. 7529–7549, doi: 10.1175/JCLI-D-12-00282.1.

Bartoli, G., Honisch, B., and Zeebe, R.E., 2011, Atmospheric CO2 decline during the Pliocene intensification of Northern Hemisphere glaciations: Paleoceanography, v. 26, no. 4, p. 1–14, doi: 10.1029/2010PA002055.

Bershaw, J., Penny, S.M., and Garzione, C.N., 2012, Stable isotopes of modern water across the Himalaya and eastern Tibetan Plateau: Implications for estimates of paleoelevation and paleoclimate: Journal of Geophysical Research Atmospheres, v. 117, no. 2, p. 1–18, doi: 10.1029/2011JD016132.

Breecker, D.O., Sharp, Z.D., and McFadden, L.D., 2009, Seasonal bias in the formation and stable isotopic composition of pedogenic carbonate in modern soils from central New Mexico, USA: Bulletin of the Geological Society of America, v. 121, no. 3-4, p. 630–640, doi: 10.1130/B26413.1.

Caves, J.K., Sjostrom, D.J., Mix, H.T., Winnick, M.J., and Chamberlain, C.P., 2014, Aridification of Central Asia and uplift of the Altai and Hangay mountains, Mongolia: Stable isotope evidence: American Journal of Science, v. 314, no. 8, p. 1171–1201, doi: 10.2475/08.2014.01.

Caves, J.K., Winnick, M.J., Graham, S.A., Sjostrom, D.J., Mulch, A., and Chamberlain, C.P., 2015, Role of the westerlies in Central Asia climate over the Cenozoic: Earth and Planetary Science Letters, v. 428, p. 33–43, doi: 10.1016/j.epsl.2015.07.023.

Chamberlain, C.P., Mix, H.T., Mulch, A., Hren, M.T., Kent-Corson, M.L., Davis, S.J., Horton, T.W., and Graham, S.A., 2012, The Cenozoic climatic and topographic evolution of the Western North American Cordillera: American Journal of Science, v. 312, p. 213–262.

Chamberlain, C.P., Winnick, M.J., Mix, H.T., Chamberlain, S.D., and Maher, K., 2014, The impact of neogene grassland expansion and aridification on the isotopic composition of continental precipitation: Global Biogeochemical Cycles, v. 28, no. 9, p. 992–1004, doi: 10.1002/2014GB004822.

Cook, B.I., Ault, T.R., and Smerdon, J.E., 2015, Unprecedented 21st century drought risk in the American Southwest and Central Plains: Science Advances, v. 1, no. February, p. 1–7, doi: 10.1126/sciadv.1400082.

Cook, B.I., Smerdon, J.E., Seager, R., and Coats, S., 2014, Global warming and 21st century drying: Climate Dynamics, p. 1–21, doi: 10.1007/s00382-014-2075-y.

Dansgaard, W., 1964, Stable isotopes in precipitation: Tellus, v. 16, no. 4, p. 436–468, doi: 10.3402/tellusa.v16i4.8993.

Diffenbaugh, N.S., Swain, D.L., and Touma, D., 2015, Anthropogenic warming has increased drought risk in California: Proceedings of the National Academy of Sciences, v. 112, no. 13, p. 3931–3936, doi: 10.1073/pnas.1422385112.

Dowsett, H.J., Robinson, M.M., and Foley, K.M., 2009, Pliocene three-dimensional global ocean temperature reconstruction: Climate of the Past, v. 5, p. 769–783, doi: 10.5194/cpd-5-1901-2009.

Draxler, R.R., and Hess, G.D., 1998, An overview of the HYSPLIT_4 modelling system for trajectories, dispersion and deposition: Australian Meteorological Magazine, v. 47, p. 295–308.

Durack, P.J., Wijffels, S.E., and Matear, R.J., 2012, During 1950 to 2000: Science, v. 336, no. April, p. 455–458, doi: 10.1126/science.1212222.

Fan, M., Heller, P., Allen, S.D., and Hough, B.G., 2014, Middle Cenozoic uplift and concomitant drying in the central Rocky Mountains and adjacent Great Plains: Geology, v. 42, no. 6, p. 547–550, doi: 10.1130/G35444.1.

Fedorov, A. V., Brierley, C.M., Lawrence, K.T., Liu, Z., Dekens, P.S., and Ravelo, A.C., 2013, Patterns and mechanisms of early Pliocene warmth.: Nature, v. 496, no. 7443, p. 43–9, doi: 10.1038/nature12003.

Feng, S., and Fu, Q., 2013, Expansion of global drylands under a warming climate: Atmospheric

Chemistry and Physics, v. 13, no. 19, p. 10081–10094, doi: 10.5194/acp-13-10081-2013.

Fu, Q., and Feng, S., 2014, Journal of Geophysical Research : Atmospheres: , p. 1–13, doi: 10.1002/2014JD021608.

Garzione, C.N., Quade, J., DeCelles, P.G., and English, N.B., 2000, Predicting paleoelevation of Tibet and the Himalaya from δ18O vs. altitude gradients in meteoric water across the Nepal Himalaya: Earth and Planetary Science Letters, v. 183, p. 215–229.

Gosling, S.N., Taylor, R.G., Arnell, N.W., and Todd, M.C., 2011, A comparative analysis of projected impacts of climate change on river runoff from global and catchment-scale hydrological models: Hydrology and Earth System Sciences, v. 15, no. 1, p. 279–294, doi: 10.5194/hess-15-279-2011.

Greve, P., Orlowsky, B., Mueller, B., Sheffield, J., Reichstein, M., and Seneviratne, S.I., 2014, Global assessment of trends in wetting and drying over land: Nature Geoscience, v. 7, no. 10, p. 716–721, doi: 10.1038/ngeo2247.

Haddeland, I., Clark, D.B., Franssen, W., Ludwig, F., Voß, F., Arnell, N.W., Bertrand, N., Best, M., Folwell, S., Gerten, D., Gomes, S., Gosling, S.N., Hagemann, S., Hanasaki, N., et al., 2011, Multimodel estimate of the global terrestrial water balance: setup and first results: Journal of Hydrometeorology, v. 12, p. 869–884, doi: 10.1175/2011JHM1324.1.

Hagemann, S., Chen, C., Clark, D.B., Folwell, S., Gosling, S.N., Haddeland, I., Hanasaki, N., Heinke, J., Ludwig, F., Voss, F., and Wiltshire, A.J., 2013, Climate change impact on available water resources obtained using multiple global climate and hydrology models: Earth System Dynamics, v. 4, no. 1, p. 129–144, doi: 10.5194/esd-4-129-2013.

Hansen, J., and Sato, M., 2004, Greenhouse gas growth rates.: Proceedings of the National Academy of Sciences of the United States of America, v. 101, no. 46, p. 16109–16114, doi: 10.1073/pnas.0406982101.

Haywood, A.M., Hill, D.J., Dolan, A.M., Otto-Bliesner, B.L., Bragg, F., Chan, W.L., Chandler, M.A., Contoux, C., Dowsett, H.J., Jost, A., Kamae, Y., Lohmann, G., Lunt, D.J., Abe-Ouchi, A., et al., 2013, Large-scale features of Pliocene climate: Results from the Pliocene Model Intercomparison Project: Climate of the Past, v. 9, no. 1, p. 191–209, doi: 10.5194/cp-9-191-2013.

Haywood, A.M., and Valdes, P.J., 2004, Modelling Pliocene warmth: Contribution of atmosphere, oceans and cryosphere: Earth and Planetary Science Letters, v. 218, no. 3-4, p.

363–377, doi: 10.1016/S0012-821X(03)00685-X.

Haywood, A.M., Valdes, P.J., and Sellwood, B.W., 2000, Global scale palaeoclimate reconstruction of the middle Pliocene climate using the UKMO GCM: Initial results: Global and Planetary Change, v. 25, no. 3-4, p. 239–256, doi: 10.1016/S0921-8181(00)00028-X.

Held, I.M., and Soden, B.J., 2006, Robust Responses of the Hydrological Cycle to Global Warming: Journal of Climate, v. 19, p. 5686–5699, doi: 10.1175/JCLI3990.1.

Hendricks, M.B., DePaolo, D.J., and Cohen, R.C., 2000, Space and time variation of d18O and dD in precipitation: Can paleotemperature be estimated from ice cores? v. 14, no. 3, p. 1–11, doi: 10.1029/1999GB001198.

Hoke, G.D., Aranibar, J.N., Viale, M., Araneo, D.C., and Llano, C., 2013, Seasonal moisture sources and the isotopic composition of precipitation, rivers, and carbonates across the Andes at 32.5-35.5°S: Geochemistry, Geophysics, Geosystems, v. 14, no. 4, p. 962–978, doi: 10.1002/ggge.20045.

Hoke, G.D., Garzione, C.N., Araneo, D.C., Latorre, C., Strecker, M.R., and Williams, K.J., 2009, The stable isotope altimeter: Do Quaternary pedogenic carbonates predict modern elevations? Geology, v. 37, no. 11, p. 1015–1018, doi: 10.1130/G30308A.1.

Huntington, T.G., 2006, Evidence for intensification of the global water cycle: Review and synthesis: Journal of Hydrology, v. 319, no. 1-4, p. 83–95, doi: 10.1016/j.jhydrol.2005.07.003.

Ingraham, N., and Taylor, B., 1991, Light Stable Isotope Systematics of Large-Scale Hydrologic Refimes in California and Nevada: Water Resources Research, v. 27, no. 1, p. 77–90.

Johnsen, S.J., Dansgaard, W., and White, J.W.C., 1989, The origin of Arctic precipitation under present and glacial conditions: Tellus B, v. 41B, no. 4, p. 452–468, doi: 10.1111/j.1600-0889.1989.tb00321.x.

Jones, C., Robertson, E., Arora, V., Friedlingstein, P., Shevliakova, E., Bopp, L., Brovkin, V., Hajima, T., Kato, E., Kawamiya, M., Liddicoat, S., Lindsay, K., Reick, C.H., Roelandt, C., et al., 2013, Twenty-first-century compatible co2 emissions and airborne fraction simulated by cmip5 earth system models under four representative concentration pathways: Journal of Climate, v. 26, no. 13, p. 4398–4413, doi: 10.1175/JCLI-D-12-00554.1.

Jouzel, J., and Merlivat, L., 1984, Deuterium and Oxygen 18 in Precipitation' Modeling of the Isotopic Effects During Snow Formation: Journal of Geophysical Research, v. 89, p.

11749–11757.

Keeling, C.D., Piper, S.C., Bacastow, R.B., Whorf, T.P., Heimann, M., and Meijer, H.A., 2001, Exchanges of atmospheric CO2 and 13CO2 with the terrestrial biosphere and oceans from 1978 to 2000: Global Aspects: SIO Reverence Series, v. 01, no. 06, p. 1–28.

Knott, J.R., Klinger, R.E., Liddicoat, J.C., and David, B.T., 2008, lakes and river systems as a test of pupfish (Cyprinodontidae) dispersal hypotheses: v. 2439, no. 01, p. 1–26, doi: 10.1130/2008.2439(01).

Lau, W.K.-M., and Kim, K.-M., 2015, Robust Hadley Circulation changes and increasing global dryness due to CO2 warming from CMIP5 model projections.: Proceedings of the National Academy of Sciences of the United States of America, v. 112, no. 12, p. 1–6, doi: 10.1073/pnas.1418682112.

Lechler, A.R., and Galewsky, J., 2013, Refining paleoaltimetry reconstructions of the Sierra Nevada: California, using air parcel trajectories: Geology, v. 41, no. 2, p. 259–262, doi: 10.1130/G33553.1.

Lunt, D.J., Haywood, A.M., Schmidt, G.A., Salzmann, U., Valdes, P.J., Dowsett, H.J., and Loptson, C.A., 2012, On the causes of mid-Pliocene warmth and polar amplification: Earth and Planetary Science Letters, v. 321-322, p. 128–138, doi: 10.1016/j.epsl.2011.12.042.

Luz, B., and Barkan, E., 2010, Variations of 17O/16O and 18O/16O in meteoric waters: Geochimica et Cosmochimica Acta, v. 74, no. 22, p. 6276–6286, doi: 10.1016/j.gca.2010.08.016.

Merlivat, L., and Jouzel, J., 1979, Global climatic interpretation of the deuterium-oxygen 18 relationship for precipitation: Journal of Geophysical Research, v. 84, no. C8, p. 5029, doi: 10.1029/JC084iC08p05029.

Mesinger, F., DiMego, G., Kalnay, E., Mitchell, K., Shafran, P.C., Ebisuzaki, W., Jović, D., Woollen, J., Rogers, E., Berbery, E.H., Ek, M.B., Fan, Y., Grumbine, R., Higgins, W., et al., 2004, North American regional reanalysis: Bulletin of the American Meteorological Society, v. 87, no. 3, p. 343–360, doi: 10.1175/BAMS-87-3-343.

Miller, R.L., Schmidt, G.A., and Shindell, D.T., 2006, Forced annular variations in the 20th century Intergovernmental Panel on Climate Change Fourth Assessment Report models: Journal of Geophysical Research Atmospheres, v. 111, no. 18, p. 1–17, doi: 10.1029/2005JD006323.

Mix, H.T., Ibarra, D.E., Mulch, A., Graham, S.A., and Chamberlain, C.P., 2015, A hot and high Eocene Sierra Nevada: Geological Society of America Bulletin, v. 128, no. 3-4, p. 531–542, doi: 10.1130/B31294.1.

Mix, H.T., Mulch, A., Kent-Corson, M.L., and Chamberlain, C.P., 2011, Cenozoic migration of topography in the North American Cordillera: Geology, v. 39, no. 1, p. 87–90, doi: 10.1130/G31450.1.

Mix, H.T., Winnick, M.J., Mulch, A., and Page Chamberlain, C., 2013, Grassland expansion as an instrument of hydrologic change in Neogene western North America: Earth and Planetary Science Letters, v. 377-378, p. 73–83, doi: 10.1016/j.epsl.2013.07.032.

Mulch, A., 2016, Stable isotope paleoaltimetry and the evolution of landscapes and life: Earth and Planetary Science Letters, v. 433, p. 180–191, doi: 10.1016/j.epsl.2015.10.034.

Mulch, A., Graham, S.A., and Chamberlain, C.P., 2006, Hydrogen isotopes in Eocene river gravels and paleoelevation of the Sierra Nevada: Science, v. 313, no. 5783, p. 87–89, doi: 10.1126/science.1125986.

Mulch, A., Sarna-Wojcicki, A.M., Perkins, M.E., and Chamberlain, C.P., 2008, A Miocene to Pleistocene climate and elevation record of the Sierra Nevada (California).: Proceedings of the National Academy of Sciences of the United States of America, v. 105, no. 19, p. 6819–6824, doi: 10.1073/pnas.0708811105.

Oster, J.L., Montañez, I.P., and Kelley, N.P., 2012, Response of a modern cave system to large seasonal precipitation variability: Geochimica et Cosmochimica Acta, v. 91, p. 92–108, doi: 10.1016/j.gca.2012.05.027.

Pagani, M., Liu, Z., LaRiviere, J., and Ravelo, A.C., 2010, High Earth-system climate sensitivity determined from Pliocene carbon dioxide concentrations: Nature Geoscience, v. 3, no. 1, p. 27–30, doi: 10.1038/ngeo724.

Passey, B.H., Levin, N.E., Cerling, T.E., BROWN, F.H., Eiler, J.M., and Turekian, K.K., 2010, High-temperature environments of human evolution in East Africa based on bond ordering in paleosol carbonates: Proceedings of the National Academy of Sciences of the United States of America, v. 107, no. 25, p. 11245–11249, doi: 10.1073/pnas.1001824107.

Peters, N.A., Huntington, K.W., and Hoke, G.D., 2013, Hot or not? Impact of seasonally variable soil carbonate formation on paleotemperature and O-isotope records from clumped isotope thermometry: Earth and Planetary Science Letters, v. 361, p. 208–218, doi:

10.1016/j.epsl.2012.10.024.

Phillips, F.M., 2008, Geological and hydrological history of the paleo – Owens River drainage since the late Miocene: History, v. 2439, no. 06, p. 1–36, doi: 10.1130/2008.2439(06).

Poage, M.A., and Chamberlain, C.P., 2001, Empirical relationships between elevation and the stable isotope composition of precipitation and surface waters: Considerations for studies of paleoelevation change: American Journal of Science, v. 301, no. 1, p. 1–15, doi: 10.2475/ajs.301.1.1.

Pound, M.J., Tindall, J., Pickering, S.J., Haywood, A.M., Dowsett, H.J., and Salzmann, U., 2014, Late Pliocene lakes and soils: A global data set for the analysis of climate feedbacks in a warmer world: Climate of the Past, v. 10, no. 1, p. 167–180, doi: 10.5194/cp-10-167-2014.

Previdi, M., and Liepert, B.G., 2007, Annular modes and Hadley cell expansion under global warming: Geophysical Research Letters, v. 34, no. 22, p. 5–9, doi: 10.1029/2007GL031243.

Quade, J., Breecker, D.O., Daeron, M., and Eiler, J., 2011, The paleoaltimetry of Tibet: An isotopic perspective: American Journal of Science, v. 311, no. 2, p. 77–115, doi: 10.2475/02.2011.01.

Quade, J., Eiler, J., Daeron, M., and Achyuthan, H., 2013, The clumped isotope geothermometer in soil and paleosol carbonate: Geochimica et Cosmochimica Acta, v. 105, p. 92–107, doi: 10.1016/j.gca.2012.11.031.

Raymo, M.E., Grant, B., Horowitz, M., and Rau, G.H., 1996, Mid-Pliocene warmth: stronger greenhouse and stronger conveyor: Marine Micropaleontology, v. 27, no. 1-4, p. 313–326, doi: 10.1016/0377-8398(95)00048-8.

Reheis, M., Sarna-Wojcicki, A.M., Reynolds, R.L., Repenning, C.A., and Mifflin, M.D., 2002, Pliocene to middle Pleistocene lakes in the western Great Basin: ages and connections: Great Basin Aquatic Systems History, p. 53–108.

Risi, C., Landais, A., Bony, S., Jouzel, J., Masson-Delmotte, V., and Vimeux, F., 2010, Understanding the ^{17}O excess glacial-interglacial variations in Vostok precipitation: Journal of Geophysical Research Atmospheres, v. 115, no. 10, p. 1–15, doi: 10.1029/2008JD011535.

Rohrmann, A., Strecker, M.R., Bookhagen, B., Mulch, A., Sachse, D., Pingel, H., Alonso, R.N., Schildgen, T.F., and Montero, C., 2014, Can stable isotopes ride out the storms? The role of convection for water isotopes in models, records, and paleoaltimetry studies in the central

Andes: Earth and Planetary Science Letters, v. 407, p. 187–195, doi: 10.1016/j.epsl.2014.09.021.

Salzmann, U., Haywood, A.M., Lunt, D.J., Valdes, P.J., and Hill, D.J., 2008, A new global biome reconstruction and data-model comparison for the Middle Pliocene: Global Ecology and Biogeography, v. 17, no. 3, p. 432–447, doi: 10.1111/j.1466-8238.2008.00381.x.

Salzmann, U., Williams, M., Haywood, A.M., Johnson, A.L.A., Kender, S., and Zalasiewicz, J., 2011, Climate and environment of a Pliocene warm world: Palaeogeography, Palaeoclimatology, Palaeoecology, v. 309, no. 1-2, p. 1–8, doi: 10.1016/j.palaeo.2011.05.044.

Scheff, J., and Frierson, D.M.W., 2012, Robust future precipitation declines in CMIP5 largely reflect the poleward expansion of model subtropical dry zones: Geophysical Research Letters, v. 39, no. 17, p. 1–6, doi: 10.1029/2012GL052910.

Seager, R., Naik, N., and Vecchi, G.A., 2010, Thermodynamic and dynamic mechanisms for large-scale changes in the hydrological cycle in response to global warming: Journal of Climate, v. 23, no. 17, p. 4651–4668, doi: 10.1175/2010JCLI3655.1.

Seager, R., Ting, M., Held, I., Kushnir, Y., Lu, J., Vecchi, G., Huang, H.-P., Harnik, N., Leetmaa, A., Lau, N.-C., Li, C., Velez, J., and Naik, N., 2007, Model projections of an imminent transition to a more arid climate in southwestern North America.: Science (New York, N.Y.), v. 316, no. 5828, p. 1181–4, doi: 10.1126/science.1139601.

Seager, R., Ting, M., Li, C., Naik, N., Cook, B., Nakamura, J., and Liu, H., 2012, Projections of declining surface-water availability for the southwestern United States: Nature Climate Change, v. 3, no. 5, p. 482–486, doi: 10.1038/nclimate1787.

Seager, R., and Vecchi, G., 2010, Greenhouse warming and the 21st century hydroclimate of southwestern North America: Proceedings of the National Academy of Sciences, v. 107, no. 50, p. 21277–21282, doi: doi: 10.1073/pnas.0910856107.

Sherwood, S., and Fu, Q., 2014, A Drier Future? Science, v. 343, no. 6172, p. 737–739, doi: 10.1126/science.1247620.

Sjostrom, D.J., and Welker, J.M., 2009, The influence of air mass source on the seasonal isotopic composition of precipitation, eastern USA: Journal of Geochemical Exploration, v. 102, no. 3, p. 103–112, doi: 10.1016/j.gexplo.2009.03.001.

Stein, A.F., Draxler, R.R., Rolph, G.D., Stunder, B.J.B., Cohen, M.D., and Ngan, F., 2015, NOAA's HYSPLIT atmospheric transport and dispersion modeling system: Bulletin of the American Meteorological Society, v. 96, no. 12, p. 2059–2077, doi: 10.1175/BAMS-D-14-00110.1.

Thompson, R.S., 1991, Pliocene environments and climates in the western United States: Quaternary Science Reviews, v. 10, no. 2-3, p. 115–132, doi: 10.1016/0277-3791(91)90013-K.

Tjiputra, J.F., Olsen, A., Bopp, L., Lenton, A., Pfeil, B., Roy, T., Segschneider, J., Totterdell, I., and Heinze, C., 2014, Long-term surface pCO2 trends from observations and models: Tellus, Series B: Chemical and Physical Meteorology, v. 66, no. 1, p. 1–24, doi: 10.3402/tellusb.v66.23083.

Wheeler, L.B., Galewsky, J., Herold, N., and Huber, M., 2016, Late Cenozoic surface uplift of the southern Sierra Nevada (California, USA): A paleoclimate perspective on lee-side stable isotope paleoaltimetry: Geology, , no. 6, p. G37718.1, doi: 10.1130/G37718.1.

Winnick, M.J., Caves, J.K., and Chamberlain, C.P., 2015, A mechanistic analysis of early Eocene latitudinal gradients of isotopes in precipitation: Geophysical Research Letters, v. 42, no. 19, p. 8216–8224, doi: 10.1002/2015GL064829.

Winnick, M.J., Chamberlain, C.P., Caves, J.K., and Welker, J.M., 2014, Quantifying the isotopic "continental effect": Earth and Planetary Science Letters, v. 406, p. 123–133, doi: 10.1016/j.epsl.2014.09.005.

Winnick, M.J., Welker, J.M., and Chamberlain, C.P., 2013, Stable isotopic evidence of El Ni no-like atmospheric circulation in the Pliocene western United States: Climate of the Past, v. 9, no. 2, p. 9–12, doi: 10.5194/cp-9-903-2013.

Woodhouse, C.A., Meko, D.M., MacDonald, G.M., Stahle, D.W., and Cook, E.R., 2010, A 1,200-year perspective of 21st century drought in southwestern North America.: Proceedings of the National Academy of Sciences of the United States of America, v. 107, no. 50, p. 21283–8, doi: 10.1073/pnas.0911197107.

Yin, J.H., 2005, A consistent poleward shift of the storm tracks in simulations of 21st century climate: Geophysical Research Letters, v. 32, no. 18, p. 1–4, doi: 10.1029/2005GL023684.

Zimmermann, U., Ehhalt, D., and Munnich, K.O., 1967, Soil-water movement and evapotranspiration: Changes in the isotopic composition of water, *in* Isotopes and Hydrology, International Atomic Energy Agency, Vienna, p. 567–586.

Appendix A. Large Data Tables and Files

A number of large data tables and files are alluded to in the text or are used in the code for the model. These tables and files are outlined below and are included in the electronic version of this submission.

A.1 Large Data Tables

The data tables containing modern isotopic values for soils and waters can be found in the following attached files:

1) "modern soils.csv"
2) "modern water d18O dD – no USGS.csv"
3) "modern water d18O dD.xlsx"
4) "I&Ttable.xlsx"
5) "Cotton 2012.csv"
6) "Cotton 2013.csv"

1) "modern soils.csv" contains isotopic values for six soil profiles, including some not currently published.
2-3) "modern water d18O dD – no USGS.csv" contains isotopic values for modern waters, mostly collected during the winter months. Many of the referenced publications contain further non-winter values that were not included in the compilation. "modern water d18O dD.xlsx" is similar, except that it also includes many data from USGS, many of which are from groundwater and/or were not collected during winter months.
4) "I&Ttable.xlsx" contains the data from Ingraham and Taylor's (1991) Traverse II, used for plot generation.
5-6) "Cotton 2012.csv" and "Cotton 2013.csv" contain data for soil respiration – mean annual precipitation relationships. These are neither addressed nor relevant to this project, but they are included because they are utilized in the attached code.

A.2 Large Data Files

A number of large data files were created and used throughout this project, attached here to limit time from re-creating or re-downloading.

1) "CaliforniaNevadaStormTrack_NARR_July2.RData"
2) “NARRgriddata.RData”
3) “NARR_RData” – contained in “/Elements/NARR_RData”
4) “HYSPLIT_output_NARR” – contained in “/Elements/HYSPLIT_output_NARR”

1) "CaliforniaNevadaStormTrack_NARR_July2.RData" contains the NARR monthly-mean values for the climatological variables input to the vapor transport and soil moisture models.
2) “NARRgriddata.RData” contains all of the latitude, longitude, and land mask cover values for the entire NARR dataset.
3) “NARR_RData” is a folder containing a file for each year of 3-hourly NARR data in files named “NARRyyyy.RData.” Each of these files contains the Damköhler number, precipitable water, relative humidity, air temperature, and timestamp (hours since 1800-1-1 00:00:0.0) associated with each NARR observation for that year. Note that each of these files is quite large (~2.3 GB) and *they should not all be loaded into the R environment at the same time*!
4) “HYSPLIT_output_NARR” is a folder containing all of the direct HYSPLIT output for each location, the .nc histogram files produced by “Trajectory Plotter-NARR_AJR” (*e.g.* “histhay.DJF.nc”), the .RData files generated by “Trajectory Plotter-NARR_AJR” (*e.g.* “hay.DJF.precip.RData”), the lists of HYSPLIT output integrated with appropriate NARR data (*e.g.* “list.Hay.DJF.withNARR.RData”), and the model results along each individual trajectory (*e.g.* “Hay.DJF.modelresults.RData”).

Appendix B. Description of Code

Many R files were used and included in this project. Those used to run the model and to create the plots shown in this document are described below.

B.1 "bulk code.R"

This file contains the bulk of the code used to generate plots throughout this project. Ten hypothetical NARR trajectories are considered and averaged, and they are used to feed the vapor transport model and soil moisture models. This file also contains the code for soil carbon isotopic values, though those are not addressed in this project. "bulk code.R" contains many sub-sections:
1) "parameters and inputs" – This section defines constants and establishes vectors for evaporation/transpiration partitioning ratios and depths in a soil profile. This section also defines the initial values for δ^{18}O to initialize model runs.
2) "necessary equations" – This section defines equations for fractionations and carbon concentration calculations used later in the code.
3) "basic atmosphere model for multiple storm tracks" – This section runs the Hendricks model for both advection-only and eddies-only cases along the ten storm tracks input from the NARR data.
4) "average tracks together by distance from coast" – This section runs the same model (the code is copy-pasted from section 3), except along a mean storm track where the NARR values are averaged by their distance from the coast.
5) "model plots" – This section produces plots of model output, without any data.
6) "data time!" – This section loads, organizes, and plots data (both as maps and in conjunction with model output).
7) "Ingraham and Taylor, 1991 plots" – This section creates plots using just the Ingraham and Taylor (1991) Traverse II data.
8) "resp-MAP relationships" – This section defines three respiration-mean annual precipitation relationships relevant to carbon isotope calculations. This section is not relevant to this project.
9) "soil carbonate function" – This section defines a function that calculates δ^{13}C of soil carbonate from CO_2 concentrations and mean annual precipitation rates. This is not relevant to this project.
10) "soil carbon profile calculations" – This section calculates profiles of soil carbon isotope

values along the mean storm track. This section is not relevant to this project.

11) "plot model soil carbonate profiles" – This section plots the soil carbon profiles. This section is not relevant to this project.

12) "soil water component" – This section calculates the δ^{18}O of soil water profiles along the mean storm track.

13) "plot model soil water profiles" – This section plots the soil water profiles.

14) "modern soil profiles" – This section plots modern soil δ^{18}O data in conjunction with the model-output profiles.

15) "remaking map" – This section replots the map, including locations for compiled water and soil data and the NARR data locations used.

16) "porosity sensitivity analysis" – This section performs a sensitivity analysis on the initial value of porosity used.

B.2 "non-dimensional model.R"

This file contains the code to predict δ^{18}O, d-excess, and ^{17}O-excess along hypothetical storm tracks with non-dimensional distance. A plot is created for these values at varying E/ET partitioning ratios. All climatological inputs are mostly arbitrary, but the equations are otherwise the same as those in the "bulk code.R" file.

B.3 HYSPLIT/Model Integration

All code relevant to the integration of the model with individual HYSPLIT trajectories is described in Appendix C.

Appendix C. HYSPLIT and Model Integration

A large portion of this project was devoted to integrating the model with individual storm path trajectories output by HYSPLIT, but the results thus far have been unsatisfactory. HYSPLIT trajectories are output as a set of coordinates associated with the number of hours since trajectory initialization (note that these are negative hours due to the back-trajectory calculation performed by HYSPLIT). NARR also contains data as a grid of coordinates and data values associated with a timestamp (hours since 1800-1-1 00:00:0.0). Integrating these for use in the model requires placing NARR data, of appropriate time and location, in the format of a trajectory. The following outlines the steps required to integrate the two dataset and to run the model on individual trajectories. Note that the majority of the code mentioned below requires a computer with access to large volumes of memory (RAM).

C.1 Prepare HYSPLIT trajectories for use in R: "Trajectory Plotter-NARR_AJR.R"

The "Trajectory Plotter-NARR_AJR.R" file is used to manipulate the raw HYSPLIT output. It is used to separate out the precipitating trajectories, generate their histograms, plot them, and output RData files for all of the precipitation-producing trajectories from a given initialization point. These RData files, stored in "HYSPLIT output" (see Appendix A.2), will be used later for integration.

C.2 Prepare the NARR dataset for use in R: "dataset processing.R"

Our first step in integrating the model with HYSPLIT trajectories was to make the HYSPLIT and NARR datasets compatible with each other, firstly by making all of the NARR data available for use in R. This was done in "dataset processing.R," in which all of the NARR NetCDF files are opened and saved as .RData files by year (stored in "NARR_RData"; see Appendix A.2). It is important to mention that each of these is a very large data file, and care must be taken so as not to exceed a computer's memory capacity. This code functions with on a computer with 16 GB RAM (lower RAM values were not tested).

C.3 Convert HYSPLIT timestamps to NARR-compatible timestamps: "timechanger.R"

The next step is to make the timestamps for HYSPLIT (year/month/day/hour) compatible with those for the NARR dataset (hours since 1800-1-1 00:00:0.0). This is done through the function defined in "timechanger.R." This function takes a dataframe of meteorological variables (the direct HYSPLIT output), and adds an additional column for the modified timestamp to match the hours since 1800-1-1 00:00:0.0 format used in the NARR data.

C.4 Convert HYSPLIT coordinates to NARR-compatible grid: "latlonNARR.R"

Though the HYSPLIT output is based on the raw NARR data, the output coordinates are interpolated values from the NARR grid values. Thus, to integrate the two datasets, we could either interpolate the NARR dataset or set the HYSPLIT output to be compatible with the NARR grid. We choose the latter both to avoid increased error due to interpolation, and because interpolating the NARR dataset to the resolution of HYSPLIT output is computationally quite expensive. Thus, we use the function defined in "latlonNARR.R" to use a nearest-neighbor search to select the NARR coordinate closest in space to each HYSPLIT trajectory coordinate. This function takes in a dataframe of a time-corrected HYSPLIT output trajectory and replaces the latitude and longitude columns with the appropriate NARR coordinates for each point in the trajectory.

C.5 Integrate NARR data into trajectory points: "NARRgrabber.R"

The final step in integrating the NARR and HYSPLIT output datasets is to associate each point in each trajectory with the appropriate NARR climatological data at that location and time. This is handled by the functions defined in "NARRgrabber.R," which take in a dataframe of corrected coordinates and timestamps and append columns for the NARR climatological variables needed for the vapor transport model (Damköhler number, precipitable water, relative humidity, air temperature, and land mask cover). This function needs access to all of the NARR dataset; each year is about 2.3 GB of data. The number of files that are in use at any given point is minimized, but large volumes of memory are still required. This function will not crash a computer with 16 GB of RAM, but lower RAM values were not tested.

C.6 Establish equations necessary to run the model: "model.sourcefunctions.R"

Several functions to calculate isotopic values are used by the vapor transport model. Those for fractionation and for calculating the integrated isotopic composition of evapotranspiration are defined in "model.sourcefunctions.R." This file also contains functions for soil carbon calculations, which are not currently relevant to this aspect of the project.

C.7 Run the vapor transport model along a single trajectory: "Hendricksmodel.onetraj.R"

Once the coordinates and timestamps have been corrected and the appropriate NARR data has been extracted for each trajectory, we can run the model along a single trajectory using the function defined by "Hendricksmodel.onetraj.R." This function takes in a dataframe of chronologically ordered meteorological values (each row is a time point, each column is an input), runs the model along this trajectory, and appends columns for the model-output isotopic values. The current setup of the function only outputs for δ^{18}O, but including δD and δ^{17}O only requires uncommenting one line of code and is clearly demarcated. It is also worth noting that the "Hendricksmodel.onetraj.R" file also contains the values for constants needed for the model. This function is meant to be used as part of the script "atmosphere.fromHYSPLIT.R," described below with a more complete description of the necessary input format.

C.8 Full application of model to HYSPLIT output: "atmosphere.fromHYSPLIT.R"

The final aspect of integrating the vapor transport model to individual HYSPLIT trajectories was to run the model along multiple individual trajectories. This is handled in "atmosphere.fromHYSPLIT.R."

First, all appropriate function files are loaded. Next, a list of dataframes as generated: the list represents all of the trajectories for a given location/season, and each list element contains a dataframe corresponding to an individual trajectory. For each dataframe in the list, the coordinates and timestamps are corrected. This will take time. Next, for each list element (each dataframe trajectory), the "NARRgrabber" function is called to append appropriate NARR data to each row of each dataframe. This step is extremely computationally expensive and will take quite some time. The list is then saved (*e.g.* "list.Hay.DJF.withNARR.RData" as in Appendix A.4) such that re-computing is not necessary. Next, all NA points (those without NARR data) are removed from the dataframes, and each dataframe is re-ordered to chronological order (because original trajectory outputs are in reverse chronological order).

To finalize the dataframes before input into the model, we must append values of "t," defined as the ratio of transpiration to total evapotranspiration (T/ET), to each point along each trajectory/dataframe in the list. The current setup of the code appends a single value to all points along each trajectory, though it is possible to assign unique values to each point (not handled here) to allow for changing T/ET values along a trajectory. Now, we have a list of dataframes to which we can apply the “Hendricksmodel.onetraj” function. Each dataframe is formatted as follows: Each row represents a point along a single trajectory, where rows are in chronological order (earliest timestamp at first row). There are 11 columns: Year, Back.Hour, Lat, Lon, Time (hours since 1800-1-1 00:00:0.0), Land (land mask cover; 0 or 1), Nd (Damköhler number), Prw (precipitable water), rhum (relative humidity as percent), Temp_2m (2m air temperature, Kelvin), t (T/ET ratio). The output is a list of dataframes, where each dataframe represents a single trajectory with meteorological variables and the isotopic values output by the vapor transport model for each point along the trajectory.

While we were successfully able to produce code for the computationally expensive problem of integrating individual trajectories with NARR data and our vapor transport model, the results are unsatisfactory and suggest room for improvement due to the following three limitations. First, the NARR dataset is imperfect, particularly at local scales. The presence of negative precipitation rates and otherwise questionable values at some locations introduces many potential problems when these data are used at the scale of individual complexities. Second, evapotranspiration must be better constrained. At this point, we are assuming ratios that are likely not fully representative of the actual values experienced along a storm path. Third, and perhaps most importantly, this model assumes constant initial isotopic compositions for each trajectory. Parts of the model will have to be adjusted to consider spatially and temporally variable initial compositions.

Appendix D. Code

Below are copies of the code for the files mentioned in Appendices B and C.

D.1 "bulk code.R"

```
#this script creates C and O isotope profiles in soils along a designated
storm track
#both advection-only and eddy diffusion-only cases explored
#fixing the diffusion coefficient units, because that was a major fail before
#adapted from multiple different script components:
   #"model w evap balance and dxs.R" by Matt Winnick (atmosphere)
   #"SoilModule_d18OCodeV2.R" by D.Ibarra and Chamberlain (soil moisture)
   #"SR_preciprelationship.R" and "Soil Resp-forward.R" by J. Caves (soil
respiration)
#created 4-28-15 by AJR
#last modified: 5-28-16 by AJR


setwd('/Users/Annie/Documents/model')
load("CaliforniaNevadaStormTrack_NARR_July2.RData")
##^this RData file contains 28x10 (28 points of 10 tracks) dataframes of the
following variables:
#MAP.data          #mean annual precipitation rate (mm/month)
#Nd1.data          #Nd (evap/runoff) unitless DJF average, calculated with
evap and NARR runoff
#Nd2.data          #Nd (ET/P-ET) unitless DJF average, calculated using lhtfl
as ET
#Prw.data          #Precipitable water (w) kg/m^2 DJF average
#Temp_2m.data      #2m air temperature DFJ average (K)
#Temp_surf.data    #surface temperature JJASO average (K)
#evap.JJASO.data   #evapotranspiration (from lhtfl), JJASO average (kg/m^2)
#land.mask.data    #land mask (0 if not land, 1 if land)
#lat.data          #latitude
#lon.data          #longitude
#rhum.data         #2m relative humidity (%) DJF average
####additionally, 28x10x801 array at 801 depths (0-800cm)
#tsoil.data        #soil temperature (K) from depth 0-800cm (linearly
interpolated from original NARR data)

Nd.data=Nd2.data
evap.data=evap.JJASO.data #for coding simplification purposes

#####parameters and inputs#####

e=seq(1,0, by = -0.1) #evap as % of ET
t=1-e #transpiration as % of ET

Rpdb=0.0112372 #dimensionless PDB Belemnite Ratio
smow18=0.0020052 # smow ratio of 18O/16O
smowD=155.76*10^(-6) # smow ratio of D/H
smow17=379.9*10^(-6) # vsmow ratio of 17O/16O
theta.eq=0.529 #Barkan and Luz, 2005; equilibrium (l-v and v-l)
theta.diff=0.5185 #Barkan and Luz, 2007; diffusion
k18=1-0.01872 # kinetic fractionation factor for 18O, wet soils from Mathieu
and Bariac (1996), slightly higher than lake evap (~14.3)
kD=1-0.01676 # kinetic fractionation factor for dD, wet soils from Mathieu
and Bariac (1996), slightly higher than lake evap (~12.5)
k17=k18^theta.diff #Barkan and Luz, 2005, 2007

#INPUT d18O VALUES! (in permil)
initial_delta_p=-6.96 #average value from Ingraham and Taylor 1991, Traverse
II
```

```
initial_delta_a=-17.476 #assuming equlibrium with precip
initial_delta_p_eddy=initial_delta_p #this is what Matt used, at least
initial_delta_a_eddy=initial_delta_a


#####soil model inputs!!!

pH=8
delta18_atm_initial=initial_delta_p
delta18_precip_initial=initial_delta_a
maxdepth=100 #in cm, maximum soil profile depth
zplot=seq(0,maxdepth) #for plotting profiles

#keep only subset of soil temp data above (and at) maxdepth of soil:
tsoil.data=tsoil.data[,,1:(maxdepth+1)]

#soil properties
d=2.3e-5 #cm2/sec self diffusion coefficient of soil water
Dair=0.144 #cm2/s #CO2 diffusion coefficient in air (Cerling 1991)
p=0.5 #Free-air porosity (from Cerling 1991); Cerling and Quade 1993 use 0.6
tau=0.6 #Toturosity (0.7 is uniform sand from) Barnes and Allison (1983)
Ds=Dair*p*tau #cm2/s #Bulk CO2 diffusion coefficient in the air of soils

#leaving these here as equation reminder; they're handled elsewhere
#zstar=(p*tau*d)/E
#E=(p*tau*d)/zstar

M12=44 #Mass of 12CO2
M13=45 #Mass of 13CO2
Mair=29 #Average atomic mass of air
MCO2=44.00964 #Mass of bulk CO2
diff=sqrt(((MCO2+Mair)/(MCO2*Mair))*((M13*Mair)/(M13+Mair))) #Fractional
increase in 13CO2 diffusion coefficient over bulk CO2
#diff=1.0044

Ds13=Ds/diff #cm2/s #should be Ds/1.0044 relative to 12C

delta_13_o=-27 #per mil composition of soil-respired CO2
pCO2ppm_atm=280
CO2_atm_mg.m3=543 #mg/m3 #Atmospheric CO2 #Currently set for 280ppm
CO2_atm=CO2_atm_mg.m3/1000/100^3/MCO2 #moles/cm3 CO2
#CO2_atm=1.23381e-8 #moles/cm3 This is at 280ppm
delta_13_atm=-6.5 #per mil of atmospheric CO2

z=50 #cm depth IN soil CRITICAL VALUE!! for determining depth-MAP
relationship
L=100 #cm depth of soil column with linear production (for resp-MAP
relationship)
#Soil parameters assume linear production with depth with a soil depth (L) of
100 cm.


########necessary equations##############

# Calculates the fractionation factor to soil carbonate as a function
# of temperature. Default is 25C.
fract=function(Temp=25){ # From Cerling (1999)--see Figure 5
  fractionation=-(11.709-0.116*Temp+2.16e-4*Temp^2)
  fractionation
}

# Converts CO2 from ppmv to mols/cm3 for use in the soil respiration
# equation.  This conversion is a function of temperature (default is 25C).
CO2convert=function(CO2ppmv,Temp=25){
  CO2v=CO2ppmv/1e6
  MCO2=44.00964 # Molecular mass of bulk CO2
  SP=101325 # Standard pressure (Pa) at 1atm
  R=8.31441 # Universal Gas Constant
  TempK=Temp+273.15 # Correction to Kelvin
  CO2m=(MCO2*SP*CO2v)/(R*TempK) # g/m3 of CO2
  CO2moles=CO2m/MCO2/1e6 # moles/cm3
  CO2moles #moles/cm3
}

# This function does the opposite of the CO2convert function. It takes
# CO2 as moles/cm3 and converts back to ppm.
CO2backconvert=function(CO2_atm,Temp=25){
  MCO2=44.00964 # Molecular mass of bulk CO2
  SP=101325 # Standard pressure (Pa) at 1atm
  R=8.31441 # Universal Gas Constant
  TempK=Temp+273.15 # Correction to Kelvin
  CO2mm3=CO2_atm*MCO2*1e6 # g/m3 of CO2
  CO2v=(CO2mm3*R*TempK)/(SP*MCO2) # absolute quantity of CO2
  CO2ppm=CO2v*1e6 # ppm CO2
  CO2ppm #ppm
}


##function to calculate either d18O (if d18 = True in input; default) or dD
(if d18 = False)
#of ET using Craig and Gordon equations and closure assumption
craig_gordon <- function(t, h, delta_precip, ts, d18 = T){
  if(d18 == T){
    alpha18=1 - (-7.685 + (6.7123*10^3)/(ts) - (1.6664*10^6)/(ts)^2 +
(0.35041*10^9)/(ts)^3)/1000 #equilibrium fractionation 18O, l-v
    r18_s <- (delta_precip/1000 + 1)*smow18  #ratio of precip sample
    Re_18 <- (1-t)*((alpha18*k18*r18_s)/(1-h+(1-t)*k18*h)) +
t*r18_s*(1/(1+(1-t)*k18*(h/(1-h))))
    d18_evap <- (Re_18/smow18 - 1)*1000
    (d18_evap - delta_precip)/1000}
  else{
    alphaD=1 - ((1158.8*ts^3)/10^9 - (1620.1*ts^2)/10^6 + (794.84*ts)/10^3 -
161.04 + (2.9992*10^9)/ts^3)/1000 #equilibrium fractionation D, l-v
    rD_s <- (delta_precip/1000 + 1)*smowD
    Re_D <- (1-t)*((alphaD*kD*rD_s)/(1-h+(1-t)*kD*h)) + t*rD_s*(1/(1+(1-
t)*kD*(h/(1-h))))
    dD_evap <- (Re_D/smowD - 1)*1000
    (dD_evap - delta_precip)/1000}
}

##function to calculate d17O of ET using Craig-Gordon and closure assumption
craig_gordon17=function(t,h,delta_precip,ts) {
  alpha18=1 - (-7.685 + (6.7123*10^3)/(ts) - (1.6664*10^6)/(ts)^2 +
(0.35041*10^9)/(ts)^3)/1000 #equilibrium fractionation 18O, l-v
  alpha17=alpha18^(theta.eq) #Barkan and Luz, 2005
  r17_s=(delta_precip/1000+1)*smow17
  Re_17=(1-t)*((alpha17*k17*r17_s)/(1-h+(1-t)*k17*h))+t*r17_s*(1/(1+(1-
t)*k17*(h/(1-h))))
  d17_evap=(Re_17/smow17-1)*1000
  (d17_evap-delta_precip)/1000
}


##function to calculate integrated isotopic composition of ET with
#integration of Rayleigh distillation for soil moisture reservoir
isotope_e = function(resid, delta_precip, eps_e){
  delt = -resid*((1000/(1+eps_e))*resid^(eps_e)-1000)
  delta_precip + delt
```

```
}

#function to convert negative values to NAs (to get rid of neg. Nds)
#only used when averaging tracks
neg.toNA=function(matrix) {
  for (i in 1:dim(matrix)[1]) {
    for (j in 1:dim(matrix)[2]) {
      if (matrix[i,j]<=0) {
        matrix[i,j]=NA
      }
    }
  }
  matrix
}


#########basic atmosphere model for multiple storm tracks###########

#model input NARR data
w=as.matrix(Prw.data) #precipitatble water, kg/m^2
ntracks=length(w[1,]) #number of storm tracks
tracklength=length(w[,1]) #length of storm tracks
Nd=as.matrix(Nd.data) #input array from NARR data
ts=as.matrix(Temp_2m.data) #2m surface temperature (K)
rh=as.matrix(rhum.data/100) #relative humidity (fraction)


#advection only (Hendricks):
#delta_a=(delta_a_initial-delta_a_inf)*exp((alpha+alpha*Nd-1)*(-
x/l))+delta_a_inf
#delta_a_inf=(Nd*delta_et-(1+Nd)*(alpha-1)*1000)/(alpha+alpha*Nd-1)
#(-x/l)=-x' ; l=-w*(dx/dw) ==> -x'=(x/w)*(dw/dx)

#eddy diffusion only (Hendricks):
#delta_a_eddy=(delta_a_initial-delta_a_inf)*exp((sqrt(alpha+alpha*Nd)-1)*(-
x/l))+delta_a_inf

#constants for isotope value calculations:
#note that a18 != alpha18 (and same for D)
a18=1 + (-7.685 + (6.7123*10^3)/(ts) - (1.6664*10^6)/(ts)^2 +
(0.35041*10^9)/(ts)^3)/1000 #equilibrium fractionation 18O, v-l
aD=1 + ((1158.8*ts^3)/10^9 - (1620.1*ts^2)/10^6 + (794.84*ts)/10^3 - 161.04 +
(2.9992*10^9)/ts^3)/1000 #equilibrium fractionation, D v-l
alpha18=1 - (-7.685 + (6.7123*10^3)/(ts) - (1.6664*10^6)/(ts)^2 +
(0.35041*10^9)/(ts)^3)/1000 #equilibrium fractionation 18O, l-v
alphaD=1 - ((1158.8*ts^3)/10^9 - (1620.1*ts^2)/10^6 + (794.84*ts)/10^3 -
161.04 + (2.9992*10^9)/ts^3)/1000 #equilibrium fractionation D, l-v
alpha17=alpha18^(theta.eq) #Barkan and Luz, 2005
a17=a18^theta.eq

#build empty arrays to store data for d18O and dD and d17O for advection-
only/eddy-only cases
delta_a <- delta_et <- delta_inf<- delta_p <- delta_a_eddy <- delta_et_eddy
<- delta_inf_eddy <- delta_p_eddy <-
array(dim=c(tracklength,ntracks,length(e)))
delta_D_a <- delta_D_et <- delta_D_inf <- delta_D_p <- delta_D_a_eddy <-
delta_D_et_eddy <- delta_D_inf_eddy <- delta_D_p_eddy <-
array(dim=c(tracklength,ntracks,length(e)))
delta_17_a <- delta_17_et <- delta_17_inf <- delta_17_p <- delta_17_a_eddy <-
delta_17_et_eddy <- delta_17_inf_eddy <- delta_17_p_eddy <-
array(dim=c(tracklength,ntracks,length(e)))

delta_a[1,,] = initial_delta_a #sets initial d18O atmospheric vapor for all
tracks and E/ETs
```

```
delta_p[1,,] = initial_delta_p #sets intial d18O precip  #ntracksxlength(e)
delta_D_a[1,,] = delta_a[1,,]*8+10 #sets initial atmospheric vapor dD
delta_D_p[1,,] = delta_p[1,,]*8+10 #sets initial precip dD
delta_17_a[1,,] = (exp(0.528*log(delta_a[1,,]/1000+1)+0.000033)-1)*1000
#GMWL, Luz and Barkan, 2010
delta_17_p[1,,] = (exp(0.528*log(delta_p[1,,]/1000+1)+0.000033)-1)*1000
#GMWL, Luz and Barkan, 2010
#same for eddies:
delta_a_eddy[1,,] = initial_delta_a_eddy #sets initial d18O atmospheric vapor
for all tracks and E/ETs
delta_p_eddy[1,,] = initial_delta_p_eddy #sets intial d18O precip
#ntracksxlength(e)
delta_D_a_eddy[1,,] = delta_a_eddy[1,,]*8+10 #sets initial atmospheric vapor
dD
delta_D_p_eddy[1,,] = delta_p_eddy[1,,]*8+10 #sets initial precip dD
delta_17_a_eddy[1,,] = (exp(0.528*log(delta_a_eddy[1,,]/1000+1)+0.000033)-
1)*1000  #GMWL, Luz and Barkan, 2010
delta_17_p_eddy[1,,] = (exp(0.528*log(delta_p_eddy[1,,]/1000+1)+0.000033)-
1)*1000  #GMWL, Luz and Barkan, 2010


#make (mostly empty) matrix of total distance from initial point
distance.m=array(dim=c(tracklength,ntracks))
distance.m[1,]=0 #first distance zero because we haven't gone anywhere yet

###run the model!

library(geosphere) #to calculate distance between coordinates (Haversine)

#iterate over all points along each track for each E/ET ratio
for (h in 1:length(e)) {
  for(j in 1:ntracks) {
    for (i in 2:tracklength) {
      #calculate residual soil moisture for isotope_e function
      if (land.mask.data[i,j]==0) {
        residual=0   #sets 0 residual soil moisture when not over land
      } else {
        residual=1-(Nd[i,j]/(Nd[i,j]+1))*e[h]/(1-(Nd[i,j]/(Nd[i,j]+1)*t[h]))}
      #if everything evaporates, set isotope value of evaporation to isotope
value of precip
      if(residual == 0){
        delta_et[i,j,h] <- delta_p[i-1,j,h]
        delta_D_et[i,j,h] <- delta_D_p[i-1,j,h]
        delta_17_et[i,j,h] = delta_17_p[i-1,j,h]
        delta_et_eddy[i,j,h]=delta_p_eddy[i-1,j,h]
        delta_D_et_eddy[i,j,h] <- delta_D_p_eddy[i-1,j,h]
        delta_17_et_eddy[i,j,h] = delta_17_p_eddy[i-1,j,h]

        #otherwise, first calculate the combined equilibrium/kinetic
fractionation using Craig-Gordon function
      } else {
        eps_e <- craig_gordon(t[h], rh[i,j], delta_p[i-1,j,h], ts[i,j])
        eps_e_D <- craig_gordon(t[h], rh[i,j], delta_D_p[i-1,j,h], ts[i,j],
d18 = F)
        eps_e_17 = craig_gordon17(t[h],rh[i,j],delta_17_p[i-1,j,h],ts[i,j])
        eps_e_eddy <- craig_gordon(t[h], rh[i,j], delta_p_eddy[i-1,j,h],
ts[i,j])
        eps_e_D_eddy <- craig_gordon(t[h], rh[i,j], delta_D_p_eddy[i-1,j,h],
ts[i,j], d18 = F)
        eps_e_17_eddy = craig_gordon17(t[h],rh[i,j],delta_17_p_eddy[i-
1,j,h],ts[i,j])
        #then calculate integrated ET value from location
        delta_et[i,j,h] <- t[h]*delta_p[i-1,j,h] +
e[h]*isotope_e(residual,delta_p[i-1,j,h], eps_e)
```

```
        delta_D_et[i,j,h] <- t[h]*delta_D_p[i-1,j,h] +
e[h]*isotope_e(residual,delta_D_p[i-1,j,h], eps_e_D)
        delta_17_et[i,j,h] = t[h]*delta_17_p[i-
1,j,h]+e[h]*isotope_e(residual,delta_17_p[i-1,j,h],eps_e_17)
        delta_et_eddy[i,j,h] <- t[h]*delta_p_eddy[i-1,j,h] +
e[h]*isotope_e(residual,delta_p_eddy[i-1,j,h], eps_e_eddy)
        delta_D_et_eddy[i,j,h] <- t[h]*delta_D_p_eddy[i-1,j,h] +
e[h]*isotope_e(residual,delta_D_p_eddy[i-1,j,h], eps_e_D_eddy)
        delta_17_et_eddy[i,j,h] = t[h]*delta_17_p_eddy[i-
1,j,h]+e[h]*isotope_e(residual,delta_17_p_eddy[i-1,j,h],eps_e_17_eddy)
      }


      ###there's an issue with some of the values after the initial point
being NAs
      #this loop will make it so all of the NAs stay NAs *except* the last NA
before
      #hitting land. the first ones stay NAs so that we're only calculating
over land.
      #note: still necessary to keep the original initials above in the event
of no NAs
      if (land.mask.data[i,j]==0) {
        delta_p[i,j,]=initial_delta_p
        delta_p[i-1,j,]=NA
        delta_a[i,j,]=initial_delta_a
        delta_a[i-1,j,]=NA
        delta_D_p[i,j,]=delta_p[i,j,]*8+10 #assume initial points are on MWL
        delta_D_p[i-1,j,]=NA
        delta_D_a[i,j,]=delta_a[i,j,]*8+10
        delta_D_a[i-1,j,]=NA
        delta_17_p[i,j,]=(exp(0.528*log(delta_p[i,j,]/1000+1)+0.000033)-
1)*1000  #GMWL, Luz and Barkan, 2010
        delta_17_p[i-1,j,]=NA
        delta_17_a[i,j,]=(exp(0.528*log(delta_a[i,j,]/1000+1)+0.000033)-
1)*1000  #GMWL, Luz and Barkan, 2010
        delta_17_a[i-1,j,]=NA
        #repeat for eddies (copypasta)
        delta_p_eddy[i,j,]=initial_delta_p_eddy
        delta_p_eddy[i-1,j,]=NA
        delta_a_eddy[i,j,]=initial_delta_a_eddy
        delta_a_eddy[i-1,j,]=NA
        delta_D_p_eddy[i,j,]=delta_p_eddy[i,j,]*8+10 #assume initial points
are on MWL
        delta_D_p_eddy[i-1,j,]=NA
        delta_D_a_eddy[i,j,]=delta_a_eddy[i,j,]*8+10
        delta_D_a_eddy[i-1,j,]=NA

delta_17_p_eddy[i,j,]=(exp(0.528*log(delta_p_eddy[i,j,]/1000+1)+0.000033)-
1)*1000  #GMWL, Luz and Barkan, 2010
        delta_17_p_eddy[i-1,j,]=NA

delta_17_a_eddy[i,j,]=(exp(0.528*log(delta_a_eddy[i,j,]/1000+1)+0.000033)-
1)*1000  #GMWL, Luz and Barkan, 2010
        delta_17_a_eddy[i-1,j,]=NA
      } else {
        #haversine formula to calculate distance between coordinates:
        x=distVincentyEllipsoid(c(lon.data[i,j],lat.data[i,j]),c(lon.data[i-
1,j],lat.data[i-1,j]))
        dx=x #meters, but the units don't matter. slightly >32km
        dw=w[i,j]-w[i-1,j]
        dw=ifelse(dw>0,0,dw) #gets rid of positive gradients (b/c we don't
like explosions here)
        #Hendricks model (advection only)
        delta_inf[i,j,h] <- (Nd[i,j]*delta_et[i,j,h]-(1+Nd[i,j])*(a18[i,j]-
1)*1000)/(a18[i,j]+Nd[i,j]*a18[i,j]-1)
        delta_a[i,j,h]=(delta_a[i-1,j,h]-
delta_inf[i,j,h])*exp((a18[i,j]+a18[i,j]*Nd[i,j]-
1)*(x/w[i,j])*(dw/dx))+delta_inf[i,j,h]
        delta_p[i,j,h] <-delta_a[i,j,h]+(a18[i,j]-1)*1000

        delta_D_inf[i,j,h] <- (Nd[i,j]*delta_D_et[i,j,h]-
(1+Nd[i,j])*(aD[i,j]-1)*1000)/(aD[i,j]+Nd[i,j]*aD[i,j]-1)
        delta_D_a[i,j,h]=(delta_D_a[i-1,j,h]-
delta_D_inf[i,j,h])*exp((aD[i,j]+aD[i,j]*Nd[i,j]-
1)*(x/w[i,j])*(dw/dx))+delta_D_inf[i,j,h]
        delta_D_p[i,j,h] <-delta_D_a[i,j,h]+(aD[i,j]-1)*1000

        delta_17_inf[i,j,h] <- (Nd[i,j]*delta_17_et[i,j,h]-
(1+Nd[i,j])*(a17[i,j]-1)*1000)/(a17[i,j]+Nd[i,j]*a17[i,j]-1)
        delta_17_a[i,j,h]=(delta_17_a[i-1,j,h]-
delta_17_inf[i,j,h])*exp((a17[i,j]+a17[i,j]*Nd[i,j]-
1)*(x/w[i,j])*(dw/dx))+delta_17_inf[i,j,h]
        delta_17_p[i,j,h] <-delta_17_a[i,j,h]+(a17[i,j]-1)*1000

        #Hendricks model (eddy diffusion only)
        delta_inf_eddy[i,j,h] <- (Nd[i,j]*delta_et_eddy[i,j,h]-
(1+Nd[i,j])*(a18[i,j]-1)*1000)/(a18[i,j]+Nd[i,j]*a18[i,j]-1)
        delta_a_eddy[i,j,h]=(delta_a_eddy[i-1,j,h]-
delta_inf_eddy[i,j,h])*exp((sqrt(a18[i,j]+a18[i,j]*Nd[i,j])-
1)*(x/w[i,j])*(dw/dx))+delta_inf_eddy[i,j,h]
        delta_p_eddy[i,j,h] <-delta_a_eddy[i,j,h]+(a18[i,j]-1)*1000

        delta_D_inf_eddy[i,j,h] <- (Nd[i,j]*delta_D_et_eddy[i,j,h]-
(1+Nd[i,j])*(aD[i,j]-1)*1000)/(aD[i,j]+Nd[i,j]*aD[i,j]-1)
        delta_D_a_eddy[i,j,h]=(delta_D_a_eddy[i-1,j,h]-
delta_D_inf_eddy[i,j,h])*exp((sqrt(aD[i,j]+aD[i,j]*Nd[i,j])-
1)*(x/w[i,j])*(dw/dx))+delta_D_inf_eddy[i,j,h]
        delta_D_p_eddy[i,j,h] <-delta_D_a_eddy[i,j,h]+(aD[i,j]-1)*1000

        delta_17_inf_eddy[i,j,h] <- (Nd[i,j]*delta_17_et_eddy[i,j,h]-
(1+Nd[i,j])*(a17[i,j]-1)*1000)/(a17[i,j]+Nd[i,j]*a17[i,j]-1)
        delta_17_a_eddy[i,j,h]=(delta_17_a_eddy[i-1,j,h]-
delta_17_inf_eddy[i,j,h])*exp((sqrt(a17[i,j]+a17[i,j]*Nd[i,j])-
1)*(x/w[i,j])*(dw/dx))+delta_17_inf_eddy[i,j,h]
        delta_17_p_eddy[i,j,h] <-delta_17_a_eddy[i,j,h]+(a17[i,j]-1)*1000
      }

      #this is essentially the distance from the last point before landfall
      #(or first point on land, but not for our lovely NARR swath)
      if (land.mask.data[i,j]==0) {
        distance.m[i-1,j]=0
        distance.m[i,j]=0
      } else {
        distance.m[i,j]=distance.m[i-1,j]+dx
      }
    }
  }
}
#rename all the resulting dataframes
delta_p.all=delta_p
delta_D_p.all=delta_D_p
delta_17_p.all=delta_17_p
delta_p_eddy.all=delta_p_eddy
delta_D_p_eddy.all=delta_D_p_eddy
delta_17_p_eddy.all=delta_17_p_eddy
distance.all=distance.m/1000
distance.fake = seq(0,by=32,length=tracklength) #approximate distances
```

```
###calculate D-excess and 17O-excess:
Dxs.all=delta_D_p.all-8*delta_p.all #in permil
O17xs.all=(log(delta_17_p.all/1000+1)-0.528*log(delta_p.all/1000+1))*10^6 #in
permeg
Dxs_eddy.all=delta_D_p_eddy.all-8*delta_p_eddy.all #in permil
O17xs_eddy.all=(log(delta_17_p_eddy.all/1000+1)-
0.528*log(delta_p_eddy.all/1000+1))*10^6 #in permeg

#delta_p[,,1] #quick look for E/ET=0
#^ NAs at beginning of tracks from where we're still over the ocean
#^ NaNs where Damkohler goes negative

#distance plot of delta_p

library(fields)
colors=tim.colors(ntracks)
colors_eddy=paste('gray',seq(5,86,length=10))
plot(distance.all[,1],delta_p.all[,1,1],type="o",pch=16,cex=0.5,xaxs="i",
     xlab='Distance from "Coast" (km)',ylab="delta_p",ylim=c(-
30,0),col=colors[1])
for (j in 2:10){

points(distance.all[,j],delta_p.all[,j,1],type="o",pch=16,cex=0.5,col=colors[
j])
}
for(j in 1:10) {

points(distance.all[,j],delta_p_eddy.all[,j,1],type='o',pch=15,cex=0.5,col=co
lors_eddy[j])
}


#plot of Nd
#for plotting distance from coast:
#convert Nd to zero when not over land
Nd.plot=Nd
for (j in 1:ntracks) {
  for (i in 2:tracklength) {
    if (land.mask.data[i,j]==0) {
      Nd.plot[i-1,j]=NA
    }}}
plot(distance.all[,1],Nd.plot[,1],type='o',pch=16,cex=0.5,col=colors[1],yaxs=
'i',xaxs='i',
     xlab='Distance from "Coast" (km)',ylab='Nd',ylim=c(0,20))
for (j in 2:10){
  points(distance.all[,j],Nd.plot[,j],type="o",pch=16,cex=0.5,col=colors[j])
}

##plots of Dxs and 17Oxs from "coast"
#Dxs
plot(distance.all[,1],Dxs.all[,1,1],type="o",pch=16,cex=0.5,xaxs="i",
     xlab='Distance from "Coast" (km)',ylim=c(0,132),ylab="d-
excess",col=colors[1])
for (j in 2:10){

points(distance.all[,j],Dxs.all[,j,1],type="o",pch=16,cex=0.5,col=colors[j])
}
for (j in 1:10) {

points(distance.all[,j],Dxs_eddy.all[,j,1],type='o',pch=15,cex=0.5,col=colors
_eddy[j])
}
#17xs
plot(distance.all[,1],O17xs.all[,1,1],type="o",pch=16,cex=0.5,xaxs="i",
     xlab='Distance from "Coast" (km)',ylim=c(-100,10),ylab="17O-excess
(permeg)",col=colors[1])
for (j in 2:10){

points(distance.all[,j],O17xs.all[,j,1],type="o",pch=16,cex=0.5,col=colors[j]
)
}
for (j in 1:10) {

points(distance.all[,j],O17xs_eddy.all[,j,1],type='o',pch=15,cex=0.5,col=colo
rs_eddy[j])
}


####

#######average tracks together by distance from coast#########
####

###values from each track are averaged together by the distance from
#the coast rather than by the distance from the first data point
###note that these are approximate averages as the distance
#between points of different storm tracks are not equivalent
#ie, I'm assuming a distance of 32km between each point for each track.

#turn all of the inputs into one storm track:
#pre for pre-averaging
w.pre=as.matrix(Prw.data) #precipitatble water, kg/m^2
ntracks.pre=dim(w.pre)[2]
ntracks=1 #number of storm tracks
tracklength=dim(w.pre)[1] #length of storm track
Nd.NA=neg.toNA(as.matrix(Nd.data)) #convert negatives to NA to remove from
averaging
ts.pre=as.matrix(Temp_2m.data) #2m surface temperature (K)
rh.pre=as.matrix(rhum.data/100) #relative humidity (fraction)

#if land.mask=0, cut off the previous data point and add NA to the end
#cutting off previous allows for initiation just before land, consistent with
sections above
#adding NA at the end will allow for averaging of all the non-NAs for farther
out on track
for (j in 1:ntracks.pre) {
  for (i in 2:tracklength) {
    if (land.mask.data[i,j]==0) {
      w.pre[,j]=c(w.pre[-(i-1),j],NA)
      Nd.NA[,j]=c(Nd.NA[-(i-1),j],NA)
      ts.pre[,j]=c(ts.pre[-(i-1),j],NA)
      rh.pre[,j]=c(rh.pre[-(i-1),j],NA)
    }
  }
}

#now average these. this average is based on distance from coast!
w=rowMeans(w.pre,na.rm=T)
Nd=rowMeans(Nd.NA,na.rm=T)
ts=rowMeans(ts.pre,na.rm=T)
rh=rowMeans(rh.pre,na.rm=T)

#now copypasta from above section.

#constants for isotope value calculations:
#note that a18 != alpha18 (and same for D)
a18=1 + (-7.685 + (6.7123*10^3)/(ts) - (1.6664*10^6)/(ts)^2 +
(0.35041*10^9)/(ts)^3)/1000 #equilibrium fractionation 18O, v-l
```

```
aD=1 + ((1158.8*ts^3)/10^9 - (1620.1*ts^2)/10^6 + (794.84*ts)/10^3 - 161.04 +
(2.9992*10^9)/ts^3)/1000 #equilibrium fractionation, D v-l
alpha18=1 - (-7.685 + (6.7123*10^3)/(ts) - (1.6664*10^6)/(ts)^2 +
(0.35041*10^9)/(ts)^3)/1000 #equilibrium fractionation 18O, l-v
alphaD=1 - ((1158.8*ts^3)/10^9 - (1620.1*ts^2)/10^6 + (794.84*ts)/10^3 -
161.04 + (2.9992*10^9)/ts^3)/1000 #equilibrium fractionation D, l-v
alpha17=alpha18^(theta.eq) #Barkan and Luz, 2005
a17=a18^theta.eq

#build empty arrays to store data for d18O and dD and d17O for advection-
only/eddy-only cases
delta_a <- delta_et <- delta_inf<- delta_p <- delta_a_eddy <- delta_et_eddy
<- delta_inf_eddy <- delta_p_eddy <- array(dim=c(tracklength,length(e)))
delta_D_a <- delta_D_et <- delta_D_inf <- delta_D_p <- delta_D_a_eddy <-
delta_D_et_eddy <- delta_D_inf_eddy <- delta_D_p_eddy <-
array(dim=c(tracklength,length(e)))
delta_17_a <- delta_17_et <- delta_17_inf <- delta_17_p <- delta_17_a_eddy <-
delta_17_et_eddy <- delta_17_inf_eddy <- delta_17_p_eddy <-
array(dim=c(tracklength,length(e)))

delta_a[1,] = initial_delta_a #sets initial d18O atmospheric vapor for all
tracks and E/ETs
delta_p[1,] = initial_delta_p #sets intial d18O precip  #ntracksxlength(e)
delta_D_a[1,] = delta_a[1,]*8+10 #sets initial atmospheric vapor dD
delta_D_p[1,] = delta_p[1,]*8+10 #sets initial precip dD
delta_17_a[1,] = (exp(0.528*log(delta_a[1,]/1000+1)+0.000033)-1)*1000  #GMWL,
Luz and Barkan, 2010
delta_17_p[1,] = (exp(0.528*log(delta_p[1,]/1000+1)+0.000033)-1)*1000  #GMWL,
Luz and Barkan, 2010
#same for eddies:
delta_a_eddy[1,] = initial_delta_a_eddy #sets initial d18O atmospheric vapor
for all tracks and E/ETs
delta_p_eddy[1,] = initial_delta_p_eddy #sets intial d18O precip
#ntracksxlength(e)
delta_D_a_eddy[1,] = delta_a_eddy[1,]*8+10 #sets initial atmospheric vapor dD
delta_D_p_eddy[1,] = delta_p_eddy[1,]*8+10 #sets initial precip dD
delta_17_a_eddy[1,] = (exp(0.528*log(delta_a_eddy[1,]/1000+1)+0.000033)-
1)*1000  #GMWL, Luz and Barkan, 2010
delta_17_p_eddy[1,] = (exp(0.528*log(delta_p_eddy[1,]/1000+1)+0.000033)-
1)*1000  #GMWL, Luz and Barkan, 2010


###run the model!

#iterate over all points along each track for each E/ET ratio
for (h in 1:length(e)) {
  for (i in 2:tracklength) {
    #calculate residual soil moisture for isotope_e function
    residual=1-(Nd[i]/(Nd[i]+1))*e[h]/(1-(Nd[i]/(Nd[i]+1)*t[h]))
    #if everything evaporates, set isotope value of evaporation to isotope
value of precip
    if(residual == 0){
      delta_et[i,h] <- delta_p[i-1,h]
      delta_D_et[i,h] <- delta_D_p[i-1,h]
      delta_17_et[i,h] = delta_17_p[i-1,h]
      delta_et_eddy[i,h]=delta_p_eddy[i-1,h]
      delta_D_et_eddy[i,h] <- delta_D_p_eddy[i-1,h]
      delta_17_et_eddy[i,h] = delta_17_p_eddy[i-1,h]

      #otherwise, first calculate the combined equilibrium/kinetic
fractionation using Craig-Gordon function
    } else {
      eps_e <- craig_gordon(t[h], rh[i], delta_p[i-1,h], ts[i])
      eps_e_D <- craig_gordon(t[h], rh[i], delta_D_p[i-1,h], ts[i], d18 = F)
      eps_e_17 = craig_gordon17(t[h],rh[i],delta_17_p[i-1,h],ts[i])
      eps_e_eddy <- craig_gordon(t[h], rh[i], delta_p_eddy[i-1,h], ts[i])
      eps_e_D_eddy <- craig_gordon(t[h], rh[i], delta_D_p_eddy[i-1,h], ts[i],
d18 = F)
      eps_e_17_eddy = craig_gordon17(t[h],rh[i],delta_17_p_eddy[i-1,h],ts[i])
      #then calculate integrated ET value from location
      delta_et[i,h] <- t[h]*delta_p[i-1,h] +
e[h]*isotope_e(residual,delta_p[i-1,h], eps_e)
      delta_D_et[i,h] <- t[h]*delta_D_p[i-1,h] +
e[h]*isotope_e(residual,delta_D_p[i-1,h], eps_e_D)
      delta_17_et[i,h] = t[h]*delta_17_p[i-
1,h]+e[h]*isotope_e(residual,delta_17_p[i-1,h],eps_e_17)
      delta_et_eddy[i,h] <- t[h]*delta_p_eddy[i-1,h] +
e[h]*isotope_e(residual,delta_p_eddy[i-1,h], eps_e_eddy)
      delta_D_et_eddy[i,h] <- t[h]*delta_D_p_eddy[i-1,h] +
e[h]*isotope_e(residual,delta_D_p_eddy[i-1,h], eps_e_D_eddy)
      delta_17_et_eddy[i,h] = t[h]*delta_17_p_eddy[i-
1,h]+e[h]*isotope_e(residual,delta_17_p_eddy[i-1,h],eps_e_17_eddy)
    }


    ###there's an issue with some of the values after the initial point being
NAs
    #this loop will make it so all of the NAs stay NAs *except* the last NA
before
    #hitting land. the first ones stay NAs so that we're only calculating
over land.
    #note: still necessary to keep the original initials above in the event
of no NAs
    #only where rowMeans==0 will they all be above water==>this track will be
above water
    if (rowMeans(land.mask.data)[i]==0) {
      delta_p[i,]=initial_delta_p
      delta_p[i-1,]=NA
      delta_a[i,]=initial_delta_a
      delta_a[i-1,]=NA
      delta_D_p[i,]=delta_p[i,]*8+10 #assume initial points are on MWL
      delta_D_p[i-1,]=NA
      delta_D_a[i,]=delta_a[i,]*8+10
      delta_D_a[i-1,]=NA
      delta_17_p[i,]=(exp(0.528*log(delta_p[i,]/1000+1)+0.000033)-1)*1000
#GMWL, Luz and Barkan, 2010
      delta_17_p[i-1,]=NA
      delta_17_a[i,]=(exp(0.528*log(delta_a[i,]/1000+1)+0.000033)-1)*1000
#GMWL, Luz and Barkan, 2010
      delta_17_a[i-1,]=NA
      #repeat for eddies (copypasta)
      delta_p_eddy[i,]=initial_delta_p_eddy
      delta_p_eddy[i-1,]=NA
      delta_a_eddy[i,]=initial_delta_a_eddy
      delta_a_eddy[i-1,]=NA
      delta_D_p_eddy[i,]=delta_p_eddy[i,]*8+10 #assume initial points are on
MWL
      delta_D_p_eddy[i-1,]=NA
      delta_D_a_eddy[i,]=delta_a_eddy[i,]*8+10
      delta_D_a_eddy[i-1,]=NA
      delta_17_p_eddy[i,]=(exp(0.528*log(delta_p_eddy[i,]/1000+1)+0.000033)-
1)*1000  #GMWL, Luz and Barkan, 2010
      delta_17_p_eddy[i-1,]=NA
      delta_17_a_eddy[i,]=(exp(0.528*log(delta_a_eddy[i,]/1000+1)+0.000033)-
1)*1000  #GMWL, Luz and Barkan, 2010
      delta_17_a_eddy[i-1,]=NA
    } else {
      x=32 #km, assumed distance between points
      dx=x #meters, but the units don't matter. slightly >32km
      dw=w[i]-w[i-1]
      dw=ifelse(dw>0,0,dw) #gets rid of positive gradients (b/c we don't like
explosions here)
      #Hendricks model (advection only)
      delta_inf[i,h] <- (Nd[i]*delta_et[i,h]-(1+Nd[i])*(a18[i]-
1)*1000)/(a18[i]+Nd[i]*a18[i]-1)
      delta_a[i,h]=(delta_a[i-1,h]-delta_inf[i,h])*exp((a18[i]+a18[i]*Nd[i]-
1)*(x/w[i])*(dw/dx))+delta_inf[i,h]
      delta_p[i,h] <-delta_a[i,h]+(a18[i]-1)*1000

      delta_D_inf[i,h] <- (Nd[i]*delta_D_et[i,h]-(1+Nd[i])*(aD[i]-
1)*1000)/(aD[i]+Nd[i]*aD[i]-1)
      delta_D_a[i,h]=(delta_D_a[i-1,h]-
delta_D_inf[i,h])*exp((aD[i]+aD[i]*Nd[i]-
1)*(x/w[i])*(dw/dx))+delta_D_inf[i,h]
      delta_D_p[i,h] <-delta_D_a[i,h]+(aD[i]-1)*1000

      delta_17_inf[i,h] <- (Nd[i]*delta_17_et[i,h]-(1+Nd[i])*(a17[i]-
1)*1000)/(a17[i]+Nd[i]*a17[i]-1)
      delta_17_a[i,h]=(delta_17_a[i-1,h]-
delta_17_inf[i,h])*exp((a17[i]+a17[i]*Nd[i]-
1)*(x/w[i])*(dw/dx))+delta_17_inf[i,h]
      delta_17_p[i,h] <-delta_17_a[i,h]+(a17[i]-1)*1000

      #Hendricks model (eddy diffusion only)
      delta_inf_eddy[i,h] <- (Nd[i]*delta_et_eddy[i,h]-(1+Nd[i])*(a18[i]-
1)*1000)/(a18[i]+Nd[i]*a18[i]-1)
      delta_a_eddy[i,h]=(delta_a_eddy[i-1,h]-
delta_inf_eddy[i,h])*exp((sqrt(a18[i]+a18[i]*Nd[i])-
1)*(x/w[i])*(dw/dx))+delta_inf_eddy[i,h]
      delta_p_eddy[i,h] <-delta_a_eddy[i,h]+(a18[i]-1)*1000

      delta_D_inf_eddy[i,h] <- (Nd[i]*delta_D_et_eddy[i,h]-(1+Nd[i])*(aD[i]-
1)*1000)/(aD[i]+Nd[i]*aD[i]-1)
      delta_D_a_eddy[i,h]=(delta_D_a_eddy[i-1,h]-
delta_D_inf_eddy[i,h])*exp((sqrt(aD[i]+aD[i]*Nd[i])-
1)*(x/w[i])*(dw/dx))+delta_D_inf_eddy[i,h]
      delta_D_p_eddy[i,h] <-delta_D_a_eddy[i,h]+(aD[i]-1)*1000

      delta_17_inf_eddy[i,h] <- (Nd[i]*delta_17_et_eddy[i,h]-
(1+Nd[i])*(a17[i]-1)*1000)/(a17[i]+Nd[i]*a17[i]-1)
      delta_17_a_eddy[i,h]=(delta_17_a_eddy[i-1,h]-
delta_17_inf_eddy[i,h])*exp((sqrt(a17[i]+a17[i]*Nd[i])-
1)*(x/w[i])*(dw/dx))+delta_17_inf_eddy[i,h]
      delta_17_p_eddy[i,h] <-delta_17_a_eddy[i,h]+(a17[i]-1)*1000
    }
  }
}
delta_p.meanfromcoast=delta_p
delta_D_p.meanfromcoast=delta_D_p
delta_17_p.meanfromcoast=delta_17_p
delta_p_eddy.meanfromcoast=delta_p_eddy
delta_D_p_eddy.meanfromcoast=delta_D_p_eddy
delta_17_p_eddy.meanfromcoast=delta_17_p_eddy
distance.fake = seq(0,by=32,length=tracklength)

###calculate D-excess and 17O-excess:
Dxs.meanfromcoast=delta_D_p.meanfromcoast-8*delta_p.meanfromcoast #in permil
O17xs.meanfromcoast=(log(delta_17_p.meanfromcoast/1000+1)-
0.528*log(delta_p.meanfromcoast/1000+1))*10^6 #in permeg
Dxs_eddy.meanfromcoast=delta_D_p_eddy.meanfromcoast-
8*delta_p_eddy.meanfromcoast #in permil
O17xs_eddy.meanfromcoast=(log(delta_17_p_eddy.meanfromcoast/1000+1)-
0.528*log(delta_p_eddy.meanfromcoast/1000+1))*10^6 #in permeg


##########model plots###########

library(fields)
ntracks=10
colors=tim.colors(ntracks)
colors_eddy=paste('gray',seq(5,86,length=10))

###plot Nd###
Nd=rowMeans(Nd.NA,na.rm=T)
pdf('Damkohler of each track.pdf',width=5,height=3)
plot(distance.all[,1],Nd.data[,1],pch=16,type='o',cex=0.5,xaxs="i",
     col=colors[1],xlab='Distance from "Coast" (km)',ylim=c(-5,20),
     ylab='Nd',main='Damkohler of each track')
for (j in 2:ntracks){
  points(distance.all[,j],Nd.data[,j],type="o",pch=16,cex=0.5,col=colors[j])
}
points(distance.fake,Nd,pch=16,type='o',cex=0.9,col='black')
legend('topleft',legend='average
without\nnegatives',pch=16,col='black',cex=0.8)
dev.off()

###
###plot model-output d18O###

pdf('Model d18O with eddies.pdf',width=7,height=10)
par(mfrow=c(2,1))
#all evaporation
plot(distance.all[,1],delta_p.all[,1,1],type="o",pch=16,cex=0.5,xaxs="i",
     ylim=c(-35,0),col=colors[1],xlab='Distance from "Coast" (km)',
     ylab=expression(paste(delta^18,O[p],' (\u2030)')),main='All
Evaporation')
for (j in 2:ntracks){

points(distance.all[,j],delta_p.all[,j,1],type="o",pch=16,cex=0.5,col=colors[
j])
}
for(j in 1:ntracks) {

points(distance.all[,j],delta_p_eddy.all[,j,1],type='o',pch=15,cex=0.5,col=co
lors_eddy[j])
}
points(distance.fake,delta_p.meanfromcoast[,1],type='o',pch=16,col='black')
points(distance.fake,delta_p_eddy.meanfromcoast[,1],type='o',pch=15,col='gray
50')
legend('topright',c('advection only','eddies
only'),col=c('black','gray50'),pch=c(16,15))

#all tranpsiration
plot(distance.all[,1],delta_p.all[,1,11],type="o",pch=16,cex=0.5,xaxs="i",
     ylim=c(-35,0),col=colors[1],xlab='Distance from "Coast" (km)',
     ylab=expression(paste(delta^18,O[p],' (\u2030)')),main='All
Transpiration')
for (j in 2:ntracks){

points(distance.all[,j],delta_p.all[,j,11],type="o",pch=16,cex=0.5,col=colors
[j])
}
for (j in 1:ntracks) {

points(distance.all[,j],delta_p_eddy.all[,j,11],type='o',pch=15,cex=0.5,col=c
olors_eddy[j])
```

```
}
points(distance.fake,delta_p_eddy.meanfromcoast[,11],type='o',pch=15,col='gra
y50')
points(distance.fake,delta_p.meanfromcoast[,11],type='o',pch=16,col='black')
legend('topright',c('advection only','eddies
only'),col=c('black','gray50'),pch=c(16,15))
dev.off()

###
###plot model-output Dxs###

pdf('Model Dxs with eddies.pdf',width=7,height=10)
par(mfrow=c(2,1))
#all evaporation
plot(distance.all[,1],Dxs.all[,1,1],type="o",pch=16,cex=0.5,xaxs="i",
     ylim=c(10,130),col=colors[1],xlab='Distance from "Coast" (km)',
     ylab='d-excess (\u2030)',main='All Evaporation')
for (j in 2:ntracks){

points(distance.all[,j],Dxs.all[,j,1],type="o",pch=16,cex=0.5,col=colors[j])
}
for (j in 1:ntracks) {

points(distance.all[,j],Dxs_eddy.all[,j,1],type='o',pch=15,cex=0.5,col=colors
_eddy[j])
}
points(distance.fake,Dxs_eddy.meanfromcoast[,1],type='o',pch=15,col='gray50')
points(distance.fake,Dxs.meanfromcoast[,1],type='o',pch=16,col='black')
legend('topleft',c('advection only','eddies
only'),col=c('black','gray50'),pch=c(16,15))

#all transpiration
plot(distance.all[,1],Dxs.all[,1,11],type="o",pch=16,cex=0.5,xaxs="i",
     ylim=c(-50,80),col=colors[1],xlab='Distance from "Coast" (km)',
     ylab='d-excess (\u2030)',main='All Transpiration')
for (j in 2:ntracks){

points(distance.all[,j],Dxs.all[,j,11],type="o",pch=16,cex=0.5,col=colors[j])
}
for (j in 1:ntracks) {

points(distance.all[,j],Dxs_eddy.all[,j,11],type='o',pch=15,cex=0.5,col=color
s_eddy[j])
}
points(distance.fake,Dxs_eddy.meanfromcoast[,11],type='o',pch=15,col='gray50'
)
points(distance.fake,Dxs.meanfromcoast[,11],type='o',pch=16,col='black')
legend('topleft',c('advection only','eddies
only'),col=c('black','gray50'),pch=c(16,15))
dev.off()

###
###plot model-output 17Oxs###

pdf('Model 17Oxs with eddies.pdf',width=7,height=15)
par(mfrow=c(3,1))
#all evaporation
plot(distance.all[,1],O17xs.all[,1,1],type="o",pch=16,cex=0.5,xaxs="i",
     ylim=c(-150,50),col=colors[1],xlab='Distance from "Coast" (km)',
     ylab='17O-excess (permeg)',main='All Evaporation')
for (j in 2:ntracks){

points(distance.all[,j],O17xs.all[,j,1],type="o",pch=16,cex=0.5,col=colors[j]
)
```

```
}
for (j in 1:ntracks){

points(distance.all[,j],O17xs_eddy.all[,j,1],type="o",pch=15,cex=0.5,col=colo
rs_eddy[j])
}
points(distance.fake,O17xs_eddy.meanfromcoast[,1],type='o',pch=15,col='gray50
')
points(distance.fake,O17xs.meanfromcoast[,1],type='o',pch=16,col='black')
legend('bottomleft',c('advection only','eddies
only'),col=c('black','gray50'),pch=c(16,15))

#70% transpiration
plot(distance.all[,1],O17xs.all[,1,8],type="o",pch=16,cex=0.5,xaxs="i",
     ylim=c(-300,50),col=colors[1],xlab='Distance from "Coast" (km)',
     ylab='17O-excess (permeg)',main='70% Transpiration')
for (j in 2:ntracks){

points(distance.all[,j],O17xs.all[,j,8],type="o",pch=16,cex=0.5,col=colors[j]
)
}
for (j in 1:ntracks){

points(distance.all[,j],O17xs_eddy.all[,j,8],type="o",pch=15,cex=0.5,col=colo
rs_eddy[j])
}
points(distance.fake,O17xs_eddy.meanfromcoast[,8],type='o',pch=15,col='gray50
')
points(distance.fake,O17xs.meanfromcoast[,8],type='o',pch=16,col='black')
legend('bottomleft',c('advection only','eddies
only'),col=c('black','gray50'),pch=c(16,15))

#all transpiration
plot(distance.all[,1],O17xs.all[,1,11],type="o",pch=16,cex=0.5,xaxs="i",
     ylim=c(-250,200),col=colors[1],xlab='Distance from "Coast" (km)',
     ylab='17O-excess (permeg)',main='All Transpiration')
for (j in 2:ntracks){

points(distance.all[,j],O17xs.all[,j,11],type="o",pch=16,cex=0.5,col=colors[j
])
}
for (j in 1:ntracks){

points(distance.all[,j],O17xs_eddy.all[,j,11],type="o",pch=15,cex=0.5,col=col
ors_eddy[j])
}
points(distance.fake,O17xs_eddy.meanfromcoast[,11],type='o',pch=15,col='gray5
0')
points(distance.fake,O17xs.meanfromcoast[,11],type='o',pch=16,col='black')
legend('bottomleft',c('advection only','eddies
only'),col=c('black','gray50'),pch=c(16,15))
dev.off()


####

#######data time!########
####

moderndata=read.csv('modern water d18O dD - no USGS.csv',header=T)
head(moderndata)
library(dplyr)
library(geosphere)
```

```
#a few of the data points have NAs for lat/lon. take out those rows:
moderndata2=filter(moderndata,is.na(latitude)==F,is.na(longitude)==F)

##add a column for distance from "coast"
#where "coast" means first I&T (1991) point
#calculate three coastal distances then take the min of those as dist from
coast
lati=moderndata$latitude[1] #39.43807
loni=moderndata$longitude[1] #-123.7982
lati2=40.002656
loni2=-124.024106
lati3=38.962633
loni3=-123.721038

latitudes=moderndata2$latitude
longitudes=moderndata2$longitude
distmin <- dist1 <- dist2 <- dist3 <- numeric()
for (i in 1:length(latitudes)){
  dist1[i]=distVincentyEllipsoid(c(longitudes[i],latitudes[i]),c(loni,lati))

dist2[i]=distVincentyEllipsoid(c(longitudes[i],latitudes[i]),c(loni2,lati2))

dist3[i]=distVincentyEllipsoid(c(longitudes[i],latitudes[i]),c(loni3,lati3))
  distmin[i]=min(dist1[i],dist2[i],dist3[i])/1000 #distance in km
}

moderndata3=cbind(moderndata2,distmin)
head(moderndata3)

###add another column for Dxs
moderndata.Dxs=mutate(moderndata3,Dxs=dD-8*d18O)
head(moderndata.Dxs)

##separate data by sample type
#rain!
Dxs_rain=filter(moderndata.Dxs,water.type=='rain')$Dxs
d18O_rain=filter(moderndata3,water.type=='rain')$d18O
dist_rain=filter(moderndata3,water.type=='rain')$distmin
lat_rain=filter(moderndata3,water.type=='rain')$lat
lon_rain=filter(moderndata3,water.type=='rain')$lon
#snow!
Dxs_snow=filter(moderndata.Dxs,water.type=='snow')$Dxs
d18O_snow=filter(moderndata3,water.type=='snow')$d18O
dist_snow=filter(moderndata3,water.type=='snow')$distmin
lat_snow=filter(moderndata3,water.type=='snow')$lat
lon_snow=filter(moderndata3,water.type=='snow')$lon
#rivers and creeks!
Dxs_river=filter(moderndata.Dxs,water.type=='river')$Dxs
d18O_river=filter(moderndata3,water.type=='river')$d18O
dist_river=filter(moderndata3,water.type=='river')$distmin
lat_river=filter(moderndata3,water.type=='river')$lat
lon_river=filter(moderndata3,water.type=='river')$lon
#groundwater!
Dxs_gw=filter(moderndata.Dxs,water.type=='GW')$Dxs
d18O_gw=filter(moderndata3,water.type=='GW')$d18O
dist_gw=filter(moderndata3,water.type=='GW')$distmin
lat_gw=filter(moderndata3,water.type=='GW')$lat
lon_gw=filter(moderndata3,water.type=='GW')$lon
#lakes!
Dxs_lake=filter(moderndata.Dxs,water.type=='lake')$Dxs
d18O_lake=filter(moderndata3,water.type=='lake')$d18O
dist_lake=filter(moderndata3,water.type=='lake')$distmin
lat_lake=filter(moderndata3,water.type=='lake')$lat
lon_lake=filter(moderndata3,water.type=='lake')$lon
```

```
####

######plots with data#########
####


###make map of where the data are; color points by isotopic value
library(maps)
library(oce)
lim=c(-22,0)
ncol=100
col=oceColorsJet(ncol)
# Draw colorbar for d18O
pdf('map of data.pdf',width=10, height=7)
par(mfrow=c(1,2))
drawPalette(zlim=lim,col=col,
            zlab=expression(paste(delta^18,"O"[precip])),
            las=1,cex.lab=1.2) # adjust palette position with mai


par(mar=c(2.5,2.5,1,1),mgp=c(2,0.7,0)) # tighten margins
map('state',xlim=c(min(moderndata2$lon)-1,max(moderndata2$lon)),
    ylim=c(min(moderndata2$lat)-1,max(moderndata2$lat)))
for (i in 1:10) {  #add storm tracks
  points(lon.data[,i],lat.data[,i],col="blue",type="o",pch=1,cex=0.5)
}
points(lon_rain,lat_rain,pch=21,cex=0.7,
       bg=col[rescale(x=d18O_rain,xlow=lim[1],
                      xhigh=lim[2],rlow=1,rhigh=ncol)])
points(lon_snow,lat_snow,pch=22,cex=0.7,
       bg=col[rescale(x=d18O_snow,xlow=lim[1],
                      xhigh=lim[2],rlow=1,rhigh=ncol)])
points(lon_river,lat_river,pch=23,cex=0.7,
       bg=col[rescale(x=d18O_river,xlow=lim[1],
                      xhigh=lim[2],rlow=1,rhigh=ncol)])
points(lon_gw,lat_gw,pch=24,cex=0.7,
       bg=col[rescale(x=d18O_gw,xlow=lim[1],
                      xhigh=lim[2],rlow=1,rhigh=ncol)])
points(lon_lake,lat_lake,pch=25,cex=0.7,
       bg=col[rescale(x=d18O_lake,xlow=lim[1],
                      xhigh=lim[2],rlow=1,rhigh=ncol)])
legend('bottomleft',pch=c(21,22,23,24,25),pt.bg='cyan',cex=0.7,bty='n',
       legend=c('rain','snow','rivers/creeks','groundwater','lakes'))
dev.off()


###
###plot model-output d18O with data###

pdf('d18O with eddies.pdf',width=7,height=15)
par(mfrow=c(3,1))
#all evaporation
plot(distance.all[,1],delta_p.all[,1,1],type="o",pch=16,cex=0.5,xaxs="i",
     ylim=c(-35,0),col=colors[1],xlab='Distance from "Coast" (km)',
     ylab=expression(paste(delta^18,O[p],' (\u2030)')),main='All
Evaporation')
for (j in 2:ntracks){

points(distance.all[,j],delta_p.all[,j,1],type="o",pch=16,cex=0.5,col=colors[
j])
}
for(j in 1:ntracks) {
```

```
points(distance.all[,j],delta_p_eddy.all[,j,1],type='o',pch=15,cex=0.5,col=co
lors_eddy[j])
}
points(distance.fake,delta_p.meanfromcoast[,1],type='o',pch=16,col='black')
points(distance.fake,delta_p_eddy.meanfromcoast[,1],type='o',pch=15,col='gray
50')
points(dist_rain,d18O_rain,pch=21,cex=0.8,col='black',bg='blue')
points(dist_snow,d18O_snow,cex=0.8,pch=22,col='black',bg='blue')
points(dist_gw,d18O_gw,cex=0.8,pch=23,col='black',bg='blue')
points(dist_river,d18O_river,cex=0.8,pch=24,col='black',bg='blue')
points(dist_lake,d18O_lake,cex=0.8,pch=24,col='black',bg='blue')
legend('bottomleft',c('advection','eddies','rain','snow','groundwater','river
s/streams','lakes'),

cex=0.8,pch=c(16,15,21,22,23,24,25),col=c('black','gray50',rep('black',5)),pt
.bg='blue')

#70% tranpsiration
plot(distance.all[,1],delta_p.all[,1,8],type="o",pch=16,cex=0.5,xaxs="i",
     ylim=c(-35,0),col=colors[1],xlab='Distance from "Coast" (km)',
     ylab=expression(paste(delta^18,O[p],' (\u2030)')),main='70%
Transpiration')
for (j in 2:ntracks){

points(distance.all[,j],delta_p.all[,j,8],type="o",pch=16,cex=0.5,col=colors[
j])
}
for (j in 1:ntracks) {

points(distance.all[,j],delta_p_eddy.all[,j,8],type='o',pch=15,cex=0.5,col=co
lors_eddy[j])
}
points(distance.fake,delta_p_eddy.meanfromcoast[,8],type='o',pch=15,col='gray
50')
points(distance.fake,delta_p.meanfromcoast[,8],type='o',pch=16,col='black')
points(dist_rain,d18O_rain,pch=21,cex=0.8,col='black',bg='blue')
points(dist_snow,d18O_snow,cex=0.8,pch=22,col='black',bg='blue')
points(dist_gw,d18O_gw,cex=0.8,pch=23,col='black',bg='blue')
points(dist_river,d18O_river,cex=0.8,pch=24,col='black',bg='blue')
points(dist_lake,d18O_lake,cex=0.8,pch=24,col='black',bg='blue')
legend('bottomleft',c('advection','eddies','rain','snow','groundwater','river
s/streams','lakes'),

cex=0.8,pch=c(16,15,21,22,23,24,25),col=c('black','gray50',rep('black',5)),pt
.bg='blue')

#all tranpsiration
plot(distance.all[,1],delta_p.all[,1,11],type="o",pch=16,cex=0.5,xaxs="i",
     ylim=c(-35,0),col=colors[1],xlab='Distance from "Coast" (km)',
     ylab=expression(paste(delta^18,O[p],' (\u2030)')),main='All
Transpiration')
for (j in 2:ntracks){

points(distance.all[,j],delta_p.all[,j,11],type="o",pch=16,cex=0.5,col=colors
[j])
}
for (j in 1:ntracks) {

points(distance.all[,j],delta_p_eddy.all[,j,11],type='o',pch=15,cex=0.5,col=c
olors_eddy[j])
}
points(distance.fake,delta_p_eddy.meanfromcoast[,11],type='o',pch=15,col='gra
y50')
```

```
points(distance.fake,delta_p.meanfromcoast[,11],type='o',pch=16,col='black')
points(dist_rain,d18O_rain,pch=21,cex=0.8,col='black',bg='blue')
points(dist_snow,d18O_snow,cex=0.8,pch=22,col='black',bg='blue')
points(dist_gw,d18O_gw,cex=0.8,pch=23,col='black',bg='blue')
points(dist_river,d18O_river,cex=0.8,pch=24,col='black',bg='blue')
points(dist_lake,d18O_lake,cex=0.8,pch=24,col='black',bg='blue')
legend('bottomleft',c('advection','eddies','rain','snow','groundwater','river
s/streams','lakes'),

cex=0.8,pch=c(16,15,21,22,23,24,25),col=c('black','gray50',rep('black',5)),pt
.bg='blue')
dev.off()

###
###plot model-output Dxs with data###

pdf('Dxs with eddies.pdf',width=7,height=15)
par(mfrow=c(3,1))
#all evaporation
plot(distance.all[,1],Dxs.all[,1,1],type="o",pch=16,cex=0.5,xaxs="i",
     ylim=c(-70,130),xlim=c(0,1130),col=colors[1],xlab='Distance from "Coast"
(km)',
     ylab='d-excess (\u2030)',main='All Evaporation')
for (j in 2:ntracks){

points(distance.all[,j],Dxs.all[,j,1],type="o",pch=16,cex=0.5,col=colors[j])
}
for (j in 1:ntracks) {

points(distance.all[,j],Dxs_eddy.all[,j,1],type='o',pch=15,cex=0.5,col=colors
_eddy[j])
}
points(distance.fake,Dxs_eddy.meanfromcoast[,1],type='o',pch=15,col='gray50')
points(distance.fake,Dxs.meanfromcoast[,1],type='o',pch=16,col='black')
points(dist_rain,Dxs_rain,pch=21,cex=0.8,col='black',bg='blue')
points(dist_snow,Dxs_snow,cex=0.8,pch=22,col='black',bg='blue')
points(dist_gw,Dxs_gw,cex=0.8,pch=23,col='black',bg='blue')
points(dist_river,Dxs_river,cex=0.8,pch=24,col='black',bg='blue')
points(dist_lake,Dxs_lake,cex=0.8,pch=24,col='black',bg='blue')
legend('topright',c('advection','eddies','rain','snow','groundwater','rivers/
streams','lakes'),

cex=0.8,pch=c(16,15,21,22,23,24,25),col=c('black','gray50',rep('black',5)),pt
.bg='blue')

#70% transpiration
plot(distance.all[,1],Dxs.all[,1,8],type="o",pch=16,cex=0.5,xaxs="i",
     ylim=c(-70,130),xlim=c(0,1130),col=colors[1],xlab='Distance from "Coast"
(km)',
     ylab='d-excess (\u2030)',main='70% Transpiration')
for (j in 2:ntracks){

points(distance.all[,j],Dxs.all[,j,8],type="o",pch=16,cex=0.5,col=colors[j])
}
for (j in 1:ntracks) {

points(distance.all[,j],Dxs_eddy.all[,j,8],type='o',pch=15,cex=0.5,col=colors
_eddy[j])
}
points(distance.fake,Dxs_eddy.meanfromcoast[,8],type='o',pch=15,col='gray50')
points(distance.fake,Dxs.meanfromcoast[,8],type='o',pch=16,col='black')
points(dist_rain,Dxs_rain,pch=21,cex=0.8,col='black',bg='blue')
points(dist_snow,Dxs_snow,cex=0.8,pch=22,col='black',bg='blue')
points(dist_gw,Dxs_gw,cex=0.8,pch=23,col='black',bg='blue')
points(dist_river,Dxs_river,cex=0.8,pch=24,col='black',bg='blue')
points(dist_lake,Dxs_lake,cex=0.8,pch=24,col='black',bg='blue')
legend('topright',c('advection','eddies','rain','snow','groundwater','rivers/
streams','lakes'),

cex=0.8,pch=c(16,15,21,22,23,24,25),col=c('black','gray50',rep('black',5)),pt
.bg='blue')

#all transpiration
plot(distance.all[,1],Dxs.all[,1,11],type="o",pch=16,cex=0.5,xaxs="i",
     ylim=c(-70,130),xlim=c(0,1130),col=colors[1],xlab='Distance from "Coast"
(km)',
     ylab='d-excess (\u2030)',main='All Transpiration')
for (j in 2:ntracks){

points(distance.all[,j],Dxs.all[,j,11],type="o",pch=16,cex=0.5,col=colors[j])
}
for (j in 1:ntracks) {

points(distance.all[,j],Dxs_eddy.all[,j,11],type='o',pch=15,cex=0.5,col=color
s_eddy[j])
}
points(distance.fake,Dxs_eddy.meanfromcoast[,11],type='o',pch=15,col='gray50'
)
points(distance.fake,Dxs.meanfromcoast[,11],type='o',pch=16,col='black')
points(dist_rain,Dxs_rain,pch=21,cex=0.8,col='black',bg='blue')
points(dist_snow,Dxs_snow,cex=0.8,pch=22,col='black',bg='blue')
points(dist_gw,Dxs_gw,cex=0.8,pch=23,col='black',bg='blue')
points(dist_river,Dxs_river,cex=0.8,pch=24,col='black',bg='blue')
points(dist_lake,Dxs_lake,cex=0.8,pch=24,col='black',bg='blue')
legend('topright',c('advection','eddies','rain','snow','groundwater','rivers/
streams','lakes'),

cex=0.8,pch=c(16,15,21,22,23,24,25),col=c('black','gray50',rep('black',5)),pt
.bg='blue')
dev.off()


####

######Ingraham and Taylor, 1991 plots#######

library(geosphere)
library(dplyr)

IandTdata=read.csv("Ingraham and Taylor 1991 Traverse II.csv")
head(IandTdata)
#eliminate NA lat/lons
IandTdata2=filter(IandTdata,is.na(latitude)==F,is.na(longitude)==F)
###add another column for Dxs
IandTdata.Dxs=mutate(IandTdata2,Dxs=dD-8*d18O)

##separate data by sample type
#rain!
IT_Dxs_rain=filter(IandTdata.Dxs,water.type=='rain')$Dxs
IT_d18O_rain=filter(IandTdata.Dxs,water.type=='rain')$d18O
IT_dist_rain=filter(IandTdata.Dxs,water.type=='rain')$dist
IT_lat_rain=filter(IandTdata.Dxs,water.type=='rain')$lat
IT_lon_rain=filter(IandTdata.Dxs,water.type=='rain')$lon
#snow!
IT_Dxs_snow=filter(IandTdata.Dxs,water.type=='snow')$Dxs
IT_d18O_snow=filter(IandTdata.Dxs,water.type=='snow')$d18O
IT_dist_snow=filter(IandTdata.Dxs,water.type=='snow')$dist
IT_lat_snow=filter(IandTdata.Dxs,water.type=='snow')$lat
```

```
IT_lon_snow=filter(IandTdata.Dxs,water.type=='snow')$lon
#groundwater!
IT_Dxs_gw=filter(IandTdata.Dxs,water.type=='spring.GW')$Dxs
IT_d18O_gw=filter(IandTdata.Dxs,water.type=='spring.GW')$d18O
IT_dist_gw=filter(IandTdata.Dxs,water.type=='spring.GW')$dist
IT_lat_gw=filter(IandTdata.Dxs,water.type=='spring.GW')$lat
IT_lon_gw=filter(IandTdata.Dxs,water.type=='spring.GW')$lon

###
###plot model-output d18O with I&T data###

pdf('model d18O with I&T data.pdf',width=5,height=10)
par(mfrow=c(3,1))
#all evaporation
plot(distance.fake,delta_p.meanfromcoast[,1],type='o',pch=16,col='black',
     xaxs="i", ylim=c(-35,0),xlab='Distance from "Coast" (km)',
     ylab=expression(paste(delta^18,O[p],' (\u2030)')),main='All
Evaporation')
points(distance.fake,delta_p_eddy.meanfromcoast[,1],type='o',pch=15,col='gray
50')
points(IT_dist_rain,IT_d18O_rain,pch=21,cex=0.8,col='black',bg='blue')
points(IT_dist_snow,IT_d18O_snow,cex=0.8,pch=22,col='black',bg='light blue')
points(IT_dist_gw,IT_d18O_gw,cex=0.8,pch=23,col='black',bg='cyan')
legend('bottomleft',c('advection','eddies','rain','snow','groundwater'),
       cex=0.8,pch=c(16,15,21,22,23),col=c('black','gray50',rep('black',3)),
       pt.bg=c(NA,NA,'blue','light blue','cyan'))

#70% tranpsiration
plot(distance.fake,delta_p.meanfromcoast[,8],type='o',pch=16,col='black',
     xaxs="i", ylim=c(-35,0),xlab='Distance from "Coast" (km)',
     ylab=expression(paste(delta^18,O[p],' (\u2030)')),main='70%
Transpiration')
points(distance.fake,delta_p_eddy.meanfromcoast[,8],type='o',pch=15,col='gray
50')
points(IT_dist_rain,IT_d18O_rain,pch=21,cex=0.8,col='black',bg='blue')
points(IT_dist_snow,IT_d18O_snow,cex=0.8,pch=22,col='black',bg='light blue')
points(IT_dist_gw,IT_d18O_gw,cex=0.8,pch=23,col='black',bg='cyan')
legend('bottomleft',c('advection','eddies','rain','snow','groundwater'),
       cex=0.8,pch=c(16,15,21,22,23),col=c('black','gray50',rep('black',3)),
       pt.bg=c(NA,NA,'blue','light blue','cyan'))

#all tranpsiration
plot(distance.fake,delta_p.meanfromcoast[,11],type='o',pch=16,col='black',
      xaxs="i", ylim=c(-35,0),xlab='Distance from "Coast" (km)',
     ylab=expression(paste(delta^18,O[p],' (\u2030)')),main='All
Transpiration')
points(distance.fake,delta_p_eddy.meanfromcoast[,11],type='o',pch=15,col='gra
y50')
points(distance.fake,delta_p.meanfromcoast[,11],type='o',pch=16,col='black')
points(IT_dist_rain,IT_d18O_rain,pch=21,cex=0.8,col='black',bg='blue')
points(IT_dist_snow,IT_d18O_snow,cex=0.8,pch=22,col='black',bg='light blue')
points(IT_dist_gw,IT_d18O_gw,cex=0.8,pch=23,col='black',bg='cyan')
legend('bottomleft',c('advection','eddies','rain','snow','groundwater'),
       cex=0.8,pch=c(16,15,21,22,23),col=c('black','gray50',rep('black',3)),
       pt.bg=c(NA,NA,'blue','light blue','cyan'))
dev.off()


###
###plot model-output Dxs with I&T data###

pdf('model Dxs with I&T data.pdf',width=5,height=10)
par(mfrow=c(3,1))
#all evaporation
```

```
plot(distance.fake,Dxs.meanfromcoast[,1],type='o',pch=16,col='black',
     xaxs="i", ylim=c(-5,130),xlab='Distance from "Coast" (km)',
     ylab=expression(paste(Dxs,' (\u2030)')),main='All Evaporation')
points(distance.fake,Dxs_eddy.meanfromcoast[,1],type='o',pch=15,col='gray50')
points(IT_dist_rain,IT_Dxs_rain,pch=21,cex=0.8,col='black',bg='blue')
points(IT_dist_snow,IT_Dxs_snow,cex=0.8,pch=22,col='black',bg='light blue')
points(IT_dist_gw,IT_Dxs_gw,cex=0.8,pch=23,col='black',bg='cyan')
legend('topleft',c('advection','eddies','rain','snow','groundwater'),
       cex=0.8,pch=c(16,15,21,22,23),col=c('black','gray50',rep('black',3)),
       pt.bg=c(NA,NA,'blue','light blue','cyan'))

#70% tranpsiration
plot(distance.fake,Dxs.meanfromcoast[,8],type='o',pch=16,col='black',
     xaxs="i", ylim=c(-5,130),xlab='Distance from "Coast" (km)',
     ylab=expression(paste(Dxs,' (\u2030)')),main='70% Transpiration')
points(distance.fake,Dxs_eddy.meanfromcoast[,8],type='o',pch=15,col='gray50')
points(IT_dist_rain,IT_Dxs_rain,pch=21,cex=0.8,col='black',bg='blue')
points(IT_dist_snow,IT_Dxs_snow,cex=0.8,pch=22,col='black',bg='light blue')
points(IT_dist_gw,IT_Dxs_gw,cex=0.8,pch=23,col='black',bg='cyan')
legend('topleft',c('advection','eddies','rain','snow','groundwater'),
       cex=0.8,pch=c(16,15,21,22,23),col=c('black','gray50',rep('black',3)),
       pt.bg=c(NA,NA,'blue','light blue','cyan'))

#all tranpsiration
plot(distance.fake,Dxs.meanfromcoast[,11],type='o',pch=16,col='black',
     xaxs="i", ylim=c(-5,130),xlab='Distance from "Coast" (km)',
     ylab=expression(paste(Dxs,' (\u2030)')),main='All Transpiration')
points(distance.fake,Dxs_eddy.meanfromcoast[,11],type='o',pch=15,col='gray50'
)
points(IT_dist_rain,IT_Dxs_rain,pch=21,cex=0.8,col='black',bg='blue')
points(IT_dist_snow,IT_Dxs_snow,cex=0.8,pch=22,col='black',bg='light blue')
points(IT_dist_gw,IT_Dxs_gw,cex=0.8,pch=23,col='black',bg='cyan')
legend('topleft',c('advection','eddies','rain','snow','groundwater'),
       cex=0.8,pch=c(16,15,21,22,23),col=c('black','gray50',rep('black',3)),
       pt.bg=c(NA,NA,'blue','light blue','cyan'))
dev.off()


####

######resp-MAP relationships#######

#here we have three different empirical, linear SR-MAP relationships


####upload data
# Cotton and Sheldon 2012 (GSA Bulletin)
Cotton12=read.csv("Cotton 2012.csv",header=T,skip=1)
head(Cotton12)

# Cotton et al., 2013 (Chemical Geology)
Cotton13=read.csv("Cotton 2013.csv",header=T,skip=1)
head(Cotton13)

# Convert to Soil Respiration Rates
#assuming soil temp=25C (in CO2convert) because that's how Jeremy does it
Cotton12=data.frame(Cotton12,sapply(Cotton12$CO2ppmv,CO2convert))
Cotton12=data.frame(Cotton12,((Ds*Cotton12[,3])/(L*z-
z^2/2))*L*12*1e4*60*60*24*365) # in gC/m2/year
colnames(Cotton12)=c("SoilCO2","MAP","SoilCO2moles","SoilResp")
head(Cotton12)
#same for Cotton et al, 2013:
Cotton13=data.frame(Cotton13,sapply(Cotton13$CO2ppmv,CO2convert))
Cotton13=data.frame(Cotton13,((Ds*Cotton13[,3])/(L*z-
z^2/2))*L*12*1e4*60*60*24*365) # Calculates soil respiration in gC/m2/year
colnames(Cotton13)=c("SoilCO2","MAP","SoilCO2moles","SoilResp")
head(Cotton13)

# Calculate linear regression
Cotton12lm=lm((Cotton12$SoilResp)~Cotton12$MAP)
summary(Cotton12lm)
Cot12_intercept=Cotton12lm$coef[1]
Cot12_slope=Cotton12lm$coef[2]
#same for Cotton et al, 2013
Cotton13lm=lm((Cotton13$SoilResp)~Cotton13$MAP)
summary(Cotton13lm)
Cot13_intercept=Cotton13lm$coef[1]
Cot13_slope=Cotton13lm$coef[2]

#regression line for Raich and Schlesinger, 1992:
RS_intercept=155
RS_slope=0.391

#use these values if you don't feel like running the regressions
# Cot12_slope=1.010897
# Cot12_intercept=-48.0897
# Cot13_slope=1.712771
# Cot13_intercept=-71.87511


#######soil carbonate function#########


#Soil Carbonate d13C
soilcarb=function(CO2ppm_atm,MAP,Temp=298.15,MAPSR="C12",z=50,delta_13_a=-
7,delta_13_o=-27){
  #MAP in mm/yr
  #Temp is soil temperature IN KELVIN
  #MAPSR indicates the regression to use

  TempC=Temp-273.15

  if (MAPSR=="C12") {SR=MAP*Cot12_slope+Cot12_intercept}
  if (MAPSR=="C13") {SR=MAP*Cot13_slope+Cot13_intercept}
  if (MAPSR=="RS") {SR=MAP*RS_slope+RS_intercept}

  prod=SR/12/365/24/60/60/1e4/L # Converts production (g/m2/yr) to
(moles/cm3/s)

  CO2_atm=CO2convert(CO2ppm_atm,TempC) # Converts CO2 (ppm) to CO2
(moles/cm3)

  Sz=(prod/Ds)*(L*z-z^2/2) #S(z) Soil-respired CO2 concentration (moles/cm3)
  delta_13_s=((CO2_atm/Sz)*delta_13_a+1.0044*delta_13_o+4.4)/(1+CO2_atm/Sz) #
Soil d13C value

  alpha=fract(TempC)

  dcarb=delta_13_s-alpha # Soil carbonate, assuming equilibrium

  # Outputs a vector of soil-respired CO2 (ppm), soil d13C, carbonate d13C,
rounded to two decimal places
  result=c(round(CO2backconvert(Sz),3),round(delta_13_s,2),round(dcarb,2))
  names(result)=c("Soil CO2","Soil d13C","Carbonate d13C")
  result
}

#soilcarb(CO2ppm_atm=300,MAP=300)
```

```
#soilcarb(CO2ppm_atm=300,MAP=1000)
#soilcarb(CO2ppm_atm=1000,MAP=300)


#######soil carbon profile calculations########

###get MAP of "average from coast" track

#turn all of the inputs into one storm track:
#pre for pre-averaging
MAP.pre=as.matrix(MAP.data) #mean annual precip in mm/year
tsoil.pre=tsoil.data
ntracks.pre=dim(MAP.pre)[2]
ntracks=1 #number of storm tracks
tracklength=dim(MAP.pre)[1] #length of storm track

#if land.mask=0, cut off the previous data point and add NA to the end
#cutting off previous allows for initiation just before land, consistent with
sections above
#adding NA at the end will allow for averaging of all the non-NAs for farther
out on track
for (i in 2:tracklength) {
  for (j in 1:ntracks.pre) {
    if (land.mask.data[i,j]==0) {
      MAP.pre[,j]=c(MAP.pre[-(i-1),j],NA)
    }
  }
}
#same for soil temperatures
for (i in 2:tracklength) {
  for (j in 1:ntracks.pre) {
    for (k in 1:length(zplot)) {
      if (land.mask.data[i,j]==0) {
        tsoil.pre[,j,k]=c(tsoil.pre[-(i-1),j,k],NA)
      }
    }
  }
}

#now average these. this average is based on distance from coast!
MAP=rowMeans(MAP.pre,na.rm=T)
tsoil=apply(tsoil.pre,c(1,3),mean,na.rm=T) #28x101

###soil CO2 as a function of depth
soilCO2.1=array(dim=c(tracklength,length(zplot),3)) #for Cotton 2012
soilCO2.2=array(dim=c(tracklength,length(zplot),3)) #for Cotton 2013
soilCO2.3=array(dim=c(tracklength,length(zplot),3)) #for Raich and
Schlesinger
#currently assuming Temp=25C (function default)
for (i in 1:tracklength) {
  for (j in 1:length(zplot)) {

soilCO2.1[i,j,]=soilcarb(CO2ppm_atm=pCO2ppm_atm,MAP=MAP[i],Temp=tsoil[i,j],z=
zplot[j],delta_13_a=delta_13_atm)

soilCO2.2[i,j,]=soilcarb(CO2ppm_atm=pCO2ppm_atm,MAP=MAP[i],Temp=tsoil[i,j],MA
PSR='C13',z=zplot[j],delta_13_a=delta_13_atm)

soilCO2.3[i,j,]=soilcarb(CO2ppm_atm=pCO2ppm_atm,MAP=MAP[i],Temp=tsoil[i,j],MA
PSR='RS',z=zplot[j],delta_13_a=delta_13_atm)
  }
}
#names(soilCO2[i,j,])=c("Soil CO2","Soil d13C","Carbonate d13C")
#^names of the three outputs of soilcarb() function
```

```
####soil CO2 concentration (combination of soil-respired and atmospheric)
#necessary for soil water model
#Cerling 1991:
CO2ppm_soil.1=soilCO2.1[,,1]+pCO2ppm_atm #for Cotton 2012
CO2ppm_soil.2=soilCO2.2[,,1]+pCO2ppm_atm #for Cotton 2013
CO2ppm_soil.3=soilCO2.3[,,1]+pCO2ppm_atm #for Raich and Schlesinger
#28 tracks, maxdepth+1 depths


######plot model soil carbonate profiles########

#Soil-Respired CO2 at beginning of track (arbitrary)
plot(soilCO2.1[1,,1],zplot,ylim=c(max(zplot),0),xlim=c(0,12000),type='l',yaxs
='i',
     xlab=expression(paste('Soil-Respired ',CO[2],' (ppm)')),ylab='depth
(cm)')
points(soilCO2.2[1,,1],zplot,type='l',col='red')
points(soilCO2.3[1,,1],zplot,type='l',col='blue')
legend('topright',legend=c('Cotton12','Cotton13','S&R'),col=c('black','red','
blue'),lty=1)

#Soil d13C
plot(soilCO2.1[1,,2],zplot,ylim=c(max(zplot),0),type='l',yaxs='i',
     xlab=expression(paste('Soil ',delta^13,C[VPDB],' (\u2030)')),ylab='depth
(cm)')
points(soilCO2.2[1,,2],zplot,type='l',col='red')
points(soilCO2.3[1,,2],zplot,type='l',col='blue')
legend('bottomright',legend=c('Cotton12','Cotton13','S&R'),col=c('black','red
','blue'),lty=1)

#Carbonate d13C
plot(soilCO2.1[1,,3],zplot,ylim=c(max(zplot),0),type='l',yaxs='i',
     xlab=expression(paste('Soil Carbonate',delta^13,C[VPDB],'
(\u2030)')),ylab='depth (cm)')
points(soilCO2.2[1,,3],zplot,type='l',col='red')
points(soilCO2.3[1,,3],zplot,type='l',col='blue')
legend('bottomright',legend=c('Cotton12','Cotton13','S&R'),col=c('black','red
','blue'),lty=1)



#########soil water component#########
####

# The soil profile is calculated using the equations from
# Zimmerman et al. (1967) as outlined in Chamberlain et al. (2014)

###NOTE this is for ONE storm track (in this case the "average from coast")
#model inputs!
w.pre=as.matrix(Prw.data) #precipitatble water, kg/m^2
tracklength=length(w.pre[,1]) #length of storm tracks
ntracks.pre=dim(w.pre)[2]
ntracks=1 #number of storm tracks
Tsurf.pre=as.matrix(Temp_surf.data) #JJASO ground surface temperature (K)
tsoil.pre=tsoil.data
rh.pre=as.matrix(rhum.data/100) #relative humidity (fraction)
evap.pre=as.matrix(evap.data)  #evap rate in kg/m^2/month (mm/month over a
given area)
#if land.mask=0, cut off the previous data point and add NA to the end
#cutting off previous allows for initiation just before land, consistent with
sections above
#adding NA at the end will allow for averaging of all the non-NAs for farther
```

```
out on track
for (j in 1:ntracks.pre) {
  for (i in 2:tracklength) {
    if (land.mask.data[i,j]==0) {
      Tsurf.pre[,j]=c(Tsurf.pre[-(i-1),j],NA)
      rh.pre[,j]=c(rh.pre[-(i-1),j],NA)
      evap.pre[,j]=c(evap.pre[-(i-1),j],NA)
    }
  }
}
#same for soil temperatures
for (i in 2:tracklength) {
  for (j in 1:ntracks.pre) {
    for (k in 1:length(zplot)) {
      if (land.mask.data[i,j]==0) {
        tsoil.pre[,j,k]=c(tsoil.pre[-(i-1),j,k],NA)
      }
    }
  }
}
#now average these. this average is based on distance from coast!
TsurfK=rowMeans(Tsurf.pre,na.rm=T) #surface temp in Kelvin
rh=rowMeans(rh.pre,na.rm=T) #rel. humidity as fraction
tsoil=apply(tsoil.pre,c(1,3),mean,na.rm=T) #soil temp with depth in Kelvin
evap.mm.month=rowMeans(evap.pre,na.rm=T) #evap rate in mm/month
#convert evap rate to m/sec
evap.m.sec=evap.mm.month*(3.8580247e-10) #evap rate in m/sec
E=evap.m.sec
zstar=((p*tau*d)/E)/100 #in cm depth (dividing by 100 necessary for correct
units)


###functions to make carb and water profiles###

# FUNCTION TO CALCULATE d18O (VSMOW) in water to d18O (VPDB) in carbonate ###
water2carb.oxygen <- function(delta18_precip = -15,TempK,pH=8,pCO2ppm=2500){
  #TempK should be soil temperature!
  #Carbon section Equil gas Zhang et al. (1995); CaCO3 Bottinga (1968)
  #Equilibrium for carbon isotopes
  TempC=TempK-273.15
  DeltaC1 = -(0.1141*TempC) + 10.78    #Bicarb-CO2g
  DeltaC2 = -(0.052*TempC) + 7.22    #Carb-CO2g
  DeltaC3 = (0.0049*TempC) - 1.31    #CO2aq-CO2g
  DeltaC4 = -2.4612 + (7.6663*10^3/TempK) - (2.988*10^6/TempK^2) #CO2g-CaCO3s
  #Kinetic for carbon isotopes
  DeltaCK = -0.81 #Kinectic fractionation CO2aq-CO2gas @ 21C for 5C use -0.95
  #Oxygen isotope section Equil gas Beck et al. (2005); CaCO3-H2O Kim and
O'Neil (1997)
  DeltaO1 = (2.59*10^6/TempK^2) + 1.89    #Bicarb-H2O
  DeltaO2 = (2.39*10^6/TempK^2) - 2.7    #Carb-H2O
  DeltaO3 = (2.52*10^6/TempK^2) + 12.12    #CO2aq-H2O
  DeltaO4 = (18.03*10^3/TempK) - 32.42    #CaCO3-H2O
  DeltaO5 = DeltaO1 - DeltaO2            #Bicarb-Carb
  # Constants needed for calculation of carbonate species
  Ko = 0.032  #Henrys Law constant for CO2gas to CO2aq units mol/(kg*atm)
  pK1 = 3404.71/TempK + 0.032786*TempK - 14.8435 # Harned and Davis (1943)
  pK2 = 2902.39/TempK + 0.02379*TempK - 6.4980 # Harned and Davis (1943)
  K1 = 10^(-pK1)
  K2 = 10^(-pK2)
  # Calculate CO2aq from Henry's Law
  pCO2 = pCO2ppm/(1*10^6) #Conversion to atm
  CO2 = Ko * pCO2  # CO2 is in mol/kg PCO2 is in atm
  # Calculate carbonate species give CO2 conc and pH
  H = 10^(-pH)
  DIC = CO2 * (1 + K1/H + K1*K2/H^2)
  BICARB = DIC / (1 + H/K1 + K2/H)
  CARB = DIC / (1 + H/K2 + H^2/K1*K2)
  # Calculate the oxygen isotope compostion of calcite with speciation
  XBICARB = BICARB/(BICARB + CARB)
  Delta18Ocalcite = XBICARB * DeltaO5 + delta18_precip + DeltaO2
  # function outputs a vector of Delta18Ocalcite based on each temperature
value used
  #Delta18Ocalcite #VSMOW!
  #convert to VPDB:
  d18O_calciteVPDB=(Delta18Ocalcite-30.91)/1.03091
  d18O_calciteVPDB
}
# water2carb.oxygen(-5,284)

# FUNCTION TO CALCULATE SOIL WATER PROFILE ###
# spits out profile of soil water d18O given delta18_precip, zstar, maxdepth,
and tempK_surface
soil.profile <- function(delta18_precip = -15.0,TsurfK,rh,zstar =
50,maxdepth=200){
  #defaults to zstar=50, maxdepth=200cm if none given *in function call*
  #temperature should be ground surface temp, not air 2m temp!
  # kinetic and equilibrium fracionation factors
  alpha_eq = ((-7.685+6712.3/(TsurfK)-
1666400/((TsurfK)^2)+350410000/((TsurfK)^3)) + 1000)/1000
  alpha_kin = ((14.2*(1-rh)) + 1000)/1000

  R_precip = (delta18_precip/1000)*0.0020052 + 0.0020052
  R_atm = R_precip/alpha_eq

  R_sur = alpha_eq*((1-rh)*alpha_kin*R_precip + rh*R_atm)
  #z = seq(1,maxdepth,1) # assuming soil depth of 2 m max (z is depth in cm)
  z=seq(0,maxdepth)
  # zstar = (p*t*d)/E # this is z*
  profile <- (((R_sur - R_precip)*exp(-z/zstar) + R_precip)/0.0020052 -
1)*1000
  profile #vector of soil water d18O with depth
}
#soil.profile(-7,284,0.85,50,100)

####calculate soil profiles

#calculate soil water profile at every point along track for each E/T ratio
soilwater18=array(dim=c(tracklength,length(zplot),length(e)))
soilcarb18=array(dim=c(tracklength,length(zplot),length(e)))
#and also for eddies!
soilwater18_eddy=array(dim=c(tracklength,length(zplot),length(e)))
soilcarb18_eddy=array(dim=c(tracklength,length(zplot),length(e)))
for (i in 1:tracklength) {
  for (j in 1:length(zplot)) {
    for (h in 1:length(e)) {

soilwater18[i,,h]=soil.profile(delta_p.meanfromcoast[i,h],TsurfK[i],rh[i],zst
ar[i],maxdepth)

soilwater18_eddy[i,,h]=soil.profile(delta_p_eddy.meanfromcoast[i,h],TsurfK[i]
,rh[i],zstar[i],maxdepth)
      soilcarb18[i,j,h]=water2carb.oxygen(soilwater18[i,j,h],tsoil[i,j],pH)

soilcarb18_eddy[i,j,h]=water2carb.oxygen(soilwater18_eddy[i,j,h],tsoil[i,j],p
H)
    }
  }
}
```

```
#these have dim 28x219x11 #28 points, 219 depth values, 11 E/T ratios

##FOR COTTON 2012
#calculate soil water profile at every point along track for each E/T ratio
soilwater18.1=array(dim=c(tracklength,length(zplot),length(e)))
soilcarb18.1=array(dim=c(tracklength,length(zplot),length(e)))
#and also for eddies!
soilwater18_eddy.1=array(dim=c(tracklength,length(zplot),length(e)))
soilcarb18_eddy.1=array(dim=c(tracklength,length(zplot),length(e)))
for (i in 1:tracklength) {
  for (j in 1:length(zplot)) {
    for (h in 1:length(e)) {

soilwater18.1[i,,h]=soil.profile(delta_p.meanfromcoast[i,h],TsurfK[i],rh[i],z
star[i],maxdepth)

soilwater18_eddy.1[i,,h]=soil.profile(delta_p_eddy.meanfromcoast[i,h],TsurfK[
i],rh[i],zstar[i],maxdepth)

soilcarb18.1[i,j,h]=water2carb.oxygen(soilwater18.1[i,j,h],tsoil[i,j],pH,CO2p
pm_soil.1[i,j])

soilcarb18_eddy.1[i,j,h]=water2carb.oxygen(soilwater18_eddy.1[i,j,h],tsoil[i,
j],pH,CO2ppm_soil.1[i,j])
    }
  }
}

##COTTON 2013
soilwater18.2=array(dim=c(tracklength,length(zplot),length(e)))
soilcarb18.2=array(dim=c(tracklength,length(zplot),length(e)))
#and also for eddies!
soilwater18_eddy.2=array(dim=c(tracklength,length(zplot),length(e)))
soilcarb18_eddy.2=array(dim=c(tracklength,length(zplot),length(e)))
for (i in 1:tracklength) {
  for (j in 1:length(zplot)) {
    for (h in 1:length(e)) {

soilwater18.2[i,,h]=soil.profile(delta_p.meanfromcoast[i,h],TsurfK[i],rh[i],z
star[i],maxdepth)

soilwater18_eddy.2[i,,h]=soil.profile(delta_p_eddy.meanfromcoast[i,h],TsurfK[
i],rh[i],zstar[i],maxdepth)

soilcarb18.2[i,j,h]=water2carb.oxygen(soilwater18.2[i,j,h],tsoil[i,j],pH,CO2p
pm_soil.2[i,j])

soilcarb18_eddy.2[i,j,h]=water2carb.oxygen(soilwater18_eddy.2[i,j,h],tsoil[i,
j],pH,CO2ppm_soil.2[i,j])
    }
  }
}

###RAICH and SCHLESINGER 1992
soilwater18.3=array(dim=c(tracklength,length(zplot),length(e)))
soilcarb18.3=array(dim=c(tracklength,length(zplot),length(e)))
#and also for eddies!
soilwater18_eddy.3=array(dim=c(tracklength,length(zplot),length(e)))
soilcarb18_eddy.3=array(dim=c(tracklength,length(zplot),length(e)))
for (i in 1:tracklength) {
  for (j in 1:length(zplot)) {
    for (h in 1:length(e)) {

soilwater18.3[i,,h]=soil.profile(delta_p.meanfromcoast[i,h],TsurfK[i],rh[i],z
star[i],maxdepth)

soilwater18_eddy.3[i,,h]=soil.profile(delta_p_eddy.meanfromcoast[i,h],TsurfK[
i],rh[i],zstar[i],maxdepth)

soilcarb18.3[i,j,h]=water2carb.oxygen(soilwater18.3[i,j,h],tsoil[i,j],pH,CO2p
pm_soil.3[i,j])

soilcarb18_eddy.3[i,j,h]=water2carb.oxygen(soilwater18_eddy.3[i,j,h],tsoil[i,
j],pH,CO2ppm_soil.3[i,j])
    }
  }
}

#######plot model soil water profiles########


#we have 11 E/ET ratios, 28 points along track, 3 MAP-resp relationships, and
eddies vs advection.
#arbitrarily choosing distance of 576km from coast (approximately Hay data
from below)

point=which(distance.fake==576) #arbitrary point on storm track (19)
#dim(soilcarb18) #28 points, 219 depths, 11 E/ETs

#start with advection only, constant soil CO2
plot(soilcarb18[point,,1],zplot,yaxs='i',ylim=c(max(zplot),0),lwd=3,type='l',
lty=1,col='red',
     xlim=c(-25,0),xlab=expression(paste(delta^18,'O'[VPDB],'
(\u2030)')),ylab='depth (cm)')
points(soilcarb18[point,,8],zplot,yaxs='i',lwd=3,type='l',lty=1,col='green')
points(soilcarb18[point,,11],zplot,yaxs='i',lwd=3,type='l',lty=1,col='blue')
#Cotton 2012, advection only
points(soilcarb18.1[point,,1],zplot,yaxs='i',lwd=3,type='l',lty=2,col='red')
points(soilcarb18.1[point,,8],zplot,yaxs='i',lwd=3,type='l',lty=2,col='green'
)
points(soilcarb18.1[point,,11],zplot,yaxs='i',lwd=3,type='l',lty=2,col='blue'
)
#Cotton 2013, advection only
points(soilcarb18.2[point,,1],zplot,yaxs='i',lwd=3,type='l',lty=3,col='red')
points(soilcarb18.2[point,,8],zplot,yaxs='i',lwd=3,type='l',lty=3,col='green'
)
points(soilcarb18.2[point,,11],zplot,yaxs='i',lwd=3,type='l',lty=3,col='blue'
)
#Raich and Schlesinger, advection only
points(soilcarb18.3[point,,1],zplot,yaxs='i',lwd=3,type='l',lty=4,col='red')
points(soilcarb18.3[point,,8],zplot,yaxs='i',lwd=3,type='l',lty=4,col='green'
)
points(soilcarb18.3[point,,11],zplot,yaxs='i',lwd=3,type='l',lty=4,col='blue'
)
#constant soil CO2, eddies only
points(soilcarb18_eddy[point,,1],zplot,yaxs='i',lwd=1,type='l',lty=1,col='red
')
points(soilcarb18_eddy[point,,8],zplot,yaxs='i',lwd=1,type='l',lty=1,col='gre
en')
points(soilcarb18_eddy[point,,11],zplot,yaxs='i',lwd=1,type='l',lty=1,col='bl
ue')
#Cotton 2012, advection only
points(soilcarb18_eddy.1[point,,1],zplot,yaxs='i',lwd=1,type='l',lty=2,col='r
ed')
points(soilcarb18_eddy.1[point,,8],zplot,yaxs='i',lwd=1,type='l',lty=2,col='g
reen')
points(soilcarb18_eddy.1[point,,11],zplot,yaxs='i',lwd=1,type='l',lty=2,col='
blue')
```

```
#Cotton 2013, advection only
points(soilcarb18_eddy.2[point,,1],zplot,yaxs='i',lwd=1,type='l',lty=3,col='r
ed')
points(soilcarb18_eddy.2[point,,8],zplot,yaxs='i',lwd=1,type='l',lty=3,col='g
reen')
points(soilcarb18_eddy.2[point,,11],zplot,yaxs='i',lwd=1,type='l',lty=3,col='
blue')
#Raich and Schlesinger, advection only
points(soilcarb18_eddy.3[point,,1],zplot,yaxs='i',lwd=1,type='l',lty=4,col='r
ed')
points(soilcarb18_eddy.3[point,,8],zplot,yaxs='i',lwd=1,type='l',lty=4,col='g
reen')
points(soilcarb18_eddy.3[point,,11],zplot,yaxs='i',lwd=1,type='l',lty=4,col='
blue')
#####THERE seems to be NO sensitivity to soil pCO2 in the d18O values



#######modern soil profiles########

soildata=read.csv('modern soils.csv')
head(soildata)

#add column for distance from "coast"
library(geosphere)
library(dplyr)
lati4=37.7
loni4=-122.5
lati5=36.27
loni5=-121.8
latitudes=soildata$latitude
longitudes=soildata$longitude
distmin <- dist1 <- dist2 <- dist3 <- dist4 <- dist5 <- numeric()
for (i in 1:length(latitudes)){
  dist1[i]=distVincentyEllipsoid(c(longitudes[i],latitudes[i]),c(loni,lati))

dist2[i]=distVincentyEllipsoid(c(longitudes[i],latitudes[i]),c(loni2,lati2))

dist3[i]=distVincentyEllipsoid(c(longitudes[i],latitudes[i]),c(loni3,lati3))

dist4[i]=distVincentyEllipsoid(c(longitudes[i],latitudes[i]),c(loni4,lati4))

dist5[i]=distVincentyEllipsoid(c(longitudes[i],latitudes[i]),c(loni5,lati5))
  distmin[i]=min(dist1[i],dist2[i],dist3[i],dist4[i],dist5[i])/1000 #distance
in km
}

soildata2=cbind(soildata,distmin) #added column for distance from coast
head(soildata2)

#our comparable distances for the modern soil data (left) and model output
(right) are:
#640.1107     #640  #Newark Valley
#791.6856     #800  #South of Wendover
#586.1828     #576  #Hay
#367.1008     #352  #Pendall et al 1994
#551.1161     #544  #Quade et al 1989
#447.7820     #448  #Quade et al 1989

#I'm going to plot these six for data-model comparison
point1=which(distance.fake==352)
point2=which(distance.fake==448)
point3=which(distance.fake==544)
point4=which(distance.fake==576)
```

```
point5=which(distance.fake==640)
point6=which(distance.fake==800)


###
#####d18O plots!
###


#Profile1: distance 367.1008/352
profile1.1=soilcarb18[point1,,1] #all evap
profile1.2=soilcarb18[point1,,8] #70% transpiration
profile1.3=soilcarb18[point1,,11] #all transpiration
profile1.1_eddy=soilcarb18_eddy[point1,,1] #all evap, eddies only
profile1.2_eddy=soilcarb18_eddy[point1,,8] #70% transpiration, eddies only
profile1.3_eddy=soilcarb18_eddy[point1,,11] #all transpiration, eddies only
profile1.d18O=filter(soildata2,latitude==37.6947)$d18O
profile1.depth=filter(soildata2,latitude==37.6947)$depth

#Profile2: distance 448
profile2.1=soilcarb18[point2,,1] #all evap
profile2.2=soilcarb18[point2,,8] #70% transpiration
profile2.3=soilcarb18[point2,,11] #all transpiration
profile2.1_eddy=soilcarb18_eddy[point2,,1] #all evap, eddies only
profile2.2_eddy=soilcarb18_eddy[point2,,8] #70% transpiration, eddies only
profile2.3_eddy=soilcarb18_eddy[point2,,11] #all transpiration, eddies only
profile2.d18O=filter(soildata2,latitude==36.2469)$d18O
profile2.depth=filter(soildata2,latitude==36.2469)$depth

#Profile3: distance 544
profile3.1=soilcarb18[point3,,1] #all evap
profile3.2=soilcarb18[point3,,8] #70% transpiration
profile3.3=soilcarb18[point3,,11] #all transpiration
profile3.1_eddy=soilcarb18_eddy[point3,,1] #all evap, eddies only
profile3.2_eddy=soilcarb18_eddy[point3,,8] #70% transpiration, eddies only
profile3.3_eddy=soilcarb18_eddy[point3,,11] #all transpiration, eddies only
profile3.d18O=filter(soildata2,latitude==36.2981)$d18O
profile3.depth=filter(soildata2,latitude==36.2981)$depth

#Profile4: distance 576
profile4.1=soilcarb18[point4,,1] #all evap
profile4.2=soilcarb18[point4,,8] #70% transpiration
profile4.3=soilcarb18[point4,,11] #all transpiration
profile4.1_eddy=soilcarb18_eddy[point4,,1] #all evap, eddies only
profile4.2_eddy=soilcarb18_eddy[point4,,8] #70% transpiration, eddies only
profile4.3_eddy=soilcarb18_eddy[point4,,11] #all transpiration, eddies only
profile4.d18O=filter(soildata2,latitude==39.572481)$d18O
profile4.depth=filter(soildata2,latitude==39.572481)$depth

#Profile5: distance 640
profile5.1=soilcarb18[point5,,1] #all evap
profile5.2=soilcarb18[point5,,8] #70% transpiration
profile5.3=soilcarb18[point5,,11] #all transpiration
profile5.1_eddy=soilcarb18_eddy[point5,,1] #all evap, eddies only
profile5.2_eddy=soilcarb18_eddy[point5,,8] #70% transpiration, eddies only
profile5.3_eddy=soilcarb18_eddy[point5,,11] #all transpiration, eddies only
profile5.d18O=filter(soildata2,latitude==39.876944)$d18O
profile5.depth=filter(soildata2,latitude==39.876944)$depth

#Profile6: distance 800
profile6.1=soilcarb18[point6,,1] #all evap
profile6.2=soilcarb18[point6,,8] #70% transpiration
profile6.3=soilcarb18[point6,,11] #all transpiration
profile6.1_eddy=soilcarb18_eddy[point6,,1] #all evap, eddies only
```

```
profile6.2_eddy=soilcarb18_eddy[point6,,8] #70% transpiration, eddies only
profile6.3_eddy=soilcarb18_eddy[point6,,11] #all transpiration, eddies only
profile6.d18O=filter(soildata2,latitude==40.594903)$d18O
profile6.depth=filter(soildata2,latitude==40.594903)$depth


### d18O profile plots!

pdf('Modern d18O Profiles with eddies and Data.pdf',width=10,height=8)
par(mfrow=c(2,3))
plot(profile1.1,zplot,ylim=c(maxdepth,0),xlim=c(-
32,3),type='l',lwd=3,col='red',
     xlab=expression(paste(delta^18,'O'[VPDB],' (\u2030)')),ylab='depth
(cm)',
     main='Pendall et al, 1994 (~350km)')
points(profile1.2,zplot,type='l',lwd=3,col='green')
points(profile1.3,zplot,type='l',lwd=3,col='blue')
points(profile1.1_eddy,zplot,type='l',col='red')
points(profile1.2_eddy,zplot,type='l',col='green')
points(profile1.3_eddy,zplot,type='l',col='blue')
points(profile1.d18O,profile1.depth,pch=16)
legend('bottomleft',pch=c(16,rep(NA,5)),lty=c(NA,rep(1,5)),lwd=c(rep(1,4),3),
       col=c('black','red','green','blue','black','black'),cex=0.8,
       legend=c('data','all E','70% T','all T','advection','eddies'))

plot(profile2.1,zplot,ylim=c(maxdepth,0),xlim=c(-
32,3),type='l',lwd=3,col='red',
     xlab=expression(paste(delta^18,'O'[VPDB],' (\u2030)')),ylab='depth
(cm)',
     main='Quade et al, 1989 (~450km)')
points(profile2.2,zplot,type='l',lwd=3,col='green')
points(profile2.3,zplot,type='l',lwd=3,col='blue')
points(profile2.1_eddy,zplot,type='l',col='red')
points(profile2.2_eddy,zplot,type='l',col='green')
points(profile2.3_eddy,zplot,type='l',col='blue')
points(profile2.d18O,profile2.depth,pch=16)
legend('bottomleft',pch=c(16,rep(NA,5)),lty=c(NA,rep(1,5)),lwd=c(rep(1,4),3),
       col=c('black','red','green','blue','black','black'),cex=0.8,
       legend=c('data','all E','70% T','all T','advection','eddies'))

plot(profile3.1,zplot,ylim=c(maxdepth,0),xlim=c(-
32,3),type='l',lwd=3,col='red',
     xlab=expression(paste(delta^18,'O'[VPDB],' (\u2030)')),ylab='depth
(cm)',
     main='Quade et al, 1989 (~550km)')
points(profile3.2,zplot,type='l',lwd=3,col='green')
points(profile3.3,zplot,type='l',lwd=3,col='blue')
points(profile3.1_eddy,zplot,type='l',col='red')
points(profile3.2_eddy,zplot,type='l',col='green')
points(profile3.3_eddy,zplot,type='l',col='blue')
points(profile3.d18O,profile3.depth,pch=16)
legend('bottomright',pch=c(16,rep(NA,5)),lty=c(NA,rep(1,5)),lwd=c(rep(1,4),3)
,
       col=c('black','red','green','blue','black','black'),cex=0.8,
       legend=c('data','all E','70% T','all T','advection','eddies'))

plot(profile4.1,zplot,ylim=c(maxdepth,0),xlim=c(-
32,3),type='l',lwd=3,col='red',
     xlab=expression(paste(delta^18,'O'[VPDB],' (\u2030)')),ylab='depth
(cm)',
     main='Hay (~580km)')
points(profile4.2,zplot,type='l',lwd=3,col='green')
points(profile4.3,zplot,type='l',lwd=3,col='blue')
points(profile4.1_eddy,zplot,type='l',col='red')
points(profile4.2_eddy,zplot,type='l',col='green')
points(profile4.3_eddy,zplot,type='l',col='blue')
points(profile4.d18O,profile4.depth,pch=16)
legend('bottomright',pch=c(16,rep(NA,5)),lty=c(NA,rep(1,5)),lwd=c(rep(1,4),3)
,
       col=c('black','red','green','blue','black','black'),cex=0.8,
       legend=c('data','all E','70% T','all T','advection','eddies'))

plot(profile5.1,zplot,ylim=c(maxdepth,0),xlim=c(-
32,3),type='l',lwd=3,col='red',
     xlab=expression(paste(delta^18,'O'[VPDB],' (\u2030)')),ylab='depth
(cm)',
     main='Newark Valley (~640km)')
points(profile5.2,zplot,type='l',lwd=3,col='green')
points(profile5.3,zplot,type='l',lwd=3,col='blue')
points(profile5.1_eddy,zplot,type='l',col='red')
points(profile5.2_eddy,zplot,type='l',col='green')
points(profile5.3_eddy,zplot,type='l',col='blue')
points(profile5.d18O,profile5.depth,pch=16)
legend('bottomright',pch=c(16,rep(NA,5)),lty=c(NA,rep(1,5)),lwd=c(rep(1,4),3)
,
       col=c('black','red','green','blue','black','black'),cex=0.8,
       legend=c('data','all E','70% T','all T','advection','eddies'))

plot(profile6.1,zplot,ylim=c(maxdepth,0),xlim=c(-
32,3),type='l',lwd=3,col='red',
     xlab=expression(paste(delta^18,'O'[VPDB],' (\u2030)')),ylab='depth
(cm)',
     main='South of Wendover (~800km)')
points(profile6.2,zplot,type='l',lwd=3,col='green')
points(profile6.3,zplot,type='l',lwd=3,col='blue')
points(profile6.1_eddy,zplot,type='l',col='red')
points(profile6.2_eddy,zplot,type='l',col='green')
points(profile6.3_eddy,zplot,type='l',col='blue')
points(profile6.d18O,profile6.depth,pch=16)
legend('bottomright',pch=c(16,rep(NA,5)),lty=c(NA,rep(1,5)),lwd=c(rep(1,4),3)
,
       col=c('black','red','green','blue','black','black'), cex=0.8,
       legend=c('data','all E','70% T','all T','advection','eddies'))
dev.off()


###
######d13C plots!
####


#extract d13C data:

#Profile1: distance 352
profile1.d13C=filter(soildata2,latitude==37.6947)$d13C
profile1.depth=filter(soildata2,latitude==37.6947)$depth
#Profile2: distance 448
profile2.d13C=filter(soildata2,latitude==36.2469)$d13C
profile2.depth=filter(soildata2,latitude==36.2469)$depth
#Profile3: distance 544
profile3.d13C=filter(soildata2,latitude==36.2981)$d13C
profile3.depth=filter(soildata2,latitude==36.2981)$depth
#Profile4: distance 576
profile4.d13C=filter(soildata2,latitude==39.572481)$d13C
profile4.depth=filter(soildata2,latitude==39.572481)$depth
#Profile5: distance 640
profile5.d13C=filter(soildata2,latitude==39.876944)$d13C
profile5.depth=filter(soildata2,latitude==39.876944)$depth
```

```
#Profile6: distance 800
profile6.d13C=filter(soildata2,latitude==40.594903)$d13C
profile6.depth=filter(soildata2,latitude==40.594903)$depth

#plot for multiple points and 3 MAP-resp relationships vs depth
pdf('Modern d13C Profiles.pdf',width=10,height=8)
par(mfrow=c(2,3))
plot(soilCO2.1[point1,,3],zplot,ylim=c(max(zplot),0),yaxs='i',type='l',col='r
ed',
     xlab=expression(paste('Soil-Respired ',CO[2],' (ppm)')),ylab='depth
(cm)',
     main='Pendall et al, 1994 (~350km)')
points(soilCO2.2[point1,,3],zplot,type='l',col='blue')
points(soilCO2.3[point1,,3],zplot,type='l',col='green')
points(profile1.d13C,profile1.depth,pch=16)
legend('bottomright',legend=c('Cotton12','Cotton13','S&R'),col=c('red','blue'
,'green'),lty=1)

plot(soilCO2.1[point2,,3],zplot,ylim=c(max(zplot),0),yaxs='i',type='l',col='r
ed',xlim=c(-13,2),
     xlab=expression(paste('Soil-Respired ',CO[2],' (ppm)')),ylab='depth
(cm)',
     main='Quade et al, 1989 (~450km)')
points(soilCO2.2[point2,,3],zplot,type='l',col='blue')
points(soilCO2.3[point2,,3],zplot,type='l',col='green')
points(profile2.d13C,profile2.depth,pch=16)
legend('bottomright',legend=c('Cotton12','Cotton13','S&R'),col=c('red','blue'
,'green'),lty=1)

plot(soilCO2.1[point3,,3],zplot,ylim=c(max(zplot),0),yaxs='i',type='l',col='r
ed',xlim=c(-13,2),
     xlab=expression(paste('Soil-Respired ',CO[2],' (ppm)')),ylab='depth
(cm)',
     main='Quade et al, 1989 (~550km)')
points(soilCO2.2[point3,,3],zplot,type='l',col='blue')
points(soilCO2.3[point3,,3],zplot,type='l',col='green')
points(profile3.d13C,profile3.depth,pch=16)
legend('bottomright',legend=c('Cotton12','Cotton13','S&R'),col=c('red','blue'
,'green'),lty=1)

plot(soilCO2.1[point4,,3],zplot,ylim=c(max(zplot),0),yaxs='i',type='l',col='r
ed',xlim=c(-13,2),
     xlab=expression(paste('Soil-Respired ',CO[2],' (ppm)')),ylab='depth
(cm)',
     main='Hay (~580km)')
points(soilCO2.2[point4,,3],zplot,type='l',col='blue')
points(soilCO2.3[point4,,3],zplot,type='l',col='green')
points(profile4.d13C,profile4.depth,pch=16)
legend('bottomright',legend=c('Cotton12','Cotton13','S&R'),col=c('red','blue'
,'green'),lty=1)

plot(soilCO2.1[point5,,3],zplot,ylim=c(max(zplot),0),yaxs='i',type='l',col='r
ed',xlim=c(-13,2),
     xlab=expression(paste('Soil-Respired ',CO[2],' (ppm)')),ylab='depth
(cm)',
     main='Newark Valley (~640km)')
points(soilCO2.2[point5,,3],zplot,type='l',col='blue')
points(soilCO2.3[point5,,3],zplot,type='l',col='green')
points(profile5.d13C,profile5.depth,pch=16)
legend('bottomright',legend=c('Cotton12','Cotton13','S&R'),col=c('red','blue'
,'green'),lty=1)

plot(soilCO2.1[point6,,3],zplot,ylim=c(max(zplot),0),yaxs='i',type='l',col='r
ed',xlim=c(-13,2),
     xlab=expression(paste('Soil-Respired ',CO[2],' (ppm)')),ylab='depth
(cm)',
     main='South of Wendover (~800km)')
points(soilCO2.2[point6,,3],zplot,type='l',col='blue')
points(soilCO2.3[point6,,3],zplot,type='l',col='green')
points(profile6.d13C,profile6.depth,pch=16)
legend('bottomright',legend=c('Cotton12','Cotton13','S&R'),col=c('red','blue'
,'green'),lty=1)
dev.off()

###OOOF THESE ARE STILL BAD

##########remaking map########
#this time including profile locations and initial lat/lon points
library(maps)
library(oce)
library(dplyr)

latis=c(lati,lati2,lati3,lati4,lati5)
lonis=c(loni,loni2,loni3,loni4,loni5)
proflats=summarise(group_by(soildata2,latitude),longitude=mean(longitude))$la
titude
proflons=summarise(group_by(soildata2,latitude),longitude=mean(longitude))$lo
ngitude

lim=c(-22,0)
ncol=100
col=oceColorsJet(ncol)
# Draw colorbar for d18O
pdf('map of data with profile locations.pdf',width=10, height=7)
par(mfrow=c(1,2))
drawPalette(zlim=lim,col=col,
            zlab=expression(paste(delta^18,"O"[precip])),
            las=1,cex.lab=1.2) # adjust palette position with mai


par(mar=c(2.5,2.5,1,1),mgp=c(2,0.7,0)) # tighten margins
map('state',xlim=c(min(moderndata2$lon)-1,max(moderndata2$lon)),
    ylim=c(min(moderndata2$lat)-1,max(moderndata2$lat)))
for (i in 1:10) {  #add storm tracks
  points(lon.data[,i],lat.data[,i],col="blue",type="o",pch=1,cex=0.5)
}
points(lon_rain,lat_rain,pch=21,cex=0.7,
       bg=col[rescale(x=d18O_rain,xlow=lim[1],
                      xhigh=lim[2],rlow=1,rhigh=ncol)])
points(lon_snow,lat_snow,pch=22,cex=0.7,
       bg=col[rescale(x=d18O_snow,xlow=lim[1],
                      xhigh=lim[2],rlow=1,rhigh=ncol)])
points(lon_river,lat_river,pch=23,cex=0.7,
       bg=col[rescale(x=d18O_river,xlow=lim[1],
                      xhigh=lim[2],rlow=1,rhigh=ncol)])
points(lon_gw,lat_gw,pch=24,cex=0.7,
       bg=col[rescale(x=d18O_gw,xlow=lim[1],
                      xhigh=lim[2],rlow=1,rhigh=ncol)])
points(lon_lake,lat_lake,pch=25,cex=0.7,
       bg=col[rescale(x=d18O_lake,xlow=lim[1],
                      xhigh=lim[2],rlow=1,rhigh=ncol)])
points(lonis,latis,pch=16,col='black',cex=0.8)
points(proflons,proflats,pch=15,col='black',cex=0.8)
legend('bottomleft',pch=c(21,22,23,24,25,15,16),pt.bg='cyan',cex=0.7,bty='n',
       legend=c('rain','snow','rivers/creeks','groundwater','lakes','soil
profiles','coast points'))
dev.off()
```

```
###
#####porosity sensitivity analysis#####
####

#testing sensitivity of d18O, d13C of soil carbonate to soil porosity

point=which(distance.fake==576) #arbitrary point on storm track (19)

###OXYGEN
#run the code multiple times for multiple values of porosity
#oxygen not sensitive to MAP-Resp, so we're omitting here
#save the different values here

##BE CAREFUL not to overwrite these once already saved
#p=0.4
O18_p40=soilcarb18
O18_p40_eddy=soilcarb18_eddy
#p=0.5
O18_p50=soilcarb18
O18_p50_eddy=soilcarb18_eddy
#p=0.6
O18_p60=soilcarb18
O18_p60_eddy=soilcarb18_eddy

#pdf('O18 porosity sensitivity.pdf',width=15,height=7)
par(mfrow=c(1,3))
#all evaporation
plot(O18_p40[point,,1],zplot,yaxs='i',ylim=c(max(zplot),0),lwd=2,type='l',lty
=1,col='red',
     xlim=c(-25,0),xlab=expression(paste(delta^18,'O'[VPDB],'
(\u2030)')),ylab='depth (cm)',
     main='All Evaporation')
points(O18_p50[point,,1],zplot,lwd=2,type='l',lty=1,col='green')
points(O18_p60[point,,1],zplot,lwd=2,type='l',lty=1,col='blue')
legend('bottomright',legend=c('p=0.4','p=0.5','p=0.6'),col=c('red','green','b
lue'),lty=1)
#70% transpiration
plot(O18_p40[point,,8],zplot,yaxs='i',ylim=c(max(zplot),0),lwd=2,type='l',lty
=1,col='red',
     xlim=c(-25,0),xlab=expression(paste(delta^18,'O'[VPDB],'
(\u2030)')),ylab='depth (cm)',
     main='70% Transpiration')
points(O18_p50[point,,8],zplot,lwd=2,type='l',lty=1,col='green')
points(O18_p60[point,,8],zplot,lwd=2,type='l',lty=1,col='blue')
#all transpiration
plot(O18_p40[point,,11],zplot,yaxs='i',ylim=c(max(zplot),0),lwd=2,type='l',lt
y=1,col='red',
     xlim=c(-25,0),xlab=expression(paste(delta^18,'O'[VPDB],'
(\u2030)')),ylab='depth (cm)',
     main='All Transpiration')
points(O18_p50[point,,11],zplot,lwd=2,type='l',lty=1,col='green')
points(O18_p60[point,,11],zplot,lwd=2,type='l',lty=1,col='blue')
dev.off()


###CARBON
#like oxygen run the code multiple times for multiple values of porosity
#also save the different MAP-Resp relationships
#save the different values here

##BE CAREFUL not to overwrite these once already saved
#p=0.4
C13_p40.Cot12=soilCO2.1[,,3]
```

```
C13_p40.Cot13=soilCO2.2[,,3]
C13_p40.RS=soilCO2.3[,,3]
#p=0.5
C13_p50.Cot12=soilCO2.1[,,3]
C13_p50.Cot13=soilCO2.2[,,3]
C13_p50.RS=soilCO2.3[,,3]
#p=0.6
C13_p60.Cot12=soilCO2.1[,,3]
C13_p60.Cot13=soilCO2.2[,,3]
C13_p60.RS=soilCO2.3[,,3]

pdf('C13 porosity sensitivity.pdf',width=15,height=7)
par(mfrow=c(1,3))
#Cotton 2012
plot(C13_p40.Cot12[point,],zplot,yaxs='i',ylim=c(max(zplot),0),lwd=2,type='l'
,lty=1,col='red',
     xlim=c(-13,2),xlab=expression(paste(delta^13,'C'[VPDB],'
(\u2030)')),ylab='depth (cm)',
     main='Cotton 2012')
points(C13_p50.Cot12[point,],zplot,type='l',lwd=2,lty=1,col='green')
points(C13_p60.Cot12[point,],zplot,type='l',lwd=2,lty=1,col='blue')
legend('bottomright',legend=c('Cotton12','Cotton13','R&S'),col=c('red','green
','blue'),lty=1)
#Cotton 2013
plot(C13_p40.Cot13[point,],zplot,yaxs='i',ylim=c(max(zplot),0),lwd=2,type='l'
,lty=1,col='red',
     xlim=c(-13,2),xlab=expression(paste(delta^13,'C'[VPDB],'
(\u2030)')),ylab='depth (cm)',
     main='Cotton 2013')
points(C13_p50.Cot13[point,],zplot,type='l',lwd=2,lty=1,col='green')
points(C13_p60.Cot13[point,],zplot,type='l',lwd=2,lty=1,col='blue')
#Raich and Schlesinger 1992
plot(C13_p40.RS[point,],zplot,yaxs='i',ylim=c(max(zplot),0),lwd=2,type='l',lt
y=1,col='red',
     xlim=c(-13,2),xlab=expression(paste(delta^13,'C'[VPDB],'
(\u2030)')),ylab='depth (cm)',
     main='Raich and Scheslinger, 1992')
points(C13_p50.RS[point,],zplot,type='l',lwd=2,lty=1,col='green')
points(C13_p60.RS[point,],zplot,type='l',lwd=2,lty=1,col='blue')
dev.off()
```

D.2 “non-dimensional model.R”

```
#This script generates along-track plots of Dxs and 17xs
#Entirely theoretical

#code copied mostly from "big model.R"
#adapted from "model w evap balance and dxs.R" by Matt Winnick

#created Feb 4, 2016 by AJR

setwd('/Users/Annie/Documents/model/')
#plots will save here


#######MODEL PARAMETERS/VARIABLES#######
#change these to whatever you fancy

TempC=25 #surface temperature, degrees centigrade
TempK=TempC + 273.15 #deg K
delta18_atm_initial=-12 #initial atm. vapor d18O in VSMOW
delta18_precip_initial=-2   #initial precipitation d18O in VSMOW
```

```
rh=0.75; #relative humidity
e=seq(1,0, by = -0.2) ##evap as % of ET
t=1-e  ## transpiration as % of ET
track.length=41 #number of points along storm track

smow18=0.0020052 # smow ratio of 18O/16O
smowD=155.76*10^(-6) # smow ratio of D/H
smow17=379.9*10^(-6) # vsmow ratio of 17O/16O
theta.eq=0.529 #Barkan and Luz, 2005; equilibrium (l-v and v-l)
theta.diff=0.5185 #Barkan and Luz, 2007; diffusion
k18=1-0.01872 # kinetic fractionation factor for 18O, wet soils from Mathieu
and Bariac (1996), slightly higher than lake evap (~14.3)
kD=1-0.01676 # kinetic fractionation factor for dD, wet soils from Mathieu
and Bariac (1996), slightly higher than lake evap (~12.5)
k17=k18^theta.diff #Barkan and Luz, 2005, 2007


########necessary equations##############

##run these equations and load the data above, and then each section
#below can stand alone

##function to calculate either d18O (if d18 = True in input; default) or dD
(if d18 = False)
#of ET using Craig and Gordon equations and closure assumption
craig_gordon <- function(t, h, delta_precip, ts, d18 = T){
  if(d18 == T){
    alpha18=1 - (-7.685 + (6.7123*10^3)/(ts) - (1.6664*10^6)/(ts)^2 +
(0.35041*10^9)/(ts)^3)/1000 #equilibrium fractionation 18O, l-v
    r18_s <- (delta_precip/1000 + 1)*smow18  #ratio of precip sample
    Re_18 <- (1-t)*((alpha18*k18*r18_s)/(1-h+(1-t)*k18*h)) +
t*r18_s*(1/(1+(1-t)*k18*(h/(1-h))))
    d18_evap <- (Re_18/smow18 - 1)*1000
    (d18_evap - delta_precip)/1000}
  else{
    alphaD=1 - ((1158.8*ts^3)/10^9 - (1620.1*ts^2)/10^6 + (794.84*ts)/10^3 -
161.04 + (2.9992*10^9)/ts^3)/1000 #equilibrium fractionation D, l-v
    rD_s <- (delta_precip/1000 + 1)*smowD
    Re_D <- (1-t)*((alphaD*kD*rD_s)/(1-h+(1-t)*kD*h)) + t*rD_s*(1/(1+(1-
t)*kD*(h/(1-h))))
    dD_evap <- (Re_D/smowD - 1)*1000
    (dD_evap - delta_precip)/1000}
}

##function to calculate d17O of ET using Craig-Gordon and closure assumption
craig_gordon17=function(t,h,delta_precip,ts) {
  alpha18=1 - (-7.685 + (6.7123*10^3)/(ts) - (1.6664*10^6)/(ts)^2 +
(0.35041*10^9)/(ts)^3)/1000 #equilibrium fractionation 18O, l-v
  alpha17=alpha18^(theta.eq) #Barkan and Luz, 2005
  r17_s=(delta_precip/1000+1)*smow17
  Re_17=(1-t)*((alpha17*k17*r17_s)/(1-h+(1-t)*k17*h))+t*r17_s*(1/(1+(1-
t)*k17*(h/(1-h))))
  d17_evap=(Re_17/smow17-1)*1000
  (d17_evap-delta_precip)/1000
}


##function to calculate integrated isotopic composition of ET with
#integration of Rayleigh distillation for soil moisture reservoir
isotope_e = function(resid, delta_precip, eps_e){
  delt = -resid*((1000/(1+eps_e))*resid^(eps_e)-1000)
  delta_precip + delt
}
```

```
#######ATMOSPHERE COMPONENT########

## Set arrays of Nd, w, and x to iterate model over
nd <- c(seq(0.0025, 0.04, by = 0.0025),seq(0.05,1, by = 0.025), seq(1.1,10,
by = .1), seq(20,100, by = 10), seq(200,500, by = 50), 1100) #quasi-
logarithmic sequence of Nd's
w <- array(dim = c(track.length, length(nd)))
for(i in 1:length(nd)){
  w[,i] = seq(from = 1, by = -0.025*(1 - nd[i]/(nd[i]+1)), length.out =
track.length)
}
x <- 1-w ##build array for non-dimensional distance

ts <- TempK #surface temperature
#alpha <- 1 + (-7.685 + (6.7123*10^3)/(ts) - (1.6664*10^6)/(ts)^2 +
(0.35041*10^9)/(ts)^3)/1000 #equilibrium fractionation 18O, v-l
#aD <- 1 + ((1158.8*ts^3)/10^9 - (1620.1*ts^2)/10^6 + (794.84*ts)/10^3 -
161.04 + (2.9992*10^9)/ts^3)/1000 #equilibrium fractionation, D v-l
#constants for isotope value calculations:
#note that a18 != alpha18 (and same for D)
a18=1 + (-7.685 + (6.7123*10^3)/(ts) - (1.6664*10^6)/(ts)^2 +
(0.35041*10^9)/(ts)^3)/1000 #equilibrium fractionation 18O, v-l
aD=1 + ((1158.8*ts^3)/10^9 - (1620.1*ts^2)/10^6 + (794.84*ts)/10^3 - 161.04 +
(2.9992*10^9)/ts^3)/1000 #equilibrium fractionation, D v-l
alpha18=1 - (-7.685 + (6.7123*10^3)/(ts) - (1.6664*10^6)/(ts)^2 +
(0.35041*10^9)/(ts)^3)/1000 #equilibrium fractionation 18O, l-v
alphaD=1 - ((1158.8*ts^3)/10^9 - (1620.1*ts^2)/10^6 + (794.84*ts)/10^3 -
161.04 + (2.9992*10^9)/ts^3)/1000 #equilibrium fractionation D, l-v
alpha17=alpha18^(theta.eq) #Barkan and Luz, 2005
a17=a18^theta.eq

## build empty arrays to store data for d18O and dD
delta_a <- delta_et <- delta_inf<- delta_p <- delta_D_a <- delta_et_D <-
delta_D_inf <- delta_D_p <- delta_17_a <- delta_et_17 <- delta_17_inf <-
delta_17_p <- array(dim=c(length(w[,1]),length(nd), length(e)))
delta_a[1,,] <- delta18_atm_initial ## sets initial d18O atmospheric vapor
delta_p[1,,] <- delta18_precip_initial ## sets intial d18O precip value
exp <- a18+nd*a18-1 ## calculate exponent in Hendricks equation
delta_inf[1,,] <- (nd*(delta_p[1,,])-(1+nd)*(a18-1)*1000)/(a18+nd*a18-1) ##
calculate d18O_infinity parameter from Hendricks equation
delta_D_a[1,,] <- delta_a[1,,]*8+10 ## sets atmospheric vapor dD
#delta_d_p[1,,] <- delta_d_a[1,,] + (aD-1)*1000 ## sets intial dD precip
value
delta_D_p[1,,] <- delta_p[1,,]*8+10  #initial precip dD
exp_D <- aD+nd*aD-1 ## exponent of Hendricks for dD
delta_D_inf[1,,] <- (nd*(delta_D_p[1,,])-(1+nd)*(aD-1)*1000)/(aD+nd*aD-1) ##
calculate dD_infinity parameter from Hendricks equation
delta_17_a[1,,] = (exp(0.528*log(delta_a[1,,]/1000+1)+0.000033)-1)*1000
#GMWL, Luz and Barkan, 2010
delta_17_p[1,,] = (exp(0.528*log(delta_p[1,,]/1000+1)+0.000033)-1)*1000
#GMWL, Luz and Barkan, 2010
exp_17 <- a17+nd*a17-1 ## exponent of Hendricks for d17O
delta_17_inf[1,,] <- (nd*(delta_17_p[1,,])-(1+nd)*(a17-1)*1000)/(a17+nd*a17-
1) ## calculate dD_infinity parameter from Hendricks equation
rh <- rh ## this sets relative humidity

####
### RUN THE ATMOSPHERIC MODEL

## iterates over a storm track (w) for the arrays of e/t values and nd values
for(h in 1:length(e)){
  for(j in 1:length(nd)){
    for(i in 2:length(w[,1])){
      ## calculate residual soil moisture for isotope_e function
      residual <- 1-(nd[j]/(nd[j]+1))*e[h]/(1-(nd[j]/((nd[j]+1))*t[h]))
      ## if everything evaporates, set isotope value of evaporation to
isotope value of precip
      if(residual == 0) {
        delta_et[i,j,h] <- delta_p[i-1,j,h]
        delta_et_D[i,j,h] <- delta_D_p[i-1,j,h]
        delta_et_17[i,j,h] <- delta_17_p[i-1,j,h]
      ## otherwise, first calculate the combined equilibrium/kinetic
fractionation using craig-gordon function
      ## then calculate integrated ET value from location
      } else {
        eps_e <- craig_gordon(t[h], rh, delta_p[i-1,j,h],ts)
        eps_e_D <- craig_gordon(t[h], rh, delta_D_p[i-1,j,h],ts, d18 = F)
        eps_e_17 <- craig_gordon17(t[h], rh, delta_17_p[i-1,j,h],ts)
        delta_et[i,j,h] <- t[h]*delta_p[i-1,j,h] +
e[h]*isotope_e(residual,delta_p[i-1,j,h], eps_e)
        delta_et_D[i,j,h] <- t[h]*delta_D_p[i-1,j,h] +
e[h]*isotope_e(residual,delta_D_p[i-1,j,h], eps_e_D)
        delta_et_17[i,j,h] <- t[h]*delta_17_p[i-1,j,h] +
e[h]*isotope_e(residual,delta_17_p[i-1,j,h], eps_e_17)
        }

      ## Hendricks model
      delta_inf[i,j,h] <- (nd[j]*delta_et[i,j,h]-(1+nd[j])*(a18-
1)*1000)/(a18+nd[j]*a18-1)
      delta_a[i,j,h] <- (delta_a[1,j,h]-
delta_inf[i,j,h])*(w[i,j]^exp[j])+delta_inf[i,j,h]
      delta_p[i,j,h] <-delta_a[i,j,h]+(a18-1)*1000

      delta_D_inf[i,j,h] <- (nd[j]*delta_et_D[i,j,h]-(1+nd[j])*(aD-
1)*1000)/(aD+nd[j]*aD-1)
      delta_D_a[i,j,h] <- (delta_D_a[1,j,h]-
delta_D_inf[i,j,h])*(w[i,j]^exp_D[j])+delta_D_inf[i,j,h]
      delta_D_p[i,j,h] <-delta_D_a[i,j,h]+(aD-1)*1000

      delta_17_inf[i,j,h] <- (nd[j]*delta_et_17[i,j,h]-(1+nd[j])*(a17-
1)*1000)/(a17+nd[j]*a17-1)
      delta_17_a[i,j,h] <- (delta_17_a[1,j,h]-
delta_17_inf[i,j,h])*(w[i,j]^exp_17[j])+delta_17_inf[i,j,h]
      delta_17_p[i,j,h] <-delta_17_a[i,j,h]+(a17-1)*1000
    }}}

#calculate deuterium, 17O excesses from dD, d17O and d18O values
Dxs <- delta_D_p - 8*delta_p
O17xs <- (log(delta_17_p/1000+1)-0.528*log(delta_p/1000+1))*10^6 #in permeg


####plots!####

library(fields) #for tim.colors


###
####plots of precip d18O , 17xs, and Dxs with varying E/ET along storm track
###

#pick Nd=1
#now use all locations along storm track:

colors = tim.colors(n = length(e)) #for varying E/ETs at each location

pdf('Precipitation d18O, Dxs, and 17Oxs with varying E-T along storm
track.pdf',width=7,height=10)
par(mfrow=c(3,1))
#for d18O
plot(x[,which(nd==1)],delta_p[,which(nd==1),1],type='l',lwd=3,col=colors[1],x
lab='Non-Dimensional Distance (x)',
     ylab=expression(paste(delta^18,'O of precipitation (permil)')),
     main=expression(paste(delta^18,'O, D-excess, and 17O-excess of
precipitation along a storm track')))
for (i in 2:length(e)) {

points(x[,which(nd==1)],delta_p[,which(nd==1),i],type='l',lwd=3,col=colors[i]
)
}
legend('bottomleft',c('E=100%
ET','E=80%','E=60%','E=40%','E=20%','E=0%'),col=colors,lwd=3)

#for Dxs:
plot(x[,which(nd==1)],Dxs[,which(nd==1),1],type='l',lwd=3,col=colors[1],
     xlab='Non-Dimensional Distance (x)',ylab='Dxs of precipitation
(permil)')
for (i in 2:length(e)) {
  points(x[,which(nd==1)],Dxs[,which(nd==1),i],type='l',lwd=3,col=colors[i])
}
legend('topleft',c('E=100%
ET','E=80%','E=60%','E=40%','E=20%','E=0%'),col=colors,lwd=3)

#for 17xs:
plot(x[,which(nd==1)],O17xs[,which(nd==1),1],type='l',lwd=3,col=colors[1],
     xlab='Non-Dimensional Distance (x)',ylab='17Oxs of precipitation
(permeg)',
     ylim=c(-45,32))
for (i in 2:length(e)) {

points(x[,which(nd==1)],O17xs[,which(nd==1),i],type='l',lwd=3,col=colors[i])
}
legend('bottomleft',c('E=100%
ET','E=80%','E=60%','E=40%','E=20%','E=0%'),col=colors,lwd=3)
dev.off()
```

D.3 "Trajectory Plotter-NARR_AJR.R"

```
# Script to filter trajectories by precipitation and plot them on a map.
# Created on:  5/20/2015
# Last Modified:  3/15/16 by AJR
# BASED ON: "Precip plotter.R" by Jeremy Caves
# modified from: "Trajectory Plotter-NARR.R" (last mod. 11-8-15) by Jeremy
Caves

# This script takes HYSPLIT output, loads it into a list, which can then be
# analyzed for precipitation.  This code then plots this data as weighted by
# precipitation amount over the past hours. Because this script does not
alter
# the files, it works in the primary HYSPLIT output directory.  The code also
# allows one to plot all of the non-precipitating trajectories.  Currently,
# it is configured for plotting in the Western US.

# Notes:  Currently, this code cannot analyze the 31st day
#         of each month.

# Currently set-up for: Coso Basin
# Months: Jan-Dec
# Years: 1980-2014
# Reanalysis Dataset: NARR

library(maps)
library(gplots)
library(abind)
library(fields)
library(maptools)
library(ncdf)

#name the directories where HYSPLIT output files are located
wd1000='/Users/Annie/Desktop/HYSPLIT output/1000Verdi_1980.2014'
wd1500='/Users/Annie/Desktop/HYSPLIT output/1500Verdi_1980.2014'
wd2000='/Users/Annie/Desktop/HYSPLIT output/2000Verdi_1980.2014'

#name the filenames, excluding altitude prefix and date postscript
filename='verdi'

#number of altitudes being combined
naltitudes=3



#### Set up Time Vector ####
first.time=1980010100 # Starts January 1, 1980
years=35
year.interval=1000000 #Do not change
months=12
month.interval=10000 #Do not change
hour.interval=6
day.interval=100 # Do not change
year=numeric(length=years)
month=numeric(length=months)
hour=numeric(length=24/hour.interval)
days=array(dim=c(31,length(hour),length(month),years))
dim(days)

for(l in 1:years){
  year[l]=first.time+year.interval*(l-1)
  for (k in 1:length(month)){
    month[k]=year[l]+month.interval*(k-1)
    for (j in 1:length(hour)){
      hour[j]=month[k]+hour.interval*(j-1)
      for (i in 1:dim(days)[1]){
        days[i,j,k,l]=hour[j]+day.interval*(i-1)
      }
    }
  }
}

# days
days=days[-31,,,] #Removes 31st of each month
days=as.numeric(days)
length(days)
days=sort(days) #Sorts the days

# Eliminate 29 and 30 of all Februarys
days=days[-c((233:240)                    # Year 1
               ,((233+1440):(240+1440))     # 2
               ,((233+1440*2):(240+1440*2)) # 3
               ,((233+1440*3):(240+1440*3)) # 4
               ,((233+1440*4):(240+1440*4)) # 5
               ,((233+1440*5):(240+1440*5)) # 6
               ,((233+1440*6):(240+1440*6)) # 7
               ,((233+1440*7):(240+1440*7)) # 8
```

```
              ,((233+1440*8):(240+1440*8)) # 9
              ,((233+1440*9):(240+1440*9)) # 10
              ,((233+1440*10):(240+1440*10)) # 11
              ,((233+1440*11):(240+1440*11)) # 12
              ,((233+1440*12):(240+1440*12)) # 13
              ,((233+1440*13):(240+1440*13)) # 14
              ,((233+1440*14):(240+1440*14)) # 15
              ,((233+1440*15):(240+1440*15)) # 16
              ,((233+1440*16):(240+1440*16)) # 17
              ,((233+1440*17):(240+1440*17)) # 18
              ,((233+1440*18):(240+1440*18)) # 19
              ,((233+1440*19):(240+1440*19)) # 20
              ,((233+1440*20):(240+1440*20)) # 21
              ,((233+1440*21):(240+1440*21)) # 22
              ,((233+1440*22):(240+1440*22)) # 23
              ,((233+1440*23):(240+1440*23)) # 24
              ,((233+1440*24):(240+1440*24)) # 25
              ,((233+1440*25):(240+1440*25)) # 26
              ,((233+1440*26):(240+1440*26)) # 27
              ,((233+1440*27):(240+1440*27)) # 28
              ,((233+1440*28):(240+1440*28)) # 29
              ,((233+1440*29):(240+1440*29)) # 30
              ,((233+1440*30):(240+1440*30)) # 31
              ,((233+1440*31):(240+1440*31)) # 32
              ,((233+1440*32):(240+1440*32)) # 33
              ,((233+1440*33):(240+1440*33)) # 34
              ,((233+1440*34):(240+1440*34)) # 35
) # Need to add more rows if more years are added
]
length(days)


#### Load HYSPLIT trajectories into arrays ####
back.plot=24*7 #Number of back-hours to plot for the trajectory
met.variables=20 #Number of meteorological variables computed by HYSPLIT
jj1000=list() # Create list to store all trajectories at 1000m
jj1500=list() #list to store 1500m trajectories
jj2000=list() #list to store 2000m trajectories
runnumber=length(days) # days.run*length(hour)*length(month)*length(year)
for (i in 1:runnumber){
#
jj[[i]]=as.matrix(read.table(file=paste("1000wendover",days[i],sep=""),sep=""
,skip=6,nrows=back.plot,colClasses="numeric")) #"1000westwendover for GDAS
files"
#
jj[[i]]=as.matrix(read.table(file=paste("1000hay",days[i],sep=""),sep="",skip
=6,nrows=back.plot,colClasses="numeric")) # Hay
#
jj[[i]]=as.matrix(read.table(file=paste("1000swendover",days[i],sep=""),sep="
",skip=6,nrows=back.plot,colClasses="numeric")) # South of Wendover
#
jj[[i]]=as.matrix(read.table(file=paste("1000dubois",days[i],sep=""),sep="",s
kip=6,nrows=back.plot,colClasses="numeric")) # Dubois
#
jj[[i]]=as.matrix(read.table(file=paste("1000oupico",days[i],sep=""),sep="",s
kip=6,nrows=back.plot,colClasses="numeric")) # Oupico
#
jj[[i]]=as.matrix(read.table(file=paste("1000newark",days[i],sep=""),sep="",s
kip=6,nrows=back.plot)) # Newark Valley
#
jj[[i]]=as.matrix(read.table(file=paste("1000hay",days[i],sep=""),sep="",skip
=6,nrows=back.plot)) # Hay
#
jj[[i]]=as.matrix(read.table(file=paste("1000swendover",days[i],sep=""),sep="
",skip=6,nrows=back.plot)) # South of Wendover
#
jj[[i]]=as.matrix(read.table(file=paste("1000oupico",days[i],sep=""),sep="",s
kip=6,nrows=back.plot)) # Oupico
#
jj[[i]]=as.matrix(read.table(file=paste("1000pyramid",days[i],sep=""),sep="",
skip=6,nrows=back.plot)) # Pyramid Lake

  setwd(wd1000)

jj1000[[i]]=as.matrix(read.table(file=paste('1000',filename,days[i],sep=""),s
ep="",skip=6,nrows=back.plot))
  colnames(jj1000[[i]])=c("U","U","Year","Month","Day","Hour","U","U",
                       "Back.Hour","Lat","Long","AGL","Pressure","Temp",
                       "Rainfall","MixDepth","RH","SH","H2OMix","Terrain")
  setwd(wd1500)

jj1500[[i]]=as.matrix(read.table(file=paste('1500',filename,days[i],sep=""),s
ep="",skip=6,nrows=back.plot))
  colnames(jj1500[[i]])=c("U","U","Year","Month","Day","Hour","U","U",
                          "Back.Hour","Lat","Long","AGL","Pressure","Temp",
                          "Rainfall","MixDepth","RH","SH","H2OMix","Terrain")
  setwd(wd2000)

jj2000[[i]]=as.matrix(read.table(file=paste('2000',filename,days[i],sep=""),s
ep="",skip=6,nrows=back.plot))
  colnames(jj2000[[i]])=c("U","U","Year","Month","Day","Hour","U","U",
                         "Back.Hour","Lat","Long","AGL","Pressure","Temp",
                         "Rainfall","MixDepth","RH","SH","H2OMix","Terrain")
}
# for (i in 1:runnumber){
#   colnames(jj[[i]])=c("U","U","Year","Month","Day","Hour","U","U",
#                       "Back.Hour","Lat","Long","AGL","Pressure","Temp",
#                       "Rainfall","MixDepth","RH","SH","H2OMix","Terrain")
# }

##sometimes the file doesn't read if there's nothing there
 #skip those days and just keep adding on to jj list:
###ONLY DO THIS IF FIRST RUN ABOVE DIDN'T COMPLETE
newstart=i  #CHANGE FOR NEW START NUMBER
for (i in (newstart+1):runnumber) {
  setwd(wd1000)

jj1000[[i]]=as.matrix(read.table(file=paste('1000',filename,days[i],sep=""),s
ep="",skip=6,nrows=back.plot))
  colnames(jj1000[[i]])=c("U","U","Year","Month","Day","Hour","U","U",
                         "Back.Hour","Lat","Long","AGL","Pressure","Temp",
                         "Rainfall","MixDepth","RH","SH","H2OMix","Terrain")
  setwd(wd1500)

jj1500[[i]]=as.matrix(read.table(file=paste('1500',filename,days[i],sep=""),s
ep="",skip=6,nrows=back.plot))
  colnames(jj1500[[i]])=c("U","U","Year","Month","Day","Hour","U","U",
                          "Back.Hour","Lat","Long","AGL","Pressure","Temp",
                          "Rainfall","MixDepth","RH","SH","H2OMix","Terrain")
  setwd(wd2000)

jj2000[[i]]=as.matrix(read.table(file=paste('2000',filename,days[i],sep=""),s
ep="",skip=6,nrows=back.plot))
  colnames(jj2000[[i]])=c("U","U","Year","Month","Day","Hour","U","U",
                         "Back.Hour","Lat","Long","AGL","Pressure","Temp",
                         "Rainfall","MixDepth","RH","SH","H2OMix","Terrain")
}
```

```
maxhours1000=numeric(length=runnumber)
maxhours1500=numeric(length=runnumber)
maxhours2000=numeric(length=runnumber)
for (i in 1:runnumber){
  #maxhours[i]=dim(jj[[i]])[1]
  maxhours1000[i]=ifelse(length(dim(jj1000[[i]]))==0,0,dim(jj1000[[i]])[1])
  maxhours1500[i]=ifelse(length(dim(jj1000[[i]]))==0,0,dim(jj1000[[i]])[1])
  maxhours2000[i]=ifelse(length(dim(jj2000[[i]]))==0,0,dim(jj2000[[i]])[1])
  #^this makes it zero hours if there's no HYSPLIT output (like for Newark
32097)
}
max(maxhours1000)
max(maxhours1500)
max(maxhours2000)

###COMBINE SEPARATE ALTITUDES INTO ONE LIST
jj=c(jj1000,jj1500,jj2000)


#### Create index vectors for precip and non-precip trajectories ####
back.hour=6 #This is the time over which to calculate precipitation
precip=numeric(length=runnumber*naltitudes)
for (i in 1:length(precip)){
  if (sum(as.numeric(jj[[i]][1:back.hour,15])>0))
{precip[i]=sum(as.numeric(jj[[i]][1:back.hour,15]))}
  else {precip[i]=0}
}
# precip
precip.day=which(precip>0)
noprecip=which(precip==0)
length(precip.day)
length(noprecip)
length(precip.day)+length(noprecip)

#### Separate into precipitating vs. non-precipitating trajectories ####
# Precipitating
jj.coord=list()
for (i in 1:length(precip.day)){
  jj.coord[[i]]=jj[[precip.day[i]]][,c(3:6,9:20)]
}
jj.coord[[1]][1:10,]

# Non-precipitating
jj.noprecip=list()
for (i in 1:length(noprecip)){
  jj.noprecip[[i]]=jj[[noprecip[i]]][,c(3:6,9:20)]
}

#### Eliminate positive longitudes (ie, past the dateline) ####
# Only does this for precipitating trajectories
for (i in 1:length(precip.day)){
  temptraj=as.data.frame(jj.coord[[i]])
  temptraj.Long=numeric(length=nrow(temptraj))
  for (j in 1:nrow(temptraj)){
    temptraj$Long[j]=ifelse(temptraj$Long[j]>0,NA,temptraj$Long[j])
    temptraj2=cbind(temptraj,temptraj.Long)
    temptraj2=na.omit(temptraj)
    temptraj2=temptraj2[,-21]
  }
  jj.coord[[i]]=temptraj2
}

#### Partition by Season ####
Month.vector=numeric(length=length(precip.day))
```

```
for (i in 1:length(precip.day)){
  Month.vector[i]=jj.coord[[i]]$Month[1]
}
length(Month.vector)
MAM.index=which(Month.vector>=3 & Month.vector<=5)
length(MAM.index)
JJA.index=which(Month.vector>=6 & Month.vector<=8)
length(JJA.index)
SON.index=which(Month.vector>=9 & Month.vector<=11)
length(SON.index)
D.index=which(Month.vector>=12)
JF.index=which(Month.vector<=2)
DJF.index=sort(c(D.index,JF.index))
length(DJF.index)
#NOTE some months are of value NA so the length totals don't necessarily
match

MAM.traj=list()
for (i in 1:length(MAM.index)){
  MAM.traj[[i]]=jj.coord[[MAM.index[i]]]
}

JJA.traj=list()
for (i in 1:length(JJA.index)){
  JJA.traj[[i]]=jj.coord[[JJA.index[i]]]
}

SON.traj=list()
for (i in 1:length(SON.index)){
  SON.traj[[i]]=jj.coord[[SON.index[i]]]
}

DJF.traj=list()
for (i in 1:length(DJF.index)){
  DJF.traj[[i]]=jj.coord[[DJF.index[i]]]
}


jj.coord=jj.coord[which(is.na(Month.vector)==F)]
#^this just removes the ones with NA month values....

#### Plot all trajectories that produce precipitation over the site ####

# All Precipitating Trajectories
par(mfrow=c(1,1))
map(database="state",xlim=c(-130,-90),ylim=c(30,50))
for (i in 1:length(precip.day)){
  lines(jj.coord[[i]][,7],jj.coord[[i]][,6],lwd=0.1)
}
#points(locations$Long,locations$Lat,col="red",pch=16)
points(jj.coord[[1]][1,7],jj.coord[[1]][1,6],col="blue",pch=16)

# DJF Trajectories
par(mfrow=c(1,1))
map(database="state",xlim=c(-130,-90),ylim=c(30,50))
for (i in 1:length(DJF.index)){
  lines(DJF.traj[[i]][,7],DJF.traj[[i]][,6],lwd=0.1)
}
points(DJF.traj[[1]][1,7],DJF.traj[[1]][1,6],col="blue",pch=16)

# MAM Trajectories
par(mfrow=c(1,1))
map(database="state",xlim=c(-130,-90),ylim=c(30,50))
for (i in 1:length(MAM.index)){
  lines(MAM.traj[[i]][,7],MAM.traj[[i]][,6],lwd=0.1)
}
points(MAM.traj[[1]][1,7],MAM.traj[[1]][1,6],col="blue",pch=16)

# JJA Trajectories
par(mfrow=c(1,1))
map(database="state",xlim=c(-130,-90),ylim=c(30,50))
for (i in 1:length(JJA.index)){
  lines(JJA.traj[[i]][,7],JJA.traj[[i]][,6],lwd=0.1)
}
points(JJA.traj[[1]][1,7],JJA.traj[[1]][1,6],col="blue",pch=16)

# SON Trajectories
par(mfrow=c(1,1))
map(database="state",xlim=c(-130,-90),ylim=c(30,50))
for (i in 1:length(SON.index)){
  lines(SON.traj[[i]][,7],SON.traj[[i]][,6],lwd=0.1)
}
points(SON.traj[[1]][1,7],SON.traj[[1]][1,6],col="blue",pch=16)

#### Plot all trajectories that don't produce precipitation ####
# jj.noprecip=list()
# for (i in 1:length(noprecip)){
#   jj.noprecip[[i]]=jj[[noprecip[i]]][,c(3:6,9:20)]
# }

map(database="state",xlim=c(-130,-90),ylim=c(30,50))
for (i in 1:length(noprecip)){
  lines(jj.noprecip[[i]][,7],jj.noprecip[[i]][,6],lwd=0.1)
}
points(jj.coord[[1]][1,7],jj.coord[[1]][1,6],col="blue",pch=16)

#### Write precipitation and non-precipitation lists to file ####
#setwd("/Users/jeremycaves/Desktop/Box Sync/EESS 300 - Spring 2015
Terrestrial Paleoclimate/HYSPLIT/R Output")
#setwd("/Users/jeremycaves/Desktop/Box Sync/SoCo 2015/Guidebook/Dubois
HYSPLIT/R and NCL Files")
setwd('/Users/Annie/Desktop/HYSPLIT output/RData files/Hari')
#CHANGE NAMES AS APPROPRIATE!!!!
Verdi.precip=jj.coord
Verdi.MAM=MAM.traj
Verdi.JJA=JJA.traj
Verdi.SON=SON.traj
Verdi.DJF=DJF.traj
Verdi.noprecip=jj.noprecip
save(Verdi.precip,file="verdi.precip_annual.RData")
save(Verdi.MAM,file="verdi.MAM.precip.RData")
save(Verdi.JJA,file="verdi.JJA.precip.RData")
save(Verdi.SON,file="verdi.SON.precip.RData")
save(Verdi.DJF,file="verdi.DJF.precip.RData")
save(Verdi.noprecip,file="verdi.noprecip.RData")

# Load any presaved files
#load("Coso.SON.precip.RData")

#### Write trajectories to a file ####
# NEED TO FIX THESE NEXT LINES OF CODE
#setwd("/Users/jeremycaves/Desktop/Assignments/Mongolia/HYSPLIT/Results/Conto
ur Figure")

#Adds storm track ID to each storm track
# Currently set to only keep lat and long and add storm track ID
traj.analyze=SON.traj #jj.coord #SON.traj
jj.coord.st=list()
```

```
for (i in 1:length(traj.analyze)){

ST=matrix(rep(i,length(traj.analyze[[i]][,5])),ncol=1,nrow=length(traj.analyz
e[[i]][,5]))
  jj.coord.st[[i]]=cbind((traj.analyze[[i]][,6:7]),ST)
}

st=as.matrix(jj.coord.st[[1]])
for (i in 2:length(jj.coord.st)){
  st=rbind(st,jj.coord.st[[i]])
}
dim(st)

# write.table(jj.coord.st[[1]][,],file="swendover.traj.txt",
#              row.names=FALSE,col.names=FALSE,sep=",")
# for (i in 2:length(precip.day)){
#   write.table(jj.coord.st[[i]][,],file="swendover.traj.txt",append=TRUE,
#                col.names=FALSE,row.names=FALSE,sep=",")
# }

#Should fix this step sometime in the future
# big <- read.table('swendover.traj.txt',sep = ',', header = FALSE)
big=st
# big=as.data.frame(big)
colnames(big) = c("Lat","Long","ST")

##CREATE AN EMPTY GRID TO STORE HISTOGRAM DATA
grid_p <- array(0, dim = c(180*2-1,90*2-1))
grid_n <- array(0, dim = c(180*2-1,90*2-1))

for(i in 1:max(big$ST)){
  # MAKE ARRAY OF COORDINATES
  coords <- big[which(big$ST == i),1:2]
  # BIN COORDINATES INTO 0.5X0.5 DEGREE GRID
  coords <- floor(coords*2)/2
  # REMOVE DUPLICATES SO THAT A GRID SPACE CAN ONLY BE COUNTED ONCE PER STORM
TRACK
  coords<-unique(coords)
  # RECORD HISTOGRAM COUNT IN THE GRID, SEPERATE MATRICIES FOR NEGATIVE AND
POSITIVE LON'S
  for(k in 1:dim(coords)[1]){
    if(coords[k,2]<0){
      grid_n[abs(coords[k,2]*2)-1,coords[k,1]*2-1] <-
grid_n[abs(coords[k,2]*2)-1,coords[k,1]*2-1] + 1}
    else{
      grid_p[coords[k,2]*2-1,coords[k,1]*2-1] <- grid_p[coords[k,2]*2-
1,coords[k,1]*2-1] + 1}
  }
}

##CONVERT GRID TO PERCENTAGES
grid_p <- grid_p/max(big$ST)
grid_n <- grid_n/max(big$ST)
grid_all <- rbind(grid_n[359:1,], grid_p)

##SET CONTOUR LEVELS
levels = c(1,.7,.5,.4,.3,.2,.15, 0.1, .05,.025)
#levels=c(625,300,150,75,40,20,10,5)
colors = c('black', 'violet', 'purple', 'blue','cyan', 'green', 'yellow',
'orange', 'red',"darkred")
#colors=rainbow(length(levels))

##PLOT BACKGROUND MAP
map('state',interior=TRUE)#,xlim =c(75,120),ylim=c(20,60),col='gray')
```

```
##PLOT CONTOUR LINES
contour(c(seq(-180, -1, by = 0.5),seq(1, 180, by = 0.5)), seq(1, 90, by =
0.5),
        grid_all, add = TRUE,
        levels = levels,
        col = colors,
        drawlabels = FALSE)

##MAKE NetCDF FILE OF HISTOGRAM GRID FOR NCL MAPPING
x <- dim.def.ncdf("lon", "degrees_east",
                  c(seq(-180, -1, by = 0.5),seq(1, 180, by = 0.5)))
y <- dim.def.ncdf("lat", "degrees_north", seq(1,90, by = 0.5))

storm <- var.def.ncdf("storm", "percent", list(x,y), -1)

histogram <- create.ncdf("histverdi.SON.nc", storm) #CHANGE
put.var.ncdf(histogram, storm, grid_all)
close.ncdf(histogram)

#### Plot Specific Humidity ####
plot(jj.coord[[1]][,10]~jj.coord[[1]][,1],type="l",xlab="Back Hour",
     ylab="Specific Humidity",bty="n",ylim=c(0,15)
     #,xlim=c(40,120)
)
for (i in 2:length(precip.day)){
  lines(jj.coord[[i]][,10]~jj.coord[[i]][,1])
}
abline(v=jj.coord[[1]][1,3],lty="dashed")

meanSH=numeric(length=back.plot)
meanSHarray=array(dim=c(back.plot,length(precip.day)))
for (i in 1:back.plot){
  for (j in 1:length(precip.day)){
    meanSHarray[i,j]=jj.coord[[j]][i,10]
  }
}
for (i in 1:back.plot){
  meanSH[i]=mean(meanSHarray[i,])
}
length(meanSH)
meanSH=cbind(jj.coord[[1]][,1],meanSH)
colnames(meanSH)=c("Back.Hour","SH")
lines(x=meanSH[,1],y=meanSH[,2],lwd=5,col="blue")

#Writes all non-precipitation producing trajectories to a csv file (for
Tseterleg)
#Adds ID to each trajectory
# Currently set to only keep lat and long and add trajectory ID
jj.coord.st=array(dim=c(back.plot,3,length(noprecip)))
for (i in 1:length(noprecip)){
  ST=matrix(rep(i,back.plot),ncol=1,nrow=back.plot)
  jj.coord.st[,,i]=cbind(jj.noprecip[,2:3,i],ST)
}

#This for-loop writes all trajectories to a single file
write.table(jj.coord.st[1:back.plot,,1],file="tlgalltrajnoprecip.txt",
            row.names=FALSE,col.names=FALSE,sep=",")
for (i in 2:length(noprecip)){

write.table(jj.coord.st[1:back.plot,,i],file="tlgalltrajnoprecip.txt",append=
TRUE,
              col.names=FALSE,row.names=FALSE,sep=",")
}
```

```
#This for-loop writes each trajectory to a different csv file
for (i in 1:length(precip.day)){
  write.table(jj.coord[1:back.plot,,i],file=paste("tg",i,sep=""),sep=",",
              row.names=FALSE,quote=FALSE)
}

#Turns all of the trajectories into a single matrix
alltraj=matrix(ncol=2,nrow=(back.plot*dim(jj.coord)[3]))
s=seq(from=1,to=(back.plot*dim(jj.coord)[3]),by=back.plot)
e=seq(from=back.plot,to=(back.plot*dim(jj.coord)[3]),by=back.plot)
for (i in 1:dim(jj.coord)[3]){
  alltraj[s[i]:e[i],]=jj.coord[,2:3,i]
}
colnames(alltraj)=c("Lat","Long")
#write.table(alltraj,file="dzeregalltraj.txt",sep=",",row.names=FALSE,col.nam
es=FALSE) #Writes "alltraj" to a .txt file
```

D.4 "dataset processing.R"

```
#   in the vapor transport model

#loops over all the desired years
#outputs .RData files for NARR data required for that year

#created 10-20-15 by AJR
#last modified 11-27-15 by AJR


library(ncdf)

#switch to directory with the NARR netcdf files
setwd("/Users/Annie/Desktop/NARR_for_processing")

###Import datasets!###

#land mask
ncdf = open.ncdf("land.nc") #
land.data=get.var.ncdf(nc=ncdf,varid="land") #

#latitude/longitude
lat.data=get.var.ncdf(nc=ncdf,varid="lat")
lon.data=get.var.ncdf(nc=ncdf,varid='lon')

save(land.data,lat.data,lon.data,file='NARRgriddata.RData')
rm(land.data,lat.data,lon.data)
#clear these out so I can just save the image later!


years=seq(2005,2014,1) #ADJUST FOR YEARS DESIRED

#loop and save over each year!!!!

#for each year, save an RData file containing
 #model input climatological variables
for (year in years) {

  #Air Temperature at 2m (Kelvin)
  filename=paste('air.2m.',year,'.nc',sep='')
  dataname=paste('Temp_2m.',year,sep='')
  ncdf=open.ncdf(filename)
  assign(dataname,get.var.ncdf(nc=ncdf,varid='air'))  #Kelvin
  #Precipitable water (w) (kg/m^2)
  filename=paste('pr_wtr.',year,'.nc',sep='')
  dataname=paste('Prw.',year,sep='')
  ncdf=open.ncdf(filename)
  assign(dataname,get.var.ncdf(nc=ncdf,varid='pr_wtr'))  #kg/m2


  # Relative humidity (rhum) #% <-PERCENT!
  filename=paste('rhum.2m.',year,'.nc',sep='')
  dataname=paste('rhum.',year,sep='')
  ncdf=open.ncdf(filename)
  assign(dataname,get.var.ncdf(nc=ncdf,varid='rhum'))  #PERCENT!!!


  # 3-hourly accumulated evaporation total (evap) # kg/m^2
  filename=paste('evap.',year,'.nc',sep='')
  #dataname=paste('evap.',year,sep='')
  ncdf=open.ncdf(filename)
  #assign(dataname,get.var.ncdf(nc=ncdf,varid='evap')) #kg/m2/(3hrs) which is
mm/(3hrs)
  evap=get.var.ncdf(nc=ncdf,varid='evap') #kg/m2/(3hrs) which is mm/(3hrs)
  #don't need to save evap.year file; only need for Damkohler


  # Precipitation rate (prate) #kg/m2/(3hrs)
  filename=paste('prate.',year,'.nc',sep='')
  #dataname=paste('Pratemm.',year,sep='')
  ncdf=open.ncdf(filename)
  #assign(dataname,(get.var.ncdf(nc=ncdf,varid='prate'))*(3600*3))
#kg/m2/(3hrs) which is mm/(3hrs)
  precip=(get.var.ncdf(nc=ncdf,varid='prate'))*(3600*3) #kg/m2/(3hrs) which
is mm/(3hrs)
  #we don't actually have to save a year prate file; only needed for
Damkohler


  # Time (hours since 1800-1-1 00:00:0.0)
  #filename=paste('prate.',year,'.nc',sep='') #commented out because
continuation from prate above
  dataname=paste('time.',year,sep='')
  #ncdf=open.ncdf(filename) #commented out because continuation from prate
above
  assign(dataname,get.var.ncdf(nc=ncdf,varid='time')) #hours since 1800-1-1
00:00:0.0


  #Damkohler number (evap/(p-evap)) #optimal format determined in cees
RStudio
  dataname=paste('Nd.',year,sep='')
  assign(dataname,(evap/(precip-evap)))


  ###save the .RData file for each year!
  #create name of file we're saving to
  savename=paste('NARR',year,'.RData',sep='')
  #remove the things we don't want to save in the year file
  rm(dataname,evap,filename,ncdf,precip)

  #save everything that contains the year (variable.year)
  save(list=ls()[grepl(year,ls())],file=savename)

  #remove all the files from that year so that they're not double-saved!
  rm(list=ls()[grepl(year,ls())])

}
```

D.5 "timechanger.R"

```
#timechanger function
#converts Year/Month/Day/Hour to Hours since 1800-1-1 00:00:00.0

#created 10-22-15 by AJR
#last modified 11-10-15



#takes in dataframe of precipitating trajectories and their met. variables
#dataframe has 16 columns; add a 17th with the new time!
#also adds a new top row of empties for the initialization point
# ^this is currently commented out as it is unnecessary
timechanger=function(df) {
  vec=df[1,1:4]
  backhours=df$Back.Hour
  #vec=c(Year,Month,Day,Hour)
  Year=as.numeric(vec[1])
  Month=as.numeric(vec[2])
  Day=as.numeric(vec[3])
  Hour=as.numeric(vec[4])
  #Year is in years after 1900/2000 (ie, 2004 has Year=4, 1980 has Year=80)
  Y2K=0 #year 2000
  Y79=79 #year 1979


  #calculate number of hours since a known reference
  #must execute as either before or after year 2000
  if (Year<16) { #Execute this part for years after (and including) 2000
    #establish whether or not it's a leap year
    yeardiff=Year-Y2K #difference between Year and 2000
    remainder=yeardiff%%4
    Leap=F #default not a leap year
    if (remainder==0) Leap=T  #leap year True if ....if it's a leap year...
:P


    #how many leap years have passed between Year and 2000? (2000 counts as
one)
    yeardiv=(yeardiff-1)/4
    nleapyears=1+floor(yeardiv) #add one to include 2000 in the count

    #add on hours from 2000-01-01 to YEAR-01-01 (at midnight)
    Y2K.hours=1753152
    #number of hours at Jan 1, midnight, of year at hand
    Yearhours.Jan1=Y2K.hours+(yeardiff*365*24)+(nleapyears*24)

  } else { #Execute this part for years before 2000
    #establish whether or not it's a leap year
    yeardiff=Year-Y79 #number of years between Year and 1979
    remainder=yeardiff%%4
    Leap=F
    if (remainder==0) Leap=T #indicates leap year TRUE

    #how many leap years have passed between Year and 1979??
    yeardiv=(yeardiff-1)/4
    yeardiv=ifelse(yeardiv>0,yeardiv,0)
    nleapyears=ceiling(yeardiv) #it's magic. it works. don't question.
    #add on hours from 1979-01-01 to YEAR-01-01 (at midnight)
    Y79.hours=1569072
    #number of hours at Jan 1, midnight, of year at hand
    Yearhours.Jan1=Y79.hours+(yeardiff*365*24)+(nleapyears*24)
  }



  #add on hours until midnight of first day of each month
  #months are all different, so just do it by month, I guess

  JanHours=31*24
  FebHours=28*24
  FebHours.leap=29*24
  MarHours=31*24
  AprHours=30*24
  MayHours=31*24
  JunHours=30*24
  JulHours=31*24
  AugHours=31*24
  SepHours=30*24
  OctHours=31*24
  NovHours=30*24

  MonthHoursVec=c(0,JanHours,FebHours,MarHours,AprHours,MayHours,JunHours,
                  JulHours,AugHours,SepHours,OctHours,NovHours)

MonthHoursVec.leap=c(0,JanHours,FebHours.leap,MarHours,AprHours,MayHours,JunH
ours,
                       JulHours,AugHours,SepHours,OctHours,NovHours)
  #^zeros are placeholders because there are no hours before start of Jan

  #hours until beginning of first day of the month, depending on leap year
  MonthHours=sum(MonthHoursVec[1:Month])
  if (Leap==T) { #if current year is a leap year
    MonthHours=sum(MonthHoursVec.leap[1:Month])
  }

  #add on hours until start of the day
  DayHours=(Day-1)*24

  #add on hours until start of the hour
  HourHours=Hour

  #add them all together!
  #this is hour since 1800-01-01 00:00:00.0 at our given point
  PointHour=sum(Yearhours.Jan1,MonthHours,DayHours,HourHours)


  #okay, we've got the hour for our first point along trajectory.
  #now add an additional column to the d.f. for the PointHour (call it
"Time")
  Time=PointHour+backhours   #backhours MUST start with a zero!!!! (at
initialization point)
  #newdf=cbind(df,Time) #17 met variables total now, with Time being #17
  #keeping everything is making gigantic lists. trim unnecessary junk:
  newdf=cbind(df[c('Year','Back.Hour','Lat','Long')],Time)

  #commenting this out unless initialization point isn't included for some
reason
  # #while we're manipulating the dataframe, add the initialization point
  # #add a first column for Back.Hour=0, with appropriate Time
  # #everything else (Lat/Long, most importantly) can be filled in later
  # toprow=c(rep(NA,4),0,rep(NA,11),(PointHour+1))
  # newdf2=rbind(toprow,newdf)

  #awesome!
  #return(newdf2) #only if initialization point isn't included from
TrajectoryPlotter
  return(newdf)

}
```

D.6 "latlonNARR.R"

```
#latlonNARR function
#function to convert HYSPLIT lat/lons to NARR grid

#created 10-22-15 by AJR
#last modified 12-5-15 by AJR

#this function will take in a df with lat/lon and spit out a df with NARR-
friendly lat/lon
#where "NARR-friendly" means its regridded to the NARR grid

library(RANN) #for nearest neighbor search
library(ncdf) #to get lat/lons from file
library(dplyr) #for data structure manipulation

#load NARR coordinates
#setwd("/Users/Annie/Documents/300/HYSPLIT/")
#load('NARRgriddata.RData')

#necessary for function!
NARRlat=as.numeric(lat.data)
NARRlon=as.numeric(lon.data)
NARRlatlon=cbind(NARRlat,NARRlon) #has negatives for lon  #96673x2


#this function needs to run AFTER timechanger!!!!
#^because timechanger adds the extra first row for initialization
  #and we'll want to NARR-ify the initialization coords, too!
latlonNARR=function(df) {
  raw.latlon=cbind(df$Lat,df$Long)

  #nearest neighbor search between two coordinate sets
  nearest=nn2(NARRlatlon,raw.latlon,k=1) #k=1 to get just one nearest
neighbor
  #^the output length is the same as the second variable in the nn2 function
  nn.idx=nearest[[1]] #one-column vector of the indices (row) of the nearest
neighbor in NARRlatlon
  #nn.dists=nearest[[2]] #same, but for distance from traj.latlon to nearest
neighbor in NARRlatlon

  newcoords=NARRlatlon[nn.idx,] #extract NARR coordinates
  df$Lat=newcoords[,1] #replace coordinates in the dataframe
  df$Long=newcoords[,2]
  names(df)[4]='Lon'  #switching name to my preferable version

  #spit out new df
  return(df)
}
```

D.7 "NARRgrabber.R"

```
#grab NARR variables

#this function takes (modified) HYSPLIT output, regridded to NARR grid, and
  #tacks on the associated NARR meteorological variables to the dataframe
#the following meteorological variables are added:
    #Air Temperature at 2m (Temp_2m.year) (Kelvin)
    #Relative Humidity at 2m (rhum.year)(kg/kg)
    #Precipitable Water (Prw.year) (kg/m^2)
    #Time (time.year) (hours since 1800-1-1 00:00:0.0)
    #Damkohler Number (evap/(prate-evap))

#created: 11-4-15 by AJR
#last modified: 11-10-15 by AJR



#CHANGE WORKING DIRECTORY to the one with the NARR .RData files!
#setwd('/Users/Annie/Desktop/NARR_for_processing')



#begin with function for one row at a time; this function is called later in
NARRgrabber()
#this function uses the lat/lon/time of a row of a df and picks out
appropriate NARR data
NARRgrabber.onerow=function(vector) {
  #year.pre=vector$Year
  year.pre=as.numeric(vector['Year'])
  year=ifelse(year.pre<16,year.pre+2000,year.pre+1900) #turn into an actual
year format
  #^these lines repeat in NARRgrabber() as well. Here we need for individual
year
     #in other function we need just to download appropriate files

  #select the indices we'll be using to dig out NARR data using lat/lons
  #dlati=which(lat.data==vector$Lat,arr.ind=T) #desired lat index (dlati)
  dlati=which(lat.data==as.numeric(vector['Lat']),arr.ind=T) #desired lat
index (dlati)
  #dloni=which(lon.data==vector$Lon,arr.ind=T) #desired lon index (dloni)
  dloni=which(lon.data==as.numeric(vector['Lon']),arr.ind=T) #desired lon
index (dloni)
  #^these give vectors where indices correspond to indices in lat.data and
lon.data
  #ie lat.data[dlati[1],dlati[2]] returns desired lat VALUE

  #there are non-unique lat/lons!
  #length(as.numeric(lat.data)) #96673
  #length(unique(as.numeric(lat.data))) #96130
  #length(unique(as.numeric(lon.data))) #95662

  #desired coordinate indices because there are non-unique lats and lons
  #keep only the indices present in lat and lon (keep only duplicates)
  coordis=rbind(dlati,dloni)
  dcoordis=coordis[duplicated(coordis),,drop=F] #desired coordinate indices
(dcoordis)


  yearfilelist=ls(envir=.GlobalEnv)[grepl(year,ls(envir=.GlobalEnv))] #list
of all the files for given year
  yeartime=get(yearfilelist[grepl("time",yearfilelist)],envir=.GlobalEnv)
#list of times for given year
```

```
  #get desired time index
  #dtimei=which(yeartime==vector$Time)
  dtimei=which(yeartime==as.numeric(vector['Time']))
  #dtimei=which(yeartime==1796997)

  #NARR is every three hours, so 2/3 of our HYSPLIT hours aren't extractable
  #for those that are extractable, pick out met values using associated
lat/lon/time indices
  if (length(dtimei)==1) { #if NOT compatible, dtimei is length zero
    #extract from NARR files using desired lat/lon/time indices

vector['Nd']=get(yearfilelist[grepl('Nd',yearfilelist)])[dcoordis[1],dcoordis
[2],dtimei]

vector['Prw']=get(yearfilelist[grepl('Prw',yearfilelist)])[dcoordis[1],dcoord
is[2],dtimei]

vector['rhum']=get(yearfilelist[grepl('rhum',yearfilelist)])[dcoordis[1],dcoo
rdis[2],dtimei]

vector['Temp_2m']=get(yearfilelist[grepl('Temp_2m',yearfilelist)])[dcoordis[1
],dcoordis[2],dtimei]
  } #otherwise it all remains as NAs!

  #pick out the row's land mask from desired coord indices:
  vector['Land']=land.data[dcoordis[1],dcoordis[2]]

  return(vector)

}




#take in dataframe (with fixed coords and time), append appropriate NARR
output variables
NARRgrabber=function(df) {
  #add five extra columns to the df for all of the synced info!!!!
  #also, make it a matrix so that it's compatible with apply()
  dfnew=cbind(df,Land=NA,Nd=NA,Prw=NA,rhum=NA,Temp_2m=NA)
  #^leaving as NAs; the non-three-hourly times will remain NAs

  ###bookkeeping of necessary NARR data files
  #THIS IS CRITICAL so as to not exceed RAM space

  year.pre=df[1,]$Year #get year of initialization point
  year=ifelse(year.pre<16,year.pre+2000,year.pre+1900) #turn into an actual
year format
  openname=paste('NARR',year,'.RData',sep='')
  #^this is for the year of the initialization point
  #the trajectory may extend into previous year, so load that up too!
  year.prev=year-1
  openname.prev=paste('NARR',year.prev,'.RData',sep='')

  #Now, we don't want to have too much gunk floating around in our
environment!
  #get rid of variables from years past that we no longer need
  year.twoago=year-2

rm(list=ls(envir=.GlobalEnv)[grepl(year.twoago,ls(envir=.GlobalEnv))],envir=.
GlobalEnv)
  #^removes anything from more than a year ago
  #if the files (variables) for a year have already been loaded, do nothing!
  #otherwise load the appropriate .RData file into environment
  dummyname=paste('Nd.',year,sep='') #arbitrarily choosing Nd
  if (exists(dummyname)==F) {
    load(openname,envir=.GlobalEnv)
  }
  #do the same for the previous year's file (previous year=year prior to
initialization year)
  dummyname.prev=paste('Nd.',year.prev,sep='')
  if (exists(dummyname.prev)==F) {
    load(openname.prev,envir=.GlobalEnv)
  }



  ###grab appropriate NARR met variables using NARRgrabber.onerow() (defined
above)
  outputdf=as.data.frame(t(apply(as.matrix(dfnew),1,NARRgrabber.onerow)))

  return(outputdf)


}
```

D.8 "model.sourcefunctions.R"

```
#
#these are the source functions for the Hendricks model!
#load these before running the model!

#atmosphere and soil functions included

#file created 11-12-15 by AJR

#########necessary equations###############

# Calculates the fractionation factor to soil carbonate as a function
# of temperature. Default is 25C.
fract=function(Temp=25){ # From Cerling (1999)--see Figure 5
  fractionation=-(11.709-0.116*Temp+2.16e-4*Temp^2)
  fractionation
}

# Converts CO2 from ppmv to mols/cm3 for use in the soil respiration
# equation.  This conversion is a function of temperature (default is 25C).
CO2convert=function(CO2ppmv,Temp=25){
  CO2v=CO2ppmv/1e6
  MCO2=44.00964 # Molecular mass of bulk CO2
  SP=101325 # Standard pressure (Pa) at 1atm
  R=8.31441 # Universal Gas Constant
  TempK=Temp+273.15 # Correction to Kelvin
  CO2m=(MCO2*SP*CO2v)/(R*TempK) # g/m3 of CO2
  CO2moles=CO2m/MCO2/1e6 # moles/cm3
  CO2moles #moles/cm3
}

# This function does the opposite of the CO2convert function. It takes
# CO2 as moles/cm3 and converts back to ppm.
CO2backconvert=function(CO2_atm,Temp=25){
  MCO2=44.00964 # Molecular mass of bulk CO2
  SP=101325 # Standard pressure (Pa) at 1atm
  R=8.31441 # Universal Gas Constant
  TempK=Temp+273.15 # Correction to Kelvin
  CO2mm3=CO2_atm*MCO2*1e6 # g/m3 of CO2
  CO2v=(CO2mm3*R*TempK)/(SP*MCO2) # absolute quantity of CO2
  CO2ppm=CO2v*1e6 # ppm CO2
  CO2ppm #ppm
}


##function to calculate either d18O (if d18 = True in input; default) or dD
(if d18 = False)
#of ET using Craig and Gordon equations and closure assumption
craig_gordon <- function(t, h, delta_precip, ts, d18 = T){
  if(d18 == T){
    alpha18=1 - (-7.685 + (6.7123*10^3)/(ts) - (1.6664*10^6)/(ts)^2 +
(0.35041*10^9)/(ts)^3)/1000 #equilibrium fractionation 18O, l-v
    r18_s <- (delta_precip/1000 + 1)*smow18  #ratio of precip sample
    Re_18 <- (1-t)*((alpha18*k18*r18_s)/(1-h+(1-t)*k18*h)) +
t*r18_s*(1/(1+(1-t)*k18*(h/(1-h))))
    d18_evap <- (Re_18/smow18 - 1)*1000
    (d18_evap - delta_precip)/1000}
  else{
    alphaD=1 - ((1158.8*ts^3)/10^9 - (1620.1*ts^2)/10^6 + (794.84*ts)/10^3 -
161.04 + (2.9992*10^9)/ts^3)/1000 #equilibrium fractionation D, l-v
    rD_s <- (delta_precip/1000 + 1)*smowD
    Re_D <- (1-t)*((alphaD*kD*rD_s)/(1-h+(1-t)*kD*h)) + t*rD_s*(1/(1+(1-
t)*kD*(h/(1-h))))
    dD_evap <- (Re_D/smowD - 1)*1000
    (dD_evap - delta_precip)/1000}
}

##function to calculate d17O of ET using Craig-Gordon and closure assumption
craig_gordon17=function(t,h,delta_precip,ts) {
  alpha18=1 - (-7.685 + (6.7123*10^3)/(ts) - (1.6664*10^6)/(ts)^2 +
(0.35041*10^9)/(ts)^3)/1000 #equilibrium fractionation 18O, l-v
  alpha17=alpha18^(theta.eq) #Barkan and Luz, 2005
  r17_s=(delta_precip/1000+1)*smow17
  Re_17=(1-t)*((alpha17*k17*r17_s)/(1-h+(1-t)*k17*h))+t*r17_s*(1/(1+(1-
t)*k17*(h/(1-h))))
  d17_evap=(Re_17/smow17-1)*1000
  (d17_evap-delta_precip)/1000
}


##function to calculate integrated isotopic composition of ET with
#integration of Rayleigh distillation for soil moisture reservoir
isotope_e = function(resid, delta_precip, eps_e){
  delt = -resid*((1000/(1+eps_e))*resid^(eps_e)-1000)
  delta_precip + delt
}

#function to convert negative values to NAs (to get rid of neg. Nds)
#only used when averaging tracks
neg.toNA=function(matrix) {
  for (i in 1:dim(matrix)[1]) {
    for (j in 1:dim(matrix)[2]) {
      if (matrix[i,j]<=0) {
        matrix[i,j]=NA
      }
    }
  }
  matrix
}
```

D.9 "Hendricksmodel.onetraj.R"

```
#function for the Hendricks model
#calcuates d18O, dD, d17O of vapor and precip along a storm track
#calculates advection-only and eddy diffusion-only scenarios

#this code is adpated from "model w evap balance and dxs.R" by Matt Winnick
#created: 11/12/15 by AJR
#last modified: 12/6/15 by AJR

#NOTE!!!! initial delta values must be defined before running this function!
#other model input parameters listed below because there's less likely to
need changing

library(geosphere)

###model parameters
Rpdb=0.0112372 #dimensionless PDB Belemnite Ratio
smow18=0.0020052 # smow ratio of 18O/16O
smowD=155.76*10^(-6) # smow ratio of D/H
smow17=379.9*10^(-6) # vsmow ratio of 17O/16O
theta.eq=0.529 #Barkan and Luz, 2005; 17O equilibrium (l-v and v-l)
theta.diff=0.5185 #Barkan and Luz, 2007; 17O diffusion
k18=1-0.01872 # kinetic fractionation factor for 18O, wet soils from Mathieu
and Bariac (1996), slightly higher than lake evap (~14.3)
kD=1-0.01676 # kinetic fractionation factor for dD, wet soils from Mathieu
and Bariac (1996), slightly higher than lake evap (~12.5)
k17=k18^theta.diff #Barkan and Luz, 2005, 2007


Hendricksmodel.onetraj=function(revdf) {
  #takes in a dataframe that's optimized for model and has E/T specified per
point
  #NA rows removed to match NARR's 3-hourly resolution
  #rows already in temporal order, with HYSPLIT initialization point last (at
bottom)

  t=revdf$t
  e=1-t #evaporation as a fraction of total ET

  tracklength=nrow(revdf) #length of storm track

  w=revdf$Prw #precipitatble water, kg/m^2
  ntracks=1 #number of storm tracks
  Nd=revdf$Nd #calculated as E/(P-E) from NARR. LOTS OF NEGATIVES (like, 90%)
:/
  ts=revdf$Temp_2m #2m surface temperature (K)
  rh=(revdf$rhum)/100 #relative humidity (fraction)

  #constants for isotope value calculations:
  #note that a18 != alpha18 (and same for D)
  a18=1 + (-7.685 + (6.7123*10^3)/(ts) - (1.6664*10^6)/(ts)^2 +
(0.35041*10^9)/(ts)^3)/1000 #equilibrium fractionation 18O, v-l
  aD=1 + ((1158.8*ts^3)/10^9 - (1620.1*ts^2)/10^6 + (794.84*ts)/10^3 - 161.04
+ (2.9992*10^9)/ts^3)/1000 #equilibrium fractionation, D v-l
  alpha18=1 - (-7.685 + (6.7123*10^3)/(ts) - (1.6664*10^6)/(ts)^2 +
(0.35041*10^9)/(ts)^3)/1000 #equilibrium fractionation 18O, l-v
  alphaD=1 - ((1158.8*ts^3)/10^9 - (1620.1*ts^2)/10^6 + (794.84*ts)/10^3 -
161.04 + (2.9992*10^9)/ts^3)/1000 #equilibrium fractionation D, l-v
  alpha17=alpha18^(theta.eq) #Barkan and Luz, 2005
  a17=a18^theta.eq

  #build empty vectors to store data for d18O and dD and d17O for advection-
only/eddy-only cases
```

```
  delta_a <- delta_et <- delta_inf<- delta_p <- delta_a_eddy <- delta_et_eddy
<- delta_inf_eddy <- delta_p_eddy <- numeric(length=tracklength)
  delta_D_a <- delta_D_et <- delta_D_inf <- delta_D_p <- delta_D_a_eddy <-
delta_D_et_eddy <- delta_D_inf_eddy <- delta_D_p_eddy <-
numeric(length=tracklength)
  delta_17_a <- delta_17_et <- delta_17_inf <- delta_17_p <- delta_17_a_eddy
<- delta_17_et_eddy <- delta_17_inf_eddy <- delta_17_p_eddy <-
numeric(length=tracklength)

  delta_a[1] = initial_delta_a #sets initial d18O atmospheric vapor
  delta_p[1] = initial_delta_p #sets intial d18O precip
  delta_D_a[1] = delta_a[1]*8+10 #sets initial atmospheric vapor dD
  delta_D_p[1] = delta_p[1]*8+10 #sets initial precip dD
  delta_17_a[1] = (exp(0.528*log(delta_a[1]/1000+1)+0.000033)-1)*1000  #GMWL,
Luz and Barkan, 2010
  delta_17_p[1] = (exp(0.528*log(delta_p[1]/1000+1)+0.000033)-1)*1000  #GMWL,
Luz and Barkan, 2010
  #same for eddies-only case:
  delta_a_eddy[1] = initial_delta_a_eddy #sets initial d18O atmospheric vapor
  delta_p_eddy[1] = initial_delta_p_eddy #sets intial d18O precip
  delta_D_a_eddy[1] = delta_a_eddy[1]*8+10 #sets initial atmospheric vapor dD
  delta_D_p_eddy[1] = delta_p_eddy[1]*8+10 #sets initial precip dD
  delta_17_a_eddy[1] = (exp(0.528*log(delta_a_eddy[1]/1000+1)+0.000033)-
1)*1000  #GMWL, Luz and Barkan, 2010
  delta_17_p_eddy[1] = (exp(0.528*log(delta_p_eddy[1]/1000+1)+0.000033)-
1)*1000  #GMWL, Luz and Barkan, 2010


  ###run the model!

  for (i in 2:tracklength) {
    #calculate residual soil moisture for isotope_e function
    residual=1-(Nd[i]/(Nd[i]+1))*e[i]/(1-(Nd[i]/(Nd[i]+1)*t[i]))
    #if everything evaporates, set isotope value of evaporation to isotope
value of precip
    if(residual == 0){
      delta_et[i] <- delta_p[i-1]
      delta_D_et[i] <- delta_D_p[i-1]
      delta_17_et[i] = delta_17_p[i-1]
      delta_et_eddy[i]=delta_p_eddy[i-1]
      delta_D_et_eddy[i] <- delta_D_p_eddy[i-1]
      delta_17_et_eddy[i] = delta_17_p_eddy[i-1]

      #otherwise, first calculate the combined equilibrium/kinetic
fractionation using Craig-Gordon function
    } else {
      eps_e <- craig_gordon(t[i], rh[i], delta_p[i-1], ts[i])
      eps_e_D <- craig_gordon(t[i], rh[i], delta_D_p[i-1], ts[i], d18 = F)
      eps_e_17 = craig_gordon17(t[i],rh[i],delta_17_p[i-1],ts[i])
      eps_e_eddy <- craig_gordon(t[i], rh[i], delta_p_eddy[i-1], ts[i])
      eps_e_D_eddy <- craig_gordon(t[i], rh[i], delta_D_p_eddy[i-1], ts[i],
d18 = F)
      eps_e_17_eddy = craig_gordon17(t[i],rh[i],delta_17_p_eddy[i-1],ts[i])
      #then calculate integrated ET value from location
      delta_et[i] <- t[i]*delta_p[i-1] + e[i]*isotope_e(residual,delta_p[i-
1], eps_e)
      delta_D_et[i] <- t[i]*delta_D_p[i-1] +
e[i]*isotope_e(residual,delta_D_p[i-1], eps_e_D)
      delta_17_et[i] = t[i]*delta_17_p[i-
1]+e[i]*isotope_e(residual,delta_17_p[i-1],eps_e_17)
      delta_et_eddy[i] <- t[i]*delta_p_eddy[i-1] +
e[i]*isotope_e(residual,delta_p_eddy[i-1], eps_e_eddy)
      delta_D_et_eddy[i] <- t[i]*delta_D_p_eddy[i-1] +
e[i]*isotope_e(residual,delta_D_p_eddy[i-1], eps_e_D_eddy)
```

```
      delta_17_et_eddy[i] = t[i]*delta_17_p_eddy[i-
1]+e[i]*isotope_e(residual,delta_17_p_eddy[i-1],eps_e_17_eddy)
    }


    ###there's an issue with some of the values after the initial point being
NAs
    #this loop will make it so all of the NAs stay NAs *except* the last NA
before
    #hitting land. the first ones stay NAs so that we're only calculating
over land.
    #note: still necessary to keep the original initials above in the event
of no NAs
    #only where rowMeans==0 will they all be above water==>this track will be
above water
    if (revdf$Land[i]==0) {
      delta_p[i]=initial_delta_p
      delta_p[i-1]=NA
      delta_a[i]=initial_delta_a
      delta_a[i-1]=NA
      delta_D_p[i]=delta_p[i]*8+10 #assume initial points are on MWL
      delta_D_p[i-1]=NA
      delta_D_a[i]=delta_a[i]*8+10
      delta_D_a[i-1]=NA
      delta_17_p[i]=(exp(0.528*log(delta_p[i]/1000+1)+0.000033)-1)*1000
#GMWL, Luz and Barkan, 2010
      delta_17_p[i-1]=NA
      delta_17_a[i]=(exp(0.528*log(delta_a[i]/1000+1)+0.000033)-1)*1000
#GMWL, Luz and Barkan, 2010
      delta_17_a[i-1]=NA
      #repeat for eddies (copypasta)
      delta_p_eddy[i]=initial_delta_p_eddy
      delta_p_eddy[i-1]=NA
      delta_a_eddy[i]=initial_delta_a_eddy
      delta_a_eddy[i-1]=NA
      delta_D_p_eddy[i]=delta_p_eddy[i]*8+10 #assume initial points are on
MWL
      delta_D_p_eddy[i-1]=NA
      delta_D_a_eddy[i]=delta_a_eddy[i]*8+10
      delta_D_a_eddy[i-1]=NA
      delta_17_p_eddy[i]=(exp(0.528*log(delta_p_eddy[i]/1000+1)+0.000033)-
1)*1000  #GMWL, Luz and Barkan, 2010
      delta_17_p_eddy[i-1]=NA
      delta_17_a_eddy[i]=(exp(0.528*log(delta_a_eddy[i]/1000+1)+0.000033)-
1)*1000  #GMWL, Luz and Barkan, 2010
      delta_17_a_eddy[i-1]=NA
    } else {
      #haversine formula to calculate distance between coordinates:
      x=distVincentyEllipsoid(c(revdf$Lon[i],revdf$Lat[i]),c(revdf$Lon[i-
1],revdf$Lat[i-1]))
      dx=x #meters, but the units don't matter. slightly >32km
      dw=w[i]-w[i-1]
      dw=ifelse(dw>0,0,dw) #gets rid of positive gradients (b/c we don't like
explosions here)

      #Hendricks model (advection only)
      delta_inf[i] <- (Nd[i]*delta_et[i]-(1+Nd[i])*(a18[i]-
1)*1000)/(a18[i]+Nd[i]*a18[i]-1)
      delta_a[i]=(delta_a[i-1]-delta_inf[i])*exp((a18[i]+a18[i]*Nd[i]-
1)*(x/w[i])*(dw/dx))+delta_inf[i]
      delta_p[i] <-delta_a[i]+(a18[i]-1)*1000

      delta_D_inf[i] <- (Nd[i]*delta_D_et[i]-(1+Nd[i])*(aD[i]-
1)*1000)/(aD[i]+Nd[i]*aD[i]-1)
```

```
      delta_D_a[i]=(delta_D_a[i-1]-delta_D_inf[i])*exp((aD[i]+aD[i]*Nd[i]-
1)*(x/w[i])*(dw/dx))+delta_D_inf[i]
      delta_D_p[i] <-delta_D_a[i]+(aD[i]-1)*1000

      delta_17_inf[i] <- (Nd[i]*delta_17_et[i]-(1+Nd[i])*(a17[i]-
1)*1000)/(a17[i]+Nd[i]*a17[i]-1)
      delta_17_a[i]=(delta_17_a[i-1]-
delta_17_inf[i])*exp((a17[i]+a17[i]*Nd[i]-
1)*(x/w[i])*(dw/dx))+delta_17_inf[i]
      delta_17_p[i] <-delta_17_a[i]+(a17[i]-1)*1000

      #Hendricks model (eddy diffusion only)
      delta_inf_eddy[i] <- (Nd[i]*delta_et_eddy[i]-(1+Nd[i])*(a18[i]-
1)*1000)/(a18[i]+Nd[i]*a18[i]-1)
      delta_a_eddy[i]=(delta_a_eddy[i-1]-
delta_inf_eddy[i])*exp((sqrt(a18[i]+a18[i]*Nd[i])-
1)*(x/w[i])*(dw/dx))+delta_inf_eddy[i]
      delta_p_eddy[i] <-delta_a_eddy[i]+(a18[i]-1)*1000

      delta_D_inf_eddy[i] <- (Nd[i]*delta_D_et_eddy[i]-(1+Nd[i])*(aD[i]-
1)*1000)/(aD[i]+Nd[i]*aD[i]-1)
      delta_D_a_eddy[i]=(delta_D_a_eddy[i-1]-
delta_D_inf_eddy[i])*exp((sqrt(aD[i]+aD[i]*Nd[i])-
1)*(x/w[i])*(dw/dx))+delta_D_inf_eddy[i]
      delta_D_p_eddy[i] <-delta_D_a_eddy[i]+(aD[i]-1)*1000

      delta_17_inf_eddy[i] <- (Nd[i]*delta_17_et_eddy[i]-(1+Nd[i])*(a17[i]-
1)*1000)/(a17[i]+Nd[i]*a17[i]-1)
      delta_17_a_eddy[i]=(delta_17_a_eddy[i-1]-
delta_17_inf_eddy[i])*exp((sqrt(a17[i]+a17[i]*Nd[i])-
1)*(x/w[i])*(dw/dx))+delta_17_inf_eddy[i]
      delta_17_p_eddy[i] <-delta_17_a_eddy[i]+(a17[i]-1)*1000
    }
  }


  #for now only returning d18O of precip (for advection only and eddies only)
  outputdf=cbind(revdf,d18O.p=delta_p,d18O.p_eddy=delta_p_eddy)
  #uncomment to add all the isotopic values!
  #
outputdf=cbind(revdf,d18O.atm=delta_a,d18O.p=delta_p,dD.atm=delta_D_a,dD.p=de
lta_D_p,
  #
d17O.atm=delta_17_a,d17O.p=delta_17_p,d18O.atm_eddy=delta_a_eddy,d18O.p_eddy=
delta_p_eddy)

  return(outputdf)
}
```

D.10 "atmosphere.fromHYSPLIT.R"

```
#This code runs the atmosphere component of the model along individual
HYSPLIT trajectories
#the bulk of the model and data manipulation handled in loaded files.

#Created: 11-11-15 by AJR
#Last Modified: 12-4-15 by AJR


library(RANN) #for nearest neighbor search in latlonNARR function
library(ncdf) #to get lat/lons from file
library(dplyr) #for data structure manipulation
library(geosphere) #to calculate distance between coordinates (Haversine)

#load functions for syncing with NARR data
#NARR data must already by saved as .RData files

setwd("/Users/Annie/documents/model/") #CHANGE to wherever function files
stored
load('NARRgriddata.RData') #these necessary for latlonNARR() function
#^this loads land.data, lat.data, lon.data
source("timechanger.R") #function to help merge time on the datasets
source("latlonNARR.R") #function to regrid HYSPLIT output onto NARR grid
source("NARRgrabber.R") #function to load and grab NARR data along
trajectories
#^^^IMPORTANT! before RUNNING this function user MUST switch to directory
with NARR .RData files!!!
source('model.sourcefunctions.R') #functions used within the model
source('Hendricksmodel.onetraj.R') #function that runs the model itself


#beginning only with atmosphere portion
#first, establish a dataframe for testing!
setwd('/Users/Annie/Documents/model/HYSPLIT output/RData files')
load('Pyramid.SON.precip.RData') #list of 367 precipitation trajectories
rawlist=Pyramid.SON #CHANGE based on name of list loaded above

####NOTE uncomment below if list files are not saved elsewhere
  ##these steps are very computationally expensive, so it's best
  ##to use the pre-generated files for each location.
# #prepare list for use in model
# list.timechanged=lapply(rawlist,timechanger) #change time format to sync
datasets
# list.fixedlatlon=lapply(list.timechanged,latlonNARR) #convert HYSPLIT
output to NARR coordinates
# rm(list.timechanged) #clear space
# setwd('/Users/Annie/Desktop/NARR_for_processing') #CHANGE to directory with
NARR .RData files
# #list.withNARR=lapply(list.fixedlatlon,NARRgrabber) #grab NARR data
associated with each trajectory point
# #^files aren't closing properly due to use of lapply()
#
# #NOTE! best if ~10GB free on hard drive
# list.withNARR=list()
# for (i in 1:length(list.fixedlatlon)) { #CHANGE TO PROPER LENGTH
#   list.withNARR[i][[1]]=NARRgrabber(list.fixedlatlon[i][[1]])
# }
# setwd('/Users/Annie/Desktop/HYSPLIT output/RData files')
# save(list.withNARR,file='list.Pyramid.SON.withNARR.RData') #CHANGE to
appropriate name
#
# #clear out remaining files to save space
# rm(Nd.2013,Nd.2014)
# rm(Prw.2013,Prw.2014)
# rm(rhum.2013,rhum.2014)
# rm(Temp_2m.2013,Temp_2m.2014)
# rm(time.2013,time.2014)
# rm(list.fixedlatlon) #clear space

#now load saved lists to run the model
rm(list.withNARR)
#setwd('/Users/Annie/Desktop/HYSPLIT output/RData files')
load('list.Pyramid.SON.withNARR.RData') #CHANGE based on location/season


list.noNA=lapply(list.withNARR,filter,is.na(Prw)==F) #get rid of all NA'd
```

```
points
#^^(NARR only has data for every third hour but HYSPLIT outputs more)

#reorder so temporally (switch backhours to temporal order)
reorder=function(dataframe) {
  reversed=dataframe[rev(rownames(dataframe)),]
  return(reversed)
}
list.ready=lapply(list.noNA,reorder) #model-ready list!!!!


###TESTING
# #picking out a trajectory with only 2004/2005
# test=jj.coord[[100]] #NOTE these don't include initialization point yet!
# test.timechanged=timechanger(test)
# test.fixedlatlon=latlonNARR(test.timechanged)
# setwd('/Users/Annie/Documents/300/HYSPLIT/NARR 2004') #directory with NARR
.RData files
# test.withNARR=NARRgrabber(test.fixedlatlon)
#
# #this leaves NA's for all but every third point! delete all of these NA
rows
# test.noNA=filter(test.withNARR,is.na(Prw)==F)
#
# #now, reverse the row order because the last point of trajectory is
currently at top
# reversed=test.noNA[rev(rownames(test.noNA)),]
#
# #Now test with some multiple trajectories!
# test=jj.coord[c(100,102)]
# test2=lapply(test,timechanger)
# test3=lapply(test2,latlonNARR)
# setwd('/Users/Annie/Documents/300/HYSPLIT/NARR 2004')
# test4=lapply(test3,NARRgrabber)
# test.noNA=lapply(test4,filter,is.na(Prw)==F)


########model time!##########

#inputs to the model

initial_delta_p=-6.96 #these numbers essentially from Ingraham and Taylor
transect
initial_delta_a=-17.476
initial_delta_p_eddy=initial_delta_p #this is what Matt used, at least
initial_delta_a_eddy=initial_delta_a

#df=reversed
#tracklength=nrow(df) #length of storm track
#t=rep(0.7,times=tracklength) #T as fraction of total ET

#write a function to bind the T/ET to the prepared df so that lapply() can be
used efficiently
#not fully sure yet that this will be useful
tbinder=function(df,t) {
  #t is a T/ET fraction value
  vector=rep(t,times=nrow(df))
  outputdf=cbind(df,t=vector)
  return(outputdf)
}

#input=tbinder(df,0.7)
#results=Hendricksmodel.onetraj(input) #IT WORKS!
```

```
#test model on list
input.list=lapply(list.ready,tbinder,0.7)
results=lapply(input.list,Hendricksmodel.onetraj) #IT WORKS but tons of NAs,
NaNs....

Pyramid.SON.modelresults=results
save(Pyramid.SON.modelresults,file='Pyramid.SON.modelresults.RData')

plot(results[[1]]$Lon,results[[1]]$d18O.p,type='l')
points(results[[1]]$Lon,results[[1]]$d18O.p_eddy,type='l',col='red')
for (i in 2:length(results)) {
  points(results[[i]]$Lon,results[[i]]$d18O.p,type='l')
  points(results[[i]]$Lon,results[[i]]$d18O.p_eddy,type='l',col='red')
}
#meh this is kinda nasty
```