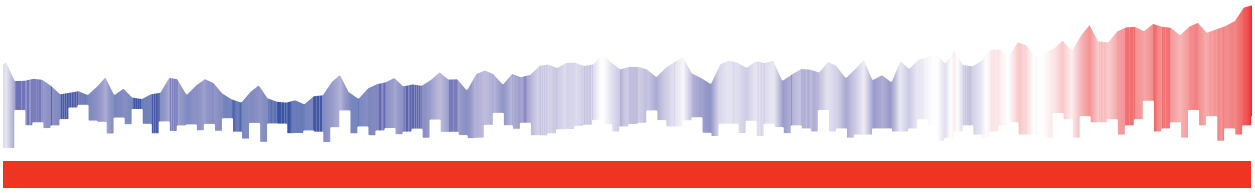


Workshops for

Global Warming Science

A Quantitative Introduction to Climate Change and Its Consequences

Eli Tziperman

March 8, 2025

Workshops for

Global Warming Science

A Quantitative Introduction to Climate Change and Its Consequences

Eli Tziperman

Princeton University Press
Princeton and Oxford

Published by Princeton University Press
41 William Street, Princeton, New Jersey, 08540
6 Oxford Street, Woodstock, Oxfordshire OX20 1TR

press.princeton.edu

The publisher would like to acknowledge the author for providing this manual in PDF form.

Contents

Following are pdf copies of the Jupyter notebook workshops that are available with corresponding data sets here:
https://courses.seas.harvard.edu/climate/eli/Courses/EPS101/
Course slides are also available at the above link.

The textbook is available at
https://press.princeton.edu/books/paperback/9780691228792/global-warming-science

1 Introduction

Global Warming Science

https://courses.seas.harvard.edu/climate/eli/Courses/EPS101/ XX

Introducing python! and other prerequisites

Please use the template below to answer the workshop questions. "XX" indicates places where you need to complete/write code or add a discussion.

Exercise: double-click the text in the cell above to open it, add your name as a subtitle above using two hashtags instead of the XX, shift-enter to recompile.

First, import needed libraries, introducing "name spaces":

[ ]:
```
# This is a code cell. you need to place the cursor in it, and press shift-enter to␣
 ↪run it.
# *Do the same for all code and documentation (markdown) cells below*

import numpy as np                # A library for all numerical caclulations
import matplotlib.pyplot as plt   # for plotting
import pickle                     # for extracting stored variables

print('I just ran my first Jupyter notebook cell!')
```

Exercise: printing

set a variable:

myname='XX your name here'

print("my name is",myname)

[ ]:
```
# insert your code below this line; note that these two lines are a comment in a
# code cell (they starts with #)

XX
```

Exercise: creating a markdown cell with an equation.

add a cell below this point by clicking + on the menu bar;

change the cell type to markdown

insert a title using hashtag(s)

insert an equation, e.g., $E = mc^2$

change the color of the equation only to blue, with everything else in green

run the cell.

XX insert new markdown cell below this point.

A for loop: reading a time series data and printing the data values

[ ]:
```
# load data:
# ----------
# load the PDI data from a pickle file to variables
with open('./introduction_python_variables.pickle', 'rb') as file:
        d = pickle.load(file)
        # print information about each extracted variable:
        for key in list(d.keys()):
            print("extracting pickled variable: name=", key, "; type=",type(d[key]), "; size=", d[key].shape)
        globals().update(d)

print("The dimensions of the input temperature data are:",T.shape)
# note that T is an array with two columns, one for years and one for temperature

# print data for a few years using a for loop:
print("\n year  , Temperature")
for i in range(0,10):
    print(T[i,0],",",T[i,1])
# note that the for loop is indicated by the indentation
# block, in the above case only a single line is in the loop.

# as another example of a for loop, calculate the mean tempearature using a for loop:
N=len(T[:,0])
Tavg=0
for i in range(0,N):
    Tavg=Tavg+T[i,1]

Tavg=Tavg/N

print("\naveraged temperature is",Tavg)
# note that the average is basically zero, as the temperature record is an anomaly␣
 ↪from the mean.

# calculate the mean using a numpy function:
Tavg_numpy=np.mean(T[:,1])
print("averaged temperature calculated by numpy function is",Tavg_numpy)
```

Exercise: for loop:

write a loop that sums the numbers from 1 to 2020

[ ]:
```
# code here:

XX
```

Plot the timeseries data:

[ ]:
```
# plot it:
# --------
fig1= plt.figure(1,dpi=300)
# python array indices are in square brackets and start from zero:
```

```
plt.plot(T[:,0],T[:,1],color="b")
plt.xlabel("Year")
plt.ylabel("Temperature (C)")
plt.title("observed Temperature record");
```

Exercise: line plot

plot $y(x) = \sin(x)$ for $x = -\pi$ to π. Hints: create an x array (vector) using x=np.arange(-1,1.1,0.1), pi is given by np.pi, sine is given by np.sin()

[ ]:
```
# code here:

XX
```

An if statement:

[ ]:
```
# first a simple if statement, note again use of indentation to define end of␣
 ↪statement:
if 2>1:
    print("yes, 2>1")
else:
    print("this will never be printed")


# find the warmest year using a combination of a loop and an if statement:
Tmax=-100
year_max=-100
# note the indentations denoting the blocks of the for loop and if statement:
for i in range(0,N):  # for loop starts here
    if T[i,1]>Tmax:   # if statement starts here
        Tmax=T[i,1]
        year_max=T[i,0] # this is the last line of both the for loop and the if␣
 ↪statement

# note how the last element of the array may be referred to via a -1 index:
print("The first and last years are",T[0,0],T[-1,0])
print("The maximum temperature is", Tmax,", and it occured in year",year_max)
```

Exercise: for and if

write a for loop to calculate the root mean square of the temperature record using the formula

$$rms(T) = \sqrt{\frac{1}{N}\sum_i T_i^2}$$

and then use an if statement to test if the rms is larger than the difference between the temperature at the end and the temperature at the beginning of the series, and print an appropriate message within the if statement:

if $rms(T) > (T_{last} - T_{first})$ print a message that the rms is larger than the difference else print a message that the rms is smaller

[ ]:
```
# insert your code below:
```

```
N=len(T[:,1])
myrms=0.0
for i in XX:
    myrms=myrms+XX
myrms=np.sqrt(myrms/N)

if XX:
    print XX
else:
    print XX
```

Plotting two time series on the same axes using 2 y-axes

[ ]:
```
fig, ax1 = plt.subplots()

color = 'tab:red'
ax1.set_xlabel('year')
ax1.set_ylabel('CO$_2$', color=color)
ax1.plot(CO2[:,0], CO2[:,1], color=color)
ax1.tick_params(axis='y', labelcolor=color)

ax2 = ax1.twinx()  # instantiate a second axes that shares the same x-axis
color = 'tab:blue'
ax2.set_ylabel('Temperature', color=color)  # we already handled the x-label with ax1
ax2.plot(T[:,0],T[:,1], color=color)
ax2.tick_params(axis='y', labelcolor=color)

fig.tight_layout()  # otherwise the right y-label is slightly clipped
plt.show()
```

Exercise: Plotting two time series on the same axes using 2 y-axes<>/font

insert a code cell below this point, and plot sine(x) and 5cos(x) from $x = 0$ to $x = 2\pi$

XX insert a code cell below this point.

Getting help from chatGPT

(a) Provide the following instruction to chatGPT, copy paste the resulting code into a new code cell below, test the code and adjust it as needed to make it work. XX

write a python code to create two numpy arrays with x going from -pi to +pi and y=3*exp(-x)*cos(x). plot y as a function of x using matplotlib. save the result to a pdf file named myfigureoutput.pdf.

(b) Ask chatGPT to correct the following buggy code, replace the code with the corrected one and make sure it runs.

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt

# XX

x=np.arange(-2*pi,pi,0.1)
```

```
f=cos(x)
plt.figure(1)
plt.clf()
plt.plot(x,f)
# always label axes and provide titles:
plt.xlabel('x')
plt.ylable("f')
plt.title('f(x)=cos(3x)')
plt.show()
```

Using a built-in python function to calculate correlation

[ ]:
```
# remove mean from each time series using np.mean():
X=T[:,1]-XX
Y=CO2[:,1]-XX

R=np.corrcoef(X,Y)

print("correlation calculated by np.corrcoef is:\n",R)
```

Writing a python function that calculates correlation

using the formula: $R = \frac{\sum X_i Y_i}{\sqrt{\sum X_i^2}\sqrt{\sum Y_i^2}}$

(note how the documentation cells can include equations!)

[ ]:
```
# here is how one defines a function:
def my_calc_correlation(X,Y):
    A=0
    B=0
    C=0
    for i in range(0,len(X)):
        A=A+X[i]*Y[i]
        B=B+X[i]*X[i]
        C=C+Y[i]*Y[i]
    R=A/(np.sqrt(B)*np.sqrt(C))
    return R

# and now we can use this function as follows
# (we use the fact that the mean has been renoved from X,Y)
my_R=my_calc_correlation(X,Y)
print("correlation calculated by my function is:\n",my_R)
```

Exercise: writing a function and plotting

write a function that takes a time series T_i and produces a smoothed version

$$\overline{T}_i = 0.25T_{i-1} + 0.5T_i + 0.25T_{i+1}$$

and then plot the two on the same axes (no need to add a second vertical axis in this case). Note that you cannot calculate the smoothed version for the first and last points, and the smoothed time series therefore has 2 less points than the original one.

[ ]:
```
# insert your code here

def smooth_timeseries(d):
    d_smooth=np.zeros(len(d))
    for i in XX:
        d_smooth[i]=XX
    return d_smooth[1:(len(d)-1)] # return only the interior points

T_smooth=smooth_timeseries(XX)
time_axis=XX # take only interior points of time axis too, for plotting

# plot it:
# --------
fig1= plt.figure()
# python array indices are in square brackets and start from zero:
plt.plot(XX) # plot original time series
plt.plot(XX) # plot smoothed one
plt.xlabel(XX)
plt.ylabel(XX)
plt.title(XX);
```

Other python data types:

We already covered numpy arrays, other important data types are: * lists * dictionaries * tuples

[ ]:
```
# create a list with the two data sets combined
# an empty list:
my_data_list=[]
my_data_list.append(T)
my_data_list.append(CO2)
print("length of list is:",len(my_data_list))
print("the type of the first element of the list (that is, the zeroth index) is:
 ↪",type(my_data_list[0]))
print("shape of first element of the list is:",my_data_list[0].shape)
print("the [4:8,1] values from the first element of the list is:",my_data_list[0][4:
 ↪8,1])
```

[ ]:
```
# create a dictionary with the two data sets combined
# an empty dictionary:
my_data_dict={}
my_data_dict["Temperature"]=T
my_data_dict["Carbon-dioxide"]=CO2

print("keys in dictionary are:",my_data_dict.keys())
print("the first few elements of the first data set are:
 ↪",my_data_dict["Temperature"][0:5,0:2])
```

Exercise: creating tuples and dictionaries

create a dictionary with two lists. one is the first 7 letters of the alphabet (as strings) with the key being the string 'letters'; the other the digits from 0 to 9 (as numbers), with the key being the string 'numbers'.

then print the keys of the dictionary

print the 2nd-6th elements (where the 1st element is zero-indexed) of the first list in the dictionary.

write a code to sum the 3rd-8th elements in the second list and print the result

[ ]:
```
# Code here:

XX
```

some tricky stuff

(1) Ranges in python do not include the last element!

Note that in the last example, when we specify a range to be 0:5,0:2, the printed output is a 5X2 rather than a 6X3 as one might expect. This is part of the python specification, as demonstrated by the following loop:

[ ]:
```
for i in range(2,6,1):
    print(i)
```

(2) Making a copy of a python variable without getting in trouble:

An assignment like B=A does not create a new variable B whose value is equal to that of A, but creates a pointer to A. This can lead to troube. Here are the details.

[ ]:
```
# consider an arrays,
A=np.array([2,3,4])
print("A=",A)
print("now set B using B=A:")
B=A
print("A=",A,",B=",B)
print("changing B using B[1]=17:")
B[1]=17
print("changes also A:")
print("A=",A,",B=",B)

# in order to avoid this, create B using, for example:
B=A*1
```

Exercise: using ranges and making copy of variables

(A) write a for loop that prints the numbers in the range -2 to 8 (B) duplicate the above example, replacing B=A with B=A*1 and verify that this creates an independent array that is not modified when A is.

[ ]:
```
# insert your code here

XX
```

Mathematical preliminaries

Be sure you are comfortable solving these, as this material will be used during the semester.

(1) what is the derivative with respect to t of

(1a) $Ae^{\alpha t}$,

(1b) $A\sin(bt)$,

(2) Calculate the integral $\int_0^1 \cos(2t)$

(3) what is the solution to each of the two following basic differential equation problems,

$$\frac{dx}{dt} = \alpha x, \quad \text{with } x(0) = -3$$
$$\frac{d^2x}{dt^2} = -\omega^2 x, \quad \text{with } x(0) = -3, \text{ and } \frac{dx(0)}{dt} = 0$$

Your solution here:

1a XX

1b XX

2 XX

3 XX

How to download a Jupyter notebook as a PDF:

In JupyterHub, go to File -> Save and Export Notebook As… -> PDF

Before downloading and submitting Jupyter notebook:

With the notebook active in JupyterHub go to Kernel -> "Restart Kernel" and "Run All Cells." This ensures that the code output is up-to-date with what you have written in each cell.

Writing assignment for first class:

Due next week with this Jupyter notebook, see guidelines in

https://courses.seas.harvard.edu/climate/eli/Courses/EPS101/Sources/Introduction/1st-class-writing-assignment.pdf

2 Greenhouse

Global Warming Science

https://courses.seas.harvard.edu/climate/eli/Courses/EPS101/

Greenhouse

Please use the template below to answer the workshop questions. "XX" indicates places where you need to complete/write code or add a discussion.

your name: first last

[ ]:
```
import numpy as np
from numpy import linalg
import matplotlib.pyplot as plt
import matplotlib
import pickle
from matplotlib.ticker import MultipleLocator
import scipy as scipy
from scipy import optimize
import warnings

# load the data from a pickle file:
with open('./greenhouse_variables.pickle', 'rb') as file:
    d = pickle.load(file)
    # print information about each extracted variable:
    for key in list(d.keys()):
        print("extracting pickled variable: name=", key,"; size=", d[key].shape)
        #print("; type=",type(d[key]))
    globals().update(d)
```

Explanation of variables:

CO2_observed is a time series of observed CO_2, as a function of time in years given by variable **CO2_observed_years**.

Similarly, **CO2_rcp*** are time series of projected CO_2 according to different RCP scenarios.

OLR_280ppm_CO2_only is the top-of-the-atmopshere outgoing longwave radiation, with 280 ppm of CO2, as function of **wavenumbers**, that are given by the variable of that name. similarly for **OLR_560ppm_CO2_only**.

1) Observed and projected increase in greenhouse gasses:

Plot CO_2 since 1850, superimpose the predicted CO_2 concentration from the RCP8.5 (sometimes referred to as "business as usual'') scenario from a previous IPCC report going to 2100.

[ ]:
```
fig = plt.figure(1,figsize=(3,3),dpi=200)
#plt.plot(XX,XX,lw=3,color="blue",label="observed CO2")
#plt.plot(XX,XX,lw=1,color="red",label="Projected rcp8.5 CO2")
plt.legend()
#plt.xlabel(XX)
plt.ylabel('CO$_2$ Concentration (ppm)');
```

2) Energy balance model:

(a) Calculate the downwelling LW and absorbed SW at the surface in the two-layer model, using an albedo of 0.25, an atmospheric emissivity of 0.8, and searching the web for the other constants.

[ ]:
```
# constants:
S0 = XX  # solar constant
albedo = XX # Albedo
epsilon = XX # emissivity
sigma = XX  # stefan-boltzmann constant


# calculate atmospheric temperature T_a:
T = XX
T_a=XX
print("T,T_a=",T,T_a)
# answer:
LW_down=XX
SW_net=XX
print("LW_down=",LW_down,",SW_net=",SW_net)
```

(b) Modify the 2-layer energy balance model to take into account the absorption of a fraction $\Delta = 0.15$ of the downward shortwave radiation by the atmosphere. Calculate the resulting temperatures, compare to the case of $\Delta = 0$. Use the results to discuss how increased atmospheric aerosol concentrations due to industrial emissions, which increases SW absorption, can mask the surface warming due to CO_2 increase.

Solution

The new energy balance equation for the surface is:

$$\frac{S_0}{4} XX(1-\alpha) + \epsilon\sigma T_a^4 = \sigma T^4 \tag{1}$$

The new energy balanance equation for the atmosphere is:

$$XX + \epsilon\sigma T^4 = 2\epsilon\sigma T_a^4 \tag{2}$$

where T is the surface temperature, and T_a is the atmospheric temperature. Substituting the second equation into the first yields:

$$XX = \sigma T^4 \tag{3}$$

Solving for T yields:

$$T = \left[\frac{XX}{\sigma(1-\frac{\epsilon}{2})}\right]^{1/4} \tag{4}$$

[ ]:
```
# evaluating the solution numerically:

Delta = .15
T = (XX)**(1/4)
print('T = ', T)
T_a = XX
print('T_a = ',T_a)
```

Discussion:

XX

(c) Consider an abrupt CO_2 doubling corresponding to the transition from panel a of Figure 2.4 to panel b. Considering the response of an atmosphere with a continuous vertical temperature profile of a prescribed lapse rate, calculate the increase in the emission height corresponding to an increase in radiative forcing by 4 W,m^{-2} assuming a lapse rate of 6.5 K/km.

[ ]:
```
# first, the temperature decrease required to lower the LW by 4 W/m^2:

# The emission height temperature is found by the balance of outgoing
# long-wave radiation from the emission height balancing the incoming solar
# radiation minus the refelected part.
T=XX
# Calclate the right value of DeltaT that leads to a 4 W/m^2 change in F:
DeltaT=XX
print("DeltaT=",DeltaT)
# check that radiative forcing changed as expected:
print("DeltaF=",sigma*(XX)**4-sigma*XX**4)

# now, given a lapse rate of 6.5 K/km, this implies the following change in emission height:
print("change in emission height (m):",XX*1000)
```

3) Radiative forcing

(a) Explain the upper and lower boundaries of the trapezoidal gap in the shape of the OLR curve around 600-800 cm^{-1} (Figure 2.5b) due to CO_2 absorption, in terms of the emission height.

(b) Plot the OLR in a standard atmosphere in the presence of 280 and 560 ppm CO2, ignoring all other greenhouse gasses.

(c) Estimate the radiative forcing of 280 ppm of CO_2 by fitting a trapezoid to your plotted OLR curve and calculating its area.

(d) Calculate the radiative forcing due to a doubling of the CO_2 from the preindustrial value of 280 ppm, by integrating over the difference of the two OLR curves.

Solutions

a) XX

[ ]:
```
# (b)

# Read in pickle data of the two OLR curves and plot
fig = plt.figure(1,figsize=(6,4),dpi=300)
ax = plt.gca()
plt.plot(XX,XX,lw=1,color="blue",label="280 ppm")
plt.plot(XX,XX,lw=1,color="red",ls='--',label="560 ppm")
plt.xlabel('Wavenumber (cm$^{-1}$)')
plt.ylabel('OLR (W/[m$^2$cm$^{-1}$])')

# plot tickmarks and grid
ax.xaxis.set_minor_locator(MultipleLocator(50))
ax.yaxis.set_minor_locator(MultipleLocator(.02))

# title and legend:
plt.title("OLR as function of wavenumber")
```

```
plt.grid(which='both',lw=0.25)
plt.legend();
plt.show();
```

c) We choose the top of the trapezoid as XX. This gives a total height of the trapezoid as XX W/m^2. The bottom width is about XX cm^{-1} and the top width is about XX cm^{-1}. Thus using the formula for a trapezoid, $A = H * (Base1 + Base2)/2$ we get a radiative forcing of XX W/m^2.

d) The integral over the OLR curve tells you the total outgoing longwave radiation. The difference between the total OLR with 280ppm and 560ppm gives you the radiative forcing of doubling CO2.

[ ]:
```
# approximate the integral by a sum:
Delta_wavenumber = wavenumbers[1] - wavenumbers[0]
RF = Delta_wavenumber*np.sum(XX-XX)
print("Radiative Forcing: "+str(round(RF,3))+' W/m^2')
```

[ ]:
```

```

4) Logarithmic dependence

if a doubling of CO_2 from 280 ppm to 560 leads to, say, 3 degree warming, what warming do you expect at an equivalent CO_2 mixing ratio of 500 ppm? Discuss the fact that warming so far is just over 1.2 degree.

[ ]:
```
# if the warming for CO2 doubling is 3 degree, we can write the warming as
CO2=560
DeltaT=XX
print("CO2=",CO2,", DeltaT=",DeltaT)
CO2=500
DeltaT=XX
print("CO2=",CO2,", DeltaT=",DeltaT)
```

Discussion: XX

5) Global warming potential:

Suppose the GWP of methane for a time horizon of 100 yr is 34. Assume methane concentration decays exponentially over 12.4 yr (so the decay function is $e^{-t/12.4}$) while that of CO_2 decays over 200 yr (e.g., due to dissolution in the ocean; note that while such a timescale is often used, the actual expected residence time of anthropogenic CO_2 in the climate system is thousands of years.) (a) How much more efficient is a kilogram of methane at absorbing LW radiation than a kilogram of CO_2? (b) Discuss the difference between GWP and absorption efficiency. (c) What is the methane GWP for a time horizon of 500 yr? Explain.

(a) Reformulate the GWP equation to solve for a_x/a_r.

XX

Therefore, we can conclude that methane molecules absorbs more than XX times more strongly than CO_2.

(b) XX discuss

(c) XX

6) Water vapor feedback to increased CO_2:

Modify the energy balance model equations (2.2) to include the effects of increasing water vapor and calculate the warming as a function of CO_2 with and without this feedback as follows. Assume the atmospheric emissivity depends on CO_2 and moisture $q(T_a)$ as $\epsilon(CO_2, q(T_a)) = \epsilon_0 + A \log_2(CO_2/280) + B \log_2(q(T_a)/q(T_{a0}))$,

where $\epsilon_0 = 0.75$ and T_{a0} is the atmospheric temperature for CO_2=280 ppm, and that the atmospheric relative humidity is the same before and after the CO_2 change. Find values for A and B such that the warming due to CO_2 doubling only is 2°C, and due to both feedbacks is 4°C. Plot $T_a(CO_2)$ for CO_2 in the range of 200 to 800 ppm with and without the water vapor feedback.

[ ]:
```
# constants:
albedo = 0.32
epsilon_0 = 0.75
Pa=500 # mid-atmospheric pressure

# atmospheric temperature at 280 ppm:
T_a0 = XX

# first, find what changes to the emissivity are needed to cause the suggested␣
 ↪warmings:
# Rerun this cell with different values of A until the temperature rise due to CO2␣
 ↪doubling alone is 2 K:
A = XX  # change due to CO2 doubling
T_a_2XCO2 = XX # energy balance solution in terms of an emissivity of epsilon0+A
print("T_a0=",T_a0,", T_a_2XCO2=", T_a_2XCO2)
```

Now find B:

[ ]:
```
# help functions: emissivity and saturation specific humidity:
def emissivity(CO2,T_a,A,B):
    # emisivity a function of CO2 and atmc temperature
    epsilon=XX
    return epsilon

def q_sat(T,P):
    # saturation specific humidity (g water vapor per g moist air):
    # inputs:
    # T: temperature, in Kelvin
    # P: pressure, in mb
    R_v = XX # Gas constant for moist air = 461 J/(kg*K)
    R_d = XX # Gas constant 287 J K^-1 kg^-1
    TT = XX # Kelvin to Celsius
    qw=
    return qw

# the desired temperature with both feedbacks as specified in the question:
T_a_2XCO2_water_vapor = T_a_2XCO2+2;

# now calculate and plot the temperature as a function of the coefficient B to find
# when it reaches the assumed warming:

Brange = np.arange(XX)
Trange=XX # a array with zeros of the same dimension of Brange, for
          # temeprature as a function of B
i=-1
for B in Brange:
    i=i+1
    epsilon = XX
    Trange[i]=XX # energy balance solution with appropriate emissivity
```

```
# Plot:
fig = plt.figure(figsize=(6,3),dpi=300)
plt.plot(XX \
         , label='temperature with water vapor feedback')
plt.plot(XX \
         , label='T_a_2XCO2_water_vapor')
plt.legend()
plt.xlabel('XX')
plt.ylabel('XX')
plt.grid(lw=0.25)
plt.title('XX');
plt.show();
```

Use A,B to solve for Ta and plot with and without the water vapor feedback:

[ ]:
```
# from the plot, we conclude that
B=XX
# results in the desired temperature response to CO2 and water vapor.

# Now plot the temperature with and without the water vapor
# feedback as a function of CO2.

# helper function to solve for T_a when emissivity depends on it:
def solve_for_T_a(T_a):
    # calculate T_a-((S0/4)*(1-albedo)/(sigma*(2-epsilon(CO2,T_a))))**(1/4),
    # to be called by scipy.optimize.fsolve in order to find a solution for T_a that
    ↪satifies
    # this equation
    equation=XX
    return equation


# range of CO2 to consider:
CO2_range=np.arange(XX)
# Prepare arrays for atmospheric temperature response to CO2
# and to both CO2 and water vapor:
Ta_CO2_only=XX
Ta_both=XX

i=-1
for CO2 in CO2_range:
    i=i+1
    B=0; epsilon=emissivity(XX,XX,XX,XX)
    Ta_CO2_only[i]=XX
    B=XX # from previous cell
    x0=XX # initial guess
    T_a=optimize.fsolve(solve_for_T_a,x0,args=(A,B))
    Ta_both[i]=T_a


# Plot Ta vs CO2 with and without water vapor feedback:
fig = plt.figure(figsize=(6,3),dpi=300)
plt.plot(CO2_range, XX, color = 'orange', label='CO$_2$ only')
```

```
plt.plot(CO2_range, XX, color = 'red', label='CO$_2$ and water vapor')
plt.legend()
plt.xlabel('XX')
plt.ylabel('XX')
plt.grid(lw=0.25)
plt.title('XX');
plt.show();
```

3 Temperature

Global Warming Science

https://courses.seas.harvard.edu/climate/eli/Courses/EPS101/

Temperature

Please use the template below to answer the workshop questions. "XX" indicates places where you need to complete/write code or add a discussion.

your name:

If not using FAS on-demand Jupyter server: Start by installing a package for plotting maps: cartopy. Option 1: from the Anaconda Navigator: click environments -> search, toggle to "uninstalled", mark and click install. **Option 2:** from a terminal: conda install cartopy or if this does not work pip install cartopy

[ ]:
```
import matplotlib.pyplot as plt
import numpy as np
import os
import pickle
# cortopy allows to plot data over maps in various spherical prjections"
import cartopy.crs as ccrs
import cartopy.feature as cfeature
from cartopy.feature import NaturalEarthFeature
from cartopy import config
from mpl_toolkits.axes_grid1 import make_axes_locatable
from scipy.integrate import solve_ivp
from scipy import optimize
from scipy.stats import linregress


# load data from pickle file:
# ---------------------------
with open('./temperature_variables.pickle', 'rb') as file:
        d = pickle.load(file)
        # print information about each extracted variable:
        for key in list(d.keys()):
            print("extracting pickled variable: name=", key, "; size=", d[key].shape)
            #print("; type=",type(d[key]))
        globals().update(d)
```

Explanation of input variables:

RCP85_zonally_avg_T_2010: zonally-averaged atmospheric temperature at 2010 as a function of height and latitude.

RCP85_zonally_avg_T_2100: same for an RCP8.5 projection at 2100

RCP85_zonally_avg_T_height: height coordinate for previous two variables

RCP85_zonally_avg_T_lat: latitude coordinate for above two variables

hockeystick_annual_temperature_anomaly_timeseries

In the followings, SAT=Surface air temperature. The variables ending with "_timeseries" include two arrays, the time axis and SAT anomaly time series:

Three Projections:

rcp26_global_sat_anomaly_timeseries, rcp45_global_sat_anomaly_timeseries, rcp85_global_sat_anomaly_timeseries, and observations:

sat_obs_GISS_annual_anomaly_timeseries

Maps of temperature as a function of latitude and longitude and the corresponding axes:

rcp85_last_5yr_sat_map, rcp85_first_5yr_sat_map, rcp85_sat_map_axes

A time series of 100 years from a "control" climate model run, with fixed pre-industrial CO2:

sat_control_series_GFDL

1) Characterize the warming, historical and future projections:

1a) "Hockey stick curve": Line plot of global-mean, annual-mean temperature anomaly as function of time, for observations continued by RCP8.5 and RCP2.6 scenarios.

[ ]:
```
"""
Plot the paleo global temperature record followed by three RCP scenarios.
"""
# Rename variables for convenience:
historic=hockeystick_annual_temperature_anomaly_timeseries
rcp26=rcp26_global_sat_anomaly_timeseries
rcp45=rcp45_global_sat_anomaly_timeseries
rcp85=rcp85_global_sat_anomaly_timeseries

# plot:
plt.plot(historic[0,:], historic[1,:], "k",label="paleo record")
# plot rcp26, rcp45, rcp85 XX

# labels and titles:
plt.xlim(500,2100)
plt.ylabel("Temperature anomaly (°C)")
plt.xlabel("Year")
plt.title("paleo record plus RCP scenarios")
plt.legend()
plt.show()
```

1b) Spatial distribution of warming: Contour the temperature at the beginning of the 21st century, end, and the change over the 21st century, according to the RCP8.5 scenario.

[ ]:
```
# Load the plot axes:
longitude = rcp85_sat_map_axes[1]
latitude = rcp85_sat_map_axes[0]
contour_levels=np.arange(250,314,1) # for temperature plots

# plot first 5 year temperature:
# ------------------------------
fig=plt.figure(figsize=(12,6))
ax = plt.axes(projection=ccrs.PlateCarree(central_longitude=0.0, globe=None))
ax.set_extent([-180, 180, -90, 90], crs=ccrs.PlateCarree())
ax.coastlines(resolution='110m')
ax.gridlines()

# plot contours:
c=plt.contourf(longitude, latitude, rcp85_first_5yr_sat_map,levels=contour_levels)
```

```
# draw the colorbar
clb=plt.colorbar(c, shrink=0.85, pad=0.02)
# add title/ labels:
plt.xlabel('Longitude')
plt.ylabel('Latitude')
clb.set_label('Temperature (K)')
plt.title('Temperature during 1st 5 years of RCP 8.5')
plt.show()

# plot last 5 year temperature:
# ---------------------------------
contour_levels=XX # for temperature plots
XX
c=plt.contourf(XX, XX, XX,levels=contour_levels)
XX

# plot the warming:
# -----------------
# loading data from file
warming_sat_map_data = rcp85_last_5yr_sat_map-rcp85_first_5yr_sat_map
# use cmap='bwr' in the contourf call,
# and symmetric contour levels centered around zero:
contour_levels_warming=np.arange(XX)
XX
c=plt.contourf(XX, XX, XX,levels=contour_levels_warming)
XX
```

1c) Polar amplification: Plot the zonally-averaged warming vs latitude for the RCP8.5 scenario.

[ ]:
```
# calcuate zonal mean warming:
# zonal_mean_rcp85_warming=XX use np.mean to calculate it, averaging over longitude

# a plot:
plt.plot(zonal_mean_rcp85_warming,latitude)
plt.xlabel('XX')
plt.ylabel('XX')
plt.title('XX')
# show the plot
plt.show()
```

2) Hiatus:

2a) Plot the global-mean, annual-mean control-run temperature variability with and without an added linear trend of 1 °C/century.

[ ]:
```
# ----------------------------------------------------------------
# plot a monthly time series of global mean SATs from the control
# dataset of the GFDL model with and without added linear trend,
# to demonstrate how "hiatuses" arise from variability plus a
# warming trend.
# [thanks to Yonathan Vardi]
# ----------------------------------------------------------------

fig = plt.figure(figsize=(12,6))
```

```
sat_control_series = sat_control_series_GFDL
# The time axis for control time series:
control_years = np.arange(1, 1+len(sat_control_series))
# generate a linear temperature trend of 0.01 degree per year
control_linear_trend = XX
# combine the trend and the series
series_with_trend = control_linear_trend + sat_control_series

#plot SAT control run and SAT control run plus linear trend
plt.subplot(1, 2, 1)
plt.plot(control_years, XX, ":")
plt.plot(control_years, XX)

plt.ylabel("Temperature (°C)")
plt.xlabel("years")
plt.title("Control run + Linear Trend")

# plot for comparison the global temperature record from the last 100 years:
# ------------------------------------------------------------------------
plt.subplot(1, 2, 2)
obs=sat_obs_GISS_annual_anomaly_timeseries
print("note that the array obs contains both the time axis as obs[0,:] \nand the data␣
  ↪as obs[1,:]. its shape is:",obs.shape)
plt.plot(XX)
plt.ylabel("Temperature (°C)")
plt.xlabel("years")
plt.title("The instrumental record")

#show the plot
plt.tight_layout()
plt.show()
```

(b) Plot the observed global temperature anomaly starting 1970 to the last year of data. Fit a linear trend to 1970-1998. Extend the plot of the linear fit through the last year of data. Comment on whether the data post the supposed 1998–2013 "hiatus" period support the idea that global warming ceased in 1998.

[ ]:
```
obs=sat_obs_GISS_annual_anomaly_timeseries
x=obs[0,:]
y=obs[1,:]

xfit=x[np.logical_and(x>=1970,x<1998)]
yfit=XX
fit=linregress(xfit, yfit)
slope=fit.slope;
intercept=fit.intercept;
fitted_line_until_1998=XX
fitted_line_to_end_of_data=XX

fig = plt.figure(figsize=(5,2.5),dpi=300)
plt.plot(x,y,color="r",linewidth=1,label="Observed GMST")
plt.plot(XX,label="Fit until 1998",linewidth=3)
plt.plot(XX,label="Fit continued",linewidth=1)
plt.ylabel("Temperature anomaly (°C)")
```

```
plt.xlabel("Year")
plt.xlim(1969,2020)
plt.ylim(XX)
plt.legend()
plt.grid(lw=0.25)
```

Comment: XX

3) Equilibrium climate sensitivity:

The current radiative forcing anomaly due to anthropogenic emissions is 2.3 Watts/m^2, the current global-mean surface warming is 1C. Assume 50% of the radiative forcing heat flux anomaly currently goes into the ocean. Estimate the equilibrium warming at a double-preindustrial CO_2 concentration (2×280 ppm), assuming that this corresponds to a radiative forcing of 4 Watts/m^2.

[ ]:
```
Delta_F=XX   # radiative forcing now
Delta_Q=XX   # heat absorbed by the ocean now
Delta_F2x=XX # radiative forcing at 2xCO2
Delta_T=XX   # warming now
Delta_T2x=XX # warming for 2xCO2
print("The expected equilibrium (steady-state) warming is ",Delta_T2x)
```

4) Transient Climate sensitivity and the role of the ocean

Solve equations 3.3 and plot ΔT_{deep} and $\Delta T_{\text{surface}}$ for $t = 0, \ldots, 5000$ years, for $H = 40$ m and for $H = 4000$ m. Show directly from the equations that the steady state solution is consistent with the plotted results. Explain the role of the ocean heat capacity in the response to global warming.

[ ]:
```
# solve the above two ordinary differential equations and plot, for a deep
# and a shallow ocean.

#############################################################################
# functions
#############################################################################
# Right hand side to be integrated:
def right_hand_side(t, T, depth):
    T_atm=T[0]
    T_ocn=T[1]
    lambda_LW = 4/3      #longwave radiative cooling coefficient
    c_atm = c_p * rho_w * 50 #specific heat of upper ocean and atmosphere
    dF = 4  #radiative forcing
    c_ocn = c_p * rho_w * depth  #the specific heat of the ocean
    gamma = 1   #strength of coupling between surface and deep
    dT_atm_dt = ((dF - lambda_LW*T_atm) - gamma*(T_atm-T_ocn))/c_atm
    dT_ocn_dt = gamma*(T_atm-T_ocn)/c_ocn
    return [dT_atm_dt, dT_ocn_dt]

def solve_ODE(depth):
    """
    Solve the ODE for a given ocean depth
    Input: Depth of the ocean.
    Output: ocean temperature time series, atmospheric temperature
            time series, and the time axis.
    """
```

```
    y0=T0
    tspan=(times_save[0],times_save[-1])
    sol = solve_ivp(fun=lambda t,T: right_hand_side(t,T,depth) \
                ,vectorized=False,y0=T0,t_span=tspan,t_eval=times_save)
    time_array=sol.t
    T_atm_array=sol.y[0,:]
    T_ocn_array=sol.y[1,:]

    return T_ocn_array,T_atm_array, time_array/YEAR


########################################################################
# Main program
########################################################################

# set parameters:
# ---------------
YEAR = 31536000 # A year in seconds
YEARS_TO_RUN = 5000
T0, time_start = (0.0,0.0), 0.0 # initial values for (T_ocn, T_atm) and time:
time_end = YEAR * YEARS_TO_RUN #the maximum value of time
times_save = np.arange(time_start,time_end,YEAR/12) # times for saving ODE solution
N=len(times_save) #the number of points calculated
c_p = 4005 # J/kg, specific heat of seawater
rho_w = 1023.6 # kg/m^3, density of seawater


# solve for temperature in both cases:
# ------------------------------------
T_ocn_array_deep,   T_atm_array_deep,   time_array_deep    = solve_ODE(depth=4000)
T_ocn_array_shallow,T_atm_array_shallow,time_array_shallow = solve_ODE(depth=40)


# print final results at 10 years and at steady state:
# ----------------------------------------------------
print("The temperatures of the atmosphere/ ocean for the case of a shallow ocean are:")
# print the state at 10 years:
print("year=",time_array_shallow[XX],",T_ocn=",T_ocn_array_shallow[XX],",T_atm=",T_atm_array_shallow[XX
# print the state at steady state (last time calculated):
print("year=",time_array_shallow[XX],",T_ocn=",T_ocn_array_shallow[XX],",T_atm=",T_atm_array_shallow[-1
print("The temperatures of the atmosphere/ ocean for the case of a deep ocean are:")
# same as above but for the deep ocean, at 10 years and at a steady state:
XX

# plot solution time series for a shallow ocean for the first 200 years:
# ----------------------------------------------------------------------
plt.figure(figsize=(8,4));
plt.subplot(3,1,1)
plt.plot(time_array_shallow, T_atm_array_shallow, "--", color='SkyBlue',
         label="Surface Temperature (40m ocean)")
plt.plot(time_array_shallow, T_ocn_array_shallow, "--", color='MidnightBlue',
         label="Ocean Temperature (40m ocean)")
plt.ylabel(' T (deg C)')
```

```
plt.xlabel('t (years)')
plt.title('Atm/ Ocean temperature anomalies given a sudden 2x CO  increase')
plt.legend(loc='lower right')
plt.xlim([0,200])

# plot solution time series for a deep ocean for the first 200 years:
# ------------------------------------------------------------------
plt.subplot(3,1,2)
XX
plt.xlim([0,200])

# plot solution time series for a deep ocean for the whole 5000 years period:
# ---------------------------------------------------------------------------
plt.subplot(3,1,3)
XX
```

writing the equations

$$C_{\text{surface}}\frac{d\Delta T_{\text{surface}}}{dt} = \Delta F_{2\times} - \lambda_{LW}\Delta T_{\text{surface}} - \gamma\left(\Delta T_{\text{surface}} - \Delta T_{\text{deep}}\right)$$
$$C_{\text{deep}}\frac{d\Delta T_{\text{deep}}}{dt} = \gamma\left(\Delta T_{\text{surface}} - \Delta T_{\text{deep}}\right). \quad (1)$$

at a steady state

XX

The second one tells us that $\Delta T_{\text{surface}} = \Delta T_{\text{deep}}$, and substituting that in the first equation we find

XX

as requested. Evaluating the solution,

Delta_T=XX

Evaluating the solution numerically,

Delta_T=XX degrees

DISCUSSION: based on this, the role of the ocean heat capacity is XX

5) Polar amplification

a) Planck Feedback:

(i) Consider $F = \epsilon\sigma T^4$, letting $\epsilon = 0.6$, calculate the warming ΔT due to an increase in radiative forcing of $\Delta F = 4$ Watt/m^2, if $T = -10$°C, and then if $T = 30$°C. (Hint: it is easier to calculate ΔF from T and ΔT for different values of ΔT).

(ii) Calculate also dF/dT for these two temperatures by taking a derivative of the black body radiation law.

[ ]:
```
# for T = -10 C:
T=XX
# Try different DeltaT until you find one for which the change in
# radiation (change in RHS of F=\epsilon\sigma{}T^4) is equal to DeltaF:
DeltaT=XX
epsilon=0.6
sigma=5.67e-8
```

```
# define RHS to be epsilon*sigma*T^4 at the warmer temperature minus at the original␣
 ↪temperature:
print("DeltaF=4, T=",T,", DeltaT=",DeltaT,",RHS=",XX)
print("dF/dT=",XX)

# for T = 30 C:
XX
print("dF/dT=",XX)
```

(iii) Plot ΔT as a function of T, in response to a constant ΔF radiative forcing based on the linearization $\Delta F = 4\sigma T^3 \Delta T$.

[ ]:
```
# plot delta T as function of T, in response to a constant deltaF readiative forcing.
# based on the linearization DeltaF=4*sigma*T^3*DeltaT
T=np.arange(240,311,1)
DeltaF=4
DeltaT=XX

fig=plt.figure(figsize=(4,3),dpi=200)
plt.plot(XX)
```

5b) Optional extra credit: Tropical lapse rate feedback:

Calculate the temperature profile of a saturated air parcel being raised in the tropical atmosphere, with initial surface temperatures of 25 °C and 27 °C, and the difference between the profiles. Discuss your results.

[ ]:
```
# PHYSICAL CONSTANTS
PZERO   = 101325.0 # sea-level pressure, N/sq.m
g       = 9.81     # gravity
R_water = 461.52   # gas constant J/kg/K
R_dry   = 287      # gas constant J/kg/K
cp_dry  = 1005     # J/kg/K
L       = 24.93e5  # latent heat J/kg (vaporization at t.p.)


def q_sat(T,P):
    """
    Calculate saturation specific humidity (gr water vapor per gram moist air).
    inputs:
    T: temperature, in Kelvin
    P: pressure, in mb
    """
    R_v = 461 # Gas constant for moist air = 461 J/(kg*K)
    R_d = 287 # Gas constant 287 J K^-1 kg^-1
    TT = T-273.15 # Kelvin to Celsius
    # Saturation water vapor pressure (mb) from Emanuel 4.4.14 p 116-117:
    ew = 6.112*np.exp((17.67 * TT) / (TT + 243.5))
    # saturation mixing ratio (gr water vapor per gram dry air):
    rw = (R_d / R_v) * ew / (P - ew)
    # saturation specific humidity (gr water vapor per gram moist air):
    qw = rw / (1 + rw)
    return qw

## Calculate moist adiabatic temperature profiles for two surface temperatures
```

```
# Define the height coordinates, calculate background pressure and temperature␣
 ↪profiles.
z = np.linspace(0,12,1001) # z in km
p = np.zeros(np.shape(z))  # initialize P array
#T_env = np.zeros(np.shape(z))  # initialize T array

# Assume isothermal atmosphere to calculate the atmospheric pressure at z:
T_const = 273.15
p = PZERO*np.exp(-g*z/R_dry/T_const)/100 # convert pressure to mb for q_sat

def MSE_conservation(T,P,z,MSE_s,q_s):
    """ This function is called by a root finder to calculate
    the temperature of a rising parcel. it is equal to zero when
    MSE conservation is satisfied.
    """
    qsat=q_sat(T,P)
    # the root finder is looking for the value of T for which the following is zero:
    ans = cp_dry*T+L*min(q_s,qsat)+g*z-MSE_s
    return(ans)


def calc_moist_adiabatic_T_profile(p_s,T_s,z_s,RH_s):
    """ Use conservation of MSE to calculate the temperature of a
        adiabatically lifted moist parcel.
    """

    q_s=RH_s*q_sat(T_s,p_s)
    # initial MSE at surface, to be conserved by rising air parcel:
    MSE_s = cp_dry*T_s+g*z_s+L*q_s
    T_parcel = np.zeros(np.shape(z))

    T_parcel[0] = T_s
    for i in range(1,len(z)):
        sol = optimize.root(MSE_conservation, T_parcel[i-1],␣
 ↪args=(p[i],z[i]*1000,MSE_s,q_s))
        T_parcel[i] = np.ndarray.item(sol.x)

    return T_parcel


# initiaize figure and two subplots for temperature and relative humidity:
plt.figure(figsize=(8,6),dpi=200)

plt.subplot(1,2,1)
# calculate and plot first convection profile:
p0 = p[0]
t0 = 273.15 + 25 #t[0]
z0 = z[0]
RH = 1.0
T_profile1 = calc_moist_adiabatic_T_profile(p0,t0,z0,RH)
plt.plot(T_profile1,z,label="$T_s=25$ °C")
```

```
# calculate and plot second convcection profile starting with Ts=27C:
# XX
#XX plt.plot(T_profile2,z,label="$T_s=27$")


# labels and title for first subplot:
plt.legend()
plt.xlabel('$T$ (K)')
plt.ylabel('$z$ (km)')
plt.title('Temperature');

# plot temperature difference as function of height:
plt.subplot(1,2,2)
plt.plot(XX,z)
plt.xlabel('$\Delta T$ (K)')
plt.ylabel('$z$ (km)')
plt.title('Warming');

plt.tight_layout()
```

discussion:

XX

5c) Optional extra credit: Lapse rate feedback: Assume the radiative forcing due to greenhouse gasses is $\Delta F = 8.5$ W/m^2, and that the radiation balance at the emission level is given by the linearized expression $\Delta F = \lambda_{LW} \Delta T_e$. Further assume that the lapse rate feedbacks are such that the tropical (Arctic) warming at the emission height is a factor X ($1/X$) times that at the surface. What should X and λ_{LW} be to explain a surface Arctic amplification of 10K, such that the averaged surface warming in the two areas is 6K?

Solution: XX

6) Stratospheric cooling

6a) Plot the profiles of the temperature averaged zonally and between latitudes of 30°N and 50°N before and after a projected RCP8.5 warming. Contour the zonally averaged temperature change over the 21st century in the RCP8.5 scenario as a function of latitude and height in the troposphere and stratosphere.

[ ]:
```
lat=RCP85_zonally_avg_T_lat
height=RCP85_zonally_avg_T_height

# calculate averaged vertical temperature profiles in midlatitudes:
TA1_midlats=RCP85_zonally_avg_T_2010[:,np.logical_and(lat>30,lat<50)]
TA2_midlats=RCP85_zonally_avg_T_2100[:,np.logical_and(lat>30,lat<50)]
TA1_midlats_avg=np.nanmean(TA1_midlats,1)
TA2_midlats_avg=np.nanmean(TA2_midlats,1)

fig=plt.figure(figsize=(7,3),dpi=200)
plt.clf()

plt.subplot(1,2,1)
# plot a vertical profile of the temperature before and after global warming:
```

```
plt.plot(XX,XX,color="b",label="2006-2010")
plt.plot(XX,XX,color="r",label="2096-2100")
plt.xlabel('Temperature (K)')
plt.ylim([0,50])
plt.legend()

plt.subplot(1,2,2)
# plot latitude-hjeight section of warming:
RCP85_zonally_avg_warming=RCP85_zonally_avg_T_2100-RCP85_zonally_avg_T_2010
# contour latitude-height section of warming:
levels = np.arange(-16,17,1)
plt.contourf(XX, XX, XX, levels=levels)
#plt.gca().invert_yaxis()
plt.set_cmap('bwr')
hcb=plt.colorbar()
plt.xlabel('Latitude')
plt.ylabel('Height (km)')
plt.ylim([0,50])

plt.tight_layout()
plt.show()
```

6b) Calculate the expected tropospheric warming and stratospheric cooling as the tropospheric and stratospheric emissivities change from $\epsilon_{tro} = 0.55$ and $\epsilon_{str} = 0.1$ to $\epsilon_{tro} = 0.6$ and $\epsilon_{str} = 0.15$, for an increase in CO_2, following the section on Stratospheric cooling in the textbook.

[ ]:
```
# constants:
S0=1361.0
sigma=5.67e-8
beta_str=0.05
# present day:
epsilon_tro0=0.55
epsilon_str0=0.1
# higher CO2:
epsilon_tro1=XX
epsilon_str1=XX

# temperatures before CO2 increase:
epsilon_tro=1.0*epsilon_tro0; epsilon_str=1.0*epsilon_str0
# use the matrix representation of energy balance equation including the stratosphere
# from course notes:
A = np.array([[ sigma, -epsilon_tro*sigma, -(1-epsilon_tro)*epsilon_str*sigma
 ↪],
              [ XX, XX, XX ],
              [ XX, XX, XX] ])
b=np.array([[(1-beta_str)*0.25*S0],[XX],[XX]])
X=np.linalg.inv(A)@b
T0=X**0.25
print(T0)

# temperatures after CO2 increase:
epsilon_tro=1.0*epsilon_tro1; epsilon_str=1.0*epsilon_str1
A = np.array([[ XX ] ])
b=XX
```

```
X=XX
T=XX
print(T)
```

6c) optional extra credit: Calculate each term in the stratospheric energy balance before and after the CO_2 increase (increase in emissivities in third equation in 3.6). Show that if the stratospheric temperature does not change, the increase in the RHS term representing outgoing LW from the stratosphere (both up and down) due to the increase in stratospheric emissivity is larger than the increase in the incoming radiation on the LHS. Deduce that the stratosphere must cool to maintain a balance.

[ ]:
```
# terms in Stratospheric temperature equation:

# before emissivity increase
T,T_tro,T_str=T0
epsilon_tro=epsilon_tro0
epsilon_str=epsilon_str0
solar0=beta_str*S0/4
surface_LW0=epsilon_str*(1-epsilon_tro)*sigma*T**4
tropos_LW0=XX
outgoing_LW0=XX
print("before:\n solar=",solar0,"surface_LW=",surface_LW0 \
      ,"tropos_LW=",tropos_LW0,"outgoing_LW=",outgoing_LW0)


# after emissivity increase
T,T_tro,T_str=T1
epsilon_tro=epsilon_tro1
epsilon_str=XX
solar1=XX
surface_LW1=XX
tropos_LW1=XX
outgoing_LW1=XX
outgoing_LW1_no_cooling=2*epsilon_str*sigma*T0[2]**4
print("after:\n solar=",solar1,"surface_LW=",surface_LW1 \
      ,"tropos_LW=",tropos_LW1,"outgoing_LW=",outgoing_LW1)

print("change:\n solar=",XX-XX,"surface_LW=",XX-XX \
      ,"tropos_LW=",XX-XX,"outgoing_LW=",XX-XX)

print("change to outgoing LW with no stratospheric cooling="
      ,outgoing_LW1_no_cooling-tropos_LW0)
```

4 Sea-level

Global Warming Science

https://courses.seas.harvard.edu/climate/eli/Courses/EPS101/

Sea Level

Please use the template below to answer the workshop questions. "XX" indicates places where you need to complete/write code or add a discussion.

your name:

If not using FAS on-demand Jupyter server, required installations first:

from a terminal try: conda install cartopy pip install gsw

Another option for cartopy: in Anaconda Navigator, click Environments, then change in the pull-down menu "Installed" to "All"; find cartopy, mark it for installation and click Apply.

If both of these options do not work, try "pip install cartopy" instead of "conda install cartopy" for cartopy.

[ ]:
```
# import needed libraries and load data:
import numpy as np
import numpy.matlib as matlib
import pickle
import matplotlib.pyplot as plt
import scipy.integrate as integrate
import gsw
import cartopy.crs as ccrs
import cartopy.feature as cfeature
from cartopy.feature import NaturalEarthFeature
from cartopy import config
from matplotlib.colors import BoundaryNorm
from matplotlib import ticker

# Read 2-d sea surface data for historical and rcp85 experiment
with open('./sea_level_variables.pickle', 'rb') as file:
    d = pickle.load(file)
    # print information about each extracted variable:
    for key in list(d.keys()):
        print("extracting pickled variable: name=", key,  "; size=", d[key].shape)
        #print("type=",type(d[key]))
    globals().update(d)
```

Explanation of input variables:

sea level as a function of time, latitude and longitude:

sealevel_historical, sealevel_historical_years

sealevel_rcp85, sealevel_rcp85_years

the corresponding lon/lat axes:

sealevel_lon, sealevel_lat

ocean temperatures as a function of depth, latitude and longitude:

Temperature_ocean_1850, Temperature_ocean_2100,

and the corresponding three axes:

Temperature_ocean_lon, Temperature_ocean_lat, Temperature_ocean_lev,

global mean sea level time series and corresponding time axes:

GMSL_thermosteric_historical, GMSL_thermosteric_historical_years,

GMSL_thermosteric_rcp85, GMSL_thermosteric_rcp85_years,

GMSL_since_1700_years, GMSL_since_1700,

Comments:

1) sealevel_historical and sealevel_rcp8.5 both have zero spatial mean, and therefore represent only the deviation from this mean. The GMSL itself (contribution due to ocean warming) is given by GMSL_thermosteric_historical and GMSL_thermosteric_rcp85.
2) areacello gives the area of each grid cell, used to calculate spatial averates
3) GMSL_since_1700 contains the Jevrejeva et al (2008) data

1) Characterizing sea level rise:

1a) GMSL: Plot the globally-averaged sea level anomaly from 1850 to 2100 for his- torical and then RCP8.5 scenarios, and a quadratic polynomial to it. Discuss how and why the rate of the sea level rise in the two scenarios changes over this period.

[ ]:
```
# combine historical and rcp85 records into one:
years=np.hstack([GMSL_thermosteric_historical_years,GMSL_thermosteric_rcp85_years]);
SSH=np.hstack([GMSL_thermosteric_historical,GMSL_thermosteric_rcp85])

## least square fit of the time series to a parabola
# x=np.polyfit(XX,XX,2);
SSH_fit=x[0]*years**2+x[1]*years+x[2];

print("SSH_fit=x[0]*years**2+x[1]*years+x[2]; where x=",x[0],x[1],x[2])

# plot GMSL and fit:
plt.figure(dpi=200)
plt.plot(XX,XX,"b",lw=1,label="thermosteric historical")
plt.plot(XX,XX,"r",lw=1,label="thermosteric rcp8.5")
plt.plot(XX,XX,'g--',label="quadratic fit")
plt.xlabel('Year')
plt.ylabel('Sea level (m)');
plt.legend();
plt.title('GMSL');
```

discussion:

XX

1b) Spatial structure sea level rise: contour the spatial structure of sea level rise across the historical period and across the RCP8.5 period, explain your results, in particular discuss the region of the Southern Ocean. Read carefully the information about the workshop variables.

[ ]:
```
# contour sea level changes across historical and rcp85 periods:
# preliminaries, defining configuration of subplots:
projection=ccrs.PlateCarree()
fig,axes=plt.subplots(1,2,figsize=(12,6)\
                      ,subplot_kw={'projection': projection},dpi=200)
```

```
cmap = plt.get_cmap('jet')

# plot historical GMSL:
axes[0].set_extent([0, 359.999, -90, 90], crs=ccrs.PlateCarree())
axes[0].coastlines(resolution='110m')
axes[0].gridlines()
c=axes[0].pcolormesh(sealevel_lon,sealevel_lat\
               ,(sealevel_historical[-1,:,:]-sealevel_historical[0,:,:])\
                +(GMSL_thermosteric_historical[-1]-GMSL_thermosteric_historical[0]) \
               ,vmin=-0.4,vmax=0.4\
                     ,cmap="bwr"\
              ,transform=ccrs.PlateCarree())
clb=plt.colorbar(c, shrink=0.45, pad=0.02,ax=axes[0],label="meter")
axes[0].set_title("sealevel rise over years "+repr(sealevel_historical_years[-1])
                                        +"-"+repr(sealevel_historical_years[0]))

# plot rcp8.5 GMSL:
axes[1].set_extent([0, 359.999, -90, 90], crs=ccrs.PlateCarree())
XX
axes[1].set_title("sealevel rise over years "+repr(sealevel_rcp85_years[-1])
                                        +"-"+repr(sealevel_rcp85_years[0]))

plt.subplots_adjust(top=0.92, bottom=0.08, left=0.01, right=0.95, hspace=0.15,wspace=0.
 ↪1)
plt.show();
```

2) Temperature, density and sea level rise.

2a) Plot ocean water density as function of temperature for $T = -2, \ldots, 30$ °C. Plot the expansion coefficient α ($^\circ\mathrm{C}^{-1}$) for the same temperature range. Discuss the implications on sea level rise of the dependence of the expansion coefficient on temperature.

[ ]:
```
T=np.arange(-2,30,0.1)
S=T*0+35
P=T*0
rho=gsw.rho(S,T,P)
# calculate the thermal expansion coefficient at the specified salinity,
# temperature and pressure:
alpha=gsw.alpha(S,T,P)

print("alpha(2C),alpha(10C)=",XX,XX
      ,"Delta rho(3-2C) Delta rho(11-10C)=",XX,XX)
plt.figure(figsize=(8,4),dpi=150)
plt.subplot(1,2,1)
plt.plot(XX,XX)
plt.xlabel("Temperature (C)")
plt.ylabel("density (kg/m$^3$)")

plt.subplot(1,2,2)
plt.plot(XX,XX)
plt.xlabel("Temperature (C)")
plt.ylabel("alpha (C$^{-1}$)")

plt.tight_layout()
```

```
plt.show();
```

Discussion: XX.

2b) Assuming warming is exponential in depth, $\delta T(z) = 4\exp(z/500)$, and extending to the ocean bottom at 4 km. Calculate the expected sea level rise. Assume the equation of state is $\rho = \rho_0\,[1 - \alpha\,(T - T_0)]$, in which $\rho_0 = 1025$ kg/m^3, $\alpha = 1.668 \times 10^{-4}$ °C^{-1} and $T_0 = 10$ °C. What is the expected sea level rise if the exponential decay scale is 1000 m instead of 500 m? Explain.

[ ]:
```
# calculate the expected GMSL rise:
h=4000.0; # average depth of the ocean
alpha=XX; # thermal expansion coefficient at T=10C [alpha is approximated to be a␣
 ↪constant here]
decay_scale=500
# use one of two approaches:
# (1) library integration routine [integrate.quad(...)] or
# (2) using the analytic integral of the exponential function.
N=101;
z=np.arange(-h,0,h/N)
Delta_T=XX
Delta_GMSL=XX
print("The warming will lead to %5.3g meters of sea level rise." % Delta_GMSL[0])

# plot the warming temperature profile:
plt.figure(dpi=200)
plt.plot(XX,XX)
plt.xlabel("$\Delta T(z)$, warming ($^\circ$C)")
plt.ylabel("Depth (m)")
plt.show();
```

[ ]:
```
# calculate the expected GMSL rise again, with a decay_scale of 1000 m:
XX

# plot the warming temperature profile:
XX
```

2c) Sea ice and sea level rise: Consider a 1 cm^3 ice cube floating in a cup of seawater whose radius is 5 cm and where the water height is 10 cm. Calculate the change in water level in the cup when the ice cube melts, assuming first that the seawater is of salinity of 35 ppt, and then when the cup is filled with fresh water. Assume that the melt water mixes with the water in the cup and that the density is linearly related to salinity as $\rho(T_0, S) = \rho_0(1 + \alpha_S(S - S_0))$ with $S_0 = 35$ ppt, $\rho_0 = 1$ gr/cm^3, and $\alpha_S = 0.8$ ppt^{-1}.

[ ]:
```
# Given
S0 = 35            # ppt
rho0 = 1025        # kg m^-3
rho_ice = XX       # kg m^-3
alpha_S = 0.8      # ppt^-1
radius = 0.05      # m
height = 0.10      # m
cube_size=XX       # m
d0 = height        # m

# calculate the mass of ice present in the ice cube
mass_melt = XX
```

```
# Calculate the initial mass occupied by the water in the cup:
# (calculate it from the water volume, but take into account the
# water displaced by the ice)
mass_water = XX

# calculate the new salinity by taking a mass weighted average of the water in the cup:
S_new = XX

# Calculate the new density of the water due to this change in salinity
rho_new = XX

# use the equation for change in depth with change in density
deltad = XX

print('the water level change is ', 100*deltad, ' cm')

# Now perform the same calculation assuming the cup is filled with freshwater
XX
```

(2d) Optional extra credit: Given a certain amount of heat (in J) added to the climate system, would it cause more sea level rise if it melts land ice or leads to expansion of seawater? Your answer should take into account the initial temperatures of the ice and of the ocean. Contour the ratio of the two possible contributions to sea level rise as a function of these two temperatures.

[ ]:
```
# available heat: J/m^2
Heat=1000

# initial temperatures:
T_i_ocean=0
T_i_ice=-5

# latent heat of freezing and specific heat capacities:
L=334e3 # J/kg
c_p_ice=XX # J/(kg C)
c_p_water=XX # J/(kg C)

# densities:
rho_ice=900
rho_water=1024

# sea level rise due to ocean warming:
Delta_SSH_ocean_warming=XX

# sea level rise due to ice melting:
heat_ice_to_melt_temperature=XX
heat_for_melting=Heat-heat_ice_to_melt_temperature
Delta_SSH_ice_melting=XX

print(" SSH due to ocean warming=",Delta_SSH_ocean_warming
     ,"\n SSH due to ice melting=",Delta_SSH_ice_melting)
```

3) Given the 3d temperature at 1850 and the CMIP5 RCP8.5 estimate at 2100, calculate and plot the local contribution to sea level rise as function of longitude and latitude, and calculate the expected GMSL rise. Use variable {areacello} for the averaging. Assume that salinity is a constant at 35 ppt, considering only the effects of warming.

[ ]:
```
# calculate ocean layer thicknesses to be used in integral for sea level rise:
dZ=np.zeros(len(Temperature_ocean_lev)) # set up an array for layer thicknessess
# calculate all layer thicknesses but first:
dZ[1:]=Temperature_ocean_lev[1:]-Temperature_ocean_lev[0:-1]
# first level thickness:
dZ[0]=Temperature_ocean_lev[0]
# last level thickness does not exist (with N levels, there are N-1 thicknesses):
dZ[-1]=0

# calculate ocean densities before and after the warming:
Salinity=np.zeros(Temperature_ocean_1850.shape)+35
theta1=Temperature_ocean_1850-273.15  # convert to Celsius
pressure=np.zeros(theta1.shape)
rho1=gsw.rho(Salinity,theta1,pressure)

theta2=Temperature_ocean_2100-273.15 # convert to Celsius
rho2=gsw.rho(Salinity,theta2,pressure)

# calculate the sea level rise at each horizontal location:
# Integrate the appropriate density ratio in the vertical to find dSSH locally:
density_ratio=XX
Nz,Ny,Nx=density_ratio.shape
dSSH=np.zeros((Ny,Nx))
for i in range(Nx):
    for j in range(Ny):
        for k in range(Nz):
            # integrate over non-NaN values only!
            if ~np.isnan(rho1[k,j,i]):
                dSSH[j,i]=dSSH[j,i]+XX

# calculate global averaged GMSL rise using weighting by the variable arealcello:
dGMSL=XX # can use np.nansum
print("GMSL rise due to warming is %5.3g m" % dGMSL)

## make a pcolormesh figure of the pattern of SSH anomaly
projection=ccrs.PlateCarree()
fig,axes=plt.subplots(1,1,figsize=(6,6)\
                      ,subplot_kw={'projection': projection},dpi=200)
XX add plot commands here
axes.set_title("local contributions to sea level rise due to warming");
```

4) Wind forcing and the Coriolis force:

a) Consider a storm in the Adriatic Sea north of Monte Gargano, with wind blowing along the length of the sea toward Venice at 30 m/s. Calculate the sea level rise at the far north end of the sea.

[ ]:
```
# define parameters and variables to be use
Rhoa=XX;  # kg/m^3, density of air
Cd=0.0013;  # bulk formula coefficient
U=XX;    # m/s, wind speed
U_km_per_hour=XX
rho0=XX;  # kg/m^3, density of sea water
g=XX;  # m/s^2, acceleration due to gracity
L=460000; # m, the length of the northern Adriatic Sea in meters
h=50; # the approx average depth of the Adriatic Sea north of Monte Gargano, in meters
# calculate wind stress using the bulk formula which says: wind stress=Cd*[atmospheric␣
 ↪density]*[surface wind squared]
tau=XX
# calculate the sea level difference along the whole length of the Adriatic sea
dH=XX

# print results:
print("given a wind speed of U= %3.3g m/s (%3.3g km/hour), the stress is tau= %3.2g N/
 ↪m^2," % (U,U_km_per_hour,tau) )
print("and the sea level rise in the northern edge of the Adriatic Sea is expected to␣
 ↪be %4.2g meters." % dH );
```

(b) Calculate the range of sea level changes expected in the center of a category 4 hurricane whose center pressure is 920 to 944 mb.

XX Perform the analytic calculation here

[ ]:
```
# and print the results here:
print("delta h for 920 mb=",XX,"meters")
print("delta h for 944 mb=",XX,"meters")
```

c) Calculate the change in sea level difference across the Gulf Stream at a latitude 30N in response to a weakening of the current by 10%. Assume the original flow was 1 m/s, and that the width of the Gulf Stream there is 50 km.

XX Perform the analytic calculation here

[ ]:
```
# and print the results here:
print("delta h=",XX,"meters")
```

5) Sea-level gravitational fingerprint:

Consider the sea level fingerprints in figure 4.7a,b. Where is melting happening? Explain these sea level change patterns.## 5) Sea-level gravitational fingerprint:

XX

6) Optional extra credit: Decadal variability vs climate change.

Reproduce Figure 2 of Jevrejeva et al. (2008) using their data that is available here as the GMSL since 1700. Discuss: what are the implications of your analysis on the interpretation of the sea level acceleration during

the most recent 20 years presented in Figure 1 of Nerem et al. (2018)? (see bibliography section of course notes)

[ ]:
```
XX
```

5 Acidification

Global Warming Science

https://courses.seas.harvard.edu/climate/eli/Courses/EPS101/

Acidification

Please use the template below to answer the workshop questions. "XX" indicates places where you need to complete/write code or add a discussion.

your name:

[ ]:
```
# Plot carbonate species as C_T varies:
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
import pickle
import cartopy.crs as ccrs
import cartopy.feature as cfeature
from cartopy.feature import NaturalEarthFeature
from cartopy import config
from mpl_toolkits.axes_grid1 import make_axes_locatable

# load variables from a pickle file:
with open('./acidification_variables.pickle', 'rb') as file:
        d = pickle.load(file)
        # print information about each extracted variable:
        for key in list(d.keys()):
            print("extracting pickled variable: name=", key, "; size=", d[key].shape)
            #print("; type=",type(d[key]))
        globals().update(d)

# print ionization/ dissociation constants:
temp=15
sal=35
patm=1
depth=0
print("T=%g, S=%g, depth=%g, Kh=%g, K1=%g, K2=%g,  Kw=%g, Kspc=%g, Kspa=%g.\n"
      % (temp, sal, depth, Kh, K1, K2,  Kw, Kspc, Kspa))
```

Explanation of input variables

Time series of observed and projected CO2 and time axes and the corresponding time axes:

CO2_obs_global_decadal_mean, CO2_obs_global_decadal_mean_years,

CO2_rcp85_global_decadal_mean, CO2_rcp85_global_decadal_mean_years

Time series of observed and projected global-mean ocean pH:

pH_obs_global_decadal_mean, pH_obs_global_decadal_mean_years,

pH_rcp85_global_decadal_mean, pH_rcp85_global_decadal_mean_years

pH maps: observed and projected:

pH_obs_1850_map, pH_obs_2000_map, pH_rcp85_2100_map,

the corresponding axes:

pH_map_latitude pH_map_longitude

equilibrium constants for the ocean carbonate system:

Kh: Henry's Law constant,

K1: first dissociation constant of H2CO3,

K2: second dissociation constant of H2CO3,

Kw: water dissociation constant

Kspa: solubility constant for CaCO3 (aragonite),

Kspc: solubility constant for CaCO3 (calcite),

Exact solution of the carbonate system as a function of pH for fixed Alkalinity: this is the solution obtained the full equations using a dedicated software (mocsy).

exact_alk, exact_co2, exact_co3, exact_hco3, exact_pH, exact_pco2

Comments:

1) Units are concentrations in millimol per kg (approximately millimol/liter).
2) exact_co2 is the dissolved CO_2, denoted $H_2CO_3^*$ in the notes.
3) exact_pco2 is CO_2(g) in ppm.

1) Characterizing pH changes:

(a) Plot a time series of pH from 1850 to 2100, combining observations and RCP8.5 projection

[ ]:
```
# plot:
plt.figure(figsize=(12,5))
#plt.plot(XX)
#plt.plot(XX)
plt.xlabel("year")
plt.ylabel("pH")
plt.legend();
```

(b) Calculate the percent change in $[H^+]$ concentration involved in the pH change observed so far from 8.16 to 8.06.

[ ]:
```
XX
```

c) Plot pH maps for 1850, 2000 (observed) and 2100 (rcp8.5):

[ ]:
```
# plot contours for 1850:
plt.figure(dpi=200)
levels=np.arange(7.2,8.4,0.01)
c=plt.contourf(pH_map_longitude, pH_map_latitude, pH_obs_1850_map,levels)
# draw the colorbar
clb=plt.colorbar(c, shrink=0.85, pad=0.02)
# add title/ labels:
plt.xlabel('Longitude')
plt.ylabel('Latitude')
clb.set_label('pH')
plt.title('pH observed 1850')
plt.show()

# plot contours for 2000:
```

```
#XX
plt.title('pH observed 2000')
plt.show()

# plot contours:
# XX
plt.title('pH rcp8.5 2100')
plt.show()
```

d) Discuss the amplitude of current spatial variations versus expected temporal changes in the context of the expected robustness of ocean biology to acidification. XX

e) Calculate and contour the percentage change in [H+] concentration due to pH changes from year 1850 to 2000 and to 2100.

[ ]:
```
#H_1850 = XX # pH is -Log10[H+]
#H_2000 = XX # pH is -Log10[H+]
# Contour the percentage change
#XX

plt.title('Percentage change in [H+] between 1850 and 2000')
plt.show();
```

2) The carbonate system solution

a) Plot the given precise solution of the carbonate species CO_3^{2-}, HCO_3^-, H_2CO_3 and the total CO_2, as function of pH, for the case of fixed Alk and varying C_T.

[ ]:
```
# print sample values in order to identify range of valid approximation:
print("\n\n        pH,           TOT_CO2,            pco2")
totCO2_exact=exact_hco3+exact_co3+exact_co2
for i in range(0,len(exact_pH),15):
    print(exact_pH[i],"    ",totCO2_exact[i])# XX add more variables to print as needed

# plot the exact solution for the different species using different colors:
plt.plot(exact_pH,exact_hco3,label="HCO3$^-$")
#XX
#plt.xlim(XX)
#plt.ylim(XX)
plt.xlabel("pH")
plt.legend();
```

2b) Plot separately the pH and carbonate ion as function of atmospheric CO_2 from the same solution.

[ ]:
```
# set defaulty font size:
fig=plt.figure(figsize=(5,3),dpi=300)
# plot pH:
axis1=plt.gca()
axis2 =axis1.twinx()
h1=axis1.plot(XX,XX,color="b",label="pH")
axis1.set_ylabel("pH",color="b")
axis1.set_xlabel("CO$_2$(g) (ppm)")

# plot CO_3^{2-}:
h2=axis2.plot(XX,XX,color="r",label="CO$_3^{2-}$")
```

```
axis2.set_ylabel("CO$_3^{2-}$ (milimole/liter)",color="r")

plt.tight_layout()

plt.show()
```

3) The carbonate system buffer effect

plot the expected change to the total dissolved CO_2 (that is, to C_T) as a function of the atmospheric CO_2 concentration in the range 200 to 1000 ppm, based on the given exact solution. Superimpose the expected change calculated based on Henry's law alone (in this case, to $H_2CO_3^*$ only), that would have occurred without the carbonate system reactions. Discuss.

[ ]:
```
# understanding buffer capacity
print("Henri's constant: Kh=",Kh,"; Kh*280/1000=",Kh*280/1000)
fig=plt.figure(figsize=(7,3),dpi=300)

plt.subplot(1,2,1)
# ------------------------------------------------------------
# plot dissolved CO2 vs atmospheric pCO2 in both scenarios:
# ------------------------------------------------------------
plt.plot(exact_pco2,XX,color="b",label="CO$_2$(aq)")
# need to convert pCO2 to ppt for calculation given units of Kh and CO2(aq):
CO2_aq_withouot_buffer=XX
plt.plot(exact_pco2,CO2_aq_withouot_buffer,color="g",label="CO$_2$(aq) w/o buffer")
plt.ylabel("CO$_2$(aq) mm/l")
plt.xlabel("CO$_2$(g) (ppm)")
plt.ylim(0,0.05)
plt.xlim(100,1000)
plt.grid(linewidth=0.25)
#plt.grid(linewidth=0.25)
plt.legend()


plt.subplot(1,2,2)
# ------------------------------------------------------------------------------------
# plot changes to total dissolved CO2 in both scenarios relative to state at pCO2=280:
# ------------------------------------------------------------------------------------
# find total CO2 from the plot in the answer to question 2:
totCO2_eculate xact_at_280=XX
plt.plot(exact_pco2,XX,color="b",label="$\Delta\Sigma$CO$_2$")
# need to convert pCO2 to ppt for calculation given units of Kh and CO2(aq):
Delta_CO2_aq_withouot_buffer=(XX
plt.plot(exact_pco2,Delta_CO2_aq_withouot_buffer,color="g",label="$\Delta$CO$_2$(aq) w/
 ↪o buffer")
plt.ylabel("dissolved/total CO$_2$ (mm/l)")
plt.xlabel("CO$_2$(g) (ppm)")
plt.ylim(-0.3,0.3)
plt.xlim(100,1000)
plt.grid(linewidth=0.25)
plt.legend()

plt.tight_layout()
```

```
plt.show()
```

Discussion:

XX.

4) Understanding the response

Calculate and plot the approximate solution to the carbonate system:

(a) Explain what are the conditions for the approximate solution discussed in section 5.2.2 to be valid, and identify the range of atmospheric CO_2 concentrations for which you expect these conditions to hold.

XX

(b) Use the approximate solution of the carbonate system to calculate the concentration of the carbonate species as function of pH for Alk=2.3 mM/l, in the range of atmospheric CO_2(g) in which you expect the approximate solution to be valid. To do so, vary C_T over an appropriate range of values, and calculate both the carbonate species and pH.

(c) Plot the approximate solution for the pH as function of atmospheric CO_2(g). In both cases, superimpose a plot of the exact solution.

[ ]:
```
def solve_approximate_carbonate_system(C_T,alk):
    """Calculate the approximate solution of carbonate system, solving
    for pH,pco2(=CO2(g)), co2 (=disolved CO2=H2CO3*), hco3 and co3 as
    a function of total CO2 and alkalinity.

    Inputs are in milimole per liter,
    outputs same, output pco2 is in ppt, needs to be multiplied
    by 10^3 to get it in ppm.

    The carbonate system reaction constants K1,K2,Kh are loaded with the
    data at the beginning of the workshop and are therefore known to
    this function.
    """
    hco3=XX
    co3=XX
    H=XX
    pH=-np.log10(H)
    co2=XX # this is $H_2CO_3^*$
    pco2=XX # this is CO2(g)

    return pH,pco2,co2,hco3,co3


# reference values for inputs for approximate solution:
C_T0=2000e-3
alk0=2300e-3
C_T_range=C_T0*np.arange(0.9,1.15,0.01)
print("\nconsidering C_T over the range of :",␣
  ↪C_T_range[0]*1000,C_T_range[-1]*1000,"micro mole/liter")
N=len(C_T_range)
# initialize array of output values:
approx_alk=np.zeros(N)
```

```
approx_pH=np.zeros(N)
approx_pco2=np.zeros(N)
approx_co2=np.zeros(N)
approx_hco3=np.zeros(N)
approx_co3=np.zeros(N)
i=-1
for C_T in C_T_range:
    i=i+1
    pH,pco2,co2,hco3,co3 = solve_approximate_carbonate_system(alk=alk0, C_T=C_T)
    approx_alk[i]=alk0
    approx_pH[i]=pH
    approx_pco2[i]=pco2
    approx_co2[i]=co2 # this is H2CO3*
    approx_hco3[i]=hco3
    approx_co3[i]=co3
    #print(pH,pco2,fco2,co2,hco3,co3)

# plot calculated approximate solution as function of pH:
plt.figure(1,figsize=(8,5),dpi=200)
approx_totCO2=approx_hco3+approx_co3+approx_co2
plt.plot(approx_pH,approx_hco3,label="HCO$_3^-$",lw=3)
#plt.plot(XX,XX,label="CO$_3^{2-}$",lw=3)
#plt.plot(XX,XX,label="H$_2$CO$_3^*$",lw=3) # this is H2CO3*
#plt.plot(XX,XX,label="$\Sigma$CO$_2$",lw=3)
plt.plot([8.2,8.2],[-0.1,3.0],color="k",linewidth=0.5,linestyle="--",label="pre-indust␣
↪pH")
#plt.plot(XX,XX,"--",label="alk",lw=3)
plt.xlabel("pH")
plt.legend()

# plot exact solution as dotted yellow line:
totCO2_exact=exact_hco3+exact_co3+exact_co2
plt.plot(exact_pH,exact_hco3,":y",label="exact")
plt.plot(exact_pH,exact_co3,":y")
plt.plot(exact_pH,exact_co2,":y") # this is H2CO3*
plt.plot(exact_pH,totCO2_exact,":y")
plt.plot(exact_pH,exact_alk,":y")
plt.xlim(6.5,9.5)
plt.ylim(-0.1,3.5)
plt.legend()

# plot pH vs co2(g):
plt.figure(2,figsize=(8,5))
#XX note units of approx_pco2, and convert to ppm for plot:
#plt.plot(XX,XX,"-r",label='approximate pco2, ppm')
#plt.plot(XX,XX,":y",label='exact pco2')
plt.ylabel('pH')
plt.xlabel('CO$_2$(g)')
plt.title("approx and exact pH as function of atmospheric CO$_2$")
plt.xlim(0,1000)

plt.legend()
plt.ylim(7.5,9.5)
```

(d) A heuristic understanding of the carbonate ion decline for increased atmospheric $CO_2(g)$: First, use the approximate solution of section 5.2.2 to calculate the concentration of all carbonate system ions at a $CO_2(g)$ of 280 ppm and for a 10% increase (308 ppm). (Hint: assume $Alk = 2.2$ mM/l, find the C_T value that results in each of the two $CO_2(g)$ concentrations, and solve using this value). Next, given an increase of 10% in atmopsheric $CO_2(g)$, use Henry's law to show that carbonic acid, $[H_2CO_3^*]$ increases by 10% as well. Then use the equation for the first dissociation to calculate the response of the bicarbonate and H^+ ions. (Hint: the increased concentration of carbonic acid leads to X more mM/l of bicarbonate and X of H^+; use the equilibrium equation between 1.1 $[H_2CO_3^*]$ and its products to derive a quadratic equation for X and solve it.) Finally, taking the calculated values of H^+ and HCO_3^-, use the equation for the second dissociation to calculate the response of the carbonate ion. Explain your results for $[H_2CO_3^*]$, $[HCO_3^-]$, and $\left[CO_3^{2-}\right]$, compare to the full solution of the approximate system for $CO_2(g)$ of 308 ppm, and discuss the limitations of this heuristic discussion.

Solution: XX

Limitations: XX

[ ]:
```
# solving for concentrations before increase in atmospheric CO2:
alk=2.2
# to calc C_T:
# (1) use the C_T curve in the plot for question 4c;
# (2) as a check, sum the relevant carbon species taking values from the same above␣
 ↪plot
C_T=XX
pH,pco2,co2,hco3,co3 = solve_approximate_carbonate_system(C_T,alk)
H=10**(-pH)
print("before perturbation (full approximate solution):\n alk=",alk,"C_T=",C_T,",␣
 ↪pco2=", pco2*1.e3 \
      ,", H=",H," ppm, pH=",pH \
      ,"\n co2,hco3,co3=",co2*1.e3,hco3*1.e3,co3*1.e3)

# after a CO2(g) increase, solving for X in first dissociation:
# equation is X^2+bX+c=0, solution is X=(-b+-sqrt(b^2-4c))/2
b=XX; c=XX
X1=(-b+np.sqrt(b**2-4*c))/2 # we expect a positive X, so negative root is ignored
X2=(-b-np.sqrt(b**2-4*c))/2 # we expect a positive X, so negative root is ignored
print("X1,X2=",X1,X2)
X=1.0*X1
H_after_1st=H+X
pH_after_1st=-np.log10(H_after_1st)
hco3_after_1st=hco3+X   # bicarbonate
print("after 1st dissociation: H_after_1st,pH_after_1st=",H_after_1st,pH_after_1st \
      ,"\n hco3_after_1st=",hco3_after_1st*1.e3);

# solving for second dissociation:
Y=XX
print("Y=",Y)
co3_after_2nd=co3+Y   # carbonate
print("after 2nd dissociation: " \
      ,"\n co3_after_2nd=",co3_after_2nd*1.e3);

# compare to approximate solution with increased CO2(g) of 308 ppm:
alk=2.2
C_T=XX
```

```
pH,pco2,co2,hco3,co3 = solve_approximate_carbonate_system(C_T,alk)
H=10**(-pH)
print("after perturbation (full approximate solution):\n alk=",alk,"C_T=",C_T,",␣
 ↪pco2=", pco2*1.e3 \
      ,", H=",H," ppm, pH=",pH \
      ,"\n co2,hco3,co3=",co2*1.e3,hco3*1.e3,co3*1.e3)
```

5) Estimating observed and future pH from CO_2:

5a) Use the approximate solution with $Alk = 2.3$ mM/liter, varying C_T to calculate the pH and CO_2(g). Find the C_T values that result in the observed CO_2 concentrations at years 1850 and 2000 and the RCP8.5 value at 2100. Deduce the pH values at these years. Compare to the above ocean pH time series for 1850–2100.

[ ]:
```
print("year, pCO2, pH")
print("---------------")
for i in range(0,len(CO2_obs_global_decadal_mean)):
    print(CO2_obs_global_decadal_mean_years[i], int(CO2_obs_global_decadal_mean[i]),␣
 ↪pH_obs_global_decadal_mean[i])

for i in range(0,len(CO2_rcp85_global_decadal_mean)):
   ␣
 ↪print(CO2_rcp85_global_decadal_mean_years[i],int(CO2_rcp85_global_decadal_mean[i]),␣
 ↪pH_rcp85_global_decadal_mean[i])
```

[ ]:
```
# now calculate CO2:
alk=2.3

year=1850.0 # 285 ppm
C_T=2.0
pH,pco2,co2,hco3,co3 = solve_approximate_carbonate_system(C_T,alk)
print("year=",year,"alk=",alk,"C_T=",C_T,", pco2=", pco2*1.e3," ppm, pH=",pH)

year=2000.0 # 369 ppm
C_T=XX
XX
XX

year=2100.0 # 935 ppm
C_T=XX
XX
XX
```

5b) Optional extra credit: Use a similar procedure to the approximate solution of the carbonate system equation in section 5.2.2 to solve for the pH given the Alk and CO2(g) (instead of using given Alk and CT as in that section). The solution involves deriving a quadratic equation for [H+] whose coefficients include Alk and CO2(g) and the various reaction coefficients. Let Alk=2.3 mM/l and plot the expected upper ocean pH given the observed CO2 concentration values from 1850 to present and the RCP8.5 values from present to 2100. Compare to the given pH values for these periods obtained from measurements and more sophisticated modeling by superimposing a plot of the given surface ocean pH for 1850–2100. Solution: Start with the

approximate carbonate equations,

$$\begin{aligned} K_H &= \frac{[\mathrm{H_2CO_3^*}]}{[\mathrm{CO_2(g)}]}, \\ K_1 &= \frac{[\mathrm{HCO_3^-}][\mathrm{H^+}]}{[\mathrm{H_2CO_3^*}]}, \\ K_2 &= \frac{[\mathrm{CO_3^{2-}}][\mathrm{H^+}]}{[\mathrm{HCO_3^-}]}, \\ Alk_C &= [\mathrm{HCO_3^-}] + 2[\mathrm{CO_3^{2-}}], \\ C_T &= [\mathrm{HCO_3^-}] + [\mathrm{CO_3^{2-}}]. \end{aligned} \tag{1}$$

In this case we are given $\mathrm{CO_2(g)}$ and Alk_C. starting to solve we have from the first equation,

$$[\mathrm{H_2CO_3^*}] = \mathrm{XX}$$

and then from the second,

$$[\mathrm{HCO_3^-}] = \mathrm{XX}$$

we can now write the third equation as

$$K_2 = \frac{[\mathrm{CO_3^{2-}}][\mathrm{H^+}]}{\mathrm{K_1 K_H}[\mathrm{CO_2(g)}]/[\mathrm{H^+}]}$$

so that,

$$[\mathrm{CO_3^{2-}}] = \mathrm{XX}$$

The (specified) alkalinity can now be written as

$$Alk_C = XX$$

which we can get a quadratic equation for this concentration,

$$XX[H^+]^2 - XX[H^+] - XX = 0$$

Because the $\mathrm{CO_2}$ and the alkalinity are known, all coefficients are known and we can solve for $[H^+]$ and thus for the pH.

[ ]:
```
def solve_approximate_pH_from_Alk_and_CO2g(alk,pCO2):
    """
    Calculate the approximate solution of the carbonate system, solving for the pH
    as function of pco2(=CO2(g)) and alkalinity.
    Input Alk is in millimole per liter, input pCO2 is in ppt (=ppm/1000).
    """
    # The quadratic equation for pH is ax^2+bx+c=0, where x=[H^+]:
    XX=np.nan # remove this line after replacing XX with proper variables and expressions
    a=XX
    b=XX
    c=XX
    H1=(-b+np.sqrt(b**2-4*a*c))/(2*a)
    # H2=(-b-np.sqrt(b**2-4*a*c))/(2*a) # this is the non-physical solution
    pH=-np.log10(H1)
    co2=XX # this is H2CO3*
    hco3=XX
    co3=XX
    C_T=XX
```

```
    return pH,co2,hco3,co3,C_T


# calculate expected pH from observed and RCP8.5 pCO2:
# ----------------------------------------------------
pH_from_CO2_obs_global_decadal_mean=CO2_obs_global_decadal_mean*0.0
for i in range(0,len(pH_from_CO2_obs_global_decadal_mean)):
    pH,co2,hco3,co3,C_T =␣
 ↪solve_approximate_pH_from_Alk_and_CO2g(alk,CO2_obs_global_decadal_mean[i]/1.0e3)
    pH_from_CO2_obs_global_decadal_mean[i]=pH

pH_from_CO2_rcp85_global_decadal_mean=CO2_rcp85_global_decadal_mean*0.0
for i in range(0,len(pH_from_CO2_rcp85_global_decadal_mean)):
    pH,co2,hco3,co3,C_T =␣
 ↪solve_approximate_pH_from_Alk_and_CO2g(alk,CO2_rcp85_global_decadal_mean[i]/1.0e3)
    pH_from_CO2_rcp85_global_decadal_mean[i]=pH

# plot:
# -----
plt.figure(figsize=(12,5),dpi=200)
plt.
 ↪plot(pH_obs_global_decadal_mean_years,pH_from_CO2_obs_global_decadal_mean,"x",color="blue",label="pH
 ↪from CO$_2$ obs")
plt.
 ↪plot(pH_rcp85_global_decadal_mean_years,pH_from_CO2_rcp85_global_decadal_mean,"+",color="red",label=
 ↪from CO$_2$ rcp8.5")
plt.
 ↪plot(pH_obs_global_decadal_mean_years,pH_obs_global_decadal_mean,"o",color="blue",markerfacecolor='n
 ↪obs")
plt.
 ↪plot(pH_rcp85_global_decadal_mean_years,pH_rcp85_global_decadal_mean,"o",color="red",markerfacecolor
 ↪rcp8.5")
plt.xlabel("year")
plt.ylabel("pH")
plt.title("pH from obs/rcp8.5 compared to approximate solution from CO$_2$")
plt.legend();
```

[ ]:

6 Ocean-circulation

Global Warming Science

https://courses.seas.harvard.edu/climate/eli/Courses/EPS101/

Ocean circulation collapse

Please use the template below to answer the workshop questions. "XX" indicates places where you need to complete/write code or add a discussion.

your name:

[ ]:
```
# import needed libraries:
# ------------------------
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
from scipy.integrate import solve_ivp
import pickle
import datetime

# load data from pickle file:
# ---------------------------
with open('./ocean_circulation_variables.pickle', 'rb') as file:
    d = pickle.load(file)
    # print information about each extracted variable:
    for key in list(d.keys()):
        print("extracting pickled variable: name=", key, "; size=", d[key].shape)
        #print("; type=",type(d[key]))
    globals().update(d)
```

Explanation of variables:

RAPID time series and time axis (given both as dates and as time in days):

RAPID_AMOC_data, RAPID_time, RAPID_dates

RAPID monthly climatology and parameters of a fit to a sine function:

RAPID_AMOC_monthly_climatology, RAPID_sine_fit_coeffs,

The sine fit is given as $a_0 + a_1 \sin(2\pi(t_{years} + a_2)/365)$, where a=RAPID_sine_fit_coeffs and t_{years} is RAPID_time.

AMOC stream function for 1st and last decades of 21ast century as a function of depth and latitude, with corresponding axes:

rcp85_AMOC_first_decade, rcp85_AMOC_last_decade, rcp85_depth, rcp85_latitude

Time series of AMOC projections and corresponding time axis:

rcp85_AMOC_timeseries, rcp85_years

AMOC value from ship obs and the corresponding times:

ship_obs_AMOC, ship_obs_months, ship_obs_years

1) Observations: Has the AMOC decreased already over the past decades?

1a) Estimate the heat transport due to AMOC (Watts) by assuming it to be composed of poleward transport of 20 Sv in the upper 1000 m and a return flow below. Assume each of

these two flows to carry water of a temperature that represents the average over these two depth ranges in the North Atlantic, as seen in the figure in Box 6.1. Compare to the world energy consumption per second.

```
[ ]: # heat transport
c_p=XX # J/(kg C)
# transport is 20 Sv or XX kg/s:
# = mass transport (kg/s) * c_p (J/kg) * (T_{upper ocean} -T_{lower ocean}) (K)
heat_transport=XX

# The annual global energy consumption (J/s):
world_consumption=XX

print(" AMOC transport=",heat_transport
      ,"\n world concumption per second=",world_consumption
      ,"\n AMOC/world consumption=",heat_transport/world_consumption)
```

1b) Draw the AMOC estimate based on historical ship observations as a function of year.

```
[ ]: # plot ship_obs data:
# -----------------
plt.figure(figsize=(4,3),dpi=300)
plt.plot(XX,XX,'-x',markersize=10)
plt.xlabel("time (years)")
plt.ylabel("Sv");
plt.title("ship obs");
```

1c) Draw the observed RAPID AMOC time series and the seasonal sine fit to it.

```
[ ]: # plot AMOC obs:
fig=plt.figure(figsize=(3.8,1.8),dpi=300)
plt.plot(XX,XX,color="b",label="RAPID obs, smoothed")
plt.xlabel("Year")
plt.ylabel("AMOC (Sv)")

# add sine fit: sine_fit=a_0+a_1*sin(2*pi*(time+a_2)/365)
a=1.0*RAPID_sine_fit_coeffs
RAPID_AMOC_sine_fit=XX
plt.plot(XX,XX,color="g",label="sine fit")
plt.legend()
plt.title("RAPID AMOC time series and sine fit to the data")
plt.show();
```

1d) Plot the monthly climatology of RAPID and the sine fit. Superimpose the few available AMOC transport estimates based on historical ship observations as if all observations were taken during different months of one year. Discuss your results.

```
[ ]: # Plot one year of AMOC monthly climatology with ship_obs data superimposed:
# ------------------------------------------------------------------------
fig=plt.figure(figsize=(6,4),dpi=200)
time_months=(0.5+np.arange(0,12))
time_sine_fit_months=np.arange(0,12,0.1)
plt.plot(time_months,XX,'x',color=XX,markersize=15,label="RAPID monthly climatology")
sine_fit_one_year=XX
plt.plot(time_sine_fit_months,sine_fit_one_year,label="sine fit".color=XX)
```

```
# superimpose ship observations:
X=ship_obs_months[:]
Y=ship_obs_AMOC[:]
plt.scatter(XX,XX,label="AMOC ship obs")
for i in range(len(X)):
    plt.text(X[i],Y[i],str(ship_obs_years[i]))
plt.xlabel("time (months)")
plt.ylabel("Sv")
plt.xlim(0,12)
plt.legend();
plt.show()
```

Discuss: XX

2) Projections:

2a) Plot the RCP8.5 projected AMOC time series (of maximum stream function) until 2100.

[ ]:
```
plt.figure(figsize=(8,4),dpi=200)
plt.plot(XX,XX)
plt.xlabel("year")
plt.ylabel("Sv")
plt.title("max AMOC time series")
plt.show()
```

2b) Contour the AMOC streamfunction averaged over the first decade of the RCP8.5 scenario (2006–2016), as well as averaged over the last decade of this century (2090–2100). Discuss.

[ ]:
```
plt.figure(figsize=(12,6))
levels=np.arange(-8,28,2) # in Sverdrup, adjust as needed

plt.subplot(1,2,1)
plt.contourf(rcp85_latitude,rcp85_depth,rcp85_AMOC_first_decade,levels)
plt.gca().invert_yaxis()
plt.colorbar()
plt.title("AMOC first decade")
plt.xlim(-40,90)

plt.subplot(1,2,2)
XX contourf last decade and add colorbar as in the first subplot above
plt.title("AMOC, last decade")
plt.xlim(-40,90)

plt.tight_layout()
plt.show();
```

discussion: XX

3) Stommel model:

3a) Plot the steady states of the AMOC for fresh water forcings of Fs = 0,..,5 m/year. Discuss.

[ ]:
```
# first cell out of three: setup parameters and needed functions:
# ---------------------------------------------------------------

# set parameters:
```

```
meter=1;
year=365*24*3600;
S0=35.0;
DeltaT=-10;
time_max=10000*year;
alpha=0.2; # (kg/m^3) K^-1
beta=0.8; # (kg/m^3) ppt^-1
k=10e9 # (m^3/s)/(kg/m^3) #  3e-8; # sec^-1
# area of each of the ocean boxes, rediculous value, needed to get
# all other results at the right order of magnitude:
area=(50000.0e3)**2
depth=4000
V=area*depth # volume of each of the ocean boxes
Sv=1.e9 # m^3/sec


#############################################################################
def q(DeltaT,DeltaS):
    #############################################################################
    # THC transport in m^3/sec as function of temperature and salinity
    # difference between the boxes
    flow=XX # this is "q" from the course notes
    return flow


#############################################################################
def steady_states(Fs,X):
    #############################################################################
    # 3 steady solutions for Y:
    y1=XX
    if XX>0:
        y2=XX
        y3=XX
    else:
        y2=np.nan
        y3=np.nan
    Y=np.array([y1,y2,y3])
    return Y

#############################################################################
def Fs_func(time,time_max,is_Fs_time_dependent):
    #############################################################################
    # total surface salt flux into northern box

    #############################################################################
    # Specify maximum and minimum of freshwater forcing range during the
    # run in m/year:
    FW_min=-0.1;
    FW_max=5;
    #############################################################################
    if is_Fs_time_dependent:
            flux=FW_min+(FW_max-FW_min)*time/time_max;
    else:
```

```
        flux=2;

    # convert to total salt flux:
    return flux*area*S0/year

print("done procedssing function definitions.")
```

[ ]:
```
# second cell out of three: calculate:
# -------------------------------------

# find steady states, including unstable ones, as function of F_s:
# ------------------------------------------------------------------

Fs_range=np.arange(0,5,0.01)*area*S0/year
# set up two arrays for results, to be calculated in loop below:
DeltaS_steady=np.zeros((3,len(Fs_range)))
q_steady=np.zeros((3,len(Fs_range)))

i=0
for Fs in Fs_range:
    Y=steady_states(Fs,alpha*DeltaT)
    # translate Y solution to a solution for DeltaS:
    DeltaS_steady[:,i]=XX
    for j in range(0,3):
        q_steady[j,i]=q(DeltaT,DeltaS_steady[j,i])
    i=i+1

print("q_steady.shape=",q_steady.shape,"; Example q_steady values=",q_steady[:,100])
```

[ ]:
```
# third cell out of three: plot
# -----------------------------

plt.figure(1,figsize=(8,4),dpi=200)

Fs_to_m_per_year=S0*area/year
plt.subplot(1,2,1)
# plot all three solutions for Delta S as function of Fs in units of m/year:
plt.plot(XX,XX,'r.',markersize=1)
plt.plot(XX,XX,'g.',markersize=1)
plt.plot(XX,XX,'b.',markersize=1)
# plot a dash think line marking the zero value:
plt.plot(XX,XX,'k--',dashes=(10, 5),linewidth=0.5)
plt.title('(a) steady states')
plt.xlabel('$F_s$ (m/year)');
plt.ylabel('$\Delta S$');
plt.xlim([min(Fs_range/Fs_to_m_per_year), max(Fs_range/Fs_to_m_per_year)])

plt.subplot(1,2,2)
# plot all three solutions for q (in Sv) as function of Fs in units of m/year:
plt.plot(XX,XX,'r.',markersize=1,label="lower stable branch")
plt.plot(XX,XX,'g.',markersize=1,label="upper stable branch")
plt.plot(XX,XX,'b.',markersize=1,label="unstable branch")
plt.plot(XX,XX,'k--',dashes=(10, 5),linewidth=0.5)
```

```
plt.title('(b) steady states')
plt.xlabel('$F_s$ (m/year)');
plt.ylabel('$q$ (Sv)');

plt.tight_layout()
plt.show()
```

3b) Integrate the time dependent differential equation for S in a scenario of slowly increasing fresh water forcing, plot the resulting AMOC (q) as function of time.

[ ]:
```
##########################################################################
def rhs_S(time,DeltaS,time_max,DeltaT,is_Fs_time_dependent):
    ##########################################################################
    # right hand side of the equation for d/dt(DeltaS):
    # Input: q in m^3/sec; FW is total salt flux into box
    rhs=XX;
    return rhs/V


# next, do a time dependent run:
# ------------------------------
is_Fs_time_dependent=True;
y0=[0]
teval=np.arange(0,time_max,time_max/1000)
tspan=(teval[0],teval[-1])
sol = solve_ivp(fun=lambda time,DeltaS:␣
 ↪rhs_S(time,DeltaS,time_max,DeltaT,is_Fs_time_dependent) \
                ,vectorized=False,y0=y0,t_span=tspan,t_eval=teval)
Time=sol.t
DeltaS=sol.y

is_Fs_time_dependent=True;
FWplot=np.zeros(len(Time))
qplot=np.zeros(len(Time))
i=0;
for t in Time:
    FWplot[i]=Fs_func(t,time_max,is_Fs_time_dependent);
    qplot[i]=q(DeltaT,DeltaS[0,i]);
    i=i+1;

N=len(qplot);


##########################################################################
## plots:
##########################################################################

plt.figure(figsize=(12,4),dpi=200)
plt.subplot(1,3,1)
plt.plot(Time/year/1000,FWplot/Fs_to_m_per_year,'b-',markersize=1)
plt.plot(Time/year/1000,0*Time,'k--',dashes=(10, 5),linewidth=0.5)
plt.xlabel('time (kyr)');
plt.ylabel('$F_s$ (m/yr)');
plt.title('(d) $F_s$ for time-dependent run');
```

```
plt.subplot(1,3,2)
plt.plot(Time/year/1000,qplot/Sv,'b-',markersize=1)
plt.plot(Time/year/1000,Time*0,'k--',dashes=(10, 5),lw=0.5)
plt.xlabel('time (kyr)');
plt.ylabel('THC $q$ (Sv)');
plt.title('(e) THC transport $q$');

plt.subplot(1,3,3)
plt.plot(Time/year/1000,DeltaS[0,:],'b-',markersize=1)
plt.title('(f) $\Delta S$ vs time');
plt.xlabel('time (kyr)');
plt.ylabel('$\Delta S$');


plt.tight_layout()
plt.show()
```

discussion: XX

3c) Plot d S/dt as function of S and discuss the stability of the three possible solutions.

[ ]:
```
# calculate dDeltaS/dt as function of DeltaS for stability analysis:
# ------------------------------------------------------------------

# use fixed (time-independent) value of fresh water forcing for this case:
is_Fs_time_dependent=False
time=0
DeltaS_range=np.arange(-3,0,0.01)
rhs=np.zeros(len(DeltaS_range))
for i in range(0,len(DeltaS_range)):
    DeltaS = DeltaS_range[i]
    rhs[i]=rhs_S(time,DeltaS,time_max,DeltaT,is_Fs_time_dependent)
    i=i+1
rhs=rhs/np.std(rhs)


plt.figure(figsize=(5,4),dpi=200)
plt.plot(DeltaS_range,rhs,'k-',lw=2)
plt.plot(DeltaS_range,rhs*0,'k--',dashes=(10, 5),lw=0.5)
# superimpose color markers of the 3 solutions
Fs=Fs_func(0.0,0.0,False)
yy=steady_states(Fs,alpha*DeltaT)/beta
plt.plot(yy[0],0,'ro',markersize=10)
plt.plot(yy[1],0,'go',markersize=10)
plt.plot(yy[2],0,'bo',markersize=10,fillstyle='none')
plt.title('(c) stability')
plt.xlabel('$\Delta S$');
plt.ylabel('$d\Delta S/dt$');
```

discussion: XX

3d) Explain how tipping points and hysteresis may occur in this model.

XX explanation

7 Clouds

Global Warming Science

https://courses.seas.harvard.edu/climate/eli/Courses/EPS101/

Clouds!

Please use the template below to answer the workshop questions. "XX" indicates places where you need to complete/write code or add a discussion.

your name:

[ ]:
```
# import libraries and get data:
import sys
import numpy as np
from matplotlib import pyplot as plt
from scipy import optimize
import pickle
from scipy.integrate import solve_ivp
# cortopy allows to plot data over maps in various spherical prjections"
import cartopy.crs as ccrs
import cartopy.feature as cfeature
from cartopy.feature import NaturalEarthFeature
from cartopy import config
from mpl_toolkits.axes_grid1 import make_axes_locatable

# load the data from a pickle file:
with open('./clouds_variables.pickle', 'rb') as file:
    while 1:
        try:
            d = pickle.load(file)
        except (EOFError):
            break
            # print information about each extracted variable:
        for key in list(d.keys()):
            print("extracting pickled variable: name=", key,  "; size=", d[key].shape)
            #print("type=",type(d[key]))
        globals().update(d)
```

Explanation of input data:

The input variables are results from two prominent climate models (GFDL, Geophysical Fluid Dynamics Laboratory in Princeton NJ, and the Hadley center in the UK. The variable Delta_LW_CRF_GFDL, for example, then represents

$$\Delta CRF_{LW} = CRF_{RCP8.5,LW} - CRF_{historical,LW}$$

in W/m^2 and similarly for the other cloud variables:

Delta_LW_CRF_GFDL, Delta_SW_CRF_GFDL,

Delta_LW_CRF_Hadley, Delta_SW_CRF_Hadley

The variables Delta_SAT_GFDL/Hadley are the difference in surface air temperature (K) between RFP8.5 at 2100 and historical run at 1850:

Delta_SAT_GFDL, Delta_SAT_Hadley

The variables clouds_... are maps of the fraction of the sky covered with clouds as function of longitude and latitude and the corresponding lat/lon axes. These are given for "historical" which is pre-wrming and for the RCP8.5 projection:

clouds_GFDL_historical, clouds_GFDL_rcp85,

clouds_Hadley_historical, clouds_Hadley_rcp85

The lat/lon axes for all of the above variables are given by:

clouds_GFDL_latitude, clouds_GFDL_longitude,

clouds_Hadley_latitude, clouds_Hadley_longitude

1) Clouds radiative forcing and climate sensitivity in climate models: Plot the results of two prominent climate models (from the Geophysical Fluid Dynamics Laboratory, Princeton, NJ, and the Hadley Center, UK), and their difference. Discuss the implications to global warming uncertainty. First, clouds (three contour plots per each of the two models):

(a) The global distribution of cloud fraction at year 1850.

(b) The projected global distribution of cloud fraction at year 2100, according to the RCP8.5 scenario.

(c) The difference between the two.

[ ]:
```
# plot clouds in two models:
# --------------------------

# preliminaries, defining configuration of subplots:
projection=ccrs.PlateCarree(central_longitude=180.0)
fig,axes=plt.subplots(3,2,figsize=(18,9),subplot_kw={'projection': projection},dpi=300)


# GFDL historical:
# ----------------
axes[0,0].set_extent([0, 360, -90, 90], crs=ccrs.PlateCarree())
axes[0,0].coastlines(resolution='110m')
axes[0,0].gridlines()
c=axes[0,0].contourf(clouds_GFDL_longitude,clouds_GFDL_latitude,␣
 ↪clouds_GFDL_historical)
clb=plt.colorbar(c, shrink=0.95, pad=0.02,ax=axes[0,0])
clb.set_label('Cloud fraction %')
axes[0,0].set_title('clouds GFDL historical')


# GFDL rcp8.5:
# ------------
axes[1,0].set_extent([0, 360, -90, 90], crs=ccrs.PlateCarree())
axes[1,0].coastlines(resolution='110m')
axes[1,0].gridlines()
#c=axes[1,0].contourf(XX)
# XX
axes[1,0].set_title('clouds GFDL rcp8.5')
```

```
# GFDL difference:
# ----------------
# XX
clb.set_label('Cloud fraction %')
axes[2,0].set_title('clouds GFDL rcp8.5$-$historical')

# Hadley historical:
# ------------------
axes[0,1].set_extent([0, 360, -90, 90], crs=ccrs.PlateCarree())
axes[0,1].coastlines(resolution='110m')
axes[0,1].gridlines()
c=axes[0,1].contourf(clouds_Hadley_longitude,clouds_Hadley_latitude,␣
 ↪clouds_Hadley_historical)
clb=plt.colorbar(c, shrink=0.95, pad=0.02,ax=axes[0,1])
clb.set_label('Cloud fraction %')
axes[0,1].set_title('clouds Hadley historical')


# Hadley rcp8.5:
# --------------
# XX
clb.set_label('Cloud fraction %')
axes[1,1].set_title('clouds Hadley rcp8.5')


# Hadley difference:
# ------------------
axes[2,1].set_extent([0, 360, -90, 90], crs=ccrs.PlateCarree())
# XX
clb.set_label('Cloud fraction %')
axes[2,1].set_title('clouds Hadley rcp8.5$-$historical')


# finalize and show plot:
plt.subplots_adjust(top=0.92, bottom=0.08, left=0.01, right=0.95, hspace=0.
 ↪15,wspace=-0.4)
plt.show()
```

Next, temperature and CRE (three contour plots for each of the two models):

(d) The global distribution of the change in surface air temperature (SAT) between the preindustrial state (often referred to as historical and representing year 1850) and the RCP8.5 scenario at 2100: SAT =SATRCP8.5 −SAT historical.

(e) Change in LW CRE: CRELW =CRELW,RCP8.5 −CRELW,historical.

(f) Change in SW CRE.

[ ]:
```
# plot CRF in two models:
# -----------------------------

# preliminaries, defining configuration of subplots:
projection=ccrs.PlateCarree(central_longitude=180.0)
```

```
fig,axes=plt.subplots(3,2,figsize=(18,9),subplot_kw={'projection': projection},dpi=300)
contour_levels_SAT=np.arange(-3,24.1,0.1)
contour_levels_CRF=np.arange(-40,41,1)

# GFDL Delta SAT:
# ----------------
axes[0,0].set_extent([0, 360, -90, 90], crs=ccrs.PlateCarree())
#XX
axes[0,0].set_title('GFDL $\Delta$ SAT')


# GFDL Delta CRF SW:
# ------------------
#XX
clb.set_label('CRF (W/M$^2$)')
axes[1,0].set_title('GFDL $\Delta$ CRF$_{SW}$')


# GFDL Delta CRF LW:
# ------------------
contour_levels_diff=np.arange(-20,20.6,0.5)
#XX
clb.set_label('CRF (W/M$^2$)')
axes[2,0].set_title('GFDL $\Delta$ CRF$_{LW}$')

# Hadley Delta SAT:
# ----------------
#XX
axes[0,1].set_title('Hadley $\Delta$ SAT')


# Hadley Delta CRF SW:
# --------------------
#XX
clb.set_label('CRF (W/M$^2$)')
axes[1,1].set_title('Hadley $\Delta$ CRF$_{SW}$')


# Hadley Delta CRF LW:
# --------------------
#XX
clb.set_label('CRF (W/M$^2$)')
axes[2,1].set_title('Hadley $\Delta$ CRF$_{LW}$')


# finalize and show plot:
plt.subplots_adjust(top=0.92, bottom=0.08, left=0.01, right=0.95, hspace=0.
 ↪15,wspace=-0.4)
plt.show()
```

Discuss:

XX

2) Convection and cloud formation: Consider an atmospheric vertical temperature profile with a prescribed lapse rate of 6.5 K/km and an air parcel starting at the surface with a temperature equal to that of the atmospheric profile there and at 70% saturation.

First, some preliminaries:

[ ]:
```
# PHYSICAL CONSTANTS
PZERO   = 101325.0 # sea-level pressure, N/sq.m
g       = 9.81     # gravity
R_water = 461.52   # gas constant J/kg/K
R_dry   = 287      # gas constant J/kg/K
cp_dry  = 1005     # J/kg/K
L       = 24.93e5  # latent heat J/kg (vaporization at t.p.)
print("pressure scale height for Tbar=260 K is",R_dry*260/g)

def SimpleAtmosphere(z):
    """ Compute temperature as a function of height in a simplified standard␣
 ↪atmosphere.
    Correct to 20 km. Approximate thereafter.
    Input: z, geometric altitude, m.
    Output: std. temperature
    """
    TZERO   = 298.15   # sea-level temperature, Kelvin

    if z<11000.0:      # troposphere
        T_a=(298.15-6.5*z/1000)/298.15
    else:       # stratosphere
        T_a=226.65/298.15
    return TZERO*T_a


def q_sat(T,P):
    """
    Calculate saturation specific humidity (gr water vapor per gram moist air).
    inputs:
    T: temperature, in Kelvin
    P: pressure, in mb
    """
    R_v = 461 # Gas constant for moist air = 461 J/(kg*K)
    R_d = 287 # Gas constant 287 J K^-1 kg^-1
    TT = T-273.15 # Kelvin to Celsius
    # Saturation water vapor pressure (mb) from Emanuel 4.4.14 p 116-117:
    ew = 6.112*np.exp((17.67 * TT) / (TT + 243.5))
    # saturation mixing ratio (gr water vapor per gram dry air):
    rw = (R_d / R_v) * ew / (P - ew)
    # saturation specific humidity (gr water vapor per gram moist air):
    qw = rw / (1 + rw)
    return qw
```

2a) The parcel is lifted from the surface to z=3 km. Plot the MSE conservation (LHS vs RHS of equation 7.1) as a function of T, and find the parcel's final temperature as the one for which the LHS equals the RHS. Is the parcel saturated at that point?

[ ]:
```
# specify properties of the parcel at the surface:
T_surf=280.0
z_surf=0.0
RH_surf=0.7
q_surf=RH_surf*q_sat(T_surf,1000)
MSE_surf = cp_dry*T_surf+g*z_surf+L*q_surf

# height and pressure to which parcel is raised:
z=3000 # meters
# Assume isothermal atmosphere to calculate the atmospheric pressure at z:
T_const = 273.15
p = PZERO*np.exp(-g*z/R_dry/T_const)/100 # convert pressure to mb for q_sat

# plot MSE conservation allowing for the parcel to be
# either undersaturated or saturated:
T_plot=np.arange(240.0,280.0,1.0)
MSE_plot=T_plot*0.0
for i in range(0,len(T_plot)):
    MSE=XX
    MSE_plot[i]=MSE

MSE_ssurf_plot=XX # a constant, does not depend on Temprature


plt.figure(figsize=(12,6))
# plot in units of temperature by dividing by cp_dry:
plt.plot(T_plot,MSE_surf_plot/cp_dry,label="XX");
plt.plot(T_plot,MSE_plot/cp_dry,label="XX");
plt.legend(XX)
plt.ylabel(XX)
plt.xlabel("$T$")

# Given the solution for the temperature of the parcel,
# check if the parcel is expected to be saturated:
T_solution= 260 # get this value from the graph
qsat=q_sat(T_solution,p)
print("q_surf,qsat(z)=",q_surf,qsat)
#if XX
#    print("q_surf>qsat, parcel must be saturated at this new level")
#else:
#    print("q_surf<qsat, parcel is not saturated at this new level")
```

2b) Plot the parcel's profiles of temperature and relative humidity as it rises for z=0, 0.1, . . . , 10 km. Repeat for an initial RH of 50, 70, and 90 percent. Describe the results: At what heights are the LCL? LFC? LNB?

[ ]:
```
# Define the height coordinates, calculate background pressure
# and temperature profiles.
z = np.linspace(0,12000,1001) # z in m
p = np.zeros(np.shape(z))  # initialize P array
T_env = np.zeros(np.shape(z))  # initialize T array

# Assume isothermal atmosphere to calculate the atmospheric pressure at z:
T_const = 273.15
```

```
p = PZERO*np.exp(-g*z/R_dry/T_const)/100 # convert pressure to mb for q_sat

for i in range(len(z)):
    T_a = SimpleAtmosphere(z[i])
    T_env[i] = T_a

def MSE_conservation(T,P,z,MSE_s,q_s):
    """ This function is called by a root finder to calculate
    the temperature of a rising parcel. It returns the RHS minus the LHS and
    is equal to zero when MSE conservation is satisfied.
    Input arguments:
        temperature, pressure, height (m) at which conservation is calculated;
        MSE_s is the MSE to which we are trying to match;
        q_s is the specific humidity of the parcel before rising to height=z;
    """
    qsat=q_sat(T,P)
    # the root finder is looking for the value of T for which the following is zero:
    ans = cp_dry*T+L*min(q_s,qsat)+g*z-MSE_s
    return(ans)


def calc_T_and_RH_profiles_in_convection(p_s,T_s,z_s,RH_s):
    """ Use conservation of MSE to calculate the temperature and RH profiles of a
        adiabatically lifted moist parcel.
    """

    q_s=RH_s*q_sat(T_s,p_s)
    # initial MSE at surface, to be conserved by rising air parcel:
    MSE_s = np.nan # XX fix this line
    T_parcel = np.zeros(np.shape(z))
    RH_parcel = np.zeros(np.shape(z))

    T_parcel[0] = T_s
    RH_parcel[0] = RH_s
    for i in range(1,len(z)):
        sol = optimize.root(MSE_conservation, T_parcel[i-1],␣
 ↪args=(p[i],z[i],MSE_s,q_s))
        T_parcel[i] = sol.x
        qsat=q_sat(T_parcel[i],p[i])
        q=min(q_s,qsat)
        RH_parcel[i] = np.nan # min(XX) fix this line

    return(RH_parcel,T_parcel)


# initiaize figure and two subplots for temperature and relative humidity:
plt.figure(figsize=(8,6),dpi=200)
ax1=plt.subplot(1,2,1)
ax2=plt.subplot(1,2,2)

# calculate and plot a convection profile:
p0 = p[0] # surface pressure
T0 = 273.15 + 25 #T[0] surface temperature
```

```
z0 = z[0] # surface height
RH = 0.7 # initial relative humidity
RHpath,T_parcel = calc_T_and_RH_profiles_in_convection(p0,T0,z0,RH)
ax1.plot(T_parcel,z/1000,label="T parcel")
# plot environment temperature profile:
ax1.plot(T_env,z/1000,color='k',linestyle='--',alpha=0.75,label="T environment")  #␣
 ↪Plot atmospheric background.
ax1.legend()
ax1.set_xlabel('T (K)')
ax1.set_ylabel('z (km)')
ax1.set_title('(a) Temperature')
ax1.grid()

# second subplot:
ax2.plot(100*RHpath,z/1000,label="RH")
ax2.legend()
ax2.set_xlabel('RH (%)')
ax2.set_title('(b) Relative Humidity')
ax2.grid();
#ax2.set_ylabel('z (km)')
```

Describe the results for an initial RH of 70 percent:

XX

2c) Qualitatively, no equations: if a volume of rising air entrains the same volume of dry air from its surroundings when at 1 km as it rises, what do you expect the consequence to be for its motion, temperature, RH and LFC? Solution: XX.

2d) Optional extra credit: calculate the parcel profiles of temperature and relative humidity as it rises as above, assuming that the parcel entrains environmental air of a fraction 0.1 of its volume per km. Assume that the environmental air has a lapse rate of 6.5 deg/km and that its RH is 70%. Hint: raise the parcel adiabatically 100 m, entrain air from the environment and calculate the resulting averaged parcel temperature and moisture. adjust the moisture to be no more than the saturation moisture, and repeat.

[ ]:
```
# code here. XX
```

3) Cloud feedbacks in a 2-level energy balance model: Global mean surface temperature is predicted to increase by 2–4 ∘C due to doubling of CO2

3a) Calculate the steady state temperatures for both CO_2=280 without cloud feedbacks, use the solution as the initial conditions as well as the reference temperature in the cloud feedback terms. Now run a warming scenario CO_2 increasing exponentially on a time scale of 150 years until doubling, and continuing at double CO_2. Repeat this run without cloud feedbacks and then with cloud feedbacks, using $\Delta_{LW} = 0.001$ and $\Delta_{SW} = -0.01$. Plot the time series of CO_2, T, T_a, emissivity and albedo for the runs with and without cloud feedbacks. Discuss the effects of cloud feedbacks on climate sensitivity (equilibrium warming due to $\times 2CO_2$) in this case.

[ ]:
```
#########################################################################
def rhs(time,T,C_ocn,C_atm,CO2_initial,CO2_fixed,Delta_SW,Delta_LW):
    #########################################################################
    # right hand side of the equations for T_ocn and T_a:
    T_ocn=T[0]
```

```
    T_a=T[1]
    # calculate the RHS of both equations (rhs is an array of two elements):
    rhs=np.array([ (S*(1-alpha(T_ocn,Delta_SW))
 ↪+epsilon(T_a,Delta_LW,CO2_initial,CO2_fixed,time)*sigma*T_a**4-sigma*T_ocn**4)/C_ocn
                  ,(XX)/C_atm])
    return rhs


##########################################################################
def alpha(T,Delta_SW):
    # albedo as function of temperature representing low cloud effects
    ##########################################################################
    alpha0=0.3
    alph=alpha0*(1+Delta_SW*(T-T_ref))
    return alph


##########################################################################
def epsilon(T_a,Delta_LW,CO2_initial,CO2_fixed,time):
    # atmospheric emissivity as function of temperature, representing high cloud
 ↪effects
    ##########################################################################
    epsilon0=0.75+0.05*np.log2(CO2(CO2_initial,CO2_fixed,time)/280)
    eps=XX
    return np.minimum(1.0,eps)


##########################################################################
def CO2(CO2_initial,CO2_fixed,time):
    # atmospheric CO2 as function of time;
    # if CO2_fixed=True, CO2_initial is used for all times.
    ##########################################################################
    if CO2_fixed:
        CO2=CO2_initial
    else:
        CO2=CO2_initial*np.exp(XX)
    return np.minimum(CO2,560)
```

[ ]:
```
##########################################################################
## Main program starts here.
##########################################################################

# set parameters:
year=365*24*3600;
time_max=200*year;
Cp_w=4000
rho_w=1000
C_ocn=Cp_w*rho_w*50
C_atm=C_ocn/5
sigma=5.67e-8
S=1361/4
CO2_initial=280
```

```
T_ref=281.41
T_a_ref=236.64


# run the energy balance model to steady state for CO2=280
# to get initial conditions for warming scenario:
# ------------------------------------------------------------
T_init=[280,250] # initial conditions
teval=np.arange(0,time_max,time_max/100)
tspan=(teval[0],teval[-1])
# without feedbacks:
Delta_SW=0.0  # try -0.02 to 0.1
Delta_LW=0.0  # try +/- 0.02
CO2_fixed=True
sol = solve_ivp(fun=lambda time,T:␣
 ↪rhs(time,T,C_ocn,C_atm,CO2_initial,CO2_fixed,Delta_SW,Delta_LW) \
                ,vectorized=False,y0=T_init,t_span=tspan,t_eval=teval,rtol=1.0e-6)
time_no_feedback=sol.t
T_280=sol.y[:,-1]
T_ref=T_280[0]
T_a_ref=T_280[1]
print("steady temperature (surface/atmospher) at CO2=280:",T_280)
```

[ ]:
```
# run the energy balance model in time-dependent warming scenario:
# ------------------------------------------------------------------
T_init=[XX,XX] # initial conditions
teval=np.arange(0,time_max,time_max/100)
tspan=(teval[0],teval[-1])
# without feedbacks:
Delta_SW=XX
Delta_LW=XX
CO2_fixed=False
sol = solve_ivp(fun=lambda time,T:␣
 ↪rhs(time,T,C_ocn,C_atm,CO2_initial,CO2_fixed,Delta_SW,Delta_LW) \
                ,vectorized=False,y0=T_init,t_span=tspan,t_eval=teval,rtol=1.0e-6)
time_no_feedback=sol.t
T_no_feedback=sol.y
# with feedbacks:
Delta_SW=XX
Delta_LW=XX
CO2_fixed=False
sol = solve_ivp(fun=lambda time,T:␣
 ↪rhs(time,T,C_ocn,C_atm,CO2_initial,CO2_fixed,Delta_SW,Delta_LW) \
                ,vectorized=False,y0=T_init,t_span=tspan,t_eval=teval,rtol=1.0e-6)
time=sol.t
T=sol.y


# save model output to be plotted:
# --------------------------------
T_plot=np.zeros(len(T[0,:]))
T_a_plot=np.zeros(len(T[0,:]))
T_plot_no_feedback=np.zeros(len(T[0,:]))
```

```
T_a_plot_no_feedback=np.zeros(len(T[0,:]))
alpha_plot=np.zeros(len(T[0,:]))
epsilon_plot=np.zeros(len(T[0,:]))
CO2_plot=np.zeros(len(T[0,:]))
i=0;
for t in T_plot:
    T_plot[i]=T[0,i];
    T_a_plot[i]=T[1,i];
    T_plot_no_feedback[i]=T_no_feedback[0,i];
    T_a_plot_no_feedback[i]=T_no_feedback[1,i];
    alpha_plot[i]=alpha(XX)
    epsilon_plot[i]=epsilon(T[1,i],Delta_LW,CO2_initial,CO2_fixed,time[i])
    CO2_plot[i]=CO2(XX)
    i=i+1;

N=len(T_plot)
```

[ ]:
```
##############################################################################
## plots:
##############################################################################

fig1=plt.figure(1,figsize=(8,4),dpi=150)

plt.subplot(2,2,1)
plt.plot(time_no_feedback/year,CO2_plot,'b-')
#plt.xlabel('years');
plt.ylabel('CO$_2$ (ppm)');
plt.grid(lw=0.25)
axes=plt.gca()
plt.text(0.05, 0.5, "(a)", transform=axes.transAxes, fontsize=14,
        verticalalignment='top')


plt.subplot(2,2,2)
h1=plt.plot(time/year,alpha_plot,'b-',markersize=1,label="$\\alpha$")
plt.grid(lw=0.25)
axis1=plt.gca()
axis2 =axis1.twinx()
h2=axis2.plot(time/year,epsilon_plot,'r-',markersize=1,label="$\\epsilon$")
plt.xlabel(XX);
plt.ylabel(XX);
plt.title(XX);
# added these three lines
h = h1+h2
legend_labels = [l.get_label() for l in h]
axis1.legend(h, legend_labels, loc="center right")
axis1.tick_params(axis='y', labelcolor="b")
axis2.tick_params(axis='y', labelcolor="r")
axes=plt.gca()
plt.text(0.05, 0.5, "(b)", transform=axes.transAxes, fontsize=14,
        verticalalignment='top')
```

```
plt.subplot(2,2,3)
plt.plot(time_no_feedback/year,T_plot_no_feedback,'b--',markersize=1,label="$T$, no␣
 ↪feedback")
plt.plot(time/year,T_plot,'b-',markersize=1,label="T, with feedback")
plt.xlabel(XX);
plt.ylabel(XX);
plt.ylim([286,291.5])
plt.title(XX);
plt.legend()
plt.grid(lw=0.25)
axes=plt.gca()
plt.text(0.05, 0.5, "(c)", transform=axes.transAxes, fontsize=14,
        verticalalignment='top')

plt.subplot(2,2,4)
plt.plot(time_no_feedback/year,T_a_plot_no_feedback,'b--',markersize=1,label="$T_a$,␣
 ↪no feedback")
plt.plot(time/year,T_a_plot,'b-',markersize=1,label="$T_a$, with feedback")
plt.xlabel(XX);
plt.ylabel(XX);
plt.ylim([240,245])
#plt.title('$T_a$');
plt.legend()
plt.grid(lw=0.25)
axes=plt.gca()
plt.text(0.05, 0.5, "(d)", transform=axes.transAxes, fontsize=14,
        verticalalignment='top')


plt.tight_layout()
plt.show()

#fig1.savefig("Output/clouds-energy-balance-with-cloud-feedbacks.pdf")
```

3b) Estimate the SW cloud feedback parameter Δ_{SW} needed to increase the predicted warming due to the CO_2 doubling by 2C. To do this, tune Δ_{SW} until the difference in temperature between 560ppm and 280ppm is now 2C higher than the difference in temperature with Δ_{SW} set to zero. Next, estimate the percent change in albedo between 560ppm and 280ppm with nonzero Δ_{SW}. Discuss the sign and is this what you would expect to cause warming?

[ ]:
```
# First, no cloud feedbacks:
# ------------------------------
CO2_fixed=True
CO2_initial=280.0
Delta_SW=0.0
Delta_LW=0.0
sol = solve_ivp(fun=lambda time,T:␣
 ↪rhs(time,T,C_ocn,C_atm,CO2_initial,CO2_fixed,Delta_SW,Delta_LW) \
                ,vectorized=False,y0=T_init,t_span=tspan,t_eval=teval,rtol=1.0e-6)
T_steady=sol.y[:,-1]
print("CO2=",CO2_initial,", Delta_SW=",Delta_SW,", Delta_LW=",Delta_LW)
print("The steady state temperature [T, T_a]=",T_steady)
```

```
CO2_initial=560.0
sol = solve_ivp(fun=lambda time,T:␣
 ↪rhs(time,T,C_ocn,C_atm,CO2_initial,CO2_fixed,Delta_SW,Delta_LW) \
                ,vectorized=False,y0=T_init,t_span=tspan,t_eval=teval,rtol=1.0e-6)
T_steady=sol.y[:,-1]
print("CO2=",CO2_initial,", Delta_SW=",Delta_SW,", Delta_LW=",Delta_LW)
print("The steady state temperature [T, T_a]=",T_steady)

# Next, with SW cloud feedbacks:
# ------------------------------
# repeat the above with SW cloud feedback
# XX
print("CO2=",CO2_initial,", Delta_SW=",Delta_SW,", Delta_LW=",Delta_LW)
print("The steady state temperature [T, T_a]=",T_steady)
```

3c) Estimate the LW cloud feedback parameter Δ_{LW} needed to decrease the predicted warming due to the CO_2 doubling by 2C. To do this, tune Δ_{LW} until the difference in temperature between 560ppm and 280ppm is now 2C lower than the difference in temperature with Δ_{LW} set to zero. Next, estimate the percent change in emissivity between 560ppm and 280ppm with nonzero Δ_{LW}. Discuss the sign and is this what you would expect to cause cooling?

[ ]:
```
# First, no cloud feedbacks:
# --------------------------
CO2_initial=280.0
Delta_SW=0.0
Delta_LW=0.0
sol = solve_ivp(fun=lambda time,T:␣
 ↪rhs(time,T,C_ocn,C_atm,CO2_initial,CO2_fixed,Delta_SW,Delta_LW) \
                ,vectorized=False,y0=T_init,t_span=tspan,t_eval=teval,rtol=1.0e-6)
T_steady=sol.y[:,-1]
print("CO2=",CO2_initial,", Delta_SW=",Delta_SW,", Delta_LW=",Delta_LW)
print("The steady state temperature [T, T_a]=",T_steady)

CO2_initial=560.0
sol = solve_ivp(fun=lambda time,T:␣
 ↪rhs(time,T,C_ocn,C_atm,CO2_initial,CO2_fixed,Delta_SW,Delta_LW) \
                ,vectorized=False,y0=T_init,t_span=tspan,t_eval=teval,rtol=1.0e-6)
T_steady=sol.y[:,-1]
print("CO2=",CO2_initial,", Delta_SW=",Delta_SW,", Delta_LW=",Delta_LW)
print("The steady state temperature [T, T_a]=",T_steady)

# Next, with LW cloud feedbacks:
# ------------------------------
# repeat the above with LW cloud feedback
# XX
print("CO2=",CO2_initial,", Delta_SW=",Delta_SW,", Delta_LW=",Delta_LW)
print("The steady state temperature [T, T_a]=",T_steady)
```

[ ]:
```

```

8 Hurricanes

Global Warming Science

https://courses.seas.harvard.edu/climate/eli/Courses/EPS101/

Hurricanes

(thanks to Yonathan Vardi 2019-07-09)

Please use the template below to answer the workshop questions. "XX" indicates places where you need to complete/write code or add a discussion.

Your name:

[ ]:
```
# load libraries and get data:
import numpy as np
import pickle
import matplotlib.pyplot as plt
from IPython import get_ipython
from scipy.stats import linregress

# load data from pickle file:
# ---------------------------
with open('./hurricanes_variables.pickle', 'rb') as file:
        d = pickle.load(file)
        # print information about each extracted variable:
        for key in list(d.keys()):
            print("extracting pickled variable: name=", key, "; size=", d[key].shape)
            #print("; type=",type(d[key]))
        globals().update(d)
```

explanation of variables

PDI, PDI_years: time series of power dissipation index and the years corresponding to each value

SST,SST_years: time series of sea surface temperature (degree C) and corresponding time axis.

SST_RCP85, SST_RCP85_years: time series of projected temperature for RCP8.5 (Kelvin)

hurricanes_number/years: time series of estimated number of atlantic hurricanes

HURSAT_*: data for hurricane speed, every 6 hours for each hurricane.

1. Observations of the number of Atlantic hurricanes and of the PDI:

(a) Number of hurricanes per year: Plot the time series of the estimated number of Atlantic hurricanes. Add a linear regression line to examine whether there is a trend, calculating the p and r2 values, and discuss your results.

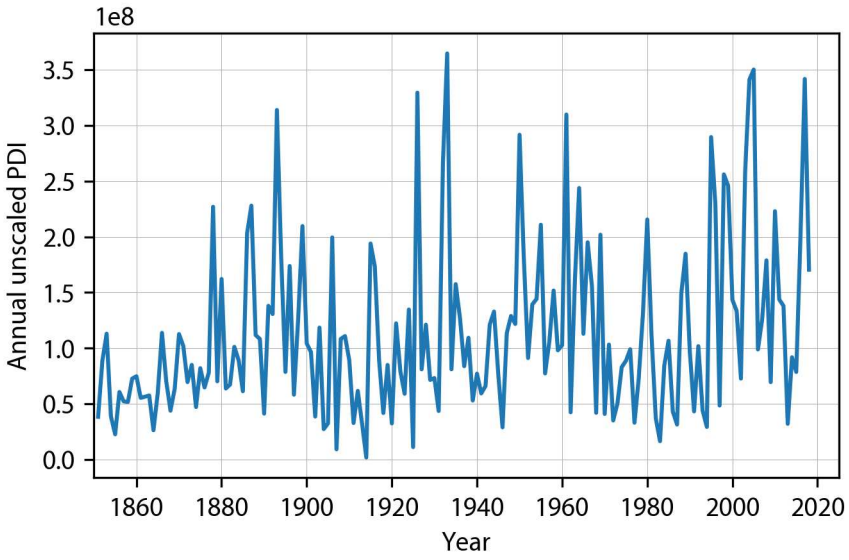
[ ]:
```
plt.figure(dpi=300)
plt.plot(XX,XX)
plt.xlabel('year')
plt.ylabel('# of Atlantic hurricanes')
plt.grid(lw=0.25)

# calculate and plot regression:
x=hurricanes_number_years; y=hurricanes_number
```

```
fit=linregress(x, y); slope=fit.slope; intercept=fit.intercept;
# calculate line as intercept+slope*years:
fitted_line=XX+XX*XX
plt.plot(XX,XX);
```

(b) Plot the time series of the PDI

[12]:
```
# plot the PDI time series:
# -------------------------
fig = plt.subplots(figsize=(5,3),dpi=300)
# labels:
plt.ylabel("XX")
plt.xlabel("XX")
# axis ranges:
plt.plot(XX, XX)
plt.grid(lw=0.25)
```



(c) Smooth the two time series using two passes of a 1-2-1 filter ("running average"),

$$\overline{T}_i = 0.25T_{i-1} + 0.5T_i + 0.25T_{i+1}.$$

Note that every pass requires eliminating the 1st and last data points, where the smoothing cannot be evaluated. Write the expression for the twice-smoothed temperature $\overline{\overline{T}}_i$ in terms of T_i. Over how many years is the average done in this case?

Plot the smoothed time series as thick lines over the un-smoothed ones shown as thin lines.

Plot the PDI and the SST averaged over the Atlantic Main Development Area (MDR), on the same axes (using two separate y axes).

[ ]:
```
def smooth_time_series(data):
    """Apply one pass of 1-2-1 filter to input data.
    Inputs: data - a time series
```

```
    Output: smoothed data, two less in length then input, as first and last cannot be␣
 ↪calculated
    """
    # one pass of smoothing:
    data_smooth = np.zeros(len(data))
    for j in range(1, len(data)-1):
        data_smooth[j]=.25*data[j-1] + 0.5*data[j] + 0.25*data[j+1]
    # return smoohted data without first and last values:
    return data_smooth[1:len(data)-1]


# calculate smoothed time series (2 passes) using above function:
# ---------------------------------------------------------------
# e.g., for the sst this is done using:
# first pass:
SST_smooth=smooth_time_series(SST)
SST_smooth_years=1.0*SST_years[1:len(SST_years)-1]
# second pass:
SST_smooth=smooth_time_series(SST_smooth)
SST_smooth_years=1.0*SST_smooth_years[1:len(SST_smooth_years)-1]
# for the PDI:
PDI_smooth=XX
PDI_smooth_years=XX
# perform a second pass of smoothing of the PDI:
XX
```

[ ]:
```
# plot the two time series on same time axis, using two vertical axes:
# ---------------------------------------------------------------------
fig, axis_PDI = plt.subplots()
# add another y-axis for plotting PDI and SST on the same graph:
axis_SST = axis_PDI.twinx()
axis_PDI.set_ylabel("XX")
axis_SST.set_ylabel("XX")
axis_PDI.set_xlabel("XX")
axis_PDI.set_title("XX")

# plot the PDI on one y axis and the SST on another
axis_SST.plot(SST_years, SST, color="XX", linewidth=0.5,label="unsmoothed SST")
#XX plot smooth sst here
axis_PDI.plot(PDI_years, PDI, color="XX", linewidth=XX,label="XX")
#XX plot smooth PDI here

# show legend and display figure:
fig.legend()
plt.tight_layout()
plt.show()
```

Smoothing details

given $\overline{T}_i = 0.25T_{i-1} + 0.5T_i + 0.25T_{i+1}$, we have

$$\overline{\overline{T}}_i = XX$$

so that two passes are equivalent to a XX-point running average.

(d) Correlation of SST and PDI: Calculate the correlation between the time series of the SST and PDI with and without smoothing for 1950 to 2005. Repeat using the data up to present-day. To calculate the correlation of the two time series, first remove their mean and then calculate the correlation by programming the explicit formula for correlation. Check your results using the NumPy function for calculating correlation. How would you justify calculating the correlation for the twice-smoothed data?

[ ]:
```
# method 1: using a built-in numpy function
# call the function for unsmoothed values
builtin_r2 = np.corrcoef(PDI, SST)[0, 1]**2
print("built-in r² of PDI/SST (unsmoothed):", builtin_r2)
# call the function for smoothed values
builtin_r2 = np.corrcoef(PDI_smooth, SST_smooth)[0, 1]**2
print("built-in r² of PDI/SST (smoothed):  ", builtin_r2)


# calculate "short" time series, with only for the years 1950-2005:
PDI_smooth_short=PDI_smooth[np.logical_and(PDI_smooth_years>XX,PDI_smooth_years<XX)]
PDI_smooth_short_years=XX
# print years selected as a check:
print("PDI_smooth_short_years=",PDI_smooth_short_years)
SST_smooth_short=XX
builtin_r2_short = XX
print("built-in r² (smoothed, 1950-2005 only):  ", builtin_r2_short)


# method 2: an explicit calculation of correlation
def my_calc_correlation(X,Y):
    # remove mean from X:
    X=X-np.mean(X)
    # remove mean from Y:
    Y=XX
    C_XY= XX # calculate using np.sum and np.sqrt
    return C_XY

# call the function for unsmoothed values
my_r2 = my_calc_correlation(PDI, SST)
print("my r² of PDI/SST (unsmoothed):", my_r2**2)
# call my_calc_correlation again for smoothed values
my_r2 = XX
print("my r² of PDI/SST (smoothed):  ", my_r2**2)


# repeat for 1950 to end of data following the same procedure used above
# for the full time series:
XX
```

2. Potential intensity:

(a) Plot the saturation specific humidity as a function of time based on the observed MDR SST and the SST increase projected by 2100 under the RCP8.5 scenario.

[ ]:
```
def q_sat(T,P):
    # saturation specific humidity (g water vapor per g moist air):
    # inputs:
```

```
    # T: temperature, in Kelvin
    # P: pressure, in mb

    R_v = 461 # Gas constant for moist air = 461 J/(kg*K)
    R_d = 287 # Gas constant 287 J K^-1 kg^-1
    TT = T-273.15 # Kelvin to Celsius
    # Saturation water vapor pressure (mb) from Emanuel 4.4.14 p 116-117:
    ew = 6.112*np.exp((17.67 * TT) / (TT + 243.5))
    # saturation mixing ratio (g water vapor per g dry air):
    rw = (R_d / R_v) * ew / (P - ew)
    # saturation specific humidity (g water vapor per g moist air):
    qw = rw / (1 + rw)
    return qw


# calculate for plotting saturation moisture as function of temperature:
T_plot=np.arange(-10.0,35.0,1.0)+273.15
q_plot=T_plot*0.0
P_surface=1000
for i in range(0,len(T_plot)):
    q_plot[i]=q_sat(T_plot[i],P_surface)


# calculate and plot saturation moisture for observed and projected MDR SST:
qsat=SST*0
for i in range(0,len(SST)):
    qsat[i]=q_sat(SST[i]+273.15,P_surface)

qsat_RCP85=SST_RCP85*0
for i in range(0,len(SST_RCP85)):
    qsat_RCP85[i]=XX

plt.figure(1,figsize=(14,5))
plt.subplot(1,3,1)
plt.plot(T_plot,q_plot,color="green")
plt.ylabel("$q_{sat}$")
plt.xlabel("T")

plt.subplot(1,3,2)
plt.plot(SST_years,SST,color="XX",label="historical SST")
XX plot also RCP case
plt.ylabel("XX")
plt.xlabel("XX")
plt.legend()

plt.subplot(1,3,3)
plt.plot(SST_years,qsat,color="green",label="historical")
XX plot also RCP case
plt.ylabel("qsat (kg/kg)")
plt.xlabel("year")
plt.legend()

plt.tight_layout()
```

```
plt.show()
```

(b) Calculate and plot the expected potential intensity as a function of time for the observed (historical) MDR SST and for the projected MDR SST increase by year 2100 under the RCP8.5 scenario (plotting the raw PI, with the smoothed time series superimposed).

[ ]:
```
def calc_potential_intensity_wind_speed_from_SST(sst):
    """Calculate the potential intensity given a sea surface temperature.
    Input: sst- Sea surface temperature, in degrees Celsius
    Output: Return a velocity in m/s
    """
    sst = sst + 273.15 # convert to Kelvin
    epsilon = (1/3)    # efficiency of Hurricane as a carnot engine
    L = 2260*1000      # latent heat of vaporization of water
    PI=sst*0           # initialize array of potential intensity
    for i in range(0,len(sst)):
        q = q_sat(sst[i], 1000) # assumes that pressure is always 1000mb (sea level)
        PI[i] = np.sqrt(epsilon*L*0.15*q)
    return PI


# calculate Potential Intensity (PI) for observed atlantic MDR SST hurricane season␣
 ↪record:
PI_SST_MDR_hurricane_season = calc_potential_intensity_wind_speed_from_SST(SST)
PI_SST_MDR_hurricane_season_years=SST_years
PI_SST_MDR_hurricane_season_smooth=smooth_time_series(PI_SST_MDR_hurricane_season)
PI_SST_MDR_hurricane_season_smooth_years=SST_years[1:len(SST_years)-1]

# calculate PI for RCP8.5 atlantic MDR SST hurricane season record:
PI_rcp_SST_MDR_hurricane_season = XX
PI_rcp_SST_MDR_hurricane_season_years=XX
PI_rcp_SST_MDR_hurricane_season_smooth=XX
PI_rcp_SST_MDR_hurricane_season_smooth_years=XX


# plot smoothed Huricane season PI and SST:
# ------------------------------------------
fig1, ax1 = plt.subplots()
plt.ylabel("Potential Intensity (m/s)")
plt.xlabel("Year")
plt.title("Hurricane season Potential Intensity")
plt.plot(XX,XX
         ,label="PI (from smoothed historical SST)",lw=2)
plt.plot(XX,XX
         ,label="PI (from smoothed RCP8.5 SST)",lw=2)

# plot un-smoothed MDR potential intensity:
# ------------------------------------------
plt.plot(XX,XX, "g-"
         ,label="PI (from historical SST)",lw=0.25)
plt.plot(XX,XX
         ,color="coral", label="PI from RCP8.5 SST)",lw=0.25)
plt.legend()
plt.show()
```

(c) Calculate and plot time series of the cube of the PI to approximate the PDI, and normalize by the mean of the PDI calculated from the historical SST. What is the expected percent increase in PDI by the end of the century? Remember that PDI is a measure of hurricane destructiveness.

[ ]:
```
# plot cubed PI, approximating the PDI:
# ---------------------------------------
# normalize by mean of PDI calculated from historical SST:
PDI=XX
mean_PDI=np.mean(PDI)
PDI_normalized=PDI/mean_PDI
PDI_RCP85=PXX
PDI_RCP85_normalized=PDI_RCP85/mean_PDI
plt.figure(figsize=(10,5))
plt.plot(XX,XX
         ,"g-",label="PDI (from historical SST)")
plt.plot(XX,XX
         ,color="coral", label="PDI (from RCP8.5 SST)")
plt.ylabel("normalized PDI (PI$^3$)")
plt.xlabel("year")
plt.title("cubed PI, as an approximation of PDI, normaized by mean historical value.")
plt.legend()
plt.tight_layout()
plt.show()
```

3. Optional extra credit: Detecting ACC in hurricane intensity: Calculate the fraction of major hurricanes every year and plot it as a function of time. Repeat after averaging the time series into groups of three years. Calculate and plot a linear fit with and without the three-year averages, and calculate the r^2 in each case. Discuss your results.

[ ]:
```
# Loop through every windspeed of every hurricane. If greater than 100
# knots, increase major hurricane and total hurricane count by 1,
# else if greater than 65 knots, increase only total hurricane count by 1
# (Please follow Kossin et al 2020: ignore 1978 and 1980 due to data quality issues;
#   your first 3-year average is therefore over 1979, 1980 (no data) and 1981)

for ind, hurricane_6hourly_winds in enumerate(HURSAT_wind_speed):
      # print ind,  HURSAT_years[ind] and hurricane to understand data structure and␣
 ↪take it from there.
      print(ind,HURSAT_years[ind],hurricane_6hourly_winds)

XX
```

[ ]:
```
# Now use 3-year averages: now average the data from each 3-year groups into
# a single point, e.g., (1982, 1983, 1984), (1985, 1986, 1987), etc.
# calculate the fit and $r^2$ using this group-averaged data.

XX
```

9 Sea-ice

Global Warming Science

https://courses.seas.harvard.edu/climate/eli/Courses/EPS101/

Arctic sea ice

Please use the template below to answer the workshop questions. "XX" indicates places where you need to complete/write code or add a discussion.

your name:

[ ]:
```
# import needed libraries:
# ------------------------
import numpy as np
import scipy as sp
import scipy.ndimage
import matplotlib.pyplot as plt
import matplotlib
from matplotlib.colors import BoundaryNorm
import pickle
import cartopy.crs as ccrs
import cartopy.feature as cfeature
#from cartopy.feature import NaturalEarthFeature
from cartopy import config
from matplotlib.colors import BoundaryNorm
from matplotlib import animation, rc
from IPython.display import HTML

# load data from pickle file:
# ---------------------------
with open('./sea_ice_variables.pickle', 'rb') as file:
    d = pickle.load(file)
    # print information about each extracted variable:
    for key in list(d.keys()):
        print("extracting pickled variable: name=", key, "; size=", d[key].shape)
        #print("; type=",type(d[key]))
    globals().update(d)

# set defaulty font size:
font = {'size' : 11}
matplotlib.rc('font', **font)
```

Explanation of sea ice variables:

Data are for the Arctic unless Antarctica is denoted in variable name

sea ice age as a function of lat/lon and corresponding axes:

sic_age_obs1984, sic_age_obs2018, sic_age_obs_lat, sic_age_obs_lon

**sea ice area time series: (_Antarctica/_Arctic) **

sic_area_obs_timeseries, sic_area_obs_timeseries_time

sea ice concentration in september as a function of year, lat,lon:

sic_concentration_obs_sept, sic_concentration_obs_lat, sic_concentration_obs_lon, sic_concentration_obs_years

Projected sea ice concentration in september as a function of year, lat,lon:

sic_concentration_rcp85_sept, sic_concentration_rcp85_sept_lat, sic_concentration_rcp85_sept_lon, sic_concentration_

sea ice extent time series:

sic_extent_obs_timeseries, sic_extent_obs_timeseries_time

observed sea ice thickness as a function of lon/lat/month during two years:

sic_thickness_obs1979, sic_thickness_obs2019, sic_thickness_obs_lat, sic_thickness_obs_lon

and cell area used for calculations:

sic_thickness_obs_cell_area

Antarctica time series:

Antarctica_sic_area_obs_timeseries, Antarctica_sic_extent_obs_timeseries, Antarctica_sic_obs_times

A monthly time series of Arctic sea ice area in km^2 from a long model control integration (that is, with CO_2 fixed to pre-industrial values):

sic_CESM_control, sic_CESM_control_years

Comments:

1) concentration maps are in percent;
2) extent and area observation timeseries are in km^2
3) thickness is in meters

1) observations over the past decades

1a) Time series: Plot the monthly sea ice area and extent (defined as the area with at least 15% sea ice cover) for the Arctic and for Antarctica, superimpose a time series of the values during the month of minimum sea ice for the Arctic and maximum sea ice for Antarctica:

[ ]:
```
fig=plt.figure(figsize=(7,5),dpi=200)
start=8;

plt.subplot(2,1,1)
#plt.plot(XX,XX,lw=0.5,label="monthly")
#plt.plot(sic_area_obs_timeseries_time[start:-1:12],XX,lw=2,label="september")
plt.xlabel("year")
plt.ylabel("sea ice area ($10^6$ km$^2$)")
plt.title("sea ice area")
plt.legend(loc="upper left");

plt.subplot(2,1,2)
#plt.plot(XX,XX,lw=0.5,label="monthly")
#plt.plot(XX,XX,lw=2,label="september")
plt.xlabel("year")
plt.ylabel("sea ice extent ($10^6$ km$^2$)")
plt.title("sea ice extent")
plt.legend(loc="upper left");

XX also for Antarctica!
XX find and superimpose the *maximum* month here

plt.tight_layout()
```

```
plt.show();
```

1b) Plot only the September Arctic sea ice, superimpose a linear line fit. Plot only the September Antarctic sea ice area, superimpose horizontal lines at the location of the mean and ± 1 standard deviation calculated for the record until 2022.

[ ]:
```
plt.figure(figsize=(6,8))

# minimum sea ice in NH:
line_fit_pars = np.polyfit(sic_extent_obs_timeseries_time_Arctic[8::
 ↪12],sic_area_obs_timeseries_Arctic[8::12], 1)
linefit=line_fit_pars[0]*XX+XX
plt.subplot(2,1,1)
plt.plot(XX,XX,'-or',label="Observed")
plt.plot(XX,XX,'b',label="Linear fit")
plt.title("Arctic September sea ice area")
plt.xlabel("Year")
plt.ylabel("$10^6$ km$^2$")
plt.grid(lw=0.25)
plt.legend()

# maximum sea ice in SH:
plt.subplot(2,1,1)
# calculate the mean of the record, not including the last year (2023):
linemean=XX
twostd=2*np.std(XX)
plt.subplot(2,1,2)
plt.plot(XX,XX,'-or',label="Observed")
plt.plot(XX,XX,'b',label="Mean")
plt.plot(XX,XX,'b',linewidth=1,linestyle='--',label="$\pm$2std") # +2std
plt.plot(XX,XX,'b',linewidth=1,linestyle='--') # -2std
plt.title("Antarctic September sea ice area")
plt.xlabel("Year")
plt.ylabel("$10^6$ km$^2$")
plt.grid(lw=0.25)
plt.legend()

plt.tight_layout()
```

1c) Concentration: Compare the sea ice September minima of 1979 vs 2017 by plotting the two maps.

[ ]:
```
# preliminaries, defining configuration of subplots:
projection=ccrs.NorthPolarStereo(central_longitude=0.0)
fig,axes=plt.subplots(1,2,figsize=(7,4),subplot_kw={'projection': projection},dpi=200)

levels=np.arange(0,101,5)
cmap = plt.get_cmap('jet')
norm = BoundaryNorm(levels, ncolors=cmap.N, clip=True)

# concentration first year:
# -------------------------
axes[0].set_extent([0, 359.999, 70, 90], crs=ccrs.PlateCarree())
axes[0].coastlines(resolution='110m')
```

```
axes[0].gridlines()
c=axes[0].pcolormesh(sic_concentration_obs_lon,sic_concentration_obs_lat\
                     , sic_concentration_obs_sept[0,:,:]\
                     ,cmap=cmap,norm=norm,transform=ccrs.PlateCarree())
clb=plt.colorbar(c, shrink=0.8, pad=0.02,ax=axes[0])
clb.set_label('concentration')
clb.set_ticks(np.arange(0,105,10))
axes[0].set_title('concentration obs, year='+repr(sic_concentration_obs_years[0]));

# concentration obs last year:
# ----------------------------
axes[1].set_extent([0, 359.999, 70, 90], crs=ccrs.PlateCarree())
axes[1].coastlines(resolution='110m')
axes[1].gridlines()
# XX
clb.set_label('concentration')
clb.set_ticks(np.arange(0,105,10))
axes[1].set_title('concentration obs, year='+repr(sic_concentration_obs_years[-1]));

# finalize and show plot:
#plt.subplots_adjust(top=0.92, bottom=0.08, left=0.01, right=0.95, hspace=0.
 ↪15,wspace=-0.03)
plt.show()
```

1d) Thickness: Plot March sea ice thickness maps for 1979 and 2019. Calculate and plot the PDF (probability distribution function) of ice thickness for these two years for the entire year.

[ ]:
```
# preliminaries, defining configuration of subplots:
projection=ccrs.NorthPolarStereo(central_longitude=0.0)
fig,axes=plt.subplots(1,2,figsize=(7,4),subplot_kw={'projection': projection},dpi=200)

levels=np.arange(0,5.3,0.2)
cmap = plt.get_cmap('jet')
norm = BoundaryNorm(levels, ncolors=cmap.N, clip=True)

# thickness first year:
# -------------------------
axes[0].set_extent([0, 359.999, 70, 90], crs=ccrs.PlateCarree())
#XX
clb.set_label('thickness (m)')
axes[0].set_title('thickness 1979');

# thickness obs last year:
# ----------------------------
axes[1].set_extent([0, 359.999, 70, 90], crs=ccrs.PlateCarree())
#XX
clb.set_label('thickness (m)')
axes[1].set_title('thickness, 2019');

# finalize and show plot:
#plt.subplots_adjust(top=0.92, bottom=0.08, left=0.01, right=0.95, hspace=0.
 ↪15,wspace=-0.03)
plt.show()
```

[ ]:
```
# ice thickness histogram (probabiligy distribution function):

def plot_thickness_histogram(thickness_array,title):
    # consider only latitudes north of 70N (use variable sic_thickness_obs_lat
    # loaded above from pickle file), and ignore thicknesses less than 0.1.
    # hint: use something like: a[b>3]=np.nan and a[a>3]=np.nan
    thickness=1.0*thickness_array
    thickness[XX]=np.nan
    thickness[XX]=np.nan
    # reshap thickness into a 1d vector:
    thickness.shape=XX # XX this should be set to the product of the dimensions of␣
 ↪thickness
    # calculate and plot the histogram:
    hist, bin_edges = np.histogram(thickness[~np.isnan(thickness)], bins=50,␣
 ↪density=True)
    x=(bin_edges[1:]+bin_edges[0:-1])/2
    # bar plot:
    width=0.9*(bin_edges[2]-bin_edges[1])
    plt.bar(x,hist,width=width);
    plt.xlabel("thickness (m)")
    plt.ylabel("probability")
    plt.title(title)

plt.figure(figsize=(8,4),dpi=200)
plt.subplot(1,2,1)
plot_thickness_histogram(XX,"thickness histogram 1979")
plt.subplot(1,2,2)
plot_thickness_histogram(XX,"thickness histogram 2019")
plt.tight_layout()
```

1e) Age: Plot sea ice age maps for 1984 and 2018 for March. Calculate and plot the PDF (probability distribution function) of ice age for these two years.

[ ]:
```
# preliminaries, defining configuration of subplots:
projection=ccrs.NorthPolarStereo(central_longitude=0.0)
fig,axes=plt.subplots(1,2,figsize=(7,4),subplot_kw={'projection': projection},dpi=200)

levels=np.arange(0,7.1,0.2)
cmap = plt.get_cmap('jet')
norm = BoundaryNorm(levels, ncolors=cmap.N, clip=True)

# thickness first year:
# -------------------------
axes[0].set_extent([0, 359.999, 70, 90], crs=ccrs.PlateCarree())
#XX clb.set_label('age (years)')
axes[0].set_title('age, 1984');

# thickness obs last year:
# ---------------------------
axes[1].set_extent([0, 359.999, 70, 90], crs=ccrs.PlateCarree())
#XX
clb.set_label('age (years)')
axes[1].set_title('age 2018');
```

```
plt.show();
```

[ ]:
```
# ice age histogram (probabiligy distribution function):

def plot_age_histogram(age_array,title):
    # consider only latitude north of 70N (use variable sic_age_obs_lat
    # loaded from pickle file), and ages larger than 0.1, smaller than 15
    age=1.0*age_array
    age[XX]=np.nan
    age[XX]=np.nan
    # reshap age into a vector:
    age.shape=XX # XX this should be set to the product of the dimensions of age
    # calculate and plot the histogram:
    hist, bin_edges = np.histogram(age[~np.isnan(age)], bins=50, density=True)
    x=(bin_edges[1:]+bin_edges[0:-1])/2
    # bar plot:
    plt.bar(x,hist);
    plt.xlim(0,10)
    plt.xlabel("age (years)")
    plt.ylabel("probability")
    plt.title(title)

plt.figure(figsize=(8,4),dpi=300)
plt.subplot(1,2,1)
plot_age_histogram(XX,"age histogram 1984")
plt.subplot(1,2,2)
plot_age_histogram(XX,"age histogram 2018")
plt.tight_layout()
```

2) How much heat is needed to melt sea ice:

Calculate how much heat per unit area (Watts/m2) is needed during the three summer months every year to completely melt a sea ice cover within 30 yr. Assume that the initial ice thickness is 3 m, that it has an initial average temperature of -3°C, and that the ocean mixed layer (the upper 50 m) is initially at the freezing temperature (-1.8°C). At the end of the period, the ocean mixed layer is at a final temperature of $T_{final} = 1.2^{\circ}$C and the ice is completely melted. Use the following guidelines:

(a) Calculate separately the heat flux required for (i) warming the ice to its melting temperature of 0°C, (ii) melting the ice, (iii) warming the melt water to T_{final}, and (iv) warming the mixed layer to T_{final}.

(b) Find out the Arctic summer downward SW, and calculate what percent change in cloud or ice albedo during the peak summer months could account for a change in absorbed solar flux that can lead to the above complete ice melting in 30 yr. Discuss the implications for identifying the cause of Arctic sea ice melting.

[ ]:
```
# some needed constants:
year=365*24*3600
L_melt=XX # latent heat of melting of ice, J/kg
Cp_ice=XX # specific heat of ice J/(kg*K)
Cp_water=XX # specific heat of water J/(kg*K)
thickness_ice=XX
thickness_water=XX
rho_ice=900
```

```
rho_water=1025

# parameter of the problem:
time=30*year*(3/12) # three months a year, for 30 years
deltaT_ocean=XX
deltaT_ice=XX # warm the ice from its initial temperature to the freezing/melting␣
 ↪temperature of -1.8C

# 2a) calculation of the results:
flux_for_warming_ice_to_melting=XX
flux_for_melting_ice=XX
flux_for_warming_water=XX

print(" flux_for_warming_ice_to_melting=%3.3g W/m^2" %␣
 ↪(flux_for_warming_ice_to_melting))
print(" flux_for_melting_ice=%3.3g W/m^2" % (flux_for_melting_ice))
print(" flux_for_warming_water=%3.3g W/m^2" % (flux_for_warming_water))

# 2b) what ice albedo change would cause this:
SW_radiation_summer_arctic=400 # W/m^2, see http://www.climate.be/textbook/
 ↪chapter2_node5_3.xml figure 2.11
# so less than %1 albedo change can melt the ice
print(" percent change for melting sea ice = %3.3g" \
      % XX);
```

3) Thickness-insulation feedback:

solve the equation

$$\rho_{ice} L_f \frac{dh}{dt} = \kappa \frac{T_f - T_{\text{surface}}}{h}$$

analytically for the ice thickness $h(t)$, using $T_f = -1.8$ °C and $T_{\text{surface}} = -15$ °C, for the initial conditions $h(t_0) = h_0 = 0.1$ m. Plot the solution for one year, use it to find the time to grow the first meter of thickness and then the second, and explain the results in terms of the negative thickness-insulation feedback.

Analytical solution:

XX

[ ]:
```
# plotting:
XX
```

4) Increasing roughness with age, melt ponds and albedo:

(i) Explain why ice surface roughness is expected to increase with ice age.

(ii) Suppose the surface structure as function of location x, y is given by $h(x, y) = h_0 \cos(kx)$ with $k = 2\pi/100$ m^{-1}. Plot the area fraction as function of the melt water volume for $h_0 = 0.5$ m and for $h_0 = 1$ m.

(iii) based on your results, explain the positive feedback leading to more melting due to the decrease in ice age and therefore decrease in ice roughness.

Solution: Given that the sea ice surface is given by $h(x, y) = h_0 \cos(kx)$ with $k = 2\pi/100$ m^{-1}, the lowest point is at $kx = \pi$. Consider a single wavelength and suppose the melt water fill the low point from

$kx = (\pi - \theta)$ to $kx = (\pi + \theta)$. We can calculate the water area and volume as function of θ. Then, we can plot the area fraction as function of volume, as requested.

$$\text{melt water area fraction}(\theta) = 2\theta/(2\pi)$$

and

$$\text{melt water volume}(\theta) = -\int_{(\pi-\theta)/k}^{(\pi+\theta)/k} h_0 \cos(kx')dx' = XX.$$

[ ]:
```
# calculate and plot the above solution:
k=2*np.pi/100
h1=0.5 # m, amplitude of ice surface height variations
h2=1 # m, amplitude of ice surface height variations
theta=np.arange(0,np.pi/2,0.01)
area_fraction=theta/np.pi
volume1=XX# vector of melt water volume calculated for h1
volume2=XX # vector of melt water volume calculated for h2

# first, plot everything as funtion of theta:
# -------------------------------------------
fig = plt.figure(dpi=200)
ax1 = fig.add_subplot(111)
ax1.plot(XX,XX,color="r",label="area fraction")
ax1.set_xlabel("$\\theta$ (degrees)")
ax1.set_ylabel("melt pond area fraction",color="r")

ax2 = ax1.twinx()  # instantiate a second axes that shares the same x-axis
ax2.plot(theta*180/np.pi,volume1,color="b",label="volume h=0.5m");
ax2.plot(theta*180/np.pi,volume2,linestyle="--",color="b",label="volume h=2m");
ax2.set_ylabel("volume",color="b")

fig.legend(loc="upper left");
ax1.grid('both')
plt.tight_layout()


# now, plot area fraction covered by melt water as function of melt water volume:
# -------------------------------------------------------------------------------
plt.figure(dpi=150)
plt.plot(XX,XX,label="volume h=0.5m")
plt.plot(XX,XX,label="volume h=2m")
plt.xlabel("melt water volume")
plt.ylabel("melt pond area fraction")
plt.title("melt pond area fraction (1-albedo) as func of melt water volume")
plt.legend();
```

5) Projections: Contour the September sea ice concentration at the beginning of the 21st century and at year 2100 according to the RCP8.5 scenario.

[ ]:
```
# preliminaries, defining configuration of subplots:
projection=ccrs.NorthPolarStereo(central_longitude=0.0)
fig,axes=plt.subplots(1,2,figsize=(12,6),subplot_kw={'projection': projection},dpi=200)
```

```
levels=np.arange(0,101,1)
cmap = plt.get_cmap('jet')
norm = BoundaryNorm(levels, ncolors=cmap.N, clip=True)

# thickness first year:
# -------------------------
axes[0].set_extent([0, 359.999, 70, 90], crs=ccrs.PlateCarree())
#XX
clb.set_label('concentration')
clb.set_ticks(np.arange(0,105,10))
axes[0].set_title('sic_concentration_rcp85_sept␣
 ↪'+repr(sic_concentration_rcp85_sept_years[0]));

# thickness obs last year:
# ----------------------------
axes[1].set_extent([0, 359.999, 70, 90], crs=ccrs.PlateCarree())
#XX
clb.set_label('concentration')
clb.set_ticks(np.arange(0,105,10))

axes[1].set_title('sic_concentration_rcp85_sept␣
 ↪'+repr(sic_concentration_rcp85_sept_years[-1]));

# finalize and show plot:
#plt.subplots_adjust(top=0.92, bottom=0.08, left=0.01, right=0.95, hspace=0.
 ↪15,wspace=-0.03)
plt.show()
```

6) Climate change detection:

(i) Examine if the trend in annual minimum sea ice area over the satellite period since 1979 is consistent with natural variability, as estimated using a long run of a climate model with no anthropogenic greenhouse forcing. Calculate the PDF for a period corresponding to the length of the observed record, and calculate the probability that the observed trend is part of natural variability. Repeat the calculation using only the record from 1979 to 1999; what differences do you see? (ii) Calculate the trends as a function of both the trend duration (2–100 yr) and amplitude. Contour the PDF, and superimpose a symbol showing the observed trend. Discuss.

[ ]:
```
# Calculate a probability distribution function (pdf) of September sea ice
# area trends from long CESM model run, as function of trend amplitude and trend␣
 ↪duration.

# get september values only:
N_months=len(sic_CESM_control)
start_month=8
sic_CESM_control_sept=sic_CESM_control[start_month:N_months:12]
sic_CESM_control_sept_years=sic_CESM_control_years[start_month:N_months:12]
Nyears=len(sic_CESM_control_sept)


# plot model time series, monthly and september values, to make sure all is well:
fig=plt.figure(figsize=(7,2.7),dpi=300)
plt.plot(XX,XX,label="monthly")
```

```
plt.plot(XX,XX,'r',label="September")
plt.xlim(0,20)
plt.title("sea ice area, a control CESM run")
plt.xlabel("time (years)")
plt.ylabel("area ($10^6$ km$^2$)")
plt.legend()
plt.show()


# caclculate the observed trend over the past 40 years,
# normalized to millions of km^2 per 10 years:
# ----------------------------------------------------
start=8
sic_area_obs_timeseries_sept_time_Arctic=sic_area_obs_timeseries_time_Arctic[start:-1:
 ↪12]
sic_area_obs_timeseries_sept_Arctic=sic_area_obs_timeseries_Arctic[start:-1:12]
interval_obs=(sic_area_obs_timeseries_sept_time_Arctic[-1] \
              -sic_area_obs_timeseries_sept_time_Arctic[0])
Nyears_obs=int(interval_obs)
x=1.0*sic_area_obs_timeseries_sept_time_Arctic
y=1.0*sic_area_obs_timeseries_sept_Arctic
fit_obs = np.polyfit(x, y, 1)
fit_obs_line=(x*fit_obs[0]+fit_obs[1])
trend_obs=10*fit_obs[0]
print("observed decadal trend = ",trend_obs,"over a period of" \
      ,interval_obs," years.")


# plot fit to observed sept sea ice:
fig=plt.figure(figsize=(4,3),dpi=300)
plt.plot(x,y,label="Observed September sea ice")
plt.plot(x,fit_obs_line,label="Linear fit")
plt.xlabel("Year")
plt.ylabel("Sea ice area ($10^6$ km$^2$)")
plt.legend()
plt.grid(lw=0.25)
plt.tight_layout()
plt.show()


# calculate the pdf:
# ------------------
# define bins for trend histogram:
bin_edges=np.arange(-1,1.06,0.05)*1
trend_intervals=np.arange(2,101,1)
# initialize trends probability distribution function as function of interval and␣
 ↪trend:
trends_to_contour=np.zeros((len(trend_intervals),len(bin_edges)-1))
# loop over time intervals from 1 to 100 years
i_interval=-1
for interval in trend_intervals:
    i_interval=i_interval+1
    # loop over all start years for intervals, calculate and accomulate trends:
```

```
    trends=[]
    for start_year in range(0,Nyears-interval-1):
        # for each start year, calculate the trend at the
        # interval being considered using a line fit, normalized to
        # trend of 10^6 km^2 per 10 years:
        x1=np.arange(start_year,(start_year+interval),1.0)
        y1=XX
        line_fit_pars = np.polyfit(x1, y1, XX) # fit a line, specify polynomial degree
        trends.append(line_fit_pars[0]*XX) # convert to units of million km per decade
    # bin trends
    #print(interval,"trends=",trends)
    hist,bin_edges_output=np.histogram(trends,bin_edges)
    # save bins to be plotted, normalized to fraction of total number
    # of bins for the given interval
    max_bins=Nyears-interval-1
    trends_to_contour[i_interval,:]=hist/max_bins

# contour the calculated pdf for trends:
trend_amplitude=bin_edges # units of million km^2/10 years
fig=plt.figure(figsize=(5,4),dpi=300)
levels=np.log10(np.array([0.001,0.00316,0.01,0.0316,0.1,0.316,1,3.16,10]))
plt.contourf(trend_intervals,bin_edges[0:-1],np.log10(trends_to_contour.T),levels)
plt.ylim(-1,0)
plt.title("log10(pdf) of trends")
plt.xlabel("trend interval (years)")
plt.ylabel("trend ($10^6$ km$^2/10$ years)");
plt.colorbar();


# now plot a large x at the location of the observed trend over the
# satellite period, normalized to millions of km^2 per 10 years:
interval_obs=(sic_area_obs_timeseries_time_Arctic[-1]-sic_area_obs_timeseries_time_Arctic[0])
trend_obs=XX
plt.plot(interval_obs,trend_obs,'x',markersize=16,color='r',label="observed trend");
```

7) Animation!

optional: Create an animation of September sea ice concentration maps for all years of the RCP8.5 projection.

[ ]:
```
projection=ccrs.NorthPolarStereo(central_longitude=0.0)
fig,axes=plt.subplots(1,1,figsize=(12,6),subplot_kw={'projection': projection} \
                      ,dpi=150)

# draw the first frame:
levels=np.arange(0,101,1)
cmap = plt.get_cmap('jet')
norm = BoundaryNorm(levels, ncolors=cmap.N, clip=True)
axes.set_extent([0, 359.999, 70, 90], crs=ccrs.PlateCarree())
axes.coastlines(resolution='110m')
axes.gridlines()
# plot the sea ice at the first available time:
c=axes.pcolormesh(XX)
axes.set_title('rcp85 sept sea ice ' \
```

```
                    +repr(sic_concentration_rcp85_sept_years[i]));
XX add a colorbar
plt.close()
clb.set_label('concentration')
clb.set_ticks(np.arange(0,105,10))
plt.show()

def animate(i):
    # update the plot with the sea ice at year i:
    c=axes.pcolormesh(XX)
    axes.set_title('rcp85 sept sea ice at year ' \
                    +repr(XX));
    return (c,)


# animate (takes a very long time to complete...):
plt.rcParams['animation.embed_limit'] = 40
anim = animation.FuncAnimation(fig, animate,
                               frames=94, interval=20,
                               blit=True)

# save for inline display:
#HTML(anim.to_html5_video())
HTML(anim.to_jshtml())
```

[ ]:

10 Ice-sheets

Global Warming Science

https://courses.seas.harvard.edu/climate/eli/Courses/EPS101/

Ice sheets!

Please use the template below to answer the workshop questions. "XX" indicates places where you need to complete/write code or add a discussion.

your name:

[ ]:
```
# import needed libraries and load data:
import numpy as np
import pickle
import matplotlib.pyplot as plt
from matplotlib import ticker, cm, colors
import cartopy.crs as ccrs
import cartopy.feature as cfeature
from cartopy.feature import NaturalEarthFeature
from cartopy import config
from mpl_toolkits.axes_grid1 import make_axes_locatable

# load variables from a pickle file:
with open('./ice_sheets_variables.pickle', 'rb') as file:
        d = pickle.load(file)
        # print information about each extracted variable:
        for key in list(d.keys()):
            print("extracting pickled variable: name=", key, "; size=", d[key].shape)
            #print("; type=",type(d[key]))
        globals().update(d)
```

Explanation of input variables:

AIS=Antarctic ice sheet; GIS=Greenland ice sheet

observed SMB from GRACE gravity sattelite: AIS_GRACE_SMB, AIS_GRACE_years

GIS_GRACE_SMB, GIS_GRACE_years

Antarctica SMB time series and map:

AIS_evap_timeseries_annual_avg, AIS_evap_timeseries_annual_std, AIS_net_SMB_timeseries_annual_avg, AIS_net_SMB_timeseries_annual_std

AIS_net_SMB_map, AIS_net_SMB_map_lat, AIS_net_SMB_map_lon

Greenland SMB time series and map:

GIS_evap_timeseries_annual_avg, GIS_evap_timeseries_annual_std, GIS_net_SMB_timeseries_annual_avg, GIS_net_SMB_timeseries_annual_std

GIS_net_SMB_map, GIS_net_SMB_map_lat, GIS_net_SMB_map_lon

Surface air temperature data as a function of day of year, lat, lon and the corresponding axes:

SAT_control, SAT_rcp85

SAT_lat, SAT_lon, SAT_time

SMB time axis: SMB_timeseries_annual_years

land masks:

landice_antarctica_mask_control, landice_antarctica_mask_rcp85, landice_greenland_mask_control, landice_greenland_

lon/lat axes: one over the relevant land ice sheet, zero elsewhere

landice_lat, landice_lon

1) Ice stream acceleration:

By what factor should ice streams in Greenland accelerate now in order to contribute a 1.5 m sea level rise by 2100.

1.5 m of sea level rise corresponds to $\rho_{water} 70\% \cdot (4\pi R_e^2) \cdot 1.5$ of mass loss (70% comes from the percentage area on Earth covered by ocean, and R_e is the radius of the Earth). $$$$

[ ]:
```
# first, calculate how many Gt ice correspond to the specified sea level rise
XX=np.nan
rho_ice=900
rho_water=1000
Re=6371000 # earth radius
dh=1.5 # change in sea level
ice_mass_loss_equiv_to_dh_Gt=XX/1.0e12
print("mass loss in Gt corresponding to %3.3g meter of sea level rise: %3.3g␣
 ↪Gt"%(dh,ice_mass_loss_equiv_to_dh_Gt))

# Currently, the Greenland ice sheet accumulates at a rate of 520   per year,
# and ablates (including runoff, basal melt and iceburg production) at a rate of 720  ␣
 ↪per year,
# yielding roughly 200   per year of mass loss:

current_accumulation_rate=520
current_yearly_ablation_Gt=720
# the needed ablation rate per year over the years until 2100 is therefore:
needed_yearly_ablation_rate=XX

acceration_factor=needed_yearly_ablation_rate/current_yearly_ablation_Gt
print("needed acceleration factor is %3.3g" %(acceration_factor))
```

2) Positive Degree Days:

2a) Calculate and plot PDD maps for Greenland and Antarctica for present-day, and for year 2100 based on temperature projection from the RCP 8.5 scenario.

[ ]:
```
# calculate PDD maps for control and rcp85:
pdd_map_control=np.sum((SAT_control-273.15)*(SAT_control>273.15),axis=0)
pdd_map_rcp85=XX

# apply masks for greenland and antarctica before plotting, so that ocean areas are␣
 ↪not contoured too:
pdd_map_greenland_control=pdd_map_control*(1.0*mask_landice_greenland_control)
pdd_map_antarctica_control=XX
pdd_map_greenland_rcp85=XX
pdd_map_antarctica_rcp85=XX
```

```
# plot Greenland present-day (control):
# -------------------------------------
lons, lats = np.meshgrid(lon, lat)
fig = plt.figure(dpi=150)#figsize=(6, 4))
ax = plt.axes(projection=ccrs.PlateCarree(central_longitude=0.0, globe=None))
ax.set_extent([-70, -10, 50, 85], crs=ccrs.PlateCarree())
ax.coastlines(resolution='110m')
ax.gridlines()

cls=10**np.arange(-3,3,0.4)
c=plt.contourf(lons, lats, np.maximum(pdd_map_greenland_control,0.0001),levels=cls\
               ,transform=ccrs.PlateCarree()
               ,norm=colors.LogNorm(min(cls),max(cls)), extend='both', cmap='RdBu_r')
plt.xticks(np.arange(-60,0,20))
plt.yticks(np.arange(50,90,10))
plt.title('current PDD  (°C day)');
plt.colorbar(c, shrink=0.7, pad=0.02)
plt.pause(0.05)



# plot Greenland RCP8.5:
# ----------------------
fig = plt.figure(dpi=150)#figsize=(6, 4))
#XX
```

[ ]:
```
# plot Antarctica present day (control):
# --------------------------------------
fig = plt.figure(figsize=(3,3),dpi=200)
ax = plt.axes(projection=ccrs.SouthPolarStereo(central_longitude=0.0, globe=None))
ax.set_extent([0, 359.99999, -90, -60], crs=ccrs.PlateCarree())
ax.coastlines(resolution='110m')
ax.gridlines()
cls=10**np.arange(-3,3,0.4)
c=plt.contourf(lons, lats, np.maximum(pdd_map_antarctica_control,0.0001),levels=cls\
                         ,transform=ccrs.PlateCarree()\
                         ,norm=colors.LogNorm(min(cls),max(cls))\
                         ,extend='both', cmap='RdBu_r')
plt.title('current PDD  (°C day)');
cbar = plt.colorbar(c, shrink=1.0, pad=0.02);


# plot Antarctica RCP8.5:
# -----------------------
fig = plt.figure(figsize=(3,3),dpi=200)
#XX
```

2b) Evaluate the expected acceleration of melt rate (assuming it is proportional to the PDD change), and the consequences to sea level rise.

[ ]:
```
# estimate total PDD over each ice sheet:

pdd_total_greenland_control=np.sum(pdd_map_greenland_control,axis=(0,1))
pdd_total_greenland_rcp85=XX
pdd_total_antarctica_control=XX
pdd_total_antarctica_rcp85=XX

print("current summed pdd on greenland ice sheet: %3.3g"%(pdd_total_greenland_control))
print("future summed pdd on greenland ice sheet: %3.3g"%(pdd_total_greenland_rcp85))
print("current summed pdd on antarctica ice sheet: %3.
 ↪3g"%(pdd_total_antarctica_control))
print("future summed pdd on antarctica ice sheet: %3.3g"%(pdd_total_antarctica_rcp85))
print("PDD on greenland ice sheet is predicted to increase by a factor of %3.3g"\
      %(XX))
print("PDD on antarctica ice sheet is predictec to increase by a factor of %3.3g"\
      %(XX))
```

2c) Compare your results for Greenland and Antarctica to the above calculated con- sequences of acceleration of ice streams and discuss.

XX

3) Calving

3a) Cliff instability: What is the highest cliff height H in Fig. 1b that allows the marked 45° section to be stable.

The projected gravity acceleration rate along the slope is $g\sin(45^\circ) = g/\sqrt{2}$, and thus the down-slope force is XX. On the other hand, the maximum tangential stress can be supplied by the underneath ice shell is XX. Thus the secion will break if

$$XX > XX$$

or when

$$L > XX$$

[ ]:
```
# calculate here:
tau_c=100000
rho_ice=900
rho_water=1000
g=9.8
print("max cliff height=",XX)
```

3b) Buoyancy forcing: What is the maximum protruding section length L in Fig. 1c that can be stable to buoyancy forces.

The density difference between water and ice gives rise to a net buoyancy force on a submerged iceberge. Following the notation in Fig. 1c, the net buoyancy force is XX. The maximum force that the ice can sustain is again XX. Thus the section will break if

$$XX > XX$$

or, when

$$H > XX$$

[ ]:
```
print("submerged shelf height=",XX)
```

3c) Hydro-fracturing and buoyancy forcing:

Consider grounded ice of height $H = 2$ km, bordering with sea water such that the top of the ice is above sea level. Calculate the minimum depth of water D that will lead to the flotation and therefore breakup of the ice, taking into account the adhesion to the rest of the ice via the yield stress τ_c. How would your answer change if there was a hydro-fracture extending to half the thickness of the ice? Let $L = 150$ m be the distance from the crack to the ocean.

[ ]:
```

```

4) Basal hydrology

4a) Basal temperature and melting:

Calculate the basal temperature, and where rele- vant also the melt rate, given a surface ice temperature of Ts $= -40$C, geothermal heat flux of G $= 0.07$ W/m2 and an ice thicknesses of H $= 1,2,3$ km.

[ ]:
```
kappa_ice=2.2 # W/m/K
G=0.07
Ts=-40
T_1km=XX
T_2km=XX
T_3km=XX
print("Basal temperature is %3.2f C at the base of a 1-km thick ice shell."%(T_1km))
print("Basal temperature is %3.2f C at the base of a 2-km thick ice shell."%(T_2km))
print("Basal temperature is %3.2f C at the base of a 3-km thick ice shell."%(T_3km))
```

4a solution continued: calculate the basal melt rate

The basal temperature of an ice shell of XX km thick is above freezing point and thus melting will occur. The melting rate is determined by the amount of geothermal heat flux that cannot be conducted upward while holding the basal temperature at the freezing point 273.15~K.

$$M/L = XX$$

and you need to convert this to m/year

[ ]:
```
# calc melt rate:

L=336000 # J/kg
sectoyear=86400*365
rho_ice=900
M_XXkm=XX # m/year
print("Basal melting rate is %3.2g m/year beneath a XX-km thick ice shell."%(M_XXkm))
```

[ ]:
```

```

11 Mountain glaciers

Global Warming Science

https://courses.seas.harvard.edu/climate/eli/Courses/EPS101/

Mountain glaciers:

Please use the template below to answer the workshop questions. "XX" indicates places where you need to complete/write code or add a discussion.

Your name:

[ ]:
```
# load libraries, read time series data:
import numpy as np
import matplotlib.pyplot as plt
from numpy import linalg
from scipy import stats
import cartopy.crs as ccrs
from scipy.stats import linregress
from scipy.signal import savgol_filter # for smoothing time series
from scipy import optimize
import pickle
import copy

# load the data from a pickle file:
with open('./mountain_glaciers_variables.pickle', 'rb') as file:
    while 1:
        try:
            d = pickle.load(file)
        except (EOFError):
            break
        # print information about each extracted variable:
        for key in list(d.keys()):
            print("extracting pickled variable: name=", key,  "; size=", d[key].shape)
            #print("type=",type(d[key]))
        globals().update(d)

# finish extracting the dictionary with glacier data, and print one entry for example:
front_variation=copy.deepcopy(front_variation.item())

# print one entry as an example:
first_glacier_name=list(front_variation.keys())[2]
print("\n\nAs an example, the third [i=2] glacier name:",first_glacier_name \
    ,", and its data: \nyears=",front_variation[first_glacier_name][0]
      ,"\nfront location (m)=",front_variation[first_glacier_name][1]
     ,"\nlon/lat=",front_variation[first_glacier_name][2])
```

Explanation of variables

front_variation: a dictionary where the key is the 'glacier name'/'2-letter country code'; e.g. VACAS/AR is in Argentina. the elements of the dictionary are the years of data, glacier length relative to 1960 and the [lon,lat] of the glacier. The example above shows how to extract each of these from the dictionary.

Ts_smooth: annually averaged smooth global mean surface temperature

Ts_smooth_years: the corresponding years

Quelccaya_delta18O1976, Quelccaya_depth1976 and Quelccaya_delta18O2016, Quelccaya_depth2016: two ice core records showing seasonal variation of $\delta^{18}O$.

Quelccaya_years, Quelccaya_delta18O: a longer combined record to be used as a temperature proxy

1) Glacier length records:

(a) Plot the time evolution of all glacier lengths relative to 1960.

[ ]:
```
# plot all glaciers as function of year:
fig=plt.figure(figsize=(6,4),dpi=200)
for glacier_name in front_variation.keys():
    years=XX
    locations=XX
    plt.plot(years,locations/1000,'-',lw=0.5 \
        ,markersize=1.5,label=glacier_name)

plt.xlabel(XX)
plt.xlim(XX)
plt.ylabel("front location relative to 1960 (km)")

plt.tight_layout()
plt.show()
```

(b) Plot the length time series of one glacier from each of the following: Argentina, Peru, New Zealand, United States, Russia, Switzerland, and Tajikistan. Choose the glacier with the record with the largest number of data points for each of these countries.

[ ]:
```
# plot a few glaciers from different countries.
# see https://www.nationsonline.org/oneworld/country_code_list.htm
# To find them, print the list of glaciers with the length of record (number of data␣
 ↪points)
# of each glacier:
if 0:  # change to 1 to see list of glaciers
    for glacier_name in front_variation.keys():
        print(glacier_name, len(front_variation[glacier_name][0]))

# insert glacier names into following list based on the printed full list above:
names_to_plot=['XX','XX',...]

# plot:
fig=plt.figure(figsize=(6,4),dpi=200)
for glacier_name in names_to_plot:
    years=XX
    locations=XX
    # calculate linear regression line for this glacier:
    fit=linregress(XX)
    slope=XX;
    intercept=XX;
    p=XX;
    r2=(fit.rvalue)**2
    regression_line=intercept+slope*years
    # plot:
    hl=plt.plot(years,locations/1000,'-',lw=0.5 \
        ,markersize=1.5,label=glacier_name)
```

```
    # add the regression line in dash, same color:
    color=hl[0].get_color()
    plt.plot(years,regression_line/1000,'--',lw=0.5,color=color)

plt.xlabel(XX)
plt.xlim(XX)
plt.ylabel("front location relative to 1960 (km)")
plt.legend(fontsize=6)

plt.tight_layout()
plt.show()
```

(c) Calculate and plot the bin-average over all glaciers of glacier length relative to 1960 as a function of year.

[ ]:
```
# calculate the bin-averaged record by average over decadal-bins

props = dict(boxstyle='round', edgecolor="white", facecolor='white', alpha=1.0)

# first, combine all glacier record into a single one:
years_combined =[]; locations_combined=[];
for glacier_name in front_variation.keys():
    years=front_variation[glacier_name][0]
    locations=np.asarray(front_variation[glacier_name][1])
    years_combined.extend(years)
    locations_combined.extend(locations)

# bin-averaged glacier length based on years_combined and locations_combined:
bin_edges=np.arange(XX,XX,3) # times of edges of bins to average over
# calculate number of observations per bin:
bin_count, bin_edges, binnumber \
      = stats.binned_statistic(XX,XX, statistic='count',bins=bin_edges)
# calculate bin averages:
bin_means, bin_edges, binnumber \
      = stats.binned_statistic(XX,XX, statistic='mean',bins=bin_edges)
years_axis=(bin_edges[0:-1]+bin_edges[1:])*0.5

# plot bin-averages:
fig=plt.figure(figsize=(7,2.5),dpi=300)
plt.subplot(1,2,1)
plt.plot(years_axis,bin_means/1000,lw=1,color="b")
plt.xlabel(XX)
plt.ylabel(XX)
plt.xlim(XX,XX)


# plot number of observations per bin:
plt.subplot(1,2,2)
plt.semilogy(years_axis, bin_count)
plt.xlabel(XX)
plt.ylabel(XX)
plt.xlim(XX)

plt.tight_layout()
```

```
plt.show()
```

2. Temperature and glacier length:

(a) Plot glacier length over time for the Morteratsch Glacier (Vadret da Morteratsch) in Switzerland.

[ ]:
```
# glacier location time series:
# -----------------------------
glacier_years=np.asarray(front_variation['MORTERATSCH, VADRET DA/CH'][0])
glacier_locations=np.asarray(front_variation['MORTERATSCH, VADRET DA/CH'][1])

# first, plot glacier length record
plt.figure(dpi=300)
plt.plot(XX)
plt.ylabel("XX")
plt.xlabel("XX")
```

(b) Plot the annually and globally averaged surface temperature.

(c) Calculate and superimpose the temperature reconstructed from the length record of the above glacier using equation (11.1): Find, via trial and error, a glacier sensitivity c and a response time that lead to a temperature that is qualitatively similar to the globally averaged one, with the "hiatus" periods and accelerated warming in the second half of the 20th century.

(d) Superimpose a line plot of the temperature that is in equilibrium with the glacier length at any given time using the sensitivity calculated above. Discuss the character and reasons for the difference between this equilibrium temperature and the one reconstructed using equation (11.1).

[ ]:
```
# smooth time series:
# -------------------
# smooth annual mean time series: use a window size of 7 and a polynomial order of 1␣
 ↪(linear fit)
glacier_locations = savgol_filter(XX,XX,XX)

# use Oerleman's formula to get temperature from glacier extent:
# --------------------------------------------------------------
dt=(glacier_years[1:]-glacier_years[:-1])
dLdt=(XX-XX)/dt
c_equilib=3000
tau=20
T_glacier=XX
T_glacier_equilib=XX


# -----
# plot:
# -----
fig=plt.figure(figsize=(5,6),dpi=300)

plt.plot(XX,label="observed T")
plt.plot(XX,label="reconstructed T")
plt.plot(XX,label="equilibrium")
plt.ylabel("Temperature")
```

```
plt.xlabel("year")
plt.legend();
```

3. Idealized glacier length adjustment scenarios:

(a) Plot time series of idealized glacier length adjustment scenarios with a temperature that changes abruptly at t=0 and then remains constant, and using the exponential solution to equation (11.1). Use the same c and parameters calculated in question 2. For the first scenario, let the initially imposed temperature anomaly T_0' be -1 °C and the initial length L_0' be 4000 m. For the second scenario, $T_0' = -2$ °C and $L_0' = 1000$ m.

[ ]:
```
# set time array and various parameters for both scenarios:
t=np.asarray(np.arange(0,100,1))
tau=XX
c=XX
T0a=XX
T0b=XX
L0a=XX
L0b=XX

# The solutions for both scenarios:
La=XX
Lb=XX
label_a="$T_0="+repr(T0a)+",\ L_0=$"+repr(L0a)
label_b="$T_0="+repr(T0b)+",\ L_0=$"+repr(L0b)

# plot both solutions:
fig=plt.figure(figsize=(5,3),dpi=300)
plt.plot(t,La/1000,color="r",label=label_a)
plt.plot(t,Lb/1000,color="b",label=label_b)
plt.xlabel("years")
plt.ylabel("lengtn anomaly, $L'(t)$ (km)")
plt.xlim(-1,100)
plt.grid(lw=0.25)
plt.legend()
plt.tight_layout();
```

(b) Optional extra credit: Glacier length response to a varying temperature: Find and plot the solution to equation (11.1) for a linearly changing temperature. Use start and end anomaly temperatures of $T_0 = -0.25$ K, $T_f = 1$ K, climate sensitivity of $c = 3000$ m/K, adjustment time of $\tau = 20$ yr, a total period of 140 yr, and an initial length anomaly of $L_0 = 6000$ m. Explain the structure of the solution you find. The equation is

$$XX \tag{1}$$

The general solution is

$$L(t) = XX \tag{2}$$

Now use an initial condition of $L(t = 0) = L_0 = 1.3$ km to find any integration constants, so that the final solution is,

$$XX \tag{3}$$

```
[ ]: # set parameters:
L0=XX # m
c=XX # m/K
tau=XX # years
T0=XX # K
Tf=XX     # K
Deltat=XX # total period to calculate/plot, in years
time=np.arange(0,Deltat,Deltat/100)
T=XX # an array with temperature at all times:
L=XX # solution

# plot prescribed linear temperature:
plt.figure(figsize=(7,3),dpi=200)
plt.subplot(1,2,1)
XX

# plot resulting glacier length anomaly record:
plt.subplot(1,2,2)
XX

plt.tight_layout()
print("initial L=",L[0])
```

4) Isotopic records from Quelccaya ice cores:

a) Plot the two provided individual ice cores from 1976 and 2016 and rationalize the amplitude of the seasonal cycle seen in them.

b) Plot the decadal bin average of the longer isotopic record for the past 2000 years or so, add shading representing the std of each decadal average. Discuss the seen trends seen since 1750 and in the last few decades in the context of the longer observed record.

```
[ ]: # calculate bin-averaged glacier length:
bin_edges=np.arange(200,2020,10)
bin_means, bin_edges, binnumber \
      = stats.binned_statistic(Quelccaya_years,Quelccaya_delta18O,␣
 ↪statistic='mean',bins=bin_edges)
# bin-averaged std:
bin_std, bin_edges, binnumber \
      = XX
# number of data points per bin:
bin_count, bin_edges, binnumber \
      = XX
years_axis=(bin_edges[0:-1]+bin_edges[1:])*0.5

# mean of decadal averages:
mean=np.nanmean(bin_means)

# plot:
fig=plt.figure(figsize=(7,3.5),dpi=200)

plt.subplot2grid((1,3), (0, 0), colspan=1)
plt.plot(XX,label="1976")
plt.plot(XX,label="2016")
```

```
plt.ylabel('Depth (m)')
plt.xlabel('$\delta^{18}$O (%)')
plt.legend()
plt.title("XX")


plt.subplot2grid((1,3), (0, 1), colspan=2)
plt.fill_between(XX,color="cyan")
plt.plot(years_axis,bin_means)
plt.xlabel("Year")
plt.ylabel('$\delta^{18}$O (%)')
plt.title("XX")

plt.tight_layout()
```

5) Optional extra credit: The global picture: Calculate a linear regression for each of the given glacier records from the global data set. Draw a global map with circles indicating the latitude and longitude of glaciers, with red indicating negative length trends (slope). Make the size of the circles reflect the magnitude of the trend. Superimpose blue crosses at the location of glaciers showing positive trends.

[ ]:
```
# calculate linear regression for each glacier:
slopes=[]
for glacier_name in front_variation.keys():
        years=XX
        locations=XX
        # calculate linear regression and store result in dictionary:
        fit=linregress(years, locations)
        slope=XX
        intercept=XX;
        slopes.append(slope)
        # add the regression result to the dictionary:
        if len(front_variation[glacier_name])<4:
            front_variation[glacier_name].append([intercept,slope])
        else:
            front_variation[glacier_name][3]=[intercept,slope]

# calculate standard deviation for normalizing symbol size based on trend magnitude:
std_slope=XX

# replace trend with normalized one for plotting purposes:
for glacier_name in front_variation.keys():
    normalized_slope=XX
    front_variation[glacier_name][3][1]=normalized_slope

# mark glacier locations on a map, indicating separately those with positive and␣
 ↪negative trends:

# draw a map with coastlines first:
projection=ccrs.PlateCarree(central_longitude=0.0);
```

```
fig,axes=plt.subplots(1,1,figsize=(6,4.5),dpi=300,subplot_kw={'projection':
 ↪projection});
axes.set_extent([-180, 180, -90, 90], crs=ccrs.PlateCarree())
axes.coastlines(resolution='110m',lw=0.3,color="grey")
axes.gridlines(linewidth=0.2)


# mark a red/blue markes at glacier locations depending if they are
# decreasing/ increasing based on the linear regression:
# negative trends
N_negative_trends=0
for glacier_name in front_variation.keys():
        [lon,lat]=front_variation[glacier_name][2]
        if front_variation[glacier_name][3][1]<0:
                N_negative_trends=N_negative_trends+1
                plt.
 ↪plot(lon,lat,'o',color='r',markerfacecolor='none',markersize=-3*front_variation[glacier_name][3][1])

# positive trends:
XX

plt.show()

print("found",N_negative_trends,"negative trends and",N_positive_trends,"positive
 ↪trends.")
```

[ ]:

12 Droughts

Global Warming Science

https://courses.seas.harvard.edu/climate/eli/Courses/EPS101/

Droughts!

Please use the template below to answer the workshop questions. "XX" indicates places where you need to complete/write code or add a discussion.

Your name:

[ ]:
```
# Load libraries and data:
import numpy as np
import pickle
import matplotlib.pyplot as plt
from scipy import stats
from scipy.integrate import solve_ivp
from scipy import signal
from scipy import optimize
import warnings
# cortopy allows to plot data over maps in various spherical prjections"
import cartopy.crs as ccrs
from datetime import datetime


# load the data from a pickle file:
with open('./droughts_variables.pickle', 'rb') as file:
    while 1:
        try:
            d = pickle.load(file)
        except (EOFError):
            break
            # print information about each extracted variable:
        for key in list(d.keys()):
                if isinstance(d[key], list):
                    print("extracting pickled variable: name=", key\
                      ,  "; len=", len(d[key]))
                else:
                    print("extracting pickled variable: name=", key\
                      ,  "; size=", d[key].shape)
        globals().update(d)

# font required by PUP:
warnings.filterwarnings("ignore", message='Myriad Pro')
warnings.filterwarnings("ignore", message="findfont")
plt.rcParams['font.family'] = 'Myriad Pro'
fontsize=9
plt.rcParams['font.size'] = fontsize

# One of the variables is a dictionary and needs to be further extracted:
# precipitation_station_data_dictionary=precipitation_station_data_dictionary[()]
# print("extracting pickled variable: name=precipitation_station_data_dictionary,␣
 ↪size="
#       ,len(precipitation_station_data_dictionary))
```

```
print("done.")
```

Explanation of variables

Tree-ring data:

(Note that all_individual_tree_records_widths/ all_individual_tree_records_widths_years are both lists. Each element of the list is an individual tree record/ time axis. The above "size" of these variables, is the number of tree records in each list. The length of each record is different. Each record can be accessed and printed, as shown further below)

all_individual_tree_records_widths, all_individual_tree_records_years

[Tree ring data are in units of 0.01 mm]

analyzing drought projections of two climate models for the Sahel and SW US: GFDL vs MIROC

(each variable contains a time axis and the corresponding quantity)

sahel_gfdl_evaporation, sahel_gfdl_moisture, sahel_gfdl_precipitation

sahel_miroc_evaporation, sahel_miroc_moisture, sahel_miroc_precipitation

sw_gfdl_evaporation, sw_gfdl_moisture, sw_gfdl_precipitation,

sw_miroc_evaporation, sw_miroc_moisture, sw_miroc_precipitation

1. Past and future of Sahel and Southwest United States droughts: Plot the provided results of the two climate models for 1850–2100, based on a projected RCP8.5 scenario, showing the following time series for the Sahel: (1) evaporation and precipitation (in m/yr), (2) evaporation minus precipitation, (3) soil moisture. Repeat, in three more panels, for the Southwest United States. Describe what you see in each panel, discussing the likelihood of future droughts in each region and the expected uncertainty.

[ ]:
```
# plot:
fig1 = plt.figure(1,figsize=(15,10)); plt.clf()
plt.subplot(2,3,1)
plt.ylabel("m/yr")
plt.xlabel("Year")
plt.title("Sahel Evaporation and precipitation")
plt.plot(sahel_gfdl_evaporation[0,:], sahel_gfdl_evaporation[1,:], lw=2, label="GFDL␣
 ↪E")
plt.plot(sahel_gfdl_precipitation[0,:], sahel_gfdl_precipitation[1,:], lw=2,␣
 ↪label="GFDL P")
plt.plot(sahel_miroc_evaporation[0,:], sahel_miroc_evaporation[1,:], lw=2,␣
 ↪label="MIROC E")
plt.plot(sahel_miroc_precipitation[0,:], sahel_miroc_precipitation[1,:], lw=2,␣
 ↪label="MIROC P")
plt.legend()


plt.subplot(2,3,2)
plt.ylabel("m/yr")
```

```
plt.xlabel("Year")
plt.title("Sahel Evaporation minus Precipitation")
plt.plot(XX,XX)
plt.legend()


plt.subplot(2,3,3)
plt.ylabel("kg/m$^2$")
plt.xlabel("Year")
plt.title("Sahel soil moisture anomalies for both models")
XX # calculate and remove time-mean from each model [use np.mean()]
plt.plot(XX,XX)
plt.legend()


plt.subplot(2,3,4)
plt.ylabel("m/yr")
plt.xlabel("Year")
plt.title("SW US Evaporation and precipitation for both models")
plt.plot(XX,XX)
plt.legend()


plt.subplot(2,3,5)
plt.ylabel("m/yr")
plt.xlabel("Year")
plt.title("SW-US Evaporation minus Precipitation")
plt.plot(XX,XX)
plt.legend()


plt.subplot(2,3,6)
plt.ylabel("kg/m$^2$")
plt.xlabel("Year")
plt.title("SW-US soil moisture anomalies")
XX remove mean
plt.plot(XX,XX)
plt.legend()
plt.tight_layout()
plt.show()
```

2. Identifying ACC in a long-term Southwest United States drought record:

Explanation of the input tree ring data:

Note that all_individual_tree_records_widths/ all_individual_tree_records_widths_years are both lists. Each element of the list is an individual tree record/ time axis. The above "size" of these variables, is the number of tree records in each list. The length of each record is different. Each record can be accessed and printed, for example using:

[ ]:
```
# print for example record #3:
print(" data=",all_individual_tree_records_widths[3])
print(" years=",all_individual_tree_records_years[3])
```

```
print("\n some of the data=",all_individual_tree_records_widths[3][3:8])
print(" and the corresponding years=",all_individual_tree_records_years[3][3:8])
```

(a) Plot the 2000 yr time series of PDSI from the North American Drought Atlas.

[ ]:
```
pdsi=PDSI_timeseries[1,:]
years=PDSI_timeseries[0,:]

pdsi_threshold=-2 # a drought is defined as being less than this value

# plot the pdsi time series:
# --------------------------
plt.figure(figsize=(15,5))
# plot the PDSI time series
plt.plot(XX ,label="pdsi")
# add a thin dash line at pdsi=0:
plt.plot(XX,XX)
# add a thin dash line at pdsi=threshold value used to define droughts:
plt.plot(XX,XX)
plt.legend()
plt.xlabel("XX")
plt.ylabel("XX")
plt.title("PDSI averaged over South West US")
plt.tight_layout()
```

(b) Use non-parametric statistics to figure out if the number of droughts in the most recent N=30 yr in the record is unusual within 95%, defining a drought as being PDSI<−2, and using 10,000 realizations.

[ ]:
```
# calculate number of drought years in the last N_years of data:
N_years=30
# initialize counter:
counter=pdsi*0.0
XX # set elements of counter that correspond to pdsi<=pdsi_threshold to 1
XX print counter to understand what it represents, and then
# calculate and print the total number of drought events in the last N_years,
# using the above variable counter
XX

# scramble years in the data to see how many droughts randomly
# occur in the lasts 50 years and compare to the actual data
N_permutations=10000
num_droughts_in_last_N_years=np.zeros(N_permutations)
for i in range(0,N_permutations):
    counter=pdsi*0.0
    pdsi_randomized=np.random.permutation(pdsi)
    # count number of droughts in the last N_years:
    XX # set elements of counter that correspond to pdsi<=pdsi_threshold to 1
    num_droughts_in_last_N_years[i]=XX

# plot probability distribution fun ction of number of drought events in last N years:
plt.figure(figsize=(6,6))
plt.hist(num_droughts_in_last_N_years,bins=list(range(0,20)));
plt.xlabel("XX")
```

```
plt.ylabel("XX")
plt.title("XX");
```

3. Tree-ring data: Plot (in thin gray lines) the time series of tree-ring width records for bigcone Douglas-fir trees from the San Bernardino Mountains as a function of time. Plot (in a thick, color line) the decadal bin-average. Comment on the scatter around the average and discuss the implications for the uncertainty of the reconstructed drought record based on these data.

[ ]:
```
# plot the individual tree-ring records in gray:
plt.figure(figsize=(15,5))
# supplement the following line with a for loop to plot all records:
for XX:
     XX
plt.plot(all_individual_tree_records_years[3][:]
            ,all_individual_tree_records_widths[3][:]\
        ,linewidth=0.5,color="grey")
plt.xlabel("year")
plt.ylabel("width")
plt.title("tree-ring width record")

# calculate the bin-averaged record by average over decadal-bins
# first combine all tree record into a single one:
years =[]; widths=[];
for sublist in all_individual_tree_records_years:
    for item in sublist:
        years.append(item)
for sublist in all_individual_tree_records_widths:
    for item in sublist:
        widths.append(item)

# now calculate bin-average:
bin_means, bin_edges, binnumber \
      = stats.binned_statistic(years,widths, statistic='mean',bins=200)
years_axis=(bin_edges[0:-1]+bin_edges[1:])*0.5

# now plot over the individual tree records:
plt.plot(XX,XX,'r-',linewidth=2,label="bin-average");
```

Discussion: XX

(b) Create a normally-distributed random data set for a variable x with a zero mean and std of 1, and with the number of points equal to $365 \times 20 \times 20 \times 10^i$, where $i = 0, 1, 2, 3$. Plot the probability distribution function histogram $PDF(x)$ for $-4 < x < 4$ in each case on a log y-axis. Discuss how the estimate of the tail of the distribution improves with an increasing number of data points.

[ ]:
```
plt.figure(dpi=300)
for i in range(0,4):
    N=365*20*10**i
    data=np.random.normal(loc=0.0, scale=1.0, size=N)
    plt.subplot(4,1,i+1)
    hist, bin_edges = np.histogram(data, bins=150, density=True, range=[-4,4])
```

```
    bin_centers = XX
    width=XX
    h=plt.bar(XX, XX, width=width)
    plt.yscale('log')
    plt.title("N=%g, or %g years" % (N,20*10**i) )

plt.tight_layout()
```

Discussion: XX

4. Bucket model for soil moisture:

(a) Run the bucket model with a precipitation rate of P1 =2.2 m/yr and then P2 =1.8 m/yr. Plot W(t) and P−E as a function of time. Why doesn't the soil completely dry when the precipitation rate is decreased from the value P1 that led to the first equilibrium to the lower value P2? Describe and explain the stages seen in the solution time series.

[ ]:
```
##############################################################################
# functions
##############################################################################

# saturation specific humidity:
def q_sat(T,P):
    # saturation specific humidity (gr water vapor per gram moist air):
    # inputs:
    # T: temperature, in K
    # P: pressure, in mb

    R_v = 461 # Gas constant for moist air = 461 J/(kg*K)
    R_d = 287 # Gas constant 287 J K^-1 kg^-1
    TT = T-273.15 # Kelvin to Celsius
    # Saturation water vapor pressure (mb) from Emanuel 4.4.14 p 116-117:
    ew = 6.112*np.exp((17.67 * TT) / (TT + 243.5))
    # saturation mixing ratio (gr water vapor per gram dry air):
    rw = (R_d / R_v) * ew / (P - ew)
    # saturation specific humidity (gr water vapor per gram moist air):
    qw = rw / (1 + rw)
    return qw


def Precip(t):
    # precipitation rate
    return Precip0

# Right hand side of bucket model ODE to be integrated:
def right_hand_side(t, W):
    # t: time
    # W: soil moisture
    E0=rho_air*C_D*V*q_sat(T,1000)*RH/rho_water
    if W >= W_FC and Precip(t)-E0 > 0.0:
        dW_dt=np.nan #XX
    elif W >= W_k:
        dW_dt=np.nan #XX
    else:
        dW_dt=np.nan #XX
```

```
    return dW_dt


# ODE solver:
def solve_ODE():
    """
    Solve the ODE for soil moisture econtent
    """
    tspan=(times_save[0],times_save[-1])
    sol = solve_ivp(fun=lambda t,W: right_hand_side(t,W) \
                ,vectorized=False,y0=W0,t_span=tspan,t_eval=times_save,rtol=1.e-6)
    time_array=sol.t[:]
    W_array=sol.y[0,:]

    # calculate the RHS of the differencial equation for all times, for plotting:
    dW_dt_array=0.0*W_array
    for i in range(0,len(W_array)):
        t=1.0*time_array[i]
        W=1.0*W_array[i]
        dW_dt_array[i]=right_hand_side(t, W)

    return W_array, time_array/YEAR, dW_dt_array


#############################################################################
# set parameters
#############################################################################

YEAR = 31536000 # a year in seconds
DAY=86400
YEARS_TO_RUN = 4
# time step specifying resolutio for plotting:
dt=YEARS_TO_RUN*YEAR/1000
times_save=np.arange(0,YEARS_TO_RUN*YEAR,dt)
C_D=0.5e-3
RH=0.6
rho_air=1.225 # kg/m^3
rho_water=1000 # kg/m^3
V=10 # wind velocity m/s
W_FC=0.15
W_k=0.75*W_FC
T=25+273.15 # temperature in Kelvin

# initial values for soil moisture, must be smaller than W_FC:
W0 = np.array([0.2]) # m, solve_ivp really wants this to be a numpy array


#############################################################################
# Main program
#############################################################################

# solve ODE:
Precip1=1.8/YEAR # in units of m/s
```

```
Precip0=Precip1
W_array1, time_array1, dW_dt_array1 = solve_ODE()
Precip2=2.2/YEAR # in units of m/s
Precip0=Precip2
W_array2, time_array2, dW_dt_array2 = solve_ODE()

# plot soil water content:
fig=plt.figure(1,figsize=(6,2),dpi=300);
plt.clf()
plt.subplot(1,2,1)
plt.plot(time_array1, W_array1, "-", color='r', label="soil moisture,␣
 ↪P="+repr(Precip1*YEAR)+" m/year")
plt.plot(time_array2, W_array2, "-", color='b', label="soil moisture,␣
 ↪P="+repr(Precip2*YEAR)+" m/year")
plt.plot(time_array1, 0*W_array1, "--", color='black', lw=0.5)
plt.title('soil water content (m)')
plt.ylabel('W (m)')
plt.xlabel('Time (years)')
plt.legend(loc='upper right')


# plot P-E, rhs of soil moisture equation:
plt.subplot(1,2,2)
plt.plot(time_array1, dW_dt_array1*YEAR, "-", color='r', label="dW/dt,␣
 ↪P="+repr(Precip1*YEAR)+" m/year")
plt.plot(time_array2, dW_dt_array2*YEAR, "-", color='b', label="dW/dt,␣
 ↪P="+repr(Precip2*YEAR)+" m/year")
plt.plot(time_array1, 0*W_array1, "--", color='black', lw=0.5)
plt.title('P-E (m/year)')
plt.ylabel('dW/dt (m/year)')
plt.xlabel('Time (years)')
plt.legend(loc='lower right')
plt.tight_layout()
plt.show()
```

(b) Optional extra credit: Drive the bucket model with a precipitation rate of 2 m/yr plus a white noise time series with a zero mean and a specified std. Compare the occurrence of meteorological versus hydrological drought events from the model output via their PDFs and power spectra, and rationalize your results Hint: a red noise sequence f_n is produced from a gaussian random white noise ν_n as follows,

$$R = \exp(-dt/T_{correlation})$$
$$f_n = R \cdot f_{n-1} + \sqrt{1-R^2} \cdot \nu_{n-1}$$

where $T_{correlation}$ is the desited correlation time of the noise, and dt is the time step between two elements in the desired red noise sequence.

[ ]:
```
# bucket model driven with stochastic precipitation:

YEAR = 31536000 # a year in seconds
DAY=86400
```

```
YEARS_TO_RUN = 100
MONTH=XX
# time step specifying resolution for plotting:
dt=YEARS_TO_RUN*YEAR/20000
times_save=np.arange(0,YEARS_TO_RUN*YEAR,dt)
N=len(times_save)

# creat a random sequence $\nu_n$:
np.random.seed(0)
random=np.random.randn(N)

# creaete a red-noise precipitation record:
rednoise=np.zeros(N)
T_corr=MONTH
R=XX
for n in range(1,N):
    rednoise[n]=XX
# dimensional precipitation red noise time series:
precipitation_fixed_points=Precip0*(1+0.05*rednoise)


# redefine precip function to include a stochastic componenet
def Precip(t):
    # returns precipitation rate at time t.

    # interpolate from red noise sequence to time t:
    precipitation=np.interp(t,times_save,precipitation_fixed_points)

    return precipitation


tspan=(times_save[0],times_save[-1])
# initial values for soil moisture, must be smaller than W_FC:
W0 = np.array([0.11]) # m, solve_ivp really wants this to be a numpy array
sol = solve_ivp(fun=lambda t,W: right_hand_side(t,W) \
     ,vectorized=False,y0=W0,t_span=tspan,t_eval=times_save,rtol=1.e-6)
time_array=sol.t[:]
W_array=sol.y[0,:]


fig=plt.figure(figsize=(6,4),dpi=300)
plt.subplot(3,2,(1,2))
# precipitation forcing vs time:
plt.plot(XX)
plt.xlabel("Time (years)")
plt.ylabel("Precip (m/yr)");
plt.xlim(80,90)

plt.subplot(3,2,(3,4))
# soil moisture vs time:
plt.plot(XX)
plt.xlabel("Time (years)")
plt.ylabel("W (m)");
```

```
plt.xlim(80,90)

plt.subplot(3,2,5)
f,psd_random=signal.welch(random,dt)
plt.semilogy(psd_random[1:])
plt.title("spectrum for precipitation")
plt.xlabel("frequency")

plt.subplot(3,2,6)
XX
plt.title("spectrum for soil moisture")
plt.xlabel("frequency")

plt.tight_layout()
```

5. Optional extra credit: Composite analyses of drought years:

(a) Analyze the provided results from a long control run of a climate model to calculate and contour the rainy season averages (composites) of sea level pressure and precipitation anomalies for times in which the area-mean precipitation over the region of your city of birth is 1 std below its long-term average. Discuss your results.

(b) Plot, describe, and explain the precipitation over the tropical Pacific, from Australia to South America, averaged over times corresponding to El Niño versus La Niña events. Define the times of these events as the NINO3.4 index being above and below 1 std.

[ ]:
```
# reproduce something like textbook Figure 12.2
XX
```

6. Guiding questions to be addressed in your report:

(a) Why are droughts important to understand and predict?

(b) Why and how can they change due to natural climate variability? Due to anthropogenic climate change?

(c) How do precipitation deficit and surplus events affect soil moisture? What are the roles of the duration and magnitude of such deficit and surplus?

(d) How and based on what data can we determine if a given drought or flood may be attributable to anthropogenic climate change? What are the uncertainties in such a determination?

(e) Discuss projected drought conditions for the Southwest United States and for the Sahel and their expected socioeconomic consequences.

[ ]:
```

```

13 Floods

Global Warming Science

https://courses.seas.harvard.edu/climate/eli/Courses/EPS101/

Floods!

Please use the template below to answer the workshop questions. "XX" indicates places where you need to complete/write code or add a discussion.

Your name:

[ ]:
```
# Load libraries and data:
import numpy as np
import pickle
import matplotlib.pyplot as plt
from scipy import stats
from scipy.integrate import solve_ivp
from scipy import signal
from scipy import optimize
import warnings
# cortopy allows to plot data over maps in various spherical prjections"
import cartopy.crs as ccrs
from datetime import datetime


# load the data from a pickle file:
with open('./floods_variables.pickle', 'rb') as file:
    while 1:
        try:
            d = pickle.load(file)
        except (EOFError):
            break
            # print information about each extracted variable:
        for key in list(d.keys()):
            if key != "precipitation_station_data_dictionary":
                print("extracting pickled variable: name=", key,  "; size=", d[key].
 ↪shape)
        globals().update(d)

# font required by PUP:
warnings.filterwarnings("ignore", message='Myriad Pro')
warnings.filterwarnings("ignore", message="findfont")
#plt.rcParams['font.family'] = 'Myriad Pro'
fontsize=9
plt.rcParams['font.size'] = fontsize

# One of the variables is a dictionary and needs to be further extracted:
precipitation_station_data_dictionary=precipitation_station_data_dictionary[()]
print("extracting pickled variable: name=precipitation_station_data_dictionary, size="
      ,len(precipitation_station_data_dictionary))

print("done.")
```

Explanation of variables

analyzing projected changes in global precipitation patterns

10-year average of precipitation vs latitude and longitude, for preidustrial and the ssp3-70 scenario:

precipitation_vs_lat_lon_1850, precipitation_vs_lat_lon_2085

latitude and longitude axes:

precipitation_lat, precipitation_lon

analyzing the wet getting wetter, dry getting drier projection:

wet_wetter_E1920: zonally-averaged, time-averaged evaporation (in m/s) vs latitude for 1920-1940

wet_wetter_E2080: same, for 2080-2100

wet_wetter_P1920: same, precipitation (in m/s)

wet_wetter_P2080: same

wet_wetter_TS1920: zonally-averaged, time averaged, surface temperature for 1920-1940

wet_wetter_TS2080: for 2080-2100

wet_wetter_lat: latitudes for above data

analyzing extreme rain events in weather station data:

precipitation_station_data_dictionary is a dictionary with precipitation time series from several weather stations. The keys are weather station names, and the values are the dates and corresponding daily precipitation values.

analyzing projections extreme precipitation events

extreme_precip_1920_1940: value of 99.9% percentile precipitation as function of latitude and ensemble member.

extreme_precip_2080_2100: same, for end of 21st century

extreme_precip_TS_1920_1940: surface temperature vs latitude

extreme_precip_TS_2080_2100: same

Data of precipitation every day for 20 years as a function of time (7300 days) and longitude (288 longitudes), at one latitude (40N):

extreme_precip_daily_precip_40N_1920_1940

extreme_precip_daily_precip_40N_2080_2100

extreme_precip_lat

1. projected changes in global precipitation patterns:

Contour the precipitation pattern for the preindustrial period, the year 2100 based on the SSP3-7.0 projection, and the difference between the two.

[ ]:
```
# plot for the preindustrial:
# ---------------------------
fig=plt.figure(figsize=(8,4))
ax = plt.axes(projection=ccrs.PlateCarree(central_longitude=0.0, globe=None))
ax.set_extent([-180, 180, -90, 90], crs=ccrs.PlateCarree())
```

```
ax.coastlines(resolution='110m')
ax.gridlines()

delta=0.2
mylevels=np.arange(0,5+delta,delta)
to_m_per_year=365*25*3600

c=plt.contourf(precipitation_lon, precipitation_lat
               ,precipitation_vs_lat_lon_1850*to_m_per_year
               ,levels=mylevels)
# draw the colorbar
clb=plt.colorbar(c, shrink=0.85, pad=0.02)
# add title/ labels:
plt.xlabel('Longitude')
plt.ylabel('Latitude')
clb.set_label('m/year')
plt.title('Precipitation, 1850-1860')
plt.show()

# plot for 2085:
# --------------
fig=plt.figure(figsize=(8,4))
XX

# plot the difference: (change the contour levels, use the bwr color map):
# ------------------------------------------------------------------------
fig=plt.figure(figsize=(8,4),dpi=200)
XX
```

2. "Wet getting wetter, dry getting drier" projections:

(a) Plot the zonally averaged precipitation minus evaporation as a function of latitude for 1920–1940 and for 2080–2100 under RCP8.5.

(b) Plot the projected change in zonally averaged precipitation minus evaporation as a function of latitude, and superimpose the estimated expected change based on the Clausius-Clapeyron scaling in equation (12.2).

[ ]:
```
# clausius clapeyron: moisture (kg/kg) over deg K
alpha=0.07

# plot:
PmE1920=wet_wetter_P1920-wet_wetter_E1920
PmE2080=XX
deltaPmE=PmE2080-PmE1920
deltaTS=XX
# "Wet getting wetter, dry getting drier" scaling here:
deltaPmE_estimate=XX
lat=wet_wetter_lat

fig=plt.figure(dpi=300)

# plot surface temperature vs latitude for the two periods
plt.subplot(2,2,1)
```

```
XX

# plot warming as a function of latitude:
plt.subplot(2,2,2)
XX


# plot precipitation-evaporation for the two periods in units of mm/day
# (convert precipitation and evaporation from m/s to mm/day)
plt.subplot(2,2,3)
XX

# plot actual change in precipitation-evaporation and superimpose the CC-scaling
plt.subplot(2,2,4)
XX

plt.tight_layout()
```

3. Analyzing extreme precipitation in weather station data:

(a) For each given weather station, draw a scatter plot of precipitation as a function of time starting at 1880. Add a histogram of values plotted as a probability distribution function for the first 20 years and for the last 20 years of the record. Discuss the difficulty of analyzing the statistics of intense rain events using a short observed record.

[ ]:
```
# Scatter plot station precipitation time series:

num_keys = len(precipitation_station_data_dictionary)
fig, axs = plt.subplots(num_keys, 2, figsize=(7, 2 * num_keys),dpi=600)#, sharex=True)
nyears_hist=20

# Iterate through each key-value pair in the dictionary
for i, (key, values) in enumerate(precipitation_station_data_dictionary.items()):

    station_name=key
    datetimes = values[0][:]
    precipitation_values = values[1][:]

    # Plot the time series:
    # ---------------------
    axs[i,0].scatter(XX, XX
                     ,marker='x',linewidth=0.2,s=12
                     ,label=station_name)
    axs[i,0].set_title(station_name)
    axs[i,0].set_ylabel('Precipitation (mm)')

    # Add labels and legend:
    axs[i,0].set_xlim(datetime(1880,1,1),datetime(2022,12,31))

    # Plot the histograms:
    # --------------------
    x=np.asarray(precipitation_station_data_dictionary[station_name][1])
    x[np.isnan(x)]=0.0
    x1=x[:XX]    # first nyears_hist years
```

```
    x2=x[XX:]    # last nyears_hist years
    xmax=XX # largest element in both periods
    xmin=XX
    hist_x1, bin_edges_x1 = np.histogram(x1, bins=50, density=True, range=[xmin,xmax])
    hist_x2, bin_edges_x2 = np.histogram(XX)
    hist_x1[0]=0
    hist_x2[0]=0
    # Calculate bin centers
    bin_centers = (bin_edges_x1[:-1] + bin_edges_x1[1:]) / 2

    # Plot the PDF using bar

    width=bin_edges_x1[1] - bin_edges_x1[0]
    label=XX # year range used
    axs[i,1].bar(bin_centers, hist_x1, width=width, alpha=0.7, color='blue'
            , edgecolor='blue',label=label)
    label=XX
    axs[i,1].bar(bin_centers, hist_x2, width=width, alpha=0.7, color='red'
            , edgecolor='red',label=label)

    axs[i,1].set_title(XX)
    axs[i,1].set_ylabel(XX)
    axs[i,1].legend()
    axs[i,1].set_xlim(0,bin_edges_x1[-1])
    axs[i,1].set_yscale('log')

axs[-1,0].set_xlabel(XX)
axs[-1,1].set_xlabel(XX)

# Adjust layout and show the plot
plt.tight_layout()
```

Discussion: XX

(b) Create a normally-distributed random data set for a variable x with a zero mean and std of 1, and with the number of points equal to $365 \times 20 \times 20 \times 10^i$, where $i = 0, 1, 2, 3$. Plot the probability distribution function histogram $PDF(x)$ for $-4 < x < 4$ in each case on a log y-axis. Discuss how the estimate of the tail of the distribution improves with an increasing number of data points.

[ ]:
```
plt.figure(dpi=300)
for i in range(0,4):
    N=365*20*10**i
    data=np.random.normal(loc=0.0, scale=1.0, size=N)
    plt.subplot(4,1,i+1)
    hist, bin_edges = np.histogram(data, bins=150, density=True, range=[-4,4])
    bin_centers = XX
    width=XX
    h=plt.bar(XX, XX, width=width)
    plt.yscale('log')
    plt.title("N=%g, or %g years" % (N,20*10**i) )

plt.tight_layout()
```

Discussion: XX

4. Projected extreme precipitation events:

(a) Plot the PDF of daily precipitation at 40°N for 1920–1940 and for 2080–2100 under RCP8.5. Calculate the 99.9th percentile precipitation rate (mm/day) for each period.

[ ]:
```
# calculate probability distribution function of precipitation, and then the CDF:
lat=extreme_precip_lat
Nbin_edges40N=50
bin_edges40N=np.arange(0,140.0,140.0/Nbin_edges40N)
# calculate distribution at one latitude:
hist,bin_edges_out=np.histogram(extreme_precip_daily_precip_40N_1920_1940,bin_edges40N\
                                ,density=True,range=(0,140))
distribution_40N_1920_1940=hist
dP=bin_edges_out[2]-bin_edges_out[1]
xaxis_from_bin_edges_40N_1920_1940=bin_edges_out[0:-2]+dP/2

# repeat for 2080-2100:
hist,bin_edges_out=np.histogram(XX)
distribution_40N_2080_2100=XX
dP=XX
xaxis_from_bin_edges_40N_2080_2100=XX

# ================================================================================
# plot precipitation distribution at 40N:
# ================================================================================
plt.figure(figsize=(6,3),dpi=300)
plt.semilogy(xaxis_from_bin_edges_40N_1920_1940,distribution_40N_1920_1940[1:
 ↪],"-",color="b",lw=0.5,label="1920-1940")
plt.semilogy(XX,distribution_40N_2080_2100[1:],"-",color="r",lw=0.5,label="2080-2100")
plt.legend()
plt.xlabel("Precipitation (mm/day)");
plt.ylabel("Probability");
plt.grid(lw=0.25)
```

(b) Plot the 99.9th percentile precipitation rate as a function of latitude for the two periods.

[ ]:
```
# average extreme precipitation over ensemble members:
extreme_precip_avg_1920_1940=np.mean(extreme_precip_1920_1940,axis=XX)
extreme_precip_avg_2080_2100=XX


# ================================================================================
# plot extreme precipitation:
# ================================================================================
plt.figure(figsize=(4,3),dpi=200)
# first period:
plt.plot(lat,extreme_precip_avg_1920_1940.flatten(),lw=1,color="b")
# second period:
plt.plot(XX...,color="r")
plt.xlabel("XX")
plt.ylabel("XX")
```

(c) Plot the projected percent change per degree Kelvin warming of the 99.9th percentile precipitation rate as a function of latitude based on the above curves. Superimpose the expected percent change in extreme precipitation per degree Kelvin based on the expected

change to the surface saturation specific humidity (that is, based on the Clausius-Clapeyron scaling). Discuss the justification for using the Clausius-Clapeyron scaling to predict changes in extreme precipitation and what factors may lead to the deviation from that scaling in the above plot.

[ ]:
```
# parameters needed below:
p_s=1013
dT_warming=1

def q_sat(T,P):
    # saturation specific humidity (gr water vapor per gram moist air):
    # inputs:
    # T: temperature, in K
    # P: pressure, in mb

    R_v = 461 # Gas constant for moist air = 461 J/(kg*K)
    R_d = 287 # Gas constant 287 J K^-1 kg^-1
    TT = T-273.15 # Kelvin to Celsius
    # Saturation water vapor pressure (mb) from Emanuel 4.4.14 p 116-117:
    ew = 6.112*np.exp((17.67 * TT) / (TT + 243.5))
    # saturation mixing ratio (gr water vapor per gram dry air):
    rw = (R_d / R_v) * ew / (P - ew)
    # saturation specific humidity (gr water vapor per gram moist air):
    qw = rw / (1 + rw)
    return qw


def calc_dq_sat_dT_CC_percent(TS):
    # CC scaling:
    dq_sat_dT_percent=0.0*TS
    i=-1
    for T in TS:
        i=i+1
        # calculate 100*(dq^*/dT)/q^*
        dq_sat_dT_percent[i]=XX
    return dq_sat_dT_percent


# average TS corresponding to extreme precipitation over ensemble members:
extreme_precip_avg_TS_1920_1940=XX
extreme_precip_avg_TS_2080_2100=XX
# Change in temperature during extreme precipitation, as a function of latitude:
Delta_TS=extreme_precip_avg_TS_2080_2100-extreme_precip_avg_TS_1920_1940

# calculate CDFs to be used to find the level of precipitation corresponding to 99.9␣
 ↪percentile events.
dP=bin_edges40N[1]-bin_edges40N[0] # bin width
# normalize CDFs such that they go from 0 to 100%:
CDF_40N_1920_1940=100*np.cumsum(distribution_40N_1920_1940)*dP
CDF_40N_2080_2100=XX

# find precipitation rate corresponding to specified percentile:
percentile=99.9
# first for 1920-1940:
```

```
i1=np.argmax(CDF_40N_1920_1940>percentile) # first index for which condition occurs
# prepare for interpolating to find the exact precipitation rate corresponding to
# desired percentile by finding the bin edges on the two sides of the desired␣
 ↪percentile:
PRECT1=bin_edges40N[i1-1]+dP/2
PRECT2=bin_edges40N[i1]+dP/2
CDF1=CDF_40N_1920_1940[i1-1]
CDF2=CDF_40N_1920_1940[i1]
# interpolate:
PRECT_percentile_1920_1940=PRECT1+(PRECT2-PRECT1)*(percentile-CDF1)/(CDF2-CDF1)

# repeat for 2080-2100:
XX

# ----------------------------------------------------------------------------
# first, plot surface temperature vs latitude, TS, for the two periods:
# ----------------------------------------------------------------------------
plt.figure(figsize=(6,6),dpi=300)
plt.subplot(2,2,1)

plt.plot(XX,color="b",label="$T_s$ 1920-1940")
plt.plot(XX,color="r",label="$T_s$ 2080-2100")
plt.xlabel("XX")
plt.ylabel("XX")


# ------------------------------------------------
# plot precipitation distribution at one latitude:
# ------------------------------------------------
plt.subplot(2,2,2)

plt.plot(XX,CDF_40N_1920_1940,color="b",lw=0.5)
plt.plot(XX,CDF_40N_2080_2100,color="r",lw=0.5,alpha=0.5)
plt.axhline(percentile,color="k",lw=0.5,alpha=0.5)
# vertical bars at specified percentile value for each of the two periods:
plt.axvline(XX,color="b",lw=1)
plt.axvline(XX,color="r",lw=1)
plt.xlabel("XX")
plt.ylabel("XX")


# ------------------------------------------------------------------
# plot fractional extreme precipitation change per degree warming:
# ------------------------------------------------------------------
plt.subplot(2,2,3)

mygray=[0.7, 0.7, 0.7]
Delta_extreme_TS=extreme_precip_avg_TS_2080_2100-extreme_precip_avg_TS_1920_1940
DeltaP_percent=100*(extreme_precip_avg_2080_2100-extreme_precip_avg_1920_1940) \
        /extreme_precip_avg_2080_2100
plt.plot(lat,DeltaP_percent/Delta_extreme_TS,color="r",lw=1,label="RCP8.5")
dq_sat_dT_CC_percent=calc_dq_sat_dT_CC_percent(extreme_precip_avg_TS_2080_2100)
plt.plot(lat,dq_sat_dT_CC_percent,color=mygray,linestyle="-",label="C-C")
```

```
plt.ylabel("Precipitation change (% K$^{-1}$)");
plt.xlabel("Latitude");
plt.legend()

plt.tight_layout(pad=0);
```

Discussion: XX

(d) Optional extra credit: Plot the Clausius-Clapeyron scaling for the expected increase in extreme precipitation rates with temperature, based on the dependence of the saturation specific humidity on temperature T, from 230 K to 320 K. Superimpose the dependence of the thermodynamical component of equation 12.3, normalized to have the same value as the Clausius-Clapeyron scaling at T =230 K. Discuss the difference between the two estimates. Why are these estimates less relevant in the tropics?

[ ]:
```
# calculate two scaling for a linear range of temperatures:
# ---------------------------------------------------------

# saturation specific humidity:
def q_sat(T,P):
    # saturation specific humidity (gr water vapor per gram moist air):
    # inputs:
    # T: temperature, in K
    # P: pressure, in mb

    R_v = 461 # Gas constant for moist air = 461 J/(kg*K)
    R_d = 287 # Gas constant 287 J K^-1 kg^-1
    TT = T-273.15 # Kelvin to Celsius
    # Saturation water vapor pressure (mb) from Emanuel 4.4.14 p 116-117:
    ew = 6.112*np.exp((17.67 * TT) / (TT + 243.5))
    # saturation mixing ratio (gr water vapor per gram dry air):
    rw = (R_d / R_v) * ew / (P - ew)
    # saturation specific humidity (gr water vapor per gram moist air):
    qw = rw / (1 + rw)
    return qw


def calc_dq_sat_dp_MSE(T):
    # calc dq_sat/dp at a constant MSE as function of T.
    # Input: array T.
    dq_sat_dp_MSE=T*0.0
    i=-1
    for T in TS:
        i=i+1
        dp=-1 # hPa
        dz=XX # height difference corresponding to dp
        q_s=q_sat(T,p_s) # surface specific humidity, assumed saturated
        MSE_s = XX
        # calculate T_parcel, raised adianatically from p_s to p_s+dp
        XX
        dT=T_parcel-T
        # use finite differencedd to calculate teh following two:
        dqdp=XX
        dqdT=XX
```

```
        dTdp_MSE=dT/dp
        dq_sat_dp_MSE[i]=XX

    return dq_sat_dp_MSE

def MSE_conservation(T,P,z,MSE_s,q_s):
    """ This function is called by a root finder to calculate
    the temperature of a rising parcel. it is equal to zero when
    MSE conservation is satisfied.
    """
    qsat=q_sat(T,P)
    # the root finder is looking for the value of T for which the following is zero:
    ans = cp_dry*T+L*min(q_s,qsat)+g*z-MSE_s
    return(ans)


Tbar=260 # needed for exponential presure, to convert between z and p.
g=9.81        # gravity
L=24.93e5     # latent heat J/kg (vaporization at t.p.)
cp_dry=1005   # J/kg/K
T1=np.arange(232,330,1)
R=287 # Gas constant 287 J K^-1 kg^-1
# calculate q_sat(T1) for
q_sat_T1=0.0*T1
i=-1
for T in T1:
    i=i+1
    q_sat_T1[i]=XX
dq_sat_dp_MSE_T1=calc_dq_sat_dp_MSE(T1)



# plot the two scalings:
fig=plt.figure(figsize=(5,3),dpi=300)

plt.semilogy(XX,XX,label="$q^*(T)$")
plt.semilogy(XX,XX,label="$\\left.\\frac{dq^*}{dp~}\\right|_{\\rm MSE}$")

plt.legend()
plt.xlim(230,320)
plt.grid(lw=0.25)
plt.xlabel("Temperature (K)")
plt.tight_layout();
```

5. Optional extra credit: Composite analyses of drought or flood years:

(a) Analyze the provided results from a long control run of a climate model to calculate and contour the rainy season averages (composites) of sea level pressure and precipitation anomalies for times in which the area-mean precipitation over the region of your city of birth is 1 std above the average. Discuss your results.

(b) Plot, describe, and explain the precipitation over the tropical Pacific, from Australia to South America, averaged over times corresponding to El Niño versus La Niña events. Define the times of these events as the NINO3.4 index being above and below 1 std.

[ ]:
```
# reproduce something like textbook Figure 12.2
XX
```

6. Guiding questions to be addressed in your report:

(a) Why are floods important to understand and predict?

(b) Why and how can they change due to natural climate variability? Due to anthropogenic climate change?

(c) How do precipitation deficit and surplus events affect soil moisture? What are the roles of the duration and magnitude of such deficit and surplus?

(d) How and based on what data can we determine if a given drought or flood may be attributable to anthropogenic climate change? What are the uncertainties in such a determination?

(e) What can we say about the projections of changes to global-scale time-mean precipitation patterns in a warmer climate? What about regional changes?

(f) Define extreme precipitation events. What are their socioeconomic consequences? How and why are they expected to change?

(g) Why are changes to the probability of rare precipitation events difficult to estimate using observations? Why are rare flood events difficult to attribute to anthropogenic climate change?

[ ]:
```

```

14 Heat waves

Global Warming Science

https://courses.seas.harvard.edu/climate/eli/Courses/EPS101/

Heat waves

Please use the template below to answer the workshop questions. "XX" indicates places where you need to complete/write code or add a discussion.

Your name:

[ ]:
```
# Load libraries and data:
import numpy as np
import pickle
import matplotlib.pyplot as plt
from scipy import stats
from scipy.stats import linregress
from scipy.signal import savgol_filter # for smoothing time series
import warnings
import cartopy.crs as ccrs
import datetime
from datetime import timedelta
import matplotlib.dates as mdates
from scipy.signal import savgol_filter # for smoothing time series
from scipy import optimize
from matplotlib.ticker import (MultipleLocator, FormatStrFormatter,
                               AutoMinorLocator)

# load the data from a pickle file:
with open('./heat_waves_variables.pickle', 'rb') as file:
    while 1:
        try:
            d = pickle.load(file)
        except (EOFError):
            break
            # print information about each extracted variable:
        for key in list(d.keys()):
            print("extracting pickled variable: name=", key,  "; size=", d[key].shape)
            #print("type=",type(d[key]))
        globals().update(d)
```

Explanation of input variables:

Siberia heat wave observations:

Verhojansk_Tmax: Siberian daily-maximum temperature records

Verhojansk_Tmax_dates: can be used to plot Verhojansk_Tmax vs time.

Verhojansk_Tmax_day_of_year: showing day of year from 1 to 366.

Model Tmax time series for multiple ensemble members:

heat_waves_Tmax_all_timeseries: time series of maximum daily temperatures over the Great Plains region of the US from 1920 to 2100 for 31 ensemble members of historical simulation followed by RCP8.5, from the CESM climate model.

heat_waves_ensemble_members_dates: dates for time series in heat_waves_Tmax_all_timeseries

Data as a function of time and longitude/latitude for calculating composites over region of heat waves:

heat_waves_weekly_averages_TREFHTMX, heat_waves_weekly_averages_Z500: weekly averaged data of daily maximum temp and height of the 500 mb surface,

as a function of latitude and longitude, for all augusts from 1920 to 2100. The structure of these data is explained below in the notebook.

heat_waves_lat, heat_waves_lon: lat/lon for calculating and contouring composites

1) The Siberian heat wave of summer 2020.

(a) Plot the daily maximum temperature climatology for Verhojansk, Siberia from January to December and then superimpose the 2020 temperature.

(b) Plot the anomaly time series, calculated as the deviation from the daily climatology, and use it to calculate the peak time, amplitude and length of the summer heat wave based on what you think are reasonable thresholds.

[ ]:
```
# calculate daily climatology:

Verhojansk_Tmax_climatology=np.zeros(365)
for day in np.arange(1,366):
    Verhojansk_Tmax_climatology[day-1]=np.mean(Verhojansk_Tmax[XX])

# smooth the climatology: # arguments are window_size,polynomial_order
Verhojansk_Tmax_climatology_nonsmooth = Verhojansk_Tmax_climatology
Verhojansk_Tmax_climatology = \
  savgol_filter(Verhojansk_Tmax_climatology, 21, 2)

Verhojansk_Tmax_anomaly=Verhojansk_Tmax*np.nan
Verhojansk_Tmax_climatology_all_years=Verhojansk_Tmax*np.nan
for day in range(len(Verhojansk_Tmax)):
    iday=Verhojansk_Tmax_day_of_year[day]-1
    Verhojansk_Tmax_anomaly[day]=XX
    Verhojansk_Tmax_climatology_all_years[day]=Verhojansk_Tmax_climatology[iday]

# plot daily climatology and anomaly timeseries:
plt.figure(figsize=(6,6),dpi=600)

plt.subplot(2,1,1)
# plot climatology over one year
plt.plot(XX)
plt.xlabel("XX")
plt.ylabel("XX")
plt.title("XX")


plt.subplot(2,1,2)
# plotting the anomaly:
T1=  # Tmax climatology
T2=  # Tmax itself
plt.fill_between(XX,XX,XX,where=XX>=XX \
```

```
                ,interpolate=True,lw=0.3,color="r")
plt.fill_between(XX,XX,XX,where=XX>=XX \
                ,interpolate=True,lw=0.3,color="b")
plt.xlabel("XX")
plt.ylabel("XX")
plt.title("XX")
plt.xlim(datetime.datetime.strptime("2020-01-01", '%Y-%m-%d')\
        ,datetime.datetime.strptime("2020-09-01", '%Y-%m-%d'))
# can make x-axis labeling nicer with these:
#plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%m/%d/%Y'))
#plt.xticks(rotation=XX,fontsize=XX,ha=XX)
```

[ ]:
```
# plot the anomaly plus climatology a-la Ciavarella-etal-2020, world weather␣
 ↪attribution project:
# lower panel of Fig 1, in pdf copy available at
# https://link.springer.com/article/10.1007/s10584-021-03052-w,
# see also https://www.worldweatherattribution.org/
 ↪siberian-heatwave-of-2020-almost-impossible-without-climate-change
fig=plt.figure(figsize=(3.6,2.2),dpi=600)
plt.subplot(3,1,3)
# plot anomaly only:
plt.plot(XX,XX)
plt.xlabel("XX")
plt.ylabel("XX")
plt.title("XX")
plt.xlim(XX,XX)
plt.ylim(XX,XX)
plt.xticks(XX)

plt.tight_layout()
```

2 Heat stress and the wet bulb temperature.

(a) Calculate the wet bulb temperature (to within 0.001 °C) for an air parcel with temperature of 37 °C and relative humidity of 60%: evaluate the appropriate energy balance for different values of T_w until you find the one that satisfies it.

[ ]:
```
c_p  = 1005      # J/kg/K
L       = 24.93e5  # latent heat J/kg (vaporization at t.p.)
P=1000

def q_sat(T,P):
    """
    Calculate saturation specific humidity (gr water vapor per gram moist air).
    inputs:
    T: temperature, in Kelvin
    P: pressure, in mb
    """
    qw=XX
    return qw

T=37  # Celsius, need to convert to Kelvin to be used in the function q_sat
rh=0.60
```

```
# try different values of T_w until the RHS equals the LHS:
T_w=XX # Celsius, need to convert to Kelvin to be used in the function q_sat
RHS=XX
LHS=XX
print("RHS=",RHS,", LHS=",LHS,", RHS-LHS=",RHS-LHS)
```

(b) Optional extra credit: Calculate and contour the WBT as a function of relative humidity and temperature.

[ ]:
```
# define a function that calculates WBT from the above heat budget

cp_dry  = 1005     # J/kg/K
c_p=1.0*cp_dry
L       = 24.93e5  # latent heat J/kg (vaporization at t.p.)
P=1000

def cost(T_w,rh,T):
    # rh - percent
    # T_w, T: Celsius
    J=(c_p*T+L*(rh/100)*q_sat(273.15+T,P))  -  (XX))
    return J


def my_WBT(rh,T):
    # input: scalars; rh: percent, T: Celsius
    sol = optimize.root(cost, T, args=(rh,T))
    T_w = sol.x[0]
    return T_w


# calculate WBT(rh,T):
# --------------------
rh=np.arange(0,100.1,0.1)
T=np.arange(30,50.1,0.1)
x,y=np.meshgrid(T,rh)
my_WBT_plot=np.zeros(x.shape)

irh=-1
for rh1 in rh:
    irh=irh+1
    iT=-1
    for T1 in T:
        iT=iT+1
        my_WBT_plot[irh,iT]=my_WBT(rh1,T1)

print("done.")
```

[ ]:
```
# Contour WBT as function of RH and T:
# ------------------------------------
fig=plt.figure(figsize=(4,3),dpi=300)
levels=np.arange(15,35.1,0.1)
plt.contour(x,y,my_WBT_plot,levels=levels)
hcb=plt.colorbar(label="°C")
plt.xlabel("Temperature (°C)")
```

```
plt.ylabel("Relative humidity (%)");
```

3. Understanding heat wave statistics for an RCP8.5 projection:

(a) Plot simulated temperature time series for all ensemble members for 5 yr periods starting in 1920, 2010, and 2090.

[ ]:
```
# note that heat_waves_Tmax_all_timeseries is in K, convert to C as needed:
Tmax_all_timeseries=heat_waves_Tmax_all_timeseries

years=heat_waves_ensemble_members_dates
ensemble_members=range(0,len(heat_waves_Tmax_all_timeseries[:,0]))
# calc avereage over all ensemble members:
Tmax_timeseries_avg=np.mean(Tmax_all_timeseries[:,:],axis=0)

# plot Tmax timeseries
fig=plt.figure(dpi=300,figsize=(4,6))
plt.clf()

plt.subplot(3,1,1)
iensemble_member=-1
for ensemble_member in ensemble_members:
    iensemble_member=iensemble_member+1
    plt.plot(XX,XX,lw=0.25,alpha=0.4)
plt.plot(XX,XX,lw=0.4,color='b',label="mean")
plt.ylabel(XX)
plt.xlim(datetime.date(year=1920,month=1,day=1),datetime.
  ↪date(year=1925,month=1,day=1));
plt.ylim([10,47]);
plt.legend()
plt.grid(lw=0.25);

plt.subplot(3,1,2) # 2000-2010
iensemble_member=-1
for XX:

plt.plot(XX,XX,lw=0.4,color='b',label="mean")
plt.ylabel(XX)
plt.xlim(XX,XX);
plt.ylim([10,47]);
plt.legend()
plt.grid(lw=0.25);

plt.subplot(3,1,3)
XX


plt.tight_layout()
plt.show();
```

(b) Calculate and plot PDFs of maximum daily surface temperatures for 1920-1950 and then for 2070-2100. Superimpose on the 2070-2100 plot a line for the PDF calculated from the time series of 1920-1950 plus a constant temperature increase corresponding to the mean difference between the two periods. [Hint: This is recreating Figure 13.7: "Understanding changes in heat wave

statistics" from the course notes.]

(c) Calculate and plot the CDFs for the maximum daily temperature for 1920-1950 (blue bars) and then for 2070-2100 (red bars). Superimpose a line for the CDF calculated from the time series of 1920-1950 plus a constant temperature increase corresponding to the mean difference between the two periods. Use the CDFs to calculate the expected probability of exceeding maximum daily temperature in the latter period, whose probabilities were 0.1% in the earlier period.

[ ]:
```
# for both 3b and 3c:

dates=heat_waves_ensemble_members_dates
years=np.asarray([date.year for date in dates])

# limit to desired year range:
Tmax_for_pdf=Tmax_all_timeseries[:,years<=1950]-273.15
# reshape temperature data into a single array:
Tmax_for_pdf.reshape(np.prod(Tmax_for_pdf.shape))

# calculate the histograms:
Tmax_hist, Tmax_bin_edges = np.histogram(Tmax_for_pdf, bins=range(XX), density=True)
# center of bins, used for x-axis when plotting and calculating CDF:
Tmax_x=(Tmax_bin_edges[1:]+Tmax_bin_edges[0:-1])/2
# width of bins, used for calculating PDF:
Tmax_dx=(Tmax_bin_edges[1:]-Tmax_bin_edges[0:-1])
# CDF=cumulative sum over Tmax_hist*dx:
Tmax_cdf=np.cumsum(XX)

# print the temperature of an event whose probability is 0.1%
print("temperatures of events whose probability is less than 0.1%:")
print(Tmax_x[100-100*Tmax_cdf<=0.1])

# Tmax pdfs:
fig=plt.figure(figsize=(7,3),dpi=300)
plt.subplot(1,2,1)
h1=plt.bar(Tmax_x,Tmax_hist,color="b");
plt.xlabel("$T_{\\rm max}$ ($^\circ$C)")
plt.ylabel("PDF")
plt.grid(lw=0.25);

plt.subplot(1,2,2)
h1=plt.bar(Tmax_x,Tmax_cdf,color="b");
plt.xlabel("$T_{\\rm max}$ ($^\circ$C)")
plt.ylabel("CDF")
plt.grid(lw=0.25);

plt.tight_layout()
```

[ ]:
```
# now repeat for year>=2070:
# limit to desired year range:
XX

# calculate the histograms:
XX
```

```
# print the temperature of an event whose probability is 0.1%
print("probability of events whose temeprature is above that that was 0.1% before␣
 ↪warming:")
print(np.max(100-100*Tmax_cdf[XX]))

# now repeat the calculation of the histogram for year<1950, this time adding the mean
# temperature difference between the two periods:
# First, find mean temperature difference between 2070-2100 and 1920-1950:
warming=np.mean(XX)-np.mean(XX)
print("mean warming between two periods is",warming,"degree C.")
# limit to desired year range:
Tmax_for_pdf_shifted=Tmax_all_timeseries[:,years<1950]-273.15+warming
# reshape temperature data into a single array:
Tmax_for_pdf_shifted.reshape(np.prod(Tmax_for_pdf_shifted.shape))

# calculate the histograms:
Tmax_hist_shifted, Tmax_bin_edges_shifted = XX
Tmax_x_shifted=XX
Tmax_dx_shifted=XX
Tmax_cdf_shifted=np.cumsum(XX)

# plot:
# Tmax pdfs:
fig=plt.figure(figsize=(7,3),dpi=300)
plt.subplot(1,2,1)
h1=plt.bar(Tmax_x,Tmax_hist,color="b");
plt.xlabel(XX")
plt.ylabel("PDF")
plt.plot(XX,XX,color="g",drawstyle='steps');

# Tmax CDFs:
plt.subplot(1,2,2)
XX
```

(d) The two complementary views of extreme event statistics: (i) Calculate the return time of a 43 °C daily maximum temperature event for 1920–1950 and then for 2070–2100. (ii) Calculate the daily maximum temperature of an event with a return time of 10 years for 1920–1950. Can you calculate that for 2070–2100? Why? Answer:

The relevant variables are Tmax_cdf_1920 and Tmax_cdf_1970. The CDF can be used to calculate the return time τ_{return} as follows: In this case the data are daily maximum temperature, given every day. Therefore, the number of events per year that exceed T, and therefore the number of years between such events are,

$$N_{\text{days in year}} = (1 - CDF(T)) \times 365$$
$$\tau_{\text{return}}(T) = 1/N_{\text{days in year}}$$

[ ]:
```
# calculate number of days per year of T_max events:
N_days_in_year_1920=
N_days_in_year_2070=
```

```
# calculate return times:
tau_return_1920=XX
tau_return_2070=XX

# plot:
fig=plt.figure(figsize=(5,3),dpi=300)
plt.ylabel("XX")
plt.xlabel("XX");
plt.bar(Tmax_x,tau_return_1920,color="b");
plt.bar(XX,color="r",alpha=0.5);
plt.gca().set_yscale('log')
plt.xlim(25,48);
```

(e) Optional extra credit: calculate and plot the PDFs of daily maximum temperature, heat wave duration and heat wave amplitudes following the above guidelines in item (b).

[2]:
```
# XX [a fairly complex programming challenge]
```

4. Heat wave composite analysis in the region of your city of birth:

(a) Calculate and plot a climatology of August daily maximum temperature, and of Z500 (height of the 500 hPa surface).

[ ]:
```
# determine region of world to zoom in on (use an area of about 50 by 50 degree)
long_min = XX
long_max = XX
lat_min = XX
lat_max = XX

# printout to see what data look like:
print("date of week 3 data:",heat_waves_weekly_averages_TREFHTMX[3][0])
print("size of week 3 data:",heat_waves_weekly_averages_TREFHTMX[3][1].shape)
N=len(heat_waves_weekly_averages_TREFHTMX) # number of weeks in data set
nx,ny=heat_waves_weekly_averages_TREFHTMX[0][1].shape # size of data for one week
lon=heat_waves_lon
lat=heat_waves_lat

# calculate averages over all weeks (climatology):
TREFHTMX_climatology=np.zeros((nx,ny))
Z500_climatology=np.zeros((nx,ny))
for iweek in range(N):
    TREFHTMX_climatology=XX
    Z500_climatology=XX


# plot climatologies:
# -------------------
projection=ccrs.PlateCarree(central_longitude=0.0);
fig,axes=plt.subplots(1,2,figsize=(7,3),dpi=300,subplot_kw={'projection': projection});

axes[0].set_extent([long_min, long_max, lat_min, lat_max], crs=ccrs.PlateCarree())
axes[0].coastlines(resolution='110m',lw=0.3)
plt.set_cmap('bwr')
DATA=1.0*TREFHTMX_climatology
levels=np.arange(275,320,1)
```

```
c=axes[0].contourf(lon,lat, DATA[:,:],levels=levels)
clb=plt.colorbar(c, shrink=0.5, pad=0.02,ax=axes[0])
clb.set_label('K')
axes[0].set_title('$T_{max}}$ climatology');

axes[1].set_extent([long_min, long_max, lat_min, lat_max], crs=ccrs.PlateCarree())
axes[1].coastlines(resolution='110m',lw=0.3)
plt.set_cmap('bwr')
DATA=1.0*Z500_climatology/1000
levels=np.arange(5.4,6,0.01)
c=axes[1].contourf(lon,lat, DATA[:,:],levels=levels)
clb=plt.colorbar(c, shrink=0.5, pad=0.02,ax=axes[1])
clb.set_label('km')
axes[1].set_title('$Z_{500}}$ climatology');
```

(b) Calculate a time series over the city of interest (averaging over an area of about 5×5 degree).

(c) Use the time series to calculate and plot a composite of both fields plotted in (a) for heat wave events. Define a heat wave as having temperature of more than an appropriate threshold (ignoring the duration condition for simplicity).

[ ]:
```
# first, time series of weekly-averaged daily max August temperatures
# at location of interest, choose an area that is some 10x10 degree large:
Tmax_weekly_timeseries=[]
lat_range=np.logical_and(lat>=XX,lat<=XX)   # latitude range, for northeast: 40,45
lon_range=np.logical_and(lon>=360-XX,lon<=360-XX) # longitude (0-360) range, for␣
 ↪northeast: 360-75,360-70
locations = np.ix_(lat_range,lon_range)
for iweek in range(N):
    Tmax_weekly_timeseries.append(np.mean(
        heat_waves_weekly_averages_TREFHTMX[iweek][1][locations]))

mydates=np.asarray([date[0] for date in heat_waves_weekly_averages_TREFHTMX])
# convert from a list to a numpy array:
Tmax_weekly_timeseries=np.asarray(Tmax_weekly_timeseries)

# choose an appropriate threshold (experiment, based on plot below)
threshold=300.5

# plot time series of daily max temperature over region of interest:
# ---------------------------------------------------------------------
plt.figure(figsize=(5,3),dpi=300)
plt.plot(XX plot the time series XX,".",markersize=3,color='b',lw=0.5,label="TREFHTMX")
plt.plot(XX plot a line at the threshold XX,"-",color='r',lw=0.5,label="threshold")
plt.title("daily max temperature over region of interest")
plt.xlabel("year")
plt.ylabel("TREFHTMX (K)")
plt.legend()
plt.grid(lw=0.25)


# next, calculate a mask time series, with 1 for days with temperature over threshold
# during 1920-1980, 0 otherwise:
```

```
mask=np.zeros(Tmax_weekly_timeseries.shape,dtype=np.int)
mask[XX set threshold condition XX]=1
```

[ ]:
```
# calculate composite:
# --------------------
TREFHTMX_composite=np.zeros((nx,ny))
Z500_composite=np.zeros((nx,ny))
# number of hot days to average over:
Navg=np.nansum(mask)
for iweek in range(N):
    if mask[iweek]==1:
        TREFHTMX_composite=XX
        Z500_composite=XX


# plot composites:
# ----------------
projection=ccrs.PlateCarree(central_longitude=0.0);
fig,axes=plt.subplots(1,2,figsize=(7,3),dpi=300,subplot_kw={'projection': projection});

XX define axes for first plot etc
DATA=1.0*(TREFHTMX_composite-TREFHTMX_climatology)
XX contour

XX second plot
DATA=1.0*(Z500_composite-Z500_climatology)
XX contour
```

5. Decadal heat wave statistics.

(a) Calculate and plot decadal time series of the number of heat waves days over the great plains from 1920 to 2100.

[ ]:
```
Ndecades=int(np.floor((years[-1]-years[0])/10))

# initialize arrays:
Nhot_days_per_year=np.zeros(Ndecades)
year_decade_start=np.zeros(Ndecades)

# calculate for each decade:
for idec in range(Ndecades):
    year_start=years[0]+idec*10
    year_end=XX
    year_decade_start[idec]=year_start+5 # mid of decade for plotting
    # heat wave days per year: create an array Tmax that is 1 if temperature
    # is above 40C, nan otherwise
    # first, initialize Tmax to temperature in Celisus:
    Tmax=1.0*Tmax_all_timeseries[:,np.logical_and(years<XX,years>=XX)]-273.15
    # count number of days above 40C:
    heatwave_threshold=40.0
    heatwave_counter = 0 # set up counter variable to increase whenever T threshold met
    for T in Tmax.flatten(): # flatten makes T_array 1D to iterate over with the for␣
 ↪loop easily
        if XX:
```

```
            heatwave_counter = XX # increase counter by 1 when temperature > threshold
    Nhot_days_per_year[idec]=heatwave_counter
    # normalize by number of ensemble members and number of years per decade:
    Nhot_days_per_year[idec]=Nhot_days_per_year[idec]/len(ensemble_members)/10.0

# plot number of hot days per decade
fig=plt.figure(figsize=(7,3),dpi=300)
h1=plt.plot(XX,XX,color="b");
plt.xlabel("year")
plt.ylabel("# days above 40C per year")
plt.grid(lw=0.25);
plt.tight_layout();
```

(b) ***Optional extra credit:*** **calculate and plot the decadal statistics of the averaged amplitude, duration and frequency (number of events per year) of heat waves for 1920–2100.**

[ ]:
```
# first, calculate a mask timeseries indicating which days are above the temperature␣
  ↪threshold:
Tmax_threshold=40
Ndays_threshold=3
XX
Tmax_mask_days_above_threshold=XX

# next, a mask indicating which days are part of a heat wave event taking into
# account the minimum temperature and minimum duration threshold:
iensemble_member=-1
for ensemble_member in ensemble_members:
    iensemble_member=iensemble_member+1
    XX loop over all days in each ensemble member, see if each day is part of a
    XX    continuous period of days above threhsold temperature:

# now calculate decadal heat wave statistics:
XX

# plot time series of decadal statistics:
# ---------------------------------------

fig=plt.figure(figsize=(6,4),dpi=300)
plt.subplot(2,2,1)
plt.plot(year_decade_start,Ndays_heat_waves_per_year)
plt.ylabel("# days/year")
#plt.xlabel("year");
plt.xlim(1920,2100)
plt.grid(lw=0.25)

plt.subplot(2,2,2)
plt.plot(year_decade_start,avg_Tmax_heat_waves)
plt.ylabel("$\\langle T_{\\rm max}\\rangle$, ($^\circ$C)")
#plt.xlabel("year");
plt.xlim(1920,2100)
plt.grid(lw=0.25)

plt.subplot(2,2,3)
plt.plot(year_decade_start,num_heat_waves_per_year)
```

```
plt.ylabel("# events/year")
plt.xlabel("year");
plt.xlim(1920,2100)
plt.grid(lw=0.25)

plt.subplot(2,2,4)
plt.plot(year_decade_start,avg_duration_heat_waves)
plt.ylabel("averaged duration")
plt.xlabel("year");
plt.xlim(1920,2100)
plt.grid(lw=0.25)

plt.tight_layout()
plt.show();
```

[ ]:

15 Forest fires

Global Warming Science

https://courses.seas.harvard.edu/climate/eli/Courses/EPS101/

Forest Fires!

Please use the template below to answer the workshop questions. "XX" indicates places where you need to complete/write code or add a discussion.

Your name:

[ ]:
```
# Load libraries and data:
import numpy as np
import pickle
import matplotlib.pyplot as plt
from scipy import stats
from scipy.stats import linregress
from scipy.signal import savgol_filter # for smoothing time series
import warnings
import cartopy.crs as ccrs

# load the data from a pickle file:
with open('./forest_fires_variables.pickle', 'rb') as file:
    while 1:
        try:
            d = pickle.load(file)
        except (EOFError):
            break
            # print information about each extracted variable:
        for key in list(d.keys()):
            print("extracting pickled variable: name=", key,  "; size=", d[key].shape)
            #print("type=",type(d[key]))
        globals().update(d)
```

Explanation of input variables:

Canada fires:

canada_fire_area, canada_fire_years, canada_fire_number,

these represent area burnt in km^2 per year, and the number of fires per year, with corresponding time axes.

US fires:

US_fire_area, US_fire_years, US_fire_number,

these represent total US area burnt in km^2 per year, and the number of total US fires per year, with corresponding time axes.

global fires:

global annual burnt area time series in km^2 by type of burnt area. all fire area is the sum of the other components.

global_fire_years, global_crop_fire_area, global_forest_fire_area, global_savanna_fire_area, global_shrub_grass_fire_ar
global_all_fire_area,

Time series for extracting the ACC signal in forest area burnt over Western US:

west_US_VPD, west_US_VPD_NOACC (VPD without anthropogenic climate change), west_US_area_burnt (km^2/year)

VPD is in normalized dimensionless units, see notes.

Time series for analyzing the droughts that led to the 2019-2020 Australian fires:

australia_rain_Murray_Darling (mm/month), australia_rain_Murray_Darling_years, australia_IOD (deg C), australia_IOD_years,

australia_NINO34 (deg C), australia_NINO34_years, australia_SAM (non dimensional), australia_SAM_years

Ensemble time series for extracting ACC signal in surface temperature over the western US, separately from natural variability:

forest_fires_west_US_TS_ensemble_timeseries, forest_fires_west_US_TS_timeseries_ensemble_years

Surface temperature (K) time series averaged over the western US from multiple model ensembles from 1920 to 2000.

Data for calculating El Nino/La Nina SST composites:

SST_4composite, NINO34_timeseries_4composite, lat_4composite, lon_4composite, months_4composite, dates_4composite

1. Observed fire trends: (a) Plot the total burnt area and number of fires in the US as a function of year; (b) same in Candada; (c) burnt area in western US, (d) global burnt area, plotting each of the land types. Superimpose a regression line for each of the plots, discuss your findings.

[ ]:
```
plt.figure(figsize=(6,8),dpi=600)

# plot area burnt in US fires:
plt.subplot(4,1,1)
plt.plot(US_fire_years,US_fire_area/1.e3,'r')
plt.xlabel("year")
plt.ylabel("area burnt US ($10^3$ km$^2$)",color='r')
plt.grid(lw=0.25)
# calculate and plot regression:
x=US_fire_years; y=US_fire_area/1.e3
fit=linregress(x, y); slope=fit.slope; intercept=fit.intercept;
p=fit.pvalue; r2=(fit.rvalue)**2; fitted_line=intercept+slope*x
plt.plot(x,fitted_line,'r')
#plt.title("p=%g, r$^2$=%g" % (p,r2))

# add a plot of the number of US fires on same axes:
ax = plt.gca()
ax2 = ax.twinx()
ax2.plot(XX,XX,'b')
plt.xlabel("year")
ax2.set_ylabel("number of US fires",color='b')
# calculate and plot regression:
XX
ax2.plot(x,fitted_line,'b')
#plt.title("p=%g, r$^2$=%g" % (p,r2))
```

```
# plot area burnt in Canada fires:
plt.subplot(4,1,2)
plt.plot(XX,XX,'r')
plt.xlabel("year")
plt.ylabel("area burnt canada ($10^3$ km$^2$)")
plt.grid(lw=0.25)
# calculate and plot regression:
x=canada_fire_years; y=canada_fire_area/1.e3
fit=linregress(x, y); slope=XX; intercept=XX;
p=XX; r2=XX; fitted_line=XX+XX*x
plt.plot(x,fitted_line)
plt.title("p=%g, r$^2$=%g" % (p,r2))

# add number of Canada fires:
XX

# plot area burnt in western US fires:
plt.subplot(4,1,3)
plt.plot(west_US_years,west_US_area_burnt)
plt.xlabel("year")
plt.ylabel("west US burnt area")
plt.grid(lw=0.25)
# calculate and plot regression:
XX
plt.plot(x,fitted_line)
plt.title("p=%g, r$^2$=%g" % (p,r2))


# plot global fire area time series:
plt.subplot(4,1,4)
plt.plot(global_fire_years,global_forest_fire_area/1.e6,'r',label="Forest")
plt.plot(XX,XX,'g',label="Savanna")
plt.plot(XX,XX,'b',label="Shrub/grass")
plt.plot(XX,XX,'m',label="crop")
plt.plot(XX,XX,'k',label="All")
plt.xlabel("Year")
plt.ylabel("Area burnt ($10^6$ km$^2$)")
plt.legend(ncol=3)
plt.grid(lw=0.25)

# calculate and plot regression:
x=global_fire_years.flatten()
y=global_all_fire_area.flatten()/1.e6
fit=linregress(x, y); slope=fit.slope; intercept=fit.intercept;
p=fit.pvalue; r2=(fit.rvalue)**2; fitted_line=intercept+slope*x
plt.plot(x,fitted_line,'--k',linewidth=1)
plt.title("Global fire area; p=%g, r$^2$=%g" % (p,r2))

plt.tight_layout()
```

Discussion: XX

2. VPD as a fire danger index.

(a) Understanding VPD: Plot the saturation water vapor pressure and the water vapor partial pressure assuming an 80% relative humidity versus temperature. Plot the difference between the two curves (i.e., the VPD for a constant relative humidity) versus temperature.

[ ]:
```
# calculate *dimensional* VPD as function of temperature:
plt.figure(figsize=(7,3),dpi=300)

# plot moisture and saturation moisture vs temperature
plt.subplot(1,2,1)
Trange=np.asarray(np.arange(0,40,1))
# use Clausius-Clapeyron formula at a pressure of 1000 mb to calculate q_sat for␣
 ↪Trange:
Saturation_vapor_pressure=XX
# calculate the vapor pressure assuming relative humidity of 80%:
vapor_pressure_RH80=XX
# plot q^*(T) and q at RH=0.8:
plt.plot(Trange,XX,label="q^*(T)")
plt.plot(Trange,XX,label="q(T)=RH*q^*(T)")
plt.xlabel("Temperature")
plt.ylabel("Moisture measure")

# plot VPD (difference between moisture and saturation moisture) vs temperature
# VPD for Trange and RH=80%:
VPD=XX
plt.subplot(1,2,2)
plt.plot(XX,XX)
plt.grid(lw=0.25)
plt.xlabel("T (C)")
plt.ylabel("VPD (mb)")

plt.tight_layout();
```

(b) Plot the western US VPD and area burnt versus time on the same axes. Then repeat using VPD and the log10 of area burnt. Calculate the correlation coefficient between the two plotted time series for each of the two cases.

[ ]:
```
# plot VPD given in input in nondimensional normalized units:

fig=plt.figure(figsize=(5,4),dpi=300)

plt.subplot(1,2,1)
# plot burnt area in 1000 of km^2 and VPD:
plt.plot(XX,XX,lw=1,color="r",label="area burnt")
ax=plt.gca();
plt.xlabel("XX")
plt.ylabel("area burnt (km$^2\\times1000$)",color="r")
plt.grid(lw=0.25)
# plot VPD on same axes:
ax1=plt.gca()
ax2 = ax1.twinx()
ax2.plot(XX,XX,color="b")
ax2.set_ylabel('XX', color='b')
print("correlation (VPD,area)=",np.corrcoef(west_US_VPD,west_US_area_burnt)[0,1])
```

```
plt.subplot(1,2,2)
# plot log10 burnt area in 1000 of km^2 and VPD:
XX
print("correlation (VPD,log10(area))=",XX)

plt.tight_layout();
```

3. Separating ACC from variability: Extract the ACC signal in temperature from an ensemble of model runs for 1920–2020. Plot all ensemble time series and superimpose the estimated ACC signal.

[ ]:
```
# calc avereage over all ensemble members:
TS_timeseries=1.0*forest_fires_west_US_TS_ensemble_timeseries
dates=1.0*forest_fires_west_US_TS_timeseries_ensemble_years
TS_timeseries_ensemble_avg=XX

# plot:
fig=plt.figure(dpi=300,figsize=(6,4))
plt.clf()
for ensemble_member in range(len(TS_timeseries[:,0])):
    # plot all ensemble averages with thin transparent lines (alpha parameter)
    plt.plot(XX,XX,lw=0.25,alpha=0.4)
TS_timeseries_avg_smooth = savgol_filter(XX) # check python docs for how to use filter
plt.plot(XX,XX,label="ensemble mean")
plt.plot(XX,XX,label="ensemble mean, smoothed")
plt.xlabel("year")
plt.ylabel("Surface Temperature")
plt.legend()

plt.tight_layout()
plt.show();
```

4. Estimate the contribution of ACC to western US forest fire area:

following Abatzoglou and Williams (2016):

(a) Scatter plot area burnt vs VPD, calculate and plot a regression line, calculate the r^2 .

(b) Given the VPD increase due to ACC and using your calculated regression, calculate area that would have burned without ACC, and then calculate and plot the contribution of ACC as function of time.

[ ]:
```
warnings.filterwarnings("ignore", message="invalid value encountered in less")

# calculate the effects of ACC on VPD:
west_US_VPD_ACC=west_US_VPD-west_US_VPD_NOACC
# Fit a regression line to the VPD-log10(burnt area) relation:
# -------------------------------------------------------
x=XX; y=XX;
fit=linregress(x, y)
slope=fit.slope;
intercept=fit.intercept;
```

```
p=fit.pvalue;
r2=(fit.rvalue)**2
print("p,r2=",p,r2)


# Calculate the effect of ACC VPD on fires using the calculated regression:
# ------------------------------------------------------------------------
west_US_burnt_area_due_to_ACC=10**(XX)
# burnt area without effect of ACC:
west_US_area_burnt_NOACC=XX
# set meaningless negaative area to zero:
west_US_area_burnt_NOACC[west_US_area_burnt_NOACC<0]=0
years=west_US_years

fig=plt.figure(figsize=(9,2.3),dpi=200)

# ----------------
# plot burnt area: in 1000 of km^2
# ----------------
plt.subplot(1,3,1)
plt.plot(XX,XX,label="area burnt")
plt.plot(XX,XX,label="due to ACC")
plt.plot(XX,XX,label="no ACC")
plt.xlabel("year")
plt.ylabel("area burnt (km$^2\\times1000$)",color="r")
plt.legend()
plt.grid(lw=0.25)


# ----------------
# plot VPD:
# ----------------
plt.subplot(1,3,2)
plt.plot(XX,XX,label="VPD")
plt.plot(XX,XX,label="due to ACC")
plt.plot(XX,XX,label="without ACC")
plt.xlabel("year")
plt.ylabel("VPD",color="b")
plt.legend()
plt.grid(lw=0.25)


# ---------------------------------
# plot VPD to burnt area regression:
# ---------------------------------
plt.subplot(1,3,3)
# scatter plot of VPD vs log10(area burnt):
plt.scatter(XX,XX)
# plot the fitted regression line:
plt.plot(XX,XX)
plt.xlabel("VPD",color="b")
plt.ylabel("log$_{10}$(area burnt)",color="r")
```

```
# finalize plot:
plt.tight_layout()
plt.show()
```

(c) What percent of area burnt over the last 15 years of data is attributable to ACC?

[ ]:
```
# total area burnt since 2000:
total_burnt=XX sum over area burnt over years after 2000
total_burnt_ACC=XX sum over area burnt due to ACC over years after 2000
print("total area burnt since 2000=",total_burnt,"km^2*1000")
print("total area burnt since 2000 due to ACC=",total_burnt_ACC,"km^2*1000")
print("percent area burnt attributable to ACC=",100*total_burnt_ACC/total_burnt,"%")
print("total forested area in the US according to Wikipedia: XX km^2*1000")
```

(d) Crudely estimate the change in normalized VPD due to a 2 °C warming by examining the provided time series of normalized VPD due to ACC and the western US temperature record in Figure 14.2. XX

(e) Given the linear dependence of log of the area burnt on normalized VPD, how much of the western US forest area (in both 10^3 km^2 and in percent of the total current forest area) might burn per year if the temperature increases by two more degree Celsius? XX

5. Role of variability modes: Calculate the averaged SAM/NINO3.4/IOD indices for rainy vs dry years (defined to be above and below one standard deviation, respectively) in south-east Australia, compare to the average over years 2018–9.

[this follows King et al. 2020]

[ ]:
```
# calculate std and mean of the four Australia time series (use np.mean and np.std):
rain_std=XX
rain_mean=XX
SAM_std=XX
SAM_mean=XX
IOD_std=XX
IOD_mean=XX
NINO34_std=XX
NINO34_mean=XX

print("rain_mean, rain_std=", rain_mean, rain_std)
print("SAM_mean, SAM_std=", SAM_mean, SAM_std)
print("IOD_mean, IOD_std=", IOD_mean, IOD_std)
print("NINO34_mean, NINO34_std=", NINO34_mean, NINO34_std)

print("\nrainy averages: averaged over rain>mean+std")
IOD_rainy_avg=np.mean(australia_IOD[australia_rain_Murray_Darling>rain_mean+rain_std])
print("IOD_rainy_avg=",IOD_rainy_avg)

SAM_rainy_avg=XX
print("SAM_rainy_avg=",SAM_rainy_avg)

NINO34_rainy_avg=XX
print("NINO34_rainy_avg=",NINO34_rainy_avg)
```

```
print("\ndry averages: averaged over rain<mean-std")
XX


# similarly calculate average over entire period of data.

print("\naverages for 2018-2019:")
print("rain:",XX)
print("SAM:",XX)
print("IOD:",XX)
print("NINO34:",XX)
```

6. Optional extra credit: ENSO SST composites: Calculate El Niño and La Niña SST composites by removing the monthly averaged SST and then averaging over months in which the NINO3.4 time series is 1.5 std above or below its mean.

[ ]:
```
# ------------------------------------------------------------------------
# (1) prepare mask time series for calculating composites:
# El Nino composite mask: set to 1 if NINO34>mean+1.5*std, NaN otherwise
# La nina composite mask: set to 1 if NINO34<mean-1.5*std, NaN otherwise
# ------------------------------------------------------------------------


# ---------------------------------------------------------------
# (2) plot mask time series super imposed on NINO34 time series:
# ---------------------------------------------------------------


# ------------------------------------------------------------------
# (3) calculate monthly climatology of SST, remove it from SST data:
# ------------------------------------------------------------------


# ------------------------------------------------------------------------
# (4) use mask timeseries to calculate SST composites for El Nino, Lanina:
# ------------------------------------------------------------------------

# ------------------------------------------------
# (5) contour composites for El Nino and La Nina:
# ------------------------------------------------

# add a column to lon/lat arrays to eliminate white gap at dateline:
# for atmospheric plots:
lon1=1.0*lon[-1]+lon[2]-lon[1]; lon1=np.hstack((lon,lon1))

# initialize figure:
plt.clf();
projection=ccrs.PlateCarree(central_longitude=0.0);
fig=plt.figure(figsize=(9,6),dpi=300);
grid = plt.GridSpec(1, 2)#, wspace=0.4, hspace=0.3)
shrink=0.53
```

```
# El Nino:
# --------
axes=fig.add_subplot(1,2,1, projection=ccrs.PlateCarree(180))
axes.set_extent([-240, -60, -40, 40], crs=ccrs.PlateCarree())
axes.coastlines(resolution='110m',lw=0.3)
axes.stock_img()
plt.set_cmap('bwr')
DATA=SST_composite_ElNino
c=axes.pcolormesh(lon+179,lat-1, DATA,vmin=-3,vmax=3)
clb=plt.colorbar(c, shrink=shrink, pad=0.02,ax=axes)
#clb.set_label('°C')
axes.set_title('El Niño')


# La Nina:
# --------
axes=fig.add_subplot(1,2,2, projection=ccrs.PlateCarree(180))#grid[0, 0])
axes.set_extent([-240, -60, -40, 40], crs=ccrs.PlateCarree())
axes.coastlines(resolution='110m',lw=0.3)
axes.stock_img()
plt.set_cmap('bwr')
DATA=1.0*SST_composite_LaNina
c=axes.pcolormesh(lon+179,lat-1, DATA,vmin=-3,vmax=3)
clb=plt.colorbar(c, shrink=shrink, pad=0.02,ax=axes)
#clb.set_label('°C')
axes.set_title('La Niña')

# finalize and show plot:
plt.subplots_adjust(top=0.92, bottom=0.08, left=0.01, right=0.95, hspace=-0.
  ↪3,wspace=-0.01);
plt.show();
```

[ ]: