2_compute_delta_T

February 14, 2020

1 Compute change in temperature since 1850 (Δ T) from RCMIP Effective radiative forcings (ERFs)

1.1 General about computing ΔT:

We compute the change in GSAT temperature (ΔT) from the effective radiative forcing (ERF) estimated from the RCMIP models (Nicholls et al 2020), by integrating with the impulse response function (IRF(t-t')) (Geoffroy at al 2013). See Nicholls et al (2020) for description of the RCMIP models and output.

For any forcing agent x, with estimated ERF$_x$, the change in temperature ΔT is calculated as:

$$\Delta T_x(t) = \int_0^t ERF_x(t')IRF(t-t')dt'$$

1.1.1 The Impulse response function (IRF):

In these calculations we use the impulse response function (Geoffroy et al 2013):

$$\begin{aligned}\text{IRF}(t) =& 0.885 \cdot (\frac{0.587}{4.1} \cdot exp(\frac{-t}{4.1}) + \frac{0.413}{249} \cdot exp(\frac{-t}{249}))\\ \text{IRF}(t) =& \frac{1}{\lambda}\sum_{i=1}^{2}\frac{a_i}{\tau_i} \cdot exp(\frac{-t}{\tau_i})\end{aligned}$$

with $\frac{1}{\lambda} = 0.885$ (K/Wm^{-2}), $a_1 = 0.587$, $\tau_1 = 4.1$(yr), $a_2 = 0.413$ and $\tau_2 = 249$(yr) (note that $i = 1$ is the fast response and $i = 2$ is the slow response and that $a_1 + a_2 = 1$)

1.2 Input data:

See README.md

1.3 Set values:

ECS parameter:

[6]:
```
csf = 0.884
```

Year to integrate from (i.e. reference for ΔT) and to:

[6]:
```
first_y ='1850'
last_y = '2100'
```

1.4 Code + figures

1.4.1 Imports:

[3]:
```
import xarray as xr
from IPython.display import clear_output
import numpy as np
import os
import re
from pathlib import Path
import pandas as pd
import tqdm
from scmdata import df_append, ScmDataFrame
import matplotlib.pyplot as plt

%load_ext autoreload
%autoreload 2
```

```
<IPython.core.display.Javascript object>
```

```
pyam - INFO: Running in a notebook, setting `pyam` logging level to
`logging.INFO` and adding stderr handler
```

[4]:
```
from ar6_ch6_rcmipfigs.constants import BASE_DIR
from ar6_ch6_rcmipfigs.constants import OUTPUT_DATA_DIR, INPUT_DATA_DIR

PATH_DATASET = OUTPUT_DATA_DIR + '/forcing_data_rcmip_models.nc'
PATH_DT_OUTPUT = OUTPUT_DATA_DIR + '/dT_data_rcmip_models.nc'
```

```
/home/sarambl/PHD/IPCC/public/AR6_CH6_RCMIPFIGS/ar6_ch6_rcmipfigs
/home/sarambl/PHD/IPCC/public/AR6_CH6_RCMIPFIGS/ar6_ch6_rcmipfigs/data_in
```

[5]:
```
climatemodel = 'climatemodel'
scenario = 'scenario'
variable = 'variable'
time = 'time'
```

1.4.2 IRF:

[7]:
```
def IRF(t, l=0.885, alpha1=0.587 / 4.1, alpha2=0.413 / 249, tau1=4.1, tau2=249):
    """
    Returns the IRF function for:
    :param t: Time in years
    :param l: climate sensitivity factor
    :param alpha1:
    :param alpha2:
    :param tau1:
    :param tau2:
    :return:
    IRF
    """
    return l * (alpha1 * np.exp(-t / tau1) + alpha2 * np.exp(-t / tau2))
```

1.4.3 ERF:

Read ERF from file

Define variables to look at:

[8]:
```
# variables to plot:
variables_erf_comp = [
    'Effective Radiative Forcing|Anthropogenic|CH4',
    'Effective Radiative Forcing|Anthropogenic|Aerosols',
    'Effective Radiative Forcing|Anthropogenic|Tropospheric Ozone',
    'Effective Radiative Forcing|Anthropogenic|F-Gases|HFC',
    'Effective Radiative Forcing|Anthropogenic|Other|BC on Snow']
# total ERFs for anthropogenic and total:
variables_erf_tot = ['Effective Radiative Forcing|Anthropogenic',
                     'Effective Radiative Forcing']
# Scenarios to plot:
scenarios_fl = ['ssp119', 'ssp126', 'ssp245', 'ssp370',␣
 ↪'ssp370-lowNTCF-aerchemmip',
                'ssp370-lowNTCF-gidden',
                # 'ssp370-lowNTCF', Due to mistake here
                'ssp585', 'historical']
```

Open dataset:

[9]:
```
ds = xr.open_dataset(PATH_DATASET)
```

1.5 Integrate:

The code below integrates the read in ERFs with the pre defined impulse response function (IRF).

$$\Delta T(t) = \int_0^t ERF(t')IRF(t-t')dt'$$

[15]:
```
from ar6_ch6_rcmipfigs.utils.misc_func import new_varname
```

[16]:
```
# Name of equivalent delta T to ERF
name_deltaT = 'Delta T'



def integrate_(i, var, nvar, ds, ds_DT, csfac=0.885):
    """

    Parameters
    ----------
    i:int
        the index for the integral
    var:str
        the name of the EFR variables to integrate
    nvar:str
        the name of output integrated value

    ds:xr.Dataset
        the ds with the intput data
    ds_DT: xr.Dataset
        the ouptut ds with the integrated results
    csfac: climate sensitivity factor (for IRF)
    Returns
    -------
    None

    """
    # lets create a ds that goes from 0 to i inclusive
    ds_short = ds[{'time': slice(0, i + 1)}].copy()
    # lets get the current year
    current_year = ds_short['time'][{'time': i}].dt.year
    # lets get a list of years
    years = ds_short['time'].dt.year
    # lets get the year delta until current year(i)
    ds_short['end_year_delta'] = current_year - years

    # lets get the irf values from 0 until i
```

```
    ds_short['irf'] = IRF(
        ds_short['end_year_delta'] * ds_short['delta_t'], l=csfac
    )

    # lets do the famous integral
    ds_short['to_integrate'] = \
        ds_short[var] * \
        ds_short['irf'] * \
        ds_short['delta_t']

    # lets sum all the values up until i and set
    # this value at ds_DT
    # If whole array is null, set value to nan
    if np.all(ds_short['to_integrate'].isnull()):  # or last_null:
        _val = np.nan
    else:
        #

        _ds_int = ds_short['to_integrate'].sum(['time'])
        # mask where last value is null (in order to not get intgral
        _ds_m1 = ds_short['to_integrate'].isel(time=-1)
        # where no forcing data)
        _val = _ds_int.where(_ds_m1.notnull())
    # set value in dataframe:
    ds_DT[nvar][{'time': i}] = _val


def integrate_to_dT(ds, from_t, to_t, variables, csfac=0.885):
    """
    Integrate forcing to temperature change.

    :param ds: dataset containing the focings
    :param from_t: start time
    :param to_t: end time
    :param variables: variables to integrate
    :param csfac: climate sensitivity factor
    :return:
    """
    # slice dataset
    ds_sl = ds.sel(time=slice(from_t, to_t))
    len_time = len(ds_sl['time'])
    # lets create a result DS
    ds_DT = ds_sl.copy()

    # lets define the vars of the ds
    vars = variables  # variables_erf_comp+ variables_erf_tot #['EFR']
    for var in variables:
```

```
        namevar = new_varname(var, name_deltaT)
        # set all values to zero for results dataarray:
        ds_DT[namevar] = ds_DT[var] * 0
        # Units Kelvin:
        ds_DT[namevar].attrs['unit'] = 'K'
        if 'unit' in ds_DT[namevar].coords:
            ds_DT[namevar].coords['unit'] = 'K'

    for i in range(len_time):
        # da = ds[var]
        if (i % 20) == 0:
            print('%s of %s done' % (i, len_time))
        for var in variables:
            namevar = new_varname(var, name_deltaT)  # 'Delta T|' + '|'.
 ↪join(var.split('|')[1:])

            # print(var)
            integrate_(i, var, namevar, ds_sl, ds_DT, csfac=csfac)
    clear_output()

    fname = 'DT_%s-%s.nc' % (from_t, to_t)
    # save dataset.
    ds_DT.to_netcdf(fname)
    return ds_DT
```

[12]:

```
_vars = variables_erf_comp + variables_erf_tot
ds_DT = integrate_to_dT(ds, first_y, last_y, _vars, csfac=csf)
# list of computed delta T variables:
variables_dt_comp = [new_varname(var, name_deltaT) for var in␣
 ↪variables_erf_comp]
```

[13]:

```
ds_DT
```

[13]:

```
<xarray.Dataset>
Dimensions:                                                  (climatemodel:
5, scenario: 8, time: 251)
Coordinates:
  * time                                                     (time)
datetime64[ns] 1850-01-01 … 2100-01-01
    model                                                    object …
  * scenario                                                 (scenario)
object 'historical' … 'ssp585'
    region                                                   object …
    unit                                                     object …
  * climatemodel                                             (climatemodel)
object 'Cicero-SCM' … 'OSCARv3.0'
    unit_context                                             object …
```

```
Data variables:
    Effective Radiative Forcing                                   (scenario,
climatemodel, time) float64 0.1519 … 8.846
    Effective Radiative Forcing|Anthropogenic                     (scenario,
climatemodel, time) float64 0.1465 … 8.808
    Effective Radiative Forcing|Anthropogenic|Other|BC on Snow    (scenario,
climatemodel, time) float64 nan … 0.143
    Effective Radiative Forcing|Anthropogenic|F-Gases|HFC         (scenario,
climatemodel, time) float64 2.505e-08 … 0.4356
    Effective Radiative Forcing|Anthropogenic|Tropospheric Ozone  (scenario,
climatemodel, time) float64 0.0222 … 0.5016
    Effective Radiative Forcing|Anthropogenic|Aerosols            (scenario,
climatemodel, time) float64 -0.03511 … -1.069
    Effective Radiative Forcing|Anthropogenic|CH4                 (scenario,
climatemodel, time) float64 0.05507 … 0.7203
    year                                                          (time) int64
…
    month                                                         (time) int64
…
    day                                                           (time) int64
…
    delta_t                                                       (time) float64
…
    Delta T|Anthropogenic|CH4                                     (scenario,
climatemodel, time) float64 0.007051 … 0.5399
    Delta T|Anthropogenic|Aerosols                                (scenario,
climatemodel, time) float64 -0.004495 … -0.8081
    Delta T|Anthropogenic|Tropospheric Ozone                      (scenario,
climatemodel, time) float64 0.002843 … 0.3977
    Delta T|Anthropogenic|F-Gases|HFC                             (scenario,
climatemodel, time) float64 3.207e-09 … 0.2765
    Delta T|Anthropogenic|Other|BC on Snow                        (scenario,
climatemodel, time) float64 nan … 0.1044
    Delta T|Anthropogenic                                         (scenario,
climatemodel, time) float64 0.01875 … 5.735
    Delta T|                                                      (scenario,
climatemodel, time) float64 0.01944 … 5.745
```

1.6 Save dataset to netCDF:

[14]:
```
ds_DT.to_netcdf(PATH_DT_OUTPUT)
```