# Documentation for TEF python function

Marvin Lorenz

February 1, 2018

## 1 Introduction

This function calculates uses the Total Exchange Flow theory (TEF) as proposed by MacCready (2011). It returns properties like transports, salinity fluxes, salinities of an arbitrary cross section which characterizes the exchange flow.

## 2 Formulas behind this

The isohaline transport function $Q(s)$ defined as the following:

$$Q(s) \equiv \Big\langle \int_{A_s} u(x,t)\, dA \Big\rangle, \tag{1}$$

with $u(x,t)$ being the velocity perpendicular to the cross section at position $x$ and time $t$, $A_s$ the part of the cross section with salinities greater than $s$ and $\langle\rangle$ denote a time average. The TEF is defined by the following integrals:

$$Q_{in} \equiv \int -\frac{\partial Q}{\partial s}\Big|_{in} ds, \qquad Q_{out} \equiv \int -\frac{\partial Q}{\partial s}\Big|_{out} ds, \tag{2}$$

with 'in' meaning one only counts $-\frac{\partial Q}{\partial s}$ if it brings water into the estuary and 'out', respectively. The salt flux is therefore given by:

$$F_{in} \equiv \int s\Big( -\frac{\partial Q}{\partial s}\Big)\Big|_{in} ds, \qquad F_{out} \equiv \int s\Big( -\frac{\partial Q}{\partial s}\Big)\Big|_{out} ds. \tag{3}$$

The flux-weighted salinities are defined as the following:

$$s_{in} \equiv \frac{F_{in}}{Q_{in}}, \qquad s_{out} \equiv \frac{F_{out}}{Q_{out}}. \tag{4}$$

Similar to the salt flux one can compute the heat exchange by:

$$q_{in} \equiv \int c_v \rho(s) T(s)\Big( -\frac{\partial Q}{\partial s}\Big)\Big|_{in} ds, \qquad q_{out} \equiv \int c_v \rho(s) T(s)\Big( -\frac{\partial Q}{\partial s}\Big)\Big|_{out} ds. \tag{5}$$

with $\rho(s)$ is the density of salinity class $s$, $T(s)$ the temperature of salinity class $s$ and $c_v = 4182$ J kg$^{-1}$K $^{-1}$ the specific heat content of water.
For further information please read the paper:
MacCready, Parker. "Calculating estuarine exchange flow using isohaline coordinates." Journal of Physical Oceanography 41.6 (2011): 1116-1124.

## 3 Installation

Simply unzip the zip file and move it to your python-packages folder. You can also use pip install by navigating into the most outer TEF folder and type (linux):

```
pip install .
```

for a local installation. If you don't want to do either of those you can put the unzipped folder anywhere and add the path to that folder to your PYTHONPATH in your program temporarily:

```
import sys

sys.path.append('yourpath')
```

You should then be able to use the package:

```
from tef import *
```

## 4 The python functions

### 4.1 TEF_salt(ncfile,t,lat_or_lon,lat_or_lon_index,i_start,i_end,s, outputtype='normal',exchange_out=True,time_mean='no',land_depth=-10, variable_list=[ ], heat_exchange=False)

This function is programmed to compute the $Q(s)$, $-\frac{\partial Q}{\partial s}$, $Q_{in}$, $Q_{out}$, $F_{in}$, $F_{out}$, $s_{in}$ and $s_{out}$ based on GETM output. The GETM variables needed are:

- salinity, "salt"[time,depth,lat,lon]
- layer height, "h"[time,depth,lat,lon]
- velocity, "uu"[time,depth,lat,lon] and/or "vv"[time,depth,lat,lon]
- bathymetry, "bathymetry"[lat,lon]
- distance between 2 c gridpoints, "dxc"[lat] and/or "dyc"[lat]. If these variables are not stored in your file, you can give the longitudes or latitudes. These are then used to compute the distance using the haversine function (see section 4.2) . Note that an equidistant grid is assumed.
- distance, "distance" if you put in a transect from nctransect

You can give a custom list with variable names, which are specifying the variable names of the provided data with variable_list. The order has to be: 'salinity', 'layer height', 'velocity', 'distance between 2 T points' for lat or lon constant. 'salinity', layer height', u-velocity', 'v-velocity' for transect.
If you want to calculate the heat exchange, further variables are needed:

- temperature, 'temp'[time,depth,lat,lon]
- density, 'sigma_t'[time,depth,lat,lon]

#### 4.1.1 Input

**ncfile**: Input netcdf file where all needed variables are stored

**t**: Timeindex for which TEF should be calculated. This could be either:

- an integer i: Then only for this one timeindex TEF is claculated (Note in python indices start from 0!)
- a tuple [i,j]: Then slicing for time is done by with [i:j+1]
- 'all': if you give the string 'all' then TEF is calculated for all timesteps

**lat_or_lon**: Here you decide if you want to keep the latitude or longitude constant or if the input is an arbitrary transect by nctransect:

- 'lat': latitude is kept constant

- 'lon': longitude is kept constant
- 'transect': a transect is given

**lat_or_lon_index**: Here you give the index for the latitude or longitude. If you give choose transect for **lat_or_lon** this variable can be any integer. It is not used then, but must be given since it is no optional variable.

**i_start**, **i_end**: Starting and ending index; slicing done with [i_start:i_end+1]

**s**: This should be an array with salinities in g/kg which is used to compute $Q(s)$.

optional **outputtype**: This specifies the outputtype of GETM:

- 'normal' (default): variables like "salt" and "uu" are used
- 'mean': the netcdf file contains GETM mean values like: "saltmean", "uumean"

optional: **exchange_out**: Returns $Q_{in}$ etc, default is True

optional: **time_mean**: You can chose here if you want to calculate time averages by giving an integer of over how many time indices should be averaged, not implemented yet. Default is 'no' which means False

optional **land_depth**: Specify the depth of land points of the bathymetry array. Default is -10 which is the depth of land points in GETM

optional **variable_list**: list of variable names of the provided data with variable_list if needed. The order has to be: 'salinity', 'layer height', 'velocity', 'distance between 2 T points'.

- optional **heat_exchange**: Calculations of heat/energy $q$[J/s], default is False, If you set this true and use variable_list, you have to extend the list by temperature and density (in that order!). Note that exchange_out has to be set to True to get the heat_exchange output!

### 4.1.2 Output

The function's output is:

$Q(s)$ [m$^3$s$^{-1}$]

$-\frac{\partial Q}{\partial s}$ [m$^3$ s$^{-1}$ (g/kg)$^{-1}$]

$s_{new}$ [g/kg]

$Q_{in}$ [m$^3$s$^{-1}$]

$Q_{out}$ [m$^3$s$^{-1}$]

$F_{in}$ [m$^3$s$^{-1}$ g/kg]

$F_{out}$ [m$^3$s$^{-1}$ g/kg]

$s_{in}$ [g/kg]

$s_{out}$ [g/kg]

$q_{in}$ [J/s]

$q_{out}$ [J/s]

time_indices: gives back the time indices of the last used timestep for each integration periods

A python line to get the output would look like the following:

Qiso [m$^3$s$^{-1}$], dqds [m$^3$ s$^{-1}$ (g/kg)$^{-1}$], snew [g/kg], exchangeproperties, time_indices = TEF_salt(...)

with the dimension of Qiso would be [time,salt] or [salt] if t is an integer, for dqds [time, snew] where snew is a basically s[:-1], and exchangeproperties is a list which looks like:

> exchangeproperties = [Qin[time],Qout[time],Fin[time],Fout[time],sin[time],sout[time],qin[time],qout[time
> qout[time]]

### 4.1.3 Important notes

- Necessary python packages: netCDF4, numpy, sys
- If you chose a transect there is a routine in the code which rotates the velocities into the transects coordinates system. I am not to sure if this is actually 100 % correct
- The time average is done before calculating the TEF profiles, not afterwards. The idea is to use this time average to do an average over a tidal period before computing the TEF profiles.
- If the length of the time axis divided by time_mean is not an integer, it only integrates parts where is has time_mean values.
- you should check for your problem if the _in properties are actually describing the inflowing part and _out respectively. You can check if you plot $-\frac{\partial Q}{\partial s}$. There everything that is positive is added to $Q_{in}$ and all negative to $Q_{out}$.
- There is a check if there is land in the transect or slice you provide. If that is the case, the programs terminates

### 4.1.4 Examples

I created a salinity array by:

> salt_array = np.arange(36,41,0.1)

and provided the data by:

> data = netCDF4.Dataset(path_to_file, 'r')

Then I used the TEF_salt function to analyse a subdomain of my Persian Gulf setup which is stored in the variable data. I want to look at the time index 5, constant latitude with index 0, from longitude index 14 to 30:

> Q_iso, dqds, s_new, Ex_prop = TEF_salt(data,5,'lat',0,14,30,salt_array)

If one now plots Q_iso ($Q_s$) and dqds ($-\frac{\partial Q}{\partial s}$), as seen in in figure 1, one can see an inflowing part in the low salinity classes and an outflow in the high salinity classes. The exchange properties are:

$$\begin{aligned}
Q_{in} &= 3.38 \cdot 10^4 \mathrm{m}^3\mathrm{s}^{-1} \\
Q_{out} &= -1.15 \cdot 10^5 \mathrm{m}^3\mathrm{s}^{-1} \\
F_{in} &= 1.25 \cdot 10^6 \mathrm{m}^3\mathrm{s}^{-1}\mathrm{g\ kg}^{-1} \\
F_{out} &= -4.46 \cdot 10^6 \mathrm{m}^3\mathrm{s}^{-1}\mathrm{g\ kg}^{-1} \\
s_{in} &= 36.99\mathrm{g\ kg}^{-1} \\
s_{out} &= 38.89\mathrm{g\ kg}^{-1}
\end{aligned}$$

Now lets use the time_average over all time steps which is in my case a month:

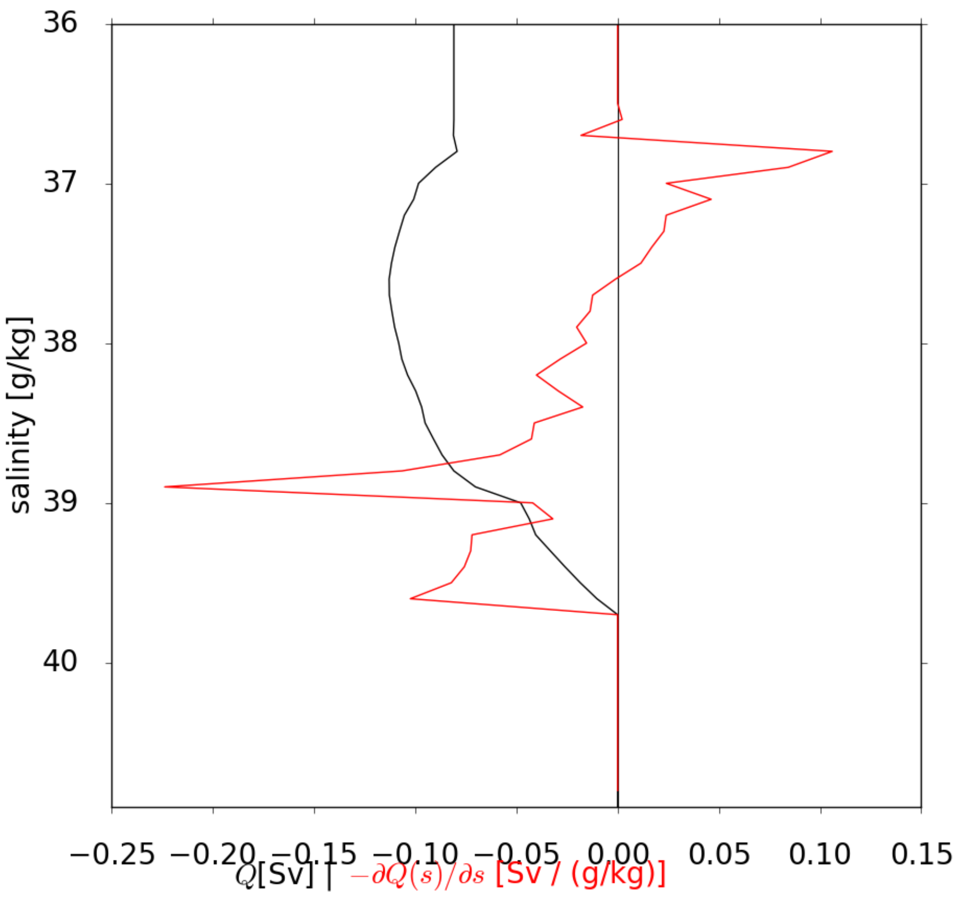

Figure 1: Example for $Q(s)$ and $-\frac{\partial Q}{\partial s}$

Q_iso, dqds, s_new, Ex_prop, time_indices = TEF_salt(data,'all','lat',0,14,30,salt_array, exchange_out=True,time_mean='all')

Note that we have one variable more on the left side for the time_indices. This leads to following numbers:

$$
\begin{aligned}
Q_{in} &= 1.21 \cdot 10^5 \mathrm{m}^3\mathrm{s}^{-1}\\
Q_{out} &= -1.04 \cdot 10^5 \mathrm{m}^3\mathrm{s}^{-1}\\
F_{in} &= 4.53 \cdot 10^6 \mathrm{m}^3\mathrm{s}^{-1}\mathrm{g\ kg}^{-1}\\
F_{out} &= -4.10 \cdot 10^6 \mathrm{m}^3\mathrm{s}^{-1}\mathrm{g\ kg}^{-1}\\
s_{in} &= 37.26 \mathrm{g\ kg}^{-1}\\
s_{out} &= 39.27 \mathrm{g\ kg}^{-1}
\end{aligned}
$$

and the plot in figure 2.
Note that when you use the time_mean = 'all' you don't have the potential to do any statistics. If you want to do statistics I would suggest to not do any averaging inside this function but outside by yourself.
Example for the use of variable_list: First we define a list with strings for the variables names in the file in the right order:

liste = ['salt','hn','vely3d','lonc']

The line the would then look similar to this:

Q_iso, dqds, s_new, Ex_prop, time_indices = TEF_salt(data,'all','lat',0,72,143,salt_array, exchange_out= True,time_mean='all',variable_list = liste)

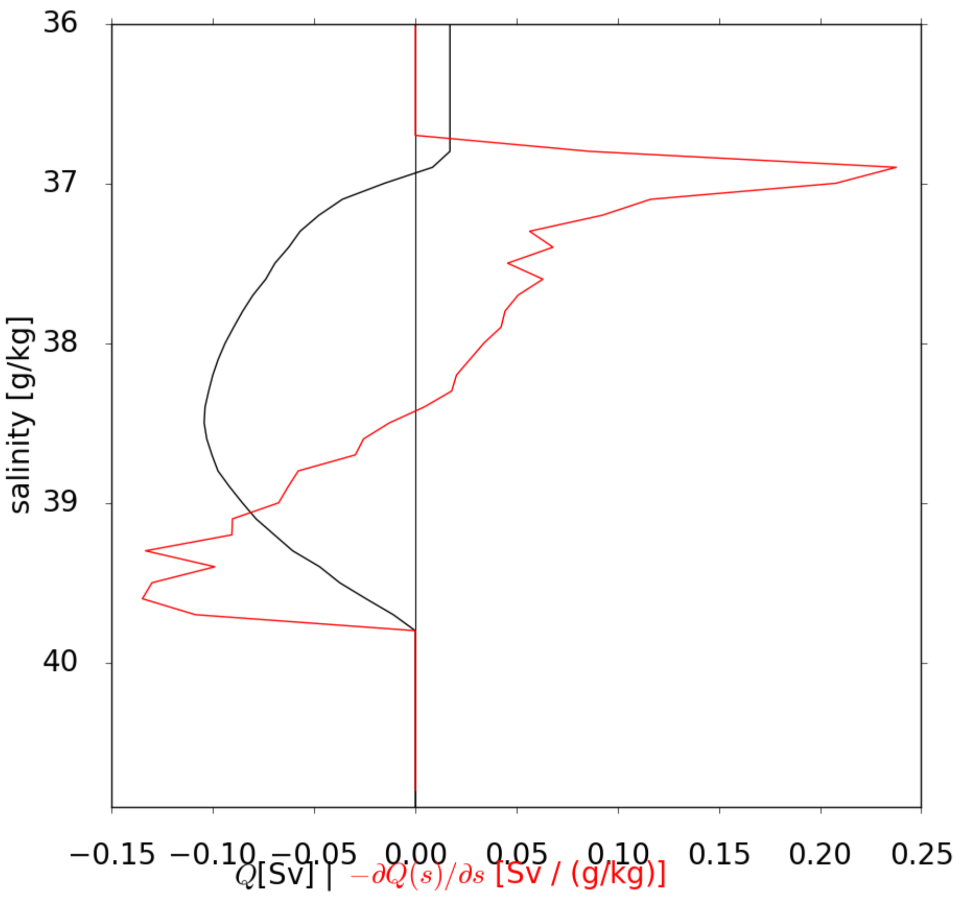

Figure 2: Example for $Q(s)$ and $-\frac{\partial Q}{\partial s}$ with time average.

Since the fourth variable is 'lonc' the Haversine fuction is used to calculate the distance between two grid points.
Example for a transect: Now lets test a transect. The transect is a cross section close to the Strait of Hormuz of my Persian Gulf setup. The line looks like this:

Q_iso, dqds, s_new, Ex_prop, time_indices = TEF_salt(data,'all','transect',0,2,22,salt_array, exchange_out=True,time_mean='all',outputtype='mean')

I had to change the outputtype to 'mean' and change i_start and i_end so that no land points were in the data. The results (without salinity fluxes) are:

$$Q_{in} = 1.62 \cdot 10^5 \text{m}^3\text{s}^{-1}$$
$$Q_{out} = -1.47 \cdot 10^5 \text{m}^3\text{s}^{-1}$$
$$s_{in} = 37.56 \text{g kg}^{-1}$$
$$s_{out} = 39.87 \text{g kg}^{-1}$$

These results are monthly means for August. The sum of $Q_{in}$ and $Q_{out}$ is 15000 $\text{m}^3\text{s}^{-1}$ which should be the amount of water that has to be imported to equalize the volume loss due to evaporation of the Persian Gulf (assuming no net volume loss).
Example of the heat exchange: Once again I take the subdomain from the first example but add the parameter heat_exchange=True. In addition I use a custom variable list with 'lonc' to test the Haversine function:

liste = ['salt','h','vv','lonc','temp','sigma_t']

Q_iso, dqds, s_new, Ex_prop, time_indices = TEF_salt(data,'all','lat',0,14,30,salt_array, time_mean='all',heat_exchange=True,variable_list = liste)

We find the same results as before but in Ex_prop are two more variables stored:

$$q_{in} = 3.15 \cdot 10^{11} \mathrm{J/s}$$
$$q_{out} = -3.24 \cdot 10^{11} \mathrm{J/s}$$

One can argue that only the difference of those two values is of importance. In this example one finds that the heat content of the outflow is higher than the one of the inflow. Since makes sense when one knows that the data is from December where warmer water from summer is leaving the Persian Gulf at the bottom and colder surface water is entering.

## 4.2 haversine(lon1,lat1,lon2,lat2)

A short function which calculates the distance between two points on a sphere using the haversine formula. See wikipedia: `https://en.wikipedia.org/wiki/Haversine_formula`