pmcpy

Release 0.0.2

Zhonghua Zheng

Jul 01, 2023

OVERVIEW

Author: Dr. Zhonghua Zheng

CHAPTER

ONE

ABOUT

1.1 Introduction

This package is used to post-process PartMC data (see the reference here). Additionally, a mixing state calculator can be used to calculate the mixing state index.

You can play with the mixing state calculator HERE

1.2 Relevant Publications

- Zheng, Z., Curtis, J. H., Yao, Y., Gasparik, J. T., Anantharaj, V. G., Zhao, L., et al. (2021). Estimating submicron aerosol mixing state at the global scale with machine learning and Earth system modeling. *Earth and Space Science*, 8, e2020EA001500. https://doi.org/10.1029/2020EA001500
- Zheng, Z., West, M., Zhao, L., Ma, P.-L., Liu, X., and Riemer, N.: Quantifying the structural uncertainty of the aerosol mixing state representation in a modal model, Atmos. Chem. Phys., 21, 17727–17741, https://doi.org/10.5194/acp-21-17727-2021, 2021.
- Riemer, N. and West, M.: Quantifying aerosol mixing state with entropy and diversity measures, Atmos. Chem. Phys., 13, 11423–11439, https://doi.org/10.5194/acp-13-11423-2013, 2013.

CHAPTER

TWO

INSTALLATION

Step 1: create a conda environment

```
$ conda create -n pmcpy python=3.8
$ conda activate pmcpy
$ conda install -c conda-forge numpy pandas xarray netcdf4
```

Step 2: install using `pip`

```
$ pip install pmcpy
```

(optional) install from source::

```
$ git clone https://github.com/zhonghua-zheng/pmcpy.git
$ cd pmcpy
$ python setup.py install
```

CHAPTER

THREE

PMCPY QUICKSTART

[1]:
```
# import necessary package
import pmcpy
import xarray as xr
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

3.1 load the raw output file from PartMC

[2]:
```
# define the path to data
p = "../data/urban_plume_0001_00000002.nc"
pmc = pmcpy.load_pmc(p)
```

3.2 number concentration

[3]:
```
print("overall number conc.:", pmc.get_num_conc(), "# m^{-3}")
print("diameter<=2.5um:", pmc.get_num_conc(pmc.get_aero_particle_diameter()<=2.5e-6), "#␣
↪m^{-3}")
print("diameter>=2.5um:", pmc.get_num_conc(pmc.get_aero_particle_diameter()>=2.5e-6), "#␣
↪m^{-3}")
```

```
overall number conc.: 8854807162.459946 # m^{-3}
diameter<=2.5um: 8854806282.274422 # m^{-3}
diameter>=2.5um: 880.1855249722205 # m^{-3}
```

3.3 aerosol mass concentration of selected species

[4]:
```
# consider BC and OC only
aero_species_ls = ["BC","OC"]
print("BC and OC mass conc,:",
      pmc.get_aero_mass_conc(aero_species_ls),
      "kg m^{-3}")
```

(continues on next page)

(continued from previous page)

```
# consider all species and dry species
all_aero_species = pmc.aero_species.copy()
dry_aero_species = all_aero_species.copy()
dry_aero_species.remove("H2O")

print("overall mass conc.:",
      pmc.get_aero_mass_conc(all_aero_species).sum(),
      "kg m^{-3}")
print("overall dry mass conc.:",
      pmc.get_aero_mass_conc(dry_aero_species).sum(),
      "kg m^{-3}")
print("PM2.5 mass conc. (with water):",
      pmc.get_aero_mass_conc(all_aero_species,
                             part_cond=(pmc.get_aero_particle_diameter()<=2.5e-6)).sum(),
      "kg m^{-3}")
```

```
BC and OC mass conc,: [6.53030409e-10 5.73984931e-09] kg m^{-3}
overall mass conc.: 2.2186891469474523e-08 kg m^{-3}
overall dry mass conc.: 1.316284475347949e-08 kg m^{-3}
PM2.5 mass conc. (with water): 2.2164289553077785e-08 kg m^{-3}
```

3.4 overall mass concentration

[5]:
```
print("overall mass conc.:",
      pmc.get_mass_conc(dry=False), "kg m^{-3}")
print("overall dry mass conc.:",
      pmc.get_mass_conc(dry=True), "kg m^{-3}")
print("PM2.5 mass conc. (with water):",
      pmc.get_mass_conc(dry=False, part_cond=pmc.get_aero_particle_diameter()<=2.5e-6),
↪"kg m^{-3}")
```

```
overall mass conc.: 2.2186891469474523e-08 kg m^{-3}
overall dry mass conc.: 1.316284475347949e-08 kg m^{-3}
PM2.5 mass conc. (with water): 2.2164289553077805e-08 kg m^{-3}
```

3.5 aerosol density

[6]:
```
print("overall density.:",
      pmc.get_aero_density(dry=False), "kg m^{-3}")
print("overall dry density:",
      pmc.get_aero_density(dry=True), "kg m^{-3}")
print("PM2.5 density (with water):",
      pmc.get_aero_density(dry=False, part_cond=pmc.get_aero_particle_diameter()<=2.5e-
↪6), "kg m^{-3}")
```

```
overall density.: 1179.020687087021 kg m^{-3}
overall dry density: 2265.3505453667535 kg m^{-3}
PM2.5 density (with water): 1178.3768174927013 kg m^{-3}
```

3.6 gas mixing ratio

[7]:
```
gas_list = ['CO','O3']
pmc.get_gas_mixing_ratio(gas_list)
```

[7]:
```
<xarray.DataArray 'gas_mixing_ratio' (gas_species: 2)>
array([354.311384,  43.069374])
Coordinates:
  * gas_species  (gas_species) int32 17 11
Attributes:
    unit:       ppb
    long_name:  mixing ratios of gas species
```

3.7 mixing state index calculation

based on `group_list`

[8]:
```
group_list = [['SO4','Cl','ARO1','ARO2','ALK1','OLE1',
               'API1','API2','LIM1','LIM2','Na'],
              ['BC','OC','OIN']]
pmc.get_mixing_state_index(group_list=group_list, diversity=False)
```

[8]:
```
0.838104186813985
```

based on all species (`without water`)

[9]:
```
pmc.get_mixing_state_index(drop_list=["H2O"], diversity=False)
```

[9]:
```
0.6471920046172539
```

specific `range of diameters` and display `diversity` (D_α, D_γ, χ)

[10]:
```
pmc.get_mixing_state_index(drop_list=["H2O"],
                           part_cond=pmc.get_aero_particle_diameter()<=2.5e-6,
                           diversity=True)
```

[10]:
```
(3.1876143541221267, 4.355134759609094, 0.6520198176412457)
```

3.8 particle size distribution visualization (number concentration v.s. diameter)

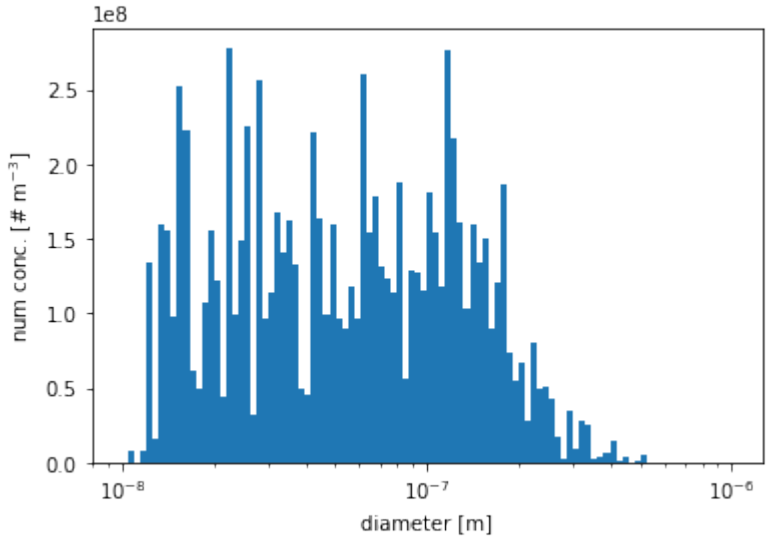
overall number concentration

[11]:
```
# get number distribution
aero_diameter = pmc.get_aero_particle_diameter()
num_conc_per_particle = pmc.ds["aero_num_conc"]

# setup the 111 bins ranged from 10^-8 to 10^-6
bins = np.logspace(-8,-6,2*50+1)
```

(continues on next page)

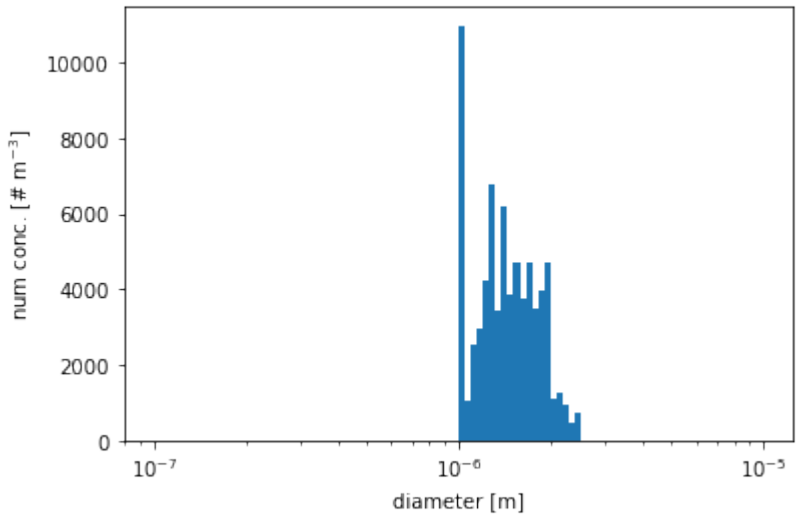
(continued from previous page)

```
# plot the number distribution
plt.hist(aero_diameter, bins=bins, weights=num_conc_per_particle)
plt.xscale('log')
plt.xlabel('diameter [m]')
plt.ylabel(r'num conc. [# m$^{-3}$]')
plt.show()
```



number concentration for particles with diameters between 1μm and 2.5μm

[12]:
```
# get the diameters of all particles
aero_diameter = pmc.get_aero_particle_diameter()
# select particles with diameters between 1um and 2.5um
part_cond = (aero_diameter<=2.5e-6) & (aero_diameter>=1.0e-6)

aero_diameter_cond = aero_diameter[part_cond]
num_conc_per_particle_cond = pmc.ds["aero_num_conc"][part_cond]

# setup the 111 bins ranged from 10^-8 to 10^-6
bins = np.logspace(-7,-5,2*50+1)

# plot the number distribution
plt.hist(aero_diameter_cond, bins=bins, weights=num_conc_per_particle_cond)
plt.xscale('log')
plt.xlabel('diameter [m]')
plt.ylabel(r'num conc. [# m$^{-3}$]')
plt.show()
```



3.9 black carbon fraction in dry particles

[13]:
```
# get dry species per parctile
all_aero_species = pmc.aero_species.copy()
dry_aero_species = all_aero_species.copy()
dry_aero_species.remove("H2O")
mass_per_dry_particle = pmc.ds["aero_particle_mass"]\
                          .sel(aero_species = pmc.aero_species_to_idx(dry_aero_
↪species))\
                          .sum(dim="aero_species").values
# get black carbon mass per particle
bc_per_particle = pmc.ds["aero_particle_mass"]\
                     .sel(aero_species=pmc.aero_species_to_idx(["BC"])).values.reshape(-
↪1)
bc_per_particle
```

[13]:
```
array([0.00000000e+00, 0.00000000e+00, 0.00000000e+00, ...,
       5.78304546e-20, 0.00000000e+00, 0.00000000e+00])
```

[14]:
```
bc_frac_per_dry_particle = bc_per_particle / mass_per_dry_particle
(bc_frac_per_dry_particle > 0.01).sum()
```

[14]:
```
269
```

3.10 black carbon core volume

[15]:
```
BC_density = pmc.ds["aero_density"].sel(aero_species = pmc.aero_species_to_idx(["BC"])).
↪values
BC_density
```

[15]:
```
array([1800.])
```

[16]:
```
# volume = mass / density
pmc.ds["aero_particle_mass"].sel(aero_species = pmc.aero_species_to_idx(["BC"])).values /
↪ BC_density
```

[16]:
```
array([[0.00000000e+00, 0.00000000e+00, 0.00000000e+00, ...,
        3.21280303e-23, 0.00000000e+00, 0.00000000e+00]])
```

CHAPTER

FOUR

MIXING STATE CALCULATOR

This script is used for calculating mixing state index
Step 0: click the link here to launch a jupyter notebook
Step 1: upload your own csv file (each `row` is a `particle`, each `column` is the `mass` of a `species`)
|upload_csv|

Step 2: edit the `file_name` and `group_list` below
Below is an exmaple with three `surrogate species`
The definition of surrogate species is here (Figure 2)
- surrogate species 1: "BC" and "POM"
- surrogate species 2: "DUST"
- surrogate species 3: "SS"

Step 3: click Run button on the top of your jupyter notebook
|run_cell|

Please edit the "file_name" and "group_list", then run the code below

[1]:
```
file_name = "sample_data.csv"
group_list = [["BC","POM"],
              ["DUST"],
              ["SS"]]

# ====== please don't change ======
import numpy as np
import pandas as pd
import pmcpy

# load data
df = pd.read_csv(file_name)
# get matrix for calculation
da = np.concatenate([df[group].values.sum(axis=1).reshape(-1,1) for group in group_list],
↪axis=1)
# calculate mixing state index
D_alpha, D_gamma, chi = pmcpy.get_chi(da)
print("mixing state index:", chi)
```

(continues on next page)

(continued from previous page)

```
print("average particle (alpha) species diversity:", D_alpha)
print("bulk population (gamma) species diversity:", D_gamma)
```

```
mixing state index: 0.9408323695414793
average particle (alpha) species diversity: 2.838147712915256
bulk population (gamma) species diversity: 2.9537462489849164
```

CHAPTER

FIVE

BENCHMARKING

This script is used to compare the pmcpy results with urban_plume_process.F90

[1]:
```
# import necessary package
import pmcpy
import xarray as xr
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# define the path to data
p = "../data/"
```

results from ``pmcpy``

[2]:
```
# define a dictionary to save the results
d = {"tot_num_conc": [], "tot_mass_conc":[],
     "chi":[], "d_alpha":[], "d_gamma":[],
     "chi_a":[], "d_alpha_a":[], "d_gamma_a":[]}

# define the surrogate groups for mixing state calculation
group_list = [["OC","BC"],
              ["API1","API2","LIM1","LIM2"],
              ["SO4","NO3","NH4"]]

# loop across scenarios using pmcpy
for i in range(1,26):
    pmc = pmcpy.load_pmc(p+"/urban_plume_0001_000000"+str(i).zfill(2)+".nc")
    d["tot_num_conc"].append(pmc.get_num_conc())
    d["tot_mass_conc"].append(pmc.get_mass_conc(dry=False))

    # calculate mixing state for grouped species with water
    D_alpha, D_gamma, chi = pmc.get_mixing_state_index(group_list, diversity=True)
    d["d_alpha"].append(D_alpha)
    d["d_gamma"].append(D_gamma)
    d["chi"].append(chi)

    # calculate mixing state for all species (without water)
    D_alpha_a, D_gamma_a, chi_a = pmc.get_mixing_state_index(drop_list=["H2O"],␣
↪diversity=True)
    d["d_alpha_a"].append(D_alpha_a)
```

(continues on next page)

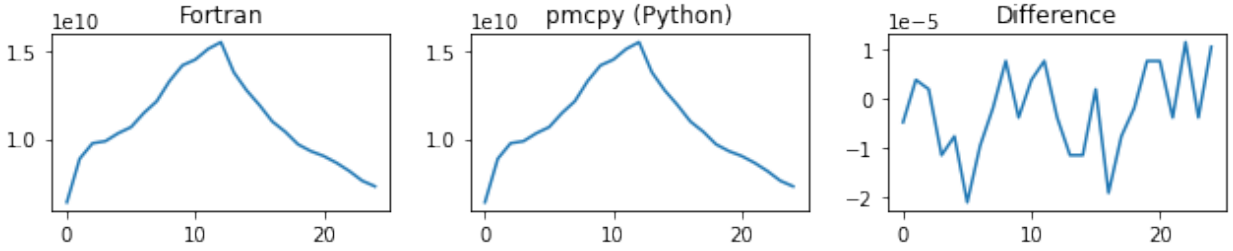
(continued from previous page)

```
    d["d_gamma_a"].append(D_gamma_a)
    d["chi_a"].append(chi_a)
```

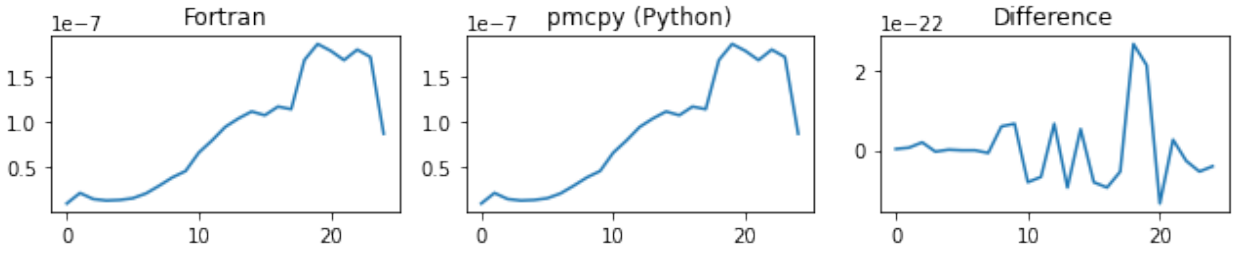
compare the results from Fortran postprocessing ``urban_plume_process.F90``

[3]:
```
ds_b = xr.open_dataset(p+"urban_plume_process.nc")

# ======= comparison =======
for k in d:

    print("######",k,"######")
    fig = plt.figure(figsize=(9,2))
    ax1 = fig.add_subplot(131)
    pd.Series(ds_b[k].values).plot(ax=ax1)
    ax1.set_title("Fortran")

    ax2 = fig.add_subplot(132)
    pd.Series(d[k]).plot(ax=ax2)
    ax2.set_title("pmcpy (Python)")

    ax3 = fig.add_subplot(133)
    pd.Series(ds_b[k].values-np.array(d[k])).plot(ax=ax3)
    ax3.set_title("Difference")

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

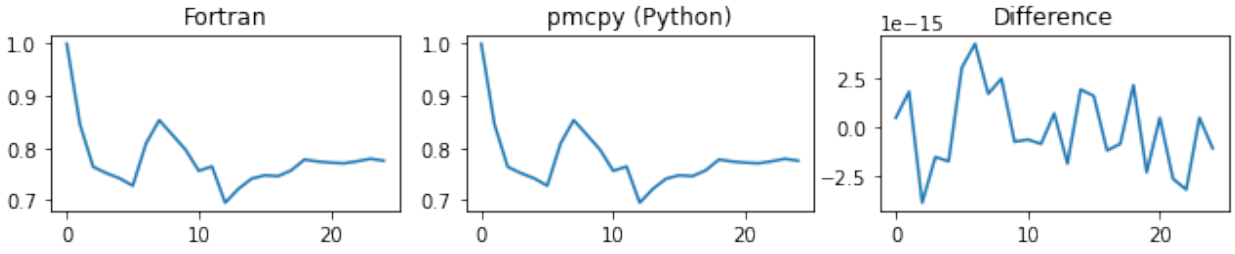
```
###### tot_num_conc ######
```



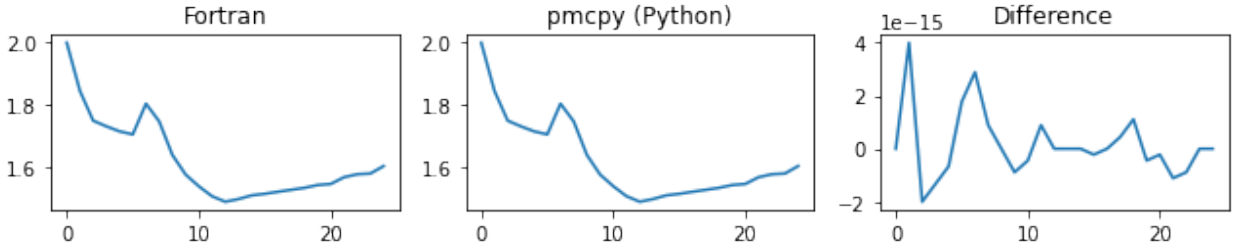

```
###### tot_mass_conc ######
```



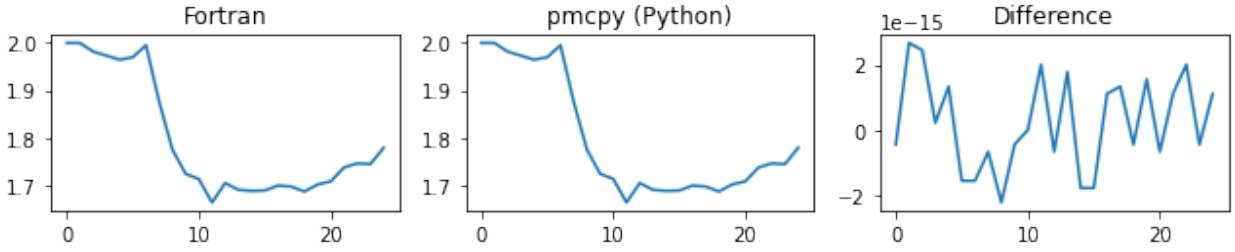

```
###### chi ######
```



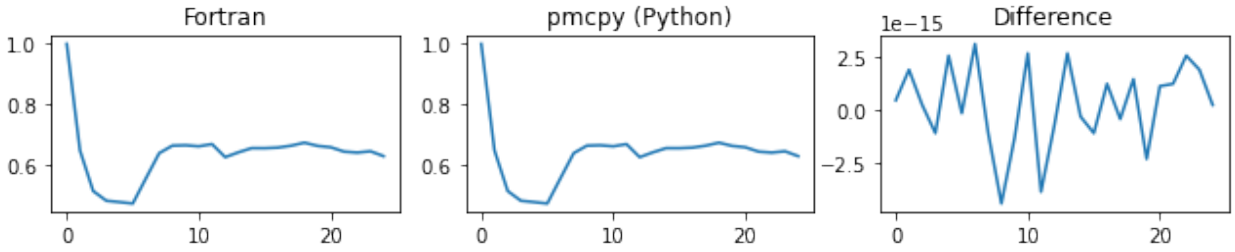

```
###### d_alpha ######
```



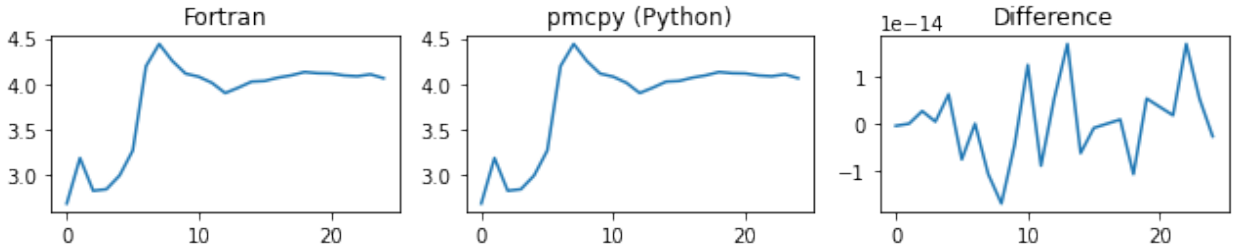

```
###### d_gamma ######
```



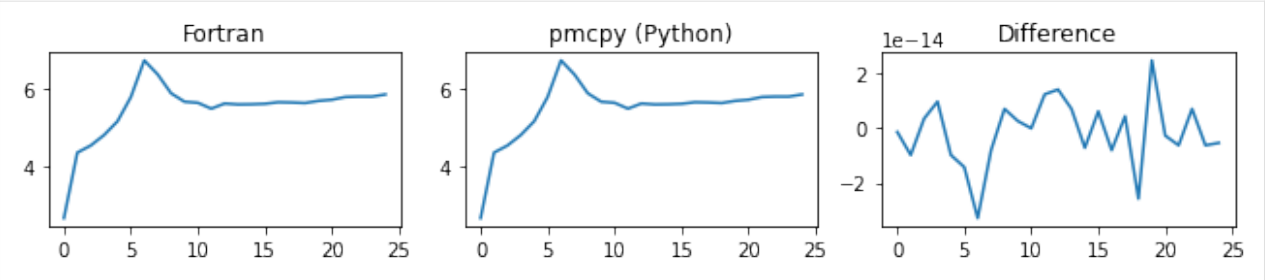

```
###### chi_a ######
```




```
###### d_alpha_a ######
```




```
###### d_gamma_a ######
```


Fortran
pmcpy (Python)
Difference
1e−14
6
4
2
0
−2
0
5
10
15
20
25

CHAPTER

SIX

FORMAT OF PARTMC RAW OUTPUT

Here is a sample of PartMC raw output

[1]:
```
import xarray as xr
p = "../data/urban_plume_0001_00000002.nc"
ds = xr.open_dataset(p)
ds
```

[1]:
```
<xarray.Dataset>
Dimensions:                      (gas_species: 77, aero_species: 20,
                                  aero_source: 8, aero_weight_group: 2,
                                  aero_weight_class: 8, aero_particle: 1246,
                                  aero_removed: 4636)
Coordinates:
  * gas_species                  (gas_species) int32 1 2 3 4 5 ... 73 74 75 76 77
  * aero_species                 (aero_species) int32 1 2 3 4 5 ... 17 18 19 20
  * aero_source                  (aero_source) int32 1 2 3 4 5 6 7 8
  * aero_weight_group            (aero_weight_group) int32 1 2
  * aero_weight_class            (aero_weight_class) int32 1 2 3 4 5 6 7 8
  * aero_particle                (aero_particle) int32 1 2 3 4 ... 1244 1245 1246
  * aero_removed                 (aero_removed) int32 1 2 3 4 ... 4634 4635 4636
Data variables: (12/48)
    time                         float64 ...
    timestep                     float64 ...
    timestep_index               int32 ...
    repeat                       int32 ...
    temperature                  float64 ...
    relative_humidity            float64 ...
    ...                           ...
    aero_refract_core_real       (aero_particle) float64 ...
    aero_refract_core_imag       (aero_particle) float64 ...
    aero_core_vol                (aero_particle) float64 ...
    aero_removed_id              (aero_removed) int32 ...
    aero_removed_action          (aero_removed) int32 ...
    aero_removed_other_id        (aero_removed) int32 ...
Attributes:
    title:         PartMC version 2.5.0 output file
    source:        PartMC version 2.5.0
    UUID:          A96876F0-F62E-45FB-AC47-2FB221548641
    history:       2021-09-22T15:54:26.773-05:00 created by PartMC version 2.5.0
    Conventions:   CF-1.4
```

CHAPTER

SEVEN

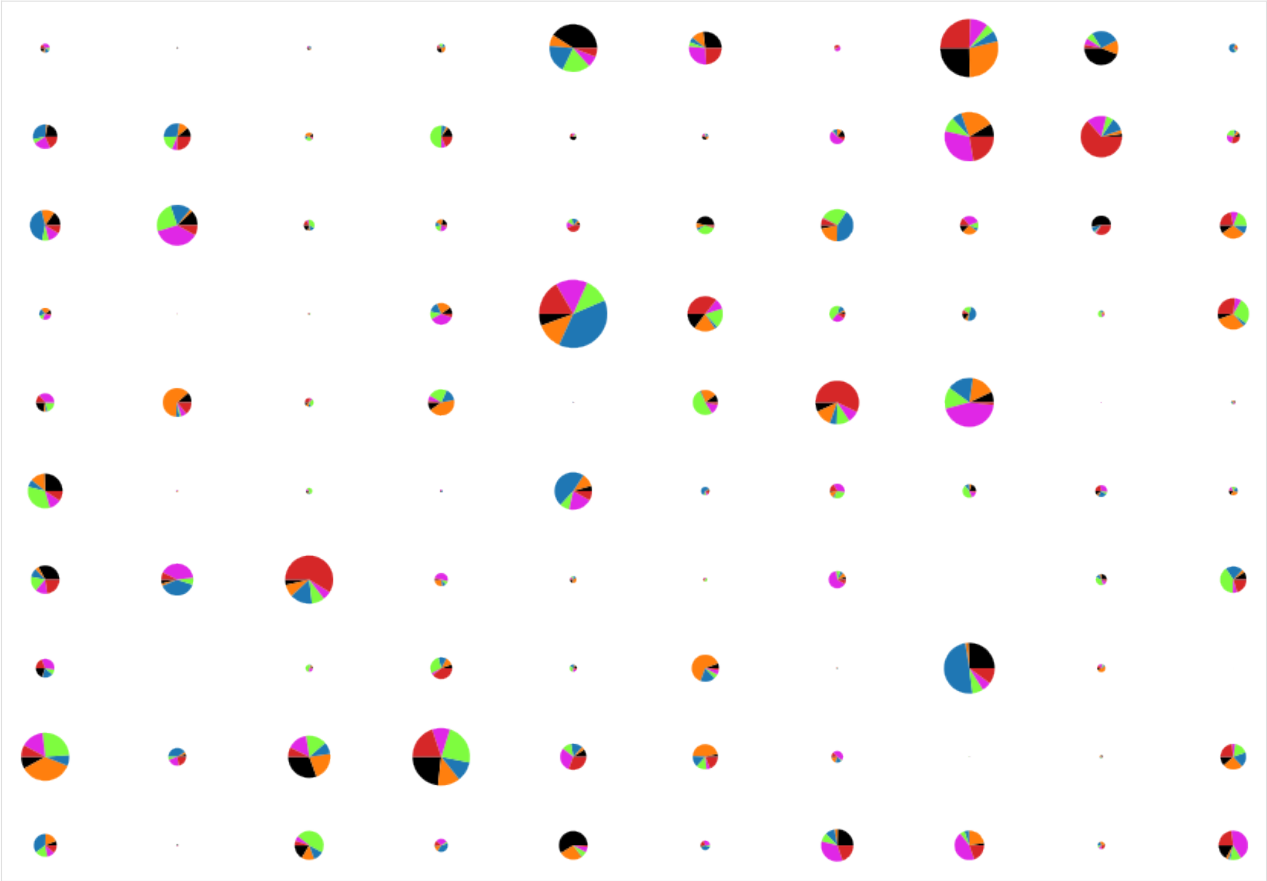
PARTICLES VISUALIZATION

This script is used for generating and plotting particles

[1]:
```
import matplotlib.pyplot as plt
import numpy as np
```

[2]:
```
mat = np.random.lognormal(size=(100,6))
radius = np.random.normal(size=(100,1))
colors = ["#000000","#FF7F0E","#1F77B4","#7efc3f","#e027e6","#D62728"]
```

[3]:
```
fig, axes = plt.subplots(10, 10, figsize=(12,8))
axes=axes.ravel()
# plot each pie chart in a separate subplot
for i in range(100):
    axes[i].pie(mat[i,],radius=radius[i], colors=colors)
plt.tight_layout()
plt.show()
```



CHAPTER

EIGHT

HOW TO ASK FOR HELP

The GitHub issue tracker is the primary place for bug reports.