

DIPContEst

July 6, 2022

Daedalus Ionospheric Profile Continuation (DIPCont) Project

The *DIPCont* project is concerned with the accuracy and the robustness of parameter estimation and (downward) continuation in the lower thermosphere-ionosphere (LTI) region based on Daedalus in situ data. Using synthetic measurements of neutral temperature T_n , neutral density N_n , electron density N_e , and ion temperature T_i along anticipated Daedalus satellite orbit sections around perigee, the *DIPCont* approach proceeds to

- estimate *ensembles of model parameters* through Monte Carlo runs,
- construct *ensembles of altitude profiles* $T_n = T_n(z)$, $N_n = N_n(z)$, $N_e = N_e(z)$, $T_i = T_i(z)$ from the model parameter ensembles,
- compute *ensembles of derived variables* such as ion-neutral collision frequency $\nu_{in} = \nu_{in}(z)$ and *Pedersen conductivity* $\sigma_P = \sigma_P(z)$,
- determine *relative deviations of variables* as functions of altitude z ,
- obtain *extrapolation horizons* for suitable error thresholds.

When using DIPCont code, please acknowledge the DIPCont publication (Joachim Vogt et al., under review).

DIPContEst.ipynb

This Jupyter Notebook is offered as supplementary documentation of the module `DIPContEst.py`.

PYTHON CODE UNDER DEVELOPMENT, PROVIDED AS IS, NO WARRANTY.

Written by Joachim Vogt, Jacobs University Bremen (JUB). Tested also by Octav Marghitu, ISS Bucharest (ISS), Adrian Blagau (ISS, JUB), Leonie Pick (DLR Neustrelitz, JUB), and Nele Stachlys (U Potsdam, JUB); in collaboration with Stephan Buchert, Swedish Institute of Space Physics in Uppsala, Theodoros Sarris, Democritus University of Thrace in Xanthi (DUTH), Stelios Tourgaidis (DUTH), Thanasis Balafoutis (DUTH), Dimitrios Baloukdis (DUTH), and Panagiotis Pirnaris (DUTH).

This version: v0.2, 2022-07-06.

1 DIPContEst.py : DIPCont parameter estimation and extrapolation

The Python module `DIPContEst.py` builds on `DIPContBas.py` that defines the basic setup of the DIPCont project, initializes parameters and variables that are exchanged between different modules and functions, and provides supplementary information. After the following import, a module description is available through `help(DCB)`.

```
[1]: import DIPContBas as DCB
```

In a similar fashion, the module `DIPContEst.py` is imported. Since no exchange variables (global model parameters) are affected, the abbreviation is in lower case letters.

```
[2]: import DIPContEst as dce
```

1.1 Parametric functions for estimates from in situ data

The first section of `DIPContEst.py` contains parametric functions that locally (i.e., at a reference altitude z_0 , z_0) represent the Daedalus observables T_n (**Tn**), N_n (**Nn**), N_e (**Ne**), and T_i (**Ti**). To ensure positivity, parameter estimation is formulated for their logarithms, hence four additional wrapper functions are needed. The parameters that are to be estimated form the argument list of a function, while implicit parameters assumed to be available are provided as exchange variables through `DIPContBas` (**DCB**).

```
[3]: import numpy as np
z = np.linspace(DCB.zBot,DCB.zTop,6)
print('Sample altitude profile [km] : ',z/1e3)
DCB.TnBotEst = 200
print('Neutral temperature estimate [K] at zBot      : TnBotEst = {} K'\
      .format(DCB.TnBotEst))
print('Further implicit parameters of TnModzLoc() : z0 = {} km, zBot = {} km'\
      .format(DCB.z0/1e3,DCB.zBot/1e3))
Tn0 = 500
print('Neutral temperature [K] at altitude z0      : Tn0 = {} K'.format(Tn0))
Tn = dce.TnModzLoc(z,Tn0)
print('Neutral temperature profile [K]:',Tn)
```

```
Sample altitude profile [km] : [100. 120. 140. 160. 180. 200.]
Neutral temperature estimate [K] at zBot      : TnBotEst = 200 K
Further implicit parameters of TnModzLoc() : z0 = 150.0 km, zBot = 100.0 km
Neutral temperature [K] at altitude z0      : Tn0 = 500 K
Neutral temperature profile [K]: [200. 320. 440. 560. 680. 800.]
```

The function `ScaleHeightParLoc()` computes density scale height parameters from temperature data in local representation.

```
[4]: Tn0 = 500
TnBot = 200
print('Neutral temperatures at z0 and zBot          : Tn0 = {} K, TnBotEst = {}_\
      ↪K'\
      .format(Tn0,TnBot))
print('Implicit parameters of ScaleHeightParLoc() : z0 = {:.2f} km, zBot = {}_\
      ↪km'\
      .format(DCB.z0/1e3,DCB.zBot/1e3))
IGHNn,HNn0 = dce.ScaleHeightParLoc('n',Tn0,TnBot)
print('Inverse gradient of density scale height    : IGHNn = {:.2f}'\
      .format(IGHNn))
```

```

        .format(IGHNn))
print('Density scale height at altitude z0          : HNn0 = {:.2f} km'\
      .format(HNn0/1e3))

```

```

Neutral temperatures at z0 and zBot          : Tn0 = 500 K, TnBotEst = 200 K
Implicit parameters of ScaleHeightParLoc()   : z0 = 150.00 km, zBot = 100.0 km
Inverse gradient of density scale height     : IGHNn = 6.51
Density scale height at altitude z0          : HNn0 = 12.80 km

```

1.2 Models for values at the lower boundary

The second section of `DIPContEst.py` provides functions for lower boundary values that are more accurately estimated from models than from local in situ measurements, in particular, latitudinal profiles of neutral and ion temperatures at the lower boundary (`zBot`). To reflect the uncertainties that come with this kind of estimation, predictions obtained from the parametric functions in this section are contaminated by random errors in the Monte Carlo simulations.

```

[5]: x = 0
     TnBotPrd = dce.TnBotPrdx(x)
     print('Lower boundary neutral temperature obtained from TnBotPrdx(): TnBotPrd =_
           ↳{ } K'.format(TnBotPrd))

```

```

Lower boundary neutral temperature obtained from TnBotPrdx(): TnBotPrd = 200.0 K

```

1.3 Monte Carlo parameter ensemble generation

The third section of `DIPContEst.py` is concerned with the generation of model parameter ensembles through Monte Carlo simulations. The workhorse is the function `ParEstMC()` performing a number (`LenSim`) of Monte Carlo runs based on local (x) satellite predictions collected in the Pandas dataframe `dfPrdWin` and, implicitly through variable exchange with `DIPContBas.py` (DCB), predictions of neutral and ion temperatures at the lower boundary. The function `ParEstMC()` is called by `GrdParEstMC()` with predictions contained in a selection window (DCB.`xWin`) at the positions of a horizontal grid (DCB.`xGrd`) to yield a Pandas dataframe (`dfGrdPar`) with all parameter ensembles.

To illustrate the procedure, define a minimalistic horizontal grid, and then display model profiles that are to be used to construct model predictions using plot functions from `DIPContMod.py`.

```

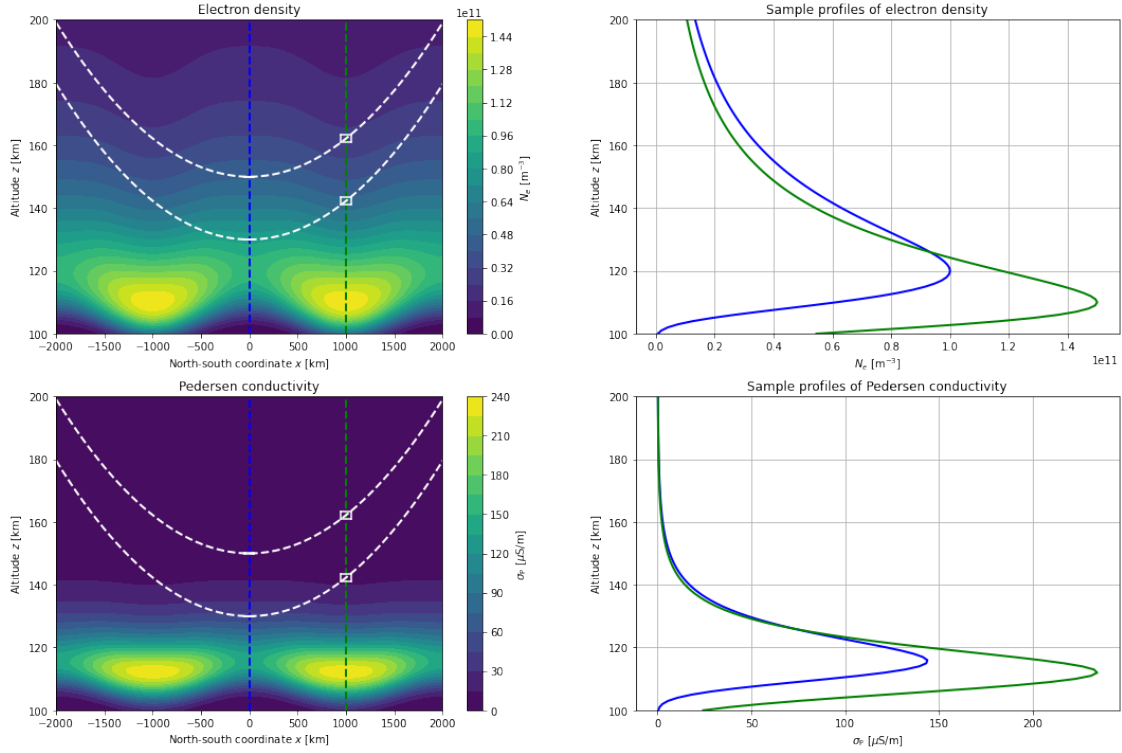
[6]: DCB.LTModelType = 'NeAuroralZoneCrossing'
     DCB.LenGrd = 2
     LenGrd = DCB.LenGrd
     DCB.xGrd = np.linspace(0,0.5*DCB.xRig,DCB.LenGrd)
     xGrd = DCB.xGrd
     DCB.xWin = 100e3
     xWin = DCB.xWin
     from DIPContMod import PltMod2d,PltMod1d
     import matplotlib.pyplot as plt
     PltVarStr = ['Ne','Cped']
     plt.figure(figsize=(18,12))
     for iVar in range(len(PltVarStr)):

```

```

Var = PltVarStr[iVar]
plt.subplot(len(PltVarStr),2,1+2*iVar)
PltMod2d(Var,xGrd=xGrd,xWin=xWin)
plt.subplot(len(PltVarStr),2,2+2*iVar)
PltMod1d(Var,xGrd)

```



Compute predictions along the orbits of satellite A and B.

```

[7]: from DIPContMod import SatPrd
dfPrd = SatPrd(TwoSat=True)
DCB.LenSim = 10
LenSim = DCB.LenSim
dfGrdPar = dce.GrdParEstMC(dfPrd,LenSim=LenSim,EstNeF=False,LinEstLogNe=True)
display(dfGrdPar)

```

iGrd	iSim	z0	Tn0	IGHNn	HNn0	Nn0 \
0	0	130000.000000	435.750904	5.207908	10645.556317	1.754789e+17
	1	130000.000000	436.211538	5.201764	10653.818385	1.776155e+17
	2	130000.000000	434.644871	5.228794	10628.615271	1.734513e+17
	3	130000.000000	436.542180	5.196734	10659.437851	1.731380e+17
	4	130000.000000	436.840620	5.190579	10663.711238	1.756744e+17
	5	130000.000000	432.764295	5.261748	10598.301474	1.766971e+17
	6	130000.000000	440.989513	5.117193	10728.149415	1.763131e+17

	7	130000.000000	443.999429	5.066905	10775.335605	1.725395e+17
	8	130000.000000	440.098899	5.132582	10714.280145	1.771870e+17
	9	130000.000000	445.257305	5.044567	10794.097309	1.723823e+17
1	0	141093.938481	535.457480	5.051436	12985.128677	6.692648e+16
	1	141093.938481	534.158418	5.067250	12963.603827	6.838724e+16
	2	141093.938481	528.658913	5.134771	12871.615912	6.773484e+16
	3	141093.938481	533.313694	5.076495	12948.898413	6.678537e+16
	4	141093.938481	524.085479	5.194351	12795.661228	6.901106e+16
	5	141093.938481	526.946873	5.157070	12843.348575	6.802107e+16
	6	141093.938481	536.267543	5.041311	12998.325969	6.734711e+16
	7	141093.938481	534.562934	5.061261	12969.646689	6.846850e+16
	8	141093.938481	526.943922	5.157573	12843.577879	6.813897e+16
	9	141093.938481	528.705786	5.135280	12873.065914	6.707606e+16

		NeO	LrOcc	NeF	TiO	IGHNi \
iGrd	iSim					
0	0	8.354142e+10	21965.089331	0.0	441.771838	5.105019
	1	8.475654e+10	22980.598571	0.0	440.202683	5.130085
	2	8.459918e+10	21552.721238	0.0	443.045494	5.082335
	3	8.377581e+10	23177.817989	0.0	438.962628	5.150738
	4	8.403643e+10	21449.169117	0.0	437.070067	5.184921
	5	8.465624e+10	21904.765703	0.0	445.380722	5.044289
	6	8.467129e+10	24932.408411	0.0	439.415480	5.144316
	7	8.552011e+10	26689.896194	0.0	436.903924	5.186859
	8	8.388669e+10	23963.045812	0.0	441.064647	5.114041
	9	8.538357e+10	27054.735552	0.0	442.174789	5.098046
1	0	5.167880e+10	184749.814333	0.0	528.045806	5.143207
	1	5.202098e+10	127753.133230	0.0	527.994542	5.144137
	2	5.236278e+10	164452.520225	0.0	530.990255	5.105833
	3	5.273388e+10	494861.312111	0.0	528.560040	5.136564
	4	5.225981e+10	137990.955995	0.0	527.409489	5.151453
	5	5.210866e+10	131847.689975	0.0	531.340531	5.101718
	6	5.199348e+10	161425.137352	0.0	529.892069	5.120375
	7	5.194353e+10	412899.345197	0.0	527.920766	5.144764
	8	5.227451e+10	139153.408433	0.0	533.478136	5.075595
	9	5.190267e+10	199428.055698	0.0	529.929046	5.119306

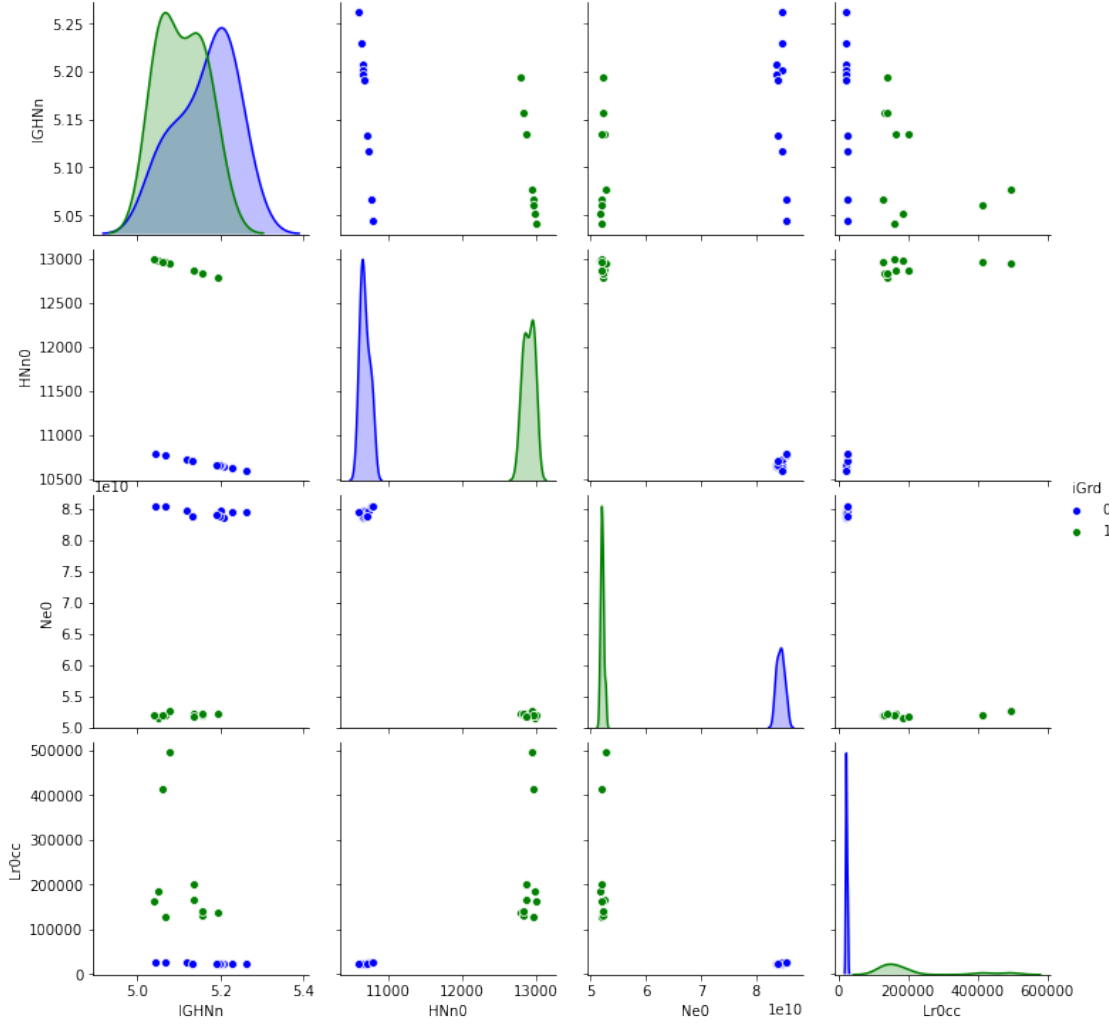
		HNiO
iGrd	iSim	
0	0	10740.956936
	1	10715.544689
	2	10760.211747
	3	10695.732531
	4	10666.533907
	5	10796.941957
	6	10703.546939
	7	10663.431342
	8	10728.371049

	9	10747.171621
1	0	12861.788416
	1	12861.100729
	2	12910.657167
	3	12870.295103
	4	12851.252752
	5	12916.636262
	6	12892.867921
	7	12859.681748
	8	12952.327702
	9	12893.113803

Display the distributions of local parameters.

```
[8]: import seaborn as sns
dfGPL = dfGrdPar.reset_index().loc[:,['IGHNn','HNn0','Ne0','LrOcc','iGrd']]
sns.pairplot(dfGPL,hue='iGrd',palette=DCB.cGrdM)
```

```
[8]: <seaborn.axisgrid.PairGrid at 0x7f9815dfa070>
```



1.4 Local-regional conversion of Ne model parameters

The two functions in the fourth section of `DIPContEst.py` convert model parameters of the electron density in local representation to electron density peak parameters and vice versa. In the following, the function `NeParCnvLoc2Reg()` is employed to obtain the ensemble of electron density peak parameters resulting from the Monte Carlo simulation above.

```
[9]: z0 = dfGrdPar.loc[:, 'z0'].values
HNn0 = dfGrdPar.loc[:, 'HNn0'].values
IGHnN = dfGrdPar.loc[:, 'IGHnN'].values
Ne0 = dfGrdPar.loc[:, 'Ne0'].values
Lr0cc = dfGrdPar.loc[:, 'Lr0cc'].values
from DIPContEst import NeParCnvLoc2Reg
NeIpk, zIpk, HNnIpk = NeParCnvLoc2Reg(Ne0, z0, HNn0, IGHnN, Lr0cc)
dfGPR = dfGrdPar.reset_index().loc[:, ['IGHnN', 'iGrd']]
```

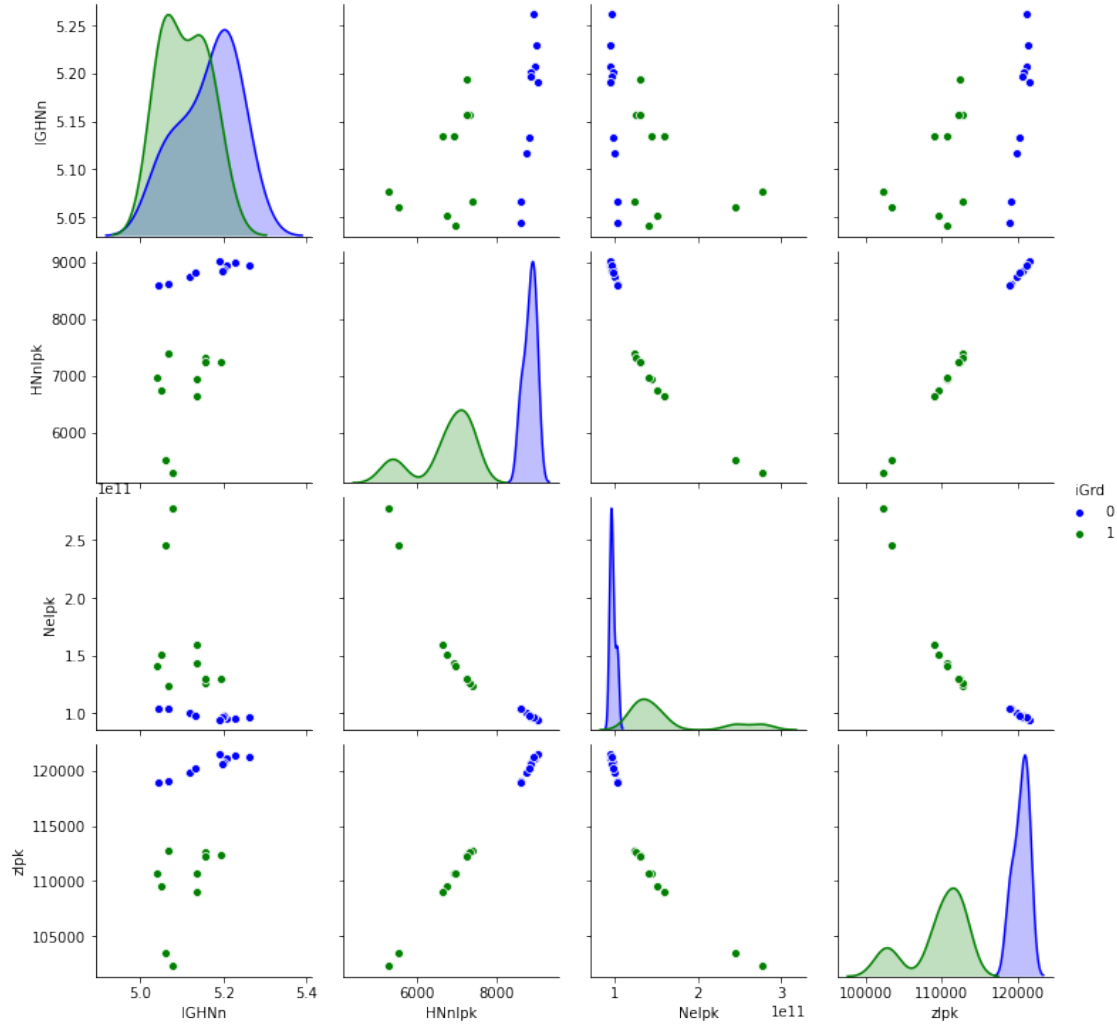
```

dfGPR['HNnIpk'] = HNnIpk
dfGPR['NeIpk'] = NeIpk
dfGPR['zIpk'] = zIpk
display(dfGPR)
sns.pairplot(dfGPR,hue='iGrd',palette=DCB.cGrdM)

```

	IGHNn	iGrd	HNnIpk	NeIpk	zIpk
0	5.207908	0	8962.161790	9.507424e+10	121233.036165
1	5.201764	0	8872.563638	9.786571e+10	120734.333431
2	5.228794	0	8992.348781	9.573960e+10	121444.299710
3	5.196734	0	8858.356749	9.699532e+10	120640.260542
4	5.190579	0	9025.754545	9.488229e+10	121498.056165
5	5.261748	0	8938.239478	9.636485e+10	121265.172693
6	5.117193	0	8741.201599	1.003631e+11	119832.405375
7	5.066905	0	8621.282327	1.037859e+11	119085.617118
8	5.132582	0	8818.039750	9.810015e+10	120267.390637
9	5.044567	0	8600.454291	1.041011e+11	118934.020453
10	5.051436	1	6742.513594	1.515228e+11	109559.765653
11	5.067250	1	7386.210400	1.236145e+11	112831.890318
12	5.134771	1	6950.994078	1.437015e+11	110692.901317
13	5.076495	1	5297.767171	2.779365e+11	102253.007237
14	5.194351	1	7258.212194	1.299260e+11	112330.485562
15	5.157070	1	7334.715557	1.262154e+11	112685.533812
16	5.041311	1	6968.871501	1.410217e+11	110697.584405
17	5.061261	1	5531.803366	2.454174e+11	103449.070707
18	5.157573	1	7240.839316	1.305055e+11	112197.407453
19	5.135280	1	6635.736017	1.591793e+11	109063.501488

[9]: <seaborn.axisgrid.PairGrid at 0x7f98159ad160>



1.5 Extrapolation (profile continuation)

The fifth section of `DIPContEst.py` is concerned with the construction of altitude profiles from model parameter ensembles, thus effectively extrapolating the local in situ satellite observations. The function `VarExtMC()` performs the task for a particular horizontal location. The function `GrdVarExtMC()` produces altitude profile ensembles for the entire horizontal grid `xGrd`. From such profile ensembles, measures of relative deviation are constructed by the function `RelErrVarMC()`.

```
[10]: from DIPContMod import VarModzx
      z1d = DCB.z1d
      dfGrdExt = dce.GrdVarExtMC(z1d,dfGrdPar)
      display(dfGrdExt)
```

iGrd	iSim	z	Tn	Nn	Ne	Ti \
0	0	100000.0	199.959545	1.014005e+19	3.014346e+08	200.070578

0	101000.0	207.819257	8.295468e+18	8.970700e+08	208.127287
0	102000.0	215.678969	6.837210e+18	2.165390e+09	216.183995
0	103000.0	223.538681	5.674435e+18	4.428759e+09	224.240704
0	104000.0	231.398393	4.739845e+18	7.936307e+09	232.297413
...
1	9	196000.0	967.830837	3.006893e+15	1.140100e+10
	9	197000.0	975.828589	2.882465e+15	1.116385e+10
	9	198000.0	983.826341	2.764140e+15	1.093346e+10
	9	199000.0	991.824093	2.651571e+15	1.070960e+10
	9	200000.0	999.821844	2.544436e+15	1.049203e+10

iGrd	iSim	FCin	Cped
0	0	1235.742206	1.321052e-07
	0	1031.101456	4.672843e-07
	0	866.137233	1.327736e-06
	0	732.109144	3.164548e-06
	0	622.418238	6.538643e-06
...
1	9	0.807178	1.711512e-07
	9	0.776969	1.613192e-07
	9	0.748124	1.521249e-07
	9	0.720571	1.435223e-07
	9	0.694241	1.354690e-07

[2020 rows x 7 columns]

1.6 Plot functions for estimation and extrapolation

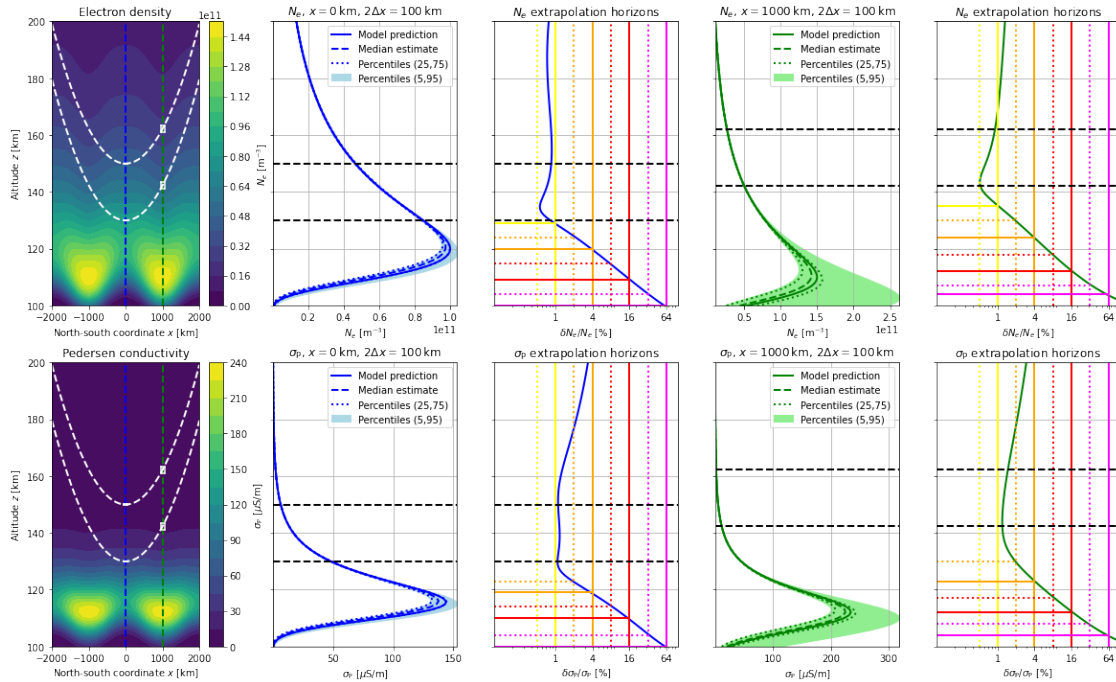
The sixth section of `DIPContEst.py` contains several functions to display the results of Monte Carlo parameter estimation and profile continuation. The function `PltQnt1d()` plots a set of quantiles for the altitude profile ensembles. The function `PltHor1d()` displays the altitude dependence of relative errors as computed from `RelErrVarMC()` for a particular horizontal location, together with a set of error thresholds giving rise to the concept of *extrapolation horizons*.

```
[11]: PltVarStr = ['Ne', 'Cped']
      PltVarTexStr = [r'$N_e$', r'$\sigma_{\mathrm{P}}$',]
      plt.figure(figsize=(20,12))
      for iVar in range(len(PltVarStr)):
          Var = PltVarStr[iVar]
          VarTex = PltVarTexStr[iVar]
          VarEzxs = dfGrdExt[Var].values.reshape((xGrd.size, LenSim, z1d.size)).
      ↪ transpose(2,0,1)
          plt.subplot(len(PltVarStr), 5, 1+5*iVar)
          PltMod2d(Var, xGrd, xWin=xWin)
          for iGrd in range(xGrd.size):
              zA = DCB.zSatA[np.argmin(np.abs(DCB.xSatA-xGrd[iGrd]))]
```

```

zB = DCB.zSatB[np.argmin(np.abs(DCB.xSatB-xGrd[iGrd]))]
plt.subplot(len(PltVarStr),5,2+5*iVar+2*iGrd)
VarPrd = VarModzx(Var,z1d,xGrd[iGrd])
VarEst = VarEzxs[:,iGrd,:]
cM = DCB.cGrdM[iGrd%len(DCB.cGrdM)]
cQ = DCB.cGrdQ[iGrd%len(DCB.cGrdQ)]
dce.
→PltQnt1d(Var,VarEst,VarPrd=VarPrd,zA=zA,zB=zB,zLabel=False,cM=cM,cQ=cQ)
plt.title(VarTex+r', $x = {:.0f}\$,km, $2 \Delta x = {:.0f}\$,km'.
→format(xGrd[iGrd]/1e3,xWin/1e3))
plt.subplot(len(PltVarStr),5,3+5*iVar+2*iGrd)
dce.
→PltHor1d(Var,VarEst,VarPrd=VarPrd,zA=zA,zB=zB,ErrTyp='RMS',zLabel=False,cM=cM)

```



Assuming that the array `xGrd` covers the entire range of horizontal distances, the function `PltHor2d()` displays extrapolation horizons as contours.

```

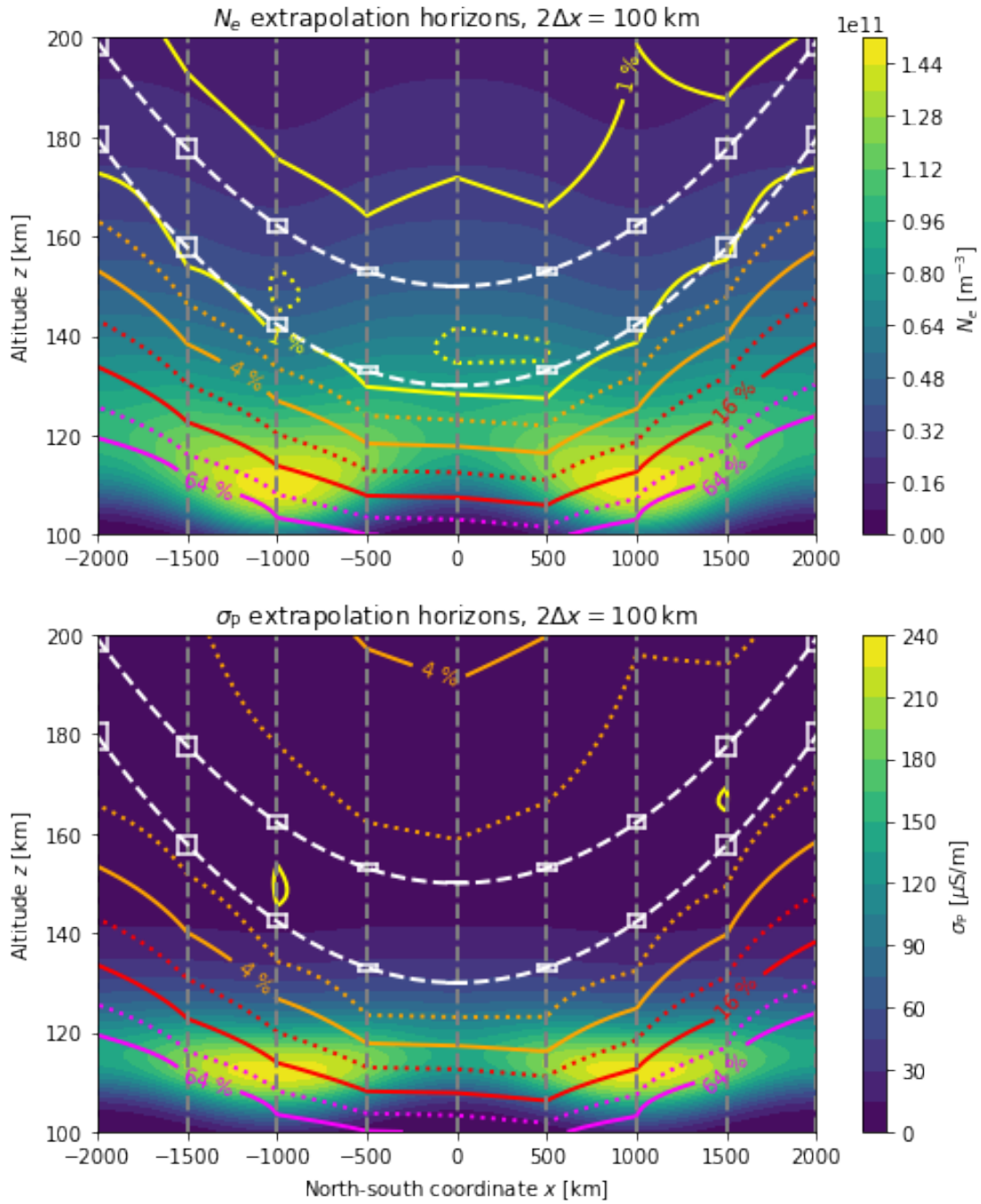
[12]: DCB.LenGrd = 9
LenGrd = DCB.LenGrd
DCB.xGrd = np.linspace(DCB.xLef,DCB.xRig,DCB.LenGrd)
xGrd = DCB.xGrd
DCB.cGrdM = ['gray','gray']
DCB.cGrdQ = ['lightgray','lightgray']
DCB.LenSim = 40
LenSim = DCB.LenSim

```

```

dfGrdPar = dce.GrdParEstMC(dfPrd,LenSim=LenSim,EstNeF=False,LinEstLogNe=True)
dfGrdExt = dce.GrdVarExtMC(z1d,dfGrdPar)
plt.figure(figsize=(8,10))
xGrd2d,zGrd2d = np.meshgrid(xGrd,z1d)
xLabel = False
for iVar in range(len(PltVarStr)):
    Var = PltVarStr[iVar]
    VarTex = PltVarTexStr[iVar]
    VarEst = dfGrdExt[Var].values.reshape((xGrd.size,LenSim,z1d.size)).
↳transpose(2,0,1)
    VarPrd = VarModzx(Var,zGrd2d,xGrd2d)
    plt.subplot(len(PltVarStr),1,1+iVar)
    if iVar==(len(PltVarStr)-1): xLabel = True
    PltMod2d(Var,xGrd=xGrd,xWin=xWin,xLabel=xLabel)
    plt.title(VarTex+r' extrapolation horizons, $2 \Delta x = {:.0f} \backslash, $km'.
↳format(xWin/1e3))
    dce.PltHor2d(VarEst,VarPrd=VarPrd,ErrTyp='RMS')

```



End of DIPContEst.ipynb