

Generation_figures_1_2_3_5_and_6

May 22, 2026

- 1 **Generation of Figures 1, 2, 3, 5 and 6 of the document entitled**
- 2 **Rebuttal to “Comment on ‘A theoretical upper limit for offshore wind energy extraction’ by Simão Ferreira et al. (2026)”:**
reproducibility analysis, audit of the vdLW implementation, and
clarification of finite wind-farm correction methodologies

This script creates Figures 1, 2, 3, 5 and 6 of the document entitled

Rebuttal to “Comment on ‘A theoretical upper limit for offshore wind energy extraction’ by Simão Ferreira et al. (2026)”:
reproducibility analysis, audit of the vdLW implementation, and clarification of finite wind-farm correction methodologies

by

Jens Nørkær Sørensen, Gunner Chr. Larsen, and Carlos Simão Ferreira

first released on 2026 May 16th

at <https://wes.copernicus.org/preprints/wes-2026-59/> CC1: ‘Comment on wes-2026-59’, Jens Nørkær Sørensen, 16 May 2026

2.1 Combined Notebook Structure and Scope

This notebook combines two originally separate analysis notebooks into a single reproducible document for publication through nbviewer and long-term archival purposes.

The original analyses were developed independently:

- **Part I — Type 1 implementation errors**
Corresponding to Figures 1–3 of the rebuttal paper.
This section focuses on direct reproducibility issues in the vdLW implementation, including coding and data-processing errors that affect the reproduced results.
- **Part II — Type 2 methodological and geometric errors**
Corresponding to Figures 5–6 of the rebuttal paper.
This section investigates limitations related to wind-farm representation, clustering procedures, sub-farm handling, and geometric interpretation within the vdLW methodology.

Because both notebooks were originally designed as standalone and self-contained analyses, some repetition in imports, variable definitions, introductory explanations, and plotting routines is in-

tentionally preserved in the merged version. This redundancy is deliberate and serves several purposes:

1. It preserves the independent reproducibility of each analysis block.
2. It allows sections to still be executed independently if extracted from the notebook.
3. It avoids introducing additional restructuring complexity that could unintentionally alter results during consolidation.
4. It keeps the merged notebook closely aligned with the original development and audit workflow used during preparation of the rebuttal.

The notebook is therefore best interpreted as a concatenation of two reproducible technical audit workflows collected into a single executable document for transparency and ease of public access.

3 Generation of Figures 1, 2 and 3 - Correction of Type 1 errors

This script creates Figures 1, 2 and 3 of the document entitled

Rebuttal to “Comment on ‘A theoretical upper limit for offshore wind energy extraction’ by Simão Ferreira et al. (2026)”: reproducibility analysis, audit of the vdLW implementation, and clarification of finite wind-farm correction methodologies

by

Jens Nørkær Sørensen, Gunner Chr. Larsen, and Carlos Simão Ferreira

first released on 2026 May 16th

at <https://wes.copernicus.org/preprints/wes-2026-59/> CC1: ‘Comment on wes-2026-59’, Jens Nørkær Sørensen, 16 May 2026

3.1 Import of necessary libraries

```
[103]: # first, we will import the necessary libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

3.2 Definition and modification of the function by van der Paul and Watson

The original function was published by van der Laan and Watson (vdLW) in <https://doi.org/10.5281/zenodo.19235300>, with the code in the file “run.py”

We could just import the function from the “run.py” file by vdLW. However, the function has an error in one of the equations. We need to modify the function to run with and without the error.

Therefore, we copy here the function by vdLW and make clear what is modified. We just added a boolean flag to run the original eps2 formulation by vdLW and the correct one, added now by SLS. This is to correct Type 1 error 4 as described in the Rebuttal document.

```
[104]: from scipy.special import gamma, gammainc
from scipy.optimize import fsolve
```

```

class MinimalisticPredictionModel():
    """Sørensen, J.N.; Larsen, G.C.
    A Minimalistic Prediction Model to Determine Energy Production and Costs of
    ↪Offshore Wind Farms.
    Energies 2021, 14, 448. https://doi.org/10.3390/en14020448"""

    def __init__(self, correction_factor, latitude, CP, Uin, Uout, rho,
    ↪correct_eps2=False):
        ##### Start of note by SLS
    ↪#####
        # The original code by vdLW had the function header as: def
    ↪__init__(self, correction_factor, latitude, CP, Uin, Uout, rho):
        #
        # We have added an optional argument 'correct_eps2' to the function
    ↪header. This argument is a boolean.
        # If False (default), the function will use the original calculation
    ↪for eps2 as given by vdLW.
        # If True, the function will use the correct formula for eps2
        #
        ##### End of note by SLS
    ↪#####
        """
        Parameters
        -----
        correction_factor : int, float or function
            Finite-size wind farm correction which multiplied with sqrt(Nturb)
    ↪gives
            the number of wind turbines exposed to the free wind
        latitude : int or float
            latitude [deg] used to calculate the coriolis parameter
        CP : float, optional
            Wind turbine power coefficient
        Uin : int or float, optional
            Wind turbine cut-in wind speed
        Uout : int or float, optional
            Wind turbine cut-out wind speed
        rho : float, optional
            Air density
        """

        self.CP = CP
        self.Uin = Uin
        self.Uout = Uout
        omega = 2 * np.pi / (24 * 60 * 60) # earth rotation speed
        self.f = 2 * omega * np.sin(np.deg2rad(latitude))
        self.correction_factor = correction_factor

```

```

self.rho = rho

##### Start of code alteration by SLS to add correct_eps2
→ argument, see note at the beginning of the class #####
self.correct_eps2 = correct_eps2
##### End of code alteration by SLS to add correct_eps2
→ argument #####

def predict(self, Pg, CT, D, H, z0, Aw, kw, Nturb, Area):
    """
    Inputs:
        Pg      - [W] Nameplate capacity (generator power)
        CT      - [-] Thrust coefficient
        D       - [m] Rotor diameter
        H       - [m] Tower height
        z0      - [m] roughness length
        Aw      - [m/s] Weibull scale parameter
        kw      - [-] Weibull shape parameter
        Nturb   - [-] Number of turbines
        Area    - [m2] Area of wind farm

    Outputs:
        power   - [Wh] Annual energy production of the wind farm
        ws_eff  - [m/s] Effective mean wind speed including wakes
    """

    kappa = 0.4 # [-] Von Karman constant
    Uin, Uout = self.Uin, self.Uout

    # factor defined by Frandsen, should be used instead of f in eq 13 and
    → 19 (typos in paper)
    fm = self.f * np.exp(4)
    delta = np.log(H / z0) # eq 19

    # Mean spacing between wt in diameters, eq 8
    S = np.sqrt(Area) / (D * (np.sqrt(Nturb) - 1))

    # Rated wind speed [m/s], eq 4
    Ur = (8 * Pg / (self.rho * np.pi * D**2 * self.CP))**(1 / 3)

    # Power modeled as  $P = \alpha * U^3 + \beta$ , eq 1
    alpha = Pg / (Ur**3 - Uin**3) # [(m/s)-3] eq 2
    beta = -Pg * Uin**3 / (Ur**3 - Uin**3) # [-], eq 2

    Uh0 = Aw * gamma(1 + 1 / kw) # [m/s] Mean velocity at hub height
    Ctau = np.pi * CT / (8 * S * S) # [-] Wake parameter, rotor ct smeared
    → on WT area

```

```

nu = np.sqrt(0.5 * Ctau) * D / (kappa**2 * H) * delta # [-] wake eddy
↳viscosity

# Finite-size wind farm correction, section 2.5
correction_factor = self.correction_factor
if hasattr(correction_factor, '__call__'):
    correction_factor = correction_factor(Uh0, S, Nturb)
Nfree = correction_factor * np.sqrt(Nturb) # Number of wt exposed to
↳the free wind
Nfree = np.minimum(Nfree, Nturb) # To make sure Nfree/Nturb is not
↳larger than one, not yet implemented in PyWake

# Geostrophic wind speed
G_last = Uh0
for n in range(10):
    G = Uh0 * (1 + np.log(G_last / (fm * H)) / delta)
    dG = abs(G - G_last)
    if dG < 1e-5:
        break
    G_last = G

gam = np.log(G / (fm * H)) # eq 19

# Mean velocity at hub height without wake effects from geostrophic wind
Uh0 = G / (1 + gam / kappa * np.sqrt((kappa / delta)**2)) # eq 13, ct=0

# Power without wake effects, eq 16 modified by
# - add gamma(1 + 3 / kw) to cancel out normalization in scipy's
↳gammainc
# - gammainc terms swapped (typo in paper)
def get_Py(Aw, Aw_out): # Yearly power
    return alpha * Aw**3 * gamma(1 + 3 / kw) * (gammainc(1 + 3 / kw,
↳(Ur / Aw)**kw) - gammainc(1 + 3 / kw, (Uin / Aw)**kw)) + \
        beta * (np.exp(-(Uin / Aw)**kw) - np.exp(-(Ur / Aw)**kw)) + \
        Pg * (np.exp(-(Ur / Aw)**kw) - np.exp(-(Uout / Aw_out)**kw))

P_y = get_Py(Aw, Aw)

# Without cutin and cutout
# def get_Py(Aw): # Yearly power
#     x = (Ur / Aw) ** kw
#     ks = 1 + 3 / kw
#     return Pg * (x ** (1.0 - ks) * gamma(ks) * gammainc(ks, x) + np.
↳exp(-x))
# def get_Py(Aw): # Yearly power
#     x = (Ur / Aw)

```

```

# ks = 1 + 3 / kw
# return Pg * (x ** (-3) * gamma(ks) * gammainc(ks, x ** kw) + np.
→exp(-x ** kw))
# P_y = get_Py(Aw)

# Mean velocity at hub height with wake effects
z0_lo = z0 # / (1 - D / (2 * H))**(nu / (1 + nu)) # ???
Uh = G / (1 + gam * np.sqrt(Ctau + (kappa / np.log(H / z0_lo))**2) /
→kappa)

# eq 18. The paper states 3/2 instead of 3.2 which is either a typo or
→an initial guess
# eps2 corresponds to eps(Uout) in paper and eps2(Ur)=eps1
eps1 = (1 + gam / delta) / (1 + gam / kappa * np.sqrt(Ctau + (kappa /
→delta)**2))
eps2 = (1 + gam / delta) / (1 + gam / kappa * np.sqrt(Ctau * (Ur /
→Uh)**3.2 + (kappa / delta)**2))

##### Start of code alteration by SLS to correct eps2, see
→note at the beginning of the class #####
if self.correct_eps2: #
    # use the correct formulation
    eps2 = (1 + gam / delta) / (1 + gam / kappa * np.sqrt(Ctau * (Ur /
→Uout)**3.2 + (kappa / delta)**2))
    ##### End of code alteration by SLS to correct eps2
→#####

# print('e', eps1, correction_factor)
# Power production with wake effects
P_WFy = get_Py(eps1 * Aw, eps2 * Aw)

power = ((Nturb - Nfree) * P_WFy + Nfree * P_y)
ws_eff = ((Nturb - Nfree) * Uh + Nfree * Uh0) / Nturb

# Phi without cutin and cutout
def get_phi(x): # Yearly power
    # x = U_r / (Uh0 eps)
    ks = 1 + 3 / kw
    Gm = gamma(1 + 1 / kw)
    return Pg * ((x * Gm) ** (-3) * gamma(ks) * gammainc(ks, (x * Gm)
→** kw) + np.exp(-(x * Gm) ** kw)) - power / Nturb

```

```

root = fsolve(get_phi, x0=1) # initial guess
phi = root[0]

return power, ws_eff, P_y, P_WFy, Uh, G, phi

```

3.3 Definition of a function to run all wind farm cases, and return the capacity factors

This function receives all input data for each wind farm case, and loops through all the cases. For each case, it creates an instance of the class **MinimalisticPredictionModel** by vdLW, and calculates the different capacity factors (isolated wind turbine, infinite wind farm and finite wind farm)

```

[105]: def run_code_all_cases(Turbine_rated_power, CT, Turbine_rotor_diameter,
    ↪ Turbine_hub_height, z0, Wind_lambda, kw, Windfarm_N_turbines, Windfarm_area,
    ↪ correction_factor, latitude, CP, Uin, Uout, rho,
    ↪ correct_eps2=False):

    # function created by SLS to run the code for all cases, this is done to
    ↪ avoid having to run the code for each case separately in the notebook,
    # which would make it more difficult to read and understand the code.
    # This function takes as input the parameters for all cases and returns the
    ↪ outputs for all cases in arrays.

    # define the number of cases that the code will be run for, this is done by
    ↪ taking the length of the Windfarm_area array,
    # but it can be any of the wind farm related input arrays as they should
    ↪ all have the same length
    n_cases = len(Windfarm_area)

    # create arrays for the outputs, now zeros with the same length as the
    ↪ number of cases
    # first, arrays for the outputs of the predict function, these will be
    ↪ filled in the loop below, and then used to calculate the correction factors
    power = np.zeros(n_cases)
    ws_eff = np.zeros(n_cases)
    P_y = np.zeros(n_cases)
    P_WFy = np.zeros(n_cases)
    Uh = np.zeros(n_cases)
    G = np.zeros(n_cases)
    phi = np.zeros(n_cases)

    # now, arrays to return as output of this function, these will be filled in
    ↪ the loop below using the outputs of the predict function
    Cf_Free = np.zeros(n_cases)
    Cf_Inf = np.zeros(n_cases)

```

```

Cf_Model = np.zeros(n_cases)

# we loop over the number of cases, and for each case we run the code and
→fill the output arrays
for i in range(n_cases):
    # check if the input is a single value or an array, if an array use the
→i-th value, if a single value use it directly
    # this is done to allow for both cases where the input is a single
→value (same for all cases) or an array (different for each case)
    correction_factor_i = correction_factor[i] if
→isinstance(correction_factor, (list, np.ndarray)) else correction_factor
    latitude_i = latitude[i] if isinstance(latitude, (list, np.ndarray))
→else latitude
    CP_i = CP[i] if isinstance(CP, (list, np.ndarray)) else CP
    Uin_i = Uin[i] if isinstance(Uin, (list, np.ndarray)) else Uin
    Uout_i = Uout[i] if isinstance(Uout, (list, np.ndarray)) else Uout
    rho_i = rho[i] if isinstance(rho, (list, np.ndarray)) else rho
    Turbine_rated_power_i = Turbine_rated_power[i] if
→isinstance(Turbine_rated_power, (list, np.ndarray)) else Turbine_rated_power
    CT_i = CT[i] if isinstance(CT, (list, np.ndarray)) else CT
    Turbine_rotor_diameter_i = Turbine_rotor_diameter[i] if
→isinstance(Turbine_rotor_diameter, (list, np.ndarray)) else
→Turbine_rotor_diameter
    Turbine_hub_height_i = Turbine_hub_height[i] if
→isinstance(Turbine_hub_height, (list, np.ndarray)) else Turbine_hub_height
    z0_i = z0[i] if isinstance(z0, (list, np.ndarray)) else z0
    Wind_lambda_i = Wind_lambda[i] if isinstance(Wind_lambda, (list, np.
→ndarray)) else Wind_lambda
    kw_i = kw[i] if isinstance(kw, (list, np.ndarray)) else kw
    Windfarm_N_turbines_i = Windfarm_N_turbines[i] if
→isinstance(Windfarm_N_turbines, (list, np.ndarray)) else Windfarm_N_turbines
    Windfarm_area_i = Windfarm_area[i] if isinstance(Windfarm_area, (list,
→np.ndarray)) else Windfarm_area

    # create an instance of the MinimalisticPredictionModel class with the
→input parameters for this case,
    # and then call the predict function to get the outputs for this case,
→and fill the output arrays
    predictionmodel_vdLW = MinimalisticPredictionModel(correction_factor_i,
→latitude_i, CP_i, Uin_i, Uout_i, rho_i, correct_eps2=correct_eps2)

    power[i], ws_eff[i], P_y[i], P_WFy[i], Uh[i], G[i], phi[i] =
→predictionmodel_vdLW.predict(Turbine_rated_power_i, CT_i,
→Turbine_rotor_diameter_i, Turbine_hub_height_i, z0_i, Wind_lambda_i, kw_i,
→Windfarm_N_turbines_i, Windfarm_area_i)

```



```

Cf_Free[i] = P_y[i] / Turbine_rated_power_i
Cf_Inf[i] = P_WFy[i] / Turbine_rated_power_i
Cf_Model[i] = power[i] / (Turbine_rated_power_i * Windfarm_N_turbines_i)

return Cf_Free, Cf_Inf, Cf_Model

```

3.4 Read input data from Output/Input files by vdLW

vdLW, with their code, create a file named “Output.csv”, which contains both inputs and outputs used by vdLW, including the inputs used by SLS.

An important note is that vdLW do not use all the wind farm cases. vdLW choose not to use the cases of the wind farms “Baltic 1”, “Baltic 2”, “Princess Amalia”, “Luchterduinen”. Therefore, after reading the files, we will need to remove those cases.

```

[106]: # read output/input data from vdLW's code
df = pd.read_csv("output.csv")
db_first_eval = df.copy() # this is just not to destroy the original dataframe,
    ↪as we will be modifying it in the next steps

# we will eliminate the cases of Baltic I and II and Princess Amalia and
    ↪Luchterduinen, as they are not evaluated by vdLW's work
db_first_eval = db_first_eval[~db_first_eval["Name"].isin(["Baltic 1", "Baltic
    ↪2", "Princess Amalia", "Luchterduinen"])]

# display the entire database for the first evaluation, indicating explicitly
    ↪to plot all the rows, to avoid the Jupyter notebook from truncating the
    ↪output
pd.set_option("display.max_rows", None)
pd.set_option("display.max_columns", None)
display(db_first_eval)

```

	Index	subindex	Cluster	Name	Country	MW	install	\
0	1	0	0	Arkona	DE	385.0		
1	2	0	0	Wikinger	DE	350.0		
2	3	0	0	Kriegers Flak	DK	604.8		
5	6	0	0	Rodsand 1	DK	165.6		
6	7	0	0	Rodsand 2	DK	207.0		
7	8	0	0	Lillgrund	SE	110.4		
8	9	0	0	Anholt 1	DK	399.6		
9	10	0	0	Horns Rev 1	DK	160.0		
10	11	0	0	Horns Rev 2	DK	209.3		
11	12	0	0	Horns Rev 3	DK	406.7		
12	13	0	0	DanTysk	DE	288.0		
13	14	0	0	Sandbank	DE	288.0		
14	15	0	0	Butendiek	DE	288.0		

15	16	0	1	Amrumbank West	DE	302.0
16	17	0	1	Kaskasi	DE	342.0
17	18	0	1	Nordsee Ost	DE	295.2
18	19	0	1	Meerwind Sud/Ost	DE	288.0
19	20	0	2	Global Tech I	DE	400.0
20	21	0	2	Albatros	DE	112.0
21	22	0	2	Hohe See	DE	497.0
22	23	0	3	Deutsche Bucht	DE	252.0
23	24	0	3	Veja Mate	DE	402.0
24	25	0	3	BARD	DE	400.0
25	26	0	4	Gode 1 and 2	DE	582.0
26	27	0	4	Nordsee One	DE	332.0
27	28	0	4	Borkum Riffgrund I	DE	312.0
28	29	0	4	Borkum Riffgrund II	DE	450.0
29	30	0	4	Trianel I and II	DE	403.2
30	31	0	4	Merkur	DE	396.0
31	32	0	4	Alpha Ventus	DE	60.0
32	33	0	0	Riffgat	DE	113.4
33	34	0	0	Gemini	NL	600.0
34	35	0	0	Fryslan	NL	382.7
37	38	0	5	Borssele I-II	NL	752.0
38	39	0	5	Norther	BE	370.0
39	40	0	5	Thortonbank	BE	325.0
40	41	0	5	Rentel	BE	309.0
41	42	0	5	Northwind	BE	216.0
42	43	0	5	Nobelwind	BE	165.0
43	44	0	5	Belwind Phase 1	BE	171.0
44	45	0	5	Northwester 2	BE	219.0
45	46	0	5	Seamade Mermaid	BE	235.0
46	47	0	5	Seamade Seastar	BE	252.0
47	48	0	0	Rampion	UK	400.0
48	49	0	0	Thanet	UK	300.0
49	50	0	0	London Array	UK	630.0
50	51	0	0	Gunfleet Sand	UK	172.8
51	52	0	6	Galloper	UK	353.0
52	53	0	6	Greater Gabbard	UK	504.0
53	54	0	0	East Anglia One	UK	714.0
54	55	0	0	Lincs	UK	270.0
55	56	0	0	Lynn and Inner Dowsing	UK	194.0
56	57	0	0	Humber Gateway	UK	219.0
57	58	0	0	Westermest Rough	UK	210.0
58	59	0	0	Sheringham Shoal	UK	317.0
59	60	0	0	Dudgeon	UK	402.0
60	61	0	0	Race Bank	UK	573.0
61	62	0	0	Triton Knoll	UK	857.0
62	63	0	7	Hornsea 1	UK	1218.0
63	64	0	7	Hornsea 2	UK	1386.0
64	65	0	8	Moray East	UK	950.0

65	66	0	8	Beatrice extension	UK	588.0
66	67	0	9	Walney Extension	UK	659.0
67	68	0	9	West of Duddon Sands	UK	389.0
68	69	0	9	Walney 1	UK	183.6
69	70	0	9	Walney 2	UK	183.6
70	71	0	0	Gwynt y Môr	UK	576.0
71	72	0	0	Robin Rigg	UK	174.0

	PG	D	WT	densi	AreaR_Afarm	Ht	A	Href	Nt	lambda	\
0	6.000000	154.0		322	0.0302	102.0	37.00	100	60	10.75	
1	5.000000	135.0		349	0.0310	97.5	32.30	100	70	10.75	
2	8.400000	167.0		383	0.0088	107.0	179.00	100	72	11.00	
5	2.300000	82.0		436	0.0173	69.0	22.00	100	72	10.80	
6	2.300000	93.0		339	0.0175	68.0	35.00	100	90	10.85	
7	2.300000	93.0		339	0.0679	65.0	4.80	100	48	10.50	
8	3.600000	120.0		318	0.0140	81.6	89.40	100	111	11.00	
9	2.000000	80.0		398	0.0201	70.0	20.00	100	80	11.50	
10	2.300000	93.0		339	0.0187	68.0	33.00	100	91	11.50	
11	8.000000	164.0		379	0.0118	112.0	88.00	100	49	11.50	
12	3.600000	120.0		318	0.0139	88.0	64.90	100	80	11.30	
13	4.000000	130.0		301	0.0159	94.8	60.00	100	72	11.35	
14	3.600000	120.0		318	0.0215	91.0	42.00	100	80	11.26	
15	3.780000	120.0		334	0.0283	90.0	32.00	100	80	11.20	
16	9.000000	167.0		411	0.0478	107.5	17.40	100	38	11.20	
17	6.150000	126.0		493	0.0166	92.0	36.00	100	48	11.20	
18	3.600000	120.0		318	0.0220	89.0	41.10	100	80	11.20	
19	5.000000	116.0		473	0.0206	82.0	41.00	100	80	11.22	
20	7.000000	154.0		376	0.0271	105.0	11.00	100	16	11.22	
21	7.000000	154.0		376	0.0315	105.0	42.00	100	71	11.22	
22	8.400000	167.0		383	0.0300	100.0	22.60	100	31	11.20	
23	6.000000	154.0		322	0.0244	103.3	51.20	100	67	11.22	
24	5.000000	122.0		428	0.0166	90.0	56.30	100	80	11.22	
25	6.000000	154.0		322	0.0246	104.0	73.30	100	97	11.15	
26	6.200000	126.0		497	0.0163	90.0	41.30	100	54	11.15	
27	4.000000	120.0		354	0.0245	83.0	36.00	100	78	11.22	
28	8.000000	164.0		379	0.0329	105.0	36.00	100	56	11.22	
29	5.580000	133.2		400	0.0180	90.0	55.60	100	72	11.22	
30	6.000000	150.0		340	0.0248	100.0	47.00	100	66	11.22	
31	5.000000	121.0		435	0.0345	92.0	4.00	100	12	11.22	
32	3.780000	120.0		334	0.0565	90.0	6.00	100	30	11.00	
33	4.000000	130.0		301	0.0284	89.0	70.00	100	150	11.22	
34	4.300000	130.0		324	0.0338	115.0	35.00	100	89	10.70	
37	8.000000	167.0		365	0.0160	116.0	128.30	100	94	10.60	
38	8.000000	164.0		379	0.0211	98.0	44.00	100	44	10.60	
39	6.018519	126.0		483	0.0339	95.0	19.84	100	54	10.60	
40	7.350000	154.0		395	0.0344	119.0	22.72	100	42	10.60	
41	3.000000	112.0		305	0.0489	71.0	14.50	100	72	10.60	
42	3.300000	112.0		335	0.0249	79.0	19.80	100	50	10.60	

43	3.000000	90.0	472	0.0210	72.0	17.00	100	56	10.60
44	9.500000	164.0	450	0.0405	105.0	12.00	100	23	10.60
45	8.400000	167.0	383	0.0367	109.0	16.70	100	28	10.60
46	8.400000	167.0	383	0.0336	109.0	19.54	100	30	10.60
47	3.450000	116.0	326	0.0170	80.0	72.00	100	116	10.56
48	3.000000	90.0	472	0.0182	70.0	35.00	100	100	10.60
49	3.600000	120.0	318	0.0162	87.0	122.00	100	175	10.57
50	3.600000	107.0	400	0.0317	75.0	13.60	100	48	10.62
51	6.300000	154.0	338	0.0060	103.0	174.00	100	56	10.70
52	3.600000	107.0	400	0.0086	77.5	147.00	100	140	10.70
53	7.000000	154.0	376	0.0063	90.0	300.00	100	102	10.75
54	3.600000	120.0	318	0.0242	100.0	35.00	100	75	11.00
55	3.600000	107.0	400	0.0286	80.0	17.00	100	54	11.00
56	3.000000	112.0	305	0.0205	80.0	35.00	100	73	11.10
57	6.000000	154.0	322	0.0186	100.0	35.00	100	35	11.10
58	3.600000	107.0	400	0.0226	80.0	35.00	100	88	11.00
59	6.000000	154.0	322	0.0227	110.0	55.00	100	67	11.05
60	6.000000	154.0	322	0.0226	100.0	75.00	100	91	11.10
61	9.500000	164.0	450	0.0131	105.0	145.00	100	90	11.10
62	7.000000	154.0	376	0.0080	113.0	407.00	100	174	11.22
63	8.000000	164.0	379	0.0075	118.0	462.00	100	165	11.22
64	9.500000	164.0	450	0.0072	105.0	295.00	100	100	11.40
65	7.000000	154.0	376	0.0119	100.0	131.00	100	84	11.06
66	7.600000	159.0	383	0.0122	115.5	142.00	100	87	11.15
67	3.600000	120.0	318	0.0182	90.0	67.00	100	108	11.15
68	3.600000	107.0	400	0.0164	83.5	28.00	100	51	11.15
69	3.600000	120.0	318	0.0128	90.0	45.00	100	51	11.15
70	3.600000	107.0	400	0.0180	98.0	80.00	100	160	10.80
71	3.000000	90.0	472	0.0284	98.0	13.00	100	58	10.62

	k	Average wind speed	Nrowscale parameter	Nturb_effective \
0	2.4	9.529683	2.5	60
1	2.4	9.529683	2.5	70
2	2.4	9.751303	2.5	72
5	2.4	9.574007	2.5	72
6	2.4	9.618331	2.5	90
7	2.4	9.308062	2.5	48
8	2.4	9.751303	2.5	111
9	2.4	10.194544	2.5	80
10	2.4	10.194544	2.5	91
11	2.4	10.194544	2.5	49
12	2.4	10.017248	2.5	80
13	2.4	10.061572	2.5	72
14	2.4	9.981789	2.5	80
15	2.4	9.928600	2.5	80
16	2.4	9.928600	2.5	38
17	2.4	9.928600	2.5	48
18	2.4	9.928600	2.5	80

19	2.4	9.946329	2.5	80
20	2.4	9.946329	2.5	16
21	2.4	9.946329	2.5	71
22	2.4	9.928600	2.5	31
23	2.4	9.946329	2.5	67
24	2.4	9.946329	2.5	80
25	2.4	9.884276	2.5	97
26	2.4	9.884276	2.5	54
27	2.4	9.946329	2.5	78
28	2.4	9.946329	2.5	56
29	2.4	9.946329	2.5	72
30	2.4	9.946329	2.5	66
31	2.4	9.946329	2.5	12
32	2.4	9.751303	2.5	30
33	2.4	9.946329	2.5	150
34	2.4	9.485359	2.5	89
37	2.4	9.396710	2.5	94
38	2.4	9.396710	2.5	44
39	2.4	9.396710	2.5	54
40	2.4	9.396710	2.5	42
41	2.4	9.396710	2.5	72
42	2.4	9.396710	2.5	50
43	2.4	9.396710	2.5	56
44	2.4	9.396710	2.5	23
45	2.4	9.396710	2.5	28
46	2.4	9.396710	2.5	30
47	2.4	9.361251	2.5	116
48	2.4	9.396710	2.5	100
49	2.4	9.370116	2.5	175
50	2.4	9.414440	2.5	48
51	2.4	9.485359	2.5	56
52	2.4	9.485359	2.5	140
53	2.4	9.529683	2.5	102
54	2.4	9.751303	2.5	75
55	2.4	9.751303	2.5	54
56	2.4	9.839951	2.5	73
57	2.4	9.839951	2.5	35
58	2.4	9.751303	2.5	88
59	2.4	9.795627	2.5	67
60	2.4	9.839951	2.5	91
61	2.4	9.839951	2.5	90
62	2.4	9.946329	2.5	174
63	2.4	9.946329	2.5	165
64	2.4	10.105896	2.5	100
65	2.4	9.804492	2.5	84
66	2.4	9.884276	2.5	87
67	2.4	9.884276	2.5	108
68	2.4	9.884276	2.5	51

69	2.4	9.884276	2.5	51
70	2.4	9.574007	2.5	160
71	2.4	9.414440	2.5	58

	Nturb_frontal_area	CF	Wind farm density \
0	10.0	40.965427	10.41
1	9.0	38.904110	10.84
2	12.0	44.188520	3.38
5	12.0	35.637120	7.53
6	19.0	42.969617	5.91
7	9.0	31.020449	23.00
8	21.9	48.477471	4.47
9	15.0	39.725597	8.00
10	19.0	46.628115	6.34
11	11.0	49.367920	4.62
12	15.0	47.598663	4.44
13	15.2	48.896499	4.80
14	14.0	46.666244	6.86
15	15.0	39.986738	9.44
16	4.0	31.209111	19.66
17	4.0	33.648840	8.20
18	12.0	43.244229	7.01
19	7.0	34.446347	9.76
20	2.0	40.234630	10.18
21	9.0	37.617256	11.83
22	7.0	42.808219	11.15
23	6.0	42.756158	7.85
24	6.0	37.813927	7.10
25	8.0	38.102187	7.94
26	9.0	37.045442	8.04
27	2.0	33.478223	8.67
28	9.0	33.282598	12.50
29	6.5	34.317216	7.25
30	5.0	41.548854	8.43
31	1.0	35.388128	15.00
32	7.0	37.377288	18.90
33	26.0	45.460426	8.57
34	12.0	37.659003	10.93
37	8.0	41.006833	5.86
38	6.0	38.596816	8.41
39	11.0	32.935253	16.38
40	6.0	35.465709	13.60
41	13.0	37.963668	14.90
42	8.0	39.815968	8.33
43	9.0	33.371311	10.06
44	3.0	31.483914	18.25
45	6.0	38.132712	14.07
46	4.0	34.714672	12.90

47	16.0	42.258847	5.56
48	17.0	32.821098	8.57
49	20.0	40.722621	5.16
50	9.0	35.470248	12.71
51	5.0	46.017825	2.03
52	12.0	40.234372	3.43
53	16.0	45.267727	2.38
54	8.0	41.335476	7.71
55	8.0	34.851817	11.41
56	10.0	42.697452	6.26
57	8.0	47.525860	6.00
58	16.0	37.880425	9.06
59	12.0	45.624238	7.31
60	15.0	43.606111	7.64
61	16.0	40.174123	5.91
62	20.0	45.430135	2.99
63	16.0	42.227559	3.00
64	12.0	40.494288	3.22
65	13.0	43.462754	4.49
66	13.0	45.227032	4.64
67	12.0	44.286107	5.81
68	4.0	39.025842	6.56
69	5.0	44.191637	4.08
70	8.0	34.058298	7.20
71	12.0	35.513285	13.38

	Year production TWh/year	Source power	Coordinates \
0	1.38	Germany sheet	54°46 59 N 14°07 16 E
1	1.19	Germany sheet	54°50 2 N 14°4 5 E
2	2.34	Danish wind farms sheet	55°04 0 N 13°01 0 E
5	0.52	Danish wind farms sheet	54°33 0 N 11°42 36 E
6	0.78	Danish wind farms sheet	54°33 36 N 11°33 0 E
7	0.30	Jens paper	55°31 0 N 12°47 0 E
8	1.70	Danish wind farms sheet	56°36 00 N 11°12 36 E
9	0.56	Danish wind farms sheet	55°31 47 N 7°54 22 E
10	0.85	Danish wind farms sheet	55°36 00 N 7°35 24 E
11	1.76	Danish wind farms sheet	55°42 00 N 7°41 20 E
12	1.20	Germany sheet	55°8 24 N 7°12 0 E
13	1.23	Germany sheet	55°11 30 N 06°51 30 E
14	1.18	Germany 2nd table sheet	55°00 49 N 7°46 07 E
15	1.06	Germany 2nd table sheet	54°30 00 N 7°41 0 E
16	0.94	Germany 2nd table sheet	54°29 00 N 7°41 0 E
17	0.87	Germany 2nd table sheet	54°26 00 N 7°41 0 E
18	1.09	Germany 2nd table sheet	54°23 00 N 7°41 0 E
19	1.21	Germany 2nd table sheet	54°30 0 N 6°21 30 E
20	0.39	Germany 2nd table sheet	54°29 1 N 6°15 8 E
21	1.64	Germany sheet	54°26 0 N 6°19 0 E
22	0.95	Germany 2nd table sheet	54°18 18 N 5°47 56 E

23	1.51	Germany 2nd table sheet	54°19 00 N 5°52 0 E
24	1.33	Germany sheet	54°21 18 N 5°58 48 E
25	1.94	Germany sheet	54°03 00 N 07°01 00 E
26	1.08	Germany 2nd table sheet	53°58 44 N 6°48 50 E
27	0.92	Germany sheet	53°58 01 N 6°33 14 E
28	1.31	Germany sheet	53°58 0 N 6°29 44 E
29	1.21	Germany 2nd table sheet	54°2 30 N 6°28 0 E
30	1.44	Germany 2nd table sheet	54°2 0 N 6°33 0 E
31	0.19	Germany 2nd table sheet	54°0 30 N 6°35 54 E
32	0.37	Germany 2nd table sheet	53°41 0 N 6°29 00 E
33	2.39	Dutch sheet	54°02 00 N 5°57 47 E
34	1.26	Dutch sheet	53°0 0 N 5°16 0 E
37	2.70	Dutch sheet	51°38 24 N 3°4 12 E
38	1.25	Belgium wind farms sheet	51°31 41 N 3°01 55 E
39	0.94	Belgium wind farms sheet	51°33 06 N 2°58 01 E
40	0.96	Belgium wind farms sheet	51°35 28 N 2°56 38 E
41	0.72	Belgium wind farms sheet	51°37 00 N 2°54 00 E
42	0.58	Belgium wind farms sheet	51°39 47 N 2°50 02 E
43	0.50	Belgium wind farms sheet	51°40 0 N 2°48 0 E
44	0.60	Belgium wind farms sheet	51°41 15 N 2°44 56 E
45	0.79	Source files Belgium	51°42 15 N 2°40 56 E
46	0.77	Source files Belgium	51°38 47 N 2°52 00 E
47	1.48	Uk wind farms sheet	50°40 00 N 0°16 00 W
48	0.86	Uk wind farms sheet	51°26 0 N 1°38 0 E
49	2.25	Uk wind farms sheet	51°38 38 N 1°33 13 E
50	0.54	Uk wind farms sheet	51°43 0 N 01°12 50 E
51	1.42	Uk wind farms sheet	51°52 48 N 2°2 24 E
52	1.78	Uk wind farms sheet	51°56 0 N 1°53 0 E
53	2.83	Uk wind farms sheet	52°14 04 N 02°29 18 E
54	0.98	Uk wind farms sheet	53°11 0 N 0°29 0 E
55	0.59	Uk wind farms sheet	53°07 39 N 0°26 10 E
56	0.82	Uk wind farms sheet	53°38 38 N 0°17 35 E
57	0.87	Uk wind farms sheet	53°48 18 N 0°8 56 E
58	1.05	Uk wind farms sheet	53°07 0 N 1°08 0 E
59	1.61	Uk wind farms sheet	53°15 00 N 1°23 00 E
60	2.19	Uk wind farms sheet	53°16 30 N 0°50 30 E
61	3.02	Uk wind farms sheet	53°30 00 N 0°48 0 E
62	4.85	Uk wind farms sheet	53°53 06 N 01°47 28 E
63	5.13	Uk wind farms sheet	53°55 12 N 01°33 36 E
64	3.37	Uk wind farms sheet	58°25 17 N 2°40 12 W
65	2.24	Uk wind farms sheet	58°30 17 N 02°50 12 W
66	2.61	Uk wind farms sheet	54°5 17 N 3°44 17 W
67	1.51	Uk wind farms sheet	53°54 00 N 3°28 0 W
68	0.63	Uk wind farms sheet	53°58 38 N 3°27 27 W
69	0.71	Uk wind farms sheet	54°4 52 N 3°36 18 W
70	1.72	Uk wind farms sheet	53°27 00 N 3°35 00 W
71	0.54	Uk wind farms sheet	54°45 0 N 3°43 12 W

	k_ref	A_ref	A_h	lat	lon	lat2 \
0	2.387553	10.739125	10.754518	54.783056	14.121111	54.781563
1	2.395581	10.765232	10.745505	54.833889	14.068056	54.836027
2	2.513793	10.873473	10.926723	55.066667	13.016667	55.030457
5	2.537728	10.654426	10.368264	54.550000	11.710000	54.547282
6	2.540336	10.646139	10.348950	54.560000	11.550000	54.556884
7	2.638001	10.241995	9.922638	55.516667	12.783333	55.512450
8	2.541026	10.816705	10.657501	56.600000	11.210000	56.601088
9	2.519353	11.176193	10.887657	55.529722	7.906111	55.486143
10	2.498294	11.289453	10.974306	55.600000	7.590000	55.599856
11	2.498944	11.252025	11.344326	55.700000	7.688889	55.694559
12	2.461537	11.316985	11.212270	55.140000	7.200000	55.140636
13	2.455438	11.363826	11.319901	55.191667	6.858333	55.201313
14	2.514646	11.230700	11.154034	55.013611	7.768611	55.016466
15	2.496547	11.184042	11.098750	54.500000	7.683333	54.523005
16	2.497592	11.178425	11.236941	54.483333	7.683333	54.488739
17	2.499364	11.172867	11.105434	54.433333	7.683333	54.444152
18	2.500722	11.161172	11.067027	54.383333	7.683333	54.389863
19	2.422494	11.266523	11.104686	54.500000	6.358333	54.508450
20	2.417010	11.256435	11.296188	54.483611	6.252222	54.487150
21	2.421081	11.250338	11.290069	54.433333	6.316667	54.439228
22	2.392638	11.205309	11.205309	54.305000	5.798889	54.304023
23	2.395366	11.209750	11.236094	54.316667	5.866667	54.318512
24	2.397256	11.215347	11.129816	54.355000	5.980000	54.356893
25	2.476727	11.127989	11.159580	54.050000	7.016667	54.042806
26	2.455207	11.093008	11.008410	53.978889	6.813889	53.980850
27	2.438148	11.108187	10.958371	53.966944	6.553889	53.967360
28	2.434529	11.110178	11.149414	53.966667	6.495556	53.951166
29	2.435233	11.145977	11.060975	54.041667	6.466667	54.048682
30	2.438502	11.133243	11.133243	54.033333	6.550000	54.044273
31	2.445058	11.118479	11.051375	54.008333	6.598333	54.010950
32	2.474616	10.849664	10.766922	53.683333	6.483333	53.692403
33	2.400893	11.134432	11.040513	54.033333	5.963056	54.035988
34	2.429959	10.282930	10.386955	53.000000	5.266667	53.001577
37	2.344485	10.503001	10.615834	51.640000	3.070000	51.691284
38	2.373847	10.422875	10.407633	51.528056	3.031944	51.525900
39	2.369521	10.477530	10.438629	51.551667	2.966944	51.550354
40	2.351213	10.523432	10.655934	51.591111	2.943889	51.591007
41	2.345420	10.560596	10.298796	51.616667	2.900000	51.619964
42	2.340552	10.611089	10.430041	51.663056	2.833889	51.660762
43	2.341836	10.619710	10.367196	51.666667	2.800000	51.669382
44	2.338124	10.636853	10.674417	51.687500	2.748889	51.686739
45	2.340222	10.652517	10.718964	51.704167	2.682222	51.717400
46	2.344169	10.588842	10.654893	51.646389	2.866667	51.633690
47	2.321773	10.378586	10.210954	50.666667	-0.266667	50.669666
48	2.476739	10.362015	10.094499	51.433333	1.633333	51.430405
49	2.472514	10.380842	10.276202	51.643889	1.553611	51.627876
50	2.520673	10.191404	9.979188	51.716667	1.213889	51.730312

51	2.394744	10.696440	10.719325	51.880000	2.040000	51.898179
52	2.429959	10.647010	10.450576	51.933333	1.883333	51.877201
53	2.307353	10.784472	10.702227	52.234444	2.488333	52.235929
54	2.516885	10.650983	10.650983	53.183333	0.483333	53.191321
55	2.557527	10.644482	10.472556	53.127500	0.436111	53.162822
56	2.550658	10.732394	10.559048	53.643889	0.293056	53.643863
57	2.545051	10.695136	10.695136	53.805000	0.148889	53.805449
58	2.489047	10.744247	10.570709	53.116667	1.133333	53.135605
59	2.490241	10.949546	11.025084	53.250000	1.383333	53.266773
60	2.488675	10.898281	10.898281	53.275000	0.841667	53.275854
61	2.460814	11.003864	11.042724	53.500000	0.800000	53.476976
62	2.338721	11.279821	11.379607	53.885000	1.791111	53.883019
63	2.372521	11.311084	11.446595	53.920000	1.560000	53.937505
64	2.264115	11.431012	11.471381	58.421389	-2.670000	58.182538
65	2.251597	11.291821	11.291821	58.504722	-2.836667	58.256933
66	2.221992	11.089158	11.204822	54.088056	-3.738056	54.089652
67	2.188498	10.964201	10.880585	53.900000	-3.466667	53.984957
68	2.185346	10.882347	10.740308	53.977222	-3.457500	54.041425
69	2.223327	10.968746	10.885096	54.081111	-3.605000	54.079912
70	2.025876	10.446958	10.431681	53.450000	-3.583333	53.458452
71	2.038650	10.570114	10.554658	54.750000	-3.720000	54.753002

	lon2	Area	Nwt_rows	Nwt_row_wf_neighbors	Cf_Free \
0	14.121575	37.099754	11.362100	9.728200	0.614928
1	14.067697	32.570127	13.269900	11.375400	0.596660
2	12.937307	223.595267	17.822682	15.809881	0.591075
5	11.713478	23.210880	15.000000	12.144300	0.548197
6	11.543768	31.781361	19.163784	17.951405	0.609996
7	12.780823	4.673805	9.620338	9.620338	0.587615
8	11.216611	90.371733	32.481652	32.481652	0.632695
9	7.840551	19.610402	14.976898	14.976898	0.612940
10	7.584526	31.420046	15.902300	14.469400	0.648015
11	7.655767	83.114136	12.131100	10.550500	0.623879
12	7.199689	65.210311	21.581100	21.581100	0.649939
13	6.852010	54.952476	19.474800	19.474800	0.664063
14	7.774015	31.353004	14.922808	14.922808	0.647695
15	7.706249	30.837802	15.645000	11.653400	0.633835
16	7.691582	17.142322	9.935506	4.524948	0.587636
17	7.675727	35.111435	13.320932	9.043104	0.544511
18	7.701282	39.460599	19.845815	16.427957	0.644298
19	6.364759	39.676923	14.516000	9.386600	0.555780
20	6.243594	10.172046	5.610661	4.307031	0.609167
21	6.332675	40.332338	19.109878	15.827834	0.609167
22	5.789471	16.963183	6.967700	6.150800	0.603400
23	5.874703	53.145265	14.576842	10.138686	0.642904
24	5.982337	56.703199	19.488400	12.710200	0.579567
25	6.992954	73.952414	15.848370	14.158132	0.638889
26	6.816552	27.074879	12.378100	10.326200	0.539313

27	6.554132	35.675918	14.000000	2.178300	0.622658
28	6.496184	81.601280	15.712600	14.083600	0.607435
29	6.461114	56.028093	13.543900	10.258000	0.594782
30	6.555739	39.599846	11.622962	3.745475	0.631582
31	6.606525	3.910719	4.892100	1.794900	0.575725
32	6.475443	5.976698	11.000000	11.000000	0.622033
33	5.963184	103.271601	22.021300	22.021300	0.656928
34	5.263472	32.602076	14.520352	14.520352	0.610533
37	3.068709	132.858419	19.940406	13.731927	0.576630
38	3.012489	35.458318	10.385300	7.846800	0.568120
39	2.950319	27.943224	9.547500	5.121600	0.509457
40	2.945012	22.551463	7.891711	3.772623	0.558385
41	2.902540	12.951521	12.115400	6.893000	0.618056
42	2.820990	32.871011	13.161700	8.335000	0.596616
43	2.802627	12.072949	13.338668	1.466593	0.515198
44	2.753222	9.618695	8.164284	4.769823	0.526831
45	2.737943	15.597665	8.059494	7.335966	0.565160
46	2.861963	16.624977	8.812081	4.857286	0.565160
47	-0.278316	61.240729	12.483200	12.483200	0.599893
48	1.632150	26.911454	15.769700	15.769700	0.515198
49	1.497255	106.410281	15.377062	15.377062	0.606248
50	1.231523	14.784066	11.345000	11.345000	0.556282
51	2.039241	121.911027	12.962796	8.647665	0.600815
52	1.940524	149.861551	16.969400	10.789000	0.561639
53	2.503320	139.175041	10.184400	10.184400	0.579803
54	0.492165	35.294653	18.335766	11.137786	0.632695
55	0.448374	22.731712	9.475800	7.861800	0.581129
56	0.292836	20.699850	11.306361	11.306361	0.647991
57	0.149434	32.528982	11.000000	11.000000	0.635990
58	1.147436	32.180785	17.000000	17.000000	0.581129
59	1.377519	50.491273	17.332467	17.332467	0.633064
60	0.842154	57.126338	18.032403	18.032403	0.635990
61	0.839283	132.001242	24.583942	23.335466	0.560172
62	1.919224	405.508811	27.680732	20.987001	0.609167
63	1.659283	386.655163	23.703230	19.896410	0.607435
64	-2.724090	227.193470	16.395500	13.109100	0.579004
65	-2.884790	87.749763	11.589000	8.404300	0.599432
66	-3.752416	107.542595	17.166600	13.895100	0.600790
67	-3.464133	60.396425	18.887700	15.406700	0.641438
68	-3.510825	23.408841	12.502800	4.343200	0.590524
69	-3.600135	39.095777	12.122400	4.175400	0.641438
70	-3.585501	67.312445	17.772200	17.772200	0.568240
71	-3.713076	10.071244	9.553100	9.553100	0.516589

	Cf_Inf	Cf_Model	U_Free	U_Inf	U_Model	G_Model	U_rated \
0	0.367128	0.439834	9.529683	6.792090	7.932753	11.494173	10.456476
1	0.338103	0.421211	9.529683	6.705698	7.613408	11.534182	10.742769
2	0.475114	0.523431	9.751303	8.311967	8.911690	11.735533	11.082324

5	0.346439	0.430505	9.574007	7.301068	8.248126	11.908577	11.562421
6	0.406472	0.513887	9.618331	7.280228	8.514226	11.981060	10.631700
7	0.203492	0.383550	9.308062	5.381286	7.221962	11.611896	10.631700
8	0.462214	0.546303	9.751303	7.678807	8.701052	11.984191	10.415067
9	0.391783	0.495450	10.194544	7.533524	8.780877	12.716600	11.219291
10	0.438359	0.547795	10.194544	7.594705	8.951764	12.745788	10.631700
11	0.491099	0.545174	10.194544	8.455699	9.431581	12.260785	11.036114
12	0.489076	0.564481	10.017248	7.958301	8.923432	12.259757	10.415067
13	0.495501	0.584464	10.061572	7.881938	9.032300	12.246620	10.226802
14	0.435961	0.528595	9.981789	7.416373	8.538742	12.181878	10.415067
15	0.382318	0.500879	9.928600	7.018547	8.382634	12.123237	10.585836
16	0.285348	0.364897	9.928600	6.562387	7.448233	11.958341	11.340144
17	0.363756	0.401413	9.928600	7.792737	8.237709	12.102601	12.052026
18	0.428289	0.509293	9.928600	7.341405	8.311603	12.133753	10.415067
19	0.336137	0.384184	9.946329	7.399013	7.956238	12.234590	11.885937
20	0.412401	0.473890	9.946329	7.597423	8.331456	12.002682	11.007810
21	0.350684	0.432598	9.946329	6.991584	7.927947	12.002682	11.007810
22	0.367668	0.517434	9.928600	7.203894	8.742034	12.024931	11.082324
23	0.423936	0.472959	9.946329	7.328202	7.914350	12.017730	10.456476
24	0.390966	0.426328	9.946329	7.708860	8.128385	12.146262	11.492968
25	0.412323	0.459038	9.884276	7.216149	7.766278	11.931813	10.456476
26	0.357863	0.437123	9.884276	7.751316	8.640049	12.065684	12.084600
27	0.385645	0.400838	9.946329	7.184170	7.361231	12.223018	10.787345
28	0.347643	0.450015	9.946329	6.980300	8.172008	12.002682	11.036114
29	0.399236	0.441786	9.946329	7.625122	8.149005	12.146262	11.243218
30	0.406873	0.449432	9.946329	7.290865	7.793794	12.047770	10.641552
31	0.352990	0.399393	9.946329	7.363149	7.901311	12.125587	11.556203
32	0.291904	0.484480	9.751303	6.185740	8.265652	11.893120	10.585836
33	0.398150	0.510287	9.946329	6.925746	8.234665	12.156798	10.226802
34	0.347496	0.436160	9.485359	6.624903	7.589102	11.332754	10.476333
37	0.403545	0.440371	9.396710	7.437340	7.854227	11.212714	10.903546
38	0.367857	0.414911	9.396710	7.175686	7.932853	11.358573	11.036114
39	0.248460	0.381375	9.396710	6.540534	7.995068	11.385868	11.965520
40	0.309044	0.397708	9.396710	6.702986	7.665030	11.190943	11.188298
41	0.274577	0.429620	9.396710	5.843100	7.447160	11.647819	10.262279
42	0.364402	0.457288	9.396710	6.855712	7.872111	11.550430	10.593546
43	0.298352	0.378715	9.396710	7.003852	7.965269	11.634971	11.873009
44	0.268643	0.352029	9.396710	6.594051	7.507962	11.298452	11.686758
45	0.312900	0.448421	9.396710	6.676318	8.133671	11.266129	11.082324
46	0.322271	0.403234	9.396710	6.765876	7.642821	11.266129	11.082324
47	0.403025	0.471146	9.361251	7.159216	7.918538	11.492747	10.503074
48	0.305045	0.394360	9.396710	7.072296	8.060172	11.660877	11.873009
49	0.414942	0.469601	9.370116	7.212615	7.829044	11.429280	10.415067
50	0.289163	0.414375	9.414440	6.542841	7.888903	11.620905	11.242435
51	0.516611	0.535103	9.485359	8.427424	8.663570	11.428738	10.627924
52	0.431077	0.459055	9.485359	7.953916	8.282082	11.683916	11.242435
53	0.481595	0.520108	9.529683	8.327643	8.799031	11.605825	11.007810
54	0.410197	0.469530	9.751303	7.167297	7.856365	11.796673	10.415067

55	0.326432	0.421632	9.751303	6.904026	7.958573	12.002762	11.242435
56	0.438668	0.510354	9.839951	7.333809	8.192077	12.118846	10.262279
57	0.461270	0.561110	9.839951	7.699693	8.922698	11.910771	10.456476
58	0.348719	0.454073	9.751303	7.121343	8.316779	12.002762	11.242435
59	0.425937	0.518680	9.795627	7.347659	8.443764	11.767349	10.456476
60	0.419509	0.484747	9.839951	7.285307	8.338045	11.910771	10.456476
61	0.401411	0.470870	9.839951	7.948946	8.789393	11.866195	11.686758
62	0.496595	0.528943	9.946329	8.496328	8.912995	11.935440	11.007810
63	0.500602	0.501430	9.946329	8.564111	8.899194	11.896131	11.036114
64	0.473719	0.505304	10.105896	8.749057	9.156109	12.207577	11.686758
65	0.452389	0.509281	9.804492	8.010823	8.704802	11.865125	11.007810
66	0.456708	0.512236	9.884276	8.106183	8.770413	11.836489	11.075292
67	0.445342	0.499556	9.884276	7.509050	8.168835	12.065684	10.415067
68	0.409146	0.444710	9.884276	7.725862	8.149080	12.136246	11.242435
69	0.496010	0.531654	9.884276	8.032515	8.486378	12.065684	10.415067
70	0.366848	0.392022	9.574007	7.299849	7.584119	11.586685	11.242435
71	0.276479	0.400674	9.414440	6.779763	8.142527	11.381372	11.873009

	Phi_Model	Phi_Model_redo	Phi_Model_a5p3	Cf_Model_redo	Cf_Model_a5p3
0	1.341719	1.346730	1.227047	0.437210	0.501830
1	1.432047	1.391041	1.286417	0.443145	0.501891
2	1.249483	1.223521	1.208781	0.538771	0.547545
5	1.414519	1.412626	1.338013	0.431515	0.472459
6	1.265772	1.275949	1.255029	0.507959	0.520174
7	1.505772	1.480956	1.294299	0.395961	0.497342
8	1.210864	1.143330	1.208060	0.586933	0.547975
9	1.297587	1.297864	1.250503	0.495291	0.522831
10	1.208362	1.252430	1.196574	0.521700	0.554842
11	1.178622	1.183686	1.135579	0.542239	0.570248
12	1.180513	1.125807	1.147517	0.597564	0.584397
13	1.147405	1.106199	1.120502	0.609484	0.600787
14	1.240715	1.230386	1.185595	0.534701	0.561426
15	1.289315	1.335429	1.236039	0.474540	0.532060
16	1.544175	1.522509	1.212635	0.375337	0.545247
17	1.470223	1.380499	1.286173	0.448891	0.502032
18	1.273656	1.222829	1.194161	0.539183	0.556287
19	1.504491	1.471886	1.349026	0.400565	0.466288
20	1.335469	1.213353	1.106721	0.544819	0.609167
21	1.410603	1.298820	1.266833	0.494741	0.513268
22	1.288404	1.316554	1.134868	0.500752	0.611799
23	1.337124	1.277990	1.178457	0.506774	0.565717
24	1.422365	1.349763	1.284979	0.465877	0.502723
25	1.362065	1.298375	1.231154	0.494997	0.534246
26	1.408980	1.388350	1.309307	0.448358	0.492854
27	1.471351	1.468697	1.241930	0.402193	0.527878
28	1.374783	1.270753	1.235561	0.508712	0.529276
29	1.390628	1.344382	1.252981	0.467211	0.519514
30	1.379510	1.399152	1.198865	0.438754	0.553470

31	1.474188	1.403744	1.161856	0.436279	0.575725
32	1.316763	1.130816	1.103130	0.594523	0.611351
33	1.271947	1.301632	1.272210	0.493127	0.510134
34	1.403965	1.369768	1.297900	0.454782	0.495270
37	1.396159	1.348187	1.292876	0.466757	0.498162
38	1.404024	1.365491	1.241947	0.434903	0.502188
39	1.510179	1.666035	1.402960	0.310345	0.436701
40	1.476744	1.543883	1.267365	0.364682	0.512459
41	1.416178	1.561335	1.308632	0.356785	0.489118
42	1.365228	1.358206	1.224056	0.461177	0.538454
43	1.501881	1.660779	1.374951	0.307066	0.444003
44	1.569800	1.468091	1.243707	0.401584	0.525628
45	1.382057	1.327956	1.179383	0.478537	0.565641
46	1.466661	1.433233	1.192470	0.420586	0.557301
47	1.340771	1.367580	1.289860	0.456217	0.500152
48	1.484127	1.497019	1.441165	0.387896	0.416426
49	1.343105	1.365809	1.304317	0.456966	0.491587
50	1.445095	1.383961	1.300971	0.447000	0.493506
51	1.229195	1.206145	1.160991	0.548807	0.575921
52	1.362034	1.367141	1.307859	0.456231	0.489560
53	1.255142	1.279135	1.233035	0.506109	0.533132
54	1.343232	1.302199	1.210761	0.492801	0.546364
55	1.432895	1.435995	1.272218	0.419999	0.511181
56	1.271832	1.255805	1.173810	0.519719	0.568515
57	1.186121	1.124183	1.092543	0.598550	0.617795
58	1.370535	1.358593	1.324605	0.460672	0.479724
59	1.257577	1.188149	1.187885	0.559893	0.560051
60	1.274644	1.243840	1.221817	0.501932	0.514349
61	1.338881	1.282219	1.306904	0.503144	0.488962
62	1.240124	1.237421	1.218382	0.530540	0.541826
63	1.244266	1.233583	1.213583	0.507437	0.518745
64	1.280522	1.275494	1.239147	0.508224	0.529520
65	1.273676	1.308542	1.225797	0.489169	0.537421
66	1.271526	1.265172	1.223846	0.515955	0.540376
67	1.290014	1.263393	1.212463	0.515012	0.545069
68	1.388162	1.382565	1.215139	0.447762	0.543755
69	1.235535	1.245498	1.115316	0.525776	0.603939
70	1.488772	1.429084	1.376226	0.422773	0.451232
71	1.471673	1.522483	1.390245	0.375350	0.443578

```
[107]: # we can now read the input parameters for all wind farm cases
latitude = db_first_eval["lat"].values
Windfarm_area = db_first_eval["A"].values*1e6
Windfarm_N_turbines = db_first_eval["Nt"].values
Windfarm_rated_power = db_first_eval["MW install"].values*1e6
Turbine_hub_height = db_first_eval["Ht"].values
Turbine_rotor_diameter = db_first_eval["D"].values
```

```

Turbine_rated_power = db_first_eval["PG"].values*1e6
Wind_lambda = db_first_eval["lambda"].values
Wind_k= db_first_eval["k"].values
Wind_reference_height = db_first_eval["Href"].values

# correction factor for the Nfree turbines
NFree = np.array(db_first_eval['Nrowscale parameter'] *
    ↪db_first_eval['Nturb_frontal_area'], dtype=float)
correction_factor = NFree / np.sqrt(Windfarm_N_turbines)

```

3.5 Here, we define the fixed input parameters, as defined by vdLW and the ones that need correction

```

[108]: # parameters as defined by vdLW, these are the same for all cases, as they are
    ↪not case specific,
# but they are used as input for the predict function, so we define them here
vdLW_z0 = 10 ** (-4) # Roughness length [m]
vdLW_zRef = 100.0 # Reference height at which GWA data is taken [m]
vdLW_kw = 2.4 # Weibull shape parameter [-]
vdLW_rho = 1.225 # Air density [kg/m^3]
vdLW_lat = 55 # Latitude [degree]
vdLW_CP = 0.46 # Turbine power coefficient [-]
vdLW_CT = 0.75 # Turbine thrust coefficient [-]
vdLW_Uin = 0.0 # Cut-in wind speed [m/s]
vdLW_Uout = 1e6 # Cut-out wind speed [m/s]

# correct inputs by SLS, as used in the original paper by SLS
SLS_Uin = 3.0 # Cut-in wind speed [m/s]
SLS_Uout = 25.0 # Cut-out wind speed [m/s]
SLS_lat = latitude # Latitude [degrees] , it is wind farm specific

```

3.6 Correcting Type 1 error 1 : removal by vdLW of the hub-height wind-speed correction of the Weibull scale parameter, causing a different input of the local unperturbed wind speed

The reference λ (scale factor of the Weibull distribution) is read at a reference height. It need to be corrected to the hub height. vdLW originally also did this, but then it was “commented out” of their code. This is a very important error. To correct it, its a single line of code

```

[109]: # correction of Type 1 error 1: removal by vdLW of the hub-height wind-speed
    ↪correction of the Weibull scale parameter, causing a different input of the
    ↪local unperturbed wind speed

Wind_lambda_corrected_to_hub_height = Wind_lambda * np.log(Turbine_hub_height /
    ↪vdLW_z0) / np.log(Wind_reference_height / vdLW_z0)

```

3.7 Calculation of all wind farm cases: original vdLW formulation, and then with the errors corrected

3.7.1 calculation of the original vdLW predictions for all cases

```
[110]: # calculation of the original vdLW predictions for all cases, using the
        ↪original inputs as given by vdLW, and using Nfree as determined by SLS
Cf_Isolated_vdLW, Cf_InfWindFarm_vdLW, Cf_Model_vdLW
        ↪=run_code_all_cases(Turbine_rated_power, vdLW_CT, Turbine_rotor_diameter,
        ↪Turbine_hub_height, vdLW_z0, Wind_lambda, vdLW_kw, Windfarm_N_turbines,
        ↪Windfarm_area,
                                correction_factor, vdLW_lat , vdLW_CP, vdLW_Uin,
        ↪vdLW_Uout, vdLW_rho, correct_eps2=False)
```

3.7.2 Check that this generates the results published by vdLW, to be sure that all is correct

First, we will read from the dataframe of “Output.csv” (the output file generated by vdLW) the results by vdLW, and then we compare with the results published above

```
[111]: # results calculated and published by vdLW
vdLW_Cf_model_published = np.array(db_first_eval["Cf_Model"].values,
        ↪dtype=float)

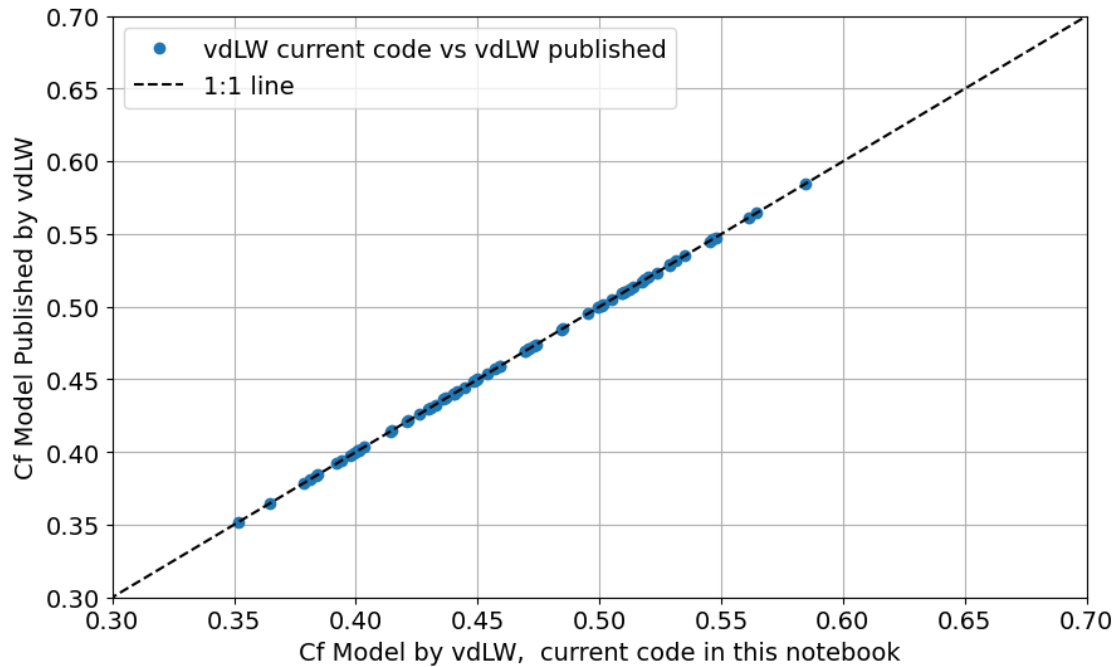
# we will plot the results of the original vdLW predictions against the
        ↪published results by vdLW, to check if we can reproduce their results, and
        ↪to see if there are any discrepancies
plt.figure(figsize=(10, 6))

# the Cf_model is multiplied by the ratio of the total rated power of the wind
        ↪farm to the rated power of the turbine, to get the same units as the
        ↪published results by vdLW,
# for 6 wind farms, there is a difference between the value of nameplate
        ↪capacity of the wind farm and the product of rated wind turbine power and
        ↪number of turbines (rounding and typos)
# in the code of vdLW, they use the nameplate capacity of rated power of the
        ↪wind farm, not N turbines*Turbine rated power
# the correct approach is to always use the number of turbines and the rated
        ↪power of each turbine,
# as in a few times, the announced nameplate capacity of the wind farm is
        ↪larger than the product of number of turbines and rated power of each turbine
# for the purpose of checking the code, this has no impact.

plt.plot(Cf_Model_vdLW*(Windfarm_N_turbines*Turbine_rated_power)/
        ↪Windfarm_rated_power, vdLW_Cf_model_published, 'o', label='vdLW current code
        ↪vs vdLW published')
# plot a 1:1 line for reference
```



```
plt.plot([0, 1], [0, 1], 'k--', label='1:1 line')
plt.xlabel('Cf Model by vdLW, current code in this notebook')
plt.ylabel('Cf Model Published by vdLW')
plt.xlim(0.3, 0.7)
plt.ylim(0.3, 0.7)
plt.grid(True)
plt.legend()
plt.show()
```



3.7.3 calculations with the Type 1 errors corrected

```
[112]: ### calculations with the Type 1 errors corrected

# correct TType 1 error 1: correction of the hub-height wind-speed correction of
↳ the Weibull scale parameter,
# causing a different input of the local unperturbed wind speed,
Cf_Free_vdLW_correct_1, Cf_Inf_vdLW_correct_1, Cf_Model_vdLW_correct_1
↳ run_code_all_cases(Turbine_rated_power,
                      vdLW_CT, Turbine_rotor_diameter, Turbine_hub_height, vdLW_z0,
↳ Wind_lambda_corrected_to_hub_height,
                      vdLW_kw, Windfarm_N_turbines, Windfarm_area,
                      correction_factor, vdLW_lat, vdLW_CP, vdLW_Uin, vdLW_Uout,
↳ vdLW_rho, correct_eps2=False)
```

```

#correct Type 1 error 1 and 2: correction of the cut-in and cut-out wind
↳speeds, using the correct cut-in and cut-out wind speeds as defined by SLS,
Cf_Free_vdLW_correct_1_and_2, Cf_Inf_vdLW_correct_1_and_2,
↳Cf_Model_vdLW_correct_1_and_2 =run_code_all_cases(Turbine_rated_power,
vdLW_CT, Turbine_rotor_diameter, Turbine_hub_height,
↳vdLW_z0, Wind_lambda_corrected_to_hub_height, vdLW_kw,
Windfarm_N_turbines, Windfarm_area,
correction_factor, vdLW_lat , vdLW_CP, SLS_Uin, SLS_Uout,
↳vdLW_rho, correct_eps2=False)

#correct Type 1 errors 1,2 and 3: using correct latitude for each wind farm case
Cf_Free_vdLW_correct_1_and_2_and_3, Cf_Inf_vdLW_correct_1_and_2_and_3,
↳Cf_Model_vdLW_correct_1_and_2_and_3 =run_code_all_cases(
Turbine_rated_power, vdLW_CT, Turbine_rotor_diameter, Turbine_hub_height,
↳vdLW_z0, Wind_lambda_corrected_to_hub_height,
vdLW_kw, Windfarm_N_turbines, Windfarm_area,
correction_factor, SLS_lat , vdLW_CP, SLS_Uin, SLS_Uout, vdLW_rho,
↳correct_eps2=False)

# correct TType 1 errors 1,2,3 and 4: correction of the eps2 formulation, using
↳the correct formulation for eps2 as defined by SLS, which is based on the
↳cut-out wind speed instead of the rated wind speed, as used by vdLW
Cf_Free_vdLW_correct_all, Cf_Inf_vdLW_correct_all, Cf_Model_vdLW_correct_all
↳=run_code_all_cases(Turbine_rated_power,
vdLW_CT, Turbine_rotor_diameter, Turbine_hub_height,
↳vdLW_z0, Wind_lambda_corrected_to_hub_height,
vdLW_kw, Windfarm_N_turbines, Windfarm_area,
correction_factor, SLS_lat, vdLW_CP, SLS_Uin, SLS_Uout,
↳vdLW_rho, correct_eps2=True)

# corrected formulation, by limit case with Uin=0 and Uout=infinity
Cf_Free_vdLW_correct_Uin0_Uoutinf, Cf_Inf_vdLW_correct_Uin0_Uoutinf,
↳Cf_Model_vdLW_correct_Uin0_Uoutinf =run_code_all_cases(Turbine_rated_power,
vdLW_CT, Turbine_rotor_diameter, Turbine_hub_height,
↳vdLW_z0, Wind_lambda_corrected_to_hub_height,
vdLW_kw, Windfarm_N_turbines, Windfarm_area,
correction_factor, SLS_lat, vdLW_CP, 0, 10e6, vdLW_rho,
↳correct_eps2=True)

```

3.8 Read the results published by SLS 2026, from comparison and generation of the figures

We already removed the wind farms “Baltic 1”, “Baltic 2”, “Princess Amalia”, “Luchterduinen” because vdLW removed them from their work

```
[113]: # read from excel file the dataframe of SLS_2026
df_SLS_2026 = pd.read_excel("SLS_2026_without_Baltic1_2_LuchD_PAM.xlsx",
    ↪sheet_name="Sheet1")

# createa array of capacity factors from the dataframe of SLS_2026,
# this will be used to compare with the results of the corrected vdLW model
SLS_2026_CF = np.array(df_SLS_2026["Capacity Factor model"].values,
    ↪dtype=float)/100
```

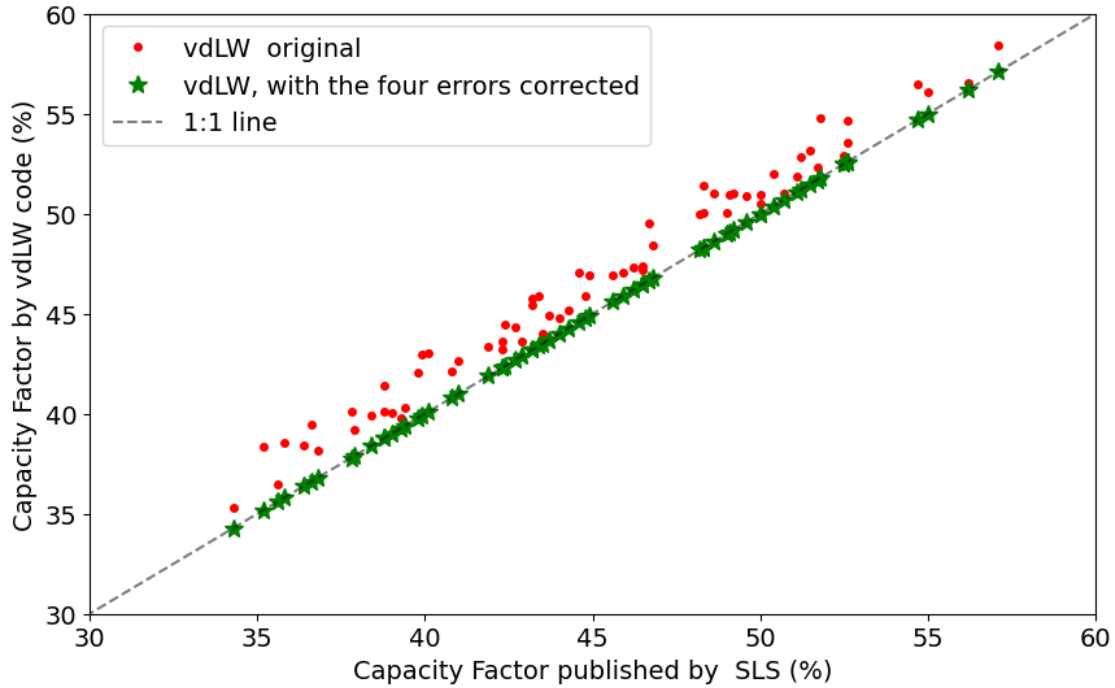
3.9 Plot figure 1

```
[114]: # Plot figure 1
plt.rcParams.update({'font.size': 14}) # fontsize 14 for all labels and ticks

fig, ax = plt.subplots(figsize=(10, 6))

ax.plot(SLS_2026_CF*100, Cf_Model_vdLW*100, 'o', color='red', markersize=4,
    ↪label=r'vdLW original')

ax.plot(SLS_2026_CF*100, np.round(Cf_Model_vdLW_correct_all*100, 1), '*',
    ↪color='green', markersize=10,
        label=r'vdLW, with the four errors corrected')
ax.plot([0., 100.], [0., 100.], "k--", alpha=0.5, label="1:1 line")
ax.set_xlim(30, 60)
ax.set_ylim(30, 60)
ax.set_xlabel('Capacity Factor published by SLS (%)')
ax.set_ylabel('Capacity Factor by vdLW code (%)')
ax.legend()
plt.show()
```



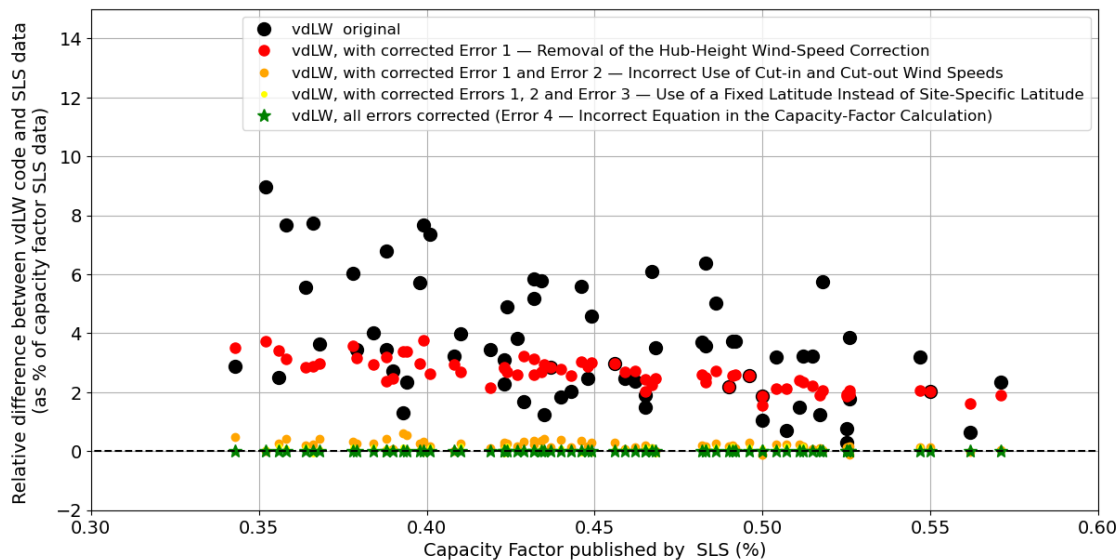
3.10 Plot figure 2

```
[115]: # same, but with 2025 data
fig, ax = plt.subplots(figsize=(14, 7))
ax.plot(SLS_2026_CF, (Cf_Model_vdLW-SLS_2026_CF)/SLS_2026_CF*100, 'o',
        color='black', markersize=10, label=r'vdLW original')
ax.plot(SLS_2026_CF, (Cf_Model_vdLW_correct_1-SLS_2026_CF)/SLS_2026_CF*100,
        'o', color='red', markersize=8,
        label=r'vdLW, with corrected Error 1 - Removal of the Hub-Height
        ↳Wind-Speed Correction')
ax.plot(SLS_2026_CF, (Cf_Model_vdLW_correct_1_and_2-SLS_2026_CF)/
        SLS_2026_CF*100, 'o', color='orange', markersize=6,
        label=r'vdLW, with corrected Error 1 and Error 2 - Incorrect Use of
        ↳Cut-in and Cut-out Wind Speeds')
ax.plot(SLS_2026_CF, (Cf_Model_vdLW_correct_1_and_2_and_3-SLS_2026_CF)/
        SLS_2026_CF*100, 'o', color='yellow', markersize=4,
        label=r'vdLW, with corrected Errors 1, 2 and Error 3 - Use of a Fixed
        ↳Latitude Instead of Site-Specific Latitude')
ax.plot(SLS_2026_CF, (np.round(Cf_Model_vdLW_correct_all, 3)-SLS_2026_CF)/
        SLS_2026_CF*100, '*', color='green', markersize=10,
        label=r'vdLW, all errors corrected (Error 4 - Incorrect Equation in the
        ↳Capacity-Factor Calculation)')
ax.plot([0., 1], [0., 0.], "k--")
```

```

ax.set_xlim(0.3, 0.6)
ax.set_ylim(-2, 15)
ax.legend(fontsize=12)
ax.set_xlabel('Capacity Factor published by SLS (%)')
# ax.set_title('Relative difference between Capacity Factor of Paul\'s model
↳and repository 2025 data \n (in % of capacity factor 2025)')
ax.set_ylabel('Relative difference between vdLW code and SLS data \n (as % of
↳capacity factor SLS data) ')
ax.grid()
# ax.legend()
plt.show()

```



3.11 Create Figure 3

First, we need to read the Nfree correction by vdLW, and then apply it to the correct capacity factors, to obtain the result without Type 1 errors

```

[116]: # get Nedge turbines, calculated by vdLW
Nedge_Neighbours = np.array(db_first_eval['Nwt_row_wf_neighbors'].values,
↳dtype=float)
# vdLW assume a factor of 2.5 to determine the fraction of power of isolated
↳turbines
m_afactor_vdLW = 2.5
a_factor_vdLW_Nfree=m_afactor_vdLW*Nedge_Neighbours/(Windfarm_N_turbines)

# calculate the fraction of power of isolated turbines, as determined by vdLW,
↳using the Nedge_Neighbours and the factor of 2.5

```

```
CF_vdLW_Type1_corrected_NfreevdLW =
↳ a_factor_vdLW_Nfree*Cf_Free_vdLW_correct_all +
↳ (1-a_factor_vdLW_Nfree)*Cf_Inf_vdLW_correct_all
```

[117]: *# simple code for the linear fit*

```
def linear_fit(x, y):
    x = np.asarray(x)
    y = np.asarray(y)

    # fit  $y = a*x + b$ 
    a, b = np.polyfit(x, y, 1)

    # predictions
    y_pred = a * x + b

    #  $R^2$ 
    ss_res = np.sum((y - y_pred)**2)
    ss_tot = np.sum((y - np.mean(y))**2)
    r2 = 1 - ss_res / ss_tot

    return a, b, r2
```

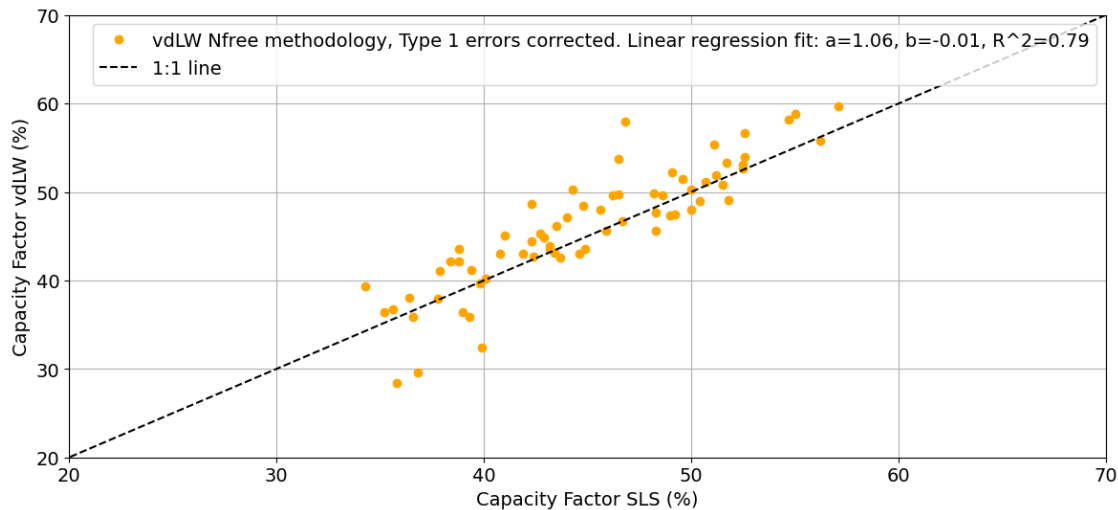
Now, we plot the figure

[118]: *# determine terms of linear regression between the capacity factor of vdLW with*
↳ Type 1 error corrected Nfree and the SLS_2026_CF,
to see how well they correlate, and to see if there is a bias in the results
a_reg, b_reg, r2_reg = linear_fit(SLS_2026_CF, CF_vdLW_Type1_corrected_NfreevdLW)
print("Linear fit: a =", a_reg, "b =", b_reg, "R² =", r2_reg)

we will now plot in the comparison between model and real CF
fig, ax = plt.subplots(figsize=(14, 6))
ax.plot(SLS_2026_CF*100, CF_vdLW_Type1_corrected_NfreevdLW*100,
↳ 'o', color='orange',
label=f'vdLW Nfree methodology, Type 1 errors corrected. Linear
↳ regression fit: a={a_reg:.2f}, b={b_reg:.2f}, R²={r2_reg:.2f}')
ax.plot([0., 100.], [0., 100.], "k--", label="1:1 line")
ax.set_xlim(20, 70)
ax.set_ylim(20, 70)
ax.set_xlabel('Capacity Factor SLS (%)')
ax.set_ylabel('Capacity Factor vdLW (%)')
ax.legend()
ax.set_title('Comparison of modelled and 2025 and Paul')

```
ax.grid()
plt.show()
```

Linear fit: a = 1.0566110500705006 b = -0.01449960900257677 R² = 0.7912085099931245



3.12 Export calculated data

We will create a dataframe with correct capacity factors to use in the next analysis, and save it

```
[119]: # we will create a dataframe, with the name of each case, and then with the
        ↪ corrected capacity factors Cf_Free_vdLW_correct_all,
        # Cf_Inf_vdLW_correct_all, Cf_Model_vdLW_correct_all, and also the corrected
        ↪ values for the limit case Uin=0 and Uout=infinity
df_corrected_vdLW = pd.DataFrame({
    "Name": db_first_eval["Name"].values,
    "Cf_Free_vdLW_correct_all": Cf_Free_vdLW_correct_all,
    "Cf_Inf_vdLW_correct_all": Cf_Inf_vdLW_correct_all,
    "Cf_Model_vdLW_correct_all": Cf_Model_vdLW_correct_all,
    "Cf_Model_vdLW_correct_Uin0_Uoutinf": Cf_Model_vdLW_correct_Uin0_Uoutinf,
    "Cf_Free_vdLW_correct_Uin0_Uoutinf": Cf_Free_vdLW_correct_Uin0_Uoutinf,
    "Cf_Inf_vdLW_correct_Uin0_Uoutinf": Cf_Inf_vdLW_correct_Uin0_Uoutinf
})

# save to an excel file
df_corrected_vdLW.to_excel("vdLW_corrected_capacity_factors.xlsx", index=False)
```

4 Generation of Figures 5 and 6: correction of some Type 2 errors for some of the cases the work of vdLW

This script creates Figures 5 and 6 of the document entitled

Rebuttal to “Comment on ‘A theoretical upper limit for offshore wind energy extraction’ by Simão Ferreira et al. (2026)”: reproducibility analysis, audit of the vdLW implementation, and clarification of finite wind-farm correction methodologies

by

Jens Nørkær Sørensen, Gunner Chr. Larsen, and Carlos Simão Ferreira

first released on 2026 May 16th

at <https://wes.copernicus.org/preprints/wes-2026-59/> CC1: ‘Comment on wes-2026-59’, Jens Nørkær Sørensen, 16 May 2026

4.1 Import of necessary libraries

```
[120]: # first, we will import the necessary libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import xarray as xr
from scipy.interpolate import interp1d
```

4.2 Import of the code of vdLW as a library

We import the code of vdLW as a library to use the function.

However, there will be three functions by vdLW that we will copy into this notebook and modify:

- The wind resource function, which prepares the input of the wind rose at the location of the wind farm. This is for two reasons: 1) vdLW use the direction rose, while the correct formulation is to use the wind power rose; and b) increase the directional integration resolution, which is not only more accurate, it also helps mitigate some of the geometry errors in the vdLW code. The impact of these changes is very small for most cases, but its important for the case of complex clusters in regions of a single predominant wind direction.
- The function to calculate the number of Nedge turbines with neighbors and wind direction. The modification is only for data management, as the code by vdLW as many hardwired elements, and the modification allows us to calculate only the cases we are interested, instead of having to calculate all the cases. This helps to keep this demonstration notebook cleaner. The change does not change the output of the original version not the method by vdLW.
- The function to get coordinates from the inputfile, as it has an unnecessary variable hardwired in the code

```
[121]: # here we import the code by vdLW in the run.py they published in zenodo, so we
      ↪ can use the function we need
import run as vdLWcode;
```



```
# the figure below is generated by the initiation of the vdLW code, it is not
↪relevant
```

4.3 Import wind farm and wind turbine data from database file published by vdLW

Here we upload the wind farm and wind turbine database used by vdLW in their code. As vdLW, we eliminate the same turbines vdLW eliminated.

The “data” folder is directly extracted from the Zenodo repository of the code by vdLW without modifications.

```
[122]: # name of the input file, which is the csv file that contains the turbine data.
↪We will read this file using pandas and store it in a dataframe called
↪wtdata.
infile = 'data\\TurbineDatabase\\20251218_eww_opendatabase.csv'
# infile = 'data/TurbineDatabase/20251218_eww_opendb_carlos_corrected.xlsx'
wtdata = pd.read_csv(infile)

# duplicate the action by vdLW to remove these turbines
# Remove two turbines from Robin Rigg which have been decommissioned
# https://www.sciencedirect.com/science/article/pii/S0267726123000489
# 5026                54.7715        -3.7006
# 5028                54.7692        -3.7093
wtdata = wtdata.drop([5026, 5028])
```

4.4 Import wind farm case list and data

We will use the same file that vdLW used, the only thing needed are the coordinated of the wind farm

```
[123]: infile = 'data\\2026-02-11_input_vars_send_by_CarlosSimaoFerreira.csv'
inputdata = pd.read_csv(infile)
```

4.5 Import GWA wind resource data

This data was previously imported by vdLW and stored in the “data” folder. The code below are the function by vdLW to read the GWA data they stored. This is the procedure in their original code.

Below is the modified function, extracted from the code by vdLW. The changes are explained, and the original code is shown and commented

```
[124]: # Original code by vdLW, to read the coordinates
# def get_latlon_decimal(coords):
#     nWF = len(coords)
#     latlon = np.zeros((nWF, 2))
#     for i in range(nWF):
```

```

#         latlon[i, 0] , latlon[i, 1]  =
↪dms_string_to_decimal(inputdata['Coordinates'][i])
#         return latlon

# issue: hardwired use of inputdata['Coordinates'] whcih are passed in as coords

def get_latlon_decimal(coords):
    nWF = len(coords)
    latlon = np.zeros((nWF, 2))
    for i in range(nWF):
        latlon[i, 0] , latlon[i, 1]  = vdLWcode.dms_string_to_decimal(coords[i])
    return latlon

```

```

[125]: # do not rerun the gwa wind resource, use data stored by vdLW
rerun_gwa = False

# Get list of lat lon in decimal format
latlon = get_latlon_decimal(inputdata['Coordinates'])
# Get Global Wind Atlas wind resource for each wind farm
# get_gwa_windresource(latlon, inputdata['Name'], rerun=True)
ds = vdLWcode.get_gwa_windresource(latlon, inputdata['Name'], rerun=rerun_gwa)

```

4.6 Definition of the cases of wind farms for which we want to correct some Type 2 errors via preprocessing and inputs

There are 11 wind farms for which we can partially mitigate the Type 2 errors in the method and code by vdLW, without changing the code in the core vdLW method part. These are:

1. Gemini - Error: the wind farm is composed of two disjointed sub-wind farms. The method of vdLW creates an artificial wind farm space in the are inbetween, creating artificial. Solution: we will eliminate the administrative-induced error by labelling the subfarms as 1 and 2.
2. Thortonbank - Same as Gemini
3. Borssele I-II - Error: missign wind farm Borssele III-V, which also affect the other farms in the Dutch-Belgium cluster. Solution: we will add the Borssele III-V as neighbours and target wind farms
4. Norther - Same as above
5. Northwester 2 - Error: Neighbor geometries incorrect and wind power rose not applied. Solution: correct neighbor geometry and wind power rose applied.
6. Belwind Phase 1 - Error: the method of vdLW takes the disjointed geoemtry of Nobelwind adn connects the two parts, creating an artificial wind farm area. This artifical area overlaps Belwind, creating an artificial infinite win-farm effect.
7. Nobelwind - Error: the issues above, but also an overestimation of the number of total free turbines in the West region of Nobelwind, by the use of 2.5*Nedge turbines in a single-row sub-windfarm.
8. Northwind - Error: fails to identify the edge turbines due to the concave/convex limitations of the vdLW approach. Solution: use the solution already implemented by vdLW of adding a larger concativity allowance factor, which vdLW applied to other farms but not these ones (or used a low value).

9. Rentel - same as above

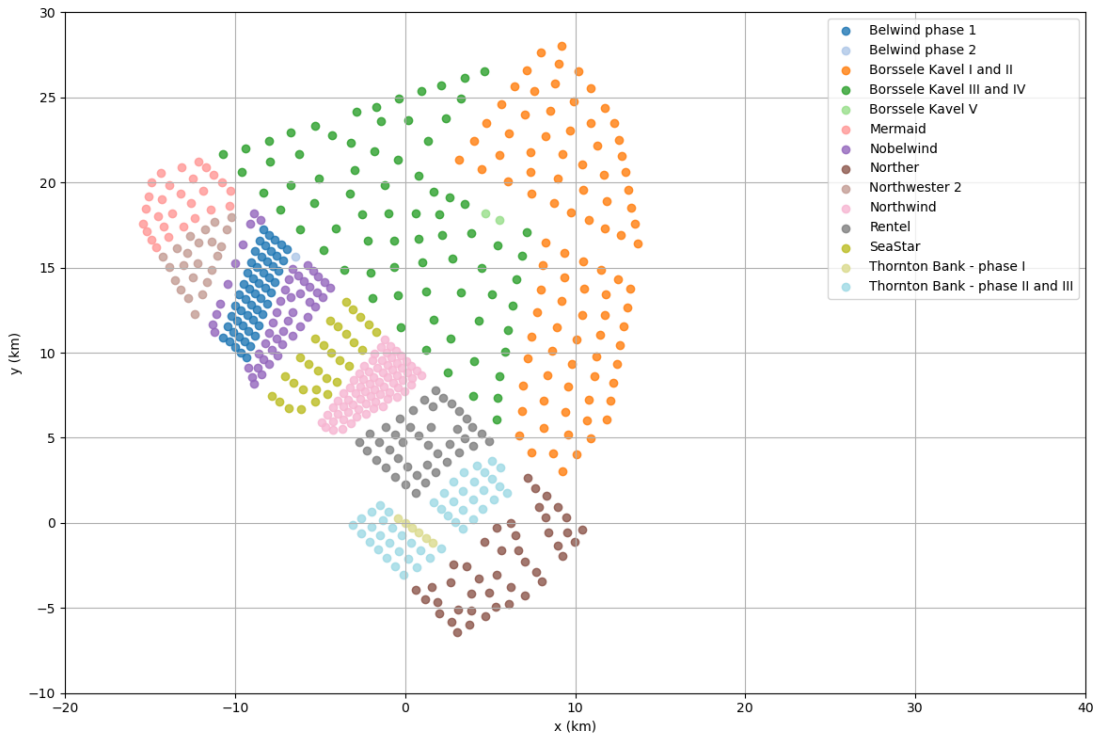
10. Rampion - same as above

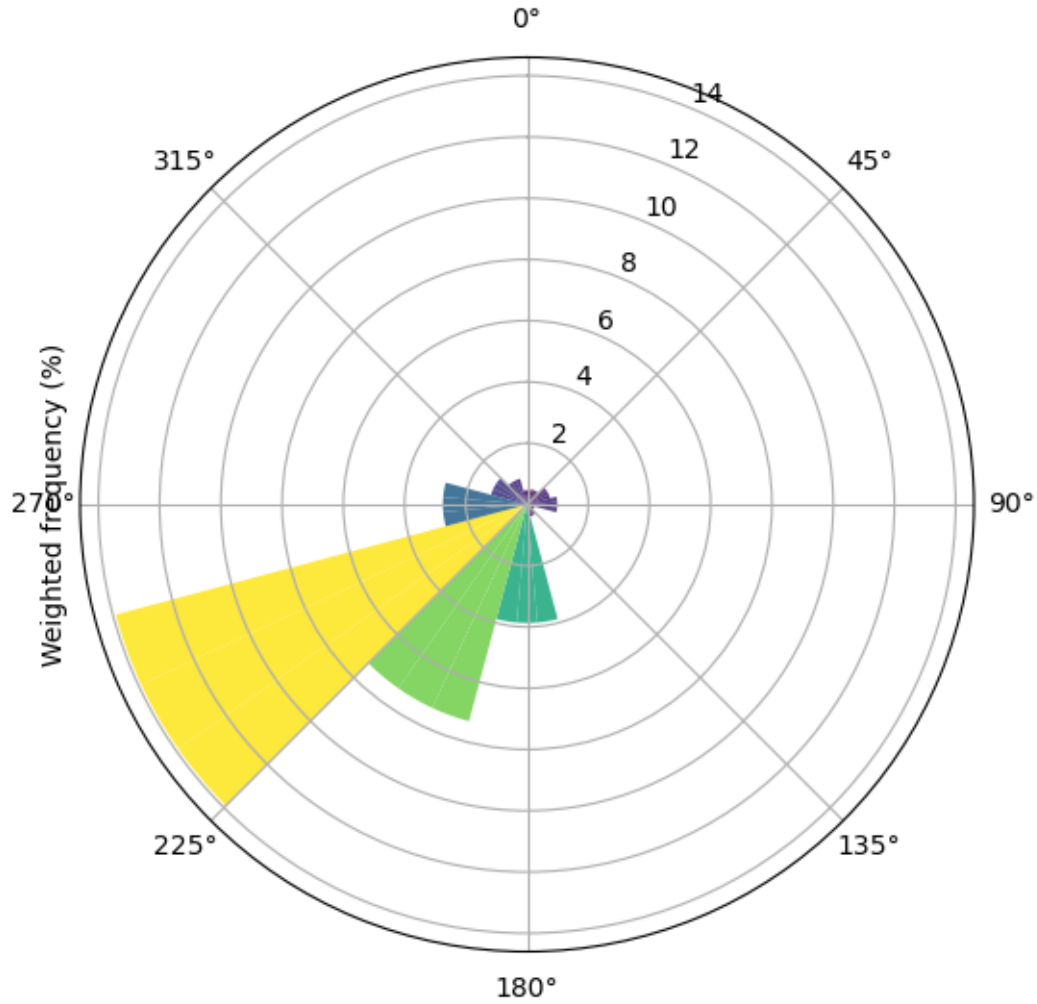
11. Robin Rigg - same as above

8 out of 11 of these cases are in the Dutch-Belgium cluster.

4.7 Map of the Belgium-Dutch cluster

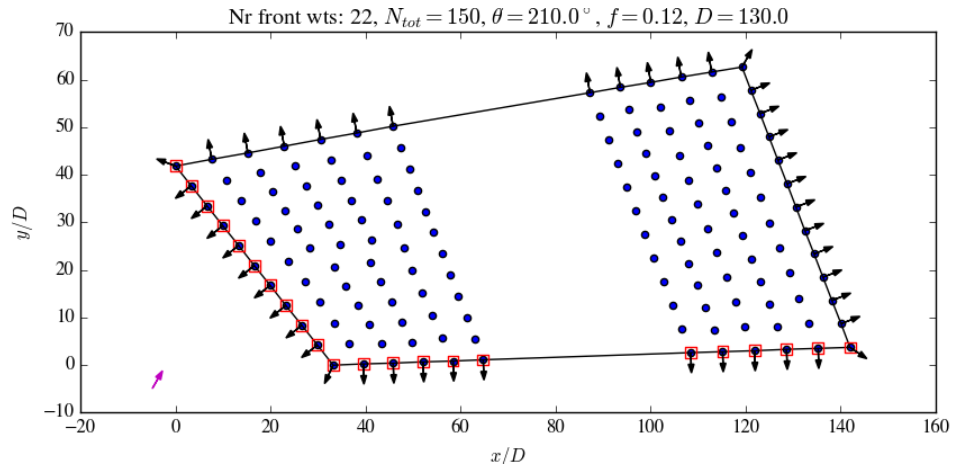
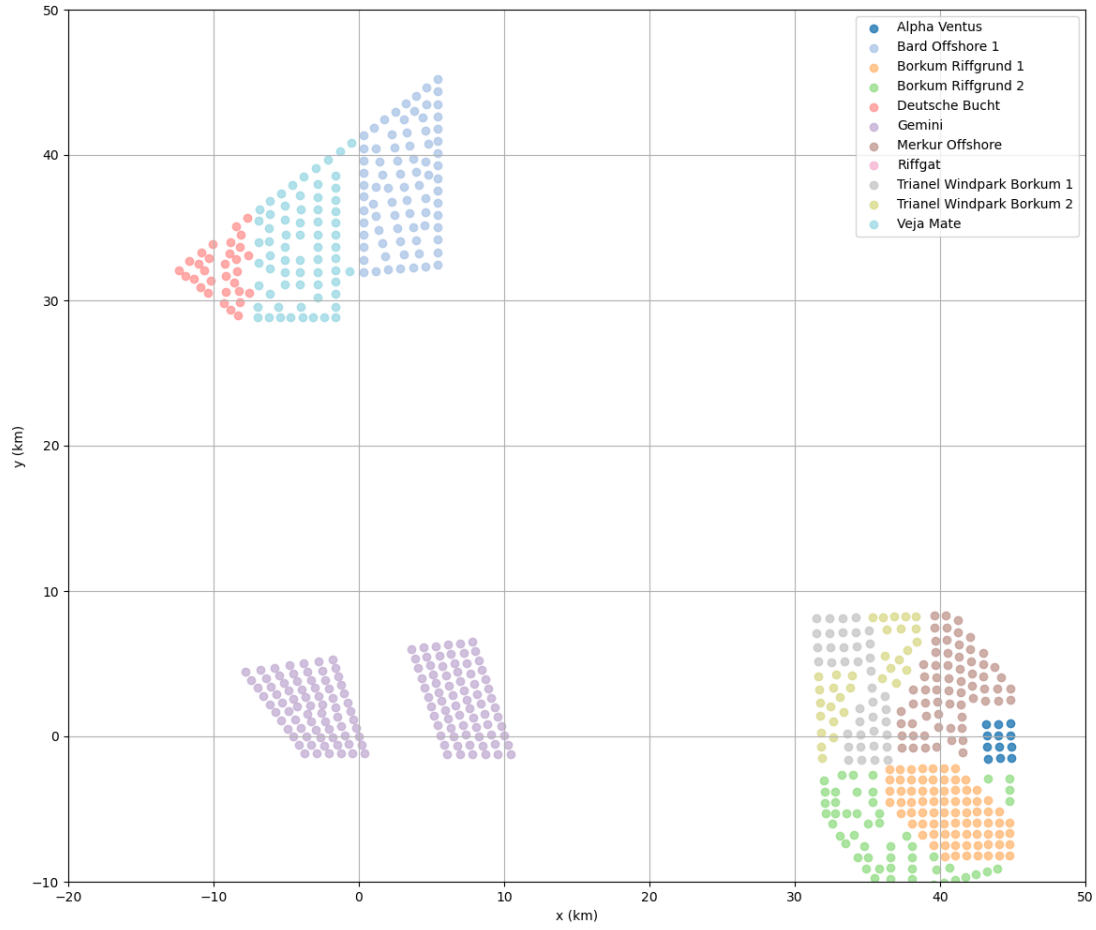
As 8 out of 11 of these cases are in the Dutch-Belgium cluster, it might help the reader to understand the complexity of the geometry, and why the code by vdLW will have so much difficulties dealing with the geometries. We leave the representation below for support. Also, a representation of the local wind power rose.





4.8 Representation of Gemini

One of the main issues in the vdLW code concerns the treatment of wind farms composed of non-contiguous sub-farms. The figure below shows a representation of the Gemini wind farm (and neighbors). The following figure was generated and published by vdLW in Zenodo, and shows how the creation of the artificial wind farm space and edge leads to underestimation of the number of wind turbines experiencing clean inflow (pink arrow represent wind direction).



4.8.1 Definition of alphas for the concave/convex edge geometry process by vdLW

We copy the original inputs defined by vdLW and add/modify for a few wind farms where the failure to identify the edge is highly relevant

```

[126]: ## Original code inputs by vdLW, we made into a function, so we can call it
        ↪with different alphas and different administrative boundaries

        ## Original code by vdLW, to define the alphas for the concave hull method
        # # A Concave hull method is used to determine the farm edge, which is not
        ↪unique and depends on alpha.
        # # A larger alpha means more concave. A too large alpha can result in
        ↪strange results.
        # alphas = [0.0001] * len(inputdata['Nt'])
        # alphas[inputdata.loc[inputdata["Name"] == 'Amrumbank West', :].index[0]]
        ↪= 0.0005
        # alphas[inputdata.loc[inputdata["Name"] == 'Kriegers Flak', :].index[0]] =
        ↪0.00025
        # alphas[inputdata.loc[inputdata["Name"] == 'Beatrice extension', :].
        ↪index[0]] = 0.0005
        # alphas[inputdata.loc[inputdata["Name"] == 'Belwind Phase 1', :].index[0]]
        ↪= 0.0005
        # alphas[inputdata.loc[inputdata["Name"] == 'Borssele I-II', :].index[0]] =
        ↪0.0005
        # alphas[inputdata.loc[inputdata["Name"] == 'DanTysk', :].index[0]] = 0.
        ↪0005
        # alphas[inputdata.loc[inputdata["Name"] == 'Dudgeon', :].index[0]] = 0.0005
        # alphas[inputdata.loc[inputdata["Name"] == 'Gallopier', :].index[0]] = 0.
        ↪00015
        # alphas[inputdata.loc[inputdata["Name"] == 'Gode 1 and 2', :].index[0]] =
        ↪0.0005
        # alphas[inputdata.loc[inputdata["Name"] == 'Greater Gabbard', :].index[0]]
        ↪= 0.00025
        # alphas[inputdata.loc[inputdata["Name"] == 'Gunfleet Sand', :].index[0]] =
        ↪0.0005
        # alphas[inputdata.loc[inputdata["Name"] == 'Gwynt y Môr', :].index[0]] = 0.
        ↪001
        # alphas[inputdata.loc[inputdata["Name"] == 'Hornsea 1', :].index[0]] = 0.
        ↪00025
        # alphas[inputdata.loc[inputdata["Name"] == 'Hornsea 2', :].index[0]] = 0.
        ↪0002
        # alphas[inputdata.loc[inputdata["Name"] == 'Horns Rev 3', :].index[0]] = 0.
        ↪0005
        # alphas[inputdata.loc[inputdata["Name"] == 'Kaskasi', :].index[0]] = 0.0005
        # alphas[inputdata.loc[inputdata["Name"] == 'Lillgrund', :].index[0]] = 0.
        ↪0005
        # alphas[inputdata.loc[inputdata["Name"] == 'Lincs', :].index[0]] = 0.001
        # alphas[inputdata.loc[inputdata["Name"] == 'Meerwind Sud/Ost', :].
        ↪index[0]] = 0.0005
        # alphas[inputdata.loc[inputdata["Name"] == 'Merkur', :].index[0]] = 0.0005

```

```

# alphas[inputdata.loc[inputdata["Name"] == 'Moray East', :].index[0]] = 0.
↪0005
# alphas[inputdata.loc[inputdata["Name"] == 'Nobelwind', :].index[0]] = 0.
↪0005
# alphas[inputdata.loc[inputdata["Name"] == 'Nordsee One', :].index[0]] = 0.
↪0005
# alphas[inputdata.loc[inputdata["Name"] == 'Norther', :].index[0]] = 0.0005
# alphas[inputdata.loc[inputdata["Name"] == 'Northwester 2', :].index[0]] = 0.0005
↪0.0005
# alphas[inputdata.loc[inputdata["Name"] == 'Princess Amalia', :].index[0]] = 0.001
↪0.001
# alphas[inputdata.loc[inputdata["Name"] == 'Race Bank', :].index[0]] = 0.
↪001
# alphas[inputdata.loc[inputdata["Name"] == 'Rampion', :].index[0]] = 0.
↪00025
# alphas[inputdata.loc[inputdata["Name"] == 'Veja Mate', :].index[0]] = 0.
↪00025
# alphas[inputdata.loc[inputdata["Name"] == 'Walney 2', :].index[0]] = 0.
↪00025
# alphas[inputdata.loc[inputdata["Name"] == 'Walney Extension', :].index[0]] = 0.00025

## Original code inputs by vdLW, we made into a function, so we can call it
↪with different alphas and different administrative boundaries

def define_alphas(inputdata):

    # A Concave hull method is used to determine the farm edge, which is not
    ↪unique and depends on alpha.
    # A larger alpha means more concave. A too large alpha can result in
    ↪strange results.

    alphas = [0.0001] * len(inputdata['Nt'])

    def set_alpha_if_exists(name, value):
        idx = inputdata.loc[inputdata["Name"] == name].index
        if len(idx) > 0:
            alphas[idx[0]] = value

    set_alpha_if_exists('Amrumbank West', 0.0005)
    set_alpha_if_exists('Kriegers Flak', 0.00025)
    set_alpha_if_exists('Beatrice extension', 0.0005)
    set_alpha_if_exists('Belwind Phase 1', 0.0005)
    set_alpha_if_exists('Borsssele I-II', 0.0005)

```

```

set_alpha_if_exists('DanTysk', 0.0005)
set_alpha_if_exists('Dudgeon', 0.0005)
set_alpha_if_exists('Gallopier', 0.00015)
set_alpha_if_exists('Gode 1 and 2', 0.0005)
set_alpha_if_exists('Greater Gabbard', 0.00025)
set_alpha_if_exists('Gunfleet Sand', 0.0005)
set_alpha_if_exists('Gwynt y Môr', 0.001)
set_alpha_if_exists('Hornsea 1', 0.00025)
set_alpha_if_exists('Hornsea 2', 0.0002)
set_alpha_if_exists('Horns Rev 3', 0.0005)
set_alpha_if_exists('Kaskasi', 0.0005)
set_alpha_if_exists('Lillgrund', 0.0005)
set_alpha_if_exists('Lincs', 0.001)
set_alpha_if_exists('Meerwind Sud/Ost', 0.0005)
set_alpha_if_exists('Merkur', 0.0005)
set_alpha_if_exists('Moray East', 0.0005)
set_alpha_if_exists('Nobelwind', 0.0005)
set_alpha_if_exists('Nordsee One', 0.0005)
set_alpha_if_exists('Norther', 0.0005)
set_alpha_if_exists('Northwester 2', 0.0005)
set_alpha_if_exists('Princess Amalia', 0.001)
set_alpha_if_exists('Race Bank', 0.001)
set_alpha_if_exists('Rampion', 0.00025)
set_alpha_if_exists('Veja Mate', 0.00025)
set_alpha_if_exists('Walney 2', 0.00025)
set_alpha_if_exists('Walney Extension', 0.00025)

##### modification and addition in this code by SLS
set_alpha_if_exists('Northwind', 0.0025)
set_alpha_if_exists('Rampion', 0.00125)
set_alpha_if_exists('Rentel', 0.001)
set_alpha_if_exists('Robin Rigg', 0.003)

return alphas

```

5 Atmospheric Boundary Layer inputs defined by vdLW

Here we define the roughness length of the ABL and the height at which the wind rose is sampled. As defined by vdLW

```

[127]: # Main input parameters
z0 = 10 ** (-4) # Roughness length [m]
zRef = 100.0 # Reference height at which GWA data is taken [m]

```


5.1 Wind Power Rose function

We modify the input processing function by vdLW, which was originally using the win direction frequency for the directional integration. Instead, it should be the Wind Power Rose weighted frequency, as we are dealing with the integration of power fractions.

```
[128]: # modification of the function for assessing the wind power rose frequency
        ↪distribution
# we first show the original function by vdLW, and then we show the corrected
        ↪function by SLS.
# Once again, this change only impacts specific locations, where the power
        ↪distribtuion as very dominant directions

# Original code by vdLW
# def calc_windresource(ds, z0, zRef, zH):
#     nWF = len(ds.wfname)
#     ARef = np.zeros((nWF))
#     AH = np.zeros((nWF))
#     kRef = np.zeros((nWF))
#     windrose = np.zeros((nWF, 12))
#     for i in range(nWF):
#         # Calculate wind rose frequency weighted A and k
#         ds_sub = ds.sel(wf=i, height=zRef).interp(roughness=0.0)
#         ds_sub = ds_sub.rename({'frequency': 'wdfreq'})
#         ds_sub = ds_sub.assign_coords(west_east=ds_sub['lon'],
#                                     south_north=ds_sub['lat'],
#                                     sector=wk.create_sector_coords(12))
#         ds_sub = wk.spatial._point._from_scalar(ds_sub)
#         A_combined, k_combined = wk.weibull_combined(ds_sub)
#         ARef[i] = A_combined
#         kRef[i] = k_combined
#         windrose[i, :] = ds.sel(wf=i, height=zRef).interp(roughness=0.
        ↪0)['frequency'] / ds.sel(wf=i, height=zRef).interp(roughness=0.
        ↪0)['frequency'].sum()
#     # Log interpolate A at hub height
#     AH = ARef * np.log(zH / z0) / np.log(zRef / z0)
#     return ARef, kRef, AH, windrose

# modified code by SLS
from scipy.special import gamma
def calc_windresource(ds, zRef):
    nWF = len(ds.wfname)
    Mfactor_increase=3 # increase the number of sectors by a factor of
        ↪Mfactor_increase to increase the detail of the wind rose
    windrose = np.zeros((nWF, 12*Mfactor_increase)) # 12 directions, increased
        ↪by factor of Mfactor_increase to increase the detail of the wind rose
```

```

for i in range(nWF):
    ds_sub = ds.sel(wf=i, height=zRef).interp(roughness=0.0)
    ds_sub = ds_sub.rename({'frequency': 'wdfreq'})
    ds_sub = ds_sub.assign_coords(west_east=ds_sub['lon'],
                                south_north=ds_sub['lat'],
                                sector=vdLWcode.wk.
↪create_sector_coords(12))
    ds_sub = vdLWcode.wk.spatial._point._from_scalar(ds_sub)

    # Wind Power Rose
    frequencydist = ds.sel(wf=i, height=zRef).interp(roughness=0.
↪0)['frequency']
    lambdadist = ds.sel(wf=i, height=zRef).interp(roughness=0.0)['A']
    kdist = ds.sel(wf=i, height=zRef).interp(roughness=0.0)['k']
    Pdist = lambdadist**3 * gamma(1 + 3 / kdist) * frequencydist
    # increase 12 sectors to 36 sectors
    Pdist36 = np.repeat(Pdist.values / Mfactor_increase, Mfactor_increase)

    # normalize
    Pdist36 = Pdist36 / np.sum(Pdist36)
    windrose[i, :] = Pdist36

return windrose

```

5.2 Define function to calculate the layout parameters

This is the function that calculates the layout parameters as defined by vdLW. The code by vdLW as some hardwired elements that decrease its flexibility. Here we copy paste the function and modify it so it can calculate a 1 to N wind farm case (the original function always calculates all wind farm cases) and checks the input data to define the for loops. Also, we removed all the plot function to keep the code short. The functionality and calculations remain the same. We do not copy the original code here, as it is long. The function still has the same name as the one in run.py, and the reader can check the original function in run.py.

```

[129]: # trimmed function form teh original in run.py

def calculate_layout_metrics(Target_case, wt_xs, wt_ys, alphas, inputdata,
↪wf_neighbors, wf_neighbors_flag, ps, windrose):
    print(' List of wind farms we will evaluate')
    for i in Target_case:
        print(' - %s : %s' % (i, inputdata['Name'][i]))

    nWF = len(inputdata['Name'])

    Areas = []
    Nwt_rows = []

```

```

Nwt_row_wf_neighbors = []
inlet_all = []
boundary_points_all = []
normals_mean_all = []
hulls = []
wf_norms = []
# Carlos modified
Nwinddirections = len(windrose[0, :])
print('Number of wind directions in wind rose', Nwinddirections)
wd_windrose = np.linspace(0, 360, Nwinddirections, endpoint=False)
# np.array([0, 30, 60, 90, 120, 150, 180, 210, 240, 270, 300, 330])

print('Calculating layout metrics for %s wind farms' % nWF)

for i in range(nWF):
    # print('Calculating layout metrics for wind farm %s' % i)
    ↪inputdata['Name'][i])
    wt_x = np.asarray(wt_xs[i])
    wt_y = np.asarray(wt_ys[i])

    wf_norm = [wt_x.min(), wt_y.min()]
    wf_norms.append(wf_norm)
    wt_x = wt_x - wf_norm[0]
    wt_y = wt_y - wf_norm[1]

    Dref = inputdata['D'][i]

    # Wind farm area and edge turbines
    area, boundary_points, hull = vdLWcode.calculate_wf_area_and_edge(wt_x, ↪
    ↪wt_y, ConcaveHull_alpha=alphas[i], plot=False, outfile='wflayout_polygon.
    ↪pdf')
    Areas.append(area)

    # inlet edge turbine and turbine normals for plotting
    inlet_wd = []
    for l in range(Nwinddirections):
        inlet, boundary_points, normals_mean = vdLWcode.
    ↪calculate_front_row_wts(boundary_points, wd_windrose[l], hull)
        inlet_wd.append(inlet)
        inlet_all.append(inlet_wd)
        boundary_points_all.append(boundary_points)
        normals_mean_all.append(normals_mean)
        hulls.append(hull)

for i in Target_case:
    print("Calculating wind farm %s" % inputdata['Name'][i])

```

```

        # Remove inlet turbines due to neighboring wind farms with a length
        ↪scale x times the turbine spacing
        # print('Removing inlet turbines due to neighboring wind farms for wind
        ↪farm %s' % inputdata['Name'][i])
        S = np.sqrt(inputdata['A'][i] * 1e6) / (inputdata['D'][i] * (np.
        ↪sqrt(inputdata['Nt'][i])-1))
        L = 10 * S * inputdata['D'][i]
        # print('L', L)
        wf_shadows = []
        inlet_with_wf_neighborss = []
        for l in range(Nwinddirections):
            inlet_with_wf_neighbors, wf_shadow, wf_neighbor_coords = vdLWcode.
            ↪remove_inlet_turbines_wf_neighbors(i, inputdata['Name'][i], inlet_all[i][1],
            ↪boundary_points_all[i], hulls, inputdata, wd_windrose[l], wf_neighbors[i],
            ↪wf_neighbors_flag[i], ps, wf_norms, L)
            wf_shadows.append(wf_shadow)
            inlet_with_wf_neighborss.append(inlet_with_wf_neighbors)

        Nwt_row = 0
        for l in range(Nwinddirections):
            Nwt_row = Nwt_row + len(boundary_points_all[i][:,
            ↪0][inlet_all[i][1]]) * windrose[i, l]
        Nwt_rows.append(Nwt_row)

        Nwt_row_wf_neighbor = 0
        for l in range(Nwinddirections):
            Nwt_row_wf_neighbor = Nwt_row_wf_neighbor +
            ↪len(boundary_points_all[i][:, 0][inlet_with_wf_neighborss[l]]) * windrose[i,
            ↪1]
        Nwt_row_wf_neighbors.append(Nwt_row_wf_neighbor)

    return Areas, Nwt_rows, Nwt_row_wf_neighbors

```

5.3 Define function to overcome the wind farm labelling limitation of the code by vdLW

Most of the errors of Type 2 are associated to trating the labelling of wind turbines (belonging to a wind farm or another) as a an aerodynamic constraint, and the process where the code of vdLW forces all turbines belonging to a wind farm to be inside a single polygon.

These limitation can be overcome by some administrative reassigning of labels, splitting wind farms in subwindfarms, adding missing wind farms, etc. The function below does this for the specific calculation cases. It is specific to each wind farm, but as there are few to which we are applying it, and taking into account the structure of vdLW code, it is the simplest way to do it.

```
[130]: # these function are needed to add or subtract cases from the wind farm case_
        ↪list, and reassigning wind turbines to subfarms or another farm names

def duplicate_wind_farm_row(inputdata, ds, row_idx, wf_dim="wf", suffix="_2"):
    """
    Duplicate one wind farm row in both:
    - pandas DataFrame inputdata
    - xarray Dataset/DataArray ds along dimension wf_dim

    The duplicated row is inserted directly below the original.
    """

    # -----
    # 1. Duplicate pandas row
    # -----
    df = inputdata.copy().reset_index(drop=True)

    row_copy = df.iloc[[row_idx]].copy()

    # optional: change name of copied row if Name exists
    if "Name" in row_copy.columns:
        # print("Original Name:", row_copy["Name"].values[0])
        row_copy["Name"] = row_copy["Name"].astype(str) + suffix
        # print("New Name:", row_copy["Name"].values[0])

    df_new = pd.concat(
        [
            df.iloc[:row_idx + 1],
            row_copy,
            df.iloc[row_idx + 1:],
        ],
        ignore_index=True,
    )

    # -----
    # 2. Duplicate xarray row
    # -----
    ds_copy = ds.copy()

    one_wf = ds_copy.isel({wf_dim: [row_idx]}).copy()

    ds_new = xr.concat(
        [
            ds_copy.isel({wf_dim: slice(0, row_idx + 1)}),
            one_wf,
            ds_copy.isel({wf_dim: slice(row_idx + 1, None)}),
        ],

```

```

        dim=wf_dim,
    )

    # reset wf coordinate to 0, 1, 2, ...
    ds_new = ds_new.assign_coords({wf_dim: range(df_new.shape[0])})

    return df_new, ds_new

def remove_wind_farm_row(inputdata, ds, row_idx, wf_dim="wf"):
    """
    Remove one wind farm row from both:
    - pandas DataFrame inputdata
    - xarray Dataset/DataArray ds along dimension wf_dim

    The row is removed based on row_idx.
    """

    # -----
    # 1. Remove pandas row
    # -----
    df = inputdata.copy().reset_index(drop=True)

    if row_idx < 0 or row_idx >= len(df):
        raise IndexError(f"row_idx {row_idx} is out of range for inputdata")

    removed_name = df.loc[row_idx, "Name"] if "Name" in df.columns else None

    df_new = df.drop(index=row_idx).reset_index(drop=True)

    # -----
    # 2. Remove xarray row
    # -----
    if row_idx < 0 or row_idx >= ds.sizes[wf_dim]:
        raise IndexError(f"row_idx {row_idx} is out of range for ds dimension_
↪{wf_dim}")

    ds_new = xr.concat(
        [
            ds.isel({wf_dim: slice(0, row_idx)}),
            ds.isel({wf_dim: slice(row_idx + 1, None)}),
        ],
        dim=wf_dim,
    )

    # reset wf coordinate to 0, 1, 2, ...

```

```

ds_new = ds_new.assign_coords({wf_dim: np.arange(len(df_new))})

return df_new, ds_new, removed_name

def merge_split_farms(farm_names, *arrays, suffix="_2"):
    """
    Merge farms like 'Gemini' and 'Gemini_2'.
    The values of Gemini_2 are added to Gemini, and Gemini_2 is removed.

    farm_names: list/array of farm names
    *arrays: any number of result arrays with same length as farm_names
    """

    farm_names = list(farm_names)
    arrays = [list(a) for a in arrays]

    keep = [True] * len(farm_names)

    for i, name in enumerate(farm_names):
        if name.endswith(suffix):
            base_name = name[:-len(suffix)]

            if base_name in farm_names:
                j = farm_names.index(base_name)

                for arr in arrays:
                    arr[j] = arr[j] + arr[i]

                keep[i] = False

    new_farm_names = [name for name, k in zip(farm_names, keep) if k]
    new_arrays = [
        [value for value, k in zip(arr, keep) if k]
        for arr in arrays
    ]

    return new_farm_names, *new_arrays

# function that rearranges the input data to calculate different administrative
↳ boundaries, and minimize Type 2 errors in the work of vdLW

def reassign_administrative_boundaries(inputdata, ds, wtdata,
                                       add_Borssele_III_V=True,
↳ split_Gemini=True, split_Thorntonbank=True):
    # here we will reassign the administrative boundaries of the wind farms, to
↳ minimize Type 2 errors in the work of vdLW

```

```

# this is a manual process, and we will do it for each wind farm in the
↳ input data

# first, the cases we will always do

# First, we will add the missing Borssele III-V, and we will call it as
↳ Borssele I-II_otherfarms, and we will assign the turbines of Borssele III-V
↳ to this new farm,
# the strange name is to keep using the same duplicate_wind_farm_row
↳ function
if add_Borssele_III_V:
    # locate the row index of Borssele I-II in the input data, to add
↳ Borssele III-V just after
    borssele_row_idx = inputdata.index[inputdata['Name'] == 'Borssele
↳ I-II'].tolist()[0]

    # add a new row to the list of cases
    inputdata, ds = duplicate_wind_farm_row(inputdata, ds, borssele_row_idx,
↳ wf_dim="wf", suffix="_otherfarms")

    # identify the turbines of Borssele III-V in the wtdata, and reassign
↳ them to the new farm name "Borssele I-II_otherfarms"
    borssele_wtdata_idx = wtdata.index[(wtdata['wind_farm'] == 'Borssele
↳ Kavel III and IV') | (wtdata['wind_farm'] == 'Borssele Kavel V')].tolist()
    wtdata.loc[borssele_wtdata_idx, 'wind_farm'] = 'Borssele
↳ I-II_otherfarms'

# now, we will split Gemini into two wind farms, to avoid the error of fake
↳ wind farm areas
if split_Gemini:
    gemini_row_idx = inputdata.index[inputdata['Name'] == 'Gemini'].
↳ tolist()[0]
    inputdata, ds = duplicate_wind_farm_row(inputdata, ds, gemini_row_idx,
↳ wf_dim="wf", suffix="_2")

    # now, we will find in wtdata the rows of Gemini and longitude above 5.
↳ 95, and will change the name to Gemini_2, to match the duplicated row in
↳ inputdata
    # we want to change wtdata, so maybe with indices, we can find the rows
↳ of Gemini and longitude above 5.95, and change the name to Gemini_2
    gemini_wtdata_idx = wtdata.index[(wtdata['wind_farm'] == 'Gemini') &
↳ (wtdata['longitude'] > 5.95)].tolist()
    wtdata.loc[gemini_wtdata_idx, 'wind_farm'] = 'Gemini_2'

```



```

    if split_Thorntonbank:
        # first, we will find in wtdata all the rows that have wind_farm that
        ↪ contains "Thornton Bank" and change it to Thorntonbank, to match the name in
        ↪ inputdata
        wtdata.loc[wtdata['wind_farm'].str.contains('Thornton Bank',
        ↪ case=False), 'wind_farm'] = 'Thortonbank' # yes, there is a typo in the name
        ↪ of the farm in inputdata, but we will keep it to avoid errors in the code,
        ↪ and we will change the name in wtdata to match it
        thorntonbank_row_idx = inputdata.index[inputdata['Name'] ==
        ↪ 'Thortonbank'].tolist()[0]
        inputdata, ds = duplicate_wind_farm_row(inputdata, ds,
        ↪ thorntonbank_row_idx, wf_dim="wf", suffix="_2")
        thorntonbank_wtdata_idx = wtdata.index[(wtdata['wind_farm'] ==
        ↪ 'Thortonbank') & (wtdata['longitude'] > 2.95) & (wtdata['latitude'] > 51.
        ↪ 54)].tolist()
        wtdata.loc[thorntonbank_wtdata_idx, 'wind_farm'] = 'Thortonbank_2'

    return inputdata, ds, wtdata

```

5.4 Run calculations for the eleven wind farms

```

[131]: # calculate windroses
def calculate_windrose_and_layout_parameters(inputdata, ds, wtdata, zRef):
    windrose = calc_windresource(ds, zRef)

    # calculate layout parameters
    wf_lats, wf_lons, wt_xs, wt_ys, wt_lats, wt_lons, ps = vdLWcode.
    ↪ get_layouts(inputdata, wtdata)

    # calculate neighbour parameters

    wf_neighbors, wf_neighbors_flag = vdLWcode.get_wf_neighbor(wf_lats,
    ↪ wf_lons, inputdata['Name'], lat_dist=0.5, lon_dist=0.5)
    return windrose, wf_lats, wf_lons, wt_xs, wt_ys, wt_lats, wt_lons, ps,
    ↪ wf_neighbors, wf_neighbors_flag

```

5.5 Evaluation of the wind farm cases ofr which we want to have more correct values

```

[132]: # this is the list of names that are contained in the names of the wind farm
        ↪ cases that we will evaluate
        # it is just a bit of a manual process to select the cases we will evaluate
        List_windfarm_names = ['Gemini', 'Thorton', 'Robin Rigg', 'Borssele',
        ↪ 'Northwind', 'Belwind', 'Northwester', 'Rentel', 'Nobelwind',

```

```
'Rampion', 'Norther'] # some are mispelled, but that is
↳not important
```

```
[133]: def identify_target_cases(inputdata, List_windfarm_names, print_cases=True):

    # identify in the list of cases to evaluate, the row index of cases that
    ↳contains in the name the words in the List_windfarm_names, and store the row
    ↳index in a list called Target_case

    Target_case = []
    for i in range(len(inputdata['Name'])):
        for name in List_windfarm_names:
            if name in inputdata['Name'][i]:
                Target_case.append(i)
                break
    # sort the Target_case list
    Target_case = sorted(Target_case)

    # print the indices and names of the cases we will evaluate
    if print_cases:
        print(' List of wind farms we will evaluate')
        for i in Target_case:
            print(' - %s : %s' % (i, inputdata['Name'][i]))

    return Target_case
```

```
[134]: #
inputdata_temp = inputdata.copy()
ds_temp = ds.copy()
wtdata_temp = wtdata.copy()

inputdata_temp, ds_temp, wtdata_temp =
↳reassing_administrative_boundaries(inputdata_temp, ds_temp, wtdata_temp)

alphas_temp = define_alphas(inputdata_temp)

windrose, wf_lats, wf_lons, wt_xs, wt_ys, wt_lats, wt_lons, ps, wf_neighbors,
↳wf_neighbors_flag = calculate_windrose_and_layout_parameters(inputdata_temp,
↳ds_temp, wtdata_temp, zRef)
```

```

# display all rows and columns of the inputdata_temp dataframe, to check the
↳changes we made in the administrative boundaries
pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
# display(inputdata_temp)

Target_case = identify_target_cases(inputdata_temp, List_windfarm_names,
↳print_cases=False)
# we now calculate the metrics
Areas, Nwt_rows, Nwt_row_wf_neighbors = calculate_layout_metrics(Target_case,
↳wt_xs, wt_ys, alphas_temp, inputdata_temp, wf_neighbors, wf_neighbors_flag,
↳ps, windrose)

```

List of wind farms we will evaluate

- 33 : Gemini
- 34 : Gemini_2
- 38 : Borssele I-II
- 39 : Borssele I-II_otherfarms
- 40 : Norther
- 41 : Thortonbank
- 42 : Thortonbank_2
- 43 : Rentel
- 44 : Northwind
- 45 : Nobelwind
- 46 : Belwind Phase 1
- 47 : Northwester 2
- 50 : Rampion
- 74 : Robin Rigg

Number of wind directions in wind rose 36

Calculating layout metrics for 75 wind farms

Calculating wind farm Gemini

Calculating wind farm Gemini_2

Calculating wind farm Borssele I-II

Calculating wind farm Borssele I-II_otherfarms

Calculating wind farm Norther

Calculating wind farm Thortonbank

Calculating wind farm Thortonbank_2

Calculating wind farm Rentel

Calculating wind farm Northwind

Calculating wind farm Nobelwind

Calculating wind farm Belwind Phase 1

Calculating wind farm Northwester 2

Calculating wind farm Rampion

Calculating wind farm Robin Rigg

we need to also evaluate Thorntonbank alone, as a self neighbor This is imposed by the structure of the code by vdLW. We need this value for the first years of operation, which are

accounted in the real capacity factor data

```
[135]: # now, we will run a special case for Thorntonbank,
# to determine the values for the years that there were no other neighboring
↳ farms
inputdata_thorntonbank = inputdata_temp.copy()
# eliminate all rows that do not contain "Thorntonbank" in the name, to keep
↳ only the row of Thorntonbank
inputdata_thorntonbank = inputdata_thorntonbank[inputdata_thorntonbank['Name'].
↳ str.contains('Thorntonbank', case=False)].reset_index(drop=True)
ds_thorntonbank = ds_temp.copy().sel(wf=inputdata_thorntonbank.index)
wtdata_thorntonbank = wtdata_temp.copy()

alphas_thorntonbank = define_alphas(inputdata_thorntonbank)
windrose_thorntonbank, wf_lats_thorntonbank, wf_lons_thorntonbank,
↳ wt_xs_thorntonbank, wt_ys_thorntonbank, wt_lats_thorntonbank,
↳ wt_lons_thorntonbank, ps_thorntonbank, wf_neighbors_thorntonbank,
↳ wf_neighbors_flag_thorntonbank =
↳ calculate_windrose_and_layout_parameters(inputdata_thorntonbank,
↳ ds_thorntonbank, wtdata_thorntonbank, zRef)
Target_case_thorntonbank = identify_target_cases(inputdata_thorntonbank,
↳ List_windfarm_names, print_cases=False)
Areas_thorntonbank, Nwt_rows_thorntonbank, Nwt_row_wf_neighbors_thorntonbank =
↳ calculate_layout_metrics(Target_case_thorntonbank, wt_xs_thorntonbank,
↳ wt_ys_thorntonbank, alphas_thorntonbank, inputdata_thorntonbank,
↳ wf_neighbors_thorntonbank, wf_neighbors_flag_thorntonbank, ps_thorntonbank,
↳ windrose_thorntonbank)
print('Thorntonbank metrics without neighboring farms')
print('Area', Areas_thorntonbank)
print('Nwt_row', Nwt_rows_thorntonbank)
print('Nwt_row_wf_neighbors', Nwt_row_wf_neighbors_thorntonbank)
```

List of wind farms we will evaluate

- 0 : Thorntonbank
- 1 : Thorntonbank_2

Number of wind directions in wind rose 36

Calculating layout metrics for 2 wind farms

Calculating wind farm Thorntonbank

Calculating wind farm Thorntonbank_2

Thorntonbank metrics without neighboring farms

Area [9785002.270990223, 9320096.605955295]

Nwt_row [np.float64(9.086712779773693), np.float64(6.527357409665044)]

Nwt_row_wf_neighbors [np.float64(8.428959513669506),
np.float64(4.472264197522777)]

5.5.1 Remove false Nobelwind overlap over Belwind and correct Nobelwind overestimation

We now correct the effect of the erroneous wind-farm region created by the vdLW code when joining the two separate parts of Nobelwind into a single region.

To isolate and quantify this error, we calculate the performance after administratively allocating the western part of Nobelwind to another wind farm. In this way, we intentionally use the vdLW coding error to our advantage.

```
[136]: # now, we will run the special case of Belwind and Nobelwind, which are
↳neighboring farms,
# but are artificially merging over each other. We will untangle them in the
↳code of vdLW
inputdata_belwind_nobelwind = inputdata_temp.copy()
# keep only the rows that contain "Belwind" or "Nobelwind" in the name, to keep
↳only the rows of Belwind and Nobelwind
inputdata_belwind_nobelwind =
↳inputdata_belwind_nobelwind[inputdata_belwind_nobelwind['Name'].str.
↳contains('Belwind', case=False) | inputdata_belwind_nobelwind['Name'].str.
↳contains('Nobelwind', case=False)].reset_index(drop=True)
ds_belwind_nobelwind = ds_temp.copy().sel(wf=inputdata_belwind_nobelwind.index)
wtdata_belwind_nobelwind = wtdata_temp.copy()

alphas_belwind_nobelwind = define_alphas(inputdata_belwind_nobelwind)

windrose_belwind_nobelwind, wf_lats_belwind_nobelwind,
↳wf_lons_belwind_nobelwind, wt_xs_belwind_nobelwind, wt_ys_belwind_nobelwind,
↳wt_lats_belwind_nobelwind, wt_lons_belwind_nobelwind, ps_belwind_nobelwind,
↳wf_neighbors_belwind_nobelwind, wf_neighbors_flag_belwind_nobelwind =
↳calculate_windrose_and_layout_parameters(inputdata_belwind_nobelwind,
↳ds_belwind_nobelwind, wtdata_belwind_nobelwind, zRef)
Target_case_belwind_nobelwind =
↳identify_target_cases(inputdata_belwind_nobelwind, List_windfarm_names,
↳print_cases=False)
Areas_belwind_nobelwind, Nwt_rows_belwind_nobelwind,
↳Nwt_row_wf_neighbors_belwind_nobelwind =
↳calculate_layout_metrics(Target_case_belwind_nobelwind,
↳wt_xs_belwind_nobelwind, wt_ys_belwind_nobelwind, alphas_belwind_nobelwind,
↳inputdata_belwind_nobelwind, wf_neighbors_belwind_nobelwind,
↳wf_neighbors_flag_belwind_nobelwind, ps_belwind_nobelwind,
↳windrose_belwind_nobelwind)
print('Belwind and Nobelwind metrics with neighboring farms')
print('Area', Areas_belwind_nobelwind)
print('Nwt_row', Nwt_rows_belwind_nobelwind)
print('Nwt_row_wf_neighbors', Nwt_row_wf_neighbors_belwind_nobelwind)
```

```

# additional special case: now, the west part of Nobelwind will be allocated to
↳ another wind farm
wtdata_belwind_nobelwindeast = wtdata_temp.copy()
nobelwind_wtdata_idx = wtdata_belwind_nobelwindeast.
↳ index[(wtdata_belwind_nobelwindeast['wind_farm'] == 'Nobelwind') &
↳ (wtdata_belwind_nobelwindeast['longitude'] < 2.808) &
↳ (wtdata_belwind_nobelwindeast['latitude'] > 51.64)].tolist()
wtdata_belwind_nobelwindeast.loc[nobelwind_wtdata_idx, 'wind_farm'] =
↳ 'Northwester 2'

# run new case with the new allocation of the west part of Nobelwind to
↳ Northwester 2
winddRose_belwind_nobelwindeast, wf_lats_belwind_nobelwindeast,
↳ wf_lons_belwind_nobelwindeast, wt_xs_belwind_nobelwindeast,
↳ wt_ys_belwind_nobelwindeast, wt_lats_belwind_nobelwindeast,
↳ wt_lons_belwind_nobelwindeast, ps_belwind_nobelwindeast,
↳ wf_neighbors_belwind_nobelwindeast, wf_neighbors_flag_belwind_nobelwindeast,
↳ = calculate_windrose_and_layout_parameters(inputdata_belwind_nobelwind,
↳ ds_belwind_nobelwind, wtdata_belwind_nobelwindeast, zRef)
Target_case_belwind_nobelwindeast =
↳ identify_target_cases(inputdata_belwind_nobelwind, List_windfarm_names,
↳ print_cases=False)
Areas_belwind_nobelwindeast, Nwt_rows_belwind_nobelwindeast,
↳ Nwt_row_wf_neighbors_belwind_nobelwindeast =
↳ calculate_layout_metrics(Target_case_belwind_nobelwindeast,
↳ wt_xs_belwind_nobelwindeast, wt_ys_belwind_nobelwindeast,
↳ alphas_belwind_nobelwind, inputdata_belwind_nobelwind,
↳ wf_neighbors_belwind_nobelwindeast, wf_neighbors_flag_belwind_nobelwindeast,
↳ ps_belwind_nobelwindeast, winddRose_belwind_nobelwindeast)
print('Belwind and Nobelwind metrics with the west part of Nobelwind allocated
↳ to Northwester 2')
print('Area', Areas_belwind_nobelwindeast)
print('Nwt_row', Nwt_rows_belwind_nobelwindeast)
print('Nwt_row_wf_neighbors', Nwt_row_wf_neighbors_belwind_nobelwindeast)

# third special case, with all neighbours, but once again we will allocate the
↳ west part of Nobelwind to Northwester 2, to see the impact of this change in
↳ the metrics
inputdata_belwind_nobelwindeast_neighbors = inputdata_temp.copy()
ds_belwind_nobelwindeast_neighbors = ds_temp.copy()
wtdata_belwind_nobelwindeast_neighbors = wtdata_belwind_nobelwindeast.copy()
alphas_belwind_nobelwindeast_neighbors =
↳ define_alphas(inputdata_belwind_nobelwindeast_neighbors)

```

```

windrose_belwind_nobelwindeast_neighbors,
    ↳ wf_lats_belwind_nobelwindeast_neighbors,
    ↳ wf_lons_belwind_nobelwindeast_neighbors,
    ↳ wt_xs_belwind_nobelwindeast_neighbors,
    ↳ wt_ys_belwind_nobelwindeast_neighbors,
    ↳ wt_lats_belwind_nobelwindeast_neighbors,
    ↳ wt_lons_belwind_nobelwindeast_neighbors, ps_belwind_nobelwindeast_neighbors,
    ↳ wf_neighbors_belwind_nobelwindeast_neighbors,
    ↳ wf_neighbors_flag_belwind_nobelwindeast_neighbors =
    ↳ calculate_windrose_and_layout_parameters(inputdata_belwind_nobelwindeast_neighbors,
    ↳ ds_belwind_nobelwindeast_neighbors, wtdata_belwind_nobelwindeast_neighbors,
    ↳ zRef)
Target_case_belwind_nobelwindeast_neighbors =
    ↳ identify_target_cases(inputdata_belwind_nobelwindeast_neighbors, ['Belwind',
    ↳ 'Nobelwind'], print_cases=False)
Areas_belwind_nobelwindeast_neighbors,
    ↳ Nwt_rows_belwind_nobelwindeast_neighbors,
    ↳ Nwt_row_wf_neighbors_belwind_nobelwindeast_neighbors =
    ↳ calculate_layout_metrics(Target_case_belwind_nobelwindeast_neighbors,
    ↳ wt_xs_belwind_nobelwindeast_neighbors,
    ↳ wt_ys_belwind_nobelwindeast_neighbors,
    ↳ alphas_belwind_nobelwindeast_neighbors,
    ↳ inputdata_belwind_nobelwindeast_neighbors,
    ↳ wf_neighbors_belwind_nobelwindeast_neighbors,
    ↳ wf_neighbors_flag_belwind_nobelwindeast_neighbors,
    ↳ ps_belwind_nobelwindeast_neighbors, windrose_belwind_nobelwindeast_neighbors)
print('Belwind and Nobelwind metrics with the west part of Nobelwind allocated,
    ↳ to Northwester 2 and with all neighbors')
print('Area', Areas_belwind_nobelwindeast_neighbors)
print('Nwt_row', Nwt_rows_belwind_nobelwindeast_neighbors)
print('Nwt_row_wf_neighbors',
    ↳ Nwt_row_wf_neighbors_belwind_nobelwindeast_neighbors)

```

List of wind farms we will evaluate

- 0 : Nobelwind
- 1 : Belwind Phase 1

Number of wind directions in wind rose 36

Calculating layout metrics for 2 wind farms

Calculating wind farm Nobelwind

Calculating wind farm Belwind Phase 1

Belwind and Nobelwind metrics with neighboring farms

Area [32871010.57842534, 12072948.683177037]

Nwt_row [np.float64(12.425411438467352), np.float64(13.483146702875047)]

Nwt_row_wf_neighbors [np.float64(12.3765054461927),
np.float64(0.9019104598810022)]

List of wind farms we will evaluate

- 0 : Nobelwind

```

- 1 : Belwind Phase 1
Number of wind directions in wind rose 36
Calculating layout metrics for 2 wind farms
Calculating wind farm Nobelwind
Calculating wind farm Belwind Phase 1
Belwind and Nobelwind metrics with the west part of Nobelwind allocated to
Northwester 2
Area [10726676.854942895, 12072948.683177037]
Nwt_row [np.float64(12.186111596891472), np.float64(13.483146702875047)]
Nwt_row_wf_neighbors [np.float64(6.380290096451673),
np.float64(11.140713532515328)]
List of wind farms we will evaluate
- 45 : Nobelwind
- 46 : Belwind Phase 1
Number of wind directions in wind rose 36
Calculating layout metrics for 75 wind farms
Calculating wind farm Nobelwind
Calculating wind farm Belwind Phase 1
Belwind and Nobelwind metrics with the west part of Nobelwind allocated to
Northwester 2 and with all neighbors
Area [37099754.20522484, 32570127.182441685, 223595266.80614904,
5979251.235271297, 29451213.849088933, 23210879.74373812, 31781360.54747212,
4673805.091410637, 90371733.30498014, 19610402.41047798, 31420045.74570222,
83114135.9947617, 65210310.59815273, 54952475.80729862, 31353003.886007603,
30837801.86281134, 17142322.26115863, 35111434.89046125, 39460598.86824943,
39676923.15282976, 10172046.081156928, 40332338.495663874, 16963182.615769465,
53145265.26528686, 56703198.60820667, 73952414.46346979, 27074879.25984568,
35675917.75261409, 81601280.2126178, 56028093.38020033, 39599845.912818,
3910718.9884651904, 5976697.656496669, 31797259.510086905, 32656088.99117545,
32602075.846320257, 13845641.20977943, 15144648.405519998, 132858419.45788719,
178726469.33224976, 35458317.79879552, 9785002.270990223, 9320096.605955295,
21196991.157630783, 12307118.185067981, 10726676.854942895, 12072948.683177037,
17937590.90789824, 15597665.481888644, 16624976.635372167, 52338193.9709714,
26911453.91092847, 106410280.80199966, 14784065.629115092, 121911027.2275688,
149861551.2205044, 139175040.87979823, 35294652.69756543, 22731711.52476581,
20699849.577990104, 32528982.077054374, 32180785.259969395, 50491272.77312366,
57126337.67786044, 132001241.57075942, 405508811.1796011, 386655163.3475174,
227193470.21265638, 87749762.79937455, 107542594.9610796, 60396425.44689198,
23408840.635509197, 39095777.18761395, 67312444.71445793, 8726245.794775898]
Nwt_row [np.float64(12.250052680104341), np.float64(13.526262512942653)]
Nwt_row_wf_neighbors [np.float64(3.733021196955479),
np.float64(4.302719732400336)]

```

5.6 Result bookkeeping

At this stage, we merge the administratively split wind farms back into their original combined entities (e.g., Gemini and Gemini_2) by summing the corresponding results and removing the duplicate bookkeeping entries.

We also merge the special cases generated through administrative name manipulation, where we intentionally exploited the limitations of the vdLW code to isolate and mitigate specific type-2 errors.

```
[137]: # make a table with the results, including the names of the wind farms
results_df = pd.DataFrame({
    'Name': [inputdata_temp['Name'][i] for i in Target_case],
    'Area_km2': [Areas[i] for i in Target_case],
    'Nwt_rows': Nwt_rows,
    'Nwt_row_wf_neighbors': Nwt_row_wf_neighbors
})

# for each wind farm, based on the name, count the number of wind turbines in
# wtdata, and add it as a column in results_df
results_df['Nwt'] = results_df['Name'].apply(lambda x:
    len(wtdata_temp[wtdata_temp['wind_farm'] == x]))

display(results_df)
```

	Name	Area_km2	Nwt_rows	Nwt_row_wf_neighbors \
0	Gemini	3.179726e+07	15.780663	14.758323
1	Gemini_2	3.265609e+07	16.358903	12.909651
2	Borssele I-II	1.328584e+08	20.141456	6.313282
3	Borssele I-II_otherfarms	1.787265e+08	15.424815	5.122311
4	Norther	3.545832e+07	9.568342	7.072099
5	Thortonbank	9.785002e+06	8.992737	6.466973
6	Thortonbank_2	9.320097e+06	6.557109	0.509539
7	Rentel	2.119699e+07	11.468065	4.571686
8	Northwind	1.230712e+07	15.136354	5.819446
9	Nobelwind	3.287101e+07	12.608470	5.698924
10	Belwind Phase 1	1.207295e+07	13.526263	0.019370
11	Northwester 2	9.618695e+06	8.333316	5.338291
12	Rampion	5.233819e+07	14.165873	14.165873
13	Robin Rigg	8.726246e+06	11.968678	11.968678

	Nwt
0	75
1	75
2	94
3	79
4	44
5	30
6	24
7	42
8	72
9	50
10	56

```

11 23
12 116
13 58

```

```

[138]: results_merge_corrected_df = results_df.copy()

# merge Gemini and Gemini_2 in the results_merge_corrected_df, by adding the
# values of Gemini_2 to Gemini, and removing the row of Gemini_2
results_merge_corrected_df = results_merge_corrected_df.
    groupby(results_merge_corrected_df['Name'].str.replace('_2', ''),
    as_index=False).agg({
        'Area_km2': 'sum',
        'Nwt_rows': 'sum',
        'Nwt_row_wf_neighbors': 'sum',
        'Nwt': 'sum'
    })

# for merging Gemini, the value of Nwt_rows is equal to the value of
# Nwt_row_wf_neighbors, there are no other wind farms nearby
results_merge_corrected_df.loc[results_merge_corrected_df['Name'] == 'Gemini',
    'Nwt_rows'] = results_merge_corrected_df.
    loc[results_merge_corrected_df['Name'] == 'Gemini', 'Nwt_row_wf_neighbors']

# for Thorntonbank, the value of Nwt_rows is equal to the sum of
# Nwt_row_wf_neighbors_thorntonbank
results_merge_corrected_df.loc[results_merge_corrected_df['Name'] ==
    'Thorntonbank', 'Nwt_rows'] = np.sum(Nwt_row_wf_neighbors_thorntonbank)

# Correct Belwind for the case with neighbours
results_merge_corrected_df.loc[results_merge_corrected_df['Name'] == 'Belwind',
    'Phase 1', 'Nwt_row_wf_neighbors'] =
    Nwt_row_wf_neighbors_belwind_nobelwindeast_neighbors[1]

# For Nobelwind, we need to decrease the value of Nwt, to compensate that vdLW
# apply a factor of 2.5 to all edge turbines.
# However, the west part of Nobelwind is a single row of turbines, so we will
# decrease the value of Nwt by the number of turbines
# in the west part of Nobelwind, to avoid that, locally, Nfree is larger than
# the actual number of turbines in the farm, which would be a mistake.
# first, the case without neighbours that is the case with Belwind as a single
# neighbor
Nwt_Nobelwind_east = Nwt_row_wf_neighbors_belwind_nobelwindeast[0]
Nwt_Nobelwind_west = results_merge_corrected_df.
    loc[results_merge_corrected_df['Name'] == 'Nobelwind', 'Nwt_rows'].values[0]
    - Nwt_Nobelwind_east

```

```

Number_turbines_Nobelwind_west = len(wtdata_temp[(wtdata_temp['wind_farm'] ==
↳ 'Nobelwind') & (wtdata_temp['longitude'] < 2.808) & (wtdata_temp['latitude']
↳ > 51.64)])
# print('Number_turbines_Nobelwind_west', Number_turbines_Nobelwind_west)
# print('Nwt_Nobelwind_west', Nwt_Nobelwind_west)
# print('Nwt_Nobelwind_east', Nwt_Nobelwind_east)
# because Number_turbines_Nobelwind_west < 2.5 * Nwt_Nobelwind_west, then
↳ Nwt_Nobelwind_west = Number_turbines_Nobelwind_west / 2.5, to avoid that Nfree
↳ is larger than the actual number of turbines in the farm
if Number_turbines_Nobelwind_west < 2.5 * Nwt_Nobelwind_west:
    Nwt_Nobelwind_west = Number_turbines_Nobelwind_west / 2.5

results_merge_corrected_df.loc[results_merge_corrected_df['Name'] ==
↳ 'Nobelwind', 'Nwt_rows'] = Nwt_Nobelwind_east + Nwt_Nobelwind_west

# and now, the same for the case with all neighbors
Nwt_Nobelwind_east_neighbors =
↳ Nwt_row_wf_neighbors_belwind_nobelwindeast_neighbors[0]
Nwt_Nobelwind_west_neighbors = results_merge_corrected_df.
↳ loc[results_merge_corrected_df['Name'] == 'Nobelwind',
↳ 'Nwt_row_wf_neighbors'].values[0] - Nwt_Nobelwind_east_neighbors
# print('Nwt_Nobelwind_west_neighbors', Nwt_Nobelwind_west_neighbors)
# print('Nwt_Nobelwind_east_neighbors', Nwt_Nobelwind_east_neighbors)
if Number_turbines_Nobelwind_west < 2.5 * Nwt_Nobelwind_west_neighbors:
    Nwt_Nobelwind_west_neighbors = Number_turbines_Nobelwind_west / 2.5
results_merge_corrected_df.loc[results_merge_corrected_df['Name'] ==
↳ 'Nobelwind', 'Nwt_row_wf_neighbors'] = Nwt_Nobelwind_east_neighbors +
↳ Nwt_Nobelwind_west_neighbors

# elimate the row of Borssele I-II_otherfarms, that we created just to allocate
↳ the turbines of Borssele III-V, because it is not a real farm, and we will
↳ not use it in the analysis
results_merge_corrected_df =
↳ results_merge_corrected_df[results_merge_corrected_df['Name'] != 'Borssele
↳ I-II_otherfarms'].reset_index(drop=True)

display(results_merge_corrected_df)

```

	Name	Area_km2	Nwt_rows	Nwt_row_wf_neighbors	Nwt
0	Belwind Phase 1	1.207295e+07	13.526263	4.302720	56
1	Borssele I-II	1.328584e+08	20.141456	6.313282	94
2	Gemini	6.445335e+07	27.667975	27.667975	150
3	Nobelwind	3.287101e+07	10.380290	5.698924	50
4	Norther	3.545832e+07	9.568342	7.072099	44
5	Northwester 2	9.618695e+06	8.333316	5.338291	23
6	Northwind	1.230712e+07	15.136354	5.819446	72
7	Rampion	5.233819e+07	14.165873	14.165873	116

8	Rentel	2.119699e+07	11.468065	4.571686	42
9	Robin Rigg	8.726246e+06	11.968678	11.968678	58
10	Thortonbank	1.910510e+07	12.901224	6.976512	54

5.7 Uploading the original edge Nfree factors by vdLW

To recall: we are only going to correct some of the Type 2 errors in some of the cases, as we are limited by not changing the core code and method by vdLW.

Therefore, we will upload the results by vdLW for the Nfree edge factors

5.8 Read Nedge values from Output/Input files by vdLW

vdLW, with their code, create a file named “Output.csv”, which contains both inputs and outputs used by vdLW, including the inputs used by SLS.

An important note is that vdLW do not use all the wind farm cases. vdLW choose not to use the cases of the wind farms “Baltic 1”, “Baltic 2”, “Princess Amalia”, “Luchterduinen”. Therefore, after reading the files, we will need to remove those cases.

```
[139]: # read output/input data from vdLW's code
df = pd.read_csv("output.csv")
df_results_vdLW = df.copy() # this is just not to destroy the original
    ↪ dataframe, as we will be modifying it in the next steps

# we will eliminate the cases of Baltic I and II and Princess Amalia and
    ↪ Luchterduinen, as they are not evaluated by vdLW's work
df_results_vdLW = df_results_vdLW[~df_results_vdLW["Name"].isin(["Baltic 1",
    ↪ "Baltic 2", "Princess Amalia", "Luchterduinen"])]

# display the entire database for the first evaluation, indicating explicitly
    ↪ to plot all the rows, to avoid the Jupyter notebook from truncating the
    ↪ output
# pd.set_option("display.max_rows", None)
# pd.set_option("display.max_columns", None)
# display(db_first_eval)
```

```
[140]: # we read the number wind turbines in the wind farm
Windfarm_N_turbines = df_results_vdLW["Nt"].values

# we again read the values of the correction factor used by SLS 2026 to
    ↪ represent the fraction of power equivalent to isolated turbine
    ↪ afactor_SLS2026
NFreeSLStemp = np.array(df_results_vdLW['Nrowscale parameter'] *
    ↪ df_results_vdLW['Nturb_frontal_area'], dtype=float)
afactor_SLS2026 = NFreeSLStemp / (Windfarm_N_turbines)
```

```

# we now read the estimates by vdLW for the edge turbines, with and without
↳neighbors
Nedge_neighbors_vdLW = df_results_vdLW["Nwt_row_wf_neighbors"].values
Nedge_without_neighbours_vdLW = df_results_vdLW["Nwt_rows"].values

# vdLW use a 2.5 multiplicative factor for the edge turbines, and use only the
↳case with neighbours.
m_afactor_vdLW = 2.5

# we will also calculate the afactor by vdLW. We do not need for creating
↳figures 4 and 5,
# but we will do it for connection to figure 3, which although published in the
↳other notebook, we will reproduce here

# vdLW use a 2.5 multiplicative factor for the edge turbines, and use only the
↳case with neighbours.
m_afactor_vdLW = 2.5
a_factor_vdLW_Nfree=m_afactor_vdLW*Nedge_neighbors_vdLW/(Windfarm_N_turbines)

```

5.9 Correct Nedge with values calculated above

```

[141]: # first, we create arrays for the Nedge metrics, and equal them to the values
↳of vdLW

Nedge_neighbors_vdLW_corrected = Nedge_neighbors_vdLW .copy()
Nedge_without_neighbours_vdLW_corrected = Nedge_without_neighbours_vdLW.copy()

# now, we will correct the values of Nedge_neighbors_vdLW_corrected and
↳Nedge_without_neighbours_vdLW_corrected for the
# cases of Gemini, Thortonbank, Belwind and Nobelwind, etc, based on the
↳calculations above

for i in range(len(results_merge_corrected_df)):
    name = results_merge_corrected_df.iloc[i]["Name"]
    # print(f"Checking wind farm {name} for correction of Nedge due to
↳geometry")

    matches = np.where(df_results_vdLW["Name"].to_numpy() == name)[0]

    if len(matches) > 0:
        pos = matches[0]    # row position, safe for iloc and arrays

        # print(f"Position of wind farm {name} in df_results_vdLW: {pos}")
        # print(
        #     f"Nwt_rows and Nwt_row_wf_neighbors before correction for wind
↳farm {name}: "

```

```

#     f"{df_results_vdLW.iloc[pos]['Nwt_rows']}, "
#     f"{df_results_vdLW.iloc[pos]['Nwt_row_wf_neighbors']}"
# )

# print(
#     f"Original Nedge_isolated: {Nedge_isolated[pos]}, "
#     f"Original Nedge_Neighbours: {Nedge_Neighbours[pos]}"
# )

Nedge_without_neighbours_vdLW_corrected [pos] =
↳results_merge_corrected_df.iloc[i]["Nwt_rows"]
Nedge_neighbors_vdLW_corrected[pos] = results_merge_corrected_df.
↳iloc[i]["Nwt_row_wf_neighbors"]
# print(
#     f"Corrected Nedge_isolated:
↳{Nedge_without_neighbours_vdLW_corrected[pos]}, "
#     f"Corrected Nedge_Neighbours:
↳{Nedge_neighbors_vdLW_corrected[pos]}"
# )

else:
    print(f"Wind farm {name} not found in results_merge_corrected_df,
↳skipping correction")

```

5.10 Correct the multiply term of edge turbines, to keep it consistent with being below the actual number of turbines in the farm

This is the correction for Error Type 2 - Error 11 — Assuming Unrealistically Large Numbers of Turbines Exposed to Clean Inflow

The vdLW methodology estimates the number of turbines exposed to clean inflow using the relation: $N_{\text{free}} = 2.5M$, where M is the number of detected inlet-edge turbines. This factor of 2.5 is not a general physical constant. It comes from applying the finite-size correction with $a = 5$, corresponding to the characteristic finite-farm correction parameter used in SLS for a square wind-farm scaling approximation, and assuming that the exposed edge of a square wind farm corresponds to approximately $2\sqrt{N}$ turbines. This leads to the approximation that the number of turbines exposed to clean inflow is about 2.5 times the number of detected inlet-edge turbines. However, applying this relation as a fixed multiplier to arbitrary wind-farm geometries is a misinterpretation of both SLS and the earlier work by Sørensen and Larsen. The correction must depend on the total number.

Here, for wind farms smaller than 25 turbines, we correct the value.

```

[142]: # define a simple approximation for very small values of number of wind
↳turbines in the wind farm

def simple_afactor_model(Nt):
    N_squared_root = np.sqrt(Nt)

```

```

    # simple formula based on the results of Larsen and Sorensen, with a
    ↪correction factor to ensure
    # that at N_squared_root=1, the fraction of clean flow is 1.0. This
    ↪approximation loses validity for any large wind farm
    a = 0.5788*(N_squared_root-1)+1.0

    # we cap a at 6 based on what was our knowledge then of the slope
    a = np.minimum(a, 5)/2

    if Nt >25:
        a =np.nan # for large wind farms, this simple model is not valid, so
        ↪we set it to NaN

    return a

```

```

[143]: ## Correct the multiply term of edge turbines, to keep it consistent with being
    ↪below the actual number of turbines in the farm

m_a_factor_vdLW_Nfree_corrected = np.
    ↪zeros_like(Nedge_neighbors_vdLW_corrected)+2.5

for i in range(len(Nedge_neighbors_vdLW_corrected)):
    Nt = Windfarm_N_turbines[i]
    if Nt < 25:
        print(f"Correcting multiplicative factor for edge turbines for wind
        ↪farm {df_results_vdLW.iloc[i]['Name']} with Nt={Nt}")
        # for small wind farms, we will use the simple approximation for the
        ↪fraction of clean flow, to correct the multiplicative factor of edge turbines
        m_a_factor_vdLW_Nfree_corrected[i] = simple_afactor_model(Nt)

```

Correcting multiplicative factor for edge turbines for wind farm Albatros with Nt=16

Correcting multiplicative factor for edge turbines for wind farm Alpha Ventus with Nt=12

Correcting multiplicative factor for edge turbines for wind farm Northwester 2 with Nt=23

5.11 Correct Type 2 Error 10— Ignoring the Temporal Evolution of Wind-Farm Clusters

The vdLW evaluation assumes a fixed 2025 wind-farm configuration, despite the measured production data spanning years during which neighboring wind farms were progressively constructed and commissioned. The vdLW simulations therefore compare measured production from partially developed wind-farm clusters against aerodynamic environments corresponding to much later cluster-development stages. As a consequence, the aerodynamic environment used in the vdLWsimulations does not correspond to the real aerodynamic environment experienced by the wind farms during the years covered by the measured production data. This creates a mismatch between: – the real neighboring wind farms present during operation, – and the neighboring wind farms included in

the vdLW simulations. The error is particularly important for rapidly evolving offshore wind-farm clusters, where neighboring wind farms were added progressively over time and the aerodynamic environment evolved continuously during the measurement period.

In this part, we upload a table with an indication of the number of years the wind farm was without and with neighbours, based on the real capacity factor database used.

```
[144]: # load table with the breakdown of year of isolated wind farma and with
        ↪neighbours
infile_years_neighbours='year_correction_isolated_neighbours.xlsx'
df_years_neighbours = pd.read_excel(infile_years_neighbours,
        ↪sheet_name='Sheet1')

years_isolated = df_years_neighbours['Years isolated'].values
years_neighbours = df_years_neighbours['Years neighbours'].values

Nedge_corrected = Nedge_neighbors_vdLW_corrected * years_neighbours /
        ↪(years_isolated + years_neighbours) +
        ↪Nedge_without_neighbours_vdLW_corrected * years_isolated / (years_isolated +
        ↪years_neighbours)
```

5.12 Calculate correct capacity factors

We have now setup all the corrections. We can now calculate the correct capacity factor. We will upload the corrected capacity factors for the isolated turbine and infinite wind farm for each wind farm case, without Type 1 errors, calculated in the other notebook

```
[145]: # load values free of Type 1 errors
infile_clean_of_type1_errors = 'vdLW_corrected_capacity_factors.xlsx'
df_clean_of_type1_errors = pd.read_excel(infile_clean_of_type1_errors,
        ↪sheet_name='Sheet1')

# display all columns and rows of the dataframe, to check the values
# pd.set_option('display.max_rows', None)
# pd.set_option('display.max_columns', None)
# display(df_clean_of_type1_errors)

CFisolated_vdLW_free_of_type1_errors = np.
        ↪array(df_clean_of_type1_errors['Cf_Free_vdLW_correct_all'].values,
        ↪dtype=float)
CFinfinetewf_vdLW_free_of_type1_errors = np.
        ↪array(df_clean_of_type1_errors['Cf_Inf_vdLW_correct_all'].values,
        ↪dtype=float)
CFmodel_vdLW_free_of_type1_errors = np.
        ↪array(df_clean_of_type1_errors['Cf_Model_vdLW_correct_all'].values,
        ↪dtype=float)
```



```

# we also upload the cased for Uin=0 and Uout=infinity, to later determine the
↳Wind Farm Wind Factor

CFisolated_Uin0_Uoutinf = np.
↳array(df_clean_of_type1_errors['Cf_Free_vdLW_correct_Uin0_Uoutinf'].values,↳
↳dtype=float)
CFinfinitlew_Uin0_Uoutinf = np.
↳array(df_clean_of_type1_errors['Cf_Inf_vdLW_correct_Uin0_Uoutinf'].values,↳
↳dtype=float)

```

```

[146]: # calculate the Capacity factor corrected of Type 1 errors and with some Type 2
↳errors mitigated

fraction_Nfree = m_a_factor_vdLW_Nfree_corrected * Nedge_corrected /↳
↳Windfarm_N_turbines

CF_vdLW_free_Type1_and_2_errors = CFisolated_vdLW_free_of_type1_errors *↳
↳fraction_Nfree + CFinfinitlew_vdLW_free_of_type1_errors * (1-fraction_Nfree)

```

5.13 Read the results published by SLS 2026, from comparison and generation of the figures

We already removed the wind farms “Baltic 1”, “Baltic 2”, “Princess Amalia”, “Luchterduinen” because vdLW removed them from their work

```

[147]: # read from excel file the dataframe of SLS_2026
df_SLS_2026 = pd.read_excel("SLS_2026_without_Baltic1_2_LuchD_PAM.xlsx",↳
↳sheet_name="Sheet1")

# createa array of capacity factors from the dataframe of SLS_2026,
# this will be used to compare with the results of the corrected vdLW model
SLS_2026_CF = np.array(df_SLS_2026["Capacity Factor model"].values,↳
↳dtype=float)/100

# also read the real values of capacity factor

SLS_2026_CF_real = np.array(df_SLS_2026["Capacity Factor real"].values,↳
↳dtype=float)/100

```

5.14 Create Figure 5

```

[148]: # simple code for the linear fit

def linear_fit(x, y):
    x = np.asarray(x)
    y = np.asarray(y)

```

```

# fit  $y = a*x + b$ 
a, b = np.polyfit(x, y, 1)

# predictions
y_pred = a * x + b

#  $R^2$ 
ss_res = np.sum((y - y_pred)**2)
ss_tot = np.sum((y - np.mean(y))**2)
r2 = 1 - ss_res / ss_tot

return a, b, r2

```

5.14.1 Now, we plot Figure 5

Note that the regression shown here is slightly better than the one published in the rebuttal document. While cleaning and restructuring the code, we identified a small error in one data point related to the merging of a split wind farm. After correcting this bookkeeping issue, the resulting regression became marginally stronger.

```

[149]: # determine terms of linear regression between the capacity factor of vdLW with
      ↪ Type 1 error corrected Nfree and the SLS_2026_CF,
# to see how well they correlate, and to see if there is a bias in the results
a_reg, b_reg, r2_reg = linear_fit(SLS_2026_CF, CF_vdLW_free_Type1_and_2_errors)
print("Linear fit: a =", a_reg, "b =", b_reg, "R^2 =", r2_reg)

# Plot figure 5

plt.rcParams.update(plt.rcParamsDefault)
plt.rcParams.update({'font.size': 14}) # fontsize 14 for all labels and ticks

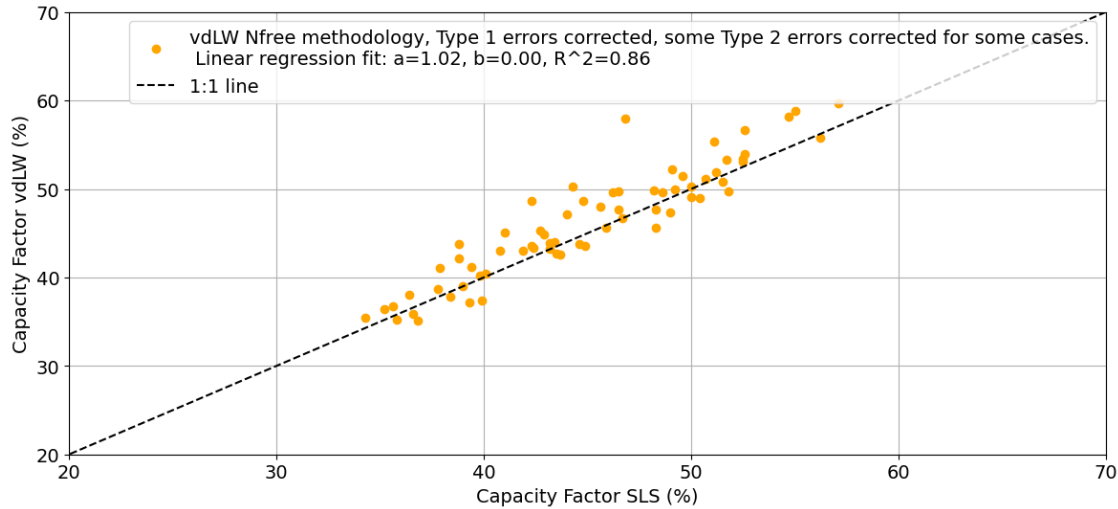
# we will now plot in the comparison between model and real CF
fig, ax = plt.subplots(figsize=(14, 6))
ax.plot(SLS_2026_CF*100, CF_vdLW_free_Type1_and_2_errors*100, 'o', color='orange',
        label=f'vdLW Nfree methodology, Type 1 errors corrected, some Type 2 ↪
        ↪ errors corrected for some cases.\n Linear regression fit: a={a_reg:.2f}, ↪
        ↪ b={b_reg:.2f}, R^2={r2_reg:.2f}')
ax.plot([0., 100.], [0., 100.], "k--", label="1:1 line")

ax.set_xlim(20, 70)
ax.set_ylim(20, 70)
ax.set_xlabel('Capacity Factor SLS (%)')
ax.set_ylabel('Capacity Factor vdLW (%)')
ax.legend()
# ax.set_title('Comparison of modelled and 2025 and Paul')
ax.grid()

```

```
plt.show()
```

Linear fit: $a = 1.0234749609142808$ $b = 0.0020294479842073642$ $R^2 = 0.863909547992426$



5.15 Plot Figure 6

Here we plot Figure 6, the corrected results using the vdLW framework, plotted against the theoretical limit.

First, we calculate the theoretical limit curves, also using the corrected vdLW code, then, the equivalent Wind Farm Wind Factor, and then the full plot

```
[150]: # again, we copy and correct the function of vdLW to eliminate the eps2 error

from scipy.special import gamma, gammaln
from scipy.optimize import fsolve

class MinimalisticPredictionModel():
    """Sørensen, J.N.; Larsen, G.C.
    A Minimalistic Prediction Model to Determine Energy Production and Costs of
    ↪Offshore Wind Farms.
    Energies 2021, 14, 448. https://doi.org/10.3390/en14020448"""

    def __init__(self, correction_factor, latitude, CP, Uin, Uout, rho,
    ↪correct_eps2=False):
        ##### Start of note by SLS
        ↪#####
        # The original code by vdLW had the function header as: def
        ↪__init__(self, correction_factor, latitude, CP, Uin, Uout, rho):
```

```

#
# We have added an optional argument 'correct_eps2' to the function
↳header. This argument is a boolean.
# If False (default), the function will use the original calculation
↳for eps2 as given by vdLW.
# If True, the function will use the correct formula for eps2
#
##### End of note by SLS  ␣
↳#####
"""
Parameters
-----
correction_factor : int, float or function
    Finite-size wind farm correction which multiplied with sqrt(Nturb)
↳gives
    the number of wind turbines exposed to the free wind
latitude : int or float
    latitude [deg] used to calculate the coriolis parameter
CP : float, optional
    Wind turbine power coefficient
Uin : int or float, optional
    Wind turbine cut-in wind speed
Uout : int or float, optional
    Wind turbine cut-out wind speed
rho : float, optional
    Air density
"""

self.CP = CP
self.Uin = Uin
self.Uout = Uout
omega = 2 * np.pi / (24 * 60 * 60) # earth rotation speed
self.f = 2 * omega * np.sin(np.deg2rad(latitude))
self.correction_factor = correction_factor
self.rho = rho

##### Start of code alteration by SLS to add correct_eps2
↳argument, see note at the beginning of the class #####
self.correct_eps2 = correct_eps2
##### End of code alteration by SLS to add correct_eps2
↳argument #####

def predict(self, Pg, CT, D, H, z0, Aw, kw, Nturb, Area):
    """
    Inputs:
        Pg      - [W] Nameplate capacity (generator power)

```

```

        CT      - [-] Thrust coefficient
        D        - [m] Rotor diameter
        H        - [m] Tower height
        z0       - [m] roughness length
        Aw       - [m/s] Weibull scale parameter
        kw       - [-] Weibull shape parameter
        Nturb    - [-] Number of turbines
        Area     - [m2] Area of wind farm

    Outputs:
        power - [Wh] Annual energy production of the wind farm
        ws_eff - [m/s] Effective mean wind speed including wakes
    """

    kappa = 0.4 # [-] Von Karman constant
    Uin, Uout = self.Uin, self.Uout

    # factor defined by Frandsen, should be used instead of f in eq 13 and
    ↪19 (typos in paper)
    fm = self.f * np.exp(4)
    delta = np.log(H / z0) # eq 19

    # Mean spacing between wt in diameters, eq 8
    S = np.sqrt(Area) / (D * (np.sqrt(Nturb) - 1))

    # Rated wind speed [m/s], eq 4
    Ur = (8 * Pg / (self.rho * np.pi * D**2 * self.CP))**(1 / 3)

    # Power modeled as  $P = \alpha * U^3 + \beta$ , eq 1
    alpha = Pg / (Ur**3 - Uin**3) # [(m/s)-3] eq 2
    beta = -Pg * Uin**3 / (Ur**3 - Uin**3) # [-], eq 2

    Uh0 = Aw * gamma(1 + 1 / kw) # [m/s] Mean velocity at hub height
    Ctau = np.pi * CT / (8 * S * S) # [-] Wake parameter, rotor ct smeared
    ↪on WT area
    nu = np.sqrt(0.5 * Ctau) * D / (kappa**2 * H) * delta # [-] wake eddy
    ↪viscosity

    # Finite-size wind farm correction, section 2.5
    correction_factor = self.correction_factor
    if hasattr(correction_factor, '__call__'):
        correction_factor = correction_factor(Uh0, S, Nturb)
    Nfree = correction_factor * np.sqrt(Nturb) # Number of wt exposed to
    ↪the free wind
    Nfree = np.minimum(Nfree, Nturb) # To make sure Nfree/Nturb is not
    ↪larger than one, not yet implemented in PyWake

```

```

# Geostrophic wind speed
G_last = Uh0
for n in range(10):
    G = Uh0 * (1 + np.log(G_last / (fm * H)) / delta)
    dG = abs(G - G_last)
    if dG < 1e-5:
        break
    G_last = G

gam = np.log(G / (fm * H)) # eq 19

# Mean velocity at hub height without wake effects from geostrophic wind
Uh0 = G / (1 + gam / kappa * np.sqrt((kappa / delta)**2)) # eq 13, ct=0

# Power without wake effects, eq 16 modified by
# - add gamma(1 + 3 / kw) to cancel out normalization in scipy's
→gammainc
# - gammainc terms swapped (typo in paper)
def get_Py(Aw, Aw_out): # Yearly power
    return alpha * Aw**3 * gamma(1 + 3 / kw) * (gammainc(1 + 3 / kw,
→(Ur / Aw)**kw) - gammainc(1 + 3 / kw, (Uin / Aw)**kw)) + \
        beta * (np.exp(-(Uin / Aw)**kw) - np.exp(-(Ur / Aw)**kw)) + \
        Pg * (np.exp(-(Ur / Aw)**kw) - np.exp(-(Uout / Aw_out)**kw))

P_y = get_Py(Aw, Aw)

# Without cutin and cutout
# def get_Py(Aw): # Yearly power
#     x = (Ur / Aw) ** kw
#     ks = 1 + 3 / kw
#     return Pg * (x ** (1.0 - ks) * gamma(ks) * gammainc(ks, x) + np.
→exp(-x))
# def get_Py(Aw): # Yearly power
#     x = (Ur / Aw)
#     ks = 1 + 3 / kw
#     return Pg * (x ** (-3) * gamma(ks) * gammainc(ks, x ** kw) + np.
→exp(-x ** kw))
# P_y = get_Py(Aw)

# Mean velocity at hub height with wake effects
z0_lo = z0 # / (1 - D / (2 * H))**(nu / (1 + nu)) # ???
Uh = G / (1 + gam * np.sqrt(Ctau + (kappa / np.log(H / z0_lo))**2) /
→kappa)

# eq 18. The paper states 3/2 instead of 3.2 which is either a typo or
→an initial guess

```

```

    # eps2 corresponds to eps(Uout) in paper and eps2(Ur)=eps1
    eps1 = (1 + gam / delta) / (1 + gam / kappa * np.sqrt(Ctau + (kappa /
↪delta)**2))
    eps2 = (1 + gam / delta) / (1 + gam / kappa * np.sqrt(Ctau * (Ur /
↪Uh)**3.2 + (kappa / delta)**2))

    ##### Start of code alteration by SLS to correct eps2, see
↪note at the beginning of the class #####
    if self.correct_eps2: #
        # use the correct formulation
        eps2 = (1 + gam / delta) / (1 + gam / kappa * np.sqrt(Ctau * (Ur /
↪Uout)**3.2 + (kappa / delta)**2))
        ##### End of code alteration by SLS to correct eps2
↪#####

    # print('e', eps1, correction_factor)
    # Power production with wake effects
    P_WFy = get_Py(eps1 * Aw, eps2 * Aw)

    power = ((Nturb - Nfree) * P_WFy + Nfree * P_y)
    ws_eff = ((Nturb - Nfree) * Uh + Nfree * Uh0) / Nturb

    # Phi without cutin and cutout
    def get_phi(x): # Yearly power
        # x = U_r / (Uh0 eps)
        ks = 1 + 3 / kw
        Gm = gamma(1 + 1 / kw)
        return Pg * ((x * Gm) ** (-3) * gamma(ks) * gammainc(ks, (x * Gm)
↪** kw) + np.exp(-(x * Gm) ** kw)) - power / Nturb

    root = fsolve(get_phi, x0=1) # initial guess
    phi = root[0]

    return power, ws_eff, P_y, P_WFy, Uh, G, phi, eps1, Ur # small
↪modification to return eps1 and Ur

```

```

[151]: # we create an instance of the class, with Uin=0 and Uout=infinity, and eps2
↪corrected

# parameters as defined by vdLW, these are the same for all cases, as they are
↪not case specific,
# but they are used as input for the predict function, so we define them here

```

```

vdLW_z0 = 10 ** (-4) # Roughness length [m]
vdLW_zRef = 100.0 # Reference height at which GWA data is taken [m]
vdLW_kw = 2.4 # Weibull shape parameter [-]
vdLW_rho = 1.225 # Air density [kg/m^3]
vdLW_lat = 55 # Latitude [degree]
vdLW_CP = 0.46 # Turbine power coefficient [-]
vdLW_CT = 0.75 # Turbine thrust coefficient [-]
vdLW_Uin = 0.0 # Cut-in wind speed [m/s]
vdLW_Uout = 1e6 # Cut-out wind speed [m/s]

vdLW_model = MinimalisticPredictionModel(correction_factor=0,
    ↪latitude=vdLW_lat, CP=vdLW_CP, Uin=0, Uout=np.inf, rho=vdLW_rho,
    ↪correct_eps2=True)

# now, we setup the limit curve

Turbine_rated_power_lim = 15e6
CT_lim = vdLW_CT
Turbine_rotor_diameter_lim = 240
Turbine_hub_height_lim = 150
z0_lim = vdLW_z0
Wind_lambda_lim = 15
kw_lim = vdLW_kw
Windfarm_area_lim = (Turbine_rotor_diameter_lim*5*100)**2
Windfarm_N_turbines_lim = np.linspace(10, 1000, 100)**2

power_lim = np.zeros(len(Windfarm_N_turbines_lim))
ws_eff_lim = np.zeros(len(Windfarm_N_turbines_lim))
P_y_lim = np.zeros(len(Windfarm_N_turbines_lim))
P_WFy_lim = np.zeros(len(Windfarm_N_turbines_lim))
Uh_lim = np.zeros(len(Windfarm_N_turbines_lim))
G_lim = np.zeros(len(Windfarm_N_turbines_lim))
phi_lim = np.zeros(len(Windfarm_N_turbines_lim))
eps1_lim = np.zeros(len(Windfarm_N_turbines_lim))
Ur_lim = np.zeros(len(Windfarm_N_turbines_lim))

for i in range(len(Windfarm_N_turbines_lim)):
    power_lim[i], ws_eff_lim[i], P_y_lim[i], P_WFy_lim[i], Uh_lim[i], G_lim[i],
    ↪phi_lim[i], eps1_lim[i], Ur_lim[i] = vdLW_model.predict(
        Turbine_rated_power_lim, CT_lim, Turbine_rotor_diameter_lim,
    ↪Turbine_hub_height_lim, z0_lim, Wind_lambda_lim, kw_lim,
        Windfarm_N_turbines_lim[i], Windfarm_area_lim)

CF_lim = power_lim / (Turbine_rated_power_lim * Windfarm_N_turbines_lim)

```



```

Uinf_lim = Wind_lambda_lim * gamma(1 + 1/kw_lim)
WFWF_lim = Ur_lim / Uh_lim

# create interpolation function for the upper limit curve, to be able to
↳ calculate the phi values for the cases in the database
CF_function_of_WFWF = interp1d(WFWF_lim, CF_lim, bounds_error=False,
↳ fill_value="extrapolate")
WFWF_function_of_CF = interp1d(CF_lim, WFWF_lim, bounds_error=False,
↳ fill_value="extrapolate")

```

5.15.1 Calculate the value of the Wind Farm Wind Factor ϕ

```

[152]: # first, we calculate the capacity factor value based in operation Uin=0 to
↳ Uinf=infinity, to be able to determine the Wind Farm Wind Factor

CF_vdLW_free_Type1_and_2_errors_Uin0_Uoutinfinity = CFisolated_Uin0_Uoutinf *
↳ fraction_Nfree + CFinfinitewf_Uin0_Uoutinf * (1-fraction_Nfree)

# determine the Wind Farm Wind Factor for each case in the database

WFWF_vdLW_free_Type1_and_2_errors =
↳ WFWF_function_of_CF(CF_vdLW_free_Type1_and_2_errors_Uin0_Uoutinfinity)

```

5.15.2 Plot Figure 6

```

[153]: ## plot CF_lim vs WFWF_lim
fig, ax = plt.subplots(figsize=(10, 6))
ax.plot(WFWF_lim, CF_lim*100, '-', color='navy', label=r'Theoretical limit
↳ curve')
ax.plot(WFWF_lim, CF_lim*100 * 0.9, '--', color='blue', label=r'90% Theoretical
↳ limit curve')
# ax.scatter(WFWF_random, CF_random*100, color='cyan', label=r'Random points on
↳ upper limit curve')
# ax.plot(wf_previous, Cf_Model_paul_correct_lambda_eps_Uin_Uout_lat*90, 'o',
↳ color='orange', label=r'previous wfwf')
# ax.plot(wf_corrected, Cf_Model_paul_correct_lambda_eps_Uin0_Uoutinf_lat*90,
↳ 'o', color='red', label=r'corrected wfwf')
ax.plot(WFWF_vdLW_free_Type1_and_2_errors, SLS_2026_CF_real*100, '*',
↳ color='green', alpha=0.95, label=r'real data, vdLW methodology with
↳ corrections')
# ax.plot(wf_corrected, CF_2025_real*100, '+', color='red', label=r'corrected
↳ wfwf')
ax.set_xlim(1, 2)
ax.set_ylim(20, 60)
ax.set_xlabel('Wind Farm Wind Factor ')

```

```

ax.set_ylabel('Capacity Factor (CF) %')
ax.legend()
ax.grid(True)
plt.show()

```

