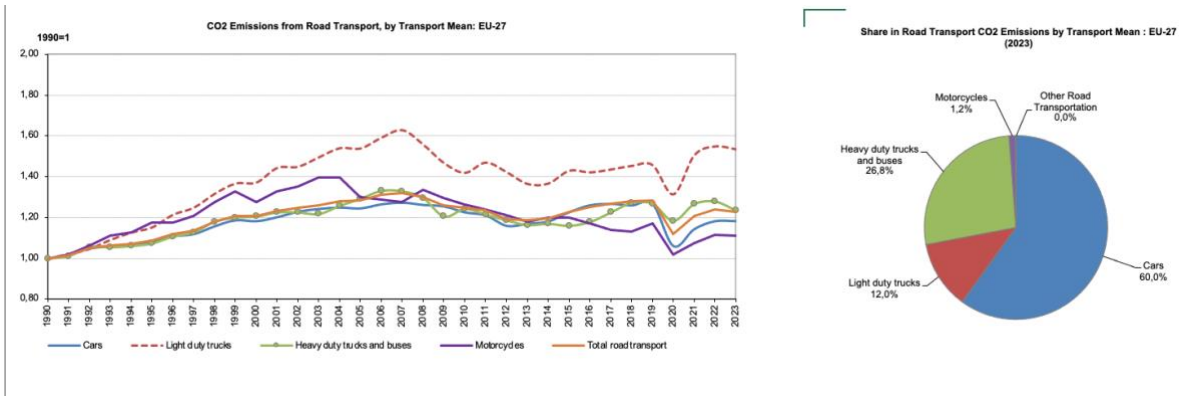
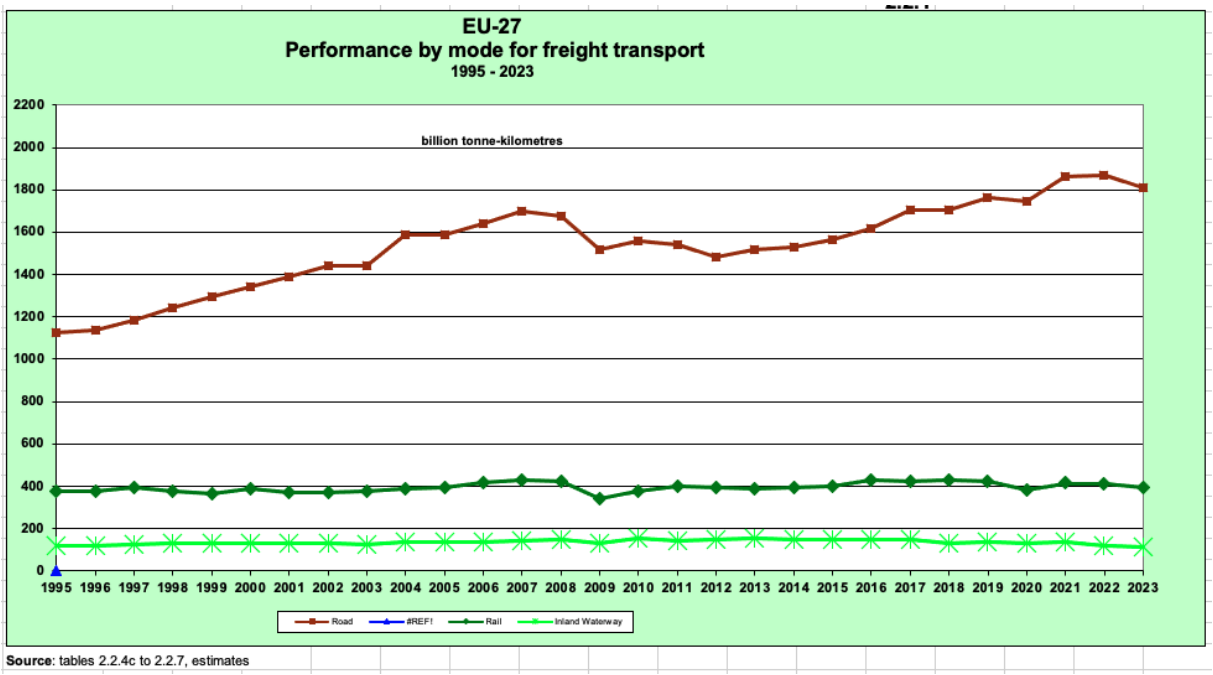
Annexes

Annexe A - Émissions de gaz à effet de serre en Europe



Source : European Commission, EU transport in figures: Statistical Pocketbook 2025, fichiers Excel, 2025. Part 3

Annexe B - Répartition modale du transport de fret en Europe



Source : European Commission, EU transport in figures: Statistical Pocketbook 2025, fichiers Excel, 2025. Part 2 section 2

Annexe C - Analyse descriptive des données

C.0 Étapes de nettoyage des données

- Étape 1 : Conversion des unités

  Appliquée en premier, identique pour NUTS 2 et NUTS 3 :

  Durées : secondes → heures (÷ 3 600)

  Longueurs : mètres → kilomètres (× 0,001)

  Quantités : inchangées (tonnes)

  Coûts : inchangés (€/tonne)

- Étape 2 : Suppression des quantités nulles

  Toutes les paires OD où la somme qty1 + qty2 + qty3 = 0 sont exclues. Aucun flux n'étant observé sur ces relations, elles n'apportent aucune information au modèle.

  NUTS 2 : 130 523 observations conservées

  NUTS 3 : 1 201 394 observations conservées après cette étape

- Étape 3 : Définition de la disponibilité des modes

  Un mode i est considéré comme disponible sur une paire OD si et seulement si son coût est non nul et non manquant : $avail_i = 1$ si $cost_i > 0$, 0 sinon. Dans le modèle NODUS, un coût nul signifie qu'il n'existe pas d'itinéraire praticable pour ce mode entre l'origine et la destination. Les paires OD sans aucun mode disponible sont ensuite exclues.

- Étape 4 : Filtrage des outliers sur les quantités fluviales

  NUTS 2 : le filtrage a été réalisé en amont par Jourquin avant la mise à disposition des données. Le jeu de données fourni est déjà nettoyé : les outliers de quantités fluviales ont été supprimés et une seule classe de péniches est retenue.

  NUTS 3 : le filtrage est réalisé dans le script Python, groupe par groupe, uniquement sur les quantités fluviales (qty2, qty2from, qty2to), selon les seuils suivants :

Filtrage des valeurs aberrantes (NUTS3 Allemagne)						
Groupe	Marchandise	N avant filtrage	N après filtrage	N supprimés	% supprimés	Filtre appliqué
0	Produits agricoles	114 164	113 412	**752**	**0,66%**	qty2to < 200 000
1	Denrées alimentaires et fourrage	93 458	93 458	0	0,00%	Aucun
2	Combustibles solides	94 247	93 570	**677**	**0,72%**	qty2from < 500 000 et qty2to < 500 000
3	Produits pétroliers	89 567	89 567	0	0,00%	Aucun
4	Minerais et ferrailles	141 435	140 345	**1 090**	**0,77%**	qty2 < 1 200 000 et qty2to < 300 000
5	Produits métallurgiques	120 323	120 323	0	0,00%	Aucun
6	Minéraux et mat. de construction	150 802	150 802	0	0,00%	Aucun
7	Engrais	124 647	124 014	**633**	**0,51%**	qty2to < 50 000
8	Produits chimiques	138 132	138 132	0	0,00%	Aucun
9	Machines et articles manufacturés	134 619	134 619	0	0,00%	Aucun
TOTAL		**1 201 394**	**1 198 242**	**3 152**	**0,26%**	

Ces 3 152 observations représentent 0,26 % du total NUTS 3 - le filtrage est donc marginal et ne modifie pas les conclusions.

- Étape 5 : Normalisation des poids

  Les quantités servent de poids dans la log-vraisemblance MNL with counts. Pour éviter que les groupes avec de grandes quantités absolues ne dominent numériquement l'optimisation, les poids sont normalisés à chaque groupe : weight = N × (qty1 + qty2 + qty3) / total_qty, où N est le nombre d'observations du groupe.

C.1 Statistique descriptive (NUTS 2)

Statistiques descriptives NUTS2 (Europe, n = 130 523 paires OD) *Coût : €/tonne | Durée : heures | Quantité : tonnes*															
Groupe	**Marchandise**	**Mode**	**N OD (mode dispo)**	**N OD (qty > 0)**	**Coût moyen**	**Coût médian**	**Écart-type du coût**	**Coût Min**	**Coût Max**	**Durée Moyenne**	**Durée Médiane**	**Durée Min**	**Durée Max**	**Quantité Moyenne**	**Quantité Médiane**
0	Produits agricoles	Route	17 118	16 613	53,96	46,98	30,45	6,36	197,03	12,10	10,43	0,72	46,28	20 119	2 697
		Voie navigable	3 287	524	10,26	9,09	4,94	3,27	32,75	121,49	106,37	30,99	412,78	26 177	6 178
		Fer	17 019	1 439	43,52	40,19	14,34	20,13	110,14	60,93	59,15	48,43	96,52	21 134	3 719
1	Denrées alimentaires et fourrage	Route	16 989	16 770	52,68	46,76	29,05	6,36	187,96	11,79	10,37	0,72	44,12	26 606	2 940
		Voie navigable	3 252	540	9,46	8,41	4,50	3,02	30,57	119,93	105,44	30,99	411,51	33 466	6 887
		Fer	16 888	514	20,08	18,24	8,88	5,25	59,94	60,63	59,12	48,43	93,41	17 187	2 709
2	Combustibles solides	Route	6 994	6 755	40,01	35,03	22,77	5,96	167,34	8,35	7,23	0,72	36,89	6 778	654
		Voie navigable	1 776	195	7,79	6,83	3,51	3,00	23,91	97,50	84,08	30,99	321,31	149 546	14 971
		Fer	6 967	1 164	15,17	13,79	6,43	4,92	52,51	56,97	55,82	48,43	88,09	123 814	16 031
3	Produits pétroliers	Route	8 245	7 444	38,37	34,13	19,86	7,07	149,92	8,54	7,48	0,72	36,43	24 532	1 858
		Voie navigable	1 905	283	11,35	9,54	6,12	3,28	39,77	85,09	69,83	17,19	324,34	207 675	66 148
		Fer	8 174	1 591	23,82	22,08	8,45	9,77	74,41	57,26	56,17	48,43	89,04	16 216	2 812
4	Minerais et ferrailles	Route	4 871	3 469	31,38	27,54	15,75	5,96	127,88	6,42	5,55	0,72	28,04	159 112	14 565
		Voie navigable	1 409	405	6,72	5,93	2,93	3,00	19,49	82,65	71,67	30,99	259,93	69 885	9 231
		Fer	4 841	2 230	12,87	11,85	4,58	4,92	39,40	55,06	54,21	48,43	77,17	32 050	4 852
5	Produits métallurgiques	Route	17 129	16 467	49,95	45,69	25,48	6,53	179,97	11,42	10,37	0,72	43,45	11 102	1 618
		Voie navigable	3 439	360	12,28	10,97	5,79	3,86	41,45	126,74	111,82	30,99	458,15	23 893	5 760
		Fer	17 032	2 406	53,29	50,87	14,47	27,45	124,80	60,17	59,08	48,43	92,68	17 074	3 490
6	Minéraux et mat. de construction	Route	9 007	8 066	37,22	33,98	18,60	6,11	192,52	7,85	7,11	0,72	43,45	124 794	8 728
		Voie navigable	2 381	499	8,11	7,24	3,61	3,01	22,97	101,89	89,77	30,99	308,20	132 907	36 744
		Fer	8 945	2 272	19,99	18,88	6,31	8,86	68,56	56,52	55,72	48,43	91,82	76 072	14 494
7	Engrais	Route	11 263	10 698	46,93	42,04	25,94	5,96	199,60	9,90	8,80	0,72	44,12	3 825	409
		Voie navigable	2 608	409	8,74	7,79	4,08	3,00	28,47	110,72	97,50	30,99	384,62	9 560	2 553
		Fer	11 218	1 903	17,12	15,69	7,30	4,92	60,42	58,60	57,41	48,43	94,68	39 829	6 324
8	Produits chimiques	Route	17 352	16 713	50,77	45,50	27,04	6,57	190,42	11,54	10,25	0,72	45,72	16 985	2 219
		Voie navigable	3 552	486	14,06	12,38	7,34	3,29	53,08	111,88	97,30	18,39	450,53	63 929	13 868
		Fer	17 300	2 331	39,59	36,99	13,02	17,48	103,95	60,35	58,94	48,43	95,02	15 799	3 258
9	Machines et articles manufacturés	Route	21 555	20 874	55,34	49,79	29,34	6,53	182,88	12,74	11,38	0,72	44,16	47 263	7 308
		Voie navigable	3 872	266	13,04	11,55	6,50	3,86	41,88	135,34	118,45	30,99	463,13	206 486	24 744
		Fer	21 457	2 331	56,31	53,06	16,72	27,45	129,04	61,55	60,08	48,43	94,61	38 874	6 536

C.2 Statistique descriptive (NUTS 3)

Statistiques descriptives NUTS3 (Allemagne, n = 1 201 394 paires OD)

Coût : €/tonne | Durée : heures | Quantité : tonnes

Groupe	Marchandise	Mode	N OD (mode dispo)	N OD (qty > 0)	Coût Moyen	Coût Médian	Écart-type du coût	Coût Min	Coût Max	Durée Moyenne	Durée Médiane	Durée Min	Durée Max	Quantité Moyenne	Quantité Médiane
0	Produits agricoles	Route	113 412	93 650	24,05	22,14	10,05	6,06	66,50	4,95	4,49	0,65	15,09	1 293	62
		Voie navigable	53 682	4 112	7,84	7,05	3,42	2,75	21,07	83,36	74,02	23,34	239,41	649	306
		Fer	113 412	26 422	28,88	28,05	4,64	19,72	47,17	53,10	52,66	48,22	62,88	207	13
1	Denrées alimentaires et fourrage	Route	93 458	91 891	20,24	20,32	6,00	6,06	54,70	4,04	4,05	0,65	12,27	2 526	116
		Voie navigable	46 172	8 248	6,45	5,92	2,75	2,51	19,31	73,08	66,43	23,34	235,53	914	307
		Fer	93 458	2 481	9,83	9,81	1,94	4,98	20,72	52,20	52,19	48,22	61,16	643	83
2	Combustibles solides	Route	93 570	87 014	20,91	20,51	7,32	5,64	63,38	4,07	3,98	0,65	13,59	296	23
		Voie navigable	38 792	2 081	6,29	5,69	2,84	2,49	21,37	71,25	63,75	23,34	261,65	2 006	307
		Fer	93 570	22 841	9,50	9,35	2,17	4,66	21,76	52,25	52,12	48,22	62,46	7 028	90
3	Produits pétroliers	Route	89 567	88 057	20,14	20,00	6,08	6,79	55,03	3,98	3,95	0,65	12,71	1 058	32
		Voie navigable	44 242	10 260	7,98	7,19	3,89	2,65	26,95	61,56	54,62	14,34	229,81	2 236	614
		Fer	89 567	1 656	15,69	15,59	2,66	9,42	30,57	52,15	52,09	48,22	61,50	2 499	166
4	Minerais et ferrailles	Route	140 345	123 517	25,50	24,89	9,35	5,64	66,59	5,10	4,96	0,65	14,31	2 751	75
		Voie navigable	64 823	4 299	7,57	7,00	3,22	2,49	21,55	87,47	80,27	23,34	264,00	860	307
		Fer	140 345	28 695	10,73	10,56	2,65	4,66	22,25	53,27	53,13	48,22	62,88	698	12
5	Produits métallurgiques	Route	120 323	86 767	24,76	22,38	10,23	6,24	64,87	5,21	4,62	0,65	15,09	894	48
		Voie navigable	58 285	2 089	9,41	8,58	3,92	3,32	26,41	85,73	77,24	23,34	259,87	1 211	307
		Fer	120 323	50 529	38,24	37,06	5,71	26,97	59,18	53,33	52,80	48,22	62,85	635	22
6	Minéraux et mat. de construction	Route	150 802	124 465	25,80	24,69	10,06	5,79	68,81	5,23	4,98	0,65	15,09	4 091	131
		Voie navigable	71 711	16 076	7,57	6,87	3,36	2,50	21,56	87,36	78,61	23,34	264,00	1 864	614
		Fer	150 802	37 965	15,70	15,35	3,32	8,56	28,73	53,41	53,15	48,22	62,88	1 404	63
7	Engrais	Route	124 014	75 810	25,38	23,11	10,66	5,64	69,44	5,07	4,56	0,65	14,95	109	11
		Voie navigable	58 394	2 811	7,22	6,50	3,15	2,49	21,37	83,03	73,98	23,34	261,65	426	304
		Fer	124 014	82 969	10,65	10,08	3,01	4,66	22,35	53,21	52,73	48,22	62,95	1 122	44
8	Produits chimiques	Route	138 132	105 018	25,31	23,36	10,25	6,28	65,29	5,31	4,83	0,65	15,09	1 568	65
		Voie navigable	66 073	5 751	9,60	8,71	4,51	2,61	28,26	78,47	70,41	15,14	247,53	1 746	612
		Fer	138 132	50 075	26,78	25,95	4,79	17,08	44,29	53,44	52,99	48,22	62,88	622	29
9	Machines et articles manufacturés	Route	134 619	115 615	24,12	23,04	9,06	6,24	64,87	5,05	4,79	0,65	15,09	3 705	124
		Voie navigable	63 600	1 756	9,45	8,72	3,90	3,32	24,09	86,19	78,69	23,34	236,18	2 078	307
		Fer	134 619	36 710	37,93	37,37	5,08	26,97	59,18	53,20	52,94	48,22	62,85	1 606	52

Annexe D - Corrélations coût/durée

D.0 Explication

Pourquoi cette annexe ?

Dans le modèle NODUS, les coûts et les durées de transport sont tous deux calculés à partir des mêmes distances de réseau, selon des barèmes fixes par mode. Concrètement, cela signifie que plus un trajet est long, plus il est à la fois coûteux et long en temps, de façon quasi proportionnelle. Il en résulte une corrélation très élevée, voire parfaite, entre ces deux variables.

Ce phénomène a une conséquence directe sur l'estimation du modèle : si le coût et la durée évoluent toujours ensemble, il devient impossible de savoir lequel des deux influence vraiment le choix modal. Le modèle ne peut pas « démêler » les deux effets. C'est ce qu'on appelle un problème de multicolinéarité, décrit en section 1.6.1 du mémoire.

Comment lire les tableaux ?

Les tableaux ci-dessous indiquent, pour chaque groupe de marchandises et chaque mode, le coefficient de corrélation entre le coût monétaire et la durée de transport. Ce coefficient varie entre -1 et +1 :

r = 1,000 : corrélation parfaite → coût et durée sont proportionnels, il est impossible de séparer leurs effets dans le modèle

r > 0,950 : corrélation très élevée → risque important de multicolinéarité

r < 0,800 : corrélation modérée → les deux variables apportent des informations distinctes

Pourquoi r = 1 pour chaque mode pris individuellement ?

Pour un mode donné sur une paire OD, le coût et la durée sont calculés exactement à partir de la même distance de réseau. La corrélation est donc égale à 1 par construction, ce n'est pas un résultat empirique, c'est une propriété mécanique du modèle NODUS.

Pourquoi la corrélation « tous modes confondus » est-elle différente ?

Quand on mélange les trois modes ensemble, les niveaux absolus de coût et de durée diffèrent fortement d'un mode à l'autre. La route est rapide et relativement chère ; la voie navigable est lente et bon marché ; le fer se situe entre les deux. Ce mélange de profils très différents peut faire apparaître une corrélation négative ou faible entre coût et durée au

niveau global. C'est la corrélation intra-mode (r = 1) qui pose le problème d'identification, pas la corrélation globale.

NUTS 2 vs NUTS 3 : pourquoi le problème est plus sévère à fine granularité ?

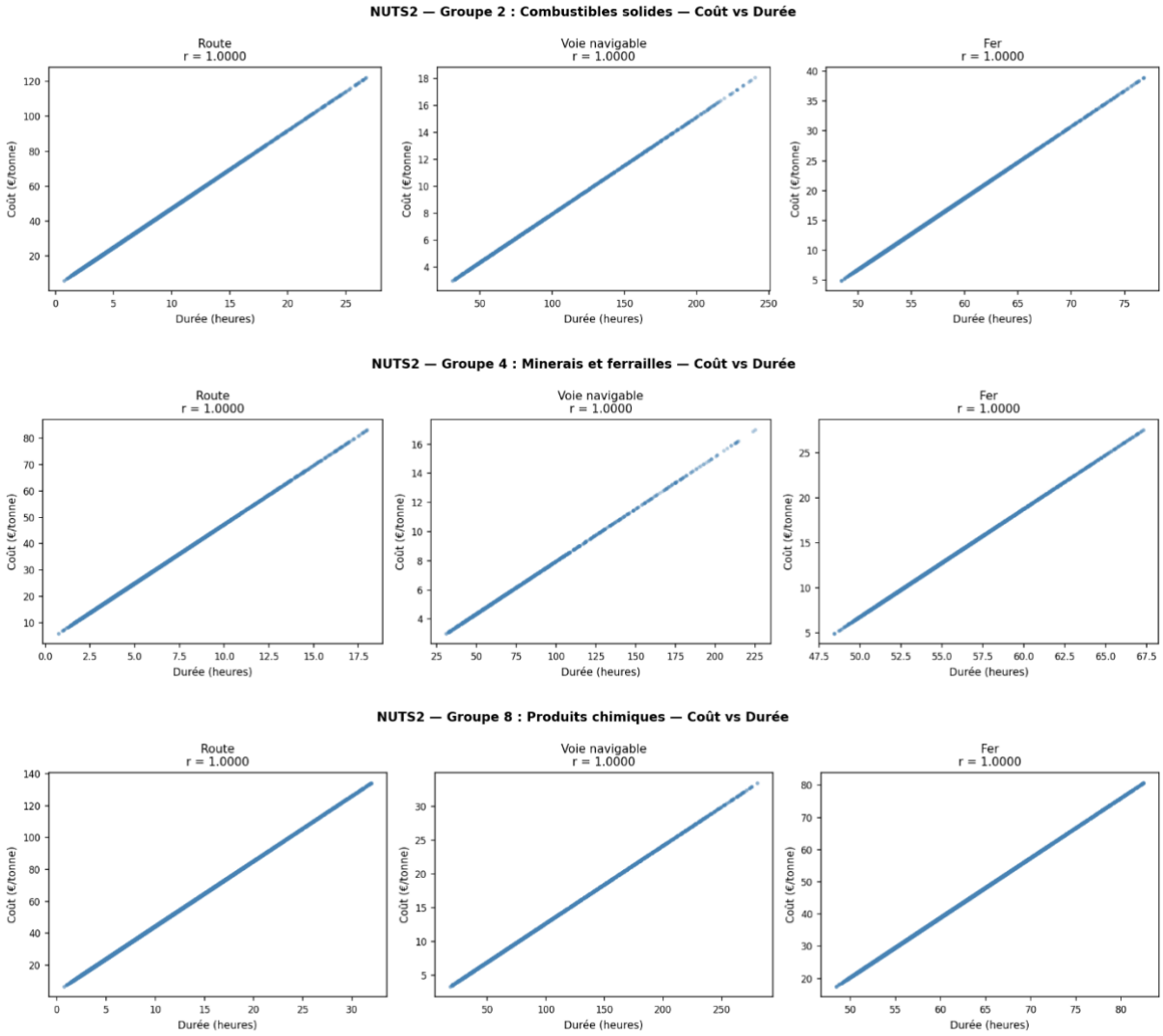
Au niveau NUTS 2, les zones géographiques sont grandes et regroupent des flux de distances très variées. Cette diversité de distances crée naturellement une certaine variance dans les coûts et les durées, ce qui laisse une légère marge pour distinguer les deux effets dans certains groupes.

Au niveau NUTS 3, les zones sont beaucoup plus petites et les distances entre paires OD sont plus homogènes. La variance est réduite, et les coûts et durées deviennent encore plus parfaitement corrélés. C'est pourquoi θ_time est impossible à identifier correctement pour 7 groupes sur 10 au niveau NUTS 3, alors que le problème est plus limité au niveau NUTS 2.

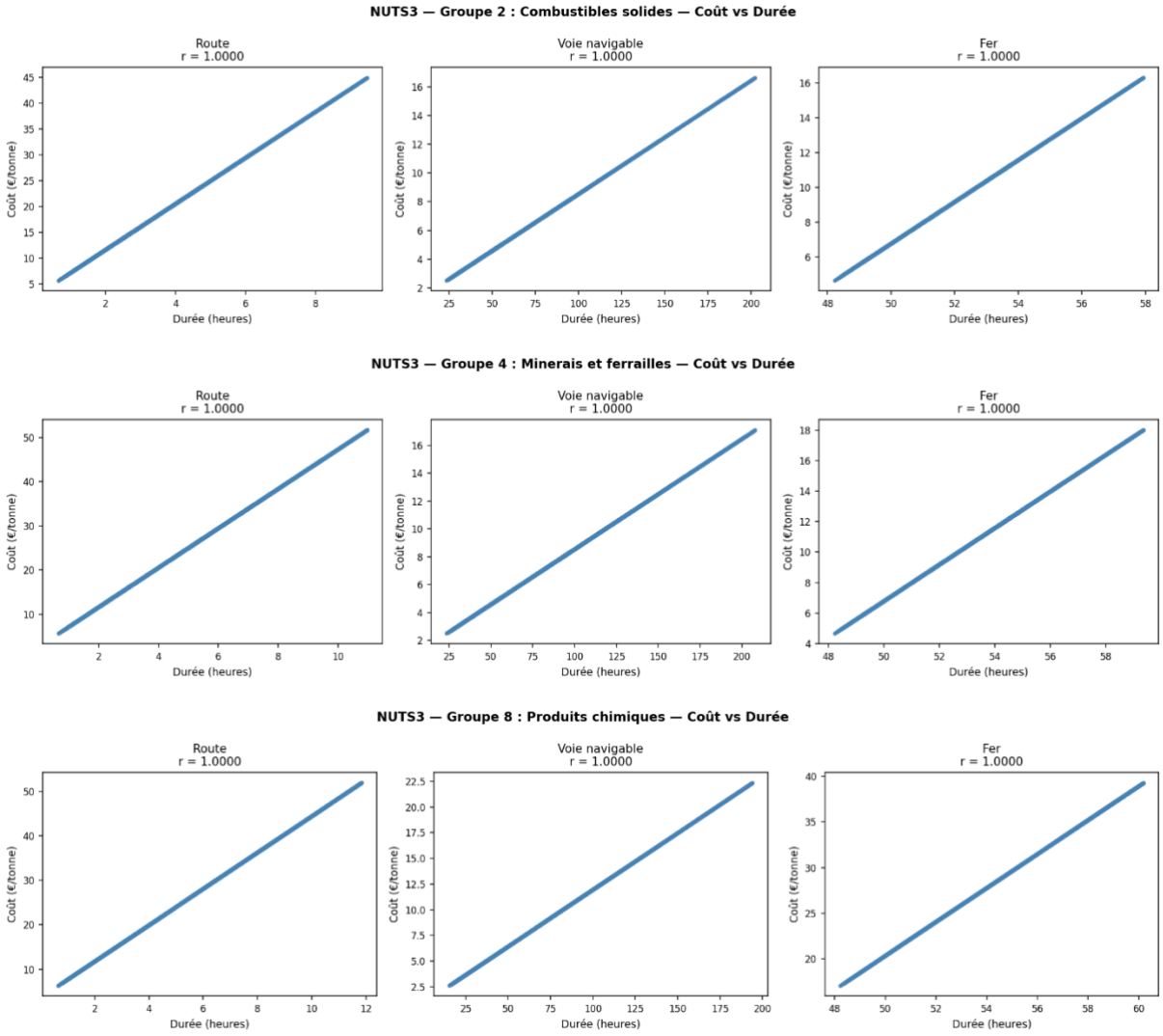
Ce que ça change pour les résultats

Ce problème justifie directement le choix du modèle à coûts généralisés (section 2.4) : plutôt que d'estimer séparément un coefficient pour le coût et un pour la durée, ce qui est impossible quand les deux sont parfaitement corrélés, on les combine en un seul indicateur composite (GC = coût + θ_time × durée). Cette approche contourne le problème de multicolinéarité tout en restant économiquement interprétable.

D.1 Nuage de point NUTS 2 (pour les groupes 2, 4, 8)



D.2 Nuage de point NUTS 3 (pour les groupes 2, 4, 8)



D.3 Comparaison NUTS 2 *vs* NUTS 3

Comparaison des plages de variation coût et durée (NUTS2 vs NUTS3)							
Durée de transport (heures)							
Mode	**NUTS2 Min**	**NUTS2 Max**	**NUTS2 Écart-type moy.**	**NUTS3 Min**	**NUTS3 Max**	**NUTS3 Écart-type moy.**	**Réduction variance**
Route	0,72	46,3	6,3	0,65	15,1	2,1	–67 %
Voie navigable	17,2	463,1	59,3	14,3	264	39,1	–34 %
Fer	48,4	96,5	6,5	48,2	62,9	2,1	–68 %
Coût de transport (€/tonne)							
Mode	**NUTS2 Min**	**NUTS2 Max**	**NUTS2 Écart-type moy.**	**NUTS3 Min**	**NUTS3 Max**	**NUTS3 Écart-type moy.**	**Réduction variance**
Route	5,96	199,6	26,3	5,64	69,4	9,6	–63 %
Voie navigable	3	53,1	5,2	2,49	28,3	3,5	–33 %
Fer	4,92	124,8	9,3	4,66	59,2	3,6	–61 %

Annexe E - Code Python

E.0 ComputeElasticities_GC.py

Langage : Python 3	**Lignes de code :** 468 lignes	**Perturbation :** +10 % (arc)

Code complet

```python
# CALCUL DES ÉLASTICITÉS DIRECTES ET CROISÉES
# Modèle à coûts généralisés (GC = cost + THETA_TIME * duration)

import sys
import os
import struct

import pandas as pd
import numpy as np
from openpyxl import load_workbook, Workbook
from openpyxl.styles import Font, PatternFill, Alignment, Border, Side
from openpyxl.utils import get_column_letter

# NUTS2
PARAMS_FILE_NUTS2 = 'resultats_NUTS2.xlsx'
DATA_FILE_NUTS2   = 'biogeme_input.dbf'     # .dbf ou .csv
OUTPUT_FILE_NUTS2 = 'elasticites_NUTS2.xlsx'

# NUTS3
PARAMS_FILE_NUTS3 = 'resultats_NUTS3.xlsx'
DATA_FILE_NUTS3   = 'biogeme_input_NUTS3.csv'    # .dbf ou .csv
OUTPUT_FILE_NUTS3 = 'elasticites_NUTS3.xlsx'

# Commun
PERTURBATION = 0.10    # +10 % pour l'élasticité arc

MODE_LABELS = {1: 'Route', 2: 'Voie navigable', 3: 'Fer'}

NST_LABELS = {
    0: 'Produits agricoles',
    1: 'Denrées alimentaires et fourrage',
    2: 'Combustibles solides',
```

```
    3: 'Produits pétroliers',
    4: 'Minerais et ferrailles',
    5: 'Produits métallurgiques',
    6: 'Minéraux et mat. de construction',
    7: 'Engrais',
    8: 'Produits chimiques',
    9: 'Machines et articles manufacturés',
}

def read_dbf(filepath):
    with open(filepath, 'rb') as f:
        header = f.read(32)
        num_records = struct.unpack('<I', header[4:8])[0]
        header_size = struct.unpack('<H', header[8:10])[0]
        record_size = struct.unpack('<H', header[10:12])[0]

        fields = []
        f.seek(32)
        while True:
            field_desc = f.read(32)
            if field_desc[0] == 0x0D:
                break
            name    = field_desc[:11].replace(b'\x00', b'').decode('latin1')
            ftype   = chr(field_desc[11])
            length  = field_desc[16]
            decimal = field_desc[17]
            fields.append((name, ftype, length, decimal))

        f.seek(header_size)
        records = []
        for _ in range(num_records):
            record = f.read(record_size)
            if not record or record[0] == 0x2A:
                continue
            row    = {}
            offset = 1
            for name, ftype, length, decimal in fields:
                val = record[offset:offset + length].decode('latin1').strip()
                if ftype == 'N':
                    try:
                        row[name] = float(val) if val else None
                    except Exception:
                        row[name] = None
```

```
                else:
                    row[name] = val
                offset += length
            records.append(row)
    return pd.DataFrame(records)


def load_data(filepath):
    ext = os.path.splitext(filepath)[1].lower()
    if ext == '.dbf':
        df = read_dbf(filepath)
    elif ext == '.csv':
        df = pd.read_csv(filepath)
    else:
        raise ValueError(f"Format non supporté : {filepath}. Utiliser .dbf ou .csv")
    df = df.apply(pd.to_numeric, errors='coerce')
    df.columns = [c.lower() for c in df.columns]
    return df

def prepare_dataframe(df_raw):
    """Conversions d'unités et calcul de la disponibilité des modes."""
    df = df_raw.copy()
    df = df.apply(pd.to_numeric, errors='coerce')
    df.columns = [c.lower() for c in df.columns]

    # Durées en heures (entrée en secondes)
    for i in [1, 2, 3]:
        if f'duration{i}' in df.columns:
            df[f'duration{i}'] = df[f'duration{i}'] / 3600.0

    # Disponibilité : mode disponible si coût non NaN
    for i in [1, 2, 3]:
        df[f'avail{i}'] = df[f'cost{i}'].notna().astype(int)

    df = df.fillna(1e10)
    return df

def compute_logit_probabilities(df, b_gc, theta_time, asc2, asc3):
    gc = {}
    for i in [1, 2, 3]:
        gc[i] = (df[f'cost{i}'] + theta_time * df[f'duration{i}']).clip(lower=1e-6)

    asc  = {1: 0.0, 2: asc2, 3: asc3}
```

```
    V    = {i: asc[i] + b_gc * np.log(gc[i]) for i in [1, 2, 3]}
    expV = {i: df[f'avail{i}'] * np.exp(V[i]) for i in [1, 2, 3]}
    denom = sum(expV[i] for i in [1, 2, 3]).replace(0, np.nan)
    P     = {i: expV[i] / denom for i in [1, 2, 3]}
    return gc, P

def elasticity_point(P, b_gc):
    E = {}
    for i in [1, 2, 3]:
        E[(i, i)] = b_gc * (1 - P[i])
        for j in [1, 2, 3]:
            if j != i:
                E[(i, j)] = -b_gc * P[j]
    return E


def elasticity_arc(df, b_gc, theta_time, asc2, asc3, perturbation=0.10):
    gc_base, P_base = compute_logit_probabilities(df, b_gc, theta_time, asc2, asc3)
    E_arc = {}
    for i_pert in [1, 2, 3]:
        df_pert = df.copy()
        df_pert[f'cost{i_pert}'] = df[f'cost{i_pert}'] * (1 + perturbation)
        gc_pert, P_pert = compute_logit_probabilities(df_pert, b_gc, theta_time, asc2, asc3)

        gc_pert_i    = gc_pert[i_pert]
        delta_GC_rel = (gc_pert_i - gc_base[i_pert]) / gc_base[i_pert].replace(0, np.nan)

        for j_meas in [1, 2, 3]:
            delta_P = P_pert[j_meas] - P_base[j_meas]
            with np.errstate(divide='ignore', invalid='ignore'):
                e_arc = np.where(
                    (P_base[j_meas] > 1e-9) & (delta_GC_rel.abs() > 1e-9),
                    (delta_P / P_base[j_meas]) / delta_GC_rel,
                    np.nan
                )
            E_arc[(i_pert, j_meas)] = pd.Series(e_arc, index=df.index)
    return E_arc

def weighted_mean(series, weights):
    mask = series.notna() & (weights > 0)
    if mask.sum() == 0:
        return np.nan
    return (series[mask] * weights[mask]).sum() / weights[mask].sum()
```

```
162
163 # Styles globaux
164 HDR_FILL   = PatternFill('solid', start_color='1F4E79')
165 HDR_FONT   = Font(bold=True, color='FFFFFF', name='Arial', size=10)
166 DATA_FONT  = Font(name='Arial', size=10)
167 TITLE_FONT = Font(bold=True, name='Arial', size=11)
168 THIN       = Side(style='thin', color='BFBFBF')
169 BORDER     = Border(left=THIN, right=THIN, top=THIN, bottom=THIN)
170 CENTER     = Alignment(horizontal='center', vertical='center', wrap_text=True)
171 ALT_FILL   = PatternFill('solid', start_color='DCE6F1')
172
173 # Couleurs pour la matrice (directe / croisée positive / croisée négative)
174 FILL_DIRECT   = PatternFill('solid', start_color='C6EFCE')   # vert clair
175 FILL_CROSS_POS = PatternFill('solid', start_color='FFEB9C')  # jaune
176 FILL_CROSS_NEG = PatternFill('solid', start_color='FFC7CE')  # rouge clair
177
178 def style_header_row(ws, row, max_col):
179     for col in range(1, max_col + 1):
180         cell = ws.cell(row=row, column=col)
181         cell.fill   = HDR_FILL
182         cell.font   = HDR_FONT
183         cell.border = BORDER
184         cell.alignment = CENTER
185
186 def style_data_row(ws, row, max_col, alt=False):
187     fill = ALT_FILL if alt else PatternFill(fill_type=None)
188     for col in range(1, max_col + 1):
189         cell = ws.cell(row=row, column=col)
190         cell.fill   = fill
191         cell.font   = DATA_FONT
192         cell.border = BORDER
193         cell.alignment = Alignment(horizontal='center', vertical='center')
194
195 def save_results_to_excel(output_file, rows_point, rows_arc, level_label):
196     wb = Workbook()
197
198     df_point = pd.DataFrame(rows_point)
199     df_arc   = pd.DataFrame(rows_arc)
200
201     # Feuille 1 : Données détaillées
202     ws1 = wb.active
203     ws1.title = 'Données détaillées'
204
```

```
    if not df_point.empty and not df_arc.empty:
        df_merged = df_point.merge(
            df_arc[['groupe', 'mode_perturbe', 'mode_observe',
                    'elasticite_arc', 'perturbation_pct']],
            on=['groupe', 'mode_perturbe', 'mode_observe'],
            how='outer'
        )
    elif not df_point.empty:
        df_merged = df_point.copy()
        df_merged['elasticite_arc'] = np.nan
        df_merged['perturbation_pct'] = int(PERTURBATION * 100)
    else:
        df_merged = pd.DataFrame()

    if not df_merged.empty:
        cols = list(df_merged.columns)
        ws1.append(cols)
        style_header_row(ws1, 1, len(cols))
        for r_idx, row_data in enumerate(df_merged.itertuples(index=False), start=2):
            ws1.append(list(row_data))
            style_data_row(ws1, r_idx, len(cols), alt=(r_idx % 2 == 0))
        for col_idx, col_name in enumerate(cols, start=1):
            ws1.column_dimensions[get_column_letter(col_idx)].width = max(14, len(col_name) + 2)

    # Feuille 2 : Matrice point (3×3 par groupe)
    ws2 = wb.create_sheet('Matrice point')
    _write_matrix_sheet(ws2, df_point, 'elasticite_point',
                        f'Élasticités POINT — {level_label}')

    # Feuille 3 : Matrice arc (3×3 par groupe)
    ws3 = wb.create_sheet('Matrice arc')
    _write_matrix_sheet(ws3, df_arc, 'elasticite_arc',
                        f'Élasticités ARC (+{int(PERTURBATION*100)}%) — {level_label}')

    wb.save(output_file)
    print(f"  -> Sauvegarde : {output_file}")


def _write_matrix_sheet(ws, df_elas, value_col, title):
    if df_elas.empty:
        ws['A1'] = 'Aucun résultat disponible.'
        return
    row_cursor = 1
```

```
    for g in sorted(df_elas['groupe'].unique()):
        label = NST_LABELS.get(int(g), f'Groupe {int(g)}')
        sub   = df_elas[df_elas['groupe'] == g]

        # Titre du groupe
        title_cell = ws.cell(row=row_cursor, column=1,
                             value=f'Groupe {int(g)} — {label}')
        title_cell.font      = TITLE_FONT
        title_cell.fill      = PatternFill('solid', start_color='2E75B6')
        title_cell.font      = Font(bold=True, color='FFFFFF', name='Arial', size=11)
        title_cell.alignment = CENTER
        ws.merge_cells(start_row=row_cursor, start_column=1,
                       end_row=row_cursor, end_column=4)
        row_cursor += 1

        # En-tête : ligne vide + colonnes modes observés
        ws.cell(row=row_cursor, column=1, value='Mode perturbé \\ Mode observé')
        for j, mlabel in MODE_LABELS.items():
            c = ws.cell(row=row_cursor, column=j + 1, value=mlabel)
        style_header_row(ws, row_cursor, 4)
        row_cursor += 1

        # Lignes : mode perturbé
        for i_pert in [1, 2, 3]:
            ws.cell(row=row_cursor, column=1, value=MODE_LABELS[i_pert]).font = Font(
                bold=True, name='Arial', size=10)
            ws.cell(row=row_cursor, column=1).border = BORDER
            ws.cell(row=row_cursor, column=1).fill = PatternFill('solid', start_color='D6E4F0')

            for j_meas in [1, 2, 3]:
                m = sub[(sub['mode_perturbe'] == MODE_LABELS[i_pert]) &
                        (sub['mode_observe']  == MODE_LABELS[j_meas])][value_col]
                val  = m.values[0] if len(m) > 0 else None
                cell = ws.cell(row=row_cursor, column=j_meas + 1)
                cell.value  = round(float(val), 4) if val is not None and not np.isnan(val) else '-'
                cell.font   = DATA_FONT
                cell.border = BORDER
                cell.alignment = CENTER
                cell.number_format = '0.0000'
                # Couleur selon type
                if i_pert == j_meas:
                    cell.fill = FILL_DIRECT
                elif val is not None and not np.isnan(val) and val > 0:
```

```
                    cell.fill = FILL_CROSS_POS
                else:
                    cell.fill = FILL_CROSS_NEG
            row_cursor += 1

        row_cursor += 1   # ligne vide entre groupes

    # Largeurs de colonnes
    ws.column_dimensions['A'].width = 32
    for col in ['B', 'C', 'D']:
        ws.column_dimensions[col].width = 18

    # Légende
    ws.cell(row=row_cursor + 1, column=1,
            value='Légende :').font = Font(bold=True, name='Arial', size=10)
    legend = [
        (FILL_DIRECT,    'Élasticité directe (Eii) — attendue < 0'),
        (FILL_CROSS_POS, 'Élasticité croisée positive — substitution entre modes'),
        (FILL_CROSS_NEG, 'Élasticité croisée négative'),
    ]
    for offset, (fill, text) in enumerate(legend, start=2):
        c = ws.cell(row=row_cursor + offset, column=1)
        c.fill  = fill
        c.value = text
        c.font  = DATA_FONT
        c.border = BORDER

def run(level: str):
    level = level.lower().strip()
    if level not in ('nuts2', 'nuts3'):
        print("Usage : python ComputeElasticities_GC.py nuts2")
        print("        python ComputeElasticities_GC.py nuts3")
        sys.exit(1)

    # Sélection des fichiers selon le niveau
    if level == 'nuts2':
        params_file  = PARAMS_FILE_NUTS2
        data_file    = DATA_FILE_NUTS2
        output_file  = OUTPUT_FILE_NUTS2
        level_label  = 'NUTS2 Europe'
    else:
        params_file  = PARAMS_FILE_NUTS3
        data_file    = DATA_FILE_NUTS3
```

```
        output_file  = OUTPUT_FILE_NUTS3
        level_label  = 'NUTS3 Belgique'


    print(f"  CALCUL DES ÉLASTICITÉS — {level_label.upper()}")

    # 1. Chargement des paramètres estimés
    df_params = pd.read_excel(params_file)
    df_params.columns = [c.lower() for c in df_params.columns]

    for col in ['groupe', 'b_gc', 'theta_time']:
        if col not in df_params.columns:
            raise ValueError(
                f"Colonne '{col}' absente de {params_file}. "
                f"Colonnes disponibles : {list(df_params.columns)}"
            )
    print(f"  -> {len(df_params)} groupes chargés : "
          f"{sorted(df_params['groupe'].dropna().astype(int).tolist())}")

    # 2. Chargement des données OD
    print(f"\nChargement des données depuis {data_file} ...")
    df_all = load_data(data_file)
    print(f"  -> {len(df_all):,} observations chargées")

    # 3. Calcul groupe par groupe
    rows_point = []
    rows_arc   = []

    for g_idx, param_row in df_params.iterrows():
        g          = int(param_row['groupe'])
        b_gc       = float(param_row['b_gc'])
        theta_time = float(param_row['theta_time'])
        asc2       = float(param_row.get('intercept.2', 0.0))
        asc3       = float(param_row.get('intercept.3', 0.0))

        label = NST_LABELS.get(g, f'Groupe {g}')
        print(f"\n{'—'*70}")
        print(f"  Groupe {g} — {label}  "
              f"(β_GC={b_gc:.4f}, θ={theta_time:.4f}, "
              f"ASC2={asc2:.4f}, ASC3={asc3:.4f})")

        # Filtrage et préparation
        df_g = df_all[df_all['grp'] == g].copy()
```

```
        df_g = prepare_dataframe(df_g)

        qty_cols = [c for c in ['qty1', 'qty2', 'qty3'] if c in df_g.columns]
        if not qty_cols:
            print(f"  WARNING : colonnes qty absentes pour le groupe {g}.")
            continue

        df_g = df_g[(df_g['avail1'] + df_g['avail2'] + df_g['avail3']) > 0]
        df_g = df_g[df_g[qty_cols].sum(axis=1) > 0].copy()

        if len(df_g) < 5:
            print(f"  Trop peu d'observations ({len(df_g)}), groupe ignoré.")
            continue
        print(f"  {len(df_g):,} observations valides")

        weights = df_g[qty_cols].sum(axis=1)

        # a) Probabilités de base
        gc, P = compute_logit_probabilities(df_g, b_gc, theta_time, asc2, asc3)

        # b) Élasticités POINT
        E_pt = elasticity_point(P, b_gc)
        print(f"\n  Élasticités POINT (moyennes pondérées) :")
        print(f"  {'Perturbé':<18} {'Observé':<18} {'Élasticité':>12} {'Type':<10}")
        print(f"  {'-'*60}")
        for i_pert in [1, 2, 3]:
            for j_meas in [1, 2, 3]:
                val   = weighted_mean(E_pt[(i_pert, j_meas)], weights)
                etype = 'Directe' if i_pert == j_meas else 'Croisée'
                print(f"  {MODE_LABELS[i_pert]:<18} {MODE_LABELS[j_meas]:<18} "
                      f"{val:>12.4f} {etype:<10}")
                rows_point.append({
                    'groupe'           : g,
                    'marchandise'      : label,
                    'mode_perturbe'    : MODE_LABELS[i_pert],
                    'mode_observe'     : MODE_LABELS[j_meas],
                    'type'             : etype,
                    'elasticite_point': round(val, 6) if not np.isnan(val) else None,
                    'b_gc'             : b_gc,
                    'theta_time'       : theta_time,
                    'n_obs'            : len(df_g),
                })

```

```
        # c) Élasticités ARC
        E_arc = elasticity_arc(df_g, b_gc, theta_time, asc2, asc3,
                               perturbation=PERTURBATION)
        print(f"\n  Élasticités ARC (+{int(PERTURBATION*100)}%, moyennes pondérées) :")
        print(f"  {'Perturbé':<18} {'Observé':<18} {'Élasticité':>12} {'Type':<10}")
        print(f"  {'-'*60}")
        for i_pert in [1, 2, 3]:
            for j_meas in [1, 2, 3]:
                val   = weighted_mean(E_arc[(i_pert, j_meas)], weights)
                etype = 'Directe' if i_pert == j_meas else 'Croisée'
                print(f"  {MODE_LABELS[i_pert]:<18} {MODE_LABELS[j_meas]:<18} "
                      f"{val:>12.4f} {etype:<10}")
                rows_arc.append({
                    'groupe'          : g,
                    'marchandise'     : label,
                    'mode_perturbe'   : MODE_LABELS[i_pert],
                    'mode_observe'    : MODE_LABELS[j_meas],
                    'type'            : etype,
                    'elasticite_arc'  : round(val, 6) if not np.isnan(val) else None,
                    'perturbation_pct': int(PERTURBATION * 100),
                    'b_gc'            : b_gc,
                    'theta_time'      : theta_time,
                    'n_obs'           : len(df_g),
                })

        # SAUVEGARDE APRÈS CHAQUE GROUPE
        print(f"\n  Sauvegarde intermédiaire ({g_idx + 1}/{len(df_params)} groupes traités)...")
        save_results_to_excel(output_file, rows_point, rows_arc, level_label)

    # 4. Sauvegarde finale
    print("  Sauvegarde finale ...")
    save_results_to_excel(output_file, rows_point, rows_arc, level_label)
    print(f"  -> Fichier final : {output_file}")

if __name__ == '__main__':
    if len(sys.argv) < 2:
        print("Usage :")
        print("  python ComputeElasticities_GC.py nuts2")
        print("  python ComputeElasticities_GC.py nuts3")
        sys.exit(1)
    run(sys.argv[1])
```

E.1 SolveGeneralizedCost_NUTS2.py

Langage : Python 3	**Lignes de code :** 338 lignes	**Perturbation :** +10 % (arc)

Code complet

```python
import pandas as pd
import numpy as np
import biogeme.database as db
import biogeme.biogeme as bio
from biogeme.expressions import Beta, log, exp, bioMax
import os
import time
import struct

# CONFIGURATION
INPUT_FILE  = 'biogeme_input.dbf'
OUTPUT_XLSX = 'resultats_NUTS2.xlsx'


def read_dbf(filepath):
    with open(filepath, 'rb') as f:
        header = f.read(32)
        num_records = struct.unpack('<I', header[4:8])[0]
        header_size = struct.unpack('<H', header[8:10])[0]
        record_size = struct.unpack('<H', header[10:12])[0]

        fields = []
        f.seek(32)
        while True:
            field_desc = f.read(32)
            if field_desc[0] == 0x0D:
                break
            name = field_desc[:11].replace(b'\x00', b'').decode('latin1')
            ftype = chr(field_desc[11])
            length = field_desc[16]
            decimal = field_desc[17]
```

```
            fields.append((name, ftype, length, decimal))

        f.seek(header_size)
        records = []
        for _ in range(num_records):
            record = f.read(record_size)
            if not record or record[0] == 0x2A:
                continue
            row = {}
            offset = 1
            for name, ftype, length, decimal in fields:
                val = record[offset:offset + length].decode('latin1').strip()
                if ftype == 'N':
                    try:
                        row[name] = float(val) if val else None
                    except Exception:
                        row[name] = None
                else:
                    row[name] = val
                offset += length
            records.append(row)
    return pd.DataFrame(records)


def prepare_dataframe(df_raw):
    df = df_raw.copy()
    df = df.apply(pd.to_numeric, errors='coerce')
    df.columns = map(str.lower, df.columns)

    # Durées
    df['duration1'] = df['duration1'] / 3600.0
    df['duration2'] = df['duration2'] / 3600.0
    df['duration3'] = df['duration3'] / 3600.0

    # Longueurs
    df['length1'] = df['length1'] * 0.001
    df['length2'] = df['length2'] * 0.001
    df['length3'] = df['length3'] * 0.001

    # Disponibilité des modes — 1 si coût non NaN, 0 sinon
    for i in [1, 2, 3]:
        df[f'avail{i}'] = df[f'cost{i}'].notna().astype(int)

```

```
    # Remplir les NaN par une grande valeur pour éviter erreurs Biogeme
    df = df.fillna(1e10)

    df['choice'] = df[['qty1', 'qty2', 'qty3']].idxmax(axis=1).str[-1].astype(int)
    # Si toutes les qty sont 0, choix = 1 par défaut
    mask_zero = (df['qty1'] == 0) & (df['qty2'] == 0) & (df['qty3'] == 0)
    df.loc[mask_zero, 'choice'] = 1

    return df


def _save_results(all_results, all_gof, output_path):

    if not all_results:
        return

    df_res = pd.concat(all_results, ignore_index=True)

    for col in ['Value', 'Rob. Std err', 'Rob. t-test', 'Rob. p-value']:
        if col in df_res.columns:
            df_res[col] = pd.to_numeric(df_res[col], errors='coerce').round(6)

    # Intervalles de confiance à 95%
    if 'Value' in df_res.columns and 'Rob. Std err' in df_res.columns:
        df_res['IC_inf_95'] = (df_res['Value'] - 1.96 * df_res['Rob. Std err']).round(6)
        df_res['IC_sup_95'] = (df_res['Value'] + 1.96 * df_res['Rob. Std err']).round(6)

    # Significativité (étoiles)
    def significance_stars(pval):
        if pd.isna(pval):
            return '-'
        if pval < 0.01:
            return '***'
        if pval < 0.05:
            return '**'
        if pval < 0.10:
            return '*'
        return 'ns'

    if 'Rob. p-value' in df_res.columns:
        df_res['Significativité'] = df_res['Rob. p-value'].apply(significance_stars)

    # Signe attendu / statut
```

```
    def expected_sign(param):
        if param == 'b_gc':
            return '< 0'
        if param == 'theta_time':
            return '>= 0'
        return 'libre'

    def sign_ok(row):
        exp_s = expected_sign(row['Parameter'])
        val = row.get('Value', None)
        if not isinstance(val, (int, float)) or pd.isna(val):
            return '-'
        if exp_s == '< 0':
            return 'OK' if val < 0 else ('Nul' if val == 0 else 'Mauvais signe')
        if exp_s == '>= 0':
            return 'OK' if val >= 0 else 'Mauvais signe'
        return '-'

    df_res['Signe attendu'] = df_res['Parameter'].apply(expected_sign)
    df_res['Statut signe']  = df_res.apply(sign_ok, axis=1)

    # Qualité d'ajustement
    df_gof = pd.DataFrame(all_gof) if all_gof else pd.DataFrame()

    with pd.ExcelWriter(output_path, engine='openpyxl') as writer:

        # Onglet principal : toutes les colonnes au format long
        df_res.to_excel(writer, sheet_name='Tous les paramètres', index=False)

        # Onglets pivotés : une colonne = un groupe
        for col, sheet in [
            ('Value',           'Valeurs par groupe'),
            ('Rob. t-test',     'T-stats par groupe'),
            ('Rob. p-value',    'P-values par groupe'),
            ('Rob. Std err',    'Std err par groupe'),
            ('IC_inf_95',       'IC inf 95% par groupe'),
            ('IC_sup_95',       'IC sup 95% par groupe'),
            ('Significativité', 'Significativité par groupe'),
            ('Statut signe',    'Statut des signes'),
        ]:
            if col in df_res.columns:
                pivot = df_res.pivot(index='Parameter', columns='Groupe', values=col)
                pivot.to_excel(writer, sheet_name=sheet)
```

```
161
162         # Onglet qualité d'ajustement
163         if not df_gof.empty:
164             df_gof.to_excel(writer, sheet_name="Qualité d'ajustement", index=False)
165
166     groupes_faits = df_res['Groupe'].unique().tolist()
167     print(f"  -> Sauvegarde : {output_path} (groupes : {groupes_faits})")
168
169
170 def run():
171     print("  MODÈLE COÛTS GÉNÉRALISÉS — NUTS2 Europe")
172
173     # Dossier de sortie
174     wd = os.path.join(os.getcwd(), 'models_nuts2')
175     os.makedirs(wd, exist_ok=True)
176
177     df_all = read_dbf(INPUT_FILE)
178     df_all = df_all.apply(pd.to_numeric, errors='coerce')
179     df_all.columns = map(str.lower, df_all.columns)
180
181     all_results = []
182     all_gof     = []   # qualité d'ajustement par groupe
183
184     for g in sorted(df_all['grp'].dropna().unique()):
185         g = int(g)
186
187         print(f"  GROUPE {g}")
188
189         df_group = df_all[df_all['grp'] == g].copy()
190         if df_group.empty:
191             print(f"  Groupe {g} vide, ignoré.")
192             continue
193
194         df = prepare_dataframe(df_group)
195
196         # Garder uniquement les OD avec au moins un mode dispo et qty > 0
197         df = df[(df['avail1'] + df['avail2'] + df['avail3']) > 0]
198         df = df[(df['qty1'] + df['qty2'] + df['qty3']) > 0].copy()
199         print(f"  -> {len(df):,} observations après filtrage")
200
201         if len(df) < 10:
202             print(f"  Trop peu d'observations, groupe ignoré.")
203             continue
```

```
204
205         # Normalisation des poids
206         total_qty = df['qty1'].sum() + df['qty2'].sum() + df['qty3'].sum()
207         n = len(df)
208
209         database = db.Database(f'nuts2_grp{g}', df)
210         globals().update(database.variables)
211
212         # Poids relatifs à la taille de l'échantillon
213         df['weight'] = n * (df['qty1'] + df['qty2'] + df['qty3']) / total_qty
214         database = db.Database(f'nuts2_grp{g}', df)
215         globals().update(database.variables)
216
217         # PARAMÈTRES
218         # Intercepts (mode 1 = référence, forcé à 0)
219         INTERCEPT1 = Beta('intercept.1', 0,    None, None, 1)
220         INTERCEPT2 = Beta('intercept.2', 0,    None, None, 0)
221         INTERCEPT3 = Beta('intercept.3', 0,    None, None, 0)
222
223         # Coût généralisé — doit être négatif
224         B_GC = Beta('b_gc', -0.5, None, 0, 0)
225
226         # Valeur du temps — doit être positif
227         THETA_TIME = Beta('theta_time', 0.5, 0, None, 0)
228
229         # COÛTS GÉNÉRALISÉS
230         GC1 = cost1 + THETA_TIME * duration1
231         GC2 = cost2 + THETA_TIME * duration2
232         GC3 = cost3 + THETA_TIME * duration3
233
234         # FONCTIONS D'UTILITÉ
235         V1 = INTERCEPT1 + B_GC * log(bioMax(GC1, 1e-6))
236         V2 = INTERCEPT2 + B_GC * log(bioMax(GC2, 1e-6))
237         V3 = INTERCEPT3 + B_GC * log(bioMax(GC3, 1e-6))
238
239         # LOG-VRAISEMBLANCE MNL WITH COUNTS
240         from biogeme.expressions import Elem
241
242         av = {1: avail1, 2: avail2, 3: avail3}
243         V  = {1: V1,     2: V2,     3: V3}
244
245         # Dénominateur
246         denom = (avail1 * exp(V1 - V1)   # = avail1 * 1
```

```
                    + avail2 * exp(V2 - V1)
                    + avail3 * exp(V3 - V1))

        # log(P_i) = V_i - V1 - log(denom)  [avec stabilisation par V1]
        logP1 = V1 - V1 - log(denom)    # = -log(denom)
        logP2 = V2 - V1 - log(denom)
        logP3 = V3 - V1 - log(denom)

        # MNL with counts
        logprob = qty1 * logP1 + qty2 * logP2 + qty3 * logP3

        formulas = {'loglike': logprob, 'weight': weight}

        biogeme_model = bio.BIOGEME(database, formulas)
        biogeme_model.modelName = f'nuts2_gc_grp{g}'

        # Supprimer les fichiers existants
        for ext in ['.html', '.pickle', '-1.html']:
            path = os.path.join(wd, biogeme_model.modelName + ext)
            if os.path.exists(path):
                os.remove(path)

        print(f"  Estimation en cours...")
        start = time.time()

        try:
            results = biogeme_model.estimate()
            params  = results.get_estimated_parameters()
            elapsed = time.time() - start

            print(f"  -> Terminé en {elapsed:.1f}s")
            print(params.to_string())

            # Vérification des signes
            print(f"\n  Vérification des signes :")
            p = params['Value']
            if 'b_gc' in p.index:
                status = "OK" if p['b_gc'] < 0 else "WARNING"
                print(f"  {status}: b_gc = {p['b_gc']:.4f} (attendu < 0)")
            if 'theta_time' in p.index:
                status = "OK" if p['theta_time'] >= 0 else "WARNING"
                print(f"  {status}: theta_time = {p['theta_time']:.4f} (attendu >= 0)")

```

```
            # Collecte paramètres au format LONG
            # params contient : Value, Rob. Std err, Rob. t-test, Rob. p-value
            params_long = params.reset_index().rename(columns={'index': 'Parameter'})
            params_long.insert(0, 'Groupe',  g)
            params_long.insert(1, 'Temps_s', round(elapsed, 1))
            params_long.insert(2, 'N_obs',   len(df))
            all_results.append(params_long)

            # Collecte qualité d'ajustement (Biogeme 3.2.11)
            try:
                gof_row = {'Groupe': g, 'N_obs': len(df), 'Temps_s': round(elapsed, 1)}
                d = results.data
                ll_final = d.logLike
                ll_null  = d.nullLogLike
                n_param  = d.nparam
                rho      = 1 - (ll_final / ll_null) if ll_null else None
                rho_bar  = 1 - ((ll_final - n_param) / ll_null) if ll_null else None
                gof_row['Final log likelihood'] = ll_final
                gof_row['Null log likelihood']  = ll_null
                gof_row['Number of parameters'] = n_param
                gof_row['Rho-square']           = round(rho, 6) if rho is not None else None
                gof_row['Rho-square-bar']       = round(rho_bar, 6) if rho_bar is not None else None
                all_gof.append(gof_row)
                print(f"  LL finale : {ll_final:.1f}   |   LL nulle : {ll_null:.1f}   |   Rho² : {rho:.4f}   |   Rho²-bar :
{rho_bar:.4f}")
            except Exception as e_gof:
                print(f"  (Qualité d'ajustement non disponible : {e_gof})")

            # Sauvegarde intermédiaire après chaque groupe
            _save_results(all_results, all_gof, OUTPUT_XLSX)

        except Exception as e:
            print(f"  ERREUR groupe {g} : {e}")
            all_results.append(pd.DataFrame([{
                'Groupe': g, 'Temps_s': round(time.time() - start, 1),
                'N_obs': len(df), 'Parameter': 'ERREUR', 'Value': str(e)
            }]))
            # Sauvegarde intermédiaire même en cas d'erreur
            _save_results(all_results, all_gof, OUTPUT_XLSX)

    # EXPORT EXCEL FINAL
    if all_results:
        _save_results(all_results, all_gof, OUTPUT_XLSX)
```

```
332          print(f"\n-> Export final : {OUTPUT_XLSX}")
333
334      print("\nDone.")
335
336
337  if __name__ == '__main__':
338      run()
```

E.2 SolveGeneralizedCost_NUTS3.py

Langage : Python 3	**Lignes de code :** 311 lignes	**Perturbation :** +10 % (arc)

Listing complet

```python
import pandas as pd
import numpy as np
import biogeme.database as db
import biogeme.biogeme as bio
from biogeme.expressions import Beta, log, exp, bioMax
import os
import time
from tqdm.auto import tqdm


# CONFIGURATION


INPUT_FILE  = 'biogeme_del3.csv'
OUTPUT_XLSX = 'resultats_NUTS3_GC.xlsx'
SEP         = ','

GROUPS_TO_RUN = list(range(10))

# FILTRES OUTLIERS PAR GROUPE
OUTLIER_FILTERS = {
    0: 'qty2to < 200000',
    2: 'qty2from < 500000 and qty2to < 500000',
    4: 'qty2 < 1200000 and qty2to < 300000',
    7: 'qty2to < 50000',
}


def prepare_dataframe(df_raw):
    df = df_raw.copy()
    df = df.apply(pd.to_numeric, errors='coerce')
```

```
    df.columns = map(str.lower, df.columns)

    # Durées en heures
    df['duration1'] = df['duration1'] / 3600.0
    df['duration2'] = df['duration2'] / 3600.0
    df['duration3'] = df['duration3'] / 3600.0

    # Longueurs en 1000 km
    df['length1'] = df['length1'] * 0.001
    df['length2'] = df['length2'] * 0.001
    df['length3'] = df['length3'] * 0.001

    # Disponibilité des modes — 1 si coût non NaN, 0 sinon
    for i in [1, 2, 3]:
        df[f'avail{i}'] = df[f'cost{i}'].notna().astype(int)

    # Remplir les NaN par une grande valeur
    df = df.fillna(1e10)

    # Choix = mode avec la plus grande quantité
    df['choice'] = df[['qty1', 'qty2', 'qty3']].idxmax(axis=1).str[-1].astype(int)
    mask_zero = (df['qty1'] == 0) & (df['qty2'] == 0) & (df['qty3'] == 0)
    df.loc[mask_zero, 'choice'] = 1

    return df


def _save_results(all_results, all_gof, output_path):
    if not all_results:
        return
    df_res = pd.concat(all_results, ignore_index=True)

    for col in ['Value', 'Rob. Std err', 'Rob. t-test', 'Rob. p-value']:
        if col in df_res.columns:
            df_res[col] = pd.to_numeric(df_res[col], errors='coerce').round(6)

    # Intervalles de confiance à 95%
    if 'Value' in df_res.columns and 'Rob. Std err' in df_res.columns:
        df_res['IC_inf_95'] = (df_res['Value'] - 1.96 * df_res['Rob. Std err']).round(6)
        df_res['IC_sup_95'] = (df_res['Value'] + 1.96 * df_res['Rob. Std err']).round(6)

    # Significativité
    def significance_stars(pval):
```

```
        if pd.isna(pval):
            return '-'
        if pval < 0.01:
            return '***'
        if pval < 0.05:
            return '**'
        if pval < 0.10:
            return '*'
        return 'ns'

    if 'Rob. p-value' in df_res.columns:
        df_res['Significativité'] = df_res['Rob. p-value'].apply(significance_stars)

    # Signe attendu / statut
    def expected_sign(param):
        if param == 'b_gc':
            return '< 0'
        if param == 'theta_time':
            return '>= 0'
        return 'libre'

    def sign_ok(row):
        exp_s = expected_sign(row['Parameter'])
        val = row.get('Value', None)
        if not isinstance(val, (int, float)) or pd.isna(val):
            return '-'
        if exp_s == '< 0':
            return 'OK' if val < 0 else ('Nul' if val == 0 else 'Mauvais signe')
        if exp_s == '>= 0':
            return 'OK' if val >= 0 else 'Mauvais signe'
        return '-'

    df_res['Signe attendu'] = df_res['Parameter'].apply(expected_sign)
    df_res['Statut signe']  = df_res.apply(sign_ok, axis=1)

    # Qualité d'ajustement
    df_gof = pd.DataFrame(all_gof) if all_gof else pd.DataFrame()

    with pd.ExcelWriter(output_path, engine='openpyxl') as writer:
        df_res.to_excel(writer, sheet_name='Tous les paramètres', index=False)

        for col, sheet in [
            ('Value',           'Valeurs par groupe'),
```

```
            ('Rob. t-test',     'T-stats par groupe'),
            ('Rob. p-value',    'P-values par groupe'),
            ('Rob. Std err',    'Std err par groupe'),
            ('IC_inf_95',       'IC inf 95% par groupe'),
            ('IC_sup_95',       'IC sup 95% par groupe'),
            ('Significativité', 'Significativité par groupe'),
            ('Statut signe',    'Statut des signes'),
        ]:
            if col in df_res.columns:
                pivot = df_res.pivot(index='Parameter', columns='Groupe', values=col)
                pivot.to_excel(writer, sheet_name=sheet)

        if not df_gof.empty:
            df_gof.to_excel(writer, sheet_name="Qualité d'ajustement", index=False)

    groupes_faits = df_res['Groupe'].unique().tolist()
    print(f"  -> Sauvegarde : {output_path} (groupes : {groupes_faits})")


def run():
    print("  MODÈLE COÛTS GÉNÉRALISÉS — NUTS3 Allemagne")

    # Dossier de sortie
    wd = os.path.join(os.getcwd(), 'models_nuts3')
    os.makedirs(wd, exist_ok=True)

    # Chargement des données
    df_all = pd.read_csv(INPUT_FILE, sep=SEP)
    df_all = df_all.apply(pd.to_numeric, errors='coerce')
    df_all.columns = map(str.lower, df_all.columns)
    df_all = df_all[df_all['qty'] > 0].copy()

    all_results = []
    all_gof     = []   # qualité d'ajustement par groupe

    for g in tqdm(GROUPS_TO_RUN, desc='Groupes', unit='grp'):

        print(f"  GROUPE {g}")

        df_group = df_all[df_all['grp'] == g].copy()
        if df_group.empty:
            print(f"  Groupe {g} vide, ignoré.")
            continue
```

```
161
162         # Appliquer filtre outliers si défini
163         if g in OUTLIER_FILTERS:
164             filtre = OUTLIER_FILTERS[g]
165             avant = len(df_group)
166             df_group = df_group.query(filtre).copy()
167             print(f"  -> Filtre outliers appliqué : {filtre}")
168             print(f"  -> {avant:,} → {len(df_group):,} observations")
169
170         df = prepare_dataframe(df_group)
171
172         # Garder OD avec au moins un mode dispo et qty > 0
173         df = df[(df['avail1'] + df['avail2'] + df['avail3']) > 0]
174         df = df[(df['qty1'] + df['qty2'] + df['qty3']) > 0].copy()
175         print(f"  -> {len(df):,} observations après filtrage")
176
177         if len(df) < 10:
178             print(f"  Trop peu d'observations, groupe ignoré.")
179             continue
180
181         # Normalisation des poids
182         total_qty = df['qty1'].sum() + df['qty2'].sum() + df['qty3'].sum()
183         n = len(df)
184         df['weight'] = n * (df['qty1'] + df['qty2'] + df['qty3']) / total_qty
185
186         database = db.Database(f'nuts3_grp{g}', df)
187         globals().update(database.variables)
188
189         # PARAMÈTRES
190         INTERCEPT1 = Beta('intercept.1', 0,    None, None, 1)
191         INTERCEPT2 = Beta('intercept.2', 0,    None, None, 0)
192         INTERCEPT3 = Beta('intercept.3', 0,    None, None, 0)
193
194         B_GC       = Beta('b_gc',        -0.5, None, 0,    0)
195         THETA_TIME = Beta('theta_time',   0.5, 0,    None, 0)
196
197         # COÛTS GÉNÉRALISÉS
198         GC1 = cost1 + THETA_TIME * duration1
199         GC2 = cost2 + THETA_TIME * duration2
200         GC3 = cost3 + THETA_TIME * duration3
201
202         # FONCTIONS D'UTILITÉ
203         V1 = INTERCEPT1 + B_GC * log(bioMax(GC1, 1e-6))
```

```
        V2 = INTERCEPT2 + B_GC * log(bioMax(GC2, 1e-6))
        V3 = INTERCEPT3 + B_GC * log(bioMax(GC3, 1e-6))

        # LOG-VRAISEMBLANCE MNL WITH COUNTS
        denom = (avail1 * exp(V1 - V1)
                 + avail2 * exp(V2 - V1)
                 + avail3 * exp(V3 - V1))

        logP1 = V1 - V1 - log(denom)
        logP2 = V2 - V1 - log(denom)
        logP3 = V3 - V1 - log(denom)

        logprob = qty1 * logP1 + qty2 * logP2 + qty3 * logP3

        formulas = {'loglike': logprob, 'weight': weight}

        biogeme_model = bio.BIOGEME(database, formulas)
        biogeme_model.modelName = f'nuts3_gc_grp{g}'

        # Supprimer fichiers existants
        for ext in ['.html', '.pickle', '-1.html']:
            path = os.path.join(wd, biogeme_model.modelName + ext)
            if os.path.exists(path):
                os.remove(path)

        print(f"  Estimation en cours...")
        start = time.time()

        try:
            results = biogeme_model.estimate()
            elapsed = time.time() - start

            # Compatibilité versions Biogeme
            try:
                params = results.get_estimated_parameters()
            except AttributeError:
                params = results.getEstimatedParameters()

            print(f"  -> Terminé en {elapsed:.1f}s")
            print(params.to_string())

            # Vérification des signes
            p = params['Value']
```

```
            print(f"\n  Vérification des signes :")
            if 'b_gc' in p.index:
                status = "OK" if p['b_gc'] < 0 else "WARNING"
                print(f"  {status}: b_gc = {p['b_gc']:.4f} (attendu < 0)")
            if 'theta_time' in p.index:
                status = "OK" if p['theta_time'] >= 0 else "WARNING"
                print(f"  {status}: theta_time = {p['theta_time']:.4f} (attendu >= 0)")

            params_long = params.reset_index().rename(columns={'index': 'Parameter'})
            params_long.insert(0, 'Groupe',  g)
            params_long.insert(1, 'Temps_s', round(elapsed, 1))
            params_long.insert(2, 'N_obs',   len(df))
            all_results.append(params_long)

            # Collecte qualité d'ajustement (Biogeme 3.2.11)
            try:
                gof_row = {'Groupe': g, 'N_obs': len(df), 'Temps_s': round(elapsed, 1)}
                d = results.data
                ll_final = d.logLike
                ll_null  = d.nullLogLike
                n_param  = d.nparam
                rho      = 1 - (ll_final / ll_null) if ll_null else None
                rho_bar  = 1 - ((ll_final - n_param) / ll_null) if ll_null else None
                gof_row['Final log likelihood'] = ll_final
                gof_row['Null log likelihood']  = ll_null
                gof_row['Number of parameters'] = n_param
                gof_row['Rho-square']           = round(rho, 6) if rho is not None else None
                gof_row['Rho-square-bar']       = round(rho_bar, 6) if rho_bar is not None else None
                all_gof.append(gof_row)
                print(f"  LL finale : {ll_final:.1f}   |   LL nulle : {ll_null:.1f}   |   Rho² : {rho:.4f}   |   Rho²-bar :
{rho_bar:.4f}")
            except Exception as e_gof:
                print(f"  (Qualité d'ajustement non disponible : {e_gof})")

        except Exception as e:
            print(f"  ERREUR groupe {g} : {e}")
            # Ligne d'erreur pour ne pas perdre la trace
            all_results.append(pd.DataFrame([{
                'Groupe': g, 'Temps_s': round(time.time() - start, 1),
                'N_obs': len(df), 'Parameter': 'ERREUR', 'Value': str(e)
            }]))

        # SAUVEGARDE INTERMÉDIAIRE après chaque groupe
```

```
289         if all_results:
290             if GROUPS_TO_RUN != list(range(10)):
291                 suffix = f"_grp{'_'.join(map(str, GROUPS_TO_RUN))}"
292                 inter_path = OUTPUT_XLSX.replace('.xlsx', f'{suffix}_intermediaire.xlsx')
293             else:
294                 inter_path = OUTPUT_XLSX.replace('.xlsx', '_intermediaire.xlsx')
295             _save_results(all_results, all_gof, inter_path)
296 
297     # EXPORT EXCEL FINAL
298     if all_results:
299         if GROUPS_TO_RUN != list(range(10)):
300             suffix = f"_grp{'_'.join(map(str, GROUPS_TO_RUN))}"
301             output = OUTPUT_XLSX.replace('.xlsx', f'{suffix}.xlsx')
302         else:
303             output = OUTPUT_XLSX
304         _save_results(all_results, all_gof, output)
305         print(f"\n-> Export final : {output}")
306 
307     print("\nDone.")
308 
309 
310 if __name__ == '__main__':
311     run()
```

E.3 GridSearch_BoxCox_Lambdas.py

Langage : Python 3	**Lignes de code :** 336 lignes	**Perturbation :** +10 % (arc)

Listing complet

```
import pandas as pd
import numpy as np
from scipy.optimize import minimize
from tqdm.auto import tqdm
import os
import time

# CONFIGURATION

DRIVE_FOLDER = r'\\student.fucam.ac.be\users$\nceci\Desktop\code'  # <-- ADAPTER

INPUT_FILE         = 'biogeme_del3.csv'
OUTPUT_XLSX        = 'lambdas_optimaux.xlsx'
SAVE_INTERMEDIAIRE = 'lambdas_intermediaires.csv'
SEP                = ','

GROUPS_TO_TEST = [0,1,2,3,4,5,6,7,8,9]

LAMBDA_MIN  = -2.4
LAMBDA_MAX  =  2.4
LAMBDA_STEP =  0.1

N_STARTS    = 50
MAX_ITER    = 500
MODEL       = 101
RANDOM_SEED = 42

# ================================================================================


def box_cox_np(x, lam):
```

```
    x_safe = np.clip(x, 1e-10, None)
    if abs(lam) < 1e-6:
        return np.log(x_safe)
    return (x_safe ** lam - 1.0) / lam


def snap_to_grid(lam):
    lam = round(lam / LAMBDA_STEP) * LAMBDA_STEP
    return float(np.clip(round(lam, 10), LAMBDA_MIN, LAMBDA_MAX))


def prepare_dataframe(df_raw):
    df = df_raw.copy()
    df['duration1'] = df['duration1'] / 3600.0
    df['duration2'] = df['duration2'] / 3600.0
    df['duration3'] = df['duration3'] / 3600.0
    df['qty1from']  = df['qty1from']  * 1e-6
    df['qty2from']  = df['qty2from']  * 1e-6
    df['qty3from']  = df['qty3from']  * 1e-6
    df['qty1to']    = df['qty1to']    * 1e-6
    df['qty2to']    = df['qty2to']    * 1e-6
    df['qty3to']    = df['qty3to']    * 1e-6
    df['length1']   = df['length1']   * 0.001
    df['length2']   = df['length2']   * 0.001
    df['length3']   = df['length3']   * 0.001
    df = df.fillna(df.max())
    total = df['qty'].sum()
    df['qty'] = len(df) * df['qty'] / total
    return df


def _loglik_from_utilities(V1, V2, V3, data):
    NEG_INF = -1e30
    V1_m = np.where(data['avail1'], V1, NEG_INF)
    V2_m = np.where(data['avail2'], V2, NEG_INF)
    V3_m = np.where(data['avail3'], V3, NEG_INF)
    V_max    = np.maximum(np.maximum(V1_m, V2_m), V3_m)
    exp1     = np.where(data['avail1'], np.exp(np.clip(V1_m - V_max, -500, 0)), 0.0)
    exp2     = np.where(data['avail2'], np.exp(np.clip(V2_m - V_max, -500, 0)), 0.0)
    exp3     = np.where(data['avail3'], np.exp(np.clip(V3_m - V_max, -500, 0)), 0.0)
    log_denom = np.log(exp1 + exp2 + exp3 + 1e-300) + V_max
    V_chosen  = np.where(data['choice'] == 1, V1_m,
                np.where(data['choice'] == 2, V2_m, V3_m))
```

```
    return -np.sum(data['qty'] * (V_chosen - log_denom))


def neg_loglik_model100(params, data):
    asc2, asc3, b_cost, b_dur1, b_dur2, b_dur3 = params
    penalty = sum(1e6 * v**2 for v in [b_cost, b_dur1, b_dur2, b_dur3] if v > 0)
    V1 = b_cost * data['bc_cost1'] + b_dur1 * data['bc_dur1']
    V2 = asc2 + b_cost * data['bc_cost2'] + b_dur2 * data['bc_dur2']
    V3 = asc3 + b_cost * data['bc_cost3'] + b_dur3 * data['bc_dur3']
    return _loglik_from_utilities(V1, V2, V3, data) + penalty


def neg_loglik_model101(params, data):
    (asc2, asc3, b_cost, b_dur1, b_dur2, b_dur3,
     b_af1, b_af2, b_af3, b_at1, b_at2, b_at3, b_decay) = params
    penalty  = sum(1e6 * v**2 for v in [b_cost, b_dur1, b_dur2, b_dur3, b_decay] if v > 0)
    penalty += sum(1e6 * v**2 for v in [b_af1, b_af2, b_af3, b_at1, b_at2, b_at3] if v < 0)
    decay1 = np.exp(np.clip(data['length1']**2 * b_decay, -500, 0))
    decay2 = np.exp(np.clip(data['length2']**2 * b_decay, -500, 0))
    decay3 = np.exp(np.clip(data['length3']**2 * b_decay, -500, 0))
    V1 = (b_cost * data['bc_cost1'] + b_dur1 * data['bc_dur1']
          + (b_af1 * data['qty1from'] + b_at1 * data['qty1to']) * decay1)
    V2 = (asc2 + b_cost * data['bc_cost2'] + b_dur2 * data['bc_dur2']
          + (b_af2 * data['qty2from'] + b_at2 * data['qty2to']) * decay2)
    V3 = (asc3 + b_cost * data['bc_cost3'] + b_dur3 * data['bc_dur3']
          + (b_af3 * data['qty3from'] + b_at3 * data['qty3to']) * decay3)
    return _loglik_from_utilities(V1, V2, V3, data) + penalty


def evaluate_loglik(lc, ld, data_arrays, model):
    data = dict(data_arrays)
    data['bc_cost1'] = box_cox_np(data_arrays['cost1'], lc)
    data['bc_cost2'] = box_cox_np(data_arrays['cost2'], lc)
    data['bc_cost3'] = box_cox_np(data_arrays['cost3'], lc)
    data['bc_dur1']  = box_cox_np(data_arrays['duration1'], ld)
    data['bc_dur2']  = box_cox_np(data_arrays['duration2'], ld)
    data['bc_dur3']  = box_cox_np(data_arrays['duration3'], ld)
    try:
        if model == 100:
            x0  = np.array([-2.0, -4.0, -0.5, -0.5, -0.1, -0.1])
            res = minimize(neg_loglik_model100, x0, args=(data,),
                           method='L-BFGS-B', options={'maxiter': 500, 'ftol': 1e-9})
        else:
```

```
            x0  = np.array([-3.0, -5.0, -0.1, -0.5, -0.05, -0.1,
                              1.0,  5.0,  2.0,  1.0,   5.0,  2.0, -1.0])
            res = minimize(neg_loglik_model101, x0, args=(data,),
                           method='L-BFGS-B', options={'maxiter': 1000, 'ftol': 1e-9})
        return -res.fun
    except Exception:
        return float('-inf')


def shotgun_hill_climbing(data_arrays, model, group_id):
    rng         = np.random.default_rng(RANDOM_SEED)
    grid_values = np.round(
        np.arange(LAMBDA_MIN, LAMBDA_MAX + LAMBDA_STEP / 2, LAMBDA_STEP), 2)
    n_grid      = len(grid_values)

    # Phase 1 — Shotgun
    print(f"\n  Phase 1 — Shotgun : {N_STARTS} random points...")
    idx_c  = rng.integers(0, n_grid, size=N_STARTS)
    idx_d  = rng.integers(0, n_grid, size=N_STARTS)
    starts = [(float(grid_values[ic]), float(grid_values[id_]))
              for ic, id_ in zip(idx_c, idx_d)]

    phase1_results = []
    t0 = time.time()
    for lc, ld in tqdm(starts, desc='  Shotgun', unit='pt'):
        ll = evaluate_loglik(lc, ld, data_arrays, model)
        phase1_results.append((lc, ld, ll))
    t1 = time.time()

    phase1_results.sort(key=lambda x: x[2], reverse=True)
    best_lc, best_ld, best_ll = phase1_results[0]
    print(f"  Phase 1 done in {t1-t0:.1f}s")
    print(f"  Best shotgun: λ_cost={best_lc:.2f}, λ_dur={best_ld:.2f}, LL={best_ll:.2f}")

    # Phase 2 — Hill climbing with backtracking
    print(f"\n  Phase 2 — Hill climbing with backtracking (max {MAX_ITER} iter)...")
    t2 = time.time()

    directions = [( LAMBDA_STEP,  0.0), (-LAMBDA_STEP,  0.0),
                  ( 0.0,  LAMBDA_STEP), ( 0.0, -LAMBDA_STEP)]

    visited         = {}
    backtrack_stack = []
```

```
    current_lc      = best_lc
    current_ld      = best_ld
    current_ll      = best_ll
    visited[(round(current_lc, 2), round(current_ld, 2))] = current_ll
    global_best_lc  = current_lc
    global_best_ld  = current_ld
    global_best_ll  = current_ll
    backtrack_stack.append((current_lc, current_ld))

    def get_unvisited_neighbors(lc, ld):
        neighbors = []
        for dlc, dld in directions:
            nlc = snap_to_grid(lc + dlc)
            nld = snap_to_grid(ld + dld)
            key = (round(nlc, 2), round(nld, 2))
            if key not in visited:
                neighbors.append((nlc, nld))
        return neighbors

    pbar   = tqdm(total=MAX_ITER, desc='  Hill climbing', unit='iter')
    n_iter = 0

    while backtrack_stack and n_iter < MAX_ITER:
        n_iter += 1
        pbar.update(1)
        pbar.set_postfix({'LL': f'{global_best_ll:.2f}',
                          'λc': f'{global_best_lc:.2f}',
                          'λd': f'{global_best_ld:.2f}',
                          'pts': len(visited)})

        neighbors = get_unvisited_neighbors(current_lc, current_ld)

        if not neighbors:
            backtrack_stack.pop()
            if backtrack_stack:
                current_lc, current_ld = backtrack_stack[-1]
                current_ll = visited[(round(current_lc, 2), round(current_ld, 2))]
            continue

        best_n_lc, best_n_ld, best_n_ll = None, None, float('-inf')
        for nlc, nld in neighbors:
            key = (round(nlc, 2), round(nld, 2))
            ll  = evaluate_loglik(nlc, nld, data_arrays, model)
```

```
            visited[key] = ll
            if ll > best_n_ll:
                best_n_ll = ll
                best_n_lc = nlc
                best_n_ld = nld

        if best_n_ll > current_ll:
            current_lc = best_n_lc
            current_ld = best_n_ld
            current_ll = best_n_ll
            backtrack_stack.append((current_lc, current_ld))
            if current_ll > global_best_ll:
                global_best_ll = current_ll
                global_best_lc = current_lc
                global_best_ld = current_ld
        else:
            backtrack_stack.pop()
            if backtrack_stack:
                current_lc, current_ld = backtrack_stack[-1]
                current_ll = visited[(round(current_lc, 2), round(current_ld, 2))]

    pbar.close()
    t3 = time.time()

    print(f"\n  Phase 2 done in {t3-t2:.1f}s | {n_iter} iterations")
    print(f"  {len(visited)} points evaluated")
    print(f"\n  *** RESULT group {group_id} ***")
    print(f"  λ_cost     = {global_best_lc:.2f}")
    print(f"  λ_duration = {global_best_ld:.2f}")
    print(f"  LL         = {global_best_ll:.2f}")

    return {
        'groupe':           group_id,
        'lambda_cost':      round(global_best_lc, 2),
        'lambda_duration':  round(global_best_ld, 2),
        'log_likelihood':   round(global_best_ll, 2),
        'n_points_evalues': len(visited),
        'n_iter_phase2':    n_iter,
        'temps_total_s':    round(t3 - t0, 1),
        'visited':          visited,
    }
```

```
def run():
    print("  SHOTGUN HILL CLIMBING")
    print(f"  Model {MODEL} | Grid [{LAMBDA_MIN}, {LAMBDA_MAX}] step {LAMBDA_STEP}")
    print(f"  Groups: {GROUPS_TO_TEST}")
    print(f"  Output folder: {DRIVE_FOLDER}")

    input_path = os.path.join(DRIVE_FOLDER, INPUT_FILE)
    print(f"\nLoading {input_path}...")
    df_all = pd.read_csv(input_path, sep=SEP)
    df_all = df_all.apply(pd.to_numeric, errors='coerce')
    df_all.columns = map(str.lower, df_all.columns)
    df_all = df_all[df_all['qty'] > 0].copy()
    print(f"  -> {len(df_all):,} observations loaded")

    base_cols = ['cost1', 'cost2', 'cost3',
                 'duration1', 'duration2', 'duration3',
                 'avail1', 'avail2', 'avail3', 'choice', 'qty']
    acc_cols  = ['length1', 'length2', 'length3',
                 'qty1from', 'qty2from', 'qty3from',
                 'qty1to',   'qty2to',   'qty3to']
    cols = base_cols + (acc_cols if MODEL == 101 else [])

    all_results = []
    t_global = time.time()

    for g in GROUPS_TO_TEST:
        df_group = df_all[df_all['grp'] == g].copy()
        if df_group.empty:
            print(f"\nGroup {g} not found, skipping.")
            continue

        print(f"\n{'='*65}")
        print(f"  GROUP {g}  |  {len(df_group):,} observations")
        print(f"{'='*65}")

        df_prepared = prepare_dataframe(df_group)
        data_arrays = {c: df_prepared[c].values.astype(np.float64) for c in cols}

        result = shotgun_hill_climbing(data_arrays, MODEL, g)
        all_results.append(result)

        df_inter = pd.DataFrame([
            {k: v for k, v in r.items() if k != 'visited'}
```

```
            for r in all_results
        ])
        inter_path = os.path.join(DRIVE_FOLDER, SAVE_INTERMEDIAIRE)
        df_inter.to_csv(inter_path, index=False)
        print(f"  -> Intermediate save: {inter_path}")

    total = time.time() - t_global
    print(f"\n{'='*65}")
    print(f"  All groups done in {total/60:.1f} min ({total/3600:.2f}h)")
    print(f"{'='*65}")

    xlsx_path = os.path.join(DRIVE_FOLDER, OUTPUT_XLSX)
    with pd.ExcelWriter(xlsx_path, engine='openpyxl') as writer:

        pd.DataFrame([
            {k: v for k, v in r.items() if k != 'visited'}
            for r in all_results
        ]).to_excel(writer, sheet_name='Lambdas optimaux', index=False)

        lines = ["# Paste into SolveIntegratedModel_NUTS3.py\n\n"]
        lines += ["LAMBDA_COST_BY_GROUP = {\n"]
        for r in all_results:
            lines.append(f"    {r['groupe']}: {r['lambda_cost']},\n")
        lines += ["}\n\nLAMBDA_DURATION_BY_GROUP = {\n"]
        for r in all_results:
            lines.append(f"    {r['groupe']}: {r['lambda_duration']},\n")
        lines += ["}\n"]
        pd.DataFrame({'Code Python': [''.join(lines)]}).to_excel(
            writer, sheet_name='Code Python', index=False)

        for r in all_results:
            df_path = pd.DataFrame(
                [(lc, ld, ll) for (lc, ld), ll in r['visited'].items()],
                columns=['lambda_cost', 'lambda_duration', 'log_likelihood']
            ).sort_values('log_likelihood', ascending=False).reset_index(drop=True)
            df_path.to_excel(writer, sheet_name=f"Grp{r['groupe']}_points", index=False)

    print(f"\n-> Excel saved: {xlsx_path}")
    print("\nFINAL SUMMARY:")
    for r in all_results:
        print(f"  Group {r['groupe']}: λ_cost={r['lambda_cost']:.2f}, "
              f"λ_duration={r['lambda_duration']:.2f}, LL={r['log_likelihood']:.2f}, "
              f"{r['n_points_evalues']} pts, {r['temps_total_s']}s")
```

```
333
334
335 if __name__ == '__main__':
336     run()
```

E.4 SolveIntegratedModel_NUTS3.py

Langage : Python 3	**Lignes de code :** 234 lignes	**Perturbation :** +10 % (arc)

Listing complet

```python
import pandas as pd
import biogeme.database as db
import biogeme.biogeme as bio
import biogeme.models as models
from biogeme.expressions import Beta, exp
import os
import time

LAMBDA_COST_BY_GROUP = {
    0: 1.0, 1: 1.0, 2: 1.0, 3: 1.0, 4: 1.0,
    5: 1.0, 6: 1.0, 7: 1.0, 8: 1.0, 9: 1.0,
}

LAMBDA_DURATION_BY_GROUP = {
    0: 1.0, 1: 1.0, 2: 1.0, 3: 1.0, 4: 1.0,
    5: 1.0, 6: 1.0, 7: 1.0, 8: 1.0, 9: 1.0,
}

INPUT_FILE = 'biogeme_del3.csv'
SEP = ','
OUTPUT_DIR = 'models'


def box_cox(x, lam):
    return (x ** lam - 1) / lam


def run():

    model = 101  # 100 = no accessibility, 101 = with accessibility

```

```
    wd = os.path.join(os.path.dirname(os.path.abspath(__file__)), OUTPUT_DIR)
    if not os.path.exists(wd):
        os.mkdir(wd)
    os.chdir(wd)

    print(f"Loading {INPUT_FILE}...")
    input_path = os.path.join(os.path.dirname(os.path.abspath(__file__)), INPUT_FILE)
    df_all = pd.read_csv(input_path, sep=SEP)
    df_all = df_all.apply(pd.to_numeric, errors='coerce')
    df_all.columns = map(str.lower, df_all.columns)
    print(f"  -> {len(df_all):,} observations, groups: {sorted(df_all['grp'].dropna().unique().astype(int).tolist())}")

    groups = [2]  # replace with sorted(df_all['grp'].dropna().unique().astype(int).tolist()) to run all groups

    outlier_filters = {
     4: 'qty2 < 1200000'
     }


    all_results = []

    for g in groups:

        print(f"\n{'='*50}")
        print(f"Solving model {model} for group {g}")
        print(f"{'='*50}")

        df = df_all[df_all['grp'] == g].copy()
        df = df[df['qty'] > 0].copy()
        print(f"  -> {len(df):,} observations (qty > 0)")

        if g in outlier_filters:
            df = df.query(outlier_filters[g]).copy()
            print(f"  -> Outlier filter applied: {outlier_filters[g]}")

        if df.empty:
            print(f"  -> No data for group {g}, skipping.")
            continue

        df['duration1'] = df['duration1'] / 3600
        df['duration2'] = df['duration2'] / 3600
        df['duration3'] = df['duration3'] / 3600

```

```
        df['qty1from'] = df['qty1from'] * 0.000001
        df['qty2from'] = df['qty2from'] * 0.000001
        df['qty3from'] = df['qty3from'] * 0.000001
        df['qty1to']   = df['qty1to']   * 0.000001
        df['qty2to']   = df['qty2to']   * 0.000001
        df['qty3to']   = df['qty3to']   * 0.000001

        df['length1'] = df['length1'] * 0.001
        df['length2'] = df['length2'] * 0.001
        df['length3'] = df['length3'] * 0.001

        df = df.fillna(df.max())

        database = db.Database('data', df)
        globals().update(database.variables)

        total = df['qty'].sum()
        df['qty'] = len(df) * df['qty'] / total

        INTERCEPT1 = Beta('intercept.1', 0, None, None, 1)
        INTERCEPT2 = Beta('intercept.2', 0, None, None, 0)
        INTERCEPT3 = Beta('intercept.3', 0, None, None, 0)

        B_COST = Beta('b_cost', -1, None, 0, 0)

        B_DURATION1 = Beta('b_duration.1', -1, None, 0, 0)
        B_DURATION2 = Beta('b_duration.2', -1, None, 0, 0)
        B_DURATION3 = Beta('b_duration.3', -1, None, 0, 0)

        lc = LAMBDA_COST_BY_GROUP.get(g, 1.0)
        ld = LAMBDA_DURATION_BY_GROUP.get(g, 1.0)
        LAMBDA_COST     = Beta('lambda_cost',     lc, None, None, 1)
        LAMBDA_DURATION = Beta('lambda_duration', ld, None, None, 1)
        print(f"  -> Fixed lambdas: lambda_cost={lc}, lambda_duration={ld}")

        B_ACC_FROM1 = Beta('b_acc_from.1', 0, 0, None, 0)
        B_ACC_FROM2 = Beta('b_acc_from.2', 0, 0, None, 0)
        B_ACC_FROM3 = Beta('b_acc_from.3', 0, 0, None, 0)
        B_ACC_TO1   = Beta('b_acc_to.1',   0, 0, None, 0)
        B_ACC_TO2   = Beta('b_acc_to.2',   0, 0, None, 0)
        B_ACC_TO3   = Beta('b_acc_to.3',   0, 0, None, 0)

        B_DECAY = Beta('b_decay', -1, None, 0, 0)
```

```
118
119         modelName = 'Model' + str(model)
120
121         if model == 100:
122             V1 = INTERCEPT1 + B_COST * box_cox(cost1, LAMBDA_COST) + B_DURATION1 * box_cox(duration1, LAMBDA_DURATION)
123             V2 = INTERCEPT2 + B_COST * box_cox(cost2, LAMBDA_COST) + B_DURATION2 * box_cox(duration2, LAMBDA_DURATION)
124             V3 = INTERCEPT3 + B_COST * box_cox(cost3, LAMBDA_COST) + B_DURATION3 * box_cox(duration3, LAMBDA_DURATION)
125
126         if model == 101:
127             V1 = (INTERCEPT1
128                   + B_COST * box_cox(cost1, LAMBDA_COST)
129                   + B_DURATION1 * box_cox(duration1, LAMBDA_DURATION)
130                   + (B_ACC_FROM1 * qty1from + B_ACC_TO1 * qty1to) * exp(length1 ** 2 * B_DECAY))
131
132             V2 = (INTERCEPT2
133                   + B_COST * box_cox(cost2, LAMBDA_COST)
134                   + B_DURATION2 * box_cox(duration2, LAMBDA_DURATION)
135                   + (B_ACC_FROM2 * qty2from + B_ACC_TO2 * qty2to) * exp(length2 ** 2 * B_DECAY))
136
137             V3 = (INTERCEPT3
138                   + B_COST * box_cox(cost3, LAMBDA_COST)
139                   + B_DURATION3 * box_cox(duration3, LAMBDA_DURATION)
140                   + (B_ACC_FROM3 * qty3from + B_ACC_TO3 * qty3to) * exp(length3 ** 2 * B_DECAY))
141
142         V  = {1: V1, 2: V2, 3: V3}
143         av = {1: avail1, 2: avail2, 3: avail3}
144
145         logprob = models.loglogit(V, av, choice)
146         formulas = {'loglike': logprob, 'weight': qty}
147
148         biogeme_model = bio.BIOGEME(database, formulas)
149         biogeme_model.modelName = modelName + '-' + str(g)
150
151         for ext in ['.html', '.pickle', '_parameters.json']:
152             f = biogeme_model.modelName + ext
153             if os.path.exists(f):
154                 os.remove(f)
155
156         print(f"  -> {len(df)} observations, {df['avail2'].sum()} with IWW available")
157         print(f"  -> Starting estimation...")
158         start_time = time.time()
159         results = biogeme_model.estimate()
160         elapsed = time.time() - start_time
```

```
161         print(f"  -> Done in {elapsed:.1f}s")
162
163         try:
164             pandasResults = results.get_estimated_parameters()
165         except AttributeError:
166             pandasResults = results.getEstimatedParameters()
167         print(pandasResults)
168
169         params = pandasResults['Value']
170         issues = []
171         for name, val in params.items():
172             if 'b_cost' in name and val >= 0:
173                 issues.append(f"  WARNING: {name} = {val:.4f} (expected < 0)")
174             if 'b_duration' in name and val >= 0:
175                 issues.append(f"  WARNING: {name} = {val:.4f} (expected < 0)")
176             if 'b_decay' in name and val >= 0:
177                 issues.append(f"  WARNING: {name} = {val:.4f} (expected < 0)")
178             if 'b_acc_from' in name and val <= 0:
179                 issues.append(f"  WARNING: {name} = {val:.4f} (expected > 0)")
180             if 'b_acc_to' in name and val <= 0:
181                 issues.append(f"  WARNING: {name} = {val:.4f} (expected > 0)")
182
183         if issues:
184             print(f"  Sign issues for group {g}:")
185             for w in issues:
186                 print(w)
187         else:
188             print(f"  All signs OK for group {g}.")
189
190         pandasResults['Group'] = g
191         all_results.append(pandasResults.reset_index().rename(columns={'index': 'Parameter'}))
192
193     if all_results:
194         df_export = pd.concat(all_results, ignore_index=True)
195
196         cols = ['Group', 'Parameter', 'Value', 'Rob. Std err', 'Rob. t-test', 'Rob. p-value']
197         cols = [c for c in cols if c in df_export.columns]
198         df_export = df_export[cols]
199
200         def expected_sign(param):
201             if any(x in param for x in ['b_cost', 'b_duration', 'b_decay']):
202                 return '< 0'
203             if 'b_acc' in param or 'lambda' in param:
```

```
                return '> 0'
            return 'free'

        def sign_ok(row):
            exp_sign = expected_sign(row['Parameter'])
            if exp_sign == '< 0':
                return 'OK' if row['Value'] < 0 else ('Nul' if row['Value'] == 0 else 'Mauvais signe')
            if exp_sign == '> 0':
                return 'OK' if row['Value'] > 0 else ('Nul' if row['Value'] == 0 else 'Mauvais signe')
            return '-'

        df_export['Expected sign'] = df_export['Parameter'].apply(expected_sign)
        df_export['Sign status']   = df_export.apply(sign_ok, axis=1)

        for c in ['Value', 'Rob. Std err', 'Rob. t-test', 'Rob. p-value']:
            if c in df_export.columns:
                df_export[c] = df_export[c].round(6)

        output_path = '/content/resultats_DE_NUTS3.xlsx'
        with pd.ExcelWriter(output_path, engine='openpyxl') as writer:
            df_export.to_excel(writer, sheet_name='Tous les parametres', index=False)
            df_export.pivot(index='Parameter', columns='Group', values='Value').to_excel(writer, sheet_name='Valeurs par
groupe')
            df_export.pivot(index='Parameter', columns='Group', values='Sign status').to_excel(writer, sheet_name='Statut des
signes')

        print(f"\n-> Results exported to: {output_path}")

    print("Done.")


if __name__ == "__main__":
    run()
```