# **Procédure de comblement et de fusion des données d’observations et TAMSAT avec correction de biais**

[**https://zenodo.org/records/2745243**](https://zenodo.org/records/2745243)

**Recommandation principale :** Aligner les séries chronologiques d’observations au sol et de données TAMSAT par localisation et date, estimer le biais systématique via une méthode d’ajustement statistique (par ex. régression linéaire ou quantile mapping), corriger la série TAMSAT, puis fusionner les deux ensembles pour obtenir une série homogène et sans lacunes.

## **1. Préparation des jeux de données**

1.1. Rassembler les données d’observations pluviométriques au sol (stationnaires ou réseaux manuels) et les données TAMSAT (satellite).  
 1.2. Assurer une structure commune :

* Variables clés : date, latitude, longitude, précipitation.
* Format de date unifié (ISO : YYYY-MM-DD).
* Résolution spatiale comparable (agrégation ou interpolation si nécessaire).

## **2. Identification des correspondances**

* Pour chaque station d’observation, extraire la série TAMSAT la plus proche (pixel satellite) :
  + Par la distance géographique : sélection du pixel TAMSAT dont le centre est le plus proche de la station.
  + Facultatif : moyenne des 4 pixels adjacents pour réduire le bruit spatial.

## **3. Estimation du biais systématique**

Deux méthodes courantes :

**3.1. Régression linéaire simple**

* Modèle : P\_obs = α + β × P\_tamsat + ε
* Estimer α (offset) et β (pente) sur la période commune.
* Correction : P\_tamsat\_corrigé = (P\_tamsat − α) / β

**3.2. Quantile Mapping**

* Comparer la distribution cumulative empirique (CDF) des précipitations observées et TAMSAT.
* Pour chaque quantile q, calculer la différence Δ(q) = F\_obs^−1(q) − F\_tamsat^−1(q).
* Correction : P\_tamsat\_corrigé = F\_obs^−1(F\_tamsat(P\_tamsat))

Le *quantile mapping* permet de corriger non seulement l’erreur de moyenne (biais), mais aussi les écarts de variances et la forme de la distribution.

## **4. Application de la correction**

1. Appliquer la relation (régression ou quantile mapping) à chaque valeur de la série TAMSAT.
2. Vérifier la cohérence :
   * Calculer les statistiques avant/après (moyenne, écart-type, quantiles).
   * Visualiser via un diagramme en nuage de points et l’histogramme des résidus.

## **5. Comblement (imputation) des données manquantes**

1. Identifier les périodes manquantes dans la série d’observations.
2. Remplacer ces lacunes par les valeurs corrigées de la série TAMSAT.
3. Optionnel : lisser la jonction au point de raccord (moyenne mobile sur ± n jours).

## **6. Fusion des séries**

* Construire une unique série temporelle pour chaque station :
  + Utiliser les données d’observations quand disponibles.
  + Sinon, utiliser la donnée TAMSAT corrigée.
* Annoter la source pour chaque point (obs vs. TAMSAT\_corrigé) pour traçabilité.

## **7. Validation finale**

* Comparer la série fusionnée à un jeu de validation indépendant si disponible.
* Effectuer un **test de cohérence temporelle** (ex. auto-corrélation) et spatiale (variogramme).
* Ajuster la méthode de correction ou la résolution spatiale si des écarts anormaux subsistent.

## **Ressources vidéo recommandées**

Pour un tutoriel visuel pas-à-pas, consulter :

1. “Bias Correction of Satellite Rainfall Estimates” (YouTube) – démonstration du quantile mapping sur R ou Python.
2. “Merging Satellite and Ground Rainfall Data” – tutoriel sur GitHub avec notebooks Python.

Ces vidéos illustrent l’importation des données, l’ajustement statistique, la correction et la fusion avec des packages comme xarray, pandas et scipy.

# **Exécution, personnalisation et distribution de CDBC avec interface graphique**

Pour rendre l’utilisation du **Climate Data Bias Corrector (CDBC)** plus conviviale, paramétrable et facilement partageable entre collègues, vous pouvez :

1. Organiser clairement vos jeux de données
2. Centraliser la configuration
3. Ajouter une interface graphique légère
4. Emballer le tout pour installation via pip
5. Documenter l’usage et distribuer

## **1. Organisation des dossiers**

Structure recommandée

text

CDBC-project/

├── data/

│ ├── input/ ← fichiers bruts (NetCDF, CSV…)

│ └── reference/ ← données d’observation climatologique

├── results/

│ └── YYYYMMDD/ ← sortie du correcteur, datée

├── src/

│ └── CDBC.py ← script principal (renommé)

├── gui/

│ └── gui.py ← interface graphique

├── config/

│ └── default.ini ← paramètres par défaut

├── setup.py ← paquet Python

└── README.md

* **data/input** : modèles GCM/RCM
* **data/reference** : climatologies observées
* **results** : sorties classées par date ou scénario
* **config/default.ini** : paramètres CLI (méthode, seuils…)
* **src/CDBC.py** : votre script de correction
* **gui/gui.py** : le frontend Tkinter ou PyQt

## **2. Centralisation de la configuration**

Créez un fichier config/default.ini :

text

[paths]

input\_dir = data/input

reference\_dir = data/reference

output\_dir = results

[method]

name = QMAP

params = {}

[logging]

level = INFO

Dans src/CDBC.py, chargez la config avec configparser et argparse :

python

**import** argparse, configparser, os

**from** cdbc\_core **import** run\_bias\_correction *# fonction principale*

*# Chargement config*

cfg = configparser.ConfigParser()

cfg.read(os.path.join(os.path.dirname(\_\_file\_\_), '../config/default.ini'))

*# CLI*

parser = argparse.ArgumentParser(description="CDBC Climate Data Bias Corrector")

parser.add\_argument('--input-dir', default=cfg['paths']['input\_dir'])

parser.add\_argument('--reference-dir', default=cfg['paths']['reference\_dir'])

parser.add\_argument('--output-dir', default=cfg['paths']['output\_dir'])

parser.add\_argument('--method', default=cfg['method']['name'])

args = parser.parse\_args()

*# Exécution*

run\_bias\_correction(

input\_dir=args.input\_dir,

reference\_dir=args.reference\_dir,

output\_dir=args.output\_dir,

method=args.method

)

## **3. Interface graphique simple**

## **Avec Tkinter (standard Python)**

Créez gui/gui.py :

python

**import** tkinter **as** tk

**from** tkinter **import** filedialog, messagebox

**from** src.CDBC **import** run\_bias\_correction

**def** select\_dir(var):

d = filedialog.askdirectory()

**if** d: var.set(d)

**def** launch():

**try**:

run\_bias\_correction(

input\_dir=inp.get(),

reference\_dir=ref.get(),

output\_dir=out.get(),

method=method.get()

)

messagebox.showinfo("CDBC", "Correction terminée avec succès")

**except** Exception **as** e:

messagebox.showerror("Erreur", str(e))

root = tk.Tk()

root.title("CDBC GUI")

*# Variables*

inp, ref, out, method = [tk.StringVar() **for** \_ **in** range(4)]

method.set("QMAP")

*# Layout*

**for** idx, (label, var) **in** enumerate([

("Input dir:", inp),

("Reference dir:", ref),

("Output dir:", out),

("Method:", method)

]):

tk.Label(root, text=label).grid(row=idx, column=0, sticky="e")

tk.Entry(root, textvariable=var, width=40).grid(row=idx, column=1)

**if** idx < 3:

tk.Button(root, text="Browse", command=**lambda** v=var: select\_dir(v)).grid(row=idx, column=2)

tk.Button(root, text="Run", command=launch).grid(row=4, column=1)

root.mainloop()

Lancez la GUI :

bash

python gui/gui.py

## **4. Emballer et distribuer**

## **setup.py**

python

**from** setuptools **import** setup, find\_packages

setup(

name="cdbc\_tool",

version="1.0.0",

packages=find\_packages(),

entry\_points={

'console\_scripts': [

'cdbc=src.CDBC:main', *# main() appelle argparse et run\_bias\_correction*

'cdbc-gui=gui.gui:main' *# main() lance la boucle Tkinter*

]

},

install\_requires=[

'numpy', 'pandas', 'netCDF4', 'xarray'

]

)

Installez localement :

bash

pip install --upgrade pip

pip install .

Vous pouvez désormais :

* Lancer en CLI :
* bash
* cdbc --help
* Lancer en GUI :
* bash
* cdbc-gui

## **Packaging pour collègues**

1. Créez un environnement virtuel vierge
2. Installez votre paquet via pip install .
3. Documentez l’usage dans README.md
4. Partagez le répertoire ou poussez sur un dépôt privé

## **5. Personnalisation et harmonisation**

* Modifiez le fichier config/default.ini pour ajuster les méthodes, chemins, niveaux de logging…
* Dans le script src/CDBC.py, exposez tous les paramètres de correction (ex. seuils, options de lissage) via argparse et dans la GUI par des widgets supplémentaires (menus déroulants, cases à cocher).
* Organisez les sorties dans results/YYYYMMDD en ajoutant au script :
* python
* **import** datetime
* date = datetime.datetime.now().strftime("%Y%m%d")
* out\_dir = os.path.join(args.output\_dir, date)
* os.makedirs(out\_dir, exist\_ok=True)
* Éventuellement, créez un **Dockerfile** pour standardiser l’environnement et simplifier le déploiement.

Avec cette architecture, vous avez un **outil modulaire**, **configurable**, doté d’une **interface graphique** et facilement **installable/distribuable** auprès de vos collègues.

Pour la correction de biais des séries temporelles climatiques, la prise en compte ou non des années bissextiles (et donc du 29 février) dans les données observées dépend essentiellement de la méthode de biais utilisée et de la cohérence à assurer entre les jeux de données (observations vs projections).

## **1. Pourquoi se préoccuper du 29 février ?**

* Les années bissextiles comportent un jour supplémentaire : sans traitement, les statistiques journalières (moyennes, quantiles, fréquences de dépassement) seront faussées si l’on mélange des calendriers à 365 et 366 jours.
* La plupart des GCM/RCM utilisent un **calendrier « proleptique »** standard (pairs nombres de jours par mois, années bissextiles incluses) ou un calendrier utopique à 360 jours. Les observations, en revanche, suivent le calendrier grégorien réel.

## **2. Stratégies de traitement**

## **A. Conserver le 29 février dans tous les jeux de données**

1. **Ajouter systématiquement le 29 février** aux observations pour chaque année bissextile.
2. Générer un jour factice (ex. interpolation linéaire entre 28 février et 1er mars) si vos projections n’ont pas ce jour.
3. Avantage : vous disposez d’un jeu “complet” à 366 jours, compatible avec un calendrier grégorien.
4. Inconvénient : si votre projection n’a pas de 29 février, vous introduisez une valeur artificielle qui peut légèrement biaiser les statistiques.

## **B. Supprimer le 29 février pour tous les jeux de données**

1. **Supprimer** ou **ignorer** le 29 février dans les observations.
2. Conserver uniquement 365 jours par année pour observations et projections.
3. Avantage : parfaite concordance journalière sans interpolation.
4. Inconvénient : perte d’information réelle (mais souvent négligeable sur les longues séries).

## **C. Traitement spécifique par méthode**

Certaines méthodes, notamment les méthodes de quantile mapping (QM), peuvent gérer les jours manquants ou les doublons :

* En QM journalière, on classe chaque jour par rang dans l’année ; la présence ou l’absence du 29 février modifie ce rang.
* Il est possible de faire un **rangement cyclique** (regrouper tous les 29 fév. comme un seul jour de l’année cyclique) ou de lisser sur 5–7 jours autour de la date.

## **3. Recommandation pratique**

1. **Vérifiez le calendrier** de vos projections :
   * Si elles incluent le 29 février, optez pour la stratégie A et conservez ce jour dans les observations (avec ou sans interpolation).
   * Si elles n’incluent pas le 29 février (calendrier à 365 jours), choisissez la stratégie B pour tout uniformiser à 365 jours.
2. **Automatisez dans votre script** CDBC.py :
3. python
4. **import** pandas **as** pd
5. **def** harmonize\_calendar(df\_obs, include\_feb29):
6. *# df\_obs : DataFrame indexé en datetime*
7. **if** include\_feb29:
8. *# ajouter ou conserver 29 février*
9. **return** df\_obs.asfreq('D') *# interpoler les journées manquantes*
10. **else**:
11. *# supprimer tous les 29 février*
12. **return** df\_obs[~((df\_obs.index.month == 2) & (df\_obs.index.day == 29))]
13. **Paramétrez par défaut** dans config/default.ini :
14. text
15. [calendar]
16. include\_feb29 = yes # ou no
    * Exposez en CLI et GUI une option “Inclure 29 février” pour que chacun choisisse.
17. **Documentez** clairement ce choix dans votre README et assurez-vous que tous les utilisateurs appliquent la même logique afin de garantir la reproductibilité.

En résumé, **le 29 février doit être soit uniformément conservé, soit uniformément supprimé** dans l’ensemble des séries pour éviter de fausser les statistiques de biais. Le mieux est d’intégrer cette logique en amont dans votre script et de la rendre paramétrable via la config et l’interface.

# **Intégration des données satellitaires (TAMSAT) et observations dans CDBC**

Pour corriger les biais des données satellitaires TAMSAT et les fusionner avec vos observations de référence, vous pouvez étendre votre script et votre interface graphique de la manière suivante :

## **1. Prétraitement des données TAMSAT**

1. **Format des fichiers TAMSAT** – Les sorties TAMSAT sont généralement en NetCDF (.nc) ou en formats raster (GeoTIFF).  
    – Assurez-vous d’avoir un format unifié (ex. NetCDF) pour faciliter la lecture via xarray/netCDF4.
2. **Chargement et alignement temporel**
3. python
4. **import** xarray **as** xr
5. **def** load\_tamsat(path):
6. ds = xr.open\_dataset(path)
7. *# renommer ou extraire la variable pluie, ex. 'rainfall'*
8. rain = ds['rainfall']
9. *# convertir l’index en pandas.DatetimeIndex si besoin*
10. rain['time'] = pd.to\_datetime(rain['time'].values)
11. **return** rain
12. **Harmonisation des calendriers** Appliquez la même fonction de gestion du 29 février (cf. plus haut) sur TAMSAT et observations pour garantir la concordance journalière.

## **2. Calcul des biais spécifiques à TAMSAT**

1. **Choix de la méthode** – *Quantile mapping* quotidien  
    – *Delta change* mensuel  
    – *Multiplicative* sur cumul mensuel
2. **Fonction dédiée** Dans src/CDBC.py, ajoutez :
3. python
4. **def** compute\_bias(obs: xr.DataArray, sat: xr.DataArray, method: str):
5. """
6. Calcule le biais (satellite vs obs) selon la méthode choisie.
7. Retourne un DataArray de facteurs ou d’ajustements à appliquer.
8. """
9. **if** method.lower() == 'delta':
10. *# biais mensuel : ratio moy\_sat / moy\_obs pour chaque mois*
11. bias = (sat.groupby('time.month').mean() /
12. obs.groupby('time.month').mean())
13. **return** bias
14. **elif** method.lower() == 'qm':
15. *# quantile mapping journalier (exemple simplifié)*
16. **return** quantile\_mapping\_bias(obs, sat)
17. **else**:
18. **raise** ValueError(f"Méthode inconnue : {method}")
19. **Application de la correction**
20. python
21. **def** apply\_bias\_correction(sat: xr.DataArray, bias: xr.DataArray):
22. *# Ex. pour delta change : diviser ou multiplier*
23. **return** sat / bias.sel(month=sat['time.month'])

## **3. Fusion (merge) des séries corrigées**

1. **Interpolation spatiale ou temporelle** – Si TAMSAT est en grille et observations ponctuelles, utilisez xarray.interp ou scipy.interpolate.
2. **Merging**
3. python
4. **def** merge\_series(obs: xr.DataArray, sat\_corr: xr.DataArray):
5. *# Rééchantillonne sat\_corr sur les dates obs ou vice versa*
6. sat\_resampled = sat\_corr.sel(time=obs.time, method='nearest')
7. *# Combiner : moyenne pondérée, ou substitution selon disponibilité*
8. merged = xr.where(obs.notnull(), obs, sat\_resampled)
9. **return** merged

## **4. Extension de l’interface graphique**

1. **Nouvel onglet “TAMSAT”** – Utilisez ttk.Notebook pour créer plusieurs onglets (CLI équivalent via --data-type).
2. **Widgets à ajouter**
   * Sélecteur de fichier ou dossier TAMSAT
   * Choix de la méthode de biais (menu déroulant : “delta”, “QM”, etc.)
   * Bouton “Calculer biais” et “Appliquer correction”
3. **Exemple Tkinter**
4. python
5. **from** tkinter **import** ttk
6. notebook = ttk.Notebook(root)
7. *# Onglet Observations (existant)*
8. tab\_obs = ttk.Frame(notebook)
9. *# Onglet TAMSAT*
10. tab\_sat = ttk.Frame(notebook)
11. notebook.add(tab\_obs, text='Observations')
12. notebook.add(tab\_sat, text='TAMSAT')
13. notebook.pack(expand=1, fill='both')
14. *# Dans tab\_sat :*
15. sat\_path = tk.StringVar()
16. method = tk.StringVar(value='delta')
17. ttk.Label(tab\_sat, text="Dossier TAMSAT:").grid(...)
18. ttk.Entry(tab\_sat, textvariable=sat\_path).grid(...)
19. ttk.Button(tab\_sat, text="Parcourir", command=**lambda**: select\_dir(sat\_path)).grid(...)
20. ttk.Label(tab\_sat, text="Méthode biais:").grid(...)
21. ttk.OptionMenu(tab\_sat, method, 'delta', 'delta', 'qm').grid(...)
22. ttk.Button(tab\_sat, text="Calculer & Appliquer", command=launch\_sat).grid(...)
23. Dans launch\_sat(), appelez les fonctions load\_tamsat(), compute\_bias(), apply\_bias\_correction(), puis merge\_series().

## **5. Mise à disposition à vos collègues**

1. Mettez à jour setup.py pour inclure la dépendance à xarray et scipy :
2. python
3. install\_requires=[
4. 'numpy','pandas','netCDF4','xarray','scipy'
5. ]
6. Publiez sur un dépôt Git privé ou un package interne PyPI.
7. Documentez le flux de travail dans README.md et ajoutez un exemple de config/default.ini pour la section TAMSAT :
8. text
9. [tamsat]
10. data\_dir = data/tamsat
11. bias\_method = delta
12. merge\_strategy = weighted
13. Vos collègues installent par :
14. bash
15. pip install cdbc\_tool *# votre package*
16. cdbc-gui

Avec cette extension, vous pouvez **charger**, **corriger les biais** et **fusionner** vos données satellites TAMSAT et observations, le tout via une interface claire et partageable.