TD Modélisation - Informatique:

Analyse de séries temporelles (4h)

Préambule sur l'UE:

Cet enseignement vise à approfondir les compétences en algorithmique, en programmation scientifique et en modélisation numérique autour d'exemples concrets issus des géosciences (séries temporelles, imagerie satellite, MNT, microstructures, volcanologie, géologie structurale, géomorphologie, etc.).

L'UE comporte deux AFs, l'un portant sur la programmation et l'autre sur la modélisation :

- **AF1** : Être capable de formuler un problème sous forme d'un modèle. Savoir mobiliser les outils numériques adaptés au problème.
- **AF2** : Être capable de développer des programmes structurés et robustes en Python, intégrant des bibliothèques scientifiques, des interfaces graphiques et des visualisations avancées.

L'examen final sera sur table et portera sur des notions vues pendant les séances de TD.

⚠️ N'hésitez pas à **abuser des commentaires** dans vos programmes (en particulier dans l'entête, docstrings). Un rappel sur la documentation python est disponible sur Arche.

⚠️ Enregistrez régulièrement vos programmes avec des noms clairs dans le répertoire dédié à chaque TP.

Préambule sur les outils Python:

Pendant les séances, nous utilisons l'éditeur de code **PyCharm** (voir documentation sur Arche). Cet éditeur propose l'analyse de code et contient un débogueur graphique.

Pour installer Python sur vos machines personnelles, il est conseillé d'utiliser **Anaconda** (voir documentation sur Arche).

Lorsque vous ouvrez PyCharm, il vous est proposé de créer un projet. À ce moment, il faut sélectionner l'interpréteur Python. Choisissez celui installé avec Anaconda, présent sur toutes les machines :
`Custom environment > Select existing > Python > C:\Anaconda\python.exe`

⚠️ Soyez attentif à **ne pas utiliser d'environnement virtuel**.
Note : un environnement virtuel permet de partir d'une installation vierge (afin d'éviter les conflits entre bibliothèques) et d'installer uniquement celles nécessaires au projet.

Il est possible de configurer l'interpréteur par défaut :

- Menu principal : `Customize > All Settings > Python interpreter > System interpreter > C:\Anaconda\python.exe`
- Ou dans un projet : `Fichier > Settings > Python interpreter > System interpreter > C:\Anaconda\python.exe`

Objectifs pédagogiques de cette séance:

- Manipuler des fichiers et des dates, produire des affichages graphiques.
- Réviser les bibliothèques **NumPy, Pandas,** et **Matplotlib**.
- Savoir manipuler des séries temporelles et effectuer des statistiques simples.

- Résoudre une équation linéaire à *N* points de mesure et *M* paramètres inconnus par la méthode des moindres carrés.

Partie 1 – Manipulation de séries temporelles avec NumPy

Le fichier `temperatures_38.25_96.75.txt` contient les températures journalières à 2 m mesurées à 11h et 23h, aux coordonnées 38.25°N – 96.75°E. Ces données proviennent du modèle météorologique **ERA-5** du *European Center for Medium-range Weather Forecasts* (ECMWF) :

👉 https://www.ecmwf.int/

- Ouvrir le fichier texte afin d'identifier la structure du fichier.

Étape 1 – Lecture et extraction des données

- Charger les colonnes 2 et 3 du fichier avec `numpy.loadtxt()` (https://numpy.org/doc/stable/reference/generated/numpy.loadtxt.html). Comme les formats des données sont différents pour chaque colonnes, il est préférable de lire les colonnes séparément avec l'option `usecols`. Explorer l'option `unpack=True` afin d'extraire les colonnes 2 et 3 dans deux vecteurs séparés.

Étape 2 – Statistiques de base

- Calculer et afficher les températures **minimales et maximales** avec 3 décimales (`numpy.nanmin`, `numpy.nanmax`). Utiliser la méthode `format()` pour un affichage clair:

```
print(« La valeur de x est {:.2f} ».format(x))
```

Remarque: Depuis Python3, on peut aussi écrire (méthode `f-strings`):

```
print(f"La valeur de x est {x:.2f} »)
```

`:` introduit le formatage

`.2` signifie qu'il y a 2 décimales

f indique que c'est un nombre flottant (float).

Autre exemples de formats :

`{x:.2e}` → `1.23e+01` Notation scientifique

`{0.256:.1%}` → `25.6%` Pourcentage

- Calculer et afficher la moyenne des températures **supérieures à 0°C** et **inférieures à 0°C**.
- Calculer les moyennes et l'écart type des T° **à 11h**.

Étape 3 – Conversion des dates en objets datetimes

- Le fichier contient les dates au format texte dans la première colonne (chaîne de caractères, ex. "2003-01-01 11:00:00"). Charger la première colonne dans une variable `dates_str`.
- Pour effectuer des opérations sur les dates (trier, filtrer, calculer des moyennes mensuelles, tracer les séries temporelles), il est plus pratique de les convertir en objets datetime de Python. Utiliser la fonction `datetime.strptime()` pour convertir la chaîne de caractère `dates_str` en objet datetime `dates` selon le format `'%Y-%m-%d %H:%M:%S'`.
- La conversion permet ensuite :
  - d'extraire facilement l'heure, le mois, ou l'année d'une date. Afficher par exemple le résultat de la commande: `dates[0].month`

- de sélectionner des périodes spécifiques,
- de tracer des séries temporelles avec des axes de dates lisibles.

Étape 4 – Analyse mensuelle

- Calculer la moyenne **mois par mois** de la série.

Étape 5 – Visualisation

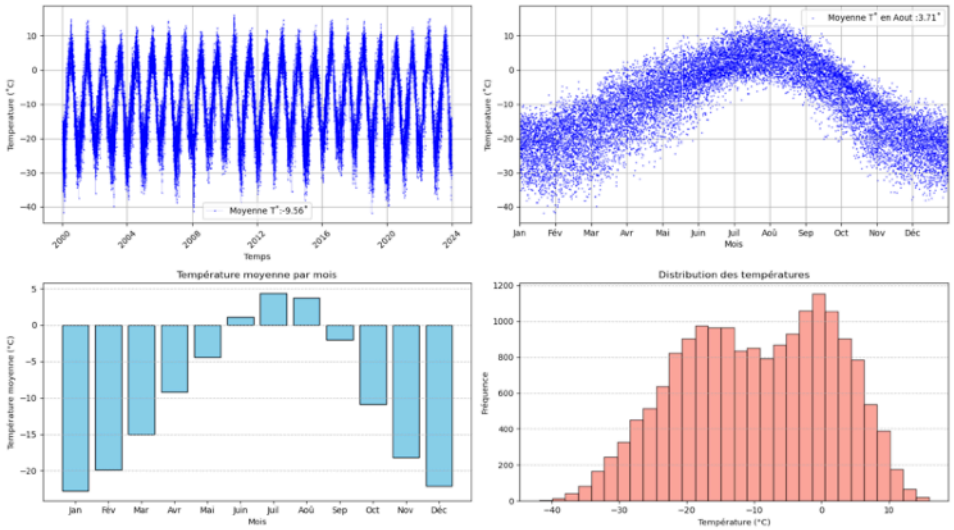
On cherche à représenter différentes vues des températures entre 2000 et 2024.

- Écrire une fonction Python qui prend en entrée les dates et les températures, puis trace la série temporelle. Utiliser `matplotlib.pyplot` :

```
import matplotlib.pyplot as plt
plt.figure()
plt.plot(…)
```

- Ajouter des noms d'axe, un titre, et des grilles. Ajouter une **légende** indiquant la température moyenne sur la période.
- Explorer les options pour améliorer la lisibilité :
  - `plt.xticks(rotation=45)` pour incliner les dates
  - `plt.tight_layout()` pour éviter le chevauchement des labels.
- Représenter les températures sur un **cycle annuel** avec l'opérateur modulo (`numpy.mod()`). L'axe des abscisses correspond aux mois (de 0 à 1), l'axe des ordonnées à la température.
- Représenter les moyennes mensuelles sous forme de **barres,** où l'axe des abscisses correspond aux 12 mois et l'axe des ordonnées à la température moyenne avec `plt.bar()`. Modifier les labels de l'axe de x avec plt.`set_xticklabels(['Jan' , 'Fev ',… ])`
- Tracer un **histogramme** de toutes les valeurs de température de la série avec `plt.hist()`. L'axe des abscisses correspond à la température (°C), l'axe des ordonnées à la fréquence.
- Explorer l'arrangement de plusieurs sous-figures dans une même figure avec: https://matplotlib.org/stable/api/_as_gen/matplotlib.figure.Figure.add_subplot.html#matplotlib.figure.Figure.add_subplot. Essayez d'afficher les **quatre représentations** (série temporelle, cycle annuel, moyennes mensuelles, histogramme) dans une même figure.

Résultats attendus:



Partie 2 – Manipulation de séries temporelles avec Pandas

Introduction à Pandas

Pandas est une bibliothèque Python spécialement conçue pour manipuler et analyser des données tabulaires (e.g. tableur Excel, fichier texte, CSV, etc.). Elle fournit une structure de données très pratique appelée **DataFrame**, qui ressemble à un tableau avec des lignes et des colonnes (Voir document *essentiel Pandas* sur Arche). On trouve sur le web de multiples présentations de cette bibliothèque : https://moncoachdata.com/blog/guide-bibliotheque-pandas/.

Un **DataFrame** permet de :

- Charger facilement des fichiers (CSV, Excel, etc.).
- Manipuler des colonnes par leur nom.
- Effectuer des calculs statistiques en une seule commande.
- Travailler de manière intuitive avec des séries temporelles (dates, heures, etc.).

1. Charger le même fichier avec `data = pandas.read_csv(fichier, skiprows, names=["dates", "dates_decimales", « Temperatures »], parse_dates=['dates']):`
   - `skiprows` permet de sauter des lignes
   - `names` permet de forcer le nom des colonnes du DataFrame, s'ils ne sont pas indiqués dans l'entête.
   - `parse_dates` permet de convertir les chaines de caractère de la premiere colonne en objet `datetime.`
2. Visualiser l'entête du DataFrame: `data.head()`. Visualiser le type de données: `data.dtypes.`
3. Essayer d'affichier la colonne « Tempereratures », la 3ème ligne, ou le 4eme element de la 3ème colonne (Voir document *essentiel Pandas* sur Arche).
4. Visualiser la série en 1 ligne : `data.plot(x="dates", y="temp").`
5. Calculer les statistiques en une ligne :
   - `data["Tempereratures"].mean(), data["Tempereratures"].min(), data["Tempereratures"].max()`
6. Choisir la première colonne comme index du DataFrame en ajoutant l'option `index_col=« dates"` lors de la lecture. ⚠️ Après cette opération d'indexage, la 1ère colonne dates est devenue l'index, elle disparait de la liste des colonnes dans le DataFrame. On y accède désormais via `df.index`. Les autres colonnes restent accessibles normalement par leur nom. Puis:

- `data.resample(« ME »).mean():` Regroupe les données par mois (fin de mois, "ME"), puis calcule la moyenne de chaque mois.
- `data["Tempereratures"].rolling(30).mean():` Crée une moyenne glissante sur 30 jours, utile pour lisser les variations journalières.

Partie 3 – Régressions linéaires

- La **regression linéaire** est une méthode numérique permettant d'ajuster une loi linéaire $y(x_i) = \sum m_k X_k(x_i)$ à un nuage de N points expérimentaux (x_i, y_i). Les M fonctions X_k sont des fonctions arbitraires de x_i servant de base pour l'ajustement.

- On définit **fonction coût** (ou misfit), Q, la fonction qui minimise les données synthétique aux données réelles ou expérimentales:
$$Q = \sum [y_i - y(x_i)]^2$$

- L'objectif est de trouver les M coefficients m qui minimisent cette fonction coût. On peut écrire le problème sous forme matricielle: $d = Gm$

Òu:
- d est le vecteur des N données expérimentales,
- m est le vecteur des M paramètres inconnus,
- G est la matrice reliant les données aux paramètres à inverser.

- Si G est inversible et carré, la solution se calcule directement:
$$\overrightarrow{m} = G^{-1}d$$

- Lorsque G^TG est inversible, la solution dite des **moindres carrés** au problème inverse linéaire s'écrit:
$$\overrightarrow{m} = (G^TG)^{-1}G^Td$$

⚠️ Calculer l'inverse d'une matrice peut être coûteux numériquement. Il est donc recommandé, pour résoudre un système linéaire, d'utiliser la fonction NumPy :

```
import numpy as np
m, residuals, rank, s = np.linalg.lstsq(G, d, rcond=None)
```

Il existe aussi de nombreuses fonctions dans **SciPy** pour résoudre des systèmes inverses non linéaires :https://docs.scipy.org/doc/scipy/tutorial/optimize.html. Mais pour des systèmes linéaires simples, il est inutile d'utiliser des algorithmes trop coûteux.

Application aux séries temporelles

- On peut modéliser les températures journalières par une fonction sinusoïdale combinée à une tendance linéaire : $T(t) = Acos(2\pi t) + Bsin(2\pi t) + Ct + D$, avec:
  - $\sqrt{A^2 + B^2}$: amplitude des températures,
  - $\arctan B/A$: phase correspondant à la date des maxima,
  - C: tendance linéaire (augmentation ou diminution des températures moyennes),
  - D : température moyenne.

- Estimer les coefficients en utilisant un ajustement linéaire (moindres carrés) en partant du code **NumPy**. Pour créer une matrice G vide avec Numpy, il suffit de faire: G = np.zeros((N,M)). De la même manière pour créer une matrice faite que de 1: G = np.ones((N,M)).

- Rajouter la courbe d'optimisation en rouge sur la première figure montrant la séries temporelles complète.
- Déterminer l'**amplitude pic-à-pic** des températures. Rajouter cette information sur la figure dans la légende du modèle avec 1 décimales après la virgule.
- Identifier la **date des maximums de température** (phase). Rajouter cette information sur la figure dans la légende du modèle avec 2 décimales après la virgule.
- Évaluer la **tendance linéaire** sur la période observée. Rajouter cette information sur la figure dans la légende du modèle avec 3 décimales après la virgule.
- Sauvegarder la figure en PDF avec `fig.savefig()`.