Un livre pour faire des livres

Simon Kamerling

Dec 11, 2020

FAIRE UN LIVRE AVEC JUPYTER

Ce tutoriel est en version Béta.

Dans le cadre de mon doctorat, j'ai commencé à utiliser Python pour tracer des courbes. Je me suis donc intéressé à Jupyter, Notebook puis Lab, car l'interface d'un notebook dans le style Mathematica (ou MatLab? Jamais utilisé) m'a toujours semblé très pratique, pour les raisons suivantes:

- exécution immédiate car pas de compilation
- interactivité: changer sa ligne de code et réappuyer sur maj + entrée
- pratique pour présenter, écriture en Latex, courbes, ..

J'ai ensuite découvert d'autres fonctionnalités qui m'ont fait penser que ce système serait très intéressant pour donner des cours. Le plus évident serait des cours de Python, néanmoins il est tout à fait possible d'écrire des livres en pdf ou html. Les versions html correspondent à un site web, comme peut-être là où vous lisez ce tutoriel. Dans la version html, il est possible d'ajouter des boutons interactif envoyant sur GitHub ou sur Binder. Binder est un serveur qui fait tourner vos notebook à partir de votre dossier GitHub: cela donne accès de manière interactive à votre livre/code en **lecture seule.**

Du code pour tracer des courbes est également disponible, comme exemple de ce qu'il est possible de faire. J'ai personellement choisi de m'orienter vers Plotly et Cufflinks avec l'usage de bouton interactif (changer le jour, les variables explorées, …). Cela permet de faire une analyse de données facile. Les données présentes sont issus d'une simulation et ne sont pas privées.

La première partie de ce tutoriel explique comment faire des livres. Cela est principalement un résumé de ce que j'ai compris, et n'est pas exempt d'erreurs. Par ailleurs, ces logiciels sont en grande évolution; de nombreuses personnes travaillent à faire ces interfaces plus maniables, et je pense que ces outils peuvent devenir une norme.

La seconde partie explique comment tracer des courbes et des exemples d'interactivité sont disponibles dans les NoteBooks.

CHAPTER

ONE

TUTORIEL PYTHON & NOTEBOOK

Un cours en **français** est présent à l'adresse suivante: https://python.sdv.univ-paris-diderot.fr/

Le chapitre 18 introduit à l'utilisation des Notebook et de JupyterLab. A mon avis, l'auteur a utilisé Jupyter Book pour écrire son cours.

L'intérêt des Notebook réside dans les cellules et l'absence de compilation. L'utilisation de Latex dans les markdown permet de faire de beaux documents avec de belles formules.

De plus, l'ouverture dans un navigateur internet est un vrai plus. Je mets ici un tuto pour une utilisation rapide, si vous connaissez un peu de Python.

1.1 Etape 1: Installer Anaconda

Tout est dans le titre, à chercher sur Qwant ou DuckDuckGo.

1.2 Etape 2 : Lancer Jupyter

Ouvrir Anaconda et cliquer sur jupyter.

Il est possible de l'ouvrir depuis l'invite de commande (Anaconda Prompt Shell) si vous n'avez pas envie de lancer le launcher d'Anaconda. La commande:

- jupyter-notebook pour le notebook
- jupyter lab pour Jupyter Lab

1.3 Etape 3 : Jouer avec Python et Jupyter

Exo 1 : Pour n entier naturel non nul, on pose $H_n = \sum_{k=1}^{n} \frac{1}{k}$ (série harmonique).

1. Montrer que : $\forall n \in \mathbb{N}, \ln(n+1) < H_n < 1 + \ln(n)$ et en déduire la limite en $+\infty$ de H_n.
2. Pour n entier naturel non nul, on pose $u_n = H_n - ln(n)$ et $v_n = H_n - ln(n+1)$. Montrer que les suites (u_n) et (v_n) convergent vers un réel γ.

```
# Import pour plotter de manière facile
import pandas as pd
from ipywidgets import interact, interactive, fixed, interact_manual, IntSlider
# Standard plotly imports
```

(continues on next page)

(continued from previous page)

```
import chart_studio.plotly as py
import plotly.graph_objs as go
from plotly.offline import iplot, init_notebook_mode
# Using plotly + cufflinks in offline mode
import cufflinks as cf
cf.go_offline(connected=False)
init_notebook_mode(connected=False)
```

1.4 Etape 4: Exporter votre NoteBook au format pdf ou html

Cela est possible en allant dans le menu > File > Export noteBook as..

1.4.1 Discussion: Avis personnel sur Lab vs notebook

Lab me semble plus pratique que NoteBook, j'ai commencé avec directement. En fait, Lab est un Wrapper autour de Notebook: on peut utiliser tout le Notebook dans Lab. Ce qui fait que c'est plus compliqué, forcément. Mais on a plus d'options, comme ouvrir une console directement dans le notebook, se déplacer plus facilement dans les fichiers, écrire des méta données dans des cellules, ..

Néanmoins, j'ai pu avoir des soucis lors de l'exportation au format pdf, que j'avais beaucoup moins avec le NoteBook.

1.5 JupyterLab: utilisation pratique

1.5.1 0. Se déplacer facilement

Il est possible de se déplacer au clavier entre les cellules, ce qui est très pratique. Probablement possible avec le notebook aussi.

Si vous êtes dans une cellule, appuyez sur échap pour passer en mode "commande": vous pourrez alors:

- vous déplacer via les flèches
- ajouter une cellule avant votre position avec a (pour avant) et b après votre position (pas de moyen mnémotechnique)
- copier une cellule avec c et coller avec v (pas de touche ctrl)
- entrer dans une cellule avec enter
- modifier le type de la cellule: y -> code, m -> markdown

1.5.2 1. Ouvrir une console

Clic droit sur l'onglet du notebook > new console for notebook.

Pratique pour ne pas salir son notebook, se déplacer et tester des choses. Je ne l'utilise pas tant que ça car il est possible d'utiliser les commandes linux (pwd, cd) directement dans le notebook (pour ouvrir des fichiers par exemple).

CHAPTER

TWO

FAIRE UN LIVRE AVEC JUPYTER BOOK

Je compte présenter ici ma manière de faire, pas donner un cours: c'est encore brouillon dans ma tête.

Leur website est très bien fait: https://jupyterbook.org/start/overview.html

Le concept est de créer un livre de notebooks, en générant chaque page et en les ajoutant les uns avec les autres. Cela correspond à une architecture de dossiers/fichiers. Certaines choses peuvent se révéler compliqués lorsque l'on exporte au format pdf, particulièrement au niveau des figures. Le fichier _toc.yml référence l'architecture.

Il y a beaucoup d'options intéressantes: mettre des liens pour Binder, pour Github, qui permettent un accès rapide à l'interactivité des Notebook

Pour contruire le livre, sous Windows: ouvrir une invite de commande conda, entrer l'instruction suivante afin d'ouvrir l'environnement windows/conda adapté et décrit dans environment_win.yml

conda env create -f environment_win.yml

conda activate wintest

Une fois l'environnement créé, il suffit de taper conda activate wintest afin de réouvrir, dans une autre console par exemple.

Cela ouvre l'environnement conda et donne un nouvel aspect à la console. Il suffit alors d'entrer la commande:

- jupyter-book build Outils/ ——> pour une version html
- jupyter-book build Outils/ –builder pdflatex ———-> pour une version pdf

CHAPTER

THREE

EXPLICATIONS

3.1 1. Utilisation d'un notebook pour plotter via pandas, plotly et cufflinks

Prérequis: Anaconda 3

Entrer dans la console (Anaconda Prompt Shell) les instructions suivantes: conda install ** avec ** les packages suivants:

- cufflinks
- pandas
- plotly
- ipywidgets

Possibilité de créer des .html facilement, .pdf, càd une page du notebook, faire tourner quelques plots pour un export facile, ..

Ouvrir un fichier .ipynb dans le dossier pour un exemple.

Il est bien sur possible d'utiliser de manière plus classique matplotlib, ou tout autre package.

3.2 2. Utilisation de Jupyter Lab

1. La poire
2. La pomme

Les différences JupyterLab/Notebook: le notebook est plus simple d'utilisation au début, mais a une interface moins pratique une fois qu'on a compris comment marche les notebooks. Entre autre, pour l'édition de livres, il est facile de changer les tags des cellules afin de choisir si elle s'affiche dans le rendu final (.html, .pdf).

Le désavantage: des fois besoin de fouiller un peu plus pour pouvoir faire facilement des exportations, etc.

Pour utilisation de tout cela dans JupyterLab (mais pas Notebook), il faut rajouter des choses à Jupyter Lab. Les packages suivants servent à relier

- conda install jupyterlab "ipywidgets=7.5"
- conda install nodejs
- jupyter labextension install jupyterlab-plotly@4.13.0
- jupyter labextension install @jupyter-widgets/jupyterlab-manager plotlywidget@4.13.0

Commande à mettre dans la console pour conversion en html, en enlevant les cellules avec tags remove_cell et avec remove_input, en étant dans le bon dossier et en remplaçant Plotting par le nom du notebook:

jupyter nbconvert Plotting.ipynb –to=html –TagRemovePreprocessor.remove_input_tags="{'remove_input','remove_cell'}" –TagRemovePreprocessor.remove_single_output_tags="{'remove_cell'}"

Ceci marche si nbconvert est bien configuré. Leur website aide bien (connexion de latex, etc.). Résoudre cela permet d'exporter au format latex puis pdf des notebooks par la commande ci-dessus, en ayant tagué les cellules dans JupyterLab.

Le jfidkp^qfos

$X_i = 10$

```
import plotFuncAndClass as hm
import pandas as pd
from ipywidgets import interact, interactive, fixed, interact_manual, IntSlider
# Standard plotly imports
import chart_studio.plotly as py
import plotly.graph_objs as go
from plotly.offline import iplot, init_notebook_mode
# Using plotly + cufflinks in offline mode
import cufflinks as cf
cf.go_offline(connected=False)
init_notebook_mode(connected=False)
```

```
fig = go.Figure(data=[go.Table(
    header=dict(values=list(dfTest.columns),
                fill_color='paleturquoise',
                align='left'),
    cells=dict(values=dfTest.transpose().values.tolist(),
               fill_color='lavender',
               align='left'))
])

fig.show()
```

```
import plotly.io as pio
pio.renderers.default = 'jupyterlab'
```

CHAPTER

FOUR

EXEMPLES DE PLOT POSSIBLE AVEC CUFFLINKS ET PANDAS

La méthode iplot() que Cufflinks amène au dataframe de Pandas (un wrapper autour de pandas?) permet de tracer facilement différentes courbes. Ci-dessous différents exemples.

```
import plotFuncAndClass as hm
import pandas as pd
from ipywidgets import interact, interactive, fixed, interact_manual, IntSlider
# Standard plotly imports
import chart_studio.plotly as py
import plotly.graph_objs as go
from plotly.offline import iplot, init_notebook_mode
# Using plotly + cufflinks in offline mode
import cufflinks as cf
cf.go_offline(connected=False)
init_notebook_mode(connected=False)
```

```
listLat=['\Delta t','DNI', 'T_{amb}', '\dot{m}_{Demand}', 'T_{demand}', '\dot{m}_{SF}
↪', 'T_{SF}', '\dot{m}_{aux}',
    '\dot{q}_{boiler}', '\dot{q}_{abs}','M_{storage}', 'T_{storage}', '\dot{m}_
↪{Storage -> boiler}', '\dot{m}_{SF->Boiler}', '\dot{m}_{SF->Storage}']
varDescription=['Hour','Direct Normal Irradiance', 'Ambient Temperature', 'Demand␣
↪Mass Flow Rate', 'Temperature Demand',
                'Mass flow rate from the Solar Field', 'Temperature of the Solar Field
↪', 'Auxiliar Mass flow rate',
                'Heat From The Boiler', 'Heat absorbed by the receiver','Mass in the␣
↪Storage', 'Temperature of the storage', 'Mass flow rate from the Storage to the␣
↪Boiler',
                'Mass flow rate from the Solar Field to the Boiler', 'Mass flow rate␣
↪from the Solar Field to the Storage']
df = pd.read_csv('data/CoSim.csv') #read a specific csv file
dfMILP= pd.read_csv('data/MILP.csv') #read another specific csv file with same fields
dfSM1= pd.read_csv('data/SM1.csv') #read another specific csv file with same fields
dfSM2= pd.read_csv('data/SM2.csv') #read another specific csv file with same fields
tabDataFrames=[df,dfSM1,dfSM2, dfMILP]
tabNameDataFrame=['CoSim','SM1','SM2','MILP','3 CoSim']
nameColumns=list(df.columns)
tabVar=[hm.nameAndUnit(nameColumns[i],nameColumns[i],varDescription[i],'',listLat[i])␣
↪for i in range(0,len(nameColumns))]
```

4.1 Plot of Daily average values

```
dfDaily = pd.read_csv('data/CoSimDaily.csv') #read a specific csv file
dfMILPDaily= pd.read_csv('data/MILPDaily.csv') #read another specific csv file with␣
↪same fields
dfSM1Daily= pd.read_csv('data/SM1Daily.csv') #read another specific csv file with␣
↪same fields
dfSM2Daily= pd.read_csv('data/SM2Daily.csv') #read another specific csv file with␣
↪same fields
tabDaily=[dfDaily, dfSM1Daily, dfSM2Daily, dfMILPDaily]
nameColumnsDay=list(dfDaily.columns)
tabVarDay=[hm.nameAndUnit(nameColumnsDay[i],nameColumnsDay[i],varDescription[i],'',
↪listLat[i]) for i in range(0,len(nameColumnsDay))]
```

```
def plot1VarYear(ControlModel, var1):
    titleFunc= var1.varDescription + ' in ' + var1.unitVar
    if(ControlModel<4):
        toPlot=tabDaily[ControlModel]
        toPlot['date']=toPlot['day(d)'].map(hm.fromNumDayToDate)
        toPlot=toPlot.set_index('date')
        toPlot.iplot(mode='markers',y=var1.keyDF , title=titleFunc, xTitle='day (d)',␣
↪symbol='square', size=4,
                     yTitle='$'+var1.lat+'('+var1.unitVar+')$',showlegend=True)
    else:
        toPlot=pd.concat([tabDaily[i].add_suffix('_'+tabNameDataFrame[i]) for i in␣
↪range(0,3)], axis=1)
        toPlot['date']=toPlot['day(d)_CoSim'].map(hm.fromNumDayToDate)
        toPlot=toPlot.set_index('date')
        toPlot[[var1.keyDF+'_'+tabNameDataFrame[i] for i in range(0,3)]].iplot(mode=
↪'markers', symbol='square', size=4,
                                    title=titleFunc, xTitle='day (d)', yTitle='$
↪'+var1.lat+'('+var1.unitVar+')$',showlegend=True)
```

```
plot1VarYear(4,tabVarDay[6])
```

4.2 HeatMaps

```
heatMapFromVarName(tabVar[1],1)
```

4.3 Matrice de corrélation

```
def plotRepart(var):
    toPlot=pd.concat([tabDaily[i][var.keyDF] for i in range(0,4)], axis=1)
    toPlot.columns=tabNameDataFrame[0:-1]
    toPlot.iplot(kind='box',xTitle="Control Model", yTitle = var.unitVar,
                 title=var.varDescription + ' average repartition trough the year of␣
↪daily values', showlegend=False)
```

```
plotRepart(tabVarDay[5])
```

CHAPTER

FIVE

DÉCRIRE L'EMPLACEMENT DU SOLEIL

Pour décrire l'emplacement du soleil, deux systèmes de coordonnées semblent très pratique: le système de coordonnées **horizontales**, qui pose un repère local; et le système de coordonnées **équatoriales**, qui se place dans un repère héliocentrée.

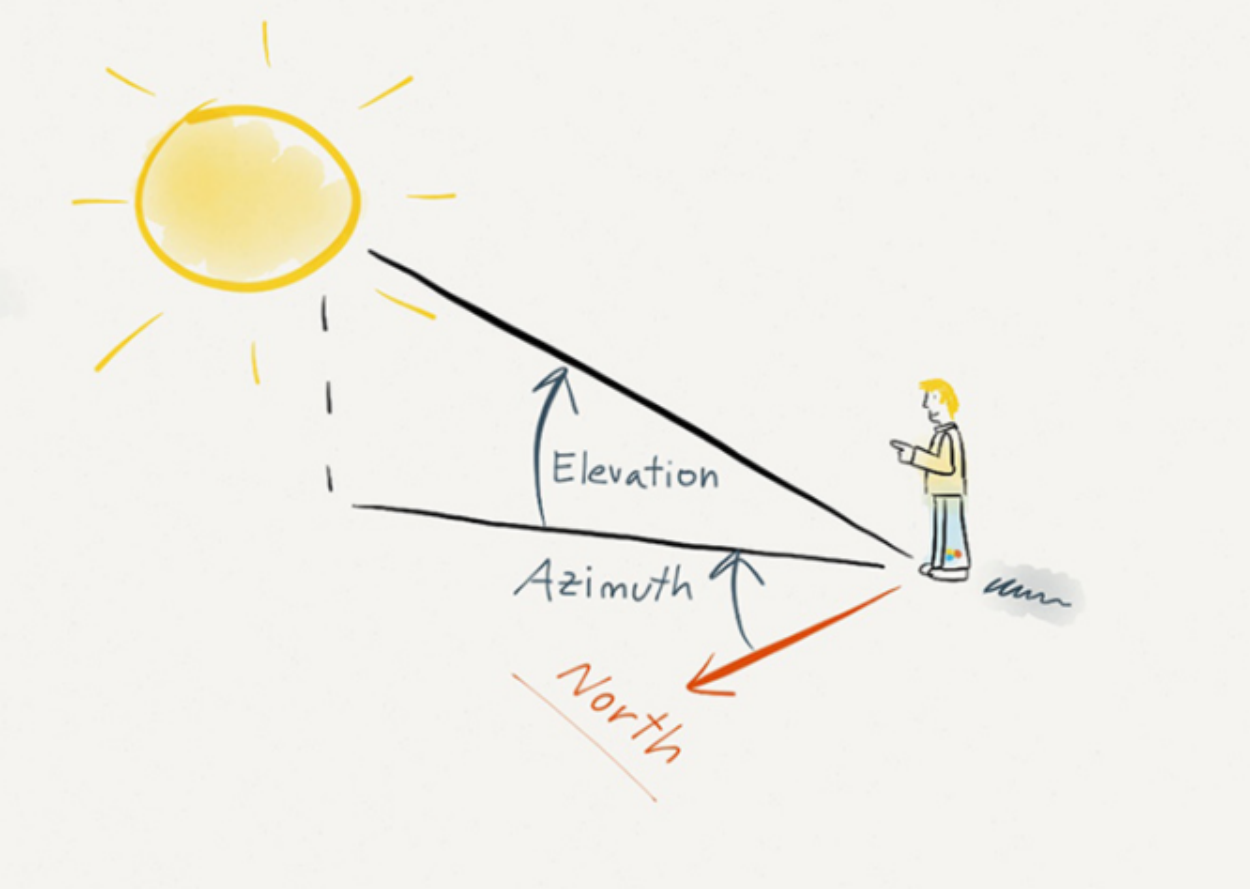
5.1 Azimuth et Elevation - Horizontale

Note

D'autres bases orthonormées directes sont possibles, comme par exemple Ox orienté à l'ouest et Oy au sud. L'azimuth est alors négatif le matin, entre autres. La définition de l'azimuth est également variable, se méfier.

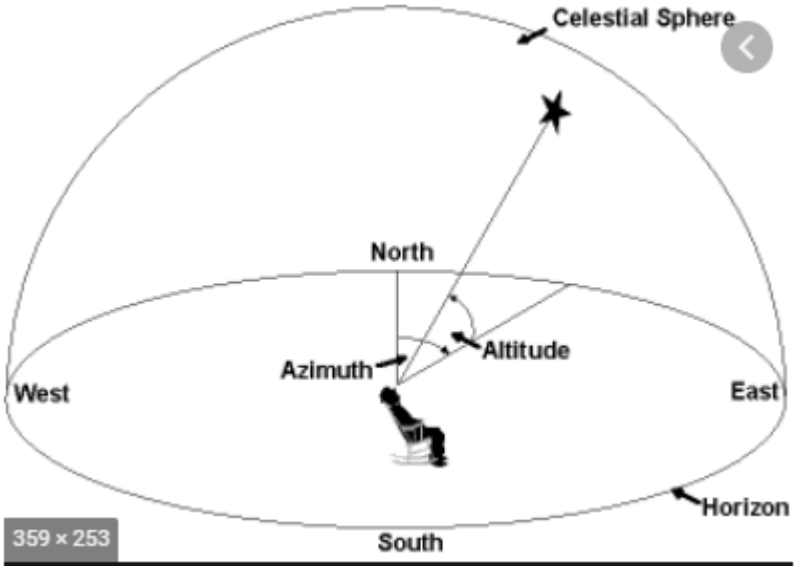
On se place dans un repère (0,x,y,z) orthonormé direct, avec x pointant vers l'est, y vers le nord et z à la verticale. Notons $\vec{s}$ le vecteur solaire, càd un vecteur de norme 1 pointant vers le soleil.

Comme le soleil se déplace à distance fixe (plus ou moins) de la Terre, il ne se déplace qu'en deux dimensions sur la "sphère céleste. Il est donc possible de décrire sa position avec deux angles, il est possible de décrire sa position avec 2 angles: l'azimuth et l'élévation.



Plus d'informations: Wikipédia

5.1.1 Azimuth - α

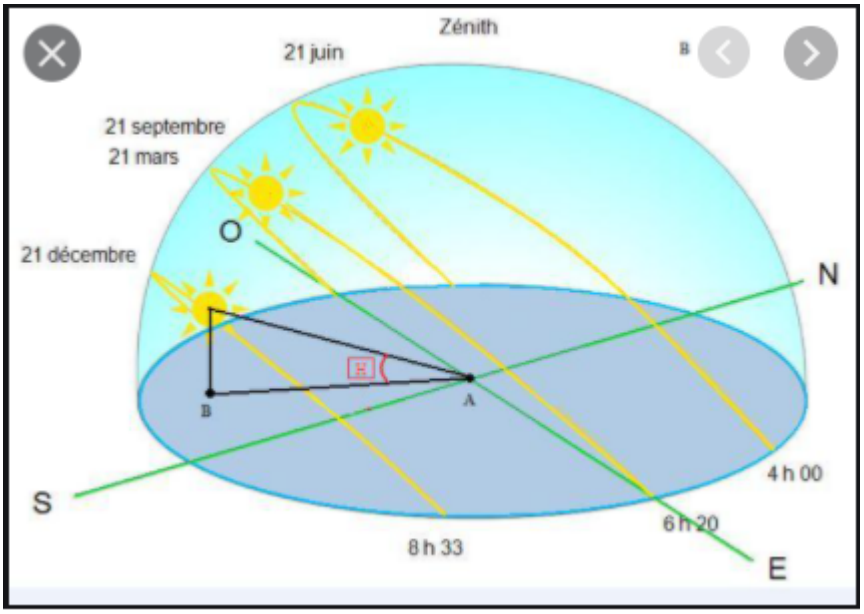


Cet angle correspond à la direction du soleil dans le plan horizontale; 0° correspond au Nord, 90° à l'est, etc.

On peut le définir comme l'angle entre

1. Le projeté du vecteur solaire $\vec{s}$ dans le plan (0,x,y)
2. Le vecteur dirigé au nord ($\vec{Oy}$)

5.1.2 Elevation - γ



L'élevation (γ) est l'angle formé entre le soleil et la ligne de l'horizon.

5.1.3 Vecteur solaire

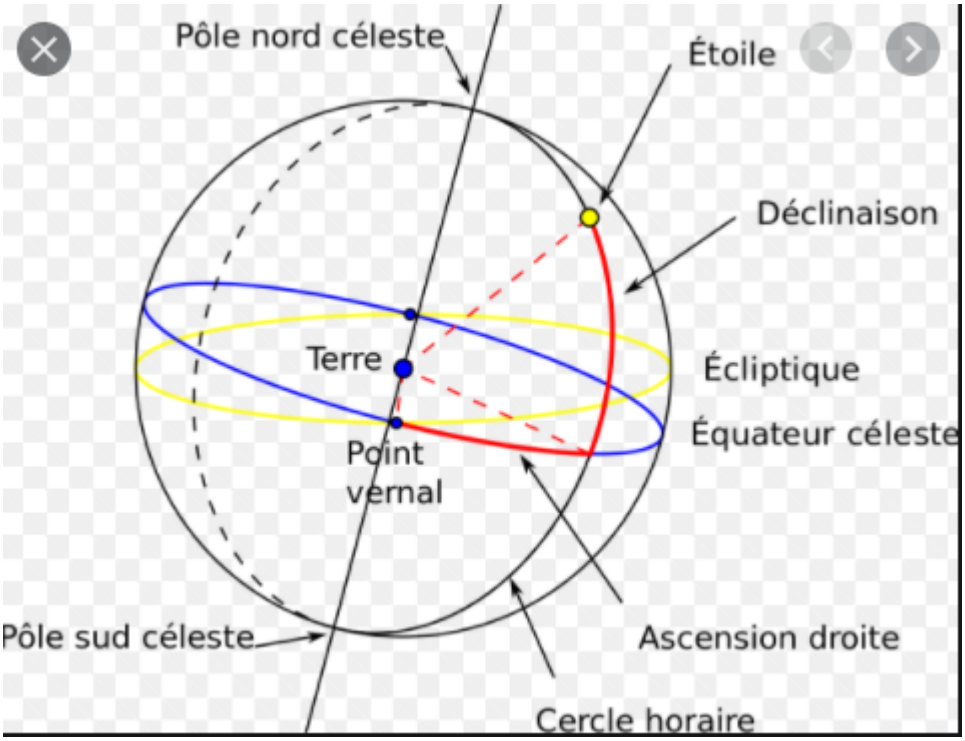
Le vecteur solaire est un vecteur pointant vers le soleil. Sans cette base et pour cette définition de l'azimuth, il s'exprime comme $\$\vec{s} = \sin(\alpha) \times \cos(\gamma)\vec{u_x} + \sin(\alpha) \times \sin(\gamma)\vec{u_y} + \sin(\gamma)\vec{u_z}\$$

```
class solarVector:
    def __init__(self,x,y,z):
        self.x=x
        self.y=y
        self.z=z
    def printVec(self):
        print("x--->"+str(self.x)+"\n y--->" +str(self.y)+"\n z---> "+str(self.z))

def solarVectorFromAzimElev(azim,elev):
    sx = sin(azim) * cos(elev);
    sy = cos(azim) * sin(elev);
    sz = sin(elev);
    return solarVector(sx,sy,sz);
```

5.2 Latitude, déclinaison et angle horaire - Systèmes Equatoriales

Ce repère correspond à un repère héliocentré. Néanmoins, le vecteur solaire est donné dans la base locale d'un observateur situé à la latitude ϕ et au temps t. Plus d'informations: Wikipédia



5.2.1 Déclinaison

La déclinaison (δ) correspond à l'inclinaison de l'axe de la terre. Ce dernier varie de -23.4° au solstice d'hiver à 23.4° au solstice d'été. Cet angle peut être relié au jour de l'année via de nombreuses corrélations, la plus reconnue étant celle de l'algorithme PSA. En voici une très proche.

```
import pandas as pd
from ipywidgets import interact, interactive, fixed, interact_manual, IntSlider
# Standard plotly imports
import chart_studio.plotly as py
import plotly.graph_objs as go
from plotly.offline import iplot, init_notebook_mode
# Using plotly + cufflinks in offline mode
import cufflinks as cf
cf.go_offline(connected=False)
init_notebook_mode(connected=False)
```

```
from math import cos, sin
def solarDeclinationByDayNumber(day):
    Pi = 3.14159;
    omega = (2 * Pi) / 365 * (day - 1.);
    return 0.006918 - 0.399912 * cos(omega) - 0.006758 * cos(2 * omega) + 0.000907 *␣
↪sin(2 * omega) - 0.002697 * cos(3 * omega) + 0.00148 * sin(3 * omega);
```

(continues on next page)

(continued from previous page)

```
xDays=[k for k in range(0,365)]
deltaDays=[solarDeclinationByDayNumber(k) for k in range(0,365)]
dfYear=pd.DataFrame([xDays,deltaDays]).T
dfYear.columns=["day","declination"]
dfYear.iplot(kind='scatter',x="day",y="declination", title="Déclinaison solaire␣
↪durant l'année", xTitle="day", yTitle=" delta (rad)")
```

5.2.2 Angle horaire

L'angle horaire (ω) est défini par la rotation de la Terre par rapport au midi heure solaire. Il vaut 0 à midi, est positif dans l'après-midi et négatif le matin.

La terre faisant 360° en 24 heures, la Terre décrit $\frac{360°}{24h} = 15°/h$. Ce qui peut s'écrire $\frac{d\omega}{dt} = 15°/h$ avec t le temps en heure. D'où, après intégration : $\omega = (t-12) * 15° = \frac{\pi}{180} \times 15 \times (t-12)\ rad$

Dans cette équation, le temps est en UTC ou heure solaire. En France, (pour l'instant?), l'heure d'été est UTC+2 et hiver UTC+1, c'est-à-dire qu'en hiver le soleil est au sud à 13h et en été à 14h.

```
from math import pi
dfDay=pd.DataFrame(data= [[k for k in range(0,24)],[(k-12)*15*pi/180 for k in range(0,
↪24)]]).T
dfDay.columns = ["hour","omega"]
dfDay.iplot(x='hour',y='omega', title = "Angle horaire sur 24 h", xTitle="hour",␣
↪yTitle="Omega (rad)")
```

L'heure maximale ω_{max}, càd l'heure à laquelle le soleil se couche, s'écrit $\omega_{max} = \arccos(-\tan(\phi) \times \tan(\delta))$

```
from math import acos, tan, pi
def omegaMax(day,lat):
    return acos(-tan(lat)*tan(solarDeclinationByDayNumber(day)))
```

```
phi=45.564601 * 3.14159/180;
dfYear['omega_max']=[omegaMax(k,phi) for k in range(0,365)]
dfYear['derivHourMax']=[(omegaMax(k+1,phi)-omegaMax(k,phi))*180/(3.14159*15)*60 for k␣
↪in range(0,365)]
dfYear['hourMax']=[12+dfYear['omega_max'][k]*180/(3.14159*15) for k in range(0,365)]
dfYear.iplot(x="day",y="hourMax",secondary_y="derivHourMax",
             title="Heure de coucher du soleil et sa dérivée au cours de l'année",
            xTitle="day", yTitle="Heure de coucher (h)", secondary_y_title="Dérivée␣
↪(minutes/day)")
```

5.2.3 Vecteur solaire

Idée de démonstrations: produit de matrices de rotation pour arriver sur le bon repère: Base 0 -> Géocentrée, avec un axe vers le soleil et un z sur la moyenne de l'axe terrestre

Le vecteur solaire dans la base locale à l'observateur s'écrit alors: $$ \vec{s} = - \cos(\delta) \times \sin(\omega) \vec{u_x}\

- (\sin(\delta) \times \cos(\phi) - \cos(\delta) \times \cos(\omega) \times \sin(\phi)) \times \vec{u_y}\
- (\sin(\delta) \times \sin(\phi) + \cos(\delta) \times \cos(\omega) \times \cos(\phi)) \times \vec{u_z} $$

```
def solarVectorFromDeclinHour(declination,hourAngle,lat):
    sx = -cos(declination) * sin(hourAngle);
    sy = sin(declination) * cos(lat) - cos(declination) * sin(lat) * cos(hourAngle);
    sz = sin(declination) * sin(lat) + cos(declination) * cos(lat) * cos(hourAngle);
    return solarVector(sx,sy,sz)
```

5.3 Passer d'un système de coordonnées à l'autre

L'idée est de passer d'un système à l'autre via le vecteur solaire. Les deux formules donnant le vecteur solaire dans le même repère, il devient facile de les inverser. Les algorithmes suivant ont été vérifiés mais on n'est jamais ampte de se planter..

5.3.1 Du vecteur solaire à l'azimuth et l'élévation

```
from math import asin, cos, pi
def azimElevFromSolarVector(s):
    elev = asin(s.z);
    interm = asin(s.x / cos(elev)-0.00000000000002);
    if (s.y>0):
        if (s.x > 0):
            azim = interm;
        else:
            azim = 2 * pi + interm;
    else:
        azim = pi - interm;
    return (azim,elev)
```

5.3.2 Du vecteur solaire à la déclinaison et l'angle horaire

```
from math import asin, cos, sin, tan
def declinHourFromSolarVector(s,lat):
    declin = asin((s.y + s.z * tan(lat)) / (cos(lat) + sin(lat) * tan(lat)));
    interm = -asin(s.x / cos(declin));
    test = (sin(declin) * cos(lat) - s.y) / (cos(declin) * sin(lat));
    if (abs(cos(interm) - test) < 0.001):
        hour = interm;
    elif (s.x < 0):
        hour = Pi - interm;
    else:
```

(continues on next page)

(continued from previous page)

```
        hour = -Pi - interm;
    return(declin, hour)
```

5.3.3 D'un système à l'autre

```
def azimElevFromDeclinHour(declin,hour,lat):
    s=solarVectorFromDeclinHour(declin,hour,lat)
    return azimElevFromSolarVector(s)
```

```
def declinHourFromAzimElev(azim,elev,lat):
    s=solarVectorFromAzimElev(azim,elev)
    return declinHourFromSolarVector(s,lat)
```

5.4 Tracés annuels

A l'aide de la déclinaison, de l'heure et de la latitude, il nous est désormais possible de trouver l'azimuth et l'élévation. Ce système de coordonnées locales est le plus utilisé lorsqu'il s'agit de réfléchir à un système solaire.

5.4.1 Azimuth et élévation

```
from math import pi
day=150
degree=pi/180;
dfDay=pd.DataFrame(data=[[k for k in range(0,24)],[(k-12)*15*pi/180 for k in range(0,
↪24)]]).T;
declin=solarDeclinationByDayNumber(day);
hourAngleMax=omegaMax(day,phi);
dfDay.columns = ["hour","omega"];
azimElev=[azimElevFromDeclinHour(declin,dfDay['omega'][k],phi) for k in range(0,24)];
dfDay['azimuth']=[azimElev[k][0]/degree for k in range(0,24)];
dfDay['elevation']=[azimElev[k][1]/degree for k in range(0,24)];
dfDay.iplot(kind='scatter',mode='markers',size=10,symbol='x', x="azimuth", y=
↪"elevation",
            title = "Azimuth vs Elevation during a summer day", xTitle= "Azimuth (°)",
↪ yTitle="Elevation (°)", text="hour")
```

Lorsque l'élévation est négative, cela correspond aux heures à laquelle le soleil ne s'est pas encore levé.

CHAPTER

SIX

L'IRRADIATION SOLAIRE

Il est possible, d'un point de vue énergétique, de catégoriser l'énergie disponible sur Terre via trois origines: la Terre, le Soleil et la Lune. En effet, du soleil vient:

- la biomasse (photosynthèse)
- les énergies fossiles (photosynthèse puis raffinement)
- le vent (différences de températures, cellules convectives, etc.)
- l'hydroélectricité (cycle de l'eau, évaporation)

La Terre donne l'énergie géothermique, càd de l'énergie nucléaire issu de son noyau en fusion. Etant doné que l'uranium est présent sur Terre, on pourrait dire que l'énergie nucléaire par fission est d'origine terrestre. Alors que l'énergie nucléaire par fusion viendrait du big bang?

La Lune, ainsi que le Soleil, génère des forces de gravité. Ce qui fait bouger les océans, via les marées, d'où l'énergie marémotrice.

L'irradiance solaire correspond à la puissance envoyée par le soleil, souvent exprimée en W/m^2. Le soleil envoie une énergie considérable sur Terre, mais peu dense: en effet, une approximation rapide au niveau du sol donnerait 1000 W/m^2, pour un beau jour d'été en France. Cette énergie thermique (rayonnement) est récupéré par les différents éléments terrestres puis condensée, stockée, .. Ce qui permet de la récupérer en brûlant du bois, par exemple, ou en stockant l'eau en haut des montagnes. L'énergie hydroéléctrique est bien plus dense que l'énergie solaire: en récupérant l'eau et en la mettant dans un barrage, en grimpant en hauteur, on peut stocker cette énergie.

L'irradiation solaire correspond à l'énergie arrivant sur une surface; elle est exprimée en Wh/m^2, c'est par définition l'intégrale au cours du temps de la puissance, càd l'irradiance.

6.1 L'absorption et le spectre solaire

Un corps noir émet une puissance rayonnante, proportionelle à sa température: σT^4. σ est la constante de Boltzmann, valant $5.67 \times 10^{-8}\ W/K^4$

Plus précisément, un corps noir émet une énergie par longueur d'onde; plus précisément, la formule de Planck nous donne cette répartition par longueur d'onde: $$\phi_\lambda = \frac{2\pi \times h \times c_0}{\lambda^5} \times \frac{1}{exp(\frac{hc_0}{\lambda k_B T})-1}$$

```
from math import pi,exp
def Planck(wave,temperature):
    h=6.62*pow(10,-34)
    c0 = 2.99 * pow(10,8)
    kB=1.380649 * pow(10,-23)
    lambd = wave*pow(10,-9) #à donner en nanomètres
    coef1= 2*pi*h*c0/pow(lambd,5)
```

(continues on next page)

(continued from previous page)

```
    coef2= h*c0/(lambd*kB*temperature)
    return coef1/(exp(coef2)-1)
```

```
Planck(500,5900)
```

```
310904.94689185283
```

CHAPTER

SEVEN

BASSES TEMPÉRATURES

CHAPTER

EIGHT

LA CONCENTRATION PONCTUELLE

CHAPTER

NINE

TEMPÉRATURE DE PAROI DANS UNE TOUR

9.1 Sur une formule générale

9.1.1 A un instant t donné

$$\dot{m} \times C_p \times \frac{dT}{dz} = \alpha \times \eta_{opt} \times DNI \times \frac{A_{col}}{L_{rec}} - h \times l_{rec} \times (T_p - T_{amb}) - \epsilon(T_p) \times \sigma \times l_{rec} \times (T_p^4 - T_{amb}^4) \quad (9.1)$$

La relation entre la température du fluide en z et la température de paroi externe peut être résolue en faisant un bilan sur un élément de la paroi du récepteur: $\dot{q}_{SF}(z) = R_{eq} \times (T_p - T_{fl}(z)) + h_a \times (T_p - T_{amb}) + \epsilon \times \sigma \times (T_p^4 - T_{amb}^4)$

Avec $\frac{1}{Req} = \frac{e}{\lambda} + \frac{1}{h_{fl}}$

Ce qui peut s'écrire comme une équation de degré 4 sur T_p: $T_p^4 + q \times T_p - r = 0$
$q = \frac{R_{eq}+h}{\epsilon \times \sigma}$
$r = T_{amb}^4 + \frac{\dot{q}_{SF}(z)+R_{eq} \times T+h \times T_{amb}}{\epsilon \times \sigma}$

Cette équation peut se résoudre de manière littérale en utilisant le fait que les coefficients des carrés et cubes sont nuls. Résultante cubique: $R(z) = z^3 + 4 \times r \times z - q^2$
$\Delta_1 = q^4 + \frac{4}{27} \times (4r)^3$

Or $z_1 = \sqrt[3]{\frac{q^2-\sqrt{\Delta_1}}{2}} + \sqrt[3]{\frac{q^2+\sqrt{\Delta_1}}{2}}$
$z_2 = j \times \sqrt[3]{\frac{q^2-\sqrt{\Delta_1}}{2}} + \bar{j} \times \sqrt[3]{\frac{q^2+\sqrt{\Delta_1}}{2}}$
$z_3 = j^2 \times \sqrt[3]{\frac{q^2-\sqrt{\Delta_1}}{2}} + \bar{j}^2 \times \sqrt[3]{\frac{q^2+\sqrt{\Delta_1}}{2}}$

```
eps = 0.9
sigma = 5.67 * pow(10,-8)
Tamb=20+273.15
Tsky=273.15
qdotSF=0.9*0.75*1000*800 #alpha*eta_opt*C*DNI
Req= 100*(400/0.1)/(100+4000)
Tfluid=600+273.15
h=10
q=(Req+h)/(eps*sigma)
r=(eps*sigma*pow(Tamb,4)+qdotSF+Req*Tfluid+h*Tamb)/(eps*sigma)
ratio=pow(r,3)/pow(q,4)
nu=4*pow(12,3)/pow(27,2)*ratio
delta0=-12*r
delta1=-27*pow(q,2)
nu2 = -4 *pow(delta0,3)/pow(delta1,2)
```

```
C=pow((1+sqrt(1+nu))/2,1/3)*pow(-delta1,1/3)*(-1)
```

```
C
```

```
-15919223.393448392
```

```
import cmath
j=-1/2+1j*sqrt(3)/2
x0=-1/3*(C+delta0/C)
x1=-1/3*(C*j+delta0/(j*C))
x2=-1/3*(C*pow(j,2)+delta0/(pow(j,2)*C))
```

```
sq0=cmath.sqrt(x0)
sq1=cmath.sqrt(x1)
sq2=cmath.sqrt(x2)
```

```
Tp=1/2*(sq0+sq1+sq2)
```

```
1/2*(sq0-sq1-sq2)
```

```
(-1517.2264175955538+0j)
```

```
from math import sqrt
sqrt(1+nu)-(1+nu)
```

```
-0.7977724087046312
```

D'où : $\$C = \sqrt[3]{\Delta_1} \times \nu^{1/6}\$$

CHAPTER

TEN

T(Z) DANS UN PANNEAU

Plus précisément, on a en général la définition: $\dot{m} \times C_p \times (T_{out} - T_{in}) = \eta_0 \times A \times I - a_1 * A * (\bar{T} - T_{amb}) - a_2 * A * (\bar{T} - T_{amb})^2$

a_1 et a_2 sont fournis par le constructeur de manière classique. Néanmoins, l'équation bilan sur un élément dz du panneau donne:

$\dot{m} \times C_p \times \frac{dT}{dz} = \eta_0 \times l \times I - \alpha_1 \times l \times (T(z) - T_{amb}) - \alpha_2 \times l \times (T(z) - T_{amb})^2$

Etant donné que cette dernière formulation est plus pratique que la première lorsqu'il s'agit d'étudier un champ solaire constitué de plusieurs panneaux à la suite (en supposant qu'il n'y a pas de pertes dans les connections..), il serait bien de pouvoir passer de l'un à l'autre. Instinctivement, on voudrait les relier par $\alpha_1 = \frac{a_1}{L}$ avec L la longueur du panneau. Comment vérifier cela?

Il est possible de résoudre de manière analytique la deuxième équation, en passant par un argth. En effet, si l'on suppose que la température ne fait que grimper dans le panneau (et donc que $\frac{dT}{dz} > 0$), il est possible d'arriver sur l'équation différentielle suivante:

$$\frac{dT^*}{1 - T^{*2}} = \nu_2 \times \sqrt{\frac{A}{\nu_2} + \frac{\nu_1^2}{4 \times \nu_2^2}}$$
$$avec\, T^* = \frac{T - T_{amb} + \frac{\nu_1}{2 \times \nu_2}}{\sqrt{\frac{A}{\nu_2} + \frac{\nu_1^2}{4 \times \nu_2^2}}},$$
$$A = \frac{\eta_0 \times l \times I}{\dot{m} \times C_p},$$
$$\nu_1 = \frac{\alpha_1}{\dot{m} \times C_p},$$
$$\nu_2 = \frac{\alpha_2}{\dot{m} \times C_p}$$

$\frac{dT}{dz} > 0$ implique que $T^* < 1$, et donc que la solution de cette équation différentielle est bien un argth. Cela donne:

$$d\, argth(T^*) = \nu_2 \times \sqrt{\frac{A}{\nu_2} + \frac{\nu_1^2}{4 \times \nu_2^2}} = M,$$
$$donc:\ T^* = th(argth(T^*(0)) + M * z) = \frac{T^*(0) + th(z \times M)}{1 + T^*(0) \times th(z \times M)}$$

L'objectif va donc être de trouver α_1, α_2 tels que les pertes linéaires et les pertes quadratiques s'égalisent. Cela est complexe analytiquement, car nous avons un grand nombre de fonctions non linéaires intervenant dans l'égalité. Nous allons donc tracer la fonction en faisant varier les divers paramètres et voir si la solution instinctive est juste (càd, si T est linéaire dans le panneau).

Au final, pareil pour le panneau ou tout autre chose: c'est linéaire SI le mdot est assez gros, that is the question. On va donc tracer T(L) en fonction de mdot.

```
import math
eta0=0.831
a1 = 3.08
a2 = 0.01
Cp=4187
l=1
Tamb=10
```

```
class Panneau:
    def __init__(self, Tin, eta0, a1, a2, I, Tamb, mdot, Cp, Ap):
        self.Tin=Tin
        a= a2*Ap/4
        b= mdot*Cp+a1*Ap/2-a2*Ap*(Tin/2-Tamb)
        c=eta0*Ap*I+a1*(Tamb-Tin/2)*Ap-a2*Ap*(Tin/2-Tamb)+mdot*Cp*Tin
        c=-c
        delta=b*b-4*a*c
        self.Tout = (-b+math.sqrt(delta))/(2*a)
```

```
Nb=12
T0=Tamb
I=750
mdot=20*12/3600
tabPan=[Panneau(T0,eta0,a1,a2,I,Tamb, mdot, 4187, 2)]
for pan in range(0,Nb):
    tabPan.append(Panneau(tabPan[-1].Tout,eta0,a1,a2,I,Tamb, mdot, 4187, 2))
```

```
for k in range(0,Nb):
    print(tabPan[k].Tin, '-->',tabPan[k].Tout)
```

```
10 --> 14.408442099727381
14.408442099727381 --> 18.719369296428567
18.719369296428567 --> 22.935603686994455
22.935603686994455 --> 27.059861394093332
27.059861394093332 --> 31.09475750905517
31.09475750905517 --> 35.04281075996687
35.04281075996687 --> 38.90644792273292
38.90644792273292 --> 42.68800799142696
42.68800799142696 --> 46.38974612312836
46.38974612312836 --> 50.01383737142646
50.01383737142646 --> 53.562380221637795
53.562380221637795 --> 57.03739993996919
```

```
for k in range(0,Nb):
    print((tabPan[k].Tout-tabPan[k].Tin)*mdot*Cp/(2*I))
```

```
0.8203620920692686
0.8022156521150162
0.7845943730355277
0.7674785342054665
0.7508493348153432
0.7346888427363236
0.718979946688954
0.7037063114498633
0.6888523358859459
0.6744031136277397
```

(continues on next page)

(continued from previous page)

```
0.6603443961704379
0.6466625582512692
```

```
eta0 - a1*(54-Tamb)/I - a2*(54-Tamb)*(54-Tamb)/I
```

```
0.6244933333333332
```