TP-1: Le modèle Réservoir + Source + Puit + Flux

Le but de ce premier TP est de vous faire apprivoiser le modèle Réservoir + Source + Puit + Flux décrit à la §2.4 des Notes de Cours. Des formes plus élaborées (i.e., un plus grand nombre de réservoirs, flux, etc) seront dévelopées et utilisées tout au long du cours, et vous aurez à le faire aussi dans au moins un TP ultérieur. D'où l'idée de s'y mettre dès maintenant.

1. Approche numérique et validation

Commençons par la version de la §2.4 des Notes de cours. Une fois la forme mathématique/physique des divers flux, sources et puits spécifiée, il ne s'agit plus que de solutionner les équations (2.47)–(2.48), soit un système de deux équations différentielles ordinaires (EDO) couplées et nonlinéaire. Le couplage vient des termes de flux, i.e., Q_B apparait dans l'équation d'évolution pour Q_A, et vice-versa. La nonlinéarité vient de la forme du terme puit introduit dans l'équation pour Q_A, i.e., $\propto Q_A^4$. Notons également que le terme source dans (2.47) dépend explicitement de la variable indépendante, soit t, la forme de cette dépendance pouvant être non-triviale (viz. le trait pointillé sur le graphique du bas de la Fig 2.7). Nous allons donc y aller numériquement, afin de produire quelque chose qui fonctionnera en toute généralité.

Il existe une multitude de méthodes numériques bien connues pour la solution de systèmes d'EDO nonlinéaires couplées. Votre première tâche sera donc d'aller chercher dans la librairie `SciPy` une routine appropriée; où d'adopter celle que je vous fournis en Annexe à cet énoncé de TP. Franchement, je vous suggère la première option...

Votre seconde tâche sera de coder le système d'EDO décrit par (2.47)–(2.48), et de reproduire les résultats présentés à la Figure 2.7. Je peux maintenant vous révéler que le terme source utilisé pour produire la solution du graphique du bas est une fonction sinusoidale tronquée de sa partie négative, i.e.

$$S(t) = S_0 \sin(2\pi t/P) \tag{1}$$

en forçant ensuite $S = 0$ aux valeurs de t pour lesquelles $S < 0$. C'est le terme source qui sera utilisé dans le reste de ce TP.

2. Petite étude paramétrique

La Figure 2.7 des Notes (graphique du bas) montre bien qu'il existe un décalage temporel entre $S(t)$, $Q_A(t)$, $Q_B(t)$; ainsi qu'une différence dans les amplitudes d'oscillation de Q_A et Q_B. Quand on développe un modèle quelconque qui implique certains paramètres (ici τ et σ dans (2.47)–(2.48), et la période P et amplitude S_0 du terme source dans l'éq. (1) ci-dessus), et avant de se mettre à la modélisation d'un système spécifique, il est important de développer une "intuition" pour le comportement du modèle face au changement dans les valeurs de ses paramètres. Donc,

1. Réfléchissez et/ou discutez entre vous, et essayer de "deviner" comment les délai temporel et amplitude des oscillations vont varier avec ces paramètres. Vous pouvez même vous risquer à faire des prédictions quantitatives.

2. Vérifiez vos prédictions en explorant le comportement du modèle pour diverses combinaisons de valeurs de ces paramètres,

3. Un modèle plus physique: l'océan

On applique maintenant le modèle RFSP au chauffage par le soleil des couches superficielles de l'océan. On commence avec une seule couche, la couche océanique de surface, qu'on supposera de profondeur $d = 30\,\mathrm{m}$, sujette à une insolation S (unités: W m^{-2}) et des pertes radiatives σT^4 (aussi en W m^{-2}), l'évolution de la température est décrite par:

$$\frac{\mathrm{d}T}{\mathrm{d}t} = -\frac{\Delta T}{\tau} + \frac{1}{\rho c_p d}\left(S(t) - \sigma T^4\right) \tag{2}$$

où ΔT est la différence de température entre la couche de surface et une couche plus profonde, si il y en avait une (comparez avec (2.47) dans les Notes!), et τ le temps caractéristique d'échange de chaleur entre ces deux couches, comme auparavant. L'apparition du facteur ρc_p est nécessaire pour passer de la température au contenu énergétique (revoir la §2.1.2 des Notes au besoin). Vous devez donc maintenant:

1. Commencer par effectuer une analyse d'unités sur l'éq. (2) ci-dessus, pour vous convaincre que tout balance bien; toujours une bonne idée!

2. Ensuite, utilisez votre code pour solutionner l'éq. (2) sans le terme de flux (i.e., $\Delta T \equiv 0$) et $S(t)$ donné par l'équation (1); attention, ici vous n'avez qu'une seule équation à résoudre. Utilisez $P =$ 24 heures (converti en secondes!) et $S_0 = 450\,\mathrm{W\,m^{-2}}$. Choissisez une condition initiale appropriée et assurez-vous d'intégrer jusqu'à ce que le système ait atteint un état périodique stationnaire.

3. À quelle heure (locale) du jour la température océanique de surface atteint-elle son maximum ? Quelle est la température moyenne, c.-à-d., la température moyennée sur une journée ?

4. Un modèle encore plus physique

Étant en contact avec les couches océaniques plus profondes, de températures plus basses, la couche océanique de surface perd de la chaleur par conductivité thermique vers ces couches profondes, ainsi que par échanges turbulents associés à la convection thermohaline (on discutera des détails de tout ça plus tard dans le cours). Pour le moment on capturera l'impact de ces phénomènes via notre temps caractéristique τ, dont on fixera la valeur à $\tau = 3 \times 10^5\,\mathrm{s}$. Il s'agit maintenant d'inclure tout ça dans le modèle.

1. Ajoutez au modèle une seconde couche, de même épaisseur $d = 30\,\mathrm{m}$. Ceci implique d'écrire une seconde EDO de forme appropriée pour cette couche, et de réintroduire le terme de flux, avec ΔT donné par la différence de température entre les deux couches.

2. Solutionnez numériquement vos deux EDO couplées, et comparez l'évolution de la température dans la couche de surface avec celle obtenue initialement sans couche profonde. Les températures maximale et moyenne ont-elles changé de manière significative ?

3. Répétez le calcul avec une couche profonde dix fois plus épaisse; observez-vous une différence significative?

Annexe: un petit code

Pour ceux/celles qui préfèrent tout faire tout seul, voici (Figure 1) un petit code Python qui solutionne un système de deux EDOs couplées, par la méthode de Runge-Kutta d'ordre 4, avec pas temporel adaptif. On considère ici des EDOs prototypiques de la forme

$$\frac{\mathrm{d}\mathbf{u}}{\mathrm{d}t} = \mathbf{g}(t,\mathbf{u}) \ , \tag{3}$$

où en général la fonction $\mathbf{g}(t,\mathbf{u})$ peut dépendre de t ou $\mathbf{u}$ de manière nonlinéaire. Géométriquement, $\mathbf{g}$ représente la pente de la fonction $\mathbf{u}(t)$. On se limitera ici à des situations où une condition initiale est donnée à $t = 0$, et le problème consiste à avancer la solution $\mathbf{u}(t)$ dans le temps. Le code présenté à la page suivante solutionne le système de deux EDO couplées nonlinéaires suivant:

$$\frac{\mathrm{d}X}{\mathrm{d}t} = A-(B+1)X+X^2Y \ , \tag{4}$$

$$\frac{\mathrm{d}Y}{\mathrm{d}t} = BX-X^2Y \ . \tag{5}$$

autrement dit, dans l'éq. (3) on a $\mathbf{u} \equiv (X,Y)$ et $\mathbf{g} \equiv (A-(B+1)X+X^2Y, BX-X^2Y)$, qui se retrouve codé dans la fonction Python `g(t0,u)`. Notons qu'ici le temps (variable `t[nn]` est fourni en argument à la fonction `rk` qui le passe à la fonction `g`... qui LANCE ET COMPTE !!!! ... s'cusez on continue..., même si cette dernière ne l'utilise pas, histoire de demeurer général (viz. l'éq. (3)). A noter également, Python permet aux fonctions de retourner un tableau (ici de dimension 1 et taille 2), ce qui n'est pas le cas de tous les langages de programmation.

```
import numpy as np
# FONCTION CALCULANT LE COTE DROIT DU SYSTEME DE DEUX EDO
def g(t0,u):
   A,B=1.,3.                          # parametres dans (4)--(5)
   gX=A-(B+1.0)*u[0]+u[0]**2*u[1]     # RHS Eq (4)
   gY=B*u[0]-u[0]**2*u[1]             # RHS Eq (5)
   eval=np.array([gX,gY])             # vecteur RHS (pentes)
   return eval
# END FONCTION G
# FONCTION CALCULANT UN SEUL PAS DE RUNGE-KUTTA D'ORDRE 4
def rk(h,t0,uu):
   g1=g(t0,uu)
   g2=g(t0+h/2.,uu+h*g1/2.)
   g3=g(t0+h/2.,uu+h*g2/2.)
   g4=g(t0+h,uu+h*g3)
   unew=uu+h/6.*(g1+2.*g2+2.*g3+g4)
   return unew
# END FONCTION RK
# OSCILLATIONS NONLINEAIRES: 2 EDOS NONLINEAIRES COUPLEES
nMax=1000                             # nombre maximal de pas de temps
eps =1.e-5                            # tolerance
tfin=10.                              # duree d'integration
t=np.zeros(nMax)                      # tableau temps
u=np.zeros([nMax,2])                  # tableau solution
u[0,:]=np.array([3.,3.])              # condition initiale
nn=0                                  # compteur iterations temporelles
h=0.1                                 # pas initial
while (t[nn] < tfin) and (nn < nMax):  # boucle temporelle
   u1 =rk(h,   t[nn],u[nn,:])         # pas pleine longueur
   u2a=rk(h/2.,t[nn],u[nn,:])         # premier demi-pas
   u2 =rk(h/2.,t[nn],u2a[:])          # second demi-pas
   delta=max(abs(u2[0]-u1[0]),abs(u2[1]-u1[1]))
   if delta > eps:                    # on rejette
      h/=1.5                          # reduction du pas
   else:                              # on accepte le pas
      nn=nn+1                         # compteur des pas de temps
      t[nn]=t[nn-1]+h                 # le nouveau pas de temps
      u[nn,:]=u2[:]                   # la solution a ce pas
      if delta <= eps/2.: h*=1.5      # on augmente le pas
   print("{0}, t {1}, X {2}, Y {3}.".format(nn,t[nn],u[nn,0],u[nn,1]))
# fin boucle temporelle
# END
```