# Calcul numérique du MGI d'un RRR par PNL

On se propose de calculer de manière numérique un modèle géométrique inverse (MGI) d'un robot RRR plan (les 3 rotations sont perpendiculaires au plan de travail) en utilisant les outils de programmation non-linéaire.
La configuration du robot est défini par le vecteur $\boldsymbol{q}= (q_1, q_2, q_3)^t$.
La situation de l'outil est défini par le vecteur $\boldsymbol{X}= (x, y, \theta)^t$ avec $\theta \in [0, 2\pi]$.

L'objectif est d'implémenter plusieurs méthodes de recherche de solution pour le calcul du MGI (recherche d'un optimum local), de les comparer et d'en tirer des conclusions. Il est demandé ensuite d'utiliser ces méthodes pour faire déplacer le robot entre deux points du plan à orientation fixe.

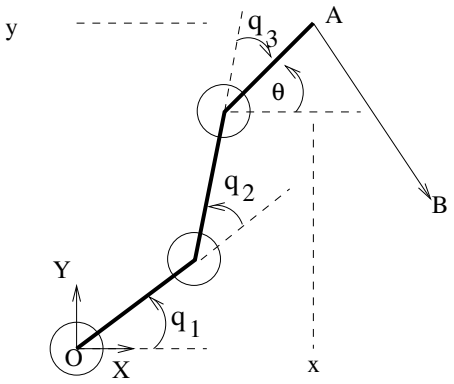


FIGURE 1 – Trajectoire dans l'espace opérationnel

Les conditions de sortie des algorithmes seront définies par un nombre maximum d'itérations et par une valeur seuil pour l'erreur de la fonction objectif.

Vous devez définir la fonction de calcul du mgi : $[qsol] = mgi(Xd, q_0, NbIter, \epsilon)$ avec $Xd$ situation désirée, $q_0$ configuration initiale du robot, $NbIter$ nombre maximum d'itérations, $\epsilon$ précision du calcul de la solution.

Pour calculer le MGI avec les outils de programmation non linéaire vous devez implémenter une fonction qui à chaque itération va calculer $\boldsymbol{q}_{k+1} = \boldsymbol{q_k} + pas * direction$.
Le calcul du *pas* et de la *direction* dépend de la méthode mise en œuvre.

Le fichier $tpEtudiant - mgi.py$ contient
- les *import* des librairies python utilisées,
- la fonction de calcul du modèle géométrique direct (MGD) du robot ($[Xd] = mgd(qrad)$),
- la fonction de calcul de la jacobienne du robot ($J = jacobienne(qrad)$)

Travail à faire :
- implémenter le calcul du MGI numérique en utilisant une approche basée Newton,
- implémenter le calcul du MGI numérique en utilisant une approche basée Gradient,
- implémenter une fonction MGI qui prenne en compte des contraintes.

# 1 Méthode de Newton

On utilise le schéma de Newton pour calculer le MGI du robot en cherchant le zéro de la fonction : $H(\boldsymbol{q}) = X_d - f(\boldsymbol{q})$ pour une situation désirée $\boldsymbol{X}_d = (x, y, \theta)^t$ avec $f(\boldsymbol{q})$ qui correspond au MGD en $\boldsymbol{q}$. Ce schéma impose de calculer l'inverse de la jacobienne analytique.

- Montrer que $direction = J(\boldsymbol{q})^{-1}.(X_d - f(\boldsymbol{q}))$.
- Implémenter la méthode de Newton en python pour résoudre le MGI. Tester pour différentes conditions initiales (proches/loin de la solution, proches/loin de singularités. Modifier la valeur du *pas*. Faire varier $N_{max}$ et $\epsilon$.
- Tracer la variation de l'erreur. Conclusions.

# 2 Méthode du gradient

On utilise la méthode du gradient pour minimiser un critère $C(q) = \frac{1}{2}\ \boldsymbol{e}^t.\boldsymbol{e} = \frac{1}{2}\ \|\boldsymbol{e}\|^2$ avec $\boldsymbol{e} = \boldsymbol{X}_d - f(\boldsymbol{q})$ avec $f(\boldsymbol{q})$ qui correspond au MGD en $\boldsymbol{q}$..

- Montrer que $direction = -\nabla C(\boldsymbol{q}) = J(\boldsymbol{q})^t.\boldsymbol{e}$.
- Implémenter la méthode du gradient pour résoudre le MGI. Tester pour différentes conditions initiales (proches/loin de la solution, proches/loin de singularités. On choisira un coefficient $pas = 0.5$, puis on le fera varier. Quel coefficient semble le plus approprié ?
- Tracer la variation de l'erreur. Comment régler le pas afin de garantir une convergence du gradient ? Proposer et implémenter une solution de calcul automatique du pas.
- Conclusions

# 3 Utilisation de scipy.optimize

Chercher les fonctions à utiliser (Optimization and root finding ) permettant de calculer les solutions des sections précédentes. Tester une méthode utilisant une approximation de la jacobienne.

# 4 Optimisation sous contrainte

## 4.1 Contrainte de butée

Définir un problème d'optimisation qui calcule la solution du MGI en imposant une butée max sur la première liaison (ex : q1 *leq* 0.1)
Implémenter et tester votre modélisation du problème en utilisant la fonction *minimize*.

## 4.2 Contrainte sur la solution

Modéliser votre problème de calcul du MGI pour une situation $X_p$ donné en cherchant la solution la plus proche de la configuration initiale.
Implémenter et tester votre modélisation du problème en utilisant la fonction *minimize*.

# 5 Travail à rendre en fin de séance

- Rapport du TP : $TPMGI\_Nom1\_Nom2.pdf$
- Fichier Python : $mgi\_Nom1\_Nom2.py$. Je dois seulement lancer le fichier pour tester votre programme. Mettre en commentaire au début du fichier le mode d'utilisation.

```
#############
# Code à utiliser pour débuter votre TP
#################
import matplotlib.pyplot as plt
import numpy as np
import time

################################################
# Calcul du MGD du robot RRR
# INPUT:  q = vecteur de configuration (radian, radian, radian)
# OUTPUT: Xc = vecteur de situation = (x,y, theta)
#              x,y en mètre et theta en radian
def mgd(qrad):
#### Paramètres du robot
    l=[1,1,1]
    c1= np.cos(qrad[0])
    s1=np.sin(qrad[0])
    c12= np.cos(qrad[0]+qrad[1])
    s12=np.sin(qrad[0]+qrad[1])
    theta= qrad[0]+qrad[1]+qrad[2]
    c123=np.cos(theta)
    s123=np.sin(theta)
    x=l[0]*c1 + l[1]*c12 +l[2]*c123
    y=l[0]*s1 + l[1]*s12 +l[2]*s123
    Xd=[x,y,theta]
    return Xd
################################################
# test de validation du MGD
##### INPUT de q en degré ###
qdeg = [90, -90, 0]
qr = np.radians(qdeg)
Xd= mgd(qr)
print("X=", Xd[0], "Y = ", Xd[1], "Theta (deg)= ", np.degrees(Xd[2]))

################################################
# Calcul de J(q) du robot RRR
# INPUT:  q = vecteur de configuration (radian, radian, radian)
# OUTPUT: jacobienne(q) analytique
def jacobienne(qrad):
#### Paramètres du robot
    l=[1,1,1]
    c1= np.cos(qrad[0])
    s1= np.sin(qrad[0])
    c12= np.cos(qrad[0]+qrad[1])
    s12=np.sin(qrad[0]+qrad[1])
    theta= qrad[0]+qrad[1]+qrad[2]
    theta= np.fmod(theta,2*np.pi)
    c123=np.cos(theta)
    s123=np.sin(theta)
```

```
    Ja=np.array([[-(l[0]*s1 + l[1]*s12 +l[2]*s123), -(l[1]*s12 +l[2]*s123), -(l[2]*s123)],
                 [(l[0]*c1 + l[1]*c12 +l[2]*c123), (l[1]*c12 +l[2]*c123),  (l[2]*c123)],
                  [1, 1, 1]])

    return Ja

##############################################################################
# Afin de donner une situation atteignable pour le robot,
# vous pouvez utiliser le mgd pour définir Xbut à partir d'une configuration en q
##############################################################################
## qbut est donné en degré
qbutdeg= np.asarray([45, 45,  -60.])
## Calcul Xbut à partir de qbut
Xbut= np.asarray(mgd(np.radians(qbutdeg)))
print("Xbut=", Xbut[0], "Ybut = ", Xbut[1], "Theta but (deg)= ", np.degrees(Xbut[2]))

##############################################
## Fct d'affichage 2D du robot dans le plan
def dessinRRR(q) :
    xA, yA = (0, 0)
    xB, yB = (np.cos(q[0]), np.sin(q[0]))
    xC, yC = (np.cos(q[0]) + np.cos(q[0]+q[1])), (np.sin(q[0]) + np.sin(q[0]+q[1]))
    xD, yD = (np.cos(q[0]) + np.cos(q[0]+q[1]) + np.cos(q[0]+q[1]+q[2])),(np.sin(q[0]) + np
    X=[xA, xB,xC,xD]
    Y=[yA,yB,yC,yD]
    plt.plot(X,Y, color="orange", lw=10, alpha=0.5, marker="o", markersize=20, mfc="red")
    plt.axis('equal')
    plt.axis('off')
    plt.show()
##############################################
############ Exemple d'affichage
dessinRRR(np.radians(qbutdeg))

########################################
######
###### Boucle principale de calcul du MGI
########################################
####
#### Définition de Xbut à partir de qbutdeg en degré
qbutdeg= np.asarray([45.,45.,-60])
qbut = np.radians(qbutdeg)
Xbut= np.asarray(mgd(qbut))
dessinRRR(qbut)
#### Définition de qinit
qinitdeg=np.asarray([120., 25,  45.])
qinit= np.radians(qinitdeg)
Xinit=np.asarray(mgd(qinit))
print("Xinit = ",Xinit)
```

```
##### A CODER
##q= qinit
##i=0
##
##
##nbpas= ???
##epsx= ???
## erx = valeur initale de du critère qu'on cherche à minimiser
##list_erreur = [erx] #
##start_time = time.process_time()
##while (????):
##    direction = ???
##    pas= ???
##    ...
##    ...
##    list_erreur.append(erx) # Stocker la valeur du critère dans une liste
##    i=i+1
##print("--- %s seconds ---" % round(time.process_time() - start_time,6))
### Visualisation des résultats
##print("Valeur finale du critère =",??," après ",i," itérations")
##print("qinit en deg =", qinitdeg)
##print("qcfinal en deg", np.degrees(qc))
##X= mgd(qc)
##print("Xinit              =",Xinit, type(Xinit))
##print("Xfinal avec qfinal = ",X)
##print("Xbut à atteindre   =", Xbut, type(Xbut))
##Xer= Xbut -X
##erx=np.linalg.norm(Xer)
##print("Erreur finale=",erx)
##abs = np.linspace(0,len(list_erreur)-1,(len(list_erreur)))
##plt.plot(abs,list_erreur,'k')
##plt.show(block=True)
##
##dessinRRR(qc)
```