IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Option Data Science - ENSI Tunis

TP Modèles statistiques de formes

06 décembre 2024 - Durée 3h

Ce TP contribue à l'évaluation du cours de Prof. Burdin. A l'issue, vous enverrez votre CR par binôme à l'adresse valerie.burdin@univ-brest.fr. Nous attendons vos réponses aux 10 questions, argumentées et illustrées de code et copie d'écran graphique.

1 Préparation du TP

Cette partie est à réaliser avant la séance de TP. Elle doit vous prendre une demi-heure environ.

1.1 Contexte

Il s'agit de vous initier à la notion de modèles statistiques 3D appliqués à l'anatomie humaine. Ces modèles sont utilisés pour faire des études morphologiques, de la classification de formes 3D, des reconstructions de parties manquantes sur une forme 3D, par exemple.

On utilisera un environnement libre, Scalismo, développé par le groupe Gravis de l'UNIBAS (Suisse) avec lequel le LaTIM collabore. Le langage utilisé est Scala (développé par l'EPFL) mais la fonctionnalité ScalismoLab permet d'éviter l'apprentissage de ce langage (proche de Java). Notez que deux barres obliques "`//`" permettent de commenter une ligne de code.

Veuillez installer l'environnement (1.2) et vérifier que cela fonctionne en exécutant le tutorial (1.3), puis en l'appliquant à la section 1.4, avant la séance de TP.

1.2 Installation de l'environnement ScalismoLab

1.2.1 Ordinateur personnel

Si vous travaillez sur votre ordinateur personnel, il faut installer ScalismoLab. Vous trouverez les instructions d'installation sur le lien suivant : `https://github.com/unibas-gravis/scalismo/wiki/scalismoLab`.

1.2.2 Préparation des données

Vérifiez qu'il existe un dossier `./scalismolab/datasets`. Il est utile pour faire le tutorial. Il ne sert pas pour le TP. Vous allez maintenant copier un répertoire appelé **"datasets-TP"** au même niveau que **"datasets"**.

A l'endroit indiqué par le professeur, téléchargez le fichier "datasets-TP.zip" et décompressez le dans le dossier **"scalismolab"**

Vous obtenez un dossier `/scalismolab/datasets-TP` **au même niveau que "datasets"**.

1.3 Tutorial

Lancer scalismolab avec la commande

```
./scalismolab.sh
```

Le warning sur le module "canberra" n'est pas gênant.

Vous obtiendrez trois fenêtres :

— Scalismo tutorial : affiche les énoncés des tutoriels
— Scalismo code pane : éditeur de code
— Scalismo Viewer : Visualisateur graphique + sortie du Terminal

Faire le tutorial "Hello scalismo !" qui permet de vous familiariser avec le logiciel scalismolab. Il est disponible dans la fenêtre "Scalismo Viewer" puis le menu "Documents/Fast Track".
A partir de la fenêtre "Scalismo tutorial", sélectionnez la ligne de code et taper Shift/Enter pour l'exécuter. La commande apparaît dans la fenêtre "Scalismo code pane" et son résultat s'affiche dans "Scalismo Viewer" (partie Terminal ou graphique).

1.4 Visualisation de maillages, de landmarks et déclaration de transformations rigides

Dans la première partie du TP, on considère que **nous avons une collection d'exemples en correspondance**, sous forme de plusieurs maillages de structure identique. Il s'agit de prendre en main les outils proposés par cet environnement ScalismoLab (manipulation des maillages 3D) et de comprendre comment faire du recalage rigide.

Q1 : D'après le cours que veut dire : les maillages sont en correspondance ?

1.4.1 Chargement dans le UI et affichage de l'ensemble des maillages d'un répertoire (fichiers au format .stl)

Copier et coller le texte suivant dans la fenêtre "Scalismo code pane". **Attention aux retours à la ligne qui peuvent couper les commandes une fois le texte collé dans la fenêtre "Scalismo code pane"!**

```
for(i<- 1 to 5){
   val mesh : TriangleMesh = MeshIO.readMesh(new File(
   "datasets-TP/nonAlignedFaces-TP/"+i+".stl")).get
   show(mesh,"face"+i)
} // Affichage et création des objets statiques (instances)
//dans le User Interface (UI)
```

Pour exécuter des lignes de commande du "Scalismo code pane", les sélectionner et taper Shift/Enter.

Donnez une couleur différente à chaque maillage et une transparence pour mieux visualiser (utiliser le UI et clic-droit sur les objets). On remarque que les instances ne sont pas alignées.

Q2 : Utilisez la commande show sur la variable "mesh" en cours pour vérifier ce qu'elle contient.

Remarque : la commande "remove" de l'UI, ne supprime pas les variables. Pour les effacer de la mémoire et pas uniquement du viewer, il faut quitter ScalismoLab et le relancer. Pensez à sauvegarder vos lignes de code dans un fichier texte au cours du TP. Dans la fenêtre "Scalismo Viewer", sélectionner "File/Save Code". A chaque partie du TP fermez le logiciel pour nettoyer la mémoire.

Astuces : Pour faire une copie d'écran de la fenêtre graphique, cliquer sur le bouton `SS` (Screen Shot). Pour faire un affichage multivues, utilisez le menu View/Perspective et choisir la modalité d'affichage. Pour remplir les fenêtres, sélectionnez votre maillage dans la liste `Static Objects`, faire un clic-droit et sélectionnez la fenêtre d'affichage avec le menu "Visible in".

1.4.2 Sélection de landmarks manuels

En utilisant l'onglet `LM` du viewer (barre du haut), définissez 6 landmarks sur les deux instances 1 et 5 dans le même ordre anatomique que vous définirez. Ces landmarks doivent être reproductibles et permettre d'analyser la variabilité 3D des objets. Vous pouvez rendre invisibles les maillages par la UI (clic-droit, visible) afin de sélectionner successivement les landmarks sur

chacun d'eux.

Q3 : Où placez-vous les landmarks ?

Remarquez qu'ils apparaissent dans la structure statique associée (UI). On peut également les supprimer (clic-droit), changer leur couleur, et les sauvegarder dans un fichier *.json (clic-droit).

Astuce : Pour agrandir un landmark, cliquer sur le landmark, dans l'onglet "Uncertainty" mettre la déviation standard à 5mm sur les trois axes et cliquer Apply.

1.4.3 Déclaration d'une transformation rigide et application à un objet maillé

La documentation se trouve dans l'onglet "Documents/Full track/Rigid alignment"

Exemple : Déclaration d'une rotation 3D d'angles 0, π, et 0 (il s'agit de float), et d'une translation 3D de 50, 0, et 0 sur les axes respectifs x, y, et z.

```
val rotation : RotationTransform[_3D] = RotationTransform(0f,3.14f,0f)
val translation = TranslationTransform[_3D](Vector(50,0,0))
```

Voici le code pour définir une transformation rigide composée, appelée `composedTransformation`, permettant de transformer les coordonnées d'un nuage de points 3D (vérifiez que la multiplication matricielle n'est pas commutative).

```
def composedTransformation(p: Point[_3D]) : Point[_3D] =
      translation(rotation(p))
```

Appliquez la transformation à un maillage "datasets-TP/nonAlignedFaces-TP/*.stl". Il faudra le relire pour actualiser la variable "mesh".

```
val mesh : TriangleMesh = MeshIO.readMesh(new File(
      "datasets-TP/nonAlignedFaces-TP/5.stl")).get
show(mesh,"face5")
val rigid : TriangleMesh = mesh.transform(rotation)
show(rigid,"rotatedface5")
val rigid2 : TriangleMesh = rigid.transform(translation)
show(rigid2,"translatedrotatedface5")
val rigid3 = mesh.transform(composedTransformation)
show(rigid3,"composedface5")
```

Le TP encadré commence ici

2 TP - Partie 1 : Recalage de la base d'exemples et analyse statistique de forme

2.1 Recalage rigide par la méthode de Procruste (landmarks)

Dans cette partie, vous allez chercher la transformation rigide qui permet de recaler une instance quelconque sur l'instance de référence et appliquer cette transformation pour superposer les deux maillages.

Voici le code pour récupérer les landmarks d'une instance donnée (par exemple L-face1) dans une variable "`originalLms`". (Dans notre cas, les landmarks se trouvent déjà sous format .json dans le répertoire : "datasets-TP/nonAlignedFaces-TP/*.json")

```
// lecture de landmarks d'après un fichier *.json :
val originalLms = LandmarkIO.readLandmarksJson[_3D](new File(
   "datasets-TP/nonAlignedFaces-TP/L-face1.json")).get
// récupération de landmarks définis manuellement sur "instance1"
// dans l'UI (landmarks A,B,...) :
val originalLms = getLandmarksOf("instance1").get
```

Pour visualiser des landmarks issus d'un fichier json sur son maillage, il faut les ajouter à l'UI. Par exemple pour "face", exécutez :

```
originalLms.map(p => addLandmarksTo(Seq(Landmark(p.id,p.point)),"face"))
```

Pour trouver la meilleure transformation rigide entre deux ensemble de points (ici les landmarks `rigidLms` et `originalLms`), exécuter

```
val bestTransform = LandmarkRegistration.rigid3DLandmarkRegistration(
                        rigidLms,originalLms)
// on envoie rigid sur original.
```

Q4 : Choisissez deux instances du répertoire "datasets-TP/nonAlignedFaces-TP/" et décider la référence et l'objet mobile. Ecrire le code permettant de les afficher et d'ajouter leurs landmarks respectifs (manuels ou sauvegardés).

Q5 : Écrivez le code pour estimer la meilleure transformation permettant de ramener l'objet mobile sur la référence en utilisant les 2 jeux de landmarks. Appliquez cette transformation à l'objet mobile pour le superposer à la référence et visualiser. Commentez.

On peut calculer une métrique entre deux maillages mesh1, mesh2. On utilise souvent la Distance de Hausdorff modifiée symétrique (voir `https://en.wikipedia.org/wiki/Hausdorff_distance`) ou la distance moyenne symétrique. Ces deux métriques peuvent être calculées à l'aide des commandes suivantes :

```
MeshMetrics.hausdorffDistance(mesh1,mesh2) // Distance de Hausdorff
MeshMetrics.avgDistance(mesh1,mesh2) // Distance moyenne
```

Q6 : Écrivez le code calculant la distance de Hausdorff entre la référence et l'objet mobile avant et après recalage. Commentez.

2.2 Préparation de la base de données pour construire le modèle statistique

Q7 : Effacez tout le UI. Créez un répertoire "datasets-TP/AlignedFaces-TP/". Choisissez un mesh pour la référence et copiez le dans le répertoire "datasets-TP/AlignedFaces-TP/". Ecrivez le code pour aligner toutes les faces sur la référence à partir de leurs landmarks associés (.json). Sauvegardez les maillages alignés dans le nouveau répertoire sous les noms

```
face_bestalignedX.stl
```

Voici le code pour sauvegarder la variable mesh dans un fichier face_bestaligned.stl

```
MeshIO.writeMesh(mesh,new File(
   "datasets-TP/AlignedFaces-TP/face_bestaligned.stl")).get
```

2.3 Création du modèle statistique

Pour créer un modèle statistique, on effectue une Analyse en Composantes Principales sur les maillages en correspondance. Il faut choisir une référence, et les variations de chaque point du maillage au travers de la base seront analysées par rapport à cette dernière. La réduction de dimension de la matrice de covariance centrée dans la base des vecteurs propres permet de garder les corrélations principales, appelés modes. Voici les lignes de commande à adapter pour la question Q8 :

```
val reference : TriangleMesh = MeshIO.readMesh(new File(
   "reference.stl")).get
val dc = DataCollection.fromMeshDirectory(reference, new File(
   "datasets-TP/repertoire/"))._1.get
// Crée une collection de tous les maillages .stl d'un même répertoire
val model1 = StatisticalMeshModel.createUsingPCA(dc).get
// Crée le modèle statistique
show(model1,"statisticFace") // Affiche le modèle dans le UI
// Sauvegarde le modèle sous format .h5
StatismoIO.writeStatismoMeshModel(model1,new File(
      "datasets-TP/TP_nonaligned.h5"))
// Calcul les valeurs propres
val valeurpropres1=model1.gp.variance
println(valeurpropres1) // Affiche le vecteur des valeurs propres
```

Préparez deux répertoires contenant uniquement les maillages (.stl) alignés (resp. non alignés). Par exemple : "datasets-TP/nonAlignedFaces-TP" et "datasets-TP/AlignedFaces-TP".

Q8 : Écrivez le code pour créer les modèles statistiques (.h5) issus de ces deux répertoires et sauvegardez les deux fichiers .h5 sous le répertoire "datasets-TP".

Remarque : Avant de créer le SSM, vérifiez que les faces sont alignées dans le dossier TP-Aligned. En particulier, faire attention au sens de la transformation utilisée. Il faut recaler toutes les faces sur celle qui est choisie comme référence.

Q9 : Comparez les deux modèles statistiques visuellement (en se plaçant sous instance 1). Que représente le maillage affiché par défaut ? Interprétez la variation de la première composante principale de chacun des deux modèles. Lequel des deux permet une analyse morphologique de la face ? Commentez.

Pour évaluer un modèle statistique, trois critères sont utilisés :

— la compacité,

— la généralité,

— la spécificité.

Voici le code pour comparer les modèles .h5 en terme de compacité :

```
var cumulVariance1 = DenseVector.zeros[Double](5)
var cumulVariance2 = DenseVector.zeros[Double](5)
cumulVariance1(0) = valeurpropres1(0)
cumulVariance2(0) = valeurpropres2(0)

for(i<-1 to 4){
  cumulVariance1(i) = cumulVariance1(i-1)+valeurpropres1(i)
  cumulVariance2(i) = cumulVariance2(i-1)+valeurpropres2(i)
  }
cumulVariance1 = cumulVariance1*(100.0/cumulVariance1(4))
cumulVariance2 = cumulVariance2*(100.0/cumulVariance2(4))
println(cumulVariance1)
println(cumulVariance2)
```

Pour tracer la courbe de variance, sortez de ScalismoLab et utilisez Excel ou autre en copiant les données de la partie Terminal de la fenêtre “Scalismo Viewer”.

Q10 : Affichez la courbe de compacité. Commentez.

La partie 2 permet d’aller plus loin notamment sur la mise en correspondance de la base qui est un prérequis à la construction du modèle
(les maillages de faces fournis en partie 1 étaient déjà en correspondance).
Cette partie 2 est à commencer en TP et à finir en autonomie.

3 TP - Partie 2 : Construction d'un modèle statistique de scapula

Dans cette partie, les instances proviennent d'acquisition CT ou IRM et les maillages sont issus d'une segmentation. Les maillages n'ont pas forcément le même nombre de points et les indices des sommets ne sont pas en correspondance. Il faut donc effectuer cette mise en correspondance avant de pouvoir réutiliser le code ci-dessus.

3.1 Création d'une forme de référence déformable (template) par processus gaussien - Voir article de M. Luethi

```
// On lit le maillage de référence
val reference = MeshIO.readMesh(new File(
      "datasets-TP/TP-donnees/reference9000.stl")).get
show(reference,"reference")

// On initialise le champs de vecteur à 0.
val zeroMean = VectorField(RealSpace[_3D], (pt:Point[_3D]) => Vector(0,0,0))

// On donne les paramètres du noyau Gaussien
val s :Double = 10.0
val l :Double = 100.0

// On crée le processus continu
val scalarValuedKernel = GaussianKernel[_3D](l) * s
val matrixValuedKernel = DiagonalKernel[_3D](scalarValuedKernel)
val gp = GaussianProcess(zeroMean, matrixValuedKernel)

// On définit les points sur lequels on applique le processus
val sampler = RandomMeshSampler3D(
  reference,
  numberOfPoints = 300,
  seed = 42)
```

```
val lowRankGP = LowRankGaussianProcess.approximateGP(
  gp,
  sampler,
  numBasisFunctions = 100)

val  model = StatisticalMeshModel(reference, lowRankGP)
show(model, "Scapula_deformable")
//il n'est pas sauvegardé car déjà présent dans le répertoire
```

3.2 Fitting du modèle déformable (template) pour mettre en correspondance la base de données

```
val listenum = List(2,23,34,48,54,76)
for (i<- listenum){
val target = MeshIO.readMesh(new File(
      "datasets-TP/TP-donnees/Scap"+i+".stl")).get
show(target,"target")

val model = StatismoIO.readStatismoMeshModel(new File(
      "datasets-TP/TP-donnees/Scapula_deformable.h5")).get
show(model, "model")

def attributeCorrespondences(pts : Seq[Point[_3D]]) : Seq[Point[_3D]] = {
      pts.map{pt => target.findClosestPoint(pt).point}
}

val littleNoise = NDimensionalNormalDistribution(Vector(0,0,0),
SquareMatrix((1f,0,0), (0,1f,0), (0,0,1f)))
```

```
def fitModel(pointIds: IndexedSeq[PointId],
             candidateCorresp: Seq[Point[_3D]]) : TriangleMesh = {
  val trainingData = (pointIds zip candidateCorresp).map{
        case (mId, pPt) => (mId, pPt, littleNoise)
  }
  val posterior = model.posterior(trainingData.toIndexedSeq)
  posterior.mean
}

val pointSamples = UniformMeshSampler3D(model.mean, 5000, 42).sample.map(
      s => s._1)
val pointIds = pointSamples.map{s => model.mean.findClosestPoint(s).id}
val initialPositions = pointIds.map(id => model.mean.point(id))
show(initialPositions, "Tous_points")

def recursion(currentPoints : Seq[Point[_3D]], nbIterations : Int) : Unit = {
  println("iterations left " + nbIterations)
  val candidates = attributeCorrespondences(currentPoints)
  val fit = fitModel(pointIds, candidates)
  remove("fit")
  show(fit,"fit")
  MeshIO.writeMesh(fit,new File("datasets-TP/TP-result/Scap"+i+"corres.stl"))

  val newPoints = pointIds.map(id => fit.point(id))

  if(nbIterations> 0) {
    Thread.sleep(2000)
    recursion(newPoints, nbIterations - 1)
  }
}
//remove("Tous_points")
recursion(initialPositions , 10) // color the target to visualize better
```

⇒ On remplace les maillages d'origine par le template ajusté par fitting. On peut vérifier le bon ajustement en calculant la distance entre les deux maillages (Hausdorff ou moyenne).

3.3 Création du modèle statistique à partir des maillages en correspondance

```
// On supprime le 54

val listenum = List(2,23,34,48,76)
val data = for (i<- listenum) yield {
val example = MeshIO.readMesh(new File(
      "datasets-TP/TP-result/Scap"+i+"corres.stl")).get
show(example,"example")
}
val reference = MeshIO.readMesh(new File(
      "datasets-TP/TP-donnees/reference9000.stl")).get
val dc = DataCollection.fromMeshSequence(reference,data)._1.get

val ModelStatScap = StatisticalMeshModel.createUsingPCA(dc).get
show(ModelStatScap,"ModelStatScap")
StatismoIO.writeStatismoMeshModel(ModelStatScap,new File(
   "datasets-TP/TP-result/ModelStatScap.h5"))
val valeurpropres = ModelStatScap.gp.variance

// calcul de la compacite
var cumulVariance = DenseVector.zeros[Double](5)
cumulVariance(0) = valeurpropres(0)
for(i<-1 to 4){
  cumulVariance(i) = cumulVariance(i-1) + valeurpropres(i)
  }
cumulVarianc e =cumulVariance*(100.0/cumulVariance(4))
println(cumulVariance)
```