Morris-chaos-Sujet

February 5, 2023

1 Morris par le chaos

1.1 Consignes du projet

Les consignes de ce projet sont les suivants.

- Vous devrez rédiger un script Python pour réaliser un calcul.
- A l'aide de ce script, vous devez afficher les résultats du calcul.
- Vous commenterez vos résultats numériques tant sur le plan de la cohérence avec la méthode mathématique que sur le plan de la cohérence avec le modèle physique.
- Vous présenterez vos résultats sous la forme d'un Notebook Jupyter.
- Vous travaillerez en binômes.

1.2 Introduction

L'objectif principal de ce devoir est d'utiliser la méthode des polynômes du chaos dans le but de séparer les variables d'une fonction de grande dimension en deux catégories : les variables influentes et les variables peu influentes.

Il arrive souvent que la fonction au travers de laquelle on souhaite propager les incertitudes soit une fonction de grande dimension d'entrée, c'est à dire possédant plus de 5 à 10 variables d'entrée.

Ce grand nombre de variables d'entrée pose de multiples difficultés.

- Les méthodes sont mises en difficulté, par exemple pour créer un échantillon remplissant correctement l'espace.
- La pratique est plus complexe, par exemple pour identifier les lois de probabilités des nombreuses marginales et, à fortiori, la structure de dépendance.
- L'interprétation des résultats est complexe, car le grand nombre de variables obscurcit l'analyse.

C'est pour cette raison que la première étape consiste souvent à réduire le nombre de variables d'entrée du modèle. Pour cela, les méthodes d'analyse de sensibilité fondées sur les indices de Sobol' peuvent être utilisées. Toutefois, lorsque le nombre de variables d'entrée est vraiment grand (par exemple supérieur à 20), il n'est pas évident que les méthodes d'analyse de sensibilité puissent fonctionner avec une qualité suffisante. L'objectif de ce projet est d'observer le comportement du chaos polynomial dans cette situation.

Dans ce projet, nous proposons d'utiliser la fonction test de Morris, une fonction réelle possédant 20 variables d'entrée.

1.3 La fonction de Morris

La fonction de Morris est une fonction de $[0,1]^{20}$ vers $\mathbb{R}$. Pour tout $x \in [0,1]^{20}$, soit

$$g(x) = \beta_0 + \sum_{i=1}^{20} \beta_i w_i + \sum_{i<j}^{20} \beta_{i,j} w_i w_j + \sum_{i<j<\ell}^{20} \beta_{i,j,\ell} w_i w_j w_\ell + \sum_{i<j<\ell<s}^{20} \beta_{i,j,\ell,s} w_i w_j w_\ell w_s$$

où

$$w_i = 2\left(x_i - \frac{1}{2}\right), \qquad i = 1, 2, 4, 6, 8, ..., 20$$

et

$$w_i = 2\left(1.1\frac{x_i}{x_i + 1} - \frac{1}{2}\right), \qquad i = 3, 5, 7.$$

Soit $\mathcal{N}(0,1)$ une variable aléatoire gaussienne de moyenne nulle et d'écart-type 1. Les coefficients du premier ordre sont :

$$\beta_i = \begin{cases} 20 & \text{si } 1, ..., 10, \\ \mathcal{N}(0,1) & \text{sinon.} \end{cases}$$

Les coefficients du second ordre sont :

$$\beta_{i,j} = \begin{cases} -15 & \text{si } i, j = 1, ..., 6, \\ \mathcal{N}(0,1) & \text{sinon.} \end{cases}$$

Les coefficients du troisième ordre sont :

$$\beta_{i,j,\ell} = \begin{cases} -10 & \text{si } i, j, \ell = 1, ..., 5, \\ 0 & \text{sinon.} \end{cases}$$

Les coefficients du quatrième ordre sont :

$$\beta_{i,j,\ell,s} = \begin{cases} 5 & \text{si } i, j, \ell, s = 1, ..., 4, \\ 0 & \text{sinon.} \end{cases}$$

1.4 Reference

- M. D. Morris, 1991, Factorial sampling plans for preliminary computational experiments,Technometrics, 33, 161–174.
- Metamodel-Based Sensitivity Analysis: Polynomial Chaos Expansions and Gaussian Processes. Loïc Le Gratiet, Stefano Marelli, and Bruno Sudret. R. Ghanem et al. (eds.), Handbook of Uncertainty Quantification, 2015
- https://github.com/openturns/otmorris/tree/master
- http://openturns.github.io/otmorris/master/

1.5 Evaluer la fonction test de Morris en Python

Pour évaluer la fonction test de Morris, nous allons utiliser la classe suivante, qui implémente cette fonction en Python.

[1]:
```
import openturns as ot
```

[2]:
```
class MorrisFunction(ot.OpenTURNSPythonFunction):
    """
    The non-monotonic function of Morris f: R^20 -> R

    Reference:
      M. D. Morris, 1991, Factorial sampling plans for preliminary
      computational experiments,Technometrics, 33, 161--174.

    This code was taken from otmorris/python/src/Morris.i.

    Examples
    --------
    >>> import openturns as ot
    >>> ot.RandomGenerator.SetSeed(123)
    >>> b0 = ot.DistFunc.rNormal()
    >>> alpha = ot.DistFunc.rNormal(10)
    >>> beta =  ot.DistFunc.rNormal(6*14)
    >>> gamma =  ot.DistFunc.rNormal(20*14)
    >>> f = ot.Function( MorrisFunction(alpha, beta, gamma, b0) )
    >>> input_sample = ot.ComposedDistribution([ot.Uniform(0,1)] * 20).
↪getSample(20)
    >>> output_sample = f(input_sample)

    """

    def __init__(
        self, alpha=ot.Point(10), beta=ot.Point(14 * 6), gamma=ot.Point(20 *␣
↪14), b0=0.0
    ):
        ot.OpenTURNSPythonFunction.__init__(self, 20, 1)
        self.b0 = float(b0)
        # Check alpha dimension
        assert len(alpha) == 10
        self.b1 = [20] * 10 + list(alpha)
        # Check beta and gamma dimension
        assert len(beta) == 6 * 14
        assert len(gamma) == 20 * 14
        self.b2 = [[0] * 20] * 20
        for i in range(6):
            for j in range(6):
                self.b2[i][j] = -15.0
```

```
        # Take into account beta
        k = 0
        for i in range(6):
            for j in range(14):
                self.b2[i][j + 6] = beta[k]
                k = k + 1
        # Take into account gamma
        k = 0
        for i in range(6, 20):
            for j in range(20):
                self.b2[i][j] = gamma[k]

        # b3
        self.b3 = [[[0] * 20] * 20] * 20
        for i in range(5):
            for j in range(5):
                for k in range(5):
                    self.b3[i][j][k] = -10.0
        # b4
        self.b4 = [[[[0] * 20] * 20] * 20] * 20
        for i in range(4):
            for j in range(4):
                for k in range(4):
                    for l in range(4):
                        self.b4[i][j][k][l] = 5

    def _exec(self, x):
        assert len(x) == 20
        b1 = self.b1
        b2 = self.b2
        b3 = self.b3
        b4 = self.b4
        # X is a list, transform it into array
        X = ot.Point(x)
        w = (X - [0.5] * 20) * 2
        for k in [2, 4, 6]:
            w[k] = 2.0 * (1.1 * X[k] / (X[k] + 0.1) - 0.5)
        y = self.b0
        y = w.dot(b1)
        # Morris function
        for i in range(19):
            for j in range(i + 1, 20):
                y += b2[i][j] * w[i] * w[j]
        for i in range(18):
            for j in range(i + 1, 19):
                for k in range(j + 1, 20):
                    y += b3[i][j][k] * w[i] * w[j] * w[k]
```

```
        for i in range(17):
            for j in range(i + 1, 18):
                for k in range(j + 1, 20):
                    for l in range(k + 1, 20):
                        y += b4[i][j][k][l] * w[i] * w[j] * w[k] * w[l]

        return [y]
```

Pour utiliser cette classe, on peut utiliser le script suivant. On commence par configurer la graîne pour obtenir une fonction dont les paramètres sont prévisibles.

[5]:
```
ot.RandomGenerator.SetSeed(1)
b0 = ot.DistFunc.rNormal()
alpha = ot.DistFunc.rNormal(10)
beta = ot.DistFunc.rNormal(6 * 14)
gamma = ot.DistFunc.rNormal(20 * 14)
g = ot.Function(MorrisFunction(alpha, beta, gamma, b0))

x = ot.Point([0] * 20)

g(x)
```

[5]:
```
class=Point name=Unnamed dimension=1 values=[-67.2839]
```

1.6 Questions

1.7 Question 1 : analyse de la fonction

L'objectif de cette question est d'analyser la fonction de Morris pour observer son comportement. La solution de cette question comporte des éléments mathématiques ainsi que des éléments Python.

Questions

- Analyser la structure de la fonction et décrire les différents termes qui la composent. Décrire la linéarité et la nonlinéarité des fonctions w_i.
- Dessiner la fonction w_1 en fonction de x : qu'observez-vous ?
- Dessiner la fonction w_3 en fonction de x : qu'observez-vous ?
- Analyser les coefficients β_i, $\beta_{i,j}$, $\beta_{i,j,k}$ et $\beta_{i,j,k,\ell}$ et décrire leur loi, leur ordre de grandeur ainsi que l'influence qu'ils ont sur le modèle. Quelles variables sont susceptibles d'être peu influentes ? Quelles interactions peuvent être attendues ?

1.8 Question 2 : création de deux échantillons d'apprentissage et de validation

L'objectif de cette question est de créer les éléments nécessaires pour valider le chaos polynomial par la méthode de validation croisée.

Questions :

- Créer la variable `myDistribution`, de type `ComposedDistribution`, associée à un vecteur aléatoire uniforme entre 0 et 1 en dimension 20, de marginales indépendantes.

- Créer un échantillon d'apprentissage de taille 100. Pour cela, créer un `Sample` nommé `inputTrain` contenant un échantillon de la variable `myDistribution`. Puis créer un `Sample` nommé `outputTrain` contenant les sorties correspondantes de la fonction `g`.
- Créer un échantillon de validation de 100. Pour cela, créer un `Sample` nommé `inputTest` contenant un échantillon de la variable `myDistribution`. Puis créer un `Sample` nommé `outputTest` contenant les sorties correspondantes de la fonction `g`.

1.9 Question 3 : création d'un chaos polynomial creux avec une règle linéaire

L'objectif de cette question est de créer un chaos polynomial pour estimer les indices de Sobol'.

Questions :

- Créer la variable `multivariateBasis` contenant la base multivariée associée au vecteur aléatoire d'entrée en utilisant la classe `OrthogonalProductPolynomialFactory`.
- Créer la variable `approximationAlgorithm` contenant une instance de la classe `LeastSquaresMetaModelSelectionFactory` pour définir la méthode de sélection de modèle.
- Créer la variable `projectionStrategy` contenant une instance de la classe `LeastSquaresStrategy` pour définir la méthode de calcul des coefficients.
- Créer la variable `totalDegree=1` contenant le degré total du polynôme.
- Créer la variable `adaptiveStrategy` contenant une instance de la classe `FixedStrategy` associée au degré total `totalDegree=1`.
- Enfin, créer la variable `chaosalgo` contenant une instance de la classe `FunctionalChaosAlgorithm`. Pour cela, utiliser le plan d'expériences d'apprentissage.
- En utilisant la classe `MetaModelValidation`, dessiner le graphique de validation sur le plan d'expériences de validation. Quelle est la valeur du coefficient de prédictivité Q2 ? Peux-t-on considérer que c'est un métamodèle satisfaisant ?
- Calculer le rapport entre le nombre d'observations et le nombre de paramètres. La situation est-elle favorable pour estimer précisément les paramètres du modèle ?

1.10 Question 4: sensibilité du chaos polynomial au degré du polynôme

L'objectif de cette question est de déterminer le degré du polynôme du chaos permettant de s'approcher au mieux de la fonction test de Morris.

Questions :

- Tester les degrés totaux 1 à 4 et relever dans chaque cas la valeur du coefficient de prédictivité Q2.
- Présentez vos résultats dans une table.
- Quel degré polynomial permet d'atteindre la plus grande valeur du coefficient Q2 ?
- Quels facteurs peuvent expliquer cette sensibilité de la qualité au degré du polynôme ?

Dans la suite, vous utiliserez ce degré polynomial optimal.

1.11 Question 5 : analyse de sensibilité du chaos polynomial

L'objectif de cette question est d'utiliser les résultats de l'analyse de sensibilité du chaos polynomial pour obtenir une classification des variables d'entrée de la fonction test de Morris. Pour $i = 0, ..., 19$, on note S_i l'indice de Sobol' du premier ordre pour la variable X_i et T_i l'indice de Sobol' total.

Pour $i = 0, ..., 19$, on classifie la variable X_i dans l'une des trois catégories suivantes.

- Une variable X_i n'a pas d'effet si $T_i \leq 0.05$.
- Une variable X_i a un effet avec interaction si $T_i > 0.05$ et $T_i > S_i + 0.01$.
- Une variable X_i a un effet sans interaction si $T_i > 0.05$ et $T_i \leq S_i + 0.01$.

Questions :

- Créer un polynôme du chaos creux dont le degré est optimal au sens de la question 3.
- Utiliser les méthodes `getSobolIndex` et `getSobolTotalIndex` pour obtenir les indices de sensibilité du premier ordre et totaux.
- Utiliser la classe `DrawSobolIndices` pour dessiner les indices de sensibilité de Sobol'.
- Commenter votre graphique.
- A l'aide d'une boucle `for`, parcourir la liste des variables et utiliser les indices de sensibilité du premier ordre et total pour classer chaque variable.
- Commenter votre résultat.

1.12 Question 6 : chaos polynomial avec règle d'énumération hyperbolique

L'objectif de cette question est de tenter d'améliorer la qualité de prédiction du métamodèle en utilisant une règle d'énumération hyperbolique.

Questions :

- Utiliser la classe `HyperbolicAnisotropicEnumerateFunction` avec le paramètre de quasi-norme $q = 0.6$.
- Créer un polynôme de chaos de degré 1 associé à cette règle d'énumération.
- Tester les degrés totaux 1 à 10 et relever dans chaque cas la valeur du coefficient de prédictivité Q2.
- Présentez vos résultats dans une table.
- Quel degré polynomial permet d'atteindre la plus grande valeur du coefficient Q2 ?
- Comment expliquez-vous les changements dans vos résultats ?

Dans la suite, vous utiliserez ce degré polynomial optimal.

1.13 Question 7 : analyse de sensibilité du chaos polynomial

Questions :

- De la même manière que dans la question 4, classer les variables d'entrée en fonction de leurs indices de senbilité.
- Commentez votre résultat.

1.14 Question 8 : recherche d'un groupe de variables optimal

L'objectif de cette question est d'identifier un groupe de variables ayant peu d'influence sur la sortie. L'intérêt de cette démarche est de diminuer le nombre de variables d'entrée, en les fixant à leur valeur moyenne. En conséquence, la variance de la sortie diminue.

Ainsi,

- si on ne fixe aucune variable d'entrée, la variance de la sortie est préservée, mais le nombre de variables d'entrée est grand,
- si on fixe trop de variables d'entrée, le nombre de variables d'entrée est réduit, mais la variance de la sortie est réduite.

L'objectif est donc de parvenir à un compromis acceptable.

Questions :

- Pour le groupe de variables d'indices [1, 5, 13, 18], les instructions suivantes calculent l'indice de sensibilité du total de ce groupe :

```
group = [1, 5, 13, 18]
chaosSI = ot.FunctionalChaosSobolIndices(chaosResult)
T = chaosSI.getSobolGroupedTotalIndex(group)
```

Quelle valeur d'indice de sensibilité du premier ordre obtenez-vous ?

- La fonction `findMinimumTotalOrderNotInGroup` suivante retourne l'indice d'une variable qui ne se trouve pas déjà dans le groupe et possédant le plus petit indice total.

[6]:
```
def findMinimumTotalOrderNotInGroup(total_order, isingroup):
    """
    Returns the indice of the variable not in the group which
    has the minimum total order indice.
    isingroup : a boolean array with size dim.
    isingroup[i] is True if i is in the group.
    """
    Tmin = 2.0
    dim = len(total_order)
    imin = dim
    for i in range(dim):
        if not isingroup[i]:
            T = total_order[i]
            if T < Tmin:
                imin = i
                Tmin = T
    return imin
```

- Utiliser itérativement la fonction `findMinimumTotalOrderNotInGroup` pour rechercher le groupe de variables associé à un indice total inférieur à 0.05.
- Quel est le nombre de variables et la liste des variables qui doivent être retenues dans ce modèle sans diminuer excessivement la variance de la sortie ?

1.15 Question 9 : Synthèse

Questions

- Quelles variables peuvent être remplacées par des constantes sans diminuer la variance de la sortie ?
- Quelle est la dimension de la fonction dans ce cas ?
- Peut-on réduire la dimension davantage ?