SORBONNE UNIVERSITÉ

Cryptanalyse du chiffrement de Vigenère

Version du 4 février 2024

TME

1 Objectif du projet

L'objectif de ce TME est d'implémenter plusieurs outils permettant de cryptanalyser un texte chiffré avec le chiffrement de Vigenère.

Le projet est à réaliser en Python. Au fur et à mesure des questions vous allez écrire plusieurs fonctions qui vont vous permettre d'analyser un texte chiffré, de retrouver sa clé et de le déchiffrer. À la fin du projet, votre programme devra être capable de casser la plupart des messages chiffrés avec Vigenère.

Ce projet, qui va durer plusieurs semaines, fera l'objet d'une note qui compte dans la note de contrôle continu de l'UE. Pensez à bien respecter toutes les consignes !

2 Le chiffrement de Vigenère

Le chiffrement de Vigenère fonctionne de la manière suivante. Le message est composé de lettres (majuscules, sans accents, sans espaces ni ponctuation). La clef est un tableau contenant des entiers entre 0 et 25. On pourra supposer pour les textes étudiés ici que la longueur de la clef est limités à 20.

On écrit les entiers de la clef sous le message. Si la clef est plus courte que le message, on répète les mêmes entiers. Pour chaque lettre du message, on calcule son rang dans l'alphabet (`A` = 0, `B` = 1, ...), et on l'ajoute à l'entier de la clef correspondant. On fait l'addition, modulo 26. Puis on remplace chaque nombre obtenu par la lettre correspondante dans l'alphabet.

Par exemple, le message `CHIFFREDEVIGENERE` chiffré avec la clef `[2, 17, 24, 15, 29, 14]` donne le message chiffré `EYGUYFGUCKBUGECGX`, comme détaillé ci-dessous.

```
   C  H  I  F  F  R    E  D  E  V  I  G    E  N  E  R  E
   2  7  8  5  5 17    4  3  4 21  8  6    4 13  4 17  4
+  C  R  Y  P  T  O    C  R  Y  P  T  O    C  R  Y  P  T
   2 17 24 15 19 14    2 17 24 15 19 14    2 17 24 15 19
-------------------------------------------------------
=  4 24  6 20 24  5    6 20  2 10  1 20    6  4  2  6 23
=  E  Y  G  U  Y  F    G  U  C  K  B  U    G  E  C  G  X
```

L'objectif de ce TME est d'étudier différentes approches qui permettent de retrouver le message clair à partir d'un message chiffré, sans connaître la clef. Par exemple, normalement, à la fin du projet, vous serez capable de déchiffrer le texte suivant.

```
PWIAPKJCYLNSEGIGYYUGXATDGOUKHJFVRMJKBZVSTZHFUWAZJRBGXHPZNDULTVNUQHXHBVRKKYLX
BWUMYGKKASHYIONYQNTZFIEGYYEFRNKSOOVZGSHKQNNSQHTWTOYXQPTEJCUOXDFZVLQWEMRCLJNG
SKZSHXUXANPYXGBSRDYVRTCREILRBTFNEZEVARJNNBQGHNHLNXBCFVLQFRHAGBITCAEKISVSNAIJ
EEDRGPBJFAGLHVPVXVCZGHFTRKJBVXWDQXXHUKVVULFHUUU
```

3 Plan des séances

Ce TME va vous occuper pendant plusieurs séances. Le travail est à réaliser par binôme. Vous devez garder le même binôme pour toute la durée du projet.

Dans ce TME, vous allez programmer trois approches qui permettent de cryptanalyser un texte chiffré avec le chiffre de Vigenère. Les grandes étapes de ce TME sont les suivantes :

0. Mise en place du projet.
1. Coder des fonctions permettant de chiffrer et déchiffrer avec le code de César et le code de Vigenère.
2. Première approche : avec l'indice de coïncidence.
3. Deuxième approche : avec l'indice de coïncidence mutuelle.
4. Troisième approche : avec les correlations de Pearson.

Remarque : ce projet est incremental. Il est inutile d'essayer de passer à l'étape suivante si vous n'avez pas fini l'étape en cours.

Bonne pratique : documenter son code. Dans tout projet de taille conséquente, il ne faut pas écrire du code pour soi mais pour les autres programmeurs participant au projet, ou pour les futurs utilisateurs de l'application. La documentation du code occupe donc une place fondamentale.

En Python, les commentaires simples commencent par `#`. Les commentaires spéciaux (sur plusieurs lignes) commencent et terminent par `"""`. En particulier il est important de décrire ce que font les différentes fonctions juste après la ligne de définition.

⇒ **Nous vous incitons vivement à commenter vos différentes fonctions, à faire des commentaires au sein des programmes et à utiliser les commentaires pour toute forme de remarque qui peut vous sembler pertinente.**

3.1 Rendu du TME (OBLIGATOIRE)

Ce TME fera l'objet d'une note qui comptera dans le contrôle continu. Pour que vos enseignants puissent évaluer votre travail, il est impératif de soumettre votre travail.

Chaque semaine, il est obligatoire de rendre le TME à votre chargé de TME en fin de la séance. Si vous le souhaitez, vous pouvez aussi rendre **une seconde version améliorée avant le début du TME suivant**.

Il est impératif de créer un *tag* pour chaque rendu, et de réaliser au moins un rendu par seance. Ces rendus réguliers (code source, validation des tests, réponses aux questions) sont évalués dans le cadre du contrôle continu.

4 Cryptanalyse du chiffre de Vigenère

4.1 Retour sur le TME précédent

Question 1. Pour effectuer cette cryptanalyse, vous aurez-besoin de vous référer à la distribution des fréquences dans un texte français. Pour cela, nous utiliserons une variable globale `freq_FR` contenant les fréquences calculées sur un long texte français. Utilisez votre fonction du TME 1 pour modifier la valeur de `freq_FR`. Vous pouvez par exemple calculer les fréquences d'apparition des lettres dans le texte *Germinal*, de Zola, fourni sur Moodle.

Question 2. Il pourra être utile d'avoir une fonction qui prend une chaîne de caractères et effectue un décalage de chaque lettres de n positions dans l'alphabet. Cela correspond exactement au chiffrement de César vu au TME 1. Écrire le corps des fonctions `chiffre_cesar` et `dechiffre_cesar`. Attention, ici la clef est un entier.

⇒ Désormais, votre programme devrait passer le premier test `test-1-cesar.py`. Essayez de lancer le test localement. Si le premier test est valide, propagez votre programme sur le serveur en suivant les instructions du `README` du Gitlab.Vérifiez que le premier test est bien validé sur la plateforme GitLab. En cas de problème, demandez à votre enseignant. C'est le bon moment pour comprendre comment utiliser `git`.

4.2 Pour bien commencer : chiffrer et déchiffrer

Avant de se lancer dans la cryptanalyse, il faut déjà pouvoir chiffrer et déchiffrer avec la méthode de Vigenère. La première partie de ce TME consiste à écrire un programme qui chiffre et déchiffre avec la méthode de chiffrement de Vigenère.

Question 3. Écrire le corps des fonctions `chiffre_vigenere` et `dechiffre_vigenere`.

⇒ Désormais, votre programme devrait passer le deuxième test. Si c'est le cas, propagez votre programme sur le serveur en suivant les instructions du `README` du Gitlab.

4.3 Analyse de fréquences, indice de coïncidence

Question 4. On le sait depuis Al Kindi, il est toujours utile de regarder la fréquence d'apparition de chaque lettre dans un texte. Vous avez déjà utilisé cet outil pour la cryptanalyse du chiffre de César. Écrire le corps de la fonction `freq` qui, étant donné une chaîne de caractères, renvoie un tableau avec le nombre d'occurences de chaque lettre de l'alphabet. Attention, on demande juste le nombre d'occurence, le tableau doit contenir des entiers, ici on ne divise pas par le nombre total de lettres.

Question 5. Dans le cas d'un chiffrement par décalage (comme pour César) il suffit de trouver la lettre qui apparaît le plus grand nombre de fois. Écrire le corps de la fonction `lettre_freq_max` qui, étant donné une chaîne de caractères, renvoie la position dans l'alphabet de la lettre qui apparaît le plus grand nombre de fois dans le texte. Si plusieurs lettres apparaissent autant de fois, on renverra celle qui apparaît la première dans l'ordre alphabétique.

Un outil précieux pour la cryptanalyse du chiffre de Vigénère est l'indice de coïncidence (cf. exercices 6 et 7 du TD). Cet outil a été proposé par le cryptologue américain Friedman et est invariant par chiffrement mono-alphabétique.

L'indice de coïncidence d'un texte est :

$$\sum_{i=A}^{i=Z} \frac{n_i(n_i-1)}{n(n-1)},$$

où n_i est le nombre d'occurences de la lettre i et n le nombre total de lettres.

Question 6. Écrire le corps de la fonction `indice_coincidence` qui prend en entrée un tableau qui correspond aux occurences des lettres d'un texte (typiquement la sortie de la fonction `freq`) et renvoie l'indice de coïncidence.

Comme vu dans la question 4 de l'exercice 8 du TD, si la clef est de longueur k, en considérant une lettre sur k du texte chiffré (ce qu'on appelle une colonne), toutes ces lettres sont décalées du même nombre de positions. Chiffrer avec Vigenère correspond à chiffrer chaque colonne du texte avec un chiffre de César, mais avec des clés différentes. Donc si l'on regarde une colonne, l'indice de coïncidence doit être proche de celui d'un texte français (0.07), puisque le chiffre de César conserve l'indice de coïncidence.

Question 7. Pour trouver la taille de la clef, on teste toutes les tailles de clef possibles (on suppose que la clef cherchée est au plus de longueur 20). On découpe le texte en colonnes et on calcule la moyenne de l'indice de coïncidence de chaque colonne. Si la moyenne est > 0.06, c'est qu'on a trouvé la bonne taille de clef. Implementer cette stratégie dans la fonction `longueur_clef`.

⇒ Désormais, votre programme devrait passer le troisième test. Si c'est le cas, propagez votre programme sur le serveur en suivant les instructions du `README` du Gitlab.

4.4 Table de décalages

Question 8. Une fois que la taille de la clef est identifiée, la stratégie consiste à raisonner indépendement par colonne. Pour chaque colonne on suppose que la lettre qui apparaît le plus correspond au chiffré de la lettre `E`. On en déduit le décalage qu'a subi chaque colonne. Cela correspond bien à la clef utilisée pour chiffrer avec le chiffre de Vigenère. Écrire la fonction `clef_par_decalages` qui prend un texte et la taille de la clé et renvoie la clé sous forme d'une table de décalages.

⇒ Désormais, votre programme devrait passer le quatrième test. Si c'est le cas, propagez votre programme sur le serveur en suivant les instructions du `README` de Gitlab.

4.5 Première cryptanalyse

Question 9. On a alors tous les outils pour effectuer une première forme de cryptanalyse. On déduit la longueur de la clef avec l'indice de coïncidence, puis on récupère la clef en observant le décalage de chaque colonne. Écrire le corps de la fonction `cryptanalyse_v1` qui utilise cette methode pour cryptanalyser un texte. Lancer le test 5. Combien de textes sont correctement cryptanalysés ? Comment expliquez-vous cela ? Expliquez cela en commentaire de la fonction.

⇒ Désormais, votre programme devrait passer le cinquième test. Si c'est le cas, propagez votre programme sur le serveur en suivant les instructions du `README` de Gitlab.

4.6 Indice de coïncidence mutuelle

Plutôt que regarder uniquement la lettre la plus présente, on propose une seconde approche, plus subtile, qui fonctionnera mieux pour de petits textes. Celle-ci utilise d'indice de coïncidence mutuelle. Cet outil, étudié dans l'exercice 9 du TD, permet de distinguer lorsque deux colonnes ont un décalage identique.

L'indice de coïncidence mutuelle (ICM) de deux textes est :

$$\sum_{i=A}^{i=Z} \frac{n_{1,i} n_{2,i}}{n_1 n_2},$$

où $n_{T,i}$ est le nombre d'occurences de la lettre i dans le texte T et n_T le nombre total de lettres dans le texte T.

Question 10. Écrire le corps de la fonction `indice_coincidence_mutuelle` qui prend en entrée deux tableaux qui correspond aux fréquences des lettres de deux textes (typiquement la sortie de la fonction `freq`) ainsi qu'un entier d et renvoie l'indice de coïncidence du texte 1 et du texte 2 qui aurait été décalé de d positions (comme par un chiffrement de César).

Question 11. Une fois que la taille de la clef est identifiée, la stratégie consiste à retrouver le décalage relatif de chaque colonne par rapport à la première colonne, qui sert de référence. Pour chaque colonne on calcule l'ICM de la première colonne et de cette colonne qu'on aurait décalée de d positions. Le d qui maximise l'ICM correspond au décalage relatif par rapport à la première colonne du texte. Écrire la fonction `tableau_decalages_ICM` qui prend un texte et une longueur de clef supposée, et calcule pour chaque colonne son décalage par rapport à la première colonne. La sortie est un tableau d'entiers, dont la première valeur est toujours 0 (puisque la première colonne a un décalage nul par rapport à elle-même).

⇒ Désormais, votre programme devrait passer le sixième test. Si c'est le cas, propagez votre programme sur le serveur en suivant les instructions du `README` de Gitlab.

4.7 Deuxième cryptanalyse

Question 12. On a alors tous les outils pour effectuer une deuxième forme de cryptanalyse. On déduit la longueur de la clef avec l'indice de coïncidence, puis on calcule les décalages relatifs de chaque colonne par rapport à la première colonne en utilisant l'ICM. On décale chaque colonne pour l'aligner avec la première colonne. Le texte obtenu est alors l'équivalent d'un texte chiffré avec César. On le déchiffre donc comme un texte chiffré avec César, en identifiant la lettre la plus fréquente. Écrire le corps de la fonction `cryptanalyse_v2` qui utilise cette methode pour cryptanalyser un texte. Lancer le test 7. Combien de textes sont correctement cryptanalysés ? Comment expliquez-vous cela ? Expliquez cela en commentaire de la fonction.

⇒ Désormais, votre programme devrait passer le septième test. Si c'est le cas, propagez votre programme sur le serveur en suivant les instructions du `README` de Gitlab.

4.8 Corrélation de Pearson

Une troisième approche consiste à utiliser la corrélation entre histogrammes de fréquence. Le coefficient de corrélation linéaire de deux variables aléatoires X et Y, défini par Pearson, est un indicateur de corrélations. Plus il est proche de 1, plus les variables sont fortement corrélées positivement.

Il se calcule avec la formule

$$\mathrm{Cor}(X,Y) = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \bar{X})^2}\sqrt{\sum_{i=1}^{n}(Y_i - \bar{Y})^2}},$$

où $\bar{Z}$ est l'espérance de la variable Z, *i.e.* $\bar{Z} = \frac{1}{n}\sum_{i=1}^{n} X_i$.

Question 13. Compléter la fonction `correlation` qui prend deux listes de même longueur, correspondant à deux variables aléatoires, et renvoie la valeur de la corrélation entre les deux.

Question 14. Écrire la fonction `clef_correlations` qui, étant donné un texte chiffré et une taille de clef, calcule pour chaque colonne le décalage qui maximise la corrélation avec un texte français. Vous pourrez utiliser le tableau des fréquences d'un texte français `freq_FR` défini à la question 1 comme référence. La fonction doit renvoyer un tuple composé de deux éléments : la moyenne sur les colonnes des corrélations maximales obtenues et un tableau contenant pour chaque colonne le décalage qui maximise la corrélation.

⇒ Désormais, votre programme devrait passer le huitième test. Si c'est le cas, propagez votre programme sur le serveur en suivant les instructions du `README` de Gitlab.

4.9 Troisième cryptanalyse

Question 15. On a alors tous les outils pour effectuer une troisième forme de cryptanalyse. On test pour chaque taille de clef (ici on suppose la clef de taille ≤ 20), on calcule les décalages qui maximisent la corrélation de Pearson à l'aide de la fonction `clef_correlations`. On suppose que la bonne taille de clef est celle qui maximise la moyenne de corrélations sur les colonnes. Il reste alors à appliquer le déchiffrement de Vigenère avec la fonction `dechiffre_vigenere` pour récupérer le texte clair. Écrire le corps de la fonction `cryptanalyse_v3` qui utilise cette methode pour cryptanalyser un texte. Lancer le test 9. Combien de textes sont correctement cryptanalysés ? Quels sont les caractéristiques des textes qui échouent ? Comment expliquez-vous cela ? Expliquez cela en commentaire de la fonction.

⇒ Désormais, votre programme devrait passer tous les tests. Si c'est le cas, propagez votre programme sur le serveur en suivant les instructions du `README` de Gitlab.