Cryptanalyse du chiffrement de Vigenère

Auteurs
Matthieu Lequesne

Version du 3 février 2022

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
   3  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
-------------------------------------------------------
=  5 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.

Pour ce projet, vous allez utiliser le logiciel de gestion de versions `git`. L'utilisation de `git` est détaillée dans la section 4.

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 et prise en main de `git`.
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.**

4 Utilisation de GitLab

Cette section et les suivantes concernant GitLab sont largement inspirées des documents d'Antoine Miné pour le cours 3I002.

Pour ce TME, vous allez utiliser le logiciel `git` et la plateforme GitLab. Gitlab est un gestionnaire de projets logiciels. `git` est un gestionnaire de versions distribué. GitLab centralise votre projet sur un serveur externe. Il vous permet de garder un historique des modifications (visible avec une interface web), de vous synchroniser avec votre binôme et de faire le rendu du projet au chargé de TME. Nous y avons ajouté une fonction d'intégration continue : toute propagation des modifications locales vers le serveur GitLab exécutera automatiquement une batterie de tests pour valider votre nouvelle version. Note : nous utilisons ici un serveur GitLab privé, dédié au cours : `https://stl.algo-prog.info`, et pas le GitLab du site public `gitlab.com`.

Nous allons travailler dans un projet nommé `Vigenere` pour toute la durée du TME 2 (séances 3 à 5).

4.1 Mise en place du projet

Pour mettre en place le projet GitLab du TME, vous devrez suivre les étapes suivantes. Faites attention à bien suivre les étapes dans l'ordre. Addressez-vous au chargé de TME en cas de problème.

1. **Vérification de la place disponible (quota).**
   Si une des étapes ci-dessous échoue, cela peut être dû à un manque d'espace sur votre compte. Avant toutes choses, utilisez la commande `quota` dans un terminal pour déterminer si votre compte dispose d'espace disponible. En particulier, si la première colonne (*blocks*) est égale à la colonne (*limit*) et une étoile (`*`) apparaît, il ne reste plus de place sur le disque ; vous devez absolument libérer de l'espace avant de continuer.
2. **Connexion au GitLab.**
   Lancez un navigateur, allez sur la page du serveur GitLab : `https://stl.algo-prog.info` et connectez-vous. Normalement, vous avez du recevoir un *email* du serveur vous précisant les informations de connexion. Si vous n'avez pas reçu cet *email*, prévenez les enseignants. Ne vous inscrivez pas directement sur le serveur, vous

n'aurez pas alors automatiquement accès aux projets des TME du cours.
Votre *username* pour la connexion est votre numéro d'étudiant. Lors de votre première connexion, vous devrez choisir un mot de passe.

3. **Configuration du proxy de navigation** (valable uniquement à la PPTI).
   Si la page web du serveur GitLab n'est pas accessible, vous devez configurer le *proxy* (serveur mandataire) de votre navigateur. Choisissez, pour les protocoles HTTP et HTTPS, le serveur `proxy.ufr-info-p6.jussieu.fr` et le numéro de port 3128.
4. **Configuration de proxy pour git** (valable uniquement à la PPTI).
   Afin d'éviter les erreurs d'authentification, vous devez ouvrir un terminal et y taper les commandes suivantes :
   ```
   git config --global http.sslVerify false
   git config --global http.proxy https://proxy.ufr-info-p6.jussieu.fr:3128
   git config --global https.proxy https://proxy.ufr-info-p6.jussieu.fr:3128
   ```
5. **Configuration des notifications de GitLab.**
   Il peut être utile de réduire les notifications envoyées par *email*. Pour cela, dans le menu « Settings » de GitLab, choisissez l'option « Notifications », ouvrez le menu « Global notification level » et cliquez sur « Custom ». Une liste apparaît, et vous pouvez désélectionner certaines notifications. Nous conseillons en particulier de désélectionner « Failed pipeline » pour éviter d'être submergé de notifications lors de l'intégration continue (voir sous-section 4.3).
6. ***Fork* du squelette de projet GitLab.**
   Vous êtes automatiquement membre du groupe `CRYPTO-1819`. Ce groupe contient le projet `Vigenere`, auquel vous pouvez accéder en lecture seule. Le projet est un squelette que vous allez compléter. Il fournit quelques sources : exemples, tests, interfaces, etc. Pour réaliser le TME et compléter ce projet, vous devez d'abord en faire **une copie** (*fork*) qui sera **privée** à votre binôme et vous et sur laquelle vous allez travailler.

   Note : assurez-vous que votre fenêtre de navigateur occupe tout l'écran. Si la fenêtre est trop petite, certains menus de GitLab sont remplacés par des icônes, et vous ne trouverez plus les noms des menus tels qu'indiqués dans la suite de l'énoncé.

   Pour chaque binôme, **un seul d'entre vous** devra faire un *fork* de ce squelette de projet, puis y ajouter son camarade de binôme, et ainsi partager le projet sous GitLab. Pour cela :
   — Dans l'onglet « Projects > Your projects », sélectionnez le projet `CRYPTO-1819/Vigenere`, cliquez sur le bouton « Fork » puis cliquez sur votre nom. Ceci crée le projet personnel privé nommé *username*`/Vigenere` (où *username* est votre numéro d'étudiant). Vous travaillerez désormais dans ce projet, où vous avez les droits d'écriture.
   — Dans l'onglet « Projects > Your projects », sélectionnez ce projet privé. Il doit avoir pour nom *username*`/Vigenere` et porter la mention « Forked from `CRYPTO-1819/Vigenere` ». Cliquez sur « Settings » dans le menu de gauche, puis « Members ». **Invitez votre binôme** en lui donnant pour rôle « Maintainer ».

   Si c'est votre binôme qui a créé le projet et vous y a ajouté avec les bons droits, vous devriez le trouver dans l'onglet « Projects > Your projects » sous le nom *partnername*`/Vigenere` où *partnername* est le numéro d'étudiant de votre binôme.
7. **Ajout du chargé de TME à votre projet GitLab.**
   Dans votre projet personnel sur GitLab, *username*`/Vigenere`, invitez également votre chargé TME avec le rôle « Maintainer » (même procédure que ci-dessus). Si vous n'ajoutez pas votre chargé de TME, celui-ci ne pourra pas suivre l'avancement de votre travail, ni vous donner une note.
8. **Faire une copie locale de votre projet.**
   Pour travailler sur votre projet, vous devez effectuer une copie locale sur votre machine. Tous les membres du projet peuvent faire une copie locale. Ensuite, vous devez régulièrement synchroniser la version sur le serveur avec le travail que vous avez effectué localement. Cela vous permet à plusieurs personnes de travailler simultanément, sur des ordinateurs différents, et de partager et d'échanger les sources développées.
   Pour effectuer une copie locale de votre projet, vous devez d'abord récupérer l'adresse du projet. Vous la trouverez sur la page GitLab de votre projet, en cliquant sur l'icône « Clone » à côté du bouton « Fork ». Copiez la chaîne de caractères sous « Clone with HTTPS ». Elle doit avoir la forme : `https://stl.algo-prog.info/`*username*`/Vig`

(où *username* est le numéro de l'étudiant qui a fait le *fork* ; attention, l'adresse doit se terminer en `.git`).
Prenez garde à bien importer votre projet de binôme, `username/Vigenere`, et pas `CRYPTO-1819/ Vigenere`. Vous ne pourrez pas travailler dans ce dernier, qui est en lecture seule.
Utilisez votre terminal pour vous rendre dans le repertoire où vous souhaitez copier le projet. Puis entrez la commande `git clone adresse-du-projet`, où *adresse-du-projet* est l'adresse que vous avez récupérée.
Vous devez entrer un nom d'utilisateur et le mot de passe associé. Le nom d'utilisateur est votre *username* sur le serveur GitLab, c'est-à-dire votre numéro d'étudiant (i.e., la personne qui fait l'import, pas forcément la personne qui a fait le *fork*) ; le mot de passe est celui que vous avez choisi en vous connectant au serveur GitLab la première fois.
Cette action va créer un répertoire `Vigenere` contenant votre copie locale du projet.

Le projet contient plusieurs fichiers mais **vous ne devez modifier qu'un seul fichier : `cryptanalyse-vigenere.py`**. Les autres fichiers servent pour les tests. Voir la section 4.3.

Vous pouvez désormais commencer à travailler sur votre copie locale du projet.

4.2 Synchronisation avec le serveur GitLab

Après avoir exécuté `git clone` en début de TME, nous avons travaillé sur une copie locale du projet. Il est nécessaire de synchroniser périodiquement votre projet local avec le projet GitLab pour :

— communiquer vos fichiers à l'enseignant pour le rendu (celui-ci a accès aux fichiers sous GitLab, mais pas à ceux sur votre compte local),
— vous synchroniser avec votre binôme,
— éventuellement synchroniser des copies locales sur plusieurs ordinateurs,
— garder une trace des modifications et pouvoir éventuellement revenir à une version précédente en cas d'erreur.

Les opérations utiles sont donc la propagation d'une copie locale vers le serveur (*push*) et depuis le serveur vers une copie locale (*pull*).

Vous pouvez consulter l'état des fichiers sur le serveur GitLab en utilisant le site web `https://stl.algo-prog.info`. Vous y trouverez la dernière version des fichiers et l'historique des modifications. Vous pourrez en particulier vérifier que le projet a bien été synchronisé pour le rendu de TME.

4.2.1 Git en ligne de commande

Dans un terminal, placez-vous dans le répertoire `Vigenere` contenant votre projet. Les commandes les plus utiles sont :

— `git status` pour connaître l'état actuel de vos fichiers, en particulier ce qui a été modifié depuis la dernière synchronisation avec le serveur ;
— `git add fichiers` pour indiquer les fichiers ajoutés ou modifiés localement ;
— `git commit -m "mon message"` pour enregistrer localement les ajouts ou modifications des fichiers spécifiés par `git add`, où *mon message* est un commentaire de votre part qui spécifie les modifications ;
— `git push` pour effectivement propager l'enregistrement local vers le serveur ;
— `git pull` pour rapatrier localement les modifications depuis le serveur.

Le système `git` est décrit dans le livre en ligne : `https://git-scm.com/book/en/v2`.

4.2.2 Bonnes pratiques.

C'est une bonne idée d'anticiper les conflits en **commençant toute session de travail par un *pull***, pour repartir avec les dernières versions des fichiers disponibles sur le serveur, et en **terminant toute session de travail par un *push***, pour que vos modifications locales soient envoyées sur le serveur et puissent être importées par votre binôme ou vous-même sur un autre ordinateur.

4.2.3 Gestion des conflits

Si des modifications ont été faites sur le serveur (par exemple par une propagation, *push*, de votre camarade) depuis votre dernier *pull*, vous ne pourrez pas propager vos modifications locales directement ; `git` refusera avec une erreur.

En effet, cela provoquerait des conflits entre deux nouvelles versions d'un fichier. `git` vous force à résoudre les conflits localement, avant de propager vos fichiers corrigés vers le serveur :

- Faites d'abord un *pull*.
- `git` s'efforce de fusionner les modifications locales avec celles présentes sur le serveur, mais il a pu faire des erreurs ; vous devez examiner chaque fichier et corriger à la main les problèmes causés par la fusion. Les zones non fusionnées sont identifiées par des balises `<<<<<<`, `------` et `>>>>>>` dans votre source Python. `git` vous indique de cette manière les deux versions disponibles (version locale et dernière version disponible sur le serveur). Il s'agit souvent de choisir une de deux versions, en supprimant les lignes redondantes et les balises.
- Après suppression de tous les conflits, vous devez faire un *commit* avec les fichiers concernés.
- Vous pouvez enfin faire un *push*.

4.3 Tests et intégration continue sous GitLab

L'intégration continue est une pratique de développement logiciel consistant à s'assurer que, à chaque instant, le dépôt est correct et passe tous les tests. Le serveur GitLab est configuré pour l'intégration continue : **après chaque propagation de votre copie locale vers le serveur (*push*), des tests sont exécutés automatiquement**.

Vous pouvez consulter le résultat des tests sur le serveur GitLab `https://stl.algo-prog.info` en cliquant sur votre projet, puis dans le menu à gauche sur « CI / CD > pipelines ». Les tests de la dernière version apparaissent en haut. Un icône « V » vert ou une croix rouge indique l'état du test (un croissant ou un symbole pause indique que le test est en cours ou en attente, il faut donc patienter). Cliquer sur l'icône dans la colonne « *Status* » permet de voir l'ensemble des classes de test. Cliquer sur un nom de test vous donne un rapport complet de test, indiquant en particulier quelles méthodes de test ont échoué, et avec quelles erreurs.

Le chargé de TME a accès aux rapports de tests sur le serveur GitLab, ce qui lui permet d'évaluer votre rendu de TME.

Le serveur est configuré pour exécuter tous les tests du TME. **Tant que vous n'avez pas programmé toutes les fonctions demandées, de nombreux tests vont échouer. Vous ignorerez donc au départ les tests liés aux questions que vous n'avez pas encore traitées.**

Vous pouvez effectuer les tests au niveau local, avant de propager votre version du projet. Pour lancer les tests localementm il suffit d'utiliser la commande `./test-all.sh`. Vous pouvez également executer chaque test séparément avec la commande `python test-N-*.py` où N est le numéro du test que vous souhaitez effectuer.

⇒ **Nous vous recommandons d'effectuer les tests au niveau local avant de propager votre version du projet, afin de vous assurer qu'il ne reste pas d'erreurs.**

Le répertoire `data` contient 100 textes chiffrés avec Vigenère. Pour chaque texte, on donne le texte clair `.plain`, le texte chiffré `.cipher` et la clef `.key`. Cela peut vous être utile pour tester votre code.

4.4 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**.

Le rendu se fait en propageant vos modifications vers le serveur GitLab, comme indiqué à la sous-section 4.2, et en y associant un *tag*. Pour cela :

- Connectez-vous sur la page de votre projet sous `https://stl.algo-prog.info`.
- Assurez-vous que votre chargé de TME est membre de votre projet, avec le rôle « Maintainer ».
- Vérifiez que toutes les classes demandées sont bien présentes sous GitLab et bien synchronisées avec le projet local.
- Vérifiez également que les tests unitaires du TME lancés par l'intégration continue sur le serveur GitLab se sont exécutés correctement (voir 4.3).
- Dans le menu de gauche, sélectionnez « Repository > Tags » et cliquez sur « New Tag ».
- Donnez un nom à votre *tag* : « rendu-fin-seance1 » ou « rendu-apres-seance1 », selon qu'il s'agit d'un rendu partiel en fin de la séance 1 ou bien d'un rendu du TME plus tard dans la semaine.

— Cliquez sur « Create tag ».
— En cas d'erreur, il est toujours possible de créer un nouveau *tag*. Pensez à donner un nom explicite.

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.

5 Cryptanalyse du chiffre de Vigenère

5.1 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 *Germilal*, 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 de la section 4.2. Verifiez 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`.

5.2 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 de la section 4.2.

5.3 Analyse de fréquence de indice de coïncidence

Question 4. On le sait depuis Alkindi, 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 7 et 8 du TD). Cet outil a été proposé par le cryptologue américain Friedman et est invariant par chiffrement monoalphabétique.

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 fréquences 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 de la section 4.2.

5.4 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 table de décalages correspondant et déchiffre le texte.

⇒ Désormais, votre programme devrait passer le quatrième test. Si c'est le cas, propagez votre programme sur le serveur en suivant les instructions de la section 4.2.

5.5 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 de la section 4.2.

5.6 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 de la section 4.2.

5.7 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 de la section 4.2.

5.8 Correlations de Pearson

Une troisième approche consiste à utiliser les corrélations 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 fontement 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 de la section 4.2.

5.9 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 de la section 4.2.