Sommaire:

Objectif du projet :

Le but de ce projet est de déterminer la distance entre le site de fixation de la protéine FMRP et les sites d'épissage, afin de savoir si elle peut jouer un rôle significatif dans le processus d'épissage des ARNm. Le programme prendra en entrée deux fichiers data like (.csv, .tsv, .xlsx).

- Fichier de la protéine:

Ce fichier contiendra les informations concernant les sites de fixation de la protéine sur les ARNm. Il se présentera soit sur la forme de données ADN (~ARN pré-messager) soit sur la forme de données ARN. L'utilisateur devra indiquer de quel type il s'agit. Les colonnes Start/End et Sequence sont complémentaires mais l'une peut substituer l'autre. Il ne sera donc pas nécessaire de connaître la séquence si les positions sont connues et inversement.

Table 1 : Colonnes du tableau de données à ADN

ID gene (ensembl)	Chromosome	Brin	Start	End	Sequence	Assemblage

Table 2 : Colonne du tableau de données à ARN

ID transcrit (ensembl)	Chromosome	Brin	Start	End	Sequence	Assemblage

- Fichier des sites d'épissage:

Ce fichier contiendra les positions des sites d'épissage alternatifs qui ont été identifiés. Le fichier d'entrée reste encore à déterminer, mais une structure similaire au fichier de la protéine est envisageable (peut-être sans la colonne séquence).

Une fois les deux fichiers renseignés, il faudra convertir les coordonnées en coordonnées ADN, si cela n'est pas le cas, afin de pouvoir les comparer. Par la suite, nous pourrons calculer les distances. Calculer toutes les distances entre tous les sites de fixations et tous les variants risque d'être trop gourmand (brute force inenvisageable), il faudra donc élaborer une stratégie un peu plus fine. Si la protéine se fixe à cheval entre deux exons la méthode pour calculer la distance entre les sites d'épissage sera assez différente.

Nous pensons calculer la distance entre le début et la fin de la la séquence de fixation de FMRP et entre le début et la fin de la séquence d'épissage (4 calculs au total). Cependant, les introns pourraient poser problème donc il faudra les enlever pour ne pas

fausser les distances. Un graphique représentant FMRP et sa fixation et la position du site d'épissage.

Conversion coordonnées ARN-ADN méthode:

Passer par les serveurs Ensembl est la solution la plus précise mais cela implique un trop long temps d'attente entre chaque requête. Le module python pyensembl permet de manipuler facilement les fichiers GTF et multi fasta contenant le génome de l'espèce en question. pyensembl permet également de télécharger simplement n'importe quel génome ou fichier GTF selon le numéro de "Release" d'Ensembl. L'assemblage 38 du génome de la souris était contenu dans la release 102. Une fois ces données chargées, nous pouvons en extraire les transcrits grâce à leurs IDs. Une fois les transcrits récupérés, nous pouvons obtenir la séquence complète de l'ADNc et la positions génomiques des composantes de l'ADNc (nommés exons par simplification).

1. Alignement des séquences de fixation sur l'ADNc

Le programme commence par aligner les séquences de fixation de la protéine sur l'ARNm sur la séquence de l'ADNc afin d'isoler les coordonnées ARN de la séquence de fixation.

```
def __alignSequences(self, parameters : list[pb.Database , dict[ tuple[str, int] :
str]]) -> dict[ tuple[str, int] : tuple[int, int]]:
        """
        Method to align the sequences on the cDNA from pyensembl
        """
        bdd, input_data = parameters # get the pyensembl database and input data
        ensembl_id = input_data.keys()
        sequences = input_data.values()
        result = {}
        for ensembl_id, sequence in zip(ensembl_id, sequences):
            try:
                transcript : pb.Transcript = bdd.transcript_by_id(ensembl_id[0])
                cDNA = transcript.sequence
            except:
                result[ensembl_id] = ("unknown", "unknown"), [("unknown", "unknown")]
            else :
                rna_start, rna_end = SequenceFinder.align(cDNA, sequence)
                if SequenceFinder.isRnaCoordNumber(rna_start) and
SequenceFinder.isRnaCoordNumber(rna_end):
                    self.genomic_coordinate_list =
self.__spliced_to_genomic(transcript, range(rna_start, rna_end+1))
                    result[ensembl_id] = ((rna_start, rna_end),
self.genomic_coordinate_list)
                else:
                    result[ensembl_id] = ("Not found", "Not found"), [("Not found",
"Not found")]
        return result
```

```python
@staticmethod
def align(cDNA : str, sequence : str) -> tuple[int, int]:
    """
    Method to align the sequence on the cDNA
    """
    matches = list(re.finditer(sequence, cDNA))

    if SequenceFinder.isNone(matches):
        return ("Not found", "Not found")
    elif SequenceFinder.isUnique(matches):
        start = matches[0].start()
    else:
        print("Multiple start found")
        for idx, match in enumerate(matches):
            print(f"{idx}: {match.start()}")
        position = int(input(f"Choose which one you want to keep: "))
        start = matches[position].start()

    end = start + len(sequence)
    return start, end
```

2.Conversion coordonnées ARN-ADN

Ensuite, le programme récupère ces coordonnées ARN et les convertit en coordonnées ADN. Pour ce faire, nous avons utilisé pyensembl pour récupérer les positions génomiques des exons ainsi que leurs longueurs. Grâce à ces deux informations nous pouvons retrouver les coordonnées ADN de la séquence de fixation.

Sur la figure 1, la position génomique des nucléotides de la séquence de fixation correspond à la position génomique de la fin de l'exon précédent plus l'écart orange. cet écart peut être calculé en soustrayant à la position ARN la somme des longueurs des exons précédents. Cette méthode nous permet de convertir les données ARN en données ADN efficacement (3000 coordonnées en moins de 2 secondes) et elle semble fiable. Nous avons comparé à la main quelques coordonnées pour vérifier si elles concordaient entre elles.

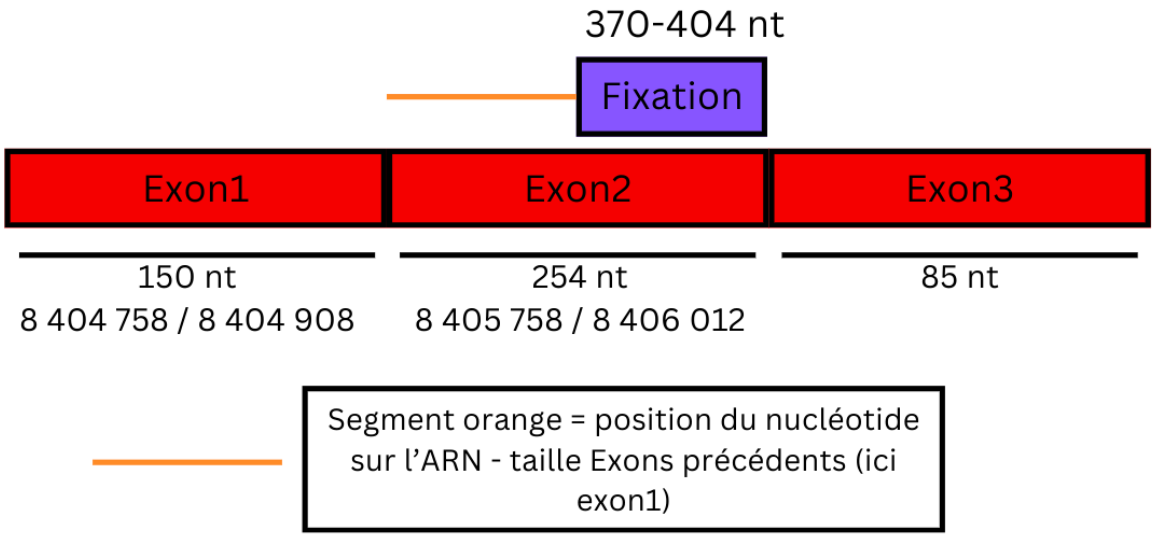

Figure 1 : Illustration de la méthode pour calculer la position sur l'ADN

```
    def __spliced_to_genomic(self, transcript : pb.Transcript, spliced_positions : list[int]):
        """
        Convert a spliced position to a genomic position.
        take the concerned transcript as argument
        and a list of spliced position
        """
        current_spliced_position = 0
        exon_number = 0
        exons = list(transcript.exons)
        genomic_positions = []

        for spliced_position in spliced_positions: # browse the list of spliced position
            if spliced_position == "Not found":
                genomic_positions.append(None)
                continue
            for i in range(exon_number, len(exons)): # browse the exon but we kept the exon
number to avoid to browse all the exons each time
                exon = exons[i]
                exon_length = exon.end - exon.start + 1
                if current_spliced_position + exon_length > spliced_position:
                    offset = spliced_position - current_spliced_position
                    genomic_positions.append(exon.start + offset)
                    break
                else:
                    current_spliced_position += exon_length
                    exon_number += 1
                if exon_number > len(exons):
                    genomic_positions.append(None)  # if the spliced position is out of range
        genomic_range = self.__determineGap(genomic_positions)
        return genomic_range
```

Code correspondant au calcul de positions génomiques

3.Calcul des étendues:

Enfin, pour terminer la conversion, le programme passe en revue toutes les positions génomiques obtenues pour les diviser selon leurs localisations sur le gène. En effet, si la séquence de fixation se trouve à cheval sur deux structures différentes (2 exons par exemple), alors il y aura une cassure entre les positions génomiques à un moment donné. Une fois ces écarts détectés, le programme les ordonne et les sauvegarde dans un nouveau fichier pour l'utilisateur.

```
  def __determineGap(self, genomic_positions : list[int]) -> list[int]:
        """
        Method to determine the gap between genomic positions thus to determine
        if the fixation sequences is between various exons or not
        """
        min_pos = 0
        self.__genomic_range_list : list[tuple[int, int]] = []
        for j in range(len(genomic_positions) - 1): # browse all the genomic
positions
            if genomic_positions[j] is None: # if we have None position we kept the
information
                self.__genomic_range_list.append((None, None))
            elif genomic_positions[j + 1] is None:
                self.__genomic_range_list.append((genomic_positions[j], None))
            else:
                gap = genomic_positions[j + 1] - genomic_positions[j]
            if gap == 1 and min_pos == 0:
                min_pos = genomic_positions[j]
            elif gap > 1 and min_pos != 0 or j == len(genomic_positions) - 2: # si on
a un gap de plus de 1 ou si on est à la fin de la liste contenant les positions
                max_pos = genomic_positions[j+1] if j == len(genomic_positions) - 2
else genomic_positions[j]
                self.__genomic_range_list.append((min_pos, max_pos))
                min_pos = 0
        return self.__genomic_range_list
```

Code pour déterminer les écarts

Interface graphique

Possibilité de choisir l'espèce et l'assemblage du génome associé
PArtie encore à définir