Universidade do Minho

Mestrado em Bioinformática

Ano Letivo 2014/2015

Docentes: Prof. Miguel Rocha

Prof. Isabel Rocha

Prof. Óscar Dias

Análise de parte do genoma da *Neisseria gonorrhoeae*

Algoritmos para Análise de Sequencias Biológicas/
Laboratórios de Bioinformática

Grupo 1:

Daniel Oliveira (PG27667)

Jorge Reis (PG26544)

Raquel Silva (PG27668)

Introdução Teórica

A *Neisseria gonorrhoeae* é uma bactéria Gram-negativa que se agrupa na forma diplococos, sendo o agente causador da gonorreia, infeção sexualmente transmitida (STI) e das mais antigas registadas no Homem [1]. De acordo com estudos recentes é a segunda maior causa de STI's de origem bacteriana, causando entre 60 a 100 milhões de novos casos por ano [1-4].

O não tratamento da infeção pode levar a infertilidade na mulher, uretrite no homem e ao aumento do risco de transmissão e aquisição do vírus da imunodeficiência humana (HIV) [1, 5].

O diagnóstico de infeção é feito, geralmente, através de uma cultura bacteriana ou ensaios de amplificação de ácidos nucleicos (NAAT). Estes testes são realizados em ambientes controlados de laboratório, com resultados geralmente disponíveis em poucos dias. Na maioria dos casos os médicos pedem então que o paciente compareça para o tratamento ser iniciado. Em clínicas de zonas urbanas a maioria dos pacientes com diagnóstico de STI's recebe tratamento atempadamente [6].

A história mostra que a *N. gonorrhoeae*, ao longo dos últimos 70 – 80 anos, foi tratada com vários tipos de fármacos diferentes que eram usados no tratamento principal na infeção. Contudo, devido ao tratamento ineficiente, a aquisição de genes de resistência e a propagação de clones de grupos isolados de linhagens resistentes, levou a problemas mais desafiantes no tratamento desta infeção e a diminuição rápida dos fármacos que se podem usar para tratar a infeção [7, 8]. Na figura 1 é possível observar alguns dos fármacos usados para tratar a infeção e a altura em que apareceram estirpes resistentes aos mesmos.

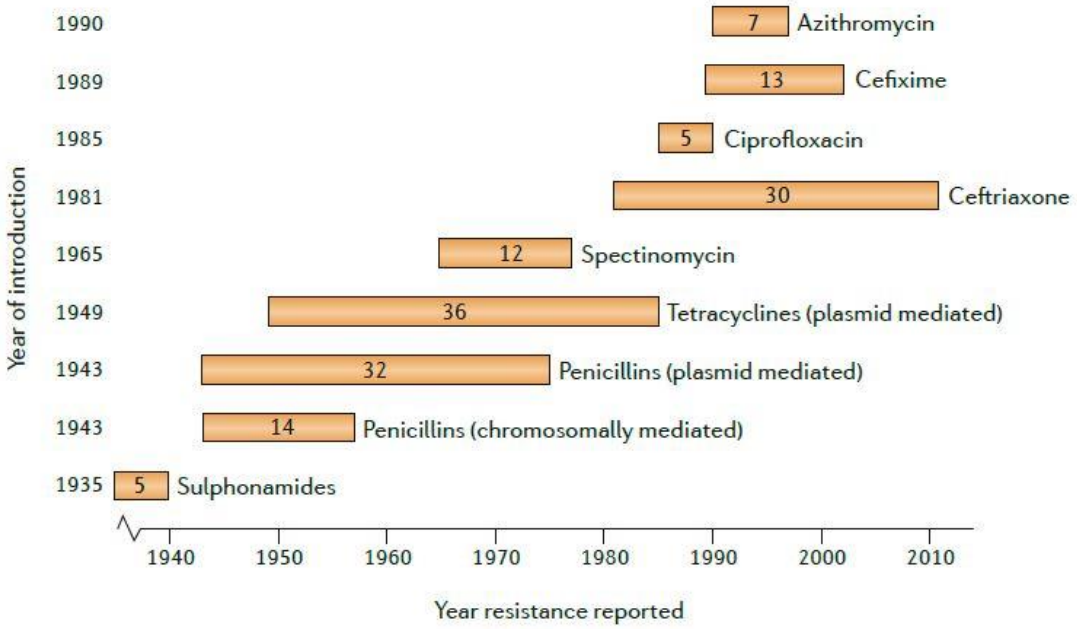


Figura 1 - Aquisição de resistências farmacológicas de *Neisseria gonorrhoeae* ao longo do tempo [9].

Existem relatos que se revelam alarmantes tendo em conta que as cefalosporinas de espectro alargado de terceira geração (ESC), ceftriaxona (injetável) e cefixima (oral) são o último tipo de antibiótico comum ainda eficiente para o tratamento da gonorreia. Contudo, a eficiência deste fármaco tem sofrido um declínio devido ao aumento de resistências a este antibiótico pela bactéria em causa [10]. Este antibiótico pertence à classe dos antibióticos β-lactâmicos, sendo que, no caso do gene *penA* que codifica a proteína PBP2, esta proteína é o alvo dos antibióticos β-lactâmicos. Existem outros genes determinantes para a *N. gonorrhoeae* adquirir resistência além do gene *penA*, como por exemplo o gene *mtrR* e *penB* [11]. Na figura 2 é apresentado um esquema ilustrativo que relaciona os antibióticos aos respetivos genes de resistência.

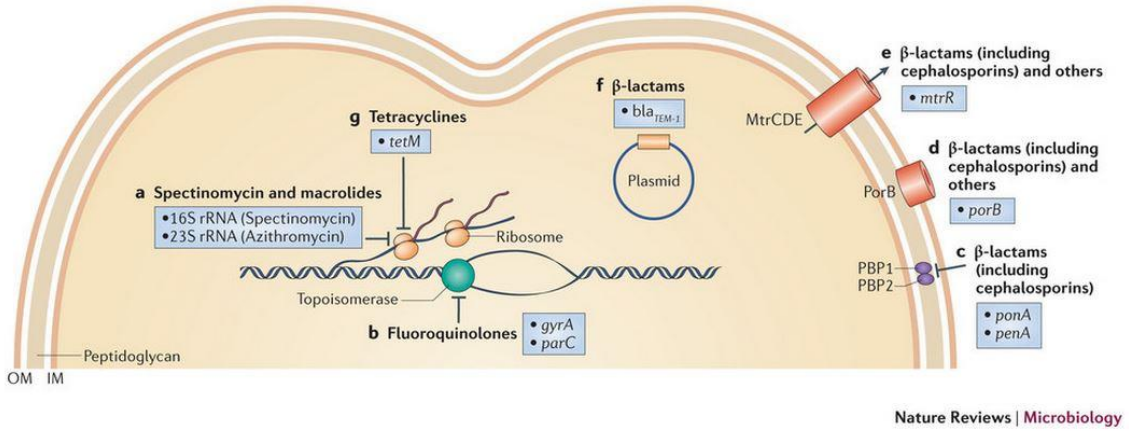


Figura 2 - Antibióticos que têm como alvo *Neisseria gonorrhoeae* e os respetivos genes de resistência [9].

Também o facto da *N. gonorrhoeae* ser uma bactéria gram-negativa indica uma maior preocupação pois, neste momento, as bactérias gram-negativas são as mais preocupantes em relação à aquisição de resistência a antibióticos. No caso da *N. gonorrhoeae*, além de algumas estirpes já serem resistentes às ESC's, também são resistentes a penicilinas, tetraciclinas, fluroquinolonas, estreptomicina, cefalosporinas de espetro estreito, combinações de sulfonamidas com trimetroprim (co-trimoxazol), macrolidos e azitromicina [8, 12].

Uma informação precisa e atualizada da resistência antimicrobiana (AMR) da *N. gonorrhoeae* é essencial para a formulação com sucesso de uma política de controlo da gonorreia. As atividades de vigilância da doença têm que ser otimizadas dependendo do país e população alvo, mas existe uma falta de dados que permita a capacidade de controlo da AMR, tanto em países desenvolvidos como em países em desenvolvimento [13].

Existem fármacos que estão a ser desenvolvidos mas, apesar de serem necessários novos fármacos, ainda nenhum está numa fase final de testes. Entre eles estão pleuromutilins, inibidores da topoisomerase bacteriana, e bombas de efluxo, inibidores de FavI e LpxC. Estes compostos podem potencialmente vir a ser explorados no futuro para melhorar o tratamento da gonorreia resistente a múltiplos fármacos (MDR) [14].

Os fatores que promovem um desenvolvimento da AMR incluem um uso indiscriminado de antibióticos e uma transmissão continuada da gonorreia. O controlo

da infeção poderá ser feito de melhor maneira por campanhas de saúde para melhorar o acesso a tratamentos de saúde, promoção de sexo seguro e regulação do uso de antibióticos, tanto no caso dos sistemas de saúde como fora deles. Este é o momento oportuno para abordar estas questões enquanto ainda existe a possibilidade de controlar a AMR [9].

Desenvolvimento

No início do trabalho estávamos a realizar um programa direccionado para um utilizador, em que este podia escolher a sua zona do genoma ou o ficheiro que pretenderia utilizar. Mais tarde, reparamos que não era necessário um programa, mas apenas funções que nos permitissem retirar a informação requerida.

Encontramos vários impasses quando pretendíamos obter a informação de forma automática, sendo que o que nos tomou mais tempo foi a obtenção dos *Uniprots Acession Numbers* (AC) e a informação associada a cada uma das proteínas, tal como a realização de *blasts* e obtenção dos *hits* Após uma procura mais aprofundada de forma a resolver o nosso primeiro impasse vimos alguns *packages*, mas estes apenas poderiam ser aplicados na versão 2 do *Python*, sendo que utilizamos *Python 3.4*. Portanto, tivemos que recorrer a outros métodos, nomeadamente aceder à informação do *site* da *Uniprot* utilizando o *proteinID* como *query* e colocando os *uniprotAC* numa lista para futura utilização, graças ao *urllib.*

Na elaboração de código para fazer *blast* tivemos problemas em arranjar um método de tornar o nosso código autónomo porque tínhamos cerca de 6528 proteínas para analisar o que era muito difícil recolher essa informação sem recorrer a um algoritmo que fizesse isso por ele. Tivemos também problemas de como organizar a nossa informação por isso recorremos a pastas com o nome das nossas proteínas para colocar toda a informação delas.

A maior parte da informação, sendo o *locus tag* do gene, o nome do gene, a sua localização e a *strand*, o *geneID* e o GI, os EC *numbers*, os *proteinID*, os produtos/nomes da proteína e a sua anotação, foi obtida através dos módulos e classes do BioPython nomeadamente, Entrez, UniprotIO e SeqIO , esta última permite-nos ver as *features* do ficheiro *genbank (gb)* . A primeira é utilizada para obter o ficheiro *gb* correspondente à *Neisseria gonorrhoeae* FA 1090, Versão: NC_002946.2 GI:59800473 na zona [0:246000] os primeiros 246000 nucleótidos correspondentes a 203 genes.

Por meio do *site* da UniProt fizemos download de um ficheiro de texto e um ficheiro *xml* em que ambos continham informação de cada gene/proteína. Como apresentavam todos os genes do organismo, tivemos que criar uma função para apenas obter os resultados dos nossos genes/proteínas correspondentes à nossa zona do genoma.

Através do ficheiro de texto obtivemos os *UniprotID*, o grau de revisão, o número de aminoácidos, a localização celular, as reacções e o *pathway*. Através do ficheiro *xml* obtivemos as identificações do KEGG *orthology*, KEGG *pathway*, BioCyc AC e GeneOntology (GO) *number*.

Com as matrizes de resultados, tivemos que recorrer às librarias *numpy* e *pandas*, de forma a criar um *data frame* com a informação e posterior criação de um ficheiro *csv* para conseguirmos fazer um ficheiro *excel* com as colunas de informação.

No caso da lista de termos do GO, também recorremos à informação apresentada no *site* da *ebi* no *QuickGO* através da *urllib*. Os domínios conservados foram obtidos através da informação de cada gene no *ncbi.*

Mesmo com estes processos automáticos tivemos que colmatar a nossa tabela *excel* manualmente recorrendo às bases de dados fornecidas no enunciado entre outras. Nomeadamente, utilizamos a base de dados da PSORT para completar a informação relativamente à localização celular, usamos o KEGG para procurar os identificadores do KEGG *pathway (ngo)* e o nome da via. Com os termos do GO obtidos, conseguimos inferir a classificação de algumas funções das proteínas, tal como a utilização dos EC *numbers* que nos permitiu conseguir os identificadores do KEGG *reactions.*

Alinhamento múltiplo e filogenia

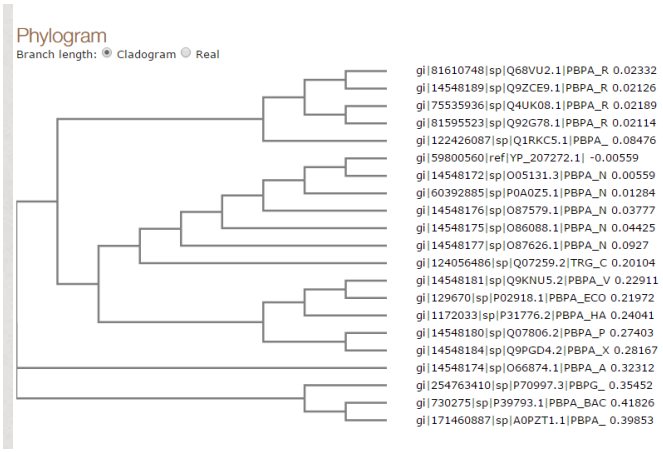


Figura 3 – Cladrograma do alinhamento múltiplo da *penicillin-binding protein*

Fizemos alinhamento múltiplo com os melhores hits da *penicillin-binding protein* do nosso organismo porque é uma proteína que apresenta resistência à penicilina, por isso usamos esta ferramenta para ver a conservação desta com outros organismos.

Utilizamos a ferramenta Clustal Omega do website http://www.ebi.ac.uk/

Através da análise do cladograma e da matriz de identidade, podemos verificar que existem 4 proteínas muito próximas à nossa proteína *Penicillin-binding protein 1A*, estas proteínas pertencem aos organismos *Neisseria meningitidis Z2491*, *Neisseria lactamica*, *Neisseria cinerea*, *Neisseria flavescens*, com percentagem de identidades de 98.57, 93.10,92.38, 83.77, respetivamente

```
     1: gi|81610748|sp|Q68VU2.1|PBPA_R  100.00   95.54   89.68   88.66   80.00   34.63   33.03   33.03   33.16   32.24   32.76   40.26   32.63   32.99   32.85   32.94   33.38   31.94
29.03   18.46   20.14
     2: gi|14548189|sp|Q9ZCE9.1|PBPA_R   95.54  100.00   90.60   89.45   81.07   35.61   33.33   33.33   33.46   32.55   33.20   39.83   32.28   33.16   33.03   32.59   33.02   31.99
28.48   19.05   19.67
     3: gi|75535936|sp|Q4UK08.1|PBPA_R   89.68   90.60  100.00   95.70   84.68   35.61   33.46   33.46   33.73   32.81   33.07   39.83   33.07   33.81   33.29   32.86   33.69   32.41
28.81   18.73   19.86
     4: gi|81595523|sp|Q92G78.1|PBPA_R   88.66   89.45   95.70  100.00   84.18   36.10   34.12   34.12   34.12   33.20   33.73   40.69   32.41   33.68   33.16   32.86   33.69   32.41
28.96   18.47   19.72
     5: gi|122426087|sp|Q1RKC5.1|PBPA_   80.00   81.07   84.68   84.18  100.00   37.32   34.33   34.46   34.60   34.20   34.60   41.13   31.88   33.16   31.68   34.29   34.08   32.69
29.22   18.70   19.70
     6: gi|59800560|ref|YP_207272.1|     34.63   35.61   35.61   36.10   37.32  100.00  100.00   98.57   93.10   92.38   83.77   53.68   39.33   40.38   37.94   40.15   34.62   35.55
33.44   27.21   24.57
     7: gi|14548172|sp|O05131.3|PBPA_N   33.03   33.33   33.46   34.12   34.33  100.00  100.00   97.37   92.23   91.73   81.68   53.68   38.25   40.00   36.84   43.47   38.99   34.58
30.64   18.57   22.39
     8: gi|60392885|sp|P0A0Z5.1|PBPA_N   33.03   33.33   33.46   34.12   34.46   98.57   97.37  100.00   92.86   91.98   81.93   53.68   38.25   39.87   37.24   43.08   39.12   34.44
30.64   18.32   21.84
     9: gi|14548176|sp|O87579.1|PBPA_N   33.16   33.46   33.73   34.12   34.60   93.10   92.23   92.86  100.00   90.73   80.68   53.25   39.03   39.61   37.50   43.47   39.65   34.72
31.24   18.19   21.98
    10: gi|14548175|sp|O86088.1|PBPA_N   32.24   32.55   32.81   33.20   34.20   92.38   91.73   91.98   90.73  100.00   82.31   52.81   38.38   39.74   37.76   42.56   39.25   34.58
30.49   18.70   22.12
    11: gi|14548177|sp|O87626.1|PBPA_N   32.76   33.20   33.07   33.73   34.60   83.77   81.68   81.93   80.68   82.31  100.00   51.95   38.82   39.92   38.29   41.57   38.50   33.38
30.45   18.19   21.70
    12: gi|124056486|sp|Q07259.2|TRG_C   40.26   39.83   39.83   40.69   41.13   53.68   53.68   53.68   53.25   52.81   51.95  100.00   42.17   45.65   44.29   41.67   42.42   40.97
38.96   38.26   31.17
    13: gi|14548181|sp|Q9KNU5.2|PBPA_V   32.63   32.28   33.07   32.41   31.88   39.33   38.25   38.25   39.03   38.38   38.82   42.17  100.00   55.12   52.07   43.61   39.35   31.58
30.61   18.88   23.08
    14: gi|129670|sp|P02918.1|PBPA_ECO   32.99   33.16   33.81   33.68   33.16   40.38   40.00   39.87   39.61   39.74   39.92   45.65   55.12  100.00   53.15   42.70   39.87   34.81
30.29   19.36   22.70
    15: gi|1172033|sp|P31776.2|PBPA_HA   32.85   33.03   33.29   33.16   31.68   37.94   36.84   37.24   37.50   37.76   38.29   44.29   52.07   53.15  100.00   40.48   39.06   33.71
28.44   18.96   22.89
    16: gi|14548180|sp|Q07806.2|PBPA_P   32.94   32.59   32.86   32.86   34.29   40.15   43.47   43.08   43.47   42.56   41.57   41.67   43.61   42.70   40.48  100.00   44.43   33.10
30.59   17.95   20.83
    17: gi|14548184|sp|Q9PGD4.2|PBPA_X   33.38   33.02   33.69   33.69   34.08   34.62   38.99   39.12   39.65   39.25   38.50   42.42   39.35   39.87   39.06   44.43  100.00   33.15
28.84   19.69   21.64
    18: gi|14548174|sp|O66874.1|PBPA_A   31.94   31.99   32.41   32.41   32.69   35.55   34.58   34.44   34.72   34.58   33.38   40.97   31.58   34.81   33.71   33.10   33.15  100.00
31.30   20.55   27.14
    19: gi|254763410|sp|P70997.3|PBPG_   29.03   28.48   28.81   28.96   29.22   33.44   30.64   30.64   31.24   30.49   30.45   38.96   30.61   30.29   28.44   30.59   28.84   31.30
100.00   19.28   22.47
    20: gi|730275|sp|P39793.1|PBPA_BAC   18.46   19.05   18.73   18.47   18.70   27.21   18.57   18.32   18.19   18.70   18.19   38.26   18.88   19.36   18.96   17.95   19.69   20.55
19.28  100.00   18.32
    21: gi|171460887|sp|A0PZT1.1|PBPA_   20.14   19.67   19.86   19.72   19.70   24.57   22.39   21.84   21.98   22.12   21.70   31.17   23.08   22.70   22.89   20.83   21.64   27.14
22.47   18.32  100.00
```

Figura 4 – Matriz identidade do alinhamento múltiplo da *penicillin-binding protein*

Implementação em Python

Relativamente aos ficheiros de código, o ficheiro *"Main_Program.py"* foi separado em vários ficheiros consoante a sua utilidade, mas só o ficheiro principal é que foi utilizado para obter as informações, ou seja, os outros ficheiros de código são para visualização das funções usadas na obtenção de informação específica, para uma melhor procura por parte de outros utilizadores que pretendam utilizar as funções. O ficheiro *"Uniprot_Parser.py"* foi obtido externamente isto é, devido a dificuldades acima descritas para obter informação do UniProt, tivemos que recorrer a funções provenientes de outros programadores na mesma situação, sendo que nenhum grupo de trabalho apresentava soluções para este problema.

As funções criadas para a obtenção das nossas informações serão explicadas brevemente a seguir.

Primeiramente, temos a função **create_file** que utiliza a função **get_genome_zone** caso não apresentemos ficheiro *gb* da nossa zona do genoma. Esta última irá buscar a informação ao *ncbi*, irá criar o ficheiro *gb* sendo movido (src (*source*)) para outra pasta (res (*resource*)) para posterior utilização, e irá ler o ficheiro (*record*).

O nosso *record* é usado para obter todas as *features* e respectivos *qualifiers* nelas contidas para cada um dos nossos genes. As *features* encontradas na nossa zona foram "gene", "CDS" e "tRNA", os *qualifiers* foram "locus_tag", "protein_id", "note", "product", "db_xref", "pseudogene", "gene", "EC_number" e "translation". Alguns genes podiam conter ou não estes *qualifiers* sendo que o "db_xref" para o "CDS" normalmente apresentava-se duas vezes, para o *GI* e para o *geneID*. Cada "CDS" apresentava a localização e a *strand*.

A função **locus_tag** irá colocar numa lista todos os locus_tag (locus) dos nossos genes, sendo utilizada para muitas outras funções de forma a obter apenas a informação dos nossos genes.

As seguintes funções irão utilizar a nossa lista locus e irão a cada "CDS" *feature* e compara o *qualifier* "locus_tag" com os nossos locus, tirando a informação apenas para este gene.

A função **protein_ID** retorna o *qualifier* "protein_id" para cada gene, caso tenha um, senão retorna "Não tem".

A função **location** retorna a localização e a *strand* de cada gene.

A função **note** retorna o *qualifier* "note" de cada gene caso este tenha alguma, senão retorna “Não tem”.

A função **product** retorna o *qualifier* "product" de cada gene caso este tenha algum, senão retorna “Não contem produtos”.

A função **gene_ID_GI** retorna os dois *qualifiers* "db_xref”, nomeadamente o GI e o *geneID*.

A função **gene_names** retorna o *qualifier* "gene” caso tenha nome do gene, senão retorna “Não tem”.

A função **EC_number** retorna o *qualifier* "EC_number” caso tenha o EC *number* do gene, senão retorna “Não contem”.

Estas funções acima irão ser utilizadas na função **info** de forma a criar uma matriz (lista de listas no caso do Python) em que cada linha contém a informação retornada para cada gene, isto é, cada linha contém informação para apenas um gene.

As seguintes funções serão utilizadas para obter a informação dos pseudogenes e dos tRNA.

A função **pseudogenes** irá buscar todos os locus dos pseudogenes e colocar numa lista para posterior utilização na obtenção das *features*.

A função **location_pseudo** vai comparar os locus dos pseudogenes para obter apenas a localização e a *strand* destes.

A função **pseudogeneID** retorna apenas os *geneID* dos pseudogenes.

As funções acima serão utilizadas na função **info_pseudogenes** para obter uma matriz com a informação para cada gene.

A função **tRNA** irá buscar todos os locus da *feature* tRNA e colocar numa lista para posterior utilização na obtenção das *features*.

A função **gene_ID_tRNA** retorna apenas os *geneID* dos tRNA.

A função **product_tRNA** retorna os produtos do tRNA.

A função **location_tRNA** vai comparar os locus dos tRNA para obter apenas a localização e a *strand* destes.

Estas funções mencionadas acima são utilizadas na função **info_tRNA** em que retorna uma matriz com a informação de cada locus de tRNA.

A função **tabela** usa a matriz retornada em **info** para criar um *array* de forma a colocar num *data frame* e criar um ficheiro *csv* com as informações obtidas na matriz.

A função **tabela_pseudogenes** usa a matriz retornada em **info_pseudogenes** para criar um *array* de forma a colocar num *data frame* e criar um ficheiro *csv* com as informações obtidas na matriz.

A função **tabela_tRNA** usa a matriz retornada em **info** para criar um *array* de forma a colocar num *data frame* e criar um ficheiro *csv* com as informações obtidas na matriz.

A função **without_note** retorna uma lista com os "protein_id" de cada gene que não contenham nota.

A função **hypoth_proteins** retorna uma lista com os "protein_id" de cada gene que tenham como produto proteínas hipotéticas.

A função **GI_number** retorna uma lista com GI *numbers* de todos os nossos genes.

A função **CDD** irá aceder ao *ncbi* com cada valor de GI, irá ler a informação obtida e às *features* de como a obter o *qualifier* "db_xref" que contém os valores de CDD, posteriormente irá coloca-los numa lista.

A função **tabela_CDD** usa a matriz retornada em **CDD** para criar um *array* de forma a colocar num *data frame* e criar um ficheiro *csv* com as informações obtidas na matriz.

A função **list_genes_names** dá-nos uma lista com todos os nomes dos genes caso haja.

A função **uniprot_ID** obtém os UniprotAC através do acesso por *url* do *site* da UniProt, utilizando o *urllib*, colocando-os numa lista.

Esta lista será usada como termo de comparação, de modo a conseguir a informação dada pelos ficheiros *xml* e *txt* de todas as proteínas do nosso organismo, nas seguintes funções, **info_uniprot** e **more_info_uniprot** em que se conseguem obter as características mencionadas no Desenvolvimento.

As funções **tabela_uniprot** e **tabela_uniprot2** usam as matrizes retornadas nas duas funções acima para criar *arrays* de forma a colocar em *data frame* e criar os ficheiros *csv* com as informações obtidas nas matrizes.

A função **tab** e **sorting** são utilizadas na **tabela_uniprot2,** a primeira é necessária pois as matrizes precisam de ter as mesmas linhas e colunas de informação adicionadas para conseguir separar por colunas de informação, a segunda função é usada para organizar por ordem de UniprotAC.

Para tirar a lista de termos do KEGG GO, foi necessário aceder a todos os GO *numbers* de cada UniprotAC dados na matriz ordenada pelo **sorting** retornada pela **tabela_uniprot2.** Após a obtenção destes números, tivemos que aceder novamente através da *urllib* de forma a retirar do *site QuickGO* os termos associados a cada GO e retornar uma lista.

Para procurar possíveis funções de proteínas que não tinham notas no NCBI e também confirmar se as funções das proteínas que tinham notas estavam corretos implementamos um código com ajuda de librarias para fazer *blast* de todas as proteínas da nossa parte do genoma da *N.Gonorrhoeae*. Para isso optamos primeiro por dividir as proteínas em dois grupos, proteínas com notas e sem notas no NCBI e criamos os respetivos scripts *py*.

Os dois scripts criados têm praticamente as mesmas funções apenas muda o tipo de proteínas que fazem o *blast* e as diretorias onde guardam e vão buscar os ficheiros.

Com estas funções quisemos no final criar um ficheiro *txt* com o GI da nossa proteína e dentro todas as funções dos *hits* do *blast*.

Para isso recorremos a algumas librarias:

- SeqIO para ler ficheiros *gb*
- shutil, para mover os ficheiros criados para outra diretoria;
- os.path, para trabalhar com diretorias;
- urllib, para ir à Uniprot recolher informação das nossas possíveis proteínas homólogas;

```
from Bio import SeqIO #reading gb file
from Bio.Blast import NCBIWWW,NCBIXML #fetching/parsing blast
import shutil#moving files
import os.path#cheking files in path
import urllib #getting info from site
from Uniprot_Parser import * #parsing uniprot text file
```

Figura 5 - Librarias importadas para fazer *blast*;

Na primeira parte do código fomos buscar os GI's das proteínas que vamos fazer *blast*.

```
#GI numbers from genes with note
def GInumbers(record,locus_tag):
    GI=[]
    for i in range(len(record.features)):
        my_cds = record.features[i]
        if my_cds.type == "CDS":
            for j in range(len(locus_tag)):
                if my_cds.qualifiers["locus_tag"][0]==str(locus_tag[j]):
                    if "note" in my_cds.qualifiers:
                        if "db_xref" in my_cds.qualifiers:
                            x=my_cds.qualifiers["db_xref"]
                            GI.append(x[0])
    return GI
```

```
#Get gi from protein without note
def giwithout_note(record):
    ID=[]
    for i in range(len(record.features)):
        my_cds = record.features[i]
        if my_cds.type == "CDS":
            if "note" not in my_cds.qualifiers:
                x=my_cds.qualifiers["db_xref"]
                gi=x[0]
                ID.append(gi[3:])
    return ID
```

```
IPython console
Console 10444/A
['59800477', '745998703', '59800483', '59800485', '59800487', '59800488',
'59800489', '59800490', '59800491', '59800492', '59800494', '59800495',
'59800497', '59800499', '59800500', '59800501', '59800502', '59800504',
'59800511', '59800513', '59800517', '59800521', '59800522', '59800523',
'59800524', '59800525', '59800528', '59800529', '59800530', '651851637',
'59800533', '59800536', '59800539', '59800540', '59800541', '59800542',
'59800545', '59800548', '59800549', '59800550', '59800551', '59800552',
'59800555', '59800556', '59800557', '59800559', '59800562', '59800563',
'59800564', '59800565', '59800570', '59800571', '59800572', '59800574',
'59800579', '59800580', '59800581', '59800582', '59800583', '59800584',
'59800585', '745998704', '59800589', '59800590', '59800591', '59800594',
'59800595', '59800596', '59800597', '59800598', '59800599', '59800600',
'59800601', '59800602', '59800603', '651851638', '59800607', '59800609',
'651851625', '59800614', '59800615', '59800616', '59800617', '59800619',
'59800620', '59800621', '59800622', '59800625', '59800626', '59800631',
'59800632', '59800633', '59800634', '59800636', '59800637', '59800641',
'59800642', '59800643', '59800647', '59800648', '59800649', '59800651',
'59800654', '59800655', '59800656', '59800657', '59800658', '59800659',
'59800661', '59800667', '59800668', '59800669', '59800673', '59800675',
'59800679', '59800680', '59800681', '59800683', '59800685', '59800686',
'59800688', '59800689', '59800690']
```

Figura 6 -Função que dá os GI numbers com notas no NCBI e o seu resultado na consola.

Com a lista criada pela função anterior efetuamos *blasts* para todas proteínas. Nesta parte recorremos primeiramente a uma procura *non redundant blast* contudo pelo fato de haver muitos *hits* que não correspondiam ao que estávamos à procura passamos a usar a *swissprot* pois tinha resultados mais credíveis.

Devido ao fato de este processo ser um pouco demorado preferimos guardar a informação de cada *blast* em ficheiros *xml* para no futuro podemos recorrer a essa informação.

Com os ficheiros anteriormente criados fizemos uma análise aos mesmos filtrando-os resultados como aceitáveis ou não. Se o *hit* tivesse um *E-value* inferior a 0.05 considerávamos esse resultado como um possível homólogo e guardávamos *hits* em um ficheiro "matches.txt" ou "nomatches.txt" consoante tivéssemos *hits* ou não. Quanto mais *e-value* for próximo de zero mais significante é o resultado, contudo temos de

olhar sempre para o tamanho da sequência porque as mais pequenas tem alta probabilidade de ocorrer numa grande.

Assim conseguimos separar as proteínas que podem ter homologas e as que não tem.

```python
#blast gi with note
def blastwithnote():
    locus=locus_tag(record)
    gi=GInumbers(record,locus)
    for i in range(len(gi)):
        gis=str(gi[i])
        GI_numb=gis[3:]
        result_handle = NCBIWWW.qblast("blastp","swissprot", GI_numb)
        save_file = open(GI_numb+'.xml', "w")
        save_file.write(result_handle.read())
        save_file.close()
        result_handle.close()
        #moving the file to another directory
        path=os.getcwd()
        src = path+"/"+GI_numb+'.xml' #source folder
        dst = "../res/blast_with_note/"#destination folder
        shutil.move(src, dst)
```

```python
#go to blast results and analise the hits, if e-value > 0.05 we don't consider this result
def blastanaliserwithnote():
    blast=[]
    for file in os.listdir("../res/blast_with_note"):
        if file.endswith(".xml"):
            blast.append(file)
    E_VALUE_THRESH = 0.05
    lista=[]
    for i in range(len(blast)):
        lista.append([])
        lista[i].append(blast[i])
        result_handle = open("../res/blast_with_note/"+blast[i])
        blast_record = NCBIXML.read(result_handle)
        for alignment in blast_record.alignments:
            for hsp in alignment.hsps:
                if hsp.expect < E_VALUE_THRESH:
                    lista[i].append(alignment.title)
                    lista[i].append(alignment.length)
                    lista[i].append(hsp.expect)
    save_file = open('nomatches.txt', "w")
    for i in range(len(lista)):
        if len(lista[i])<2:
            save_file.write(str(lista[i])+'\n')
    save_file.close()
    #        #moving the file to another directory
    path=os.getcwd()
    src = path+"/"+'nomatches.txt' #source folder
    dst = "../res/blast_with_note/nomatch/"#destination folder
    shutil.move(src, dst)

    save_file = open('matches.txt', "w")
    for i in range(len(lista)):
        if len(lista[i])>2:
            save_file.write(str(lista[i])+'\n')
    save_file.close()
    #        #moving the file to another directory
    path=os.getcwd()
    src = path+"/"+'matches.txt' #source folder
    dst = "../res/blast_with_note/match/"#destination folder
    shutil.move(src, dst)
```

Figura 7 - Função *blast* e blastanaliser

Tendo o ficheiro "matches.txt" criamos um algoritmo que para cada proteína procurava o UniprotAC dos resultados e organizava-os numa lista de listas que no final foi exportada para *txt* com o nome de "allhits.txt".

```
['Q5FAJ3.2', 'B4RNT1.1', 'A9M162.2', 'A1KS07.1', 'Q9JXT2.1', 'C1DA02.1', 'Q7P0R1.1', 'Q47IN0.2', 'A4XYW8.2', 'Q9HX33.1', 'B7V9C9.
['Q9JW31.1', 'Q9JXT8.2', 'Q7NZI3.1', 'B7H1F6.1', 'B2I2M3.1', 'B0VLU6.1', 'A3M1K0.2', 'B7I3R5.1', 'B0VE65.1', 'B3R3W5.1', 'B3QV54.
['Q9JXU9.1', 'Q9JW21.1', 'Q8XZE5.1', 'C5BH95.1', 'B7LW98.1', 'P0A6C5.1', 'P0A6C6.1', 'P0A6C7.1', 'A4XP61.1', 'P59293.1', 'Q8Z421.
['Q5FAK5.1', 'B4RNV9.1', 'A9M1F6.1', 'P65915.1', 'P65914.1', 'A1KS25.1', 'Q7NQ92.1', 'C1D6F5.1', 'A4GLL0.1', 'A6SUD4.1', 'Q3SM42.
['Q9I5Y1.1', 'O87796.1', 'Q56815.1', 'Q59101.2', 'Q59100.2', 'Q8YNK2.1', 'Q55664.3', 'Q9XDP3.1', 'P29271.1', 'P58336.1', 'P56888.
['Q5FAI2.1', 'B4RNW6.1', 'Q9JXV7.1', 'A9M1G3.1', 'Q9JW13.1', 'A1KS32.1', 'Q7NUK5.1', 'C1DAW8.1', 'B3R5H4.1', 'Q1LK34.1', 'Q3SKF1.
['Q5FAI1.1', 'B4RNW8.1', 'Q9JXV8.1', 'A1KS33.1', 'A1IQ34.1', 'A9M1G4.1', 'Q7NQI8.1', 'B1YWB9.1', 'Q0BC21.1', 'Q1BTS3.1', 'B1JY57.
['Q5FAH9.1', 'B4RNX2.1', 'A9M1G6.1', 'A1KS34.1', 'Q9JW10.1', 'C1D9B6.1', 'Q7P125.1', 'Q5P6J6.1', 'A9I1R3.1', 'Q46XZ7.
['Q5FAH8.1', 'Q9JW09.1', 'Q9JXW1.1', 'A1KS35.1', 'Q7NVH5.1', 'Q8Y0V1.1', 'A1K5N0.1', 'Q2L2P5.1', 'Q5P0T5.1', 'P57665.1', 'Q02F67.
['Q5FAH7.1', 'Q9JXW2.1', 'A1KS36.1', 'Q9JW08.1', 'P60091.1', 'C1DCV9.1', 'Q3SMB4.1', 'Q39JS9.1', 'Q13U36.1', 'B1JVC0.1', 'B2SYT3.
['P0ABE0.1', 'P0ABD9.1', 'P0ABD8.1', 'P0ABE1.1', 'P37799.1', 'P43874.1', 'Q06881.2', 'Q9LLC1.1', 'Q42783.1', 'Q42533.2', 'Q1XDK5.
['Q5FAH3.1', 'A9M1H2.1', 'Q9JXW4.1', 'Q9JW05.1', 'Q7NZD4.1', 'Q2Y6A2.1', 'A4XY53.1', 'Q0VSA0.1', 'Q0A9C4.1', 'B8GTQ5.1', 'A1S7P3.
['Q9JXW8.1', 'Q9JW02.1', 'Q59599.2', 'Q9CKV0.1', 'Q7VP67.1', 'Q8XZ83.1', 'Q88DU6.2', 'P38100.3', 'Q87WP4.1', 'P58943.1', 'P58942.
['Q50983.1', 'Q9JVZ6.1', 'Q9JXX4.1', 'Q9CKV3.1', 'Q7VP66.1', 'Q8RSS4.2', 'Q6F8M7.1', 'P38098.1', 'P14845.2', 'Q8XE85.1', 'Q8Z9L8.
['P45994.1', 'P45995.1', '59800520']
['Q5FAG1.1', 'B4RP08.1', 'A9M1J1.1', 'Q9JVY8.1', 'A1KS57.1', 'Q9JXY2.1', 'A3MZI4.1', 'B3GX43.1', 'Q7VLW6.1', 'A5WI04.1', 'Q189R2.
['P44681.2', 'Q7VMI2.3', 'Q9CP90.3', 'P0ABU3.2', 'P0ABU2.2', 'P0ABU4.2', 'P37518.1', 'Q8K9V2.3', 'P57288.3', 'P47270.1', 'Q89AR6.
['P44957.1', 'P0AG42.1', 'P0AG41.1', 'P0AG40.1', 'P0AG43.1', 'Q8K9Z1.1', 'P57250.1', 'P22990.1', 'P54575.2', 'P73651.1', 'Q59263.
['Q5FAF5.1', 'B4RPY6.1', 'A1KS62.1', 'Q9JVY4.1', 'A9M1J7.1', 'Q9JXY7.1', 'Q7NS57.1', 'Q3SHS3.1', 'C1DCX2.1', 'Q47BK5.1', 'Q5P221.
['Q5FAF3.1', 'A9M1J8.1', 'P65265.1', 'P65264.1', 'A1KS63.1', 'Q7NS58.2', 'B1KIT6.1', 'A0KG40.1', 'P57959.1', 'B0TJB0.1', 'B8C3C0.
['Q5FAF2.1', 'B4RPZ2.1', 'P65192.1', 'P65191.1', 'A1KS64.1', 'A9M1J9.1', 'Q7NS59.1', 'B8GQR4.1', 'Q12EY5.1', 'B7GVR1.1', 'B7IAU0.
['P09053.3', 'H3ZPL1.1', 'P14909.2', 'Q7ZLL6.1', 'Q4J8X2.1', 'Q796Q6.1', 'Q972A2.1', 'Q93QC6.1', 'Q64602.1', 'Q01856.1', 'O52815.
['P39853.1', 'P71052.1', 'Q0P9D4.1', 'Q58461.1', 'Q5QKR8.1', 'Q0P8W4.1', 'A8EZA9.1', 'O25511.1', 'A8GN21.1', 'Q9ZDJ5.1', 'A8GWP0.
['P71062.1', 'Q0P9D0.1', 'Q46628.2', 'Q04664.3', 'Q9AFI0.1', 'P26406.2', 'Q57491.1', 'P10498.1', 'P71063.1', 'Q56770.1', 'Q0P9D1.
['Q8KIU8.2', 'Q58469.1', 'P27127.2', '59800547']
```

Figura 8 - Ficheiro allhits.txt para os resultados das proteínas com notas no NCBI.

Com recurso à leitura do ficheiro anterior fomos à Uniprot recolher toda a informação das possíveis proteínas homólogas. Para organizar esses ficheiros criamos pastas com o GI da nossa proteína para alojar os ficheiros de possíveis proteínas homólogas.

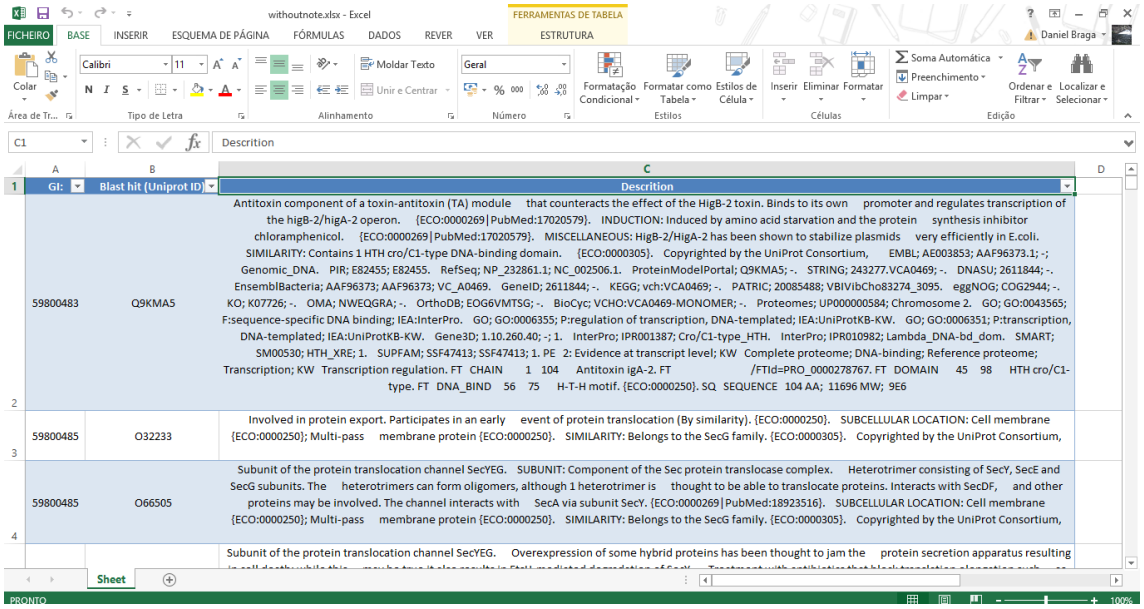
No total tivemos 6528 proteínas que podem ser homólogas das nossas.

```
#read allhits.txt, go to uniprot by hits and save txt by protein hit
def uniprotallhitswithnote():
    handle = open("../res/blast_with_note/match/allhits/allhits.txt").readlines()



    for n in range(len (handle)):
            x=handle[n].split()
            for k in range(len(x)-1,len(x)):
                e=x[k]
                limpo=e.replace("']","")
                gi=(limpo[1:])
                #to organize results we create directory with gi of our proteins and save all txt from protein in the
                if not os.path.exists("../res/blast_with_note/match/function/teste/"+gi):
                    os.makedirs("../res/blast_with_note/match/function/teste/"+gi)
            for j in range(len(x)-1):
                m=x[j]
                q=m.replace("[" ,"")
                protein=q[1:7]
                site = urllib.request.urlopen("http://www.uniprot.org/uniprot/"+protein+".txt")
                data = site.readlines()
                file = open("../res/blast_with_note/match/function/teste/"+protein+'.txt',"wb") #open file in binary
                file.writelines(data)
                file.close()
                try:
                    src = "../res/blast_with_note/match/function/teste/"+protein+'.txt' #source folder
                    dst = "../res/blast_with_note/match/function/teste/"+gi #destination folder
                    shutil.move(src, dst)
                except:
                        pass
```

Figura 9 - Programa que cria pastas com o nome do GI das nossas proteínas e guarda a informação dessas proteínas, recolhida da Uniprot , na respetiva pasta.

Para concluir fomos a cada pasta e pegamos apenas na informação que pretendíamos, a função das proteínas, posto isto criamos um ficheiro com as funções de todas as possíveis proteínas homólogas com o GI da nossa proteína. Esse ficheiro serviu para criar duas tabelas de Excel uma para as proteínas com notas e a outra para as proteínas sem notas. Para a criação das tabelas utilizamos o script “exportexcel.py”.

Estas tabelas foram utilizadas para completar a tabela principal e foram criadas para ser mais fácil trabalhar com a informação que conseguimos com os *blasts*.



	A	B	C	D
1	GI:	Blast hit (Uniprot ID)	Description	
2	59800483	Q9KMA5	Antitoxin component of a toxin-antitoxin (TA) module that counteracts the effect of the HigB-2 toxin. Binds to its own promoter and regulates transcription of the higB-2/higA-2 operon. {ECO:0000269|PubMed:17020579}. INDUCTION: Induced by amino acid starvation and the protein synthesis inhibitor chloramphenicol. {ECO:0000269|PubMed:17020579}. MISCELLANEOUS: HigB-2/HigA-2 has been shown to stabilize plasmids very efficiently in E.coli. SIMILARITY: Contains 1 HTH cro/C1-type DNA-binding domain. {ECO:0000305}. Copyrighted by the UniProt Consortium, EMBL; AE003853; AAF96373.1; -; Genomic_DNA. PIR; E82455; E82455. RefSeq; NP_232861.1; NC_002506.1. ProteinModelPortal; Q9KMA5; -. STRING; 243277.VCA0469; -. DNASU; 2611844; -. EnsemblBacteria; AAF96373; AAF96373; VC_A0469. GeneID; 2611844; -. KEGG; vch:VCA0469; -. PATRIC; 20085488; VBIVibCho83274_3095. eggNOG; COG2944; -. KO; K07726; -. OMA; NWEQGRA; -. OrthoDB; EOG6VMTSG; -. BioCyc; VCHO:VCA0469-MONOMER; -. Proteomes; UP000000584; Chromosome 2. GO; GO:0043565; F:sequence-specific DNA binding; IEA:InterPro. GO; GO:0006355; P:regulation of transcription, DNA-templated; IEA:UniProtKB-KW. GO; GO:0006351; P:transcription, DNA-templated; IEA:UniProtKB-KW. Gene3D; 1.10.260.40; -; 1. InterPro; IPR001387; Cro/C1-type_HTH. InterPro; IPR010982; Lambda_DNA-bd_dom. SMART; SM00530; HTH_XRE; 1. SUPFAM; SSF47413; SSF47413; 1. PE 2: Evidence at transcript level; KW Complete proteome; DNA-binding; Reference proteome; Transcription; KW Transcription regulation. FT CHAIN 1 104 Antitoxin igA-2. FT /FTId=PRO_0000278767. FT DOMAIN 45 98 HTH cro/C1-type. FT DNA_BIND 56 75 H-T-H motif. {ECO:0000250}. SQ SEQUENCE 104 AA; 11696 MW; 9E6	
3	59800485	O32233	Involved in protein export. Participates in an early event of protein translocation (By similarity). {ECO:0000250}. SUBCELLULAR LOCATION: Cell membrane {ECO:0000250}; Multi-pass membrane protein {ECO:0000250}. SIMILARITY: Belongs to the SecG family. {ECO:0000305}. Copyrighted by the UniProt Consortium,	
4	59800485	O66505	Subunit of the protein translocation channel SecYEG. SUBUNIT: Component of the Sec protein translocase complex. Heterotrimer consisting of SecY, SecE and SecG subunits. The heterotrimers can form oligomers, although 1 heterotrimer is thought to be able to translocate proteins. Interacts with SecDF, and other proteins may be involved. The channel interacts with SecA via subunit SecY. {ECO:0000269|PubMed:18923516}. SUBCELLULAR LOCATION: Cell membrane {ECO:0000250}; Multi-pass membrane protein {ECO:0000250}. SIMILARITY: Belongs to the SecG family. {ECO:0000305}. Copyrighted by the UniProt Consortium,	
			Subunit of the protein translocation channel SecYEG. Overexpression of some hybrid proteins has been thought to jam the protein secretion apparatus resulting	

Figura 10 -Tabela criada com a informação do Blast para as proteínas sem anotações.

Para realizar uma anotação nas proteínas que não tinham informação, foram analisados os resultados do *blast*, em comparação com outras informações retiradas das restantes bases de dados pesquisadas. Regra geral os dados possibilitavam uma análise simplificada aos resultados do *blast*, sendo assim é possível retirar uma função provável para as proteínas sem notas. Contudo existiram proteínas que não tendo dados de outras bases de dados, e sendo os resultados do *blast* pouco homogéneos, a atribuição de uma função à proteína foi consideravelmente mais problemática.

Em relação as proteínas com anotações recorremos ao *blast* apenas para confirmar as funções das proteínas para verificar se as anotações estavam corretas.

Discussão de Resultados

A tabela em ficheiro *excel* denominada **"Final_Table"** contém toda a informação recolhida durante a realização deste trabalho. Esta tabela tem 26 colunas e 215 linhas, cada linha corresponde a informações de genes/proteínas, cada coluna é uma informação obtida para os genes/proteínas.

As colunas correspondem às informações do locus, gene name, location/strand, GI number/GeneID, EC number, proteinID, Uniprot_ID, Revision, Protein length, Subcellular Location, KEGG Orthlogy, KEGG Pathway, Pathway, Product (protein name), Note, Note from blast, Classification of the protein function, BioCyc AC, Catalytic Activity, KEGG Reaction, GeneOntology Terms, GO number, CDD, Property/PATRIC ID e Prosite.

Os ficheiros que deram origem à coluna "Note from blast" foram os ficheiros *excel* "withoutnote" e "withnote", que contêm a informação de todos os *hits* do *blast* para cada proteína. Após um profundo estudo dos *hits* foi escolhido o mais aproximado para as proteínas que não contêm nota. No caso de as proteínas conterem nota, foi comparada a função dada nos *hits* como a encontrada no *ncbi.* Nas não revistas foi obtido um *blast* muito inconclusivo.

Na nossa zona [0:246000] temos 203 genes, 7 pseudogenes, 4 tRNA, 43 proteínas hipotéticas, com 137 proteínas/genes sem nota.

Um gene/proteína que encontramos na literatura, e que não apareceu nos dados que retiramos automaticamente das bases de dados, para a nossa região do genoma, foi o ponA/PBP1 respectivamente. Os antibióticos β-lactamicos tem como um dos possíveis alvos a penicillin-binding protein 1 (PBP1) e PBP2, que são necessárias para a síntese do peptidoglicano. Sendo assim, uma mutação no gene ponA que codifica a PBP1 é uma das hipóteses que confere resistência aos antibióticos β-lactamicos.

Bibliografia

1. Jarvis, G.A. and T.L. Chang, *Modulation of HIV transmission by Neisseria gonorrhoeae: molecular and immunological aspects.* Curr HIV Res, 2012. **10**(3): p. 211-7.
2. Skerlev, M. and I. Culav-Koscak, *Gonorrhea: new challenges.* Clin Dermatol, 2014. **32**(2): p. 275-81.
3. Markowicz, S., et al., *Gonococcal aneurysm of the ascending aorta: case report and review of Neisseria gonorrhoeae endovascular infections.* Sex Transm Dis, 2014. **41**(2): p. 111-3.
4. Yu, R.X., et al., *Worldwide susceptibility rates of Neisseria gonorrhoeae isolates to cefixime and cefpodoxime: a systematic review and meta-analysis.* PLoS One, 2014. **9**(1): p. e87849.
5. Barbee, L.A. and J.C. Dombrowski, *Control of Neisseria gonorrhoeae in the era of evolving antimicrobial resistance.* Infect Dis Clin North Am, 2013. **27**(4): p. 723-37.
6. Watchirs Smith, L.A., et al., *Point-of-care tests for the diagnosis of Neisseria gonorrhoeae infection: a systematic review of operational and performance characteristics.* Sex Transm Infect, 2013. **89**(4): p. 320-6.
7. Rennie, R.P., *Current and future challenges in the development of antimicrobial agents.* Handb Exp Pharmacol, 2012(211): p. 45-65.
8. Unemo, M. and R.A. Nicholas, *Emergence of multidrug-resistant, extensively drug-resistant and untreatable gonorrhea.* Future Microbiol, 2012. **7**(12): p. 1401-22.
9. Goire, N., et al., *Molecular approaches to enhance surveillance of gonococcal antimicrobial resistance.* Nat Rev Microbiol, 2014. **12**(3): p. 223-9.
10. Buono, S.A., et al., *Stemming the tide of drug-resistant Neisseria gonorrhoeae: the need for an individualized approach to treatment.* J Antimicrob Chemother, 2014.
11. Blomquist, P.B., et al., *Is gonorrhea becoming untreatable?* Future Microbiol, 2014. **9**(2): p. 189-201.
12. Rossolini, G.M., et al., *Update on the antibiotic resistance crisis.* Curr Opin Pharmacol, 2014. **18C**: p. 56-60.
13. Whiley, D.M., et al., *The ticking time bomb: escalating antibiotic resistance in Neisseria gonorrhoeae is a public health disaster in waiting.* J Antimicrob Chemother, 2012. **67**(9): p. 2059-61.
14. Lewis, D.A., *Global resistance of Neisseria gonorrhoeae: when theory becomes reality.* Curr Opin Infect Dis, 2014. **27**(1): p. 62-7.

Chang, Jeff; Chapman, Brad; Friedberg, Iddo; Hamelryck, Thomas; de Hoon, Michiel ; Cock, Peter; Antao, Tiago ; Talevich, Eric ; Wilczyński, Bartek. **Biopython Tutorial and Cookbook.**