Biopython: manipulando dados biológicos usando Python e Colab

By **Diego Mariano**

4 de julho de 2021

Biopython: uma breve introdução à manipulação de dados biológicos em Python usando Colab

Diego Mariano

Revisão: Joicymara S. Xavier

BIOINFO – Revista Brasileira de Bioinformática. Edição #01. Julho, 2021.

DOI: [10.51780/978-6-599-275326-17](https://doi.org/10.51780/978-6-599-275326-17)

Neste artigo, será apresentada uma breve introdução prática a manipulação de dados biológicos usando **Biopython**. Para implementação dos exemplos, iremos utilizar a plataforma **Colab do Google Research**.

Introdução

Biopython é uma biblioteca Python para análise de dados biológicos ^[1]. Biopython possui um conjunto de classes e métodos para análises de alinhamentos de sequências, estruturas de proteínas, consulta de bancos de dados biológicos, genética de populações, filogenia, dentre muitas outras funcionalidades ^[2]. A biblioteca pode ser baixada em <https://biopython.org/>.

Para construir códigos com Biopython usaremos a plataforma gratuita do Colaboratory, também conhecido como Colab. Colab permite a edição de código Python diretamente de um navegador por meio de um ambiente interativo baseado no Jupyter notebook. Colab pode ser acessado em <https://colab.research.google.com/> e requer um conta gratuita Google para acesso.

Figura 1. Página inicial do Colab. As células ao centro são usadas para inserção do código.

Iniciando o Biopython

Para utilizar a biblioteca Biopython, deve-se inicialmente verificar se ela está instalada. Para isso, basta importar a biblioteca com o comando:

```
import Bio
```

Caso Biopython não esteja instalado, você verá uma mensagem parecida com esta:

```
-----  
ModuleNotFoundError: Traceback (most recent call last)  
<ipython-input> in <module>()  
----> 1 import Bio  
ModuleNotFoundError: No module named 'Bio'  
-----
```

Você pode instalá-lo no Colab usando o comando:

```
!pip3 install biopython
```

Para verificar se a instalação foi realizada com sucesso, tente importar o módulo novamente. Se nenhuma mensagem for exibida, isso quer dizer que o Biopython está instalado corretamente.

Manipulando sequências

Uma das funções básicas do Biopython é a análise de sequências. Para isso, é necessário importar o módulo Seq com o comando:

```
# Manipulação básica de sequências
```

```
from Bio.Seq import Seq
```

Em Biopython, sequências devem ser declaradas com o método `Seq()`:

Declarando uma sequência

```
seq1 = Seq("ACGTAGCTACGATCACAGCTA")  
print("Minha sequência é", seq1)
```

Minha sequência é ACGTAGCTACGATCACAGCTA

Após declarada, a sequência pode ser manipulada como um objeto do tipo `Seq`. Observe por exemplo, como obter o reverso complementar da sequência declarada anteriormente:

Reverso complementar

```
rc = seq1.reverse_complement()  
print("O reverso complementar é", rc)
```

O reverso complementar é TAGCTGTGATCGTAGCTACGT

Podemos ainda obter a sequência transcrita chamando o método `transcribe()`:

Transcrição

```
rna = seq1.transcribe()  
print("A sequência transcrita é", rna)
```

A sequência transcrita é ACGUAGCUACGAUCACAGCUA

Por fim, podemos obter a sequência de aminoácidos a partir da tradução da sequência:

Tradução

```
protein = seq1.translate()  
print("A sequência da proteína é", protein)
```

A sequência da proteína é T*LRSQL

Operações básicas com sequências

Podemos realizar operações básicas com objetos do tipo `Seq` como se fossem do tipo `string`. Observe, por exemplo, como realizar buscas de fragmentos de sequências dentro de uma sequência:

Buscar substrings

```
print("Buscando a substring TACGA na variável seq1:")
print(seq1.count('TACGA'))
```

```
Buscando a substring TACGA na variável seq1 : 1
```

Podemos ainda reverter sequências da mesma forma que revertermos *strings* usando `seq1[::-1]`.

Reverter sequência

```
print("Reverter sequência")
print("Original:", seq1)
print("Reversa:", seq1[::-1])
```

Reverter sequência

```
Original: ACGTAGCTACGATCACAGCTA
```

```
Reversa: ATCGACACTAGCATCGATGCA
```

Nesse caso, os valores inseridos em colchetes indicam:

```
[posição de início : posição final : passo]
```

Como os valores de posição de início e fim foram deixados em branco, todas as letras da string são utilizadas. O valor de passo -1 indica que a sequência deve receber o valor reverso.

Podemos ainda obter o índice de cada base usando a função nativa do Python `enumerate()`:

Obtendo índices dos nucleotídeos da sequência

```
print("Obtendo índices dos nucleotídeos da sequência")
for i, n in enumerate(seq1):
    print(i, n)
```

Obtendo índices dos nucleotídeos da sequência

0 A	9 C
1 C	10 G
2 G	11 A
3 T	12 T
4 A	13 C
5 G	14 A
6 C	15 C
7 T	16 A
8 A	17 G
	18 C

19 T

20 A

Caso seja necessário, podemos converter objetos de sequência em strings usando o método `str()`. Para testar isso, vamos criar uma nova variável chamada `seq2`:

Converter objetos de sequência em strings

```
print("Convertendo objetos Seq em strings")
seq2 = str(seq1)
print(type(seq2))
```

Convertendo objetos Seq em strings

<class 'str'>

Podemos converter a string em um objeto de sequência do Biopython usando o método `Seq()`:

```
seq2 = Seq(seq2)
print(type(seq2))
```

<class 'Bio.Seq.Seq'>

Concatenando sequências

Podemos unir sequência usando o operador de concatenação (+):

Concatenar

```
print("Unindo strings")
seqs = seq1 + seq2
print(seqs)
print(type(seqs))
```

Unindo strings

ACGTAGCTACGATCACAGCTAACGTAGCTACGATCACAGCTA

<class 'Bio.Seq.Seq'>

Comparando objetos de sequência

Podemos ainda comparar objetos de sequência usando o operador de igualdade (==):

verificando se strings são iguais

```
print("Comparando sequências")
seq1 == seq2
```

Comparando sequências

True

Conteúdo GC

O conteúdo GC indica o percentual de bases nitrogenadas que são guanina ou citosina. Em uma molécula de DNA, o par guanina-citosina realiza três ligações de hidrogênio, em contrapartida ao par adenina-timina, que realiza apenas duas. Logo, moléculas com maior conteúdo GC tendem a ter uma maior estabilidade. Observe como obter o conteúdo GC usando Biopython:

```
# importando método que calcula conteúdo GC
```

```
from Bio.SeqUtils import GC

print("Conteúdo GC")
print(GC(seq1))
```

Conteúdo GC

47.61904761904762

Manipulando arquivos

A manipulação de arquivos é uma tarefa fundamental em bioinformática. Uma das formas mais comuns para armazenamento de sequências é o denominado formato FASTA. Nesse formato, uma sequência única ou várias sequências (formato Multi-FASTA) são gravadas em um arquivo de texto. Cada sequência é indicada por um cabeçalho iniciado pelo símbolo de maior (>). Veja um exemplo:

```
> Cabeçalho da sequência 1
ATCGATCGATCGATCGATCGATCGATCGATCGATCGATCGATCGATCGATCGATCG
ATCGATCGATCGATCGATCGATCGATCGATCGATCGATCGATCGATCGATCGATCG
ATCGATCGATCGATCGATCGATCGATCGATCGATCGATCGATCGATCGATCGATCG

> Cabeçalho da sequência 2
GATCGATCGATCGATCGATCGATCGATCGATCGATCGATCGATCGATCGATCGATCG
ATCGATCGATCGATCGATCGATCGATCGATCGATCGATCGATCGATCGATCGATCGA
```

Podemos ler e gravar em arquivos usando o módulo SeqIO, mas primeiro, é necessário enviar os arquivos para o Colab.

No Colab, arquivos podem ser enviados por meio da aba “arquivos” no menu lateral esquerdo:

Figura 2. Enviando arquivos para o Colab. Clique na aba "arquivos" e depois em "fazer upload para armazenamento da sessão".

Como exemplo, vamos analisar a sequência de uma lisozima (disponível em <https://www.uniprot.org/uniprot/P00720.fasta>).

```
>sp|P00720|ENLYS_BPT4 Endolysin OS=Enterobacteria phage T4 OX=10665
MNIFEMLRIDERLRLLKIYKDTEGYTIGIGHLLTKSPSLNAAKSELDKAIGRNCNGVITK
DEAEKLFNQDVDAAVRGILRNAKLKPVYDSLDAVRRCALINMVFQMGFTGVAGFTNSLRM
LQQRWDEAAVNLAKEIWNQTPNRAKRVITTFRTGTWDAYKNL
```

Vamos salvar esse arquivo com o nome **seq.fasta** e, a seguir, analisar esse arquivo com o Biopython. Como exemplo, vamos abrir esse arquivo FASTA, copiar a sequência e salvar em um outro arquivo denominado **seq2.fasta**. Entretanto, vamos alterar o cabeçalho, usando expressões regulares, para que apenas o código UNIPROT da sequência seja salvo. Para isso, precisaremos do módulo **re**:

```
# GRAVANDO ARQUIVOS

1  # Importando o método SeqIO
2  -----
3  from Bio import SeqIO
4  from Bio.SeqRecord import SeqRecord
5  import re
6
7  gravar = []
8
9  for i in SeqIO.parse("seq.fasta", "fasta"):
10     nome = i.id
11     descricao = i.description
12     seq = i.seq
13
14     # coletando o id do uniprot
15     -----
16     id_uniprot = re.findall('\|.*\|', descricao)
17     id_uniprot = id_uniprot[0].replace("|", "")
18
19     # seqrecord
20     -----
```

```

20
21     aux = SeqRecord(
22         seq,
23         id = id_uniprot,
24         description=""
25     )
26     gravar.append(aux)
27
28 # criando um novo arquivo fasta
29 -----
30 SeqIO.write(
31     gravar,
32     "seq2.fasta",
33     "fasta"
34 )

```

Ao executar esse código, o arquivo seq2.fasta será salvo no mesmo diretório. O Colab permite que você abra arquivos de texto diretamente na interface. Note que a única diferença entre os arquivos será o cabeçalho.

Figura 3. Ao executar o código, um novo arquivo será exibido no mesmo diretório.

Bioinformática estrutural de proteínas

Biopython fornece o módulo Bio.PDB que permite a leitura e manipulação de estruturas de proteínas diretamente do *Protein Data Bank* (PDB). Para isso, deve-se inicialmente importar o módulo Bio.PDB com o código:

```
from Bio.PDB import *
```

Fazendo o download de um arquivo PDB

Podemos fazer o download de arquivos diretamente do PDB usando o método *retrieve_pdb_file()* e informando o código identificador de quatro dígitos. Entretanto, deve-se inicialmente obter a lista atualizada de arquivos do PDB usando o comando *PDBList()*. Observe como realizar o download da estrutura da proteína beta-glicosidase de PDB ID 4MDP.

```
from Bio.PDB import *
pdb = PDBList()
```



```
pdb.retrieve_pdb_file('4MDP')
```

```
WARNING: The default download format has changed from PDB to  
PDBx/mmCif
```

```
Downloading PDB structure '4MDP'...
```

```
'/content/md/4mdp.cif'
```

Por padrão, o Biopython retorna um arquivo no formato **PDBx/mmCif** (extensão `.cif`).

Calculando a distância Euclidiana entre dois átomos

Agora que já realizamos o download de um arquivo de estrutura de proteína, vamos realizar algumas análises básicas nesse arquivo usando Biopython. No exemplo a seguir, vamos “navegar” pela estrutura da proteína, passando pelos modelos disponíveis, pelas cadeias, resíduos e, por fim, por cada átomo. Vamos ainda calcular a distância Euclidiana entre dois átomos. Para ilustrar vamos usar os carbonos-alfa dos resíduos 475 e 476:

Distância Euclidiana entre átomos com Biopython

```
1  from Bio.PDB import *
2
3  parser = MMCIFParser()
4  estrutura = parser.get_structure('4mdp', 'md/4mdp.cif')
5
6  print(estrutura)
7
8  # estrutura -> modelos -> cadeias -> resíduos -> átomos
9  -----
10 for modelo in estrutura:
11     print(modelo)
12     for cadeia in modelo:
13         print(cadeia)
14         for residuo in cadeia:
15             nome = residuo.get_resname()
16             if nome != 'HOH':
17                 print(nome, residuo.id[1])
18
19         # Detalhes dos átomos
20         -----
21         #for atomo in residuo:
22         -----
23         #print(atomo.id) # id
24         -----
25         #print(atomo.coord) # coordenadas x, y, z
26         -----
27
28
29 # distancia euclidiana entre lys 475 e leu 468 - ca
30 -----
31 R1 = estrutura[0]['A'][475]['CA']
32 R2 = estrutura[0]['A'][476]['CA']
33
```

```

34 distancia = R1 - R2
35
36 print(distancia, 'angstroms')

```

<Structure id=4mdp>	ASP 470
<Model id=0>	SER 471
<Chain id=A>	LEU 472
SER 0	ILE 473
MET 1	ARG 474
SER 2	LYS 475
LEU 3	GLU 476
PRO 4	BGC 501
PRO 5	PEG 502
ASP 6	PEG 503
PHE 7	PEG 504
LYS 8	P04 505
TRP 9	P04 506
GLY 10	GOL 507
[...]	3.8726804 angstroms

Observe que as linhas que exibem detalhes dos átomos foram comentadas. Isso foi feito apenas para facilitar a visualização do resultado final. Caso o símbolo de comentário (#) seja removido dessas linhas, informações dos átomos, como o código identificador do átomo e suas coordenadas, seriam exibidas abaixo de cada resíduo.

Conclusão

Biopython permite diversos tipos de análise de dados biológicos. Aqui, apresentou-se uma breve introdução com alguns exemplos práticos. Foi ainda utilizado o ambiente do Colab para edição de código por meio de um navegador. Para mais informações acesse o site oficial do Biopython:

<https://biopython.org>.

Material suplementar

Códigos-fonte disponíveis aqui:

https://colab.research.google.com/drive/1axvhu7yR4CLw-7uSQX59XdWVce_CYOs?usp=sharing

Referências

1. Cock PJA, Antao T, Chang JT, Chapman BA, Cox CJ, Dalke A, et al. **Biopython: freely available Python tools for computational molecular biology and bioinformatics**. Bioinformatics. 2009;25:1422–3. doi:10.1093/bioinformatics/btp163. ↑
2. Mariano DCB, Barroso JRP de M, Correia TS, Melo-Minardi RC de. **Introdução à Programação para Bioinformática com Biopython**. 3rd edition. CreateSpace Independent Publishing Platform; 2015. Disponível em: <https://diegomariano.com/introducao-a-programacao-para-bioinformatica-com-biopython/> ↑