Seqreppy: uma biblioteca escrita em Python para representações numéricas de sequências genômicas

Ednilson Alves Lomazi[1], Willians Luiz Bueno de Souza[1]

[1] Sistemas de Informação - Centro Universitário da Fundação Educacional de Barretos (UNIFEB) - Barretos - SP – Brasil

ednilson.lomazi@sou.unifeb.edu.br, willians.souza@unifeb.edu.br

Abstract. *The use of numerical representations of genomic sequences has been helping researchers in the structural characterization of genes since the end of the 20th century, being frequently integrated with DSP analyzes and Machine Learning models. Since this integration is not a trivial task, the present work aimed at the construction of a library that facilitates the choice and use of the numerical representations commonly used. For this, Python was used in the Seqreppy implementation, following the object-oriented paradigm and incorporating 16 numerical representation methods. Therefore, Seqreppy is expected to be another tool used in the mapping and structural characterization of genes.*

Resumo. *A utilização das representações numéricas de sequências genômicas tem auxiliado pesquisadores na caracterização estrutural de genes desde o final do século XX, sendo frequentemente integradas com análises DSP e modelos de Machine Learning. Uma vez que essa integração não é uma tarefa trivial, o presente trabalho visou a construção de uma biblioteca que facilite a escolha e a utilização das representações numéricas comumente utilizadas. Para isso, contou-se com linguagem Python na implementação da Seqreppy, seguindo o paradigma da orientação a objetos e incorporando 16 métodos de representação numérica. Espera-se, portanto, que a Seqreppy seja mais uma ferramenta usada no mapeamento e caracterização estrutural de genes.*

1. Introdução

O avanço da genética molecular ao longo dos últimos anos possibilitou um conhecimento profundo sobre a natureza do gene, o qual pode ser entendido como uma região genômica associada com a síntese de proteínas ou de RNAs não-codificantes (PESOLE, 2008).

Regiões ou sequências genômicas podem ser estudadas sob um enfoque estrutural, fazendo-se o uso das representações numéricas como forma de mapeamento

da sequência original (caracteres) em uma ou mais sequências numéricas (JEFFREY, 1990; ZHANG e ZHANG, 2003).

A maneira como as representações numéricas realizam esse mapeamento da sequência provoca uma divisão dos métodos em dois grandes grupos (KWAN e ARNIKER, 2009). De acordo com Kwan e Arniker (2009), representações cujos resultados de mapeamento correspondem a sequências de valores numéricos arbitrários, como observado nos métodos Voss (VOSS, 1992) e Complex Representation (ANASTASSIOU, 2001), estão inseridas no grupo de mapeamento fixo (fixed mapping), divergindo-se daquelas que performam um mapeamento em sequências de valores correspondentes a propriedades físico-químicas da molécula.

Hoang, Yin e Yau (2016) ainda apresentam uma outra ruptura dos métodos de representação numérica, diferenciando métodos de mapeamentos de nucleotídeo (nucleotide mapping), no qual cada nucleotídeo é associado a um valor numérico fixo, de métodos que empregam um mapeamento de sequência, onde os valores gerados carregam consigo a história obtida das partes anteriores da sequência.

Essa grande variedade de representações numéricas tem proporcionado com que características distintas de uma sequência genômica sejam reveladas, como pode ser visto na utilização do método CGR (Chaos Game Representation), pelo qual estruturas fractais podem ser evidenciadas (JEFFREY, 1990). Além disso, correlações de longo e curto alcance podem ser identificadas por meio do DNA Walk (PENG et al., 1992).

Embora o resultado de uma representação numérica tenha sua importância singular na caracterização de sequências (BARBOSA e FERNANDES, 2020), em grande parte dos casos é integrado em análises baseadas no Processamento Digital de Sinais (DSP) (RANDHAWA; HILL; KARI, 2019). Entretanto, essa integração com análises DSP não é uma tarefa trivial (MABROUK, 2017).

Como pode ser observado no trabalho de Akhtar, Epps e Ambikairajah (2007), a detecção de regiões exônicas pode ser afetada pela mudança da representação numérica utilizada. A busca por similaridades entre sequências também é influenciada pela escolha do mapeamento numérico empregado (MENDIZABAL-RUIZ et al., 2017). Em contrapartida, a precisão dos descritores genômicos de Adetiba e Olugbara (2016), utilizados na classificação taxonômica de organismos eucariotos, não se mostra dependente das representações Atomic Number, Molecular Mass e EIIP (Electron-Ion Interaction Pseudopotential).

Após serem devidamente mapeadas por uma representação numérica e, em muitos casos, tratadas por análises DSP, sequências genômicas podem ser utilizadas como dados de aprendizagem em modelos de Machine Learning (DEMELER e ZHOU, 1991). Essa utilidade de sequências genômicas pode ser observada no trabalho de Randhawa, Hill e Kari (2019), onde propõe-se um método resultante da combinação entre Machine Learning e DSP (ML-DSP) para a classificação taxonômica de genomas.

Notavelmente, estudos voltados ao Machine Learning e à computação científica têm empregado a linguagem Python em suas implementações (AGARWAL e SAXENA, 2018). Como observado em Ayer, Miguez e Toby (2014), a escolha da linguagem na computação científica é favorecida por sua sintaxe amigável e por sua associação com uma grande quantidade de bibliotecas simplificadoras de tarefas rotineiras na ciência.

De acordo com Van-Rossum e Drake Jr. (1995), Python é uma linguagem interpretada, proporcionando com que o tempo despendido no desenvolvimento de softwares seja encurtado. Essa vantagem no tempo de construção pode ser associada com a desnecessidade do programador em realizar operações de compilação e linkagem (VAN-ROSSUM e DRAKE JR., 1995), o que, consequentemente, promove uma desvantagem no tempo de execução, discutida por Varsha et al. (2019), onde levanta os esforços feitos pela comunidade Python para remediar a situação.

Considerando a versatilidade e o enviesamento científico da linguagem Python, bem como a problemática levantada concernente a utilização das representações numéricas, o presente trabalho tem por objetivo geral a implementação em Python de uma biblioteca que facilite a escolha e o uso das representações numéricas comumente utilizadas na caracterização e mapeamento de sequências genômicas.

2. Material e métodos

2.1 Estrutura e descrição

Seqreppy foi desenvolvida na linguagem Python, fazendo o uso da biblioteca NumPy (HARRIS et al., 2020), sendo orientada a objetos, com todas as classes distribuídas nos módulos data, models, view, gsp e encoder.

Dentro do módulo data estão as classes relacionadas à troca de dados com o disco, de forma que as rotinas destinadas a busca e armazenamento de sequências estão presente na classe Data, que exige o formato fasta no arquivo das sequências recolhidas do disco.

Uma vez que as sequências estejam indexadas na memória, é possível fazer o uso dos métodos de representação numérica presentes no módulo models, cada um dos quais implementados em uma classe particular, que herda da classe Model, possuindo uma assinatura que será usada para sua identificação. A classe Model, por sua vez, fornece métodos e atributos padronizados para serem sobrescritos por suas classes derivadas.

O resultado de uma única codificação retornado pelas classes derivadas de Model (classes codificadoras) estrutura-se em uma matriz (1D ou 2D), onde cada linha abriga um eixo (dimensão) do método selecionado, ordenados em ordem crescente (0

para eixo x, 1 para eixo y, 2 para eixo z, etc.). Entretanto, nos casos onde o arquivo fasta possui mais de uma sequência (Seqreppy assumi isso por verdade), o resultado da codificação de uma representação numérica torna-se em uma tupla das matrizes descritas anteriormente, cujos índices correspondem a ordem de aparecimento das sequências no arquivo.

Nesse mesmo sentido, a ordem de aparecimento das sequências também é mantida nos arquivos de salvamento. A Seqreppy estrutura os resultados salvos no disco em arquivos cuja formatação assemelha-se ao fasta, isto é, mantendo-se as informações da sequência na primeira linha e nas demais os valores das codificações, de forma que cada linha de codificação corresponda a um eixo da representação numérica (Figura 1).

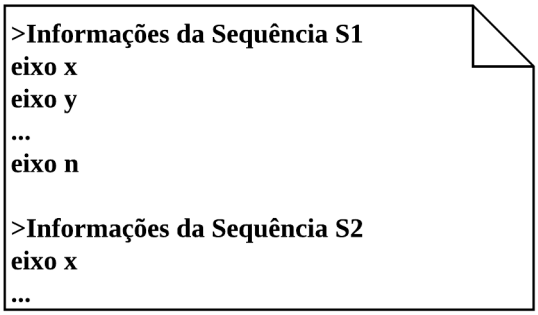

Figura 1. Padrão dos arquivos de salvamento da Seqreppy

Caso seja de interesse do usuário, a visualização em imagem do resultado de uma representação numérica pode ser obtida pela classe View, que se encontra dentro de módulo view, utilizando-se da biblioteca Matplotlib (HUNTER, 2007). Algumas customizações das imagens geradas por View podem ser alteradas no arquivo de configuração mplconfig.py, também dentro de view.

A classe View ainda faz distinção entre métodos que possuem uma representação gráfica (ou geométrica) e aqueles que não possuem, fornecendo rotinas específicas para cada um dos casos.

Nas situações onde não existe uma representação gráfica, as rotinas de View destinadas para tal cenário recebem como argumentos o resultado de uma análise DSP aplicada na sequência codificada. Nesta primeira versão, a Seqreppy fornece o Porwer Spectrum (U) como opção, o qual é calculado seguindo a Equação 1, onde F é a Transformada de Fourier Discreta (DFT) para a frequência k, e N o comprimento da sequência codificada.

$$U(k)=|F(k)|^2, \quad k = 0,1,\ldots,N-1 \qquad Eq.\ 1$$

As rotinas referentes a DFT estão presentes na classe Gsp, do módulo gsp, seguindo as especificações da Equação 2, sendo f(n) o valor da representação numérica f na posição n da sequência.

$$F(k)=\sum_{n=0}^{N-1} f(n)\mathrm{e}^{-i\frac{2\pi}{N}kn} \qquad Eq.\ 2$$

No caso da representação de Voss, o Power Spectrum (U_A, U_T, U_C, U_G) é calculado separadamente para cada um dos 4 indicadores binários, cujos resultados são somados em um único vetor (S)(ANASTASSIOU, 2000), definido na Equação 3.

$$S(k)=\sum U_x(k),\ x = A,T,C,G \qquad Eq.\ 3$$

Todas as classes mencionadas até aqui são gerenciadas pela classe SeqRep, que está dentro do módulo encoder. SeqRep é responsável por facilitar o uso das representações numéricas, fazendo o tratamento dos dados informados pelo usuário e delegando tarefas para as demais classes, além de fornecer todas as funcionalidades da Seqreppy, sendo as principais delas descritas na subseção seguinte.

2.2 Funcionamento básico

Seqreppy foi pensada em fornecer ao usuário a maior facilidade de uso possível. Devido a isso, para se obter uma ou mais representações numéricas de uma ou mais sequências genômicas seja necessário apenas que o usuário instancie a classe SeqRep, indique a fonte das sequências a serem codificadas, as representações numéricas e chame a rotina de codificação.

Entretanto, a maneira como os resultados das codificações são organizados pela SeqRep diferencia-se das tuplas de matrizes retornadas pelas classes codificadoras. Isso porque SeqRep fornece a possibilidade de se codificar em mais de uma representação a mesma sequência em um único run time. Dessa maneira, os resultados são estruturados em um dicionário, cujas chaves e valores correspondem, respectivamente, às assinaturas e às tuplas de cada representação.

Nas situações onde o método indicado não possua uma representação gráfica (ou geométrica), o parâmetro gsp da rotina de codificação pode ser configurado para verdadeiro, fazendo com que todos os resultados sejam alterados para seus respectivos valores em Power Spectrum.

Nesse momento, o usuário ainda pode salvar os resultados no disco e resgatá-los quando desejar. Nos casos onde não sejam especificados os diretórios de salvamento, diretórios padronizados da Seqreppy serão utilizados, os quais se encontram dentro do diretório results, fazendo distinção entre resultados em texto e de imagem.

Todos os módulos e diretórios expostos até aqui podem ser visualizadas na Figura 2, e o funcionamento básico descrito nesta subseção pode ser encontrado em formato de código Python no GitHub[1]

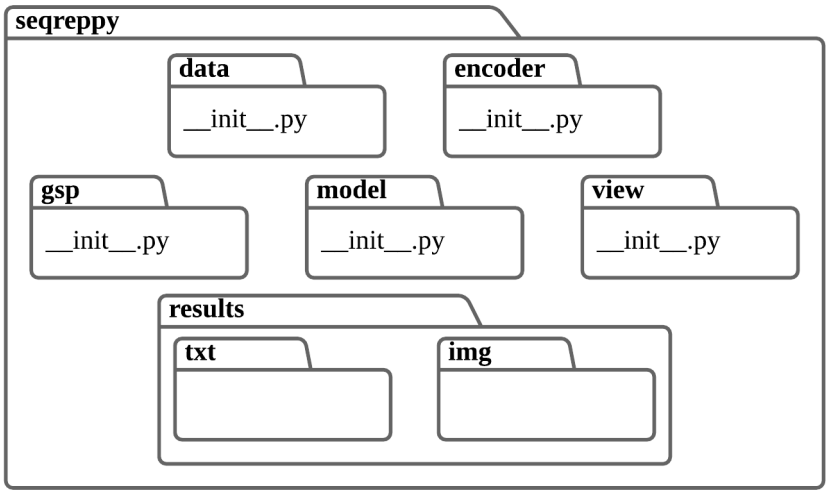

Figura 2. Estrutura e organização geral da biblioteca Seqreppy

2.3. Exceções

A biblioteca Seqreppy possui em cada módulo principal uma classe destinada ao lançamento de exceções ao usuário. Uma delas já foi mencionada anteriormente, e se refere a exigência feita pela classe Data do arquivo contendo as sequências a serem codificadas estar no formato fasta. Semelhantemente a isso, a classe Data ainda exige que os arquivos contendo os resultados salvos no disco estejam no padrão ilustrado na Figura 1.

Outra exceção diz respeito à tentativa de visualização em imagem de algum método que não possua uma representação gráfica (ou geométrica), como nos casos das representações de Voss e EIIP. Para esses casos, uma mensagem é lançada durante a exceção sugerindo a utilização de alguma análise DSP aplicada nos resultados.

Dentre as restantes, a principal exceção diz respeito a exigência dos métodos de codificação de operarem somente em sequências completas. Isto é, caso um caractere diferente daqueles representativos das bases nitrogenadas (A, T, C, G, U) for

1 https://github.com/ednilsonlomazi/seqreppy/blob/main/README.md

encontrado durante o processo de codificação, uma exceção é lançada e o processo é finalizado.

Todos os relacionamentos entre as classes mencionadas até aqui, com exceção daqueles referentes à herança das 16 classes codificadoras (motivos estéticos), podem ser visualizados na Figura 3.

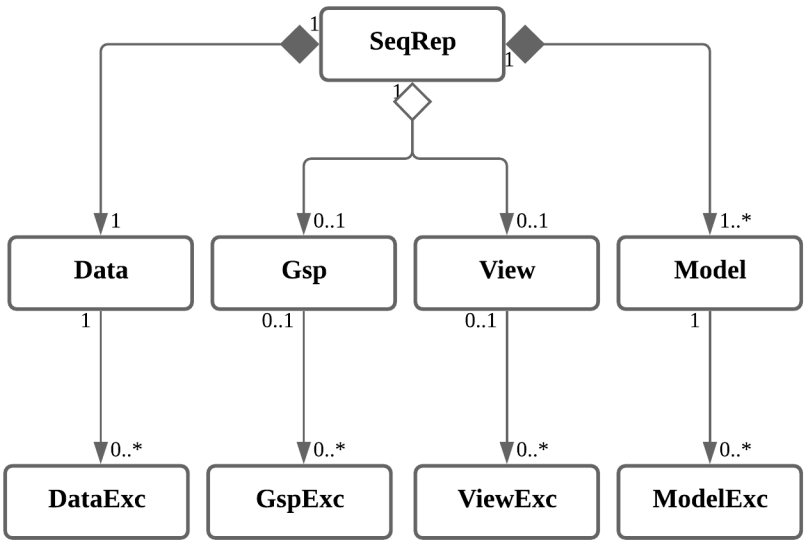

Figura 3. Diagrama de classes simplificado da Seqreppy

2.4 Teste

Para verificar o funcionamento da biblioteca, realizou-se os procedimentos referentes ao funcionamento básico da Seqreppy (subseção 2.2), utilizando o método CGR aplicado na sequência da Human Beta Globin Region do cromossomo 11 (HUMHBB), retirada do NCBI (National Center for Biotechnology Information), onde pode ser encontrada pelo identificador U01317.1.

3. Resultados e discussão

O código da Seqreppy pode ser encontrado no GitHub[2], onde também estão disponibilizadas as definições matemáticas relacionadas com as 16 representações numéricas presentes na Seqreppy[3] e listadas na Tabela 1.

---

2 https://github.com/ednilsonlomazi/seqreppy

3 https://github.com/ednilsonlomazi/seqreppy/blob/main/nr_summary.pdf

Tabela 1. Listagem das representações numéricas presentes na Seqreppy e das referências usadas na implementação.

Representações	Referências
Tetrahedron	Silverman e Linsker (1986)
CGR	Jeffrey (1990)
4-bit binary	Brunak, Engelbrecht e Knudsen (1991)
2-bit binary	Demeler e Zhou (1991)
DNA Walk	Peng et al. (1992)
Voss	Voss (1992)
Molecular Mass	Stanley et al. (1994)
Z curve	Zhang e Zhang (1994)
Complex	Anastassiou (2001)
Integer	Cristea (2002)
Real	Chakravarthy et al. (2004)
Liao	Liao, Xiang e Zhu (2006)
EIIP	Nair e Sreenadhan (2006)
Paired Numeric	Akhtar, Epps e Ambikairajah (2007)
Atomic Number	Holden et al. (2007)
Integer CGR	Yin (2019)

A visualização em CGR da HUMHBB está presente na literatura científica, e pode ser encontrada no trabalho de Jeffrey (1990). Em uma comparação superficial da visualização em CGR da HUMHBB obtida neste estudo (Figuras 4 e 5) com a gerada por Jeffrey (1990), percebe-se que ambas são praticamente idênticas.

Tanto em Jeffrey (1990) como neste trabalho (Figuras 4 e 5), pode-se observar um padrão repetitivo de regiões com baixa densidade de pontos ao longo da imagem, bem como dois segmentos de reta que se cruzam.

Embora padrões fractais em imagens CGR tenham sido encontrados para uma variedade considerável de sequências (YIN, 2019; DESCHAVANNE et al., 1999), o fenômeno nem sempre ocorre, fato evidenciado por Jeffrey (1990) em algumas sequências de bactéricas.

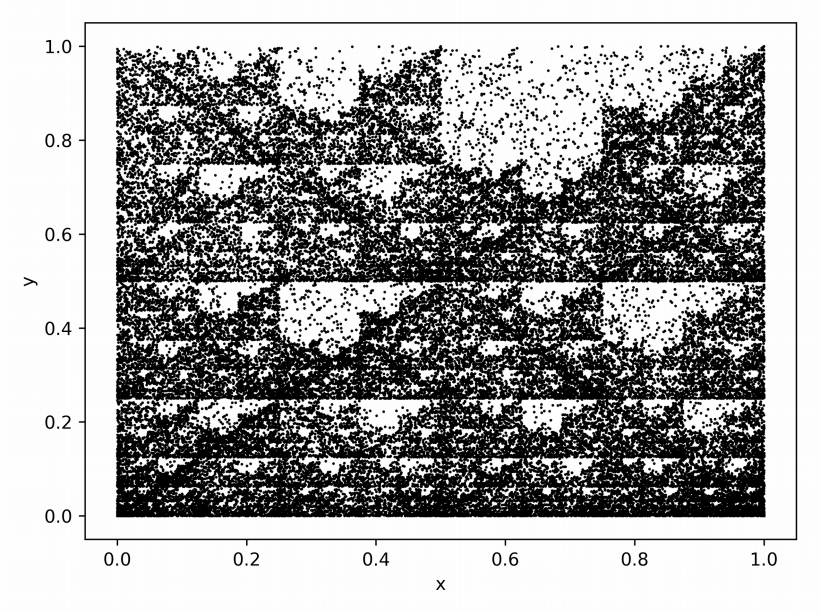

Figura 4. CGR da HUMHBB antes de ser salva no disco.

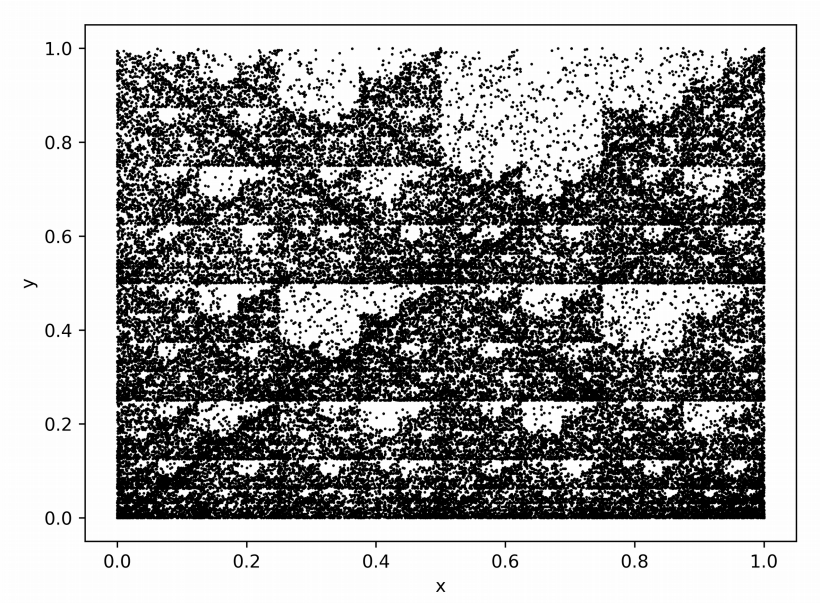

Figura 5. CGR da HUMHBB resgatada do disco.

4. Conclusão

Este trabalho apresenta à comunidade científica uma biblioteca escrita em Python que reúne funcionalidades básicas requeridas para o mapeamento de sequências genômicas em sequências numéricas.

Por isso, acredita-se que a Seqreppy será de grande utilidade para a caracterização e busca por padrões em genes, bem como no auxílio a estudos relacionados ao processamento digital de sinais aplicados na genômica.

Por fim, pretende-se continuar com a implementação das representações numéricas que ainda não estão presentes nesta biblioteca, além de incorporar extensões Python escritas na linguagem C++ na intenção de aumentar a velocidade de execução das rotinas de codificação da Seqreppy.

5. Referencias

AGARWAL, A.; SAXENA, A. Malignant tumor detection using machine learning through scikit-learn. **International Journal of Pure and Applied Mathematics**, 2018 v. 119, n. 15, p. 2863-2874.

ANASTASSIOU, D. Frequency-domain analysis of biomolecular sequences. **Bioinformatics**, 2000, v. 16, n. 12, p. 1073-1081.

ANASTASSIOU, D. Genomic signal processing. **IEEE signal processing magazine**, 2001, v. 18, n. 4, p. 8-20.

ADETIBA, E.; OLUGBARA, O. O. Classification of eukaryotic organisms through cepstral analysis of mitochondrial DNA. *In*: INTERNATIONAL CONFERENCE ON IMAGE AND SIGNAL PROCESSING, Springer, Cham, 2016, p. 243-252.

AKHTAR, M.; EPPS, J.; AMBIKAIRAJAH, E. On DNA numerical representations for period-3 based exon prediction. *In*: 2007 IEEE INTERNATIONAL WORKSHOP ON GENOMIC SIGNAL PROCESSING AND STATISTICS, IEEE, 2007, p. 1-4.

AYER, V. M.; MIGUEZ, S.; TOBY, B. H. Why scientists should learn to program in Python. **Powder Diffraction**, 2014, v. 29, n. S2, p. S48-S64.

BARBOSA, R. de M.; FERNANDES, M. A. C. Chaos game representation dataset of SARS-CoV-2 genome. **Data in Brief**, 2020, p. 105618.

BRUNAK, S.; ENGELBRECHT, J.; KNUDSEN, S. Prediction of human mRNA donor and acceptor sites from the DNA sequence. **Journal of molecular biology**, 1991, v. 220, p. 49-65.

CHAKRAVARTHY, N. *et al.* Autoregressive modeling and feature analysis of DNA sequences. **EURASIP Journal on Advances in Signal Processing**, 2004, v. 2004, n. 1, p. 952689.

CRISTEA, P. D. Genetic signal representation and analysis. *In:* PROC. OF SOCIETY OF PHOTO-OPTICAL INSTRUMENTATION ENGINEERS (SPIE) CONFERENCE, 2002, v. 4623, p. 77-84.

DEMELER, B.; ZHOU, G. W. Neural network optimization for *E.coli* promoter prediction. **Nucleic acids research**, 1991, v. 19, n.7, p. 1539-1599.

DESCHAVANNE, P. J. *et al.* Genomic signature: characterization and classification of species assessed by chaos game representation of sequences. **Molecular biology and evolution**, 1999, v. 16, n. 10, p. 1391-1399.

HARRIS, C. R. *et al*. Array programming with NumPy. **Nature**, 2020, v. 585, n. 7825, p. 357-362.

HOANG, T.; YIN, C.; YAU, S. S. -T. Numerical encoding of DNA sequences by chaos game representation with application in similarity comparison. **Genomics**, 2016, v. 108, n. 3-4, p. 134-142.

HOLDEN, T. *et al*. ATCG nucleotide fluctuation of Deinococcus radiodurans radiation genes, *In*: INSTRUMENTS, METHODS, AND MISSIONS FOR ASTROBIOLOGY X, International Society for Optics and Photonics, 2007, v. 6694, p. 669417.

HUNTER, J. D. Matplotlib: A 2D Graphics Environment. **Computing in Science & Engineering**, 2007, v. 9, n. 3, p. 90-95.

JEFFREY, H. J. Chaos game representation of gene structure. **Nucleic acids research**, 1990, v. 18, n. 8, p. 2163-2170.

KWAN, H. K.; ARNIKER, S. B. Numerical representation of DNA sequences. *In*: 2009 IEEE INTERNATIONAL CONFERENCE ON ELECTRO/INFORMATION TECHNOLOGY. IEEE, 2009, p. 307-310.

LIAO, B.; XIANG, X.; ZHU, W. Coronavirus phylogeny based on 2D graphical representation of DNA sequence. **Journal of computational chemistry**, 2006, v. 27, n. 11, p. 1196-1202.

MABROUK, M. Advanced genomic signal processing methods in DNA mapping schemes for gene prediction using digital filters. **American Journal of Signal Processing**, 2017, v. 7, n. 1, p. 12-24.

MENDIZABAL-RUIZ, G. *et al*. On DNA numerical representations for genomic similarity computation. **PloS one**, 2017, v. 12, n. 3, p. e0173288.

NAIR, A. S.; SREENADHAN, S. P. A coding measure scheme employing electron-ion interaction pseudopotential (EIIP). **Bioinformation**, 2006, v. 1, n. 6, p. 197.

PENG, C. K. *et al*. Long-range correlations in nucleotide sequences. **Nature**, 1992, v. 356, n. 6365, p. 168-170.

PESOLE, G. What is a gene? An updated operational definition. **Gene**, 2008, v. 417, n. 1-2, p. 1-4.

RANDHAWA, G. S.; HILL, K. A.; KARI, L. ML-DSP: Machine Learning with Digital Signal Processing for ultrafast, accurate, and scalable genome classification at all taxonomic levels. **BMC genomics**, 2019, v. 20, n. 1, p. 267.

SILVERMAN, B. D.; LINSKER, R. A measure of DNA periodicity. **Journal of theoretical biology**, 1986, v. 118, n. 3, p. 295.

STANLEY, H. E. *et al*. Statistical mechanics in biology: how ubiquitous are long-range correlations?. **Physica A: Statistical Mechanics and its Applications**, 1994, v. 205, n. 1-3, p. 214-253.

VAN-ROSSUM, G.; DRAKE JR, F. L. Python tutorial. Amsterdam: **Centrum voor Wiskunde en Informatica**. 1995, v. 620.

VARSHA, M. *et al*. A Review of Existing Approaches to Increase the Computational Speed of the Python. **International Journal of Research in Engineering, Science and Management**, 2019, v. 2. n. 4.

VOSS, R. F. Evolution of long-range fractal correlations and 1/f noise in DNA base sequences. **Physical review letters**, 1992, v. 68, n. 25, p. 3805-3808.

YIN, C. Encoding and Decoding DNA Sequences by Integer Chaos Game Representation. **Journal of Computational Biology**, 2019, v. 26, n. 2, p. 143-151.

ZHANG, C-T; ZHANG, R. An isochore map of the human genome based on the Z curve method. **Gene**, 2003, v. 317, p. 127-135.

ZHANG, R.; ZHANG, C-T. Z curves, an intutive tool for visualizing and analyzing the DNA sequences. **Journal of Biomolecular Structure and Dynamics**, 1994, v. 11, n. 4, p. 767-782.