Laboratório de Computação e Visualização Científica

Módulo 2

Visualização de Proteínas Desenhadas Computacionalmente

Alexandre Rodrigues, 92993
Rodrigo Kiefe, 89196
Alexandre Correia, 88770

2021-22

Departamento de Física, Universidade de Aveiro
11 de Abril de 2022

1 Introdução

Neste módulo estudamos péptidos, proteínas compostas por cerca de 100 aminoácidos. O estudo destes compostos é do interesse da humanidade, uma vez que proteínas são mutáveis e, consoante a estrutura da sua cadeia, os péptidos comportam-se de maneira diferente. Podemos construir proteínas com comportamentos específicos para desempenhar uma dada função como: coletar urânio do oceano, servir de cápsulas que guardam um medicamento, ou isótopos radioactivos, no seu interior e apenas libertam o seu conteúdo na região pretendida, gerar vacinas novas como para o caso recente de COVID-19, entre outros. Ao desenhar proteínas, criamos "máquinas"para desempenhar qualquer função em que seja necessário intervenientes de tamanho reduzido (alguns nanometros ou inferior). "Alguns exemplos de proteínas comercialmente disponíveis são homólogos sintéticos da insulina, e amilases e lipases usadas na industria alimentar."[1]

Para construir tais maquinas, as suas funções e propriedades baseiam-se na estrutura destas proteínas. Ou seja, se conseguirmos construir modelos que representem com precisão estas proteínas e as suas estruturas, torna-se possível reduzir a carga de trabalho experimental realizado em laboratório e realizam-se muitos mais testes e estruturas novas em tempo reduzido. Para além disso, "acredita-se que a selecção natural bloqueia a completa exploração de dobras e sequências, já que proteínas são nativamente escolhidas para funções dentro do contexto da célula". Assim, existe um "[...] potencial vasto por usar com funções desconhecidas e propriedades não exploradas pela natureza, mas agora acessíveis a nós [...]"[1]. "Para além da introdução de novas e/ou melhoradas funções, outro factor incitante para a implementação de ferramentas computacionais para o estudo de dobras em proteínas é a sempre-crescente distância entre as sequências conhecidas e as suas estruturas 3D correspondentes. Atender a este hiato vai desvendar funções desconhecidas de proteínas em células, bem como expandir o nosso conhecimento dos mecanismos subjacentes por detrás dos mecanismos do dobramento de proteínas."[1]

Com o intuito de realizar tal modelação computacional, utilizamos a biblioteca ProtoSyn [2] baseada na linguagem Julia [3]. Consiste num plataforma de manipulação e simulação molecular, nomeadamente para péptidos. Permite remover, adicionar ou alterar resíduos, criar péptidos a partir da sua sequência. Inclui a possibilidade de otimizar péptidos usando os algoritmos de Monte Carlo ou *Steepest Descent*. Integra *TorchANI* [4] para calcular energias e forças e permite usar bibliotecas de rotameros. Para visualização e recolha de resultados, usamos o PyMol [5].

Para simular a aplicação de rotameros e encontrar a melhor configuração, usamos o algoritmo de Monte Carlo. Trata-se de um método estatístico que faz amostragens aleatórias, criando uma configuração da péptido em cada iteração. Após avaliar a configuração usando uma função de energia, guarda-a se for o melhor resultado até ao momento. Assim, por fim, obtém-se uma configuração que minimiza o valor da energia, se o número de iterações for suficiente.

A energia do sistema é calculada em função da localização de cada átomo, as cargas envolvidas, as ligações entre átomos, os ângulos da estrutura e a força de Van der Waals, sendo a cada um destes factores atribuído um grau de importância através do parâmetro "α"(alfa). O parâmetro α define que contribuintes de energia são mais importantes tal que as interacções envolventes tenham uma hierarquia bem definida. No algoritmo usado, a medição de energia é realizada por um modelo de *Machine Learnin* (ML) treinado em *Density Function Theory* (DFT), que permite capturar as nuances referidas.

2 Simulação

Após importar a biblioteca ProtoSyn, podemos carregar e fazer a mutação para o resíduo e aminoácido escolhidos:

```
# Escolher residuo
seleResID = 48;
strResID = rid"48";
# Escolher aminoacido
seleAmino = "R";
# Carregar a proteina original
pose = ProtoSyn.Peptides.load("mol1.pdb")
# Fazer a mutacao em relacao ao especificado
ProtoSyn.Peptides.mutate!(pose, pose.graph[1, seleResID],
                          ProtoSyn.Peptides.grammar,
                          [seleAmino])
# Gravar a proteina resultante
ProtoSyn.write(pose, "mol1_mutated.pdb")
```

Incialmente escolhemos o resíduo 48 e fizemos a mutação para a Arginina ("ARG","R").

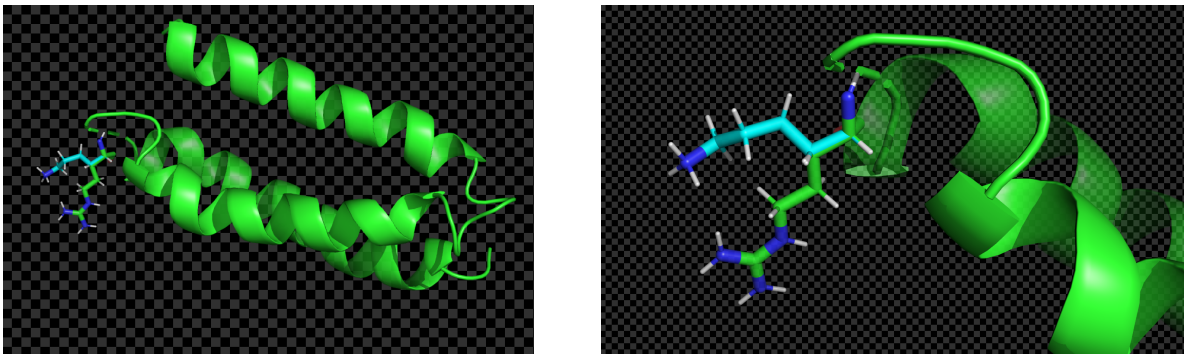

Figura 1: Mutação do resíduo 48 para a Arginina

Nas figuras 1, podemos ver a proteína original de cor verde e a alterada de cor azul. O resíduo alvo da mutação está representado no modo *sticks*. A restante proteína está sobreposta à anterior dado que não há mais alterações.

Usando o método de Monte Carlo, testou-se a aplicação de vários rotâmeros para a Arginina do resíduo 48. A simulação usou um termostato de temperatura constante, $T = 0.01$. A função de energia usada para avaliar a configuração advém da biblioteca *TorchANI* [4]. *TorchANI* é uma implementação de uma rede neuronal para obter propriedades físicas de sistemas moleculares, sendo de rápida computação mas com suficiente exatidão.

```
# Criar a funcao de energia a partir de TorchANI
energy_function = ProtoSyn.Calculators.EnergyFunction()
torchani = ProtoSyn.Calculators.TorchANI.get_default_torchani_ensemble()
push!(energy_function, torchani)
# Criar a lista de rotameros a usar
rotamer_library = ProtoSyn.Peptides.load_dunbrack()
rotamer_mutator = ProtoSyn.Peptides.Mutators.RotamerMutator(rotamer_library, 1.0,
                                                            100, strResID, false)
```

```
# Callback para gravar cada configuracao
callback = ProtoSyn.Common.default_energy_step_frame_callback(1,
                                                              "mol1_simulation.pdb")
# Termostato a temperatura constante
thermostat = ProtoSyn.Drivers.get_constant_temperature(0.01)
# Definir metodo de Monte Carlo com 100 passos e parametros definidos anteriormente
monte_carlo = ProtoSyn.Drivers.MonteCarlo(energy_function,
                                          rotamer_mutator,
                                          callback,
                                          100, #passos
                                          thermostat)
# Carregar a proteina original
pose = ProtoSyn.Peptides.load("mol1_mutated.pdb")
# Guardar a configuracao original como primeira frame do resultado
ProtoSyn.write(pose, "mol1_simulation.pdb")
# Simulacao de Monte Carlo
monte_carlo(pose)
```

Ao fim de 100 iterações, carregamos o péptido no programa de visualização *PyMOL* para criar a animação da proteína (`anim48R.gif` em anexo).

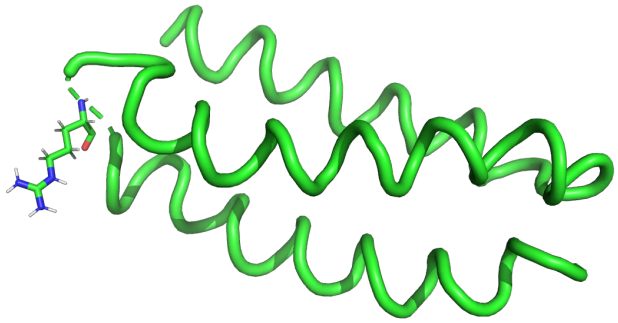

Figura 2: Configuração inicial da simulação para a Arginina

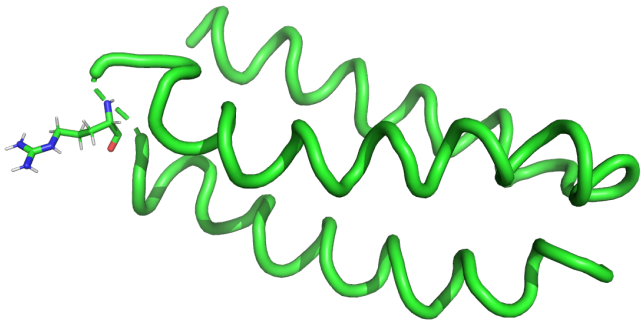

Figura 3: Configuração final da simulação para a Arginina

As figuras 2 e 3 demonstram a rotação do residuo 48 durante a simulação. A primeira configuração é mais instável, o método de Monte Carlo resulta numa configuração final mais estável.

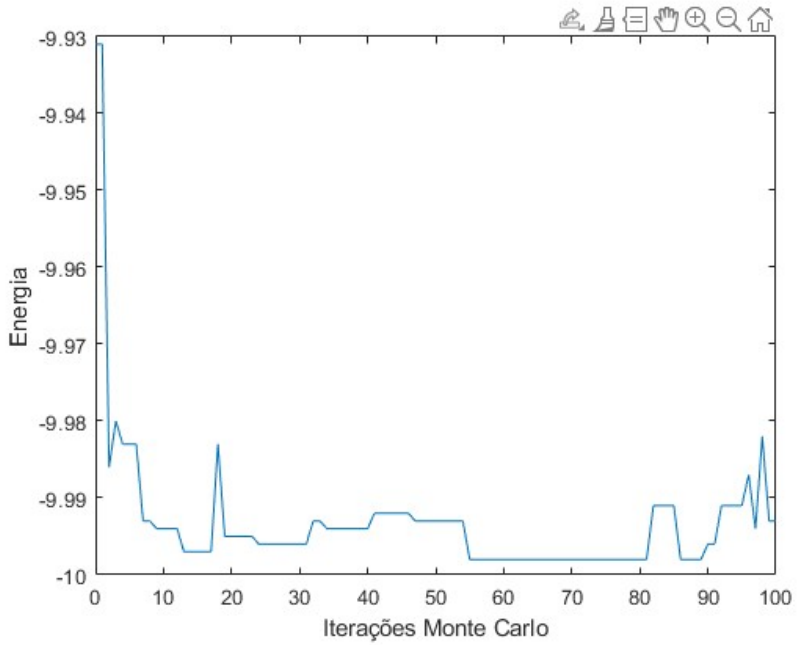

Figura 4: Variação da energia durante a simulação MC após mutação para a Arginina

Sendo a proteína mais estável quando tem um menor valor de energia, há um claro aumento de estabilidade nas primeiras iterações da simulação de Monte Carlo. Após 10 iterações, as alterações da energia são menos significativas, encontrando-se um mínimo local ao fim de 55 iterações.

Fizemos a mesma simulação para outras duas mutações iniciais, Metionina ("MET","M") e Glutamina ("GLN","Q"), de forma a comparar a energia e estrutura obtidas.

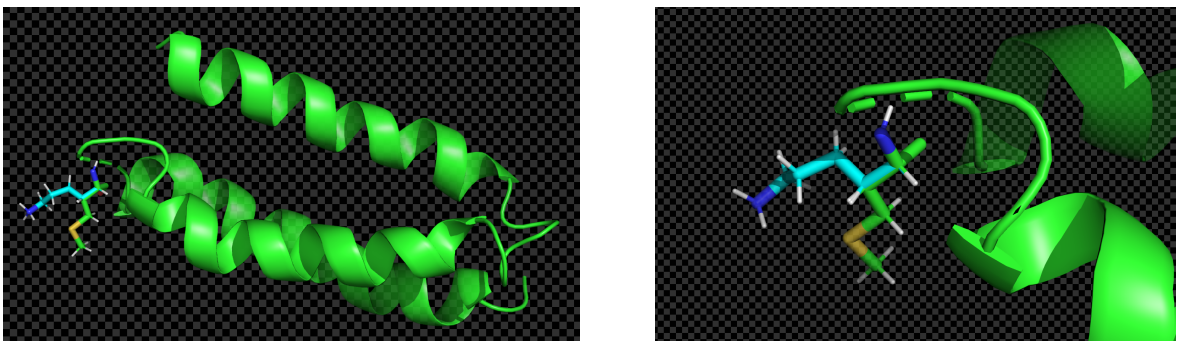

Figura 5: Mutação do resíduo 48 para a Metionina

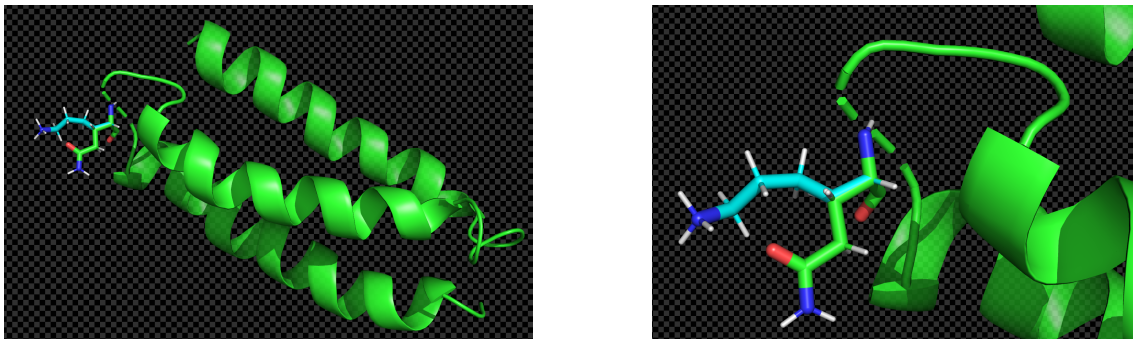

Figura 6: Mutação do resíduo 48 para a Glutamina

Nas figuras 5 e 6, pode-se verificar a diferente constituição atómica dos aminoácidos escolhidos. O péptido original está representado a azul, as mutações a verde.

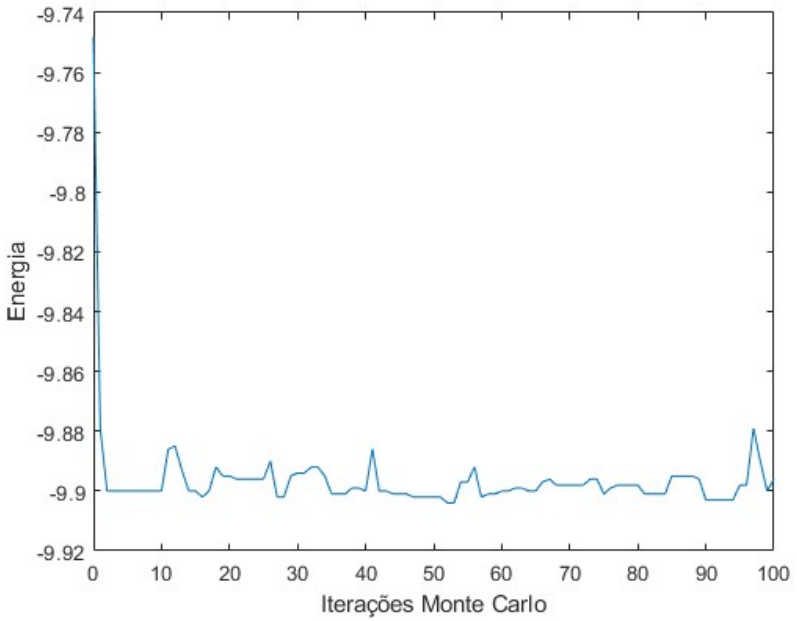

Figura 7: Variação da energia durante a simulação MC após mutação para a Metionina

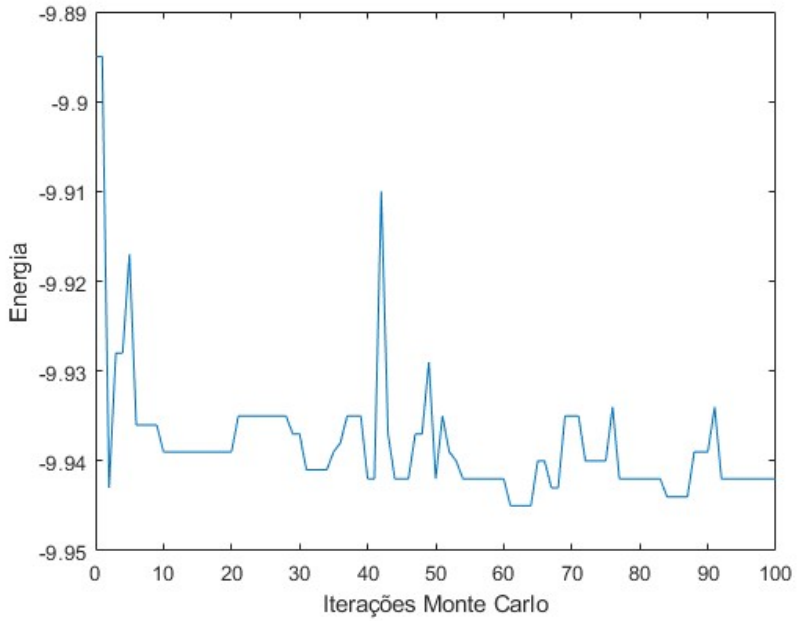

Figura 8: Variação da energia durante a simulação MC após mutação para a Glutamina

As figuras 7 e 8 são semelhantes à figura 4.

Aminoácido alvo de mutação	**Energia Inicial**	**Energia Final**
Arginina	-9.931	-9.993
Metionina	-9.748	-9.896
Glutamina	-9.845	-9.942

Tabela 1: Energias inicias e finais das simulações

Pode-se verificar que a mutação para a Arginina obtém menor energia antes e após a simulação, assim esta configuração é a mais estável das testadas.

3 Conclusão

Através da realização deste trabalho foi possível compreender e aprofundar alguns conceitos da área de desenho de proteínas. Mais especificamente, o foco principal foi sobre a aplicação de modelos de aprendizagem automática no desenho de proteínas, sendo que esta área especifica tem vindo a ganhar mais atenção por parte da comunidade cientifica nos últimos anos, devido a fatores como o aumento da capacidade computacional disponível.

Como é possível verificar nas figuras 4, 7 e 8, a aplicação do método de Monte Carlo foi realizada com sucesso, sendo que a proteína obtida no final da aplicação do método é mais estável do que a proteína inicial. Isto verifica-se devido ao facto da energia da proteína inicial ser mais elevada do que a energia da proteína final. Através destas figuras é ainda possível verificar que o método convergiu extremamente rápido nas primeira iterações, seguindo depois uma fase de relativa estabilidade, sendo o caso da Glutamina uma exceção desta estabilidade. Por fim, na tabela 1 observa-se a diferença entre os valores de energia inicial e final de cada uma das proteínas, confirmando o valor final como inferior ao valor inicial.

Referências

[1] J. M. Pereira, M. Vieira e S. M. Santos, ≪Step-by-step design of proteins for small molecule interaction: A review on recent milestones,≫ *Protein Science*, vol. 30, n.º 8, pp. 1502–1520, 2021. DOI: https://doi.org/10.1002/pro.4098. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/pro.4098.

[2] J. M. Pereira e S. M. Santos, *ProtoSyn*, https://github.com/sergio-santos-group/ProtoSyn.jl.git, 2021.

[3] *Julia*, https://julialang.org/, 2022.

[4] X. Gao, F. Ramezanghorbani, O. Isayev, J. S. Smith e A. E. Roitberg, ≪TorchANI: A Free and Open Source PyTorch-Based Deep Learning Implementation of the ANI Neural Network Potentials,≫ *Journal of Chemical Information and Modeling*, vol. 60, n.º 7, pp. 3408–3415, 2020. DOI: 10.1021/acs.jcim.0c00451. URL: https://doi.org/10.1021/acs.jcim.0c00451.

[5] Schrödinger, *PyMOL*, https://pymol.org/, 2021.