Métodos Computacionais Aplicados à Biocomplexidade

Aluno: André Gustavo Dessoy Hubner - Matrícula: 00315569
IF-UFRGS

1 de setembro de 2024

1 Introdução

Este documento relata os resultados obtidos da modelagem de um sistema biológico visto em aula através de equações diferenciais, assim como observações advindas do estudo de simulações deste.

2 Metodologia

2.1 Modelagem das reações químicas e funções de Hill

O sistema biológico modelado aqui, assim como a equação diferencial que o representa, estão representados nas duas primeiras figuras. Alguns pontos importantes podem ser levantados:

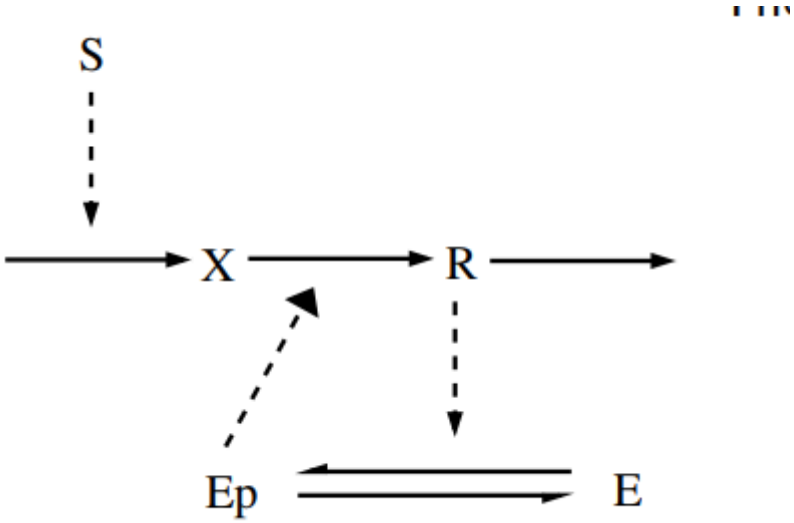


Figura 1: Série de reações do sistema biológico.

- S, de concentração constante, ativa a produção de X
- X por sua vez está diretamente responsável pela formação de R em uma reação ajudada por Ep (E em sua forma fosforilada)
- R atua como quinase em E, convertendo-o em Ep

$$
\begin{aligned}
\frac{d}{dt}[X] &= K_s[S] - k_{dx}[X] - k_0[X] - K_X \frac{\frac{[X]}{k_{mx}}}{1 + \frac{[X]}{k_{mx}}}[Ep] \\
\frac{d}{dt}[R] &= -k_{dr}[R] + k_0[X] + K_X \frac{\frac{[X]}{k_{mx}}}{1 + \frac{[X]}{k_{mx}}}[Ep] \\
\frac{d}{dt}[E] &= -K_R \frac{\frac{[E]}{k_{mk}}}{1 + \frac{[E]}{k_{mk}}}[R] + K_P \frac{\frac{[Ep]}{k_{mp}}}{1 + \frac{[Ep]}{k_{mp}}}[K] \\
E_T &= Ep + E = const.
\end{aligned}
$$

Figura 2: Equações diferenciais modelando o sistema.

- Já a conversão de Ep em E é feita por K (não representado na figura), também de concentração constante

Para adaptar essa modelagem ao código, o primeiro passo envolveu a criação das funções de hill. Essas funções são essenciais para modelar as variações das concentrações dos compostos advindas de ações enzima-ligante, uma vez que a função de ativação de Hill, por exemplo, retorna justamente uma variação correspondente à ação da enzima com o composto precursor para formar o composto produto. A função de inibição de Hill também foi adaptada, mesmo que não tenha sido utilizada.

Em seguida, preparei um dicionário contendo chaves correspondentes a todos os parâmetros necessários para representar as equações em códigos e executar as simulações. Neste dicionário são definidos todos os valores das taxas de formação e degradação dos compostos, todas as constantes de Michaelis-Mentes das respectivas reações, as concentrações iniciais de todos os compostos do sistema e os tempos iniciais, finais e de variação por uma simulação. Optei por centralizar tudo em uma estrutura pois facilita a mudança rápida das características do sistema da próxima execução, além de poder explicar a função de cada variável em uma só página.

E então, finalmente modelando uma rodada de interações do sistema, construí a função "Simulation". Ela recebe apenas o dicionário do passo anterior, atribuindo o valor de cada chave dele a variáveis locais e criando listas para armazenar as variações dos compostos não constantes (X, R e E), além de uma para armazenar cada instante de tempo. A partir disso, inicia-se um laço while que só será finalizado ao tempo atingir o limite especificado, e em cada iteração será calculado o incremento assim como especificado nas equações da figura 4, sendo estas traduzidas e adaptadas ao código. O incremento total é dependente e diretamente proporcional do valor de dt definido nos parâmetros. Ao terminar o laço, a função termina retornando todas as listas preenchidas. As oscilações resultantes da aplicação desta função às configurações utilizadas serão demonstradas na seção de resultados.

```python
def HillActivation(ligand, Km, n=1):
    ligand = round(ligand, 7)
    Km = round(Km, 7)
    return (ligand / Km)**n / (1 + (ligand / Km)**n)

def HillInibition(ligand, km, n=1):
    ligand = round(ligand, 7)
    Km = round(Km, 7)
    return 1 / (1 + (ligand / km)**n)
```

Figura 3: Funções de Hill adaptadas para o código.

```python
paramsDict = {
    "S" : 1, #Concentração inicial de S
    "X" : 1, #Concentração inicial de X
    "R" : 0, #Concentração inicial de R
    "K" : 1, #Concentração inicial de K
    "EEp" : 1, #Concentração de Enzima não fosforilada
    "Et" : 1, #Concentração total de Et = E + Ep
    "dt" : 0.02, #Variação de unidade de tempo por simulação
    "t0" : 0., #Tempo inicial das simulações
    "tf" : 500, #Tempo final das simulações
    "ks" : 1, #Taxa de formação de X por S
    "kdx" : 0.0, #Taxa de degradação de X
    "kdr" : 1, #Taxa de degradação de R
    "kr" : 1, #Taxa de E -> Ep por R
    "kp" : 1, #Taxa de formação de E por Ep
    "kbasx" : 0.1, #Taxa de formação basal de R por X
    "kmx" : 60, #Constante de Michaelis-Menten para a função de EEp em X -> R
    "kx" : 60, #Taxa de formação de X -> R por EEp
    "kmk" : .1, #Constante de Michaelis-Menten para E -> EEp por R
    "kmp" : .1, #Constante de Michaelis-Menten para EEp -> E por K
}
```

Figura 4: Dicionário com todas as variáveis parametrizáveis usadas na simulação do sistema biológico. Os valores representados aqui foram os primeiros a serem utilizados.

```python
def Simulation(params: dict):
    S, X, R, K, EEp, Et, dt, t0, tf = list(params.values())[:9]  #Variáveis de concentração e de tempo

    ks, kdx, kdr, kr, kp, kbasx, kmx, kx, kmk, kmp = list(params.values())[9:] #Variáveis de taxas e constantes

    ti = t0

    times = [ti]
    XList = [X]
    RList = [R]
    EList = [Et-EEp]

    while ti < tf:
        Xincrement = ks*S - kdx*X - kbasx*X - kx*HillActivation(X, kmx)*EEp
        Rincrement = -(kdr*R) + kbasx*X + kx*HillActivation(X, kmx)*EEp
        Eincrement = kr*HillActivation(Et-EEp, kmk)*R - kp*HillActivation(EEp, kmp)*K

        X += Xincrement*dt
        R += Rincrement*dt
        EEp += Eincrement*dt

        ti += dt

        times.append(ti)
        XList.append(X)
        RList.append(R)
        EList.append(Et-EEp)

    return times, XList, RList, EList
```

Figura 5: Função que realiza uma rodada de Simulação para os parâmetros específicados.

2.2 Obtenção dos períodos das oscilações

Para obter os períodos das oscilações, criei primeiro uma função "ExtractMaximals", que recebendo todas as listas provenientes de uma execução de Simulation, retorna ao todo 6 listas, sendo para cada composto de concentração variável uma contendo todos os seus pontos de máximo e outra os tempos de cada um destes instantes. Separando os retornos dessa maneira, seria fácil plotar os pontos de máximo de cada composto.

Em seguida criei a função "ExtractPeriodsMeans", que recebendo as listas dos tempos dos pontos maximais de cada composto retorna a média dos períodos entre as oscilações de cada unidade. Essa função chama internamente outra função "CalculatePeriods" três vezes, esta que realiza a lógica de iterar por toda a lista do parâmetro armazenando em outra lista a diferença de tempo entre cada sucessivo maximal, retornando por fim a média aritmética de todos esses elementos.

2.3 Estudo da variação dos parâmetros

Já para estudar a variação dos períodos do gráfico de acordo com a mudança dos parâmetros foram necessárias várias adaptações. De forma a rodar as simulações com valores diferentes para cada parâmetro de forma rápida e linearizada, comecei a construir a função "SearchPeriods", que gera um array com vários valores entre 0.07 e 2.11 e, para cada um desses valores e para cada parâmetro cabível de modificação do dicionário original, realiza a simulação e os cálculos dos seus períodos, retornando ao final os resultados que fossem suficientemente diferentes.

Posteriormente, feitos os primeiros testes dessa função decidi usar a mesma abordagem só que com um array com valores negativos, para obter resultados dos parâmetros com seus

```python
def ExtractMaximals(times: list, xList: list, rList: list, eList: list, len: int):
    xMaximals = list()
    rMaximals = list()
    eMaximals = list()
    xMaximalsTimes = list()
    rMaximalsTimes = list()
    eMaximalsTimes = list()

    for i in range(2, len-1):
        if xList[i-1] < xList[i] and xList[i] > xList[i+1]:
            xMaximals.append(xList[i])
            xMaximalsTimes.append(times[i])

        if rList[i-1] < rList[i] and rList[i] > rList[i+1]:
            rMaximals.append(rList[i])
            rMaximalsTimes.append(times[i])

        if eList[i-1] < eList[i] and eList[i] > eList[i+1]:
            eMaximals.append(eList[i])
            eMaximalsTimes.append(times[i])


    return xMaximals, xMaximalsTimes, rMaximals, rMaximalsTimes, eMaximals, eMaximalsTimes
```

Figura 6: Função que extrai os pontos maximais assim como o instante de tempo em que eles ocorreram de cada composto.

```python
def ExtractPeriodsMeans(xMaximalsTimes: list, rMaximalsTimes: list, eMaximalsTimes: list):
    xPeriodsMean = CalculatePeriods(xMaximalsTimes)
    rPeriodsMean = CalculatePeriods(rMaximalsTimes)
    ePeriodsMean = CalculatePeriods(eMaximalsTimes)

    return xPeriodsMean, rPeriodsMean, ePeriodsMean

def CalculatePeriods(maximalsTimesList: list):
    listPeriods = list()
    for i in range(1, len(maximalsTimesList)):
        listPeriods.append(abs(maximalsTimesList[i]-maximalsTimesList[i-1]))

    return sum(listPeriods) / len(listPeriods)
```

Figura 7: Função que retorna a média dos períodos de oscilação de cada composto, chamando CalculatePeriods interanamente para calcular a média em cada composto.

valores diminuídos e portanto gerar dados mais diversos. Neste contexto primeiro adaptei SearchPeriods para conter a lógica de utilização desses valores negativos, e depois gerei uma função "SearchLowerUpperPeriods", que chamará ambas as execuções de SearchPeriods, retornando os resultados de ambas em um só array. Esta função também cuida dos parâmetros passados para essas execuções, o que pode ser observado melhor nas figuras 8, 9 e 10.

```
def SearchLowerUpperPeriods(lowerThreshold: float, upperThreshold: float, symmetricalSearch: bool = True, satisfiableNumber: int = 20):
    """
    :param float lowerThreshold: valor máximo para considerar um período menor
    :param float upperThreshold: valor mínimo para considerar um período maior
    :param bool symmetricalSearch: indica se a busca utilizando aumento dos valores dos parâmetros só poderá retornar resultados com períodos maiores e vice-versa. Padrão True.
    :param int satisfiableNumber: número de resultados exigido para cada busca por período
    :return: as listas com os períodos maiores e menores, se conseguir achar o número especificado para pelo menos uma delas. Caso contrário returna None.

    """
    if (symmetricalSearch):
        t0 = time.time()
        upper = SearchPeriods(True, lowerThreshold, upperThreshold, satisfiableNumber/2)
        print(time.time()-t0)
        lower = SearchPeriods(False, lowerThreshold, upperThreshold, satisfiableNumber/2)

        upper.extend(lower)
        return upper
    else:
        upper = SearchPeriods(True, lowerThreshold, upperThreshold, satisfiableNumber)

    return upper
```

Figura 8: Função que chama as duas execuções de SearchPeriods, uma utilizando valores crescentes e outra decrescentes para cada parâmetro válido.

```
def SearchPeriods(increase: bool, lowerThreshold: float, upperThreshold: float, satisfiableNumber = 3):

    if increase:
        valuesArray = arange(0.07, 2.11, 0.07)
        enzimeVariable = "EEp"
    else:
        valuesArray = arange(-0.07, -0.71, -0.07)
        enzimeVariable = "E"

    satisfiableData = list()
    templateDict = dict()
    newDict = dict()

    templateDict = paramsDict.copy() #Cria o dicionário novo com os mesmos valores do original
    if increase:
        population = [a for a in templateDict.keys() if a not in ("dt", "t0", "tf", "EEp", "kmx", "kx")]
    else:
        population = [a for a in templateDict.keys() if a not in ("dt", "t0", "tf", "E", "R", "kdx", "kbasx", "kmk", "kmp", "kmx", "kx")]

    for i in valuesArray: #Para cada valor a adicionar a mais
        for key in population:
            if (len(satisfiableData) >= satisfiableNumber):
                return satisfiableData
```

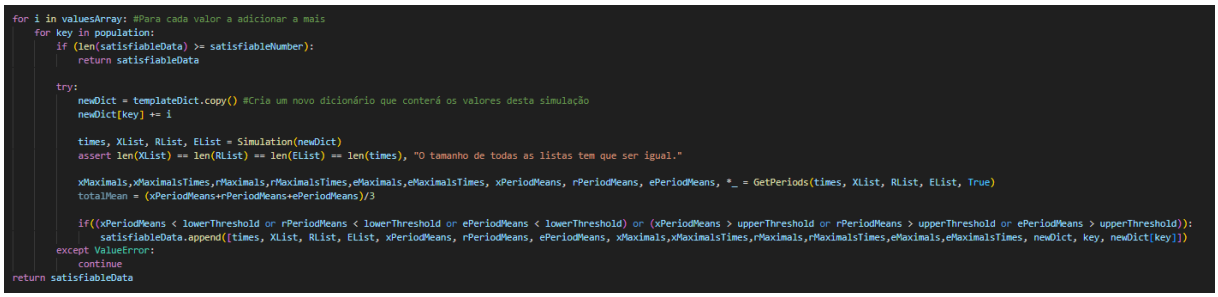
Figura 9: Primeira parte da função SearchPeriods.

Outra adaptação realizada nessa etapa foi a centralização das funções ExtractMaximals e ExtractPeriodMeans na nova função "GetPeriods", para que fosse possível chamar apenas uma função para realizar toda a lógica necessária em SearchPeriods. Além disso, esta função também serviu para encapsular a validação de resultados relevantes para a análise posterior, assim como a exclusão de casos que ocasionariam erro.

3 Resultados

3.1 Caso inicial

Logo após terminar a criação da função de simulação, executei uma rodada de simulações, chamando a função Simulation e plotando os resultados, que resultaram no gráfico da figura



```
for i in valuesArray: #Para cada valor a adicionar a mais
    for key in population:
        if (len(satisfiableData) >= satisfiableNumber):
            return satisfiableData

        try:
            newDict = templateDict.copy() #Cria um novo dicionário que conterá os valores desta simulação
            newDict[key] += i

            times, XList, RList, EList = Simulation(newDict)
            assert len(XList) == len(RList) == len(EList) == len(times), "O tamanho de todas as listas tem que ser igual."

            xMaximals,xMaximalsTimes,rMaximals,rMaximalsTimes,eMaximals,eMaximalsTimes, xPeriodMeans, rPeriodMeans, ePeriodMeans, *_ = GetPeriods(times, XList, RList, EList, True)
            totalMean = (xPeriodMeans+rPeriodMeans+ePeriodMeans)/3

            if((xPeriodMeans < lowerThreshold or rPeriodMeans < lowerThreshold or ePeriodMeans < lowerThreshold) or (xPeriodMeans > upperThreshold or rPeriodMeans > upperThreshold or ePeriodMeans > upperThreshold)):
                satisfiableData.append([times, XList, RList, EList, xPeriodMeans, rPeriodMeans, ePeriodMeans, xMaximals,xMaximalsTimes,rMaximals,rMaximalsTimes,eMaximals,eMaximalsTimes, newDict, key, newDict[key]])
        except ValueError:
            continue
return satisfiableData
```

Figura 10: Segunda parte da função SearchPeriods.

```
def GetPeriods(times, xList, rList, eList, returnNPeaks=False):
    xMaximals, xMaximalsTimes, rMaximals, rMaximalsTimes, eMaximals, eMaximalsTimes, xTotalPeaks, rTotalPeaks, eTotalPeaks = ExtractMaximals(times, xList, rList, eList)

    if (returnNPeaks):
        lenX = len(xMaximalsTimes); lenR = len(rMaximalsTimes); lenE = len(eMaximalsTimes)
        if ((lenX < 4 and lenR < 4 and lenE < 4) or (lenX < 2 or lenR < 2 or lenE < 2)):
            raise ValueError("Número de maximais encontrados insuficiente")

    xPeriodsMean, rPeriodsMean, ePeriodsMean = ExtractPeriodsMeans(xMaximalsTimes, rMaximalsTimes, eMaximalsTimes)
    if (returnNPeaks):
        if xPeriodsMean == None or rPeriodsMean == None or ePeriodsMean == None:
            raise ValueError("Uma das médias encontradas é nula")

    return xMaximals,xMaximalsTimes,rMaximals,rMaximalsTimes,eMaximals,eMaximalsTimes,xPeriodsMean,rPeriodsMean,ePeriodsMean, xTotalPeaks, rTotalPeaks, eTotalPeaks
```

Figura 11: Função que encapsula ExtractMaximals e ExtractPeriodMeans, aceitando apenas resultados válidos caso seja chamada a partir do fluxo em que eu estava trabalhando.

12. Seus resultados foram interessantes, confirmando o padrão oscilatório de sistemas biológicos envolvendo uma série de regulações entre os próprios membros. Em especial, com apenas X começando em enquanto R e E começam em 0, dá para se estimar as seguintes razões para o comportamento de cada um:

- X tem seu acúmulo muito favorecido por kdx e kbasx, que reduzem sua quantidade, estarem em valores muito baixos (ou zero no segundo), e por kx e kmx se contrabalancearem na ativação da rota X -> R por EEp, enquanto que a taxa de formação de X a partir de S tem o valor de 1 e uma quantidade constante de S para transformar em X;

- A taxa de degradação de R por sua vez tem um valor alto; ainda assim, R consegue crescer pois este é o único ponto que reduz a sua quantidade, enquanto que há taxas basais e de ativação de X -> R formando este composto

- E está em equilíbrio com o seu estado fosforilado, e uma vez que começa com 0 em estado desfosforilado, há muito mais composto para formá-lo do que o estado fosforilado, que começa saturado; o efeito vertiginoso do início é amplificado pela presença constante de K, e o sistema acaba equilibrando pela maior presença de R para fosforilá-lo em EEp

Após maiores refinamentos, incluindo a construção das funções para obter os períodos, atualizei também a plotagem destacando seus pontos de máximo, assim como o período médio entre as oscilações de cada composto (figura 13).

3.2 Estudo do período das oscilações

Já para estudar a variação dos períodos do gráfico de acordo com a mudança dos parâmetros, foram necessárias várias adaptações, como é possível ver na subseção "Estudo da variação

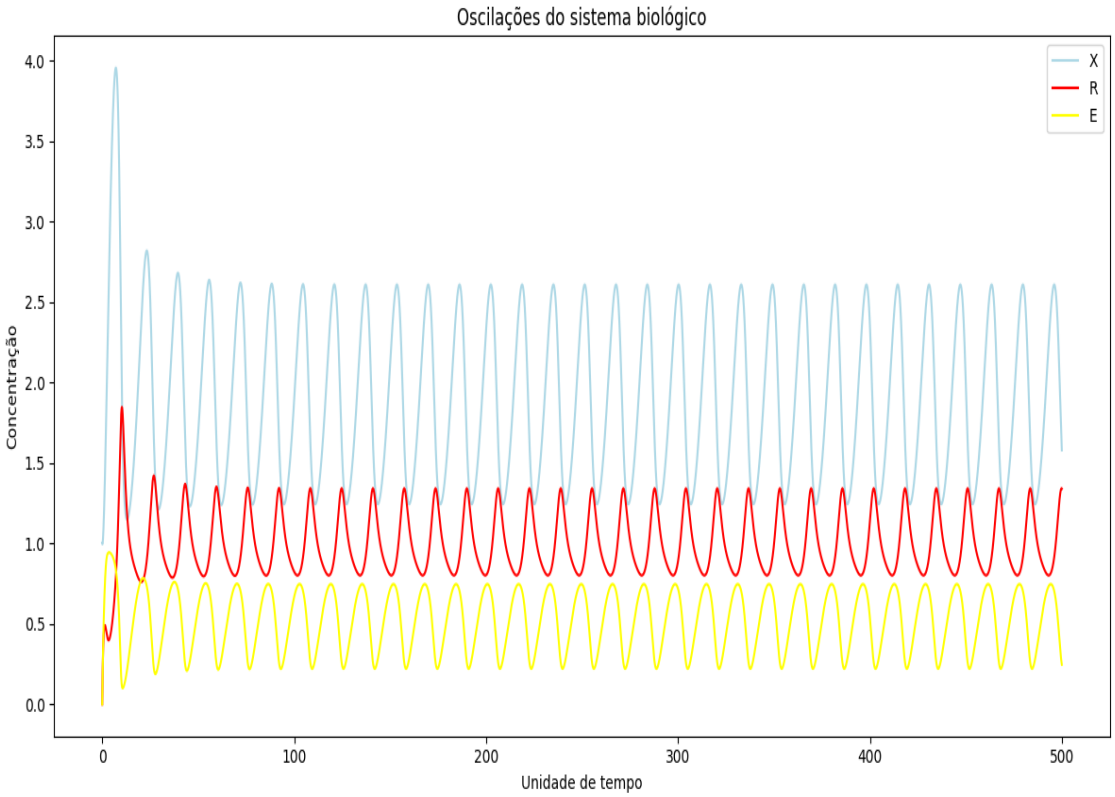


Figura 12: Gráfico representando a primeira rodada de oscilações testada. É bastante notável o crescimento vertiginoso da concentração de X, R e E nas primeiras unidades de tempo. Contudo, após o primeiro pico de X e E e o segundo de R, o sistema se estabiliza rapidamente, com oscilações entre concentrações mínimas e máximas praticamente constantes para cada composto

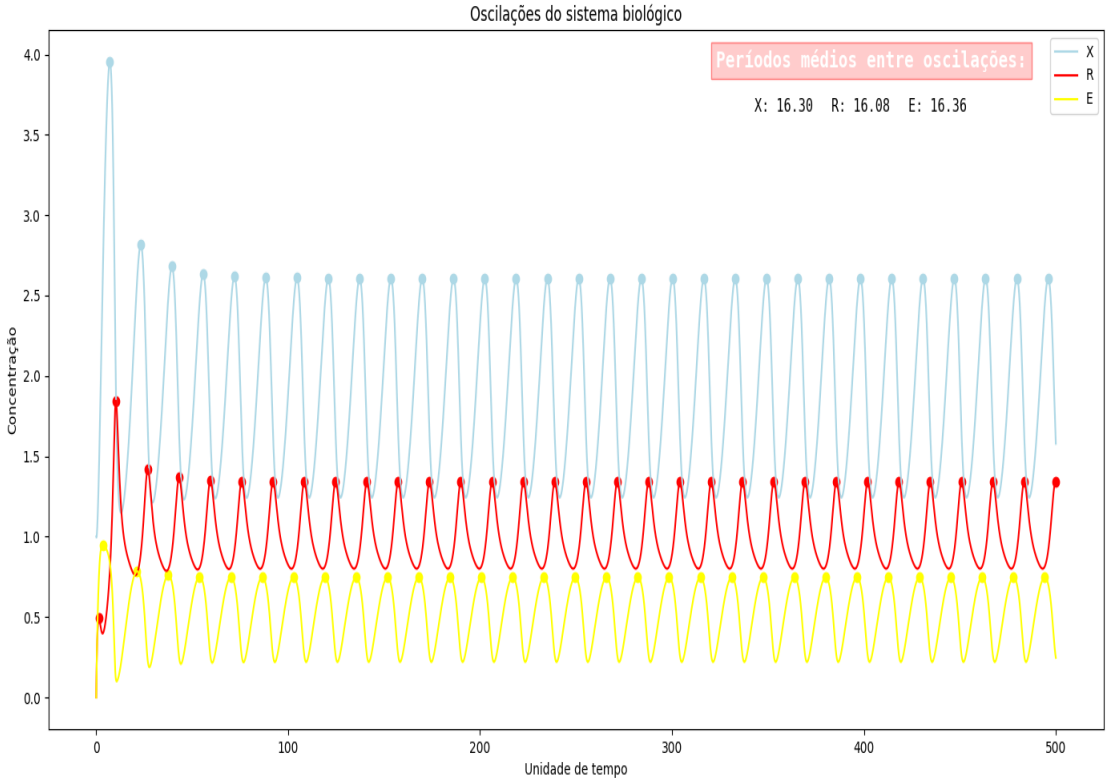


Figura 13: O mesmo resultado que o de antes, porém destacando os pontos de máximo e informando o período médio entre oscilações de cada composto. O período médio considerando todos os compostos foi de 16.24

dos parâmetros". Para tentar gerar resultados relevantes no mundo real, chamei a SearchLowerUpperPeriods com valores de lowerThreshold e upperThreshold de 14.10 e 18.10, que estão a aproximadamente 2 unidades a menos e a mais do caso inicial obtido na figura 12, buscando portanto pontos mais distantes do valor inicial. Utilizei também symmetricalSearch = True para usar a geração de parâmetros maiores e menores, e satisfiableNumber = 100 para buscar esse número de resultados (este caso não impede de retornar menos, mas garante a soma dos parâmetros por todos os valores do array).

Os resultados dessa execução foram aproveitados para gerar 3 gráficos de dispersão, um para a média de cada composto, conforme as figuras 14, 15 e 16. Nem todos os pontos nesses três gráficos estão a uma distância de mais de 2 unidades a os valores encontrados inicialmente, uma vez que SearchPeriods foi programada para retornar os resultados de uma iteração se pelo menos uma das 3 médias for diferente o suficiente. Ainda assim, esses gráficos permitem visualizar quais valores dos parâmetros são necessários para causar uma variação significativa na oscilação, além de ajudar a inferir outro valores para obter objetivos semelhantes. Um exemplo é o fato de que, como um valor de Et inicial de aproximadamente 1.5 gerou um período de oscilação médio de aproximadamente 23 para X, pode-se inferir que valores mais altos possam gerar períodos ainda maiores. Obviamente nessas inferências seria necessário levar em consideração a ação de todas as outras variáveis do sistema interagindo com o composto em questão.

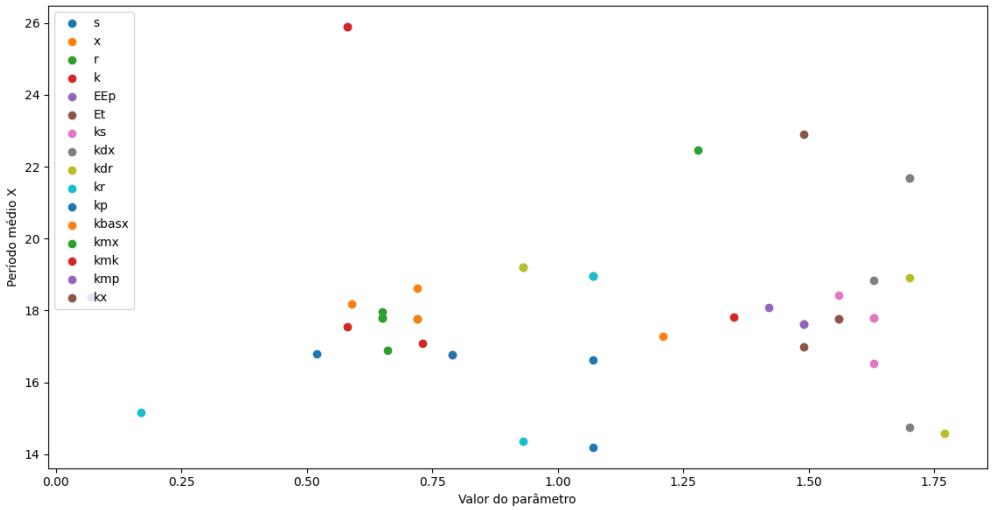


Figura 14: Gráfico de dispersão dos períodos encontrados para o composto X.

Por fim, aproveitando a série de adaptações que fiz por todas as funções, decidi amostrar aleatoriamente 3 casos do retorno de SearchPeriods e plotá-los assim como feito para a primeira execução, com o intuito de comprovar que as variações provocadas nos parâmetros podem gerar resultados relevantes. Esses plots são as figuras 17, 18 e 19.

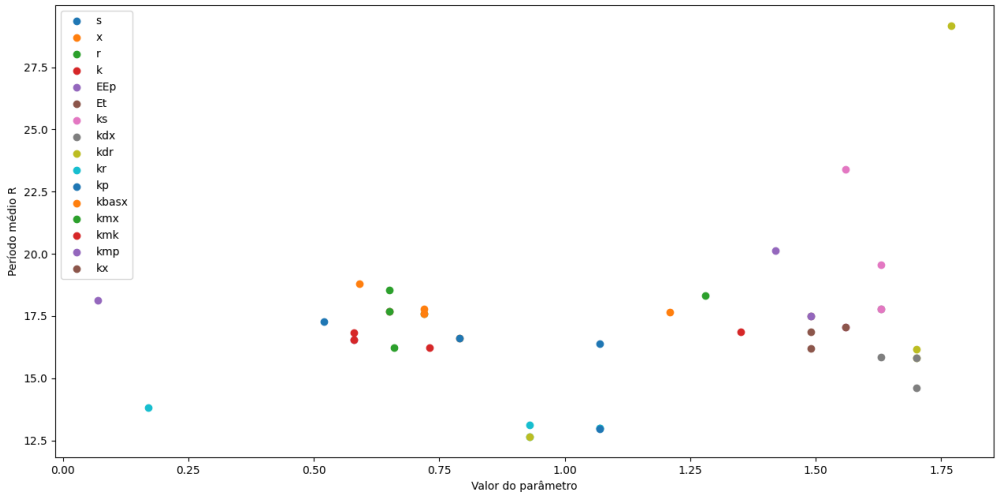


Figura 15: Gráfico de dispersão dos períodos encontrados para o composto R.

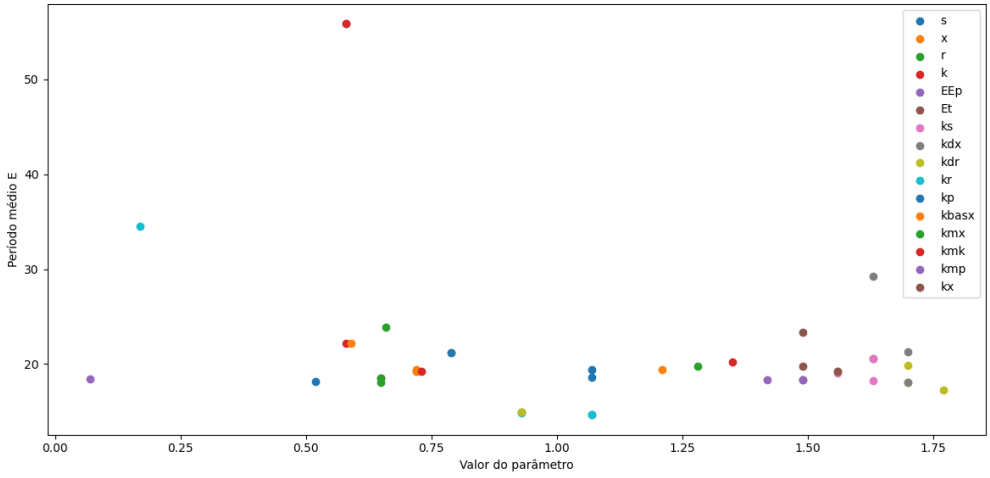


Figura 16: Gráfico de dispersão dos períodos encontrados para o composto E.

Figura 17: Primeira amostra aleatória

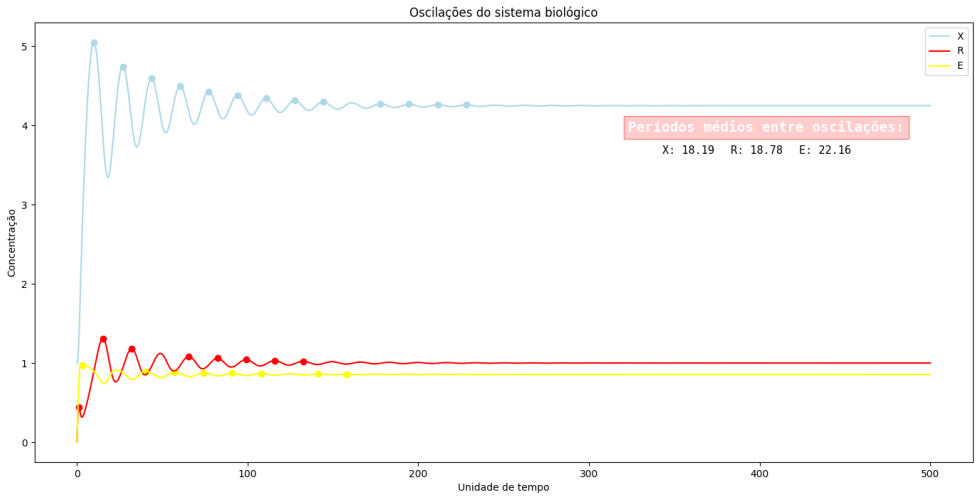


Figura 18: Segunda amostra aleatória

Figura 19: Terceira amostra aleatória