1 Introdução

O presente trabalho busca desenvolver um código numérico utilizando linguagem C# nos moldes de uma API capaz de realizar cálculos de viscoelasticidade aplicada a tecidos moles baseando-se nos modelos linear de Maxwell, quase-linear de Fung e não-linear de Schapery, além de gerar um arquivo externo em formato CSV com os resultados para comparar com os dados experimentais. Busca-se com este projeto auxiliar a pesquisa sobre viscoelasticidade em tecidos moles desenvolvida pelo professor Paulo Pedro Kenedi, sendo extraída desta os cálculos, rotinas e operações secundárias necessárias.

O conceito de viscoelasticidade é amplamente utilizado em diversos materiais, sendo empregado também em várias aplicações.

REVISÃO BIBLIOGRÁFICA SOBRE VISCOELASTICIDADE E COMENTAR SOBRE OS MODELOS.

No estudo desenvolvido por Stephanie Salles [6], é abordado os conceitos primordiais para implementar matematicamente a abordagem viscoelástica para descrever o comportamento, além de uma revisão bibliográfica detalhada para atingir esse objetivo. Seguindo nesta linha, R. F. Navarro [7] aborda as combinações possíveis entre as analogias mecânicas mola e amortecedor que caracterizam o comportamento viscoelásticos de materiais reais de uso no ramo industrial.

A importância do estudo da viscoelasticidade aplicada a tecidos não se restringe apenas à engenharia, tendo também importâncias nas áreas médicas, como demonstrado por Rossetto [8] que demonstra que esse conhecimento é de suma importância para melhor compreensão do comportamento dos tendões, permitindo que melhores análises possam ser feitas para determinar o treinamento físico, como nos casos de terapêuticos das tendinopatias. Além disso, também é visto grandes preocupações em se saber as forças aplicadas em tendões, como abordado por Bernardes et. all [9], que buscou determinar os parâmetros biomecânicos para o modelamento da articulação do joelho humano através de exercícios extensivos, junto de imagens raio-X obtidas por videofluoroscopia, e Zheng, et all, que, também em cima do joelho humano, que buscou modelar analiticamente das forças internas durante um exercício físico, além disso, usaram um programa para otimizar os resultados e minimizar os erros.

Neste estudo, diversos conceitos da área de ciência da computação serão visitados devido ao objetivo deste trabalho, sendo assim, vale citar cada um deles e explicá-los. Sendo o primeiro, a linguagem de programação escolhida, a linguagem C# que segundo a Microsoft [3], “é uma linguagem de programação moderna, orientada a objeto e de tipo seguro”, sendo comumente utilizada no desenvolvimento de sistemas por empresas por seu poder computacional, possuir uma vasta documentação ofertada gratuitamente pela Microsoft e por estar atrelada à metodologia ágil, que também vem sendo utilizado com frequência.

O próximo conceito é a API, sigla para Application Programming Interface, que traz consigo mais diversos outros conceitos. Segundo a Microsoft, a API [1] “especifica como os componentes e sistemas de software devem interagir uns com os outros” e, segundo TechTudo [2], corresponde a um conjunto de normas que possibilitam a comunicação entre sistemas através de uma série de padrões, como SOLID, que expõe princípios de boas práticas de programação orientada a objeto (POO) de modo a tornar o código limpo, de fácil de manutenção e escalável, e protocolos, como HTTP, que determina a estrutura básica de comunicação a ser usada por aplicações WEB. Vale salientar que esse protocolo e esse padrão foram utilizados no desenvolvimento do código para este trabalho com o objetivo de criar um código com grande poder computacional, de fácil compreensão e manutenibilidade.

Por utilizar os protocolos HTTP, pode-se chamar a aplicação desenvolvida para este estudo de API RESTful. [11][13] A sigla REST significa *Representational State Transfer*, que traduzido para o português significa Transferência Representacional de Estado, e, por sua utilização por completo na API, recebe o sufixo “ful”. Devido a isso, foram utilizados alguns recursos essenciais para o correto funcionamento da API RESTful, como entrada e saída de dados usando formatação JSON, chamadas HTTP através de *endpoint*, que possuem verbos HTTP específicos para cada tipo de tarefa, status HTTP na resposta da operação e estrutura de criação de URLs estáticas, sendo escolhido o framework ASP.NET MVC. Dessa forma, diversos conceitos são injetados neste contexto, sendo necessário uma melhor explicação sobre cada.

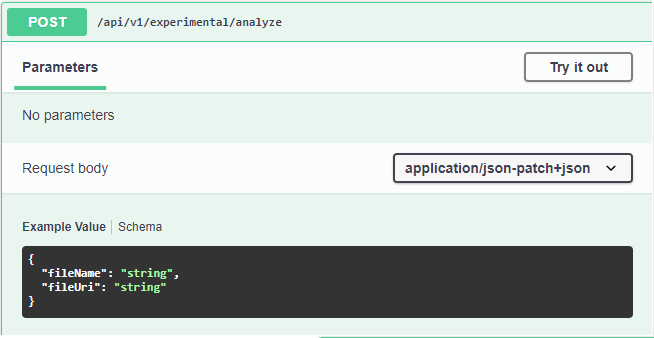
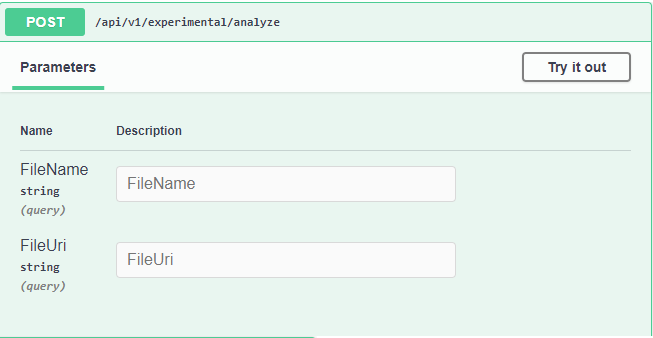
O primeiro conceito que pode ser comentado é JSON [12], sigla para *JavaScript Object Notation* ou, em português, Notação de Objetos JavaScript, que é uma formatação leve de troca de dados, por simples de ler e escrever para o ser humano e de fácil interpretação e geração para máquinas, é independente de linguagem, por usar convenções que são familiares a diversas linguagens de programação, conforme pode ser visto na figura 1.1. O JSON é construído em duas estruturas: uma coleção de pares nome-valor, conhecido no C# como objeto ou estrutura; e uma lista ordenada de valores, caracterizado como lista, sequência ou vetor. No contexto atual, será amplamente utilizada na serialização de objetos, isto é, converter uma cadeia de texto em um objeto utilizado internamente pelo código, e desserialização de objetos que é o caminho inverso da serialização, ambos são usados, principalmente, pelo Swagger na entrada e saída de dados, respectivamente.

Texto

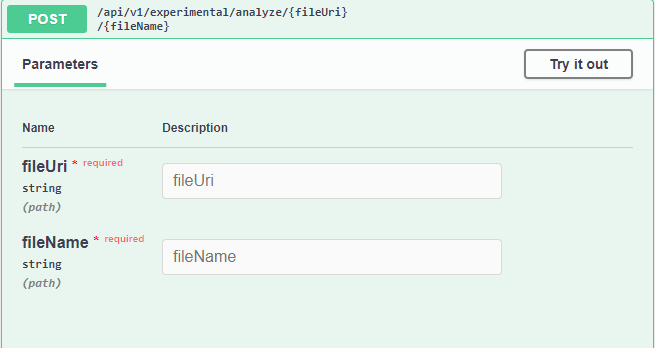
Descrição gerada automaticamente

Figura 1.1 – Exemplo de formatação JSON

Em seguida que pode ser citado é o *endpoint* [10][11], esse é um termo em inglês que pode ser traduzido para pontos de extremidade, que, no caso abordado neste documento, possui de fato esta característica, sendo o ponto de extremidade da aplicação. Nesse contexto, o endpoint corresponderá a uma URL por onde são feitas as requisições possibilitando que as operação em uma API sejam acessadas. Para efetuar essa requisição, é necessário definir dois parâmetros: o verbo HTTP definido pelo programador, que pode ser GET, para uma busca de dados, POST, para criar um dado, PUT, para atualizar algum dado, e DELETE, para excluir um dado; e o conjunto de dados da requisição passado pelo usuário. Este último pode preenchido de três maneiras diferentes: na URL, escrita juntamente à rota do *enpoint*; por *query*, em que no Swagger permite escrever cada parâmetro separado; e no corpo da requisição, usando formatação JSON.

(a) Corpo da requisição no formato JSON (b) Dados de requisição por *query*



(c) Dados de requisição a partir da URL

Figura 1.2 – Formas de passar o conjunto de dados da requisição no Swagger

O próximo conceito é o Swagger [9] que é uma ferramenta *open source* para desenvolvimento de APIs, facilitando no design, construção, documentação, teste e padronização, além disso permite ao usuário acessar os *endpoint* disponíveis através de uma interface simples e de fácil compreensão com documentações, exemplos e explicações sobre os recursos, conforme pode ser visto na figura 1.3.

COLOCAR AQUI UMA IMAGEM DO SWAGGER DA API DEPOIS DE TER AS DOCUMENTAÇÕES NELE

Figura 1.3 – Swagger da API desenvolvida para este estudo

Além disso, este recurso permite expor ao usuário a resposta detalhada da operação, que nada mais é do que uma cópia da estrutura da classe cuja responsabilidade única é conter essas informações, no caso deste estudo, todas as classes de resposta possuem a mesma estrutura com as seguintes propriedades: *Success*, que é do tipo booleano, ou seja, pode ser verdadeiro, quando a operação executar corretamente, e falso, caso falhe; HttpStatusCode, que contém o status HTTP da operação; *Errors*, que é uma lista onde são escritos os erros que ocorrem durante a aplicação, juntamente a um código pré-definido para facilitar a compreensão do motivo do erro; e *Data*, que contém os dados principais da resposta, variando através da necessidade de cada operação, já que em cada caso há informações diferentes que são importantes de retornar para o usuário.

Devido a sua grande gama de informações, vale informar que somente alguns status HTTP foram utilizados neste estudo, já que este pode informar diversos acontecimentos com a API, conforme demonstrado pela empresa Runscope [16] em seu site dedicado a isso. Os valores utilizados serão descritos abaixo na tabela 1.1 com sua nomenclatura, significado e dados de resposta obrigatórios.

|  |  |  |  |
| --- | --- | --- | --- |
| Status HTTP | Nomenclatura | Significado | Dados de resposta obrigatórios |
| 200 | OK | Operação executada com sucesso. | Não tem. |
| 201 | Created | Arquivo gerado com sucesso. | Caminho do arquivo. |
| 202 | Accepted | Dados de requisição válidos e aceitos para executar a operação. | Não tem. |
| 400 | Bad Request | Dados de requisição inválidos. | Motivo do erro descrevendo quais dados estão inválidos. |
| 401 | Unauthorized | Usuário não tem autorização para acessar o *endpoint*. | Não tem. |
| 404 | Not found | Nenhum dado foi encontrado de acordo com o filtro passado. | O dado que não foi encontrado. |
| 500 | Internal Server Error | Erro interno. | Descrição detalhada do motivo do erro. |
| 501 | Not Implemented | Recurso não implementado. | Descrição do recurso que não foi implementado. |

Para que a aplicação possa expor na WEB seus recursos, faz-se necessário a utilização de alguma ferramenta que possibilita isso, sendo escolhida para este trabalho o framework ASP.NET MVC, que possibilita criar sites estáticos com facilidade, sendo interpretado com facilidade pelo Swagger, logo, trabalhando em conjunto com este recurso. [14][15] ASP.NET é uma plataforma da Microsoft criada para desenvolvimento WEB e a sigla MVC vem de *Model View Controller*, sendo este conjunto responsável pela apresentação da aplicação, visando criar um código que não possua conexão forte entre as partes, pois facilita a manutenção e adição de funcionalidades. Dentro deste padrão, há três componentes principais: *model*, responsável por representar a entidade da lógica de negócio da aplicação, ou seja, a operação; *view*, responsável por apresentar uma interface para o usuário, que neste caso estará em conjunto com o Swagger; e *controller*, que é responsável pelo controle dos elementos, propiciando uma ligação entre eles. Um exemplo de código escrito no *controller* pode ser visto na figura 1.4, estando destaca em vermelho a parte que representa a operação e em azul a parte que representa os dados de entrada passados pelo usuário através do Swagger.

Texto

Descrição gerada automaticamente

Figura 1.4 – Exemplo de código presente no *controller*

Com o objetivo de simplificar e padronizar o código, foi desenvolvido um conjunto de classes que serviriam de base para todas as operações, de modo que cada operação da API só precisaria implementar seus parâmetros e lógicas específicas, evitando repetições de código. Dessa forma, todas as operações seguiriam o mesmo fluxograma apresentado na figura 1.5, em que, após a entrada dos dados a partir do Swagger, esses são validados no método *ValidateOperationAsync*, caso sejam válidos, segue para o método *ProcessOperationAsync*, onde a operação será de fato executada e, caso não sejam, encerra o processo retornando mensagens de erro e status HTTP 400. Além disso, foram criados contratos específicos que serviriam como base para os demais. Entende-se como contratos as classes que contêm os dados da requisição e a resposta e seus dados principais de uma determinada operação, sendo, para o presente trabalho, as classes com os sufixos *Request*, *Response* e *ResponseData*.

Diagrama

Descrição gerada automaticamente

Figura 1.5 – Fluxograma básico das operações

Com o objetivo de tornar a aplicação mais otimizada e rápida, foi utilizado a classe *Task*, nativa do C#, para que tarefas em paralelo e assíncronas fossem executadas com o objetivo de reduzir o tempo total gasto, sendo limitada a dez *threads* paralelas.

o conceito de paralelismo e tarefas assíncronas, permitindo que várias tarefas sejam executadas em paralelo otimizando o tempo gasto. Essa estratégia deve ser utilizada com cautela, visto que há um maior consumo de recursos de máquina, logo, deve ser feito uma análise de custo x benefício. Para isso, utilizou-se

EXPLICAR SOBRE PARALELISMO E TAREFAS ASSÍNCRONAS COM REFERENCIAS, EXPLICAR MELHOR O MOTIVO, EXPLICAR QUAL HTTP STATUS CODE DEVE SER UTILIZADO QUANDO ALGUMA OPERAÇÃO É EXECUTADA DE MANEIRA ASSÍNCRONA, EXPLICAR COMO FICA O FLUXO DA OPERAÇÃO QUANDO EXECUTADA DE MANEIRA ASSÍNCRONA.

[9] <https://swagger.io/about/> (29/04/2021)

[10] <https://milvus.com.br/o-que-e-endpoint/> (29/04/2021)

[11] <https://www.hostgator.com.br/blog/api-restful/> (29/04/2021)

[12] <https://www.json.org/json-pt.html#:~:text=JSON%20(JavaScript%20Object%20Notation%20%2D%20Nota%C3%A7%C3%A3o,3a%20Edi%C3%A7%C3%A3o%20%2DDezembro%20%2D%201999>. (29/04/2021)

[13] <https://www.redhat.com/pt-br/topics/api/what-is-a-rest-api> (29/04/2021)

[14] <https://docs.microsoft.com/pt-br/aspnet/overview> (29/04/2021)

[15] devmedia.com.br/introducao-ao-asp-net-mvc/31878 (29/04/2021)

[16] https://httpstatuses.com/

2 Recursos implementados

De modo a atender o objetivo deste trabalho, diversos recursos foram implementados, sendo eles enunciados resumidamente abaixo, separados em tópicos.

* Modelo experimental.
  + Análise dos dados experimentais.
  + Análise e extrapolação dos dados experimentais.
* Modelo Linear de Viscoelasticidade (modelo de Maxwell).
  + Cálculo da tensão e análise de sensibilidade das variáveis.
  + Cálculo da deformação e análise de sensibilidade das variáveis.
* Modelo Quase-Linear de Viscoelasticidade (modelo de Fung).
  + Cálculo da tensão e análise de sensibilidade das variáveis.
* Métodos numéricos.
  + Derivada.
  + Integral.
* Operações secundárias.
  + Diminuição da quantidade de pontos em um arquivo.

2.1 Modelo experimental

2.2 Modelo Linear de Viscoelasticidade

2.3 Modelo Quase-Linear de Viscoelasticidade

2.4 Métodos numéricos

2.4.1 Derivada

Derivada no ponto. Final menos inicial dividido pelo delta t.

2.4.2 Integral

Método de Simpson Composto por ter maior precisão e menor erro.

2.5 Operações secundárias

3 Modelo Linear de Viscoelasticidade de Maxwell

O modelo linear de viscoelasticidade de Maxwell considera que o tecido mole é representado por um amortecedor e uma mola em série, conforme demonstrado na figura 3.1.

Diagrama

Descrição gerada automaticamente

Figura 3.1

4 Modelo Quase-Linear de Viscoelasticidade de Fung

O modelo de Viscoelasticidade Quase-Linear de Fung aplica a não-linearidade da relação entre tensão e deformação expressando a tensão em duas partes: a função relaxação reduzida, que depende somente do tempo, e a resposta elástica, que depende da deformação, em que as equações utilizadas para cada parâmetro serão demostradas a seguir. Para facilitar os cálculos, este último pode ser expresso dependendo somente do tempo, já que a deformação depende deste parâmetro.

Ademais, para este trabalho, serão feitas duas ponderações relacionadas ao tempo de rampa: considerar e desconsiderar. Ao considerar o tempo de rampa, é possível considerar diversas relaxações, pela presença das variáveis A e B demonstradas nos cálculos da resposta elástica que só são possíveis de serem obtidas ao considerar este tempo, para este projeto foi considerado somente duas relaxações. Enquanto isso, ao desconsiderar o tempo de rampa, só será possível considerar uma relaxação devido à ausência das variáveis supracitadas.

4.1 DEFORMAÇÃO

Para a deformação, é utilizado a equação 4.1 quando é, de modo a se assemelhar a como é feito experimentalmente no laboratório, e a equação 4.2 quando o tempo de rampa não é considerado. Os parâmetros e representam, respectivamente, a taxa de deformação aplicada durante o experimento, sendo para o caso de aumento na deformação e para o caso de diminuição da deformação, e o tempo de rampa, além disso e representam os tempos em que a deformação é alterada, seja aumentando, diminuindo ou ficando constante.

|  |  |
| --- | --- |
|  | 4.1 |
|  | 4.2 |

Desse modo, pode-se calcular a derivada da deformação, que mais a frente será demonstrada sua importância ao calcular a tensão. Derivando a equação 4.2, temos:

|  |  |
| --- | --- |
|  | 4.3 |

Sendo assim, a derivada somente será diferente de zero quando a deformação não for constante.

4.2 RESPOSTA ELÁSTICA

A resposta elástica pode ser calculada através da equação 4.4, sendo em função da deformação, porém, como supracitado, esta é em função do tempo, a resposta elástica pode ser reescrita também em função somente do tempo. Os parâmetros A e B apresentados na equação representam, respectivamente, *elastic stress constant* e *elastic stress power*. Vale salientar que mesmo que essa propriedade dependa da deformação, que para diferentes intervalos de tempo são usadas diferentes equações, pode-se usar somente a equação 4.4, além disso, observando a equação 4.1, pode-se perceber que em alguns intervalos de tempo a resposta elástica será constante e não dependerá do tempo.

|  |  |
| --- | --- |
|  | 4.4 |

Além disso, para o caso em que o tempo de rampa é desconsiderado, é usado o valor de tensão inicial aplicado ao experimento, .

|  |  |
| --- | --- |
|  | 4.5 |

Assim como feito para a deformação, também será calculado a derivada da resposta elástica, devido ao mesmo motivo, sua importância no cálculo da tensão, sendo assim, derivando a equação 4.4.

|  |  |
| --- | --- |
|  | 4.6 |

Para esta equação, vale a mesma consideração feita para a 4.4, de que, mesmo tendo grandes oscilações das equações usadas para a deformação e sua derivada apresentando, é válido utilizar para fins computacionais a equação 4.6. Porém, vale relembrar que só é diferente de zero para os casos em que a deformação não é constante, sendo assim, é somente nesses momentos em que a derivada da resposta elástica é diferente de zero.

4.3 FUNÇÃO RELAXAÇÃO REDUZIDA

A função relaxação reduzida corresponde a parcela viscosa das equações de viscoelasticidade, podendo ser escrita de duas maneiras segundo Fung [3], a primeira, equação 4.7, é chamada de simplificada por apresentar maior facilidade de ser obtida em experimentos por ser representada por uma séria de somas de exponenciais, enquanto a segunda, equação 4.8, foi obtida a partir de diversos cálculos matemáticos mais complexos baseados no modelo de Kelvin (*standard linear solid*). Vale ressaltar que para o presente trabalho será considerado que a equação 4.7 é composta por somente três somas de exponenciais, pois, conforme COLOCAR AQUI O MOTIVO DE ESTAR USANDO SOMENTE 3 EXPONENCIAIS. Ademais, vale lembrar que para o tempo igual a zero, a função relaxação reduzida é igual a 1, .

Os parâmetros C e correspondem, respectivamente, a constante de relaxação e o tempo de relaxação, para a equação 4.7, os índices são somente para diferenciar cara constante, enquanto para a equação 4.8, representa o tempo de relaxação rápido e , o tempo de relaxação lendo, sendo sempre maior do que .

|  |  |
| --- | --- |
|  | 4.7 |
|  | 4.8 |

Como , pode-se afirmar que , sendo assim, pode ser reescrito como:

Além disso, temos que:

Logo:

Por fim, temos que a equação 4.8, pode ser reescrita como:

Simplificando, temos:

|  |  |
| --- | --- |
|  | 4.9 |

Como feito nas propriedades anteriores, será calculada a derivada da função relaxação reduzida para as equações 4.7 e 4.9.

Derivando a equação 4.7:

|  |  |
| --- | --- |
|  | 4.10 |

Derivando a equação 4.7:

Aplicando a definição de cálculo para derivada de uma integral determinada:

Em que:

Temos que:

Logo:

Portanto:

|  |  |
| --- | --- |
|  | 4.11 |

4.4 TENSÃO

Para se calcular a tensão existem 3 diferentes formas de acordo com Fung [3], sendo estas expressas abaixo.

|  |  |
| --- | --- |
|  | 4.12 |
|  | 4.13 |
|  | 4.14 |

Conforme demonstrado anteriormente, a resposta elástica e a função relaxação reduzida dependem somente do tempo, o que permite reescrever as derivadas parciais como derivadas totais. Ademais, temos que e .

|  |  |
| --- | --- |
|  | 4.15 |
|  | 4.16 |
|  | 4.17 |

Segundo Fung [3], essas equações são matematicamente equivalentes, logo, em teoria, devem retornar os mesmos valores ou resultados muito próximos. Sendo assim, optou-se por implementar essas três equações retornando seus valores no mesmo arquivo de resposta. Entretanto, vale ressaltar que a equação 4.15 é amplamente utilizada em diversas outras teses, sendo considerada a principal para este estudo e utilizada como base para comparação dos resultados obtidos nas demais equações. Conforme será apresentado a seguir, a utilização destas equações sem quaisquer tratamentos não apresenta viabilidade por impactar negativamente no tempo de execução do código, sendo necessário reescrevê-las com o objetivo de diminuir otimizar o intervalo de tempo da integral. Isso está descrito com melhores detalhes do Anexo 1 deste documento.

4.5 IMPLEMENTAÇÃO NUMÉRICA

A implementação numérica da operação principal do modelo de Fung, que é a responsável por calcular e escrever no arquivo a tensão e todos os outros parâmetros que esta depende, a saber: deformação, resposta elástica e função relaxação reduzida, seguiu-se o fluxograma apresentado na imagem 4.1. Vale lembrar, que este modelo, assim como os outros, não possui somente uma operação, já que o intuito deste trabalho não é somente analisar a tensão e, sim, ter uma visão mais ampla quanto aos parâmetros que a influenciam.

Conforme supracitado, antes de entrar no fluxo da operação, os dados de entrada são recebidos através do Swagger no formato JSON e são desserializados para a classe responsável por conter os dados de requisição. Após isso, conforme demonstrado no fluxograma da imagem 4.1 por *Build Input Data*, faz-se necessário construir os dados de entrada do modelo que apresenta um formato mais otimizado para se trabalhar internamente no código. Essa divergência de formatos ocorre pois os dados de requisição devem possuir um formato de fácil entendimento e preenchimento por parte de quem utilizará a API, sendo este formato, nem sempre, o ideal a ser utilizado para executar as operações. Vale salientar que, nos dados de requisição para esta operação, é possível receber uma lista contendo parâmetros de diferentes tecidos moles, dessa forma, no passo de construção dos dados de entrada para o modelo, é criada uma lista contendo essa informação, em que os próximos passos serão executados usando cada elemento dentro dessa lista. Para evitar um grande tempo de execução do código, foi utilizado o recurso de *threads* paralelas, usando a classe *Task* nativa do C#, permitindo que tarefas sejam executadas de maneira assíncrona, otimizando o tempo gasto, entretanto, usando mais recurso do computador. Para fins de conhecimento, isso apresentou-se vantajoso, devido a sua grande redução no tempo de execução, que era na ordem minutos e, em casos mais extremos, horas, foi reduzido para segundos.

Conforme dito anteriormente, as próximas etapas são feitas de maneira assíncrona para cada elemento construído no passo anterior, conforme demonstradas na imagem 4.1. A etapa mais importante que merece mais detalhamento é a sub-rotina *Calculate results* que, conforme o nome sugere, calcula de fato os resultados para o modelo. Na operação, essa sub-rotina está implementada em apenas uma linha de código em que é feita uma chamada ao método *CalculateResultsAsync*, implementado na classe *QuasiLinearViscoelasticityModel,* que é responsável por calcular de maneira paralela a deformação, a função relaxação reduzida, podendo ser a simplificada ou não, dependendo de qual *endpoint* foi utilizado, a resposta elástica e a tensão, esta última usando as três equações descritas anteriormente.

O código foi estruturado dessa forma se baseando no conceito de responsabilidade única, um dos princípios SOLID para programação orientada a objetos, em que cada classe deve ter somente uma responsabilidade. Sendo assim, por isso foi criado duas classes, uma que orquestra o fluxo da operação e outra que contém as lógicas para implementar as equações do modelo quase-linear. Esta estratégia apresenta grandes vantagens por evitar código repetido centralizando algumas lógicas e organizando de maneira mais eficiente o código.

Diagrama

Descrição gerada automaticamente

Imagem 4.1 - Fluxograma da operação que calcula a tensão para modelo Quase-Linear de viscoelasticidade

Tendo a operação já implementada, resta agora desenvolver a classe responsável por conter as equações para este modelo. Nesta, foram criados métodos específicos que calculam a deformação, resposta elástica e função relaxação reduzida, além de suas respectivas derivadas, além da tensão, calculada de 3 maneiras diferentes, conforme demonstrado acima.

No decorrer desta implementação, percebeu-se que as equações utilizadas para a tensão precisavam ser otimizadas com o objetivo de reduzir o tempo de execução. Para isso, foram adotadas duas soluções, a primeira foi de executar cada cálculo de maneira paralela, usando a classe *Task* explicada anteriormente, a segunda foi reescrever as equações utilizadas com objetivo de encontrar equações específicas para cada intervalo de tempo, conforme demonstrado no Anexo 1 deste documento. Estas propostas de fato auxiliaram e atingiram seu objetivo, entretanto, para a segunda solução, inicialmente eram utilizadas vários *ifs* no código o que o torna difícil de manutenção e prejudicava a compreensão. Para sanar este problema, aplicou-se o *design pattern* *Strategy*, que, segundo Refactoring Guru [4], “é um padrão de projeto comportamental que permite que você defina uma família de algoritmos, coloque-os em classes separadas, e faça os objetos deles intercambiáveis”, isto é, é um padrão de projeto que permite a segmentação do código de modo a ter partes separadas com lógicas diferentes e que são escolhidas baseadas no contexto. Sendo assim, foi desenvolvido uma classe para cada intervalo de tempo contendo as equações específicas deste período, além de um orquestrador, que baseado no tempo decide qual classe deve ser utilizada. Vale lembrar que todo o código utilizado para este trabalho se encontra no GitHub, no repositório referenciado neste documento [5].

5 Modelo Não-Linear de Viscoelasticidade de Schapery

Bibliografia

[1] (27/04/2021) <https://azure.microsoft.com/mediahandler/files/resourcefiles/apis-microservices-ebook/Azure_API-Microservices_eBook.pdf>

[2] (27/04/2021) <https://www.techtudo.com.br/listas/2020/06/o-que-e-api-e-para-que-serve-cinco-perguntas-e-respostas.ghtml>

[3] Livro do Fung

[4] (27/04/2021) <https://refactoring.guru/pt-br/design-patterns/strategy>

[5] <https://github.com/M3110/SoftTissue>

[6] RESEARCH ON VISCOELASTICITY MODELS FOR HUMAN KNEE LIGAMENTS

[7] Modelos Viscoelásticos Aplicáveis a Materiais Reais: uma Revisão

[8] A viscosidade no alongamento de tendões (IMPRESSO)

[9] Determinação de parâmetros biomecânicos para o modelamento da articulação do joelho.

ANEXO 1 – Descrição das equações usadas para modelo de viscoelasticidade Quase-Linear

Equações usadas para deformação e resposta elástica:

A1.1 Considerando tempo de rampa

A1.1.1

A1.1.1.1

|  |  |
| --- | --- |
|  | A1.1 |

A1.1.1.2

|  |  |
| --- | --- |
|  | A1.2 |

A1.1.1.3

|  |  |
| --- | --- |
|  | A1.3 |

A1.1.1.4

|  |  |
| --- | --- |
|  | A1.4 |

A1.1.1.5

|  |  |
| --- | --- |
|  | A1.5 |

A1.1.1.6

|  |  |
| --- | --- |
|  | A1.6 |

A1.1.2 e

Para estas equações, não foram encontradas vantagens em reescrevê-las por terem a parcela , que dificultou drasticamente, já que a equação utilizada para a resposta elástica não dependeria somente do limite de integração ou somente do tempo, fazendo com que, em um limite de integração arbitrário, mais de uma equação para a resposta elástica poderia ser utilizada, diferente do que ocorria a equação .

A1.2 Desconsiderando tempo de rampa

Ao desconsiderar o tempo de rampa, temos que a resposta elástica sempre será constante, logo, sua derivada é igual a zero.

A1.2.1

Para esta equação, não será possível utilizá-la já que, como a resposta elástica é constante, sua derivada será igual a zero, logo, para todo o domínio de tempo, essa equação será igual a zero.

A1.2.2

A1.2.3