XIII SIMMEC
Simpósio de Mecânica Computacional
29 de Outubro a 1º de Novembro de 2018
UFES - Campus Goiabeiras - Vitória, ES

FRAMEWORK DE TESTE UNITÁRIO PARA UM SOFTWARE DE BIOMECÂNICA COMPUTACIONAL BASEADO EM ELEMENTOS FINITOS

Unit testing framework for finite element based computational biomechanics software

Gilmar Ferreira da Silva Filho (1); Joventino Oliveira Campos (2); Bernardo Martins Rocha (3)

(1) Pós-Graduação em Modelagem Computacional, Universidade Federal de Juiz de Fora - MG, Brasil.
(2) Pós-Graduação em Modelagem Computacional, Universidade Federal de Juiz de Fora - MG, Brasil.
(3) Departamento de Ciência da Computacão e Pós-Graduação em Modelagem Computacional, Universidade Federal de Juiz de Fora - MG, Brasil.
Email para Correspondência: bernardomartinsrocha@ice.ufjf.br

Resumo: O desenvolvimento de softwares científicos, como, por exemplo, para simulações computacionais de biomecânica baseadas no método dos elementos finitos, se dá habitualmente com o objetivo de atingir metas específicas de uma pesquisa de interesse. Com isso, práticas pertinentes de engenharia de software, como o teste unitário, são desconsideradas e o produto final é uma aplicação que provê a resposta esperada, porém é suscetível a falhas, erros e sobretudo de pobre manutenibilidade. Este trabalho tem como objetivo aplicar e avaliar técnicas de teste unitário no desenvolvimento de um software que simula a atividade mecânica cardíaca baseado no método dos elementos finitos, de forma a garantir um software mais flexível, de fácil manutenção, reúso e depuração, e que assegure resultados mais confiáveis. A biblioteca para testes unitários automatizados *CppTest* foi utilizada em um software de elementos finitos desenvolvido em *C++* para solução de problemas de mecânica computacional. Problemas com solução analítica e com soluções de referência do tipo *benchmark* foram utilizados como base para o desenvolvimento de rotinas de teste. Além disso, outras rotinas de teste unitário dos módulos fundamentais do software (módulos de álgebra linear, entrada e saída de dados, entre outros) foram desenvolvidas. A aplicação de técnicas de teste unitário mostrou-se importante para esse tipo de software permitindo gerar relatórios e identificar erros na etapa de desenvolvimento e extensão do software. Verificou-se que esta abordagem de testes para o software científico apresenta desafios que não são encontrados em outras aplicações, dada a natureza dos softwares científicos. Ainda assim, tais técnicas se mostraram pertinentes e justificáveis, visto que o seu emprego possibilitou a obtenção de um *software* de fácil extensibilidade e depuração, tornando possível, por exemplo, a inclusão de algoritmos mais eficientes ou novos modelos matemáticos sem danificar o funcionamento do software.

Palavras chaves: elementos finitos; biomecânica; teste unitário de software.

XIII SIMMEC
Simpósio de Mecânica Computacional
29 de Outubro a 1º de Novembro de 2018
UFES - Campus Goiabeiras - Vitória, ES

Abstract: Scientific software development, such as for computational simulations of biomechanics based on the finite elements method, is usually done in order to achieve specific goals of a research. Therefore, pertinent practices of software engineering, such as unit testing, are not considered and the final product is an application that provides the expected answer, but is susceptible to failures, errors and of poor maintenance. The present work aims to apply and evaluate unit testing techniques in the development of a software that simulates cardiac mechanical activity based on the finite elements method, in order to guarantee a more flexible software that provides easy maintenance, reuse and debug, and that ensures reliable results. The CppTest library for automated unit testing was used in a finite elements software developed in C++ for solving computational mechanics problems. Benchmark problems with analytical solution and reference solutions were used as a basis for the development of testing routines. In addition, other unit testing routines of the fundamental modules of the software (linear algebra modules, data input and output, among others) have been developed. The application of unit testing techniques proved to be important for this type of software, allowing to generate logs and identify errors in the stage of software development and extension. It has been found that this testing approach to scientific software presents challenges that are not encountered in other types of applications, given the nature of scientific software. Nevertheless, such techniques have proved to be relevant and justifiable, since their use made it possible to obtain a software that is easier to extend and debug, making it possible, for example, to include more efficient algorithms or new mathematical models without damaging the software.

Keywords: finite elements; biomechanics; software unit testing.

1 INTRODUÇÃO

A eletrofisiologia e mecânica do coração são alvos de amplo estudo, e com esforços direcionados ao entendimento da complexa natureza da atividade cardíaca, tem-se atualmente modelos matemáticos robustos e modelos computacionais correspondentes bastante avançados. Modelos com alto nível de complexidade para a simulação tanto da atividade elétrica (Vigmond et al., 2008) quanto do funcionamento mecânico e eletromecânico acoplado (Kerckhoffs et al., 2006) do coração compõem um cenário amadurecido dotado dos mais variados softwares científicos que os implementam se aproveitando das arquiteturas computacionais de alto desempenho.

Neste sentido, dada a complexidade e magnitude destes modelos, a verificação da precisão e confiabilidade dos mesmos se dificulta, visto que os códigos das aplicações tendem a ficar progressivamente intrincados e que os modelos matemáticos resolvidos por estes softwares raramente apresentam soluções analíticas. Aliado a isto, softwares científicos que simulam a atividade do coração produzem resultados críticos, que podem eventualmente contribuir para o avanço da saúde humana. Desta forma, aplicações deste tipo devem possuir código confiável que assegure resultados aceitáveis.

A confiabilidade e segurança de softwares científicos são comumente avaliadas por meio de processos de verificação, que visam determinar a precisão de um programa ao resolver os modelos matemáticos de interesse, bem como processos de validação e quantificação da incerteza, que tentam respectivamente, apontar a capacidade de um modelo matemático de representar o fenômeno de interesse e quantificar incertezas associadas ao cálculo do resultado de um modelo (National Research Council, 2012). Por outro lado, apesar de poucas práticas de engenharia de software serem realmente utilizadas no escopo científico, a aplicação destas pode ser efetiva na garantia de qualidade de softwares científicos (Kelly et al. 2009), e apesar disto desenvolvedores de aplicações científicas negligenciam práticas como o teste automatizado de software (Koteska et al., 2015), incluindo a comunidade biomédica, como sugere Kane et al., 2006.

Testes unitários automatizados buscam dividir o código em unidades individuais que podem ser vistas como a menor parte testável de um código. Um framework de testes unitários é capaz então de executar diversos testes nestas unidades mínimas de código e gerar relatórios que apontam os erros encontrados. O emprego de testes automatizados de software no desenvolvimento de programas científicos pode atacar problemas que não são detectados pelos métodos de verificação e validação e, portanto, trazer benefícios como a garantia de reusabilidade e extensibilidade de código tal como facilidade de depuração e manutenção do código, além de um desenvolvimento otimizado (Kanewala e Bieman, 2014). O desenvolvimento orientado a testes se mostra benéfico inclusive em meio à modelagem cardíaca, possibilitando a obtenção de códigos mais maleáveis e extensíveis (Pitt-Francis et al., 2008).

Deve-se atentar para o fato de que aplicações científicas impõem obstáculos para a utilização destas técnicas de engenharia de software. Compõem estes obstáculos, as

diferenças culturais entre a comunidade científica e a comunidade de engenharia de software assim como os problemas inerentes à execução de testes em softwares científicos (Kanewala e Bieman, 2014), como a dificuldade de se escolher um conjunto de testes que sejam de fato rigorosos e efetivos para avaliar o software como um todo ou a dificuldade de se encontrar escopos que possam ser considerados unidades mínimas testáveis de código.

Os métodos de verificação e validação dos modelos computacionais também não devem ser negligenciados, posto que são essenciais para a análise de precisão de confiabilidade de programas científicos (Hatton e Roberts, 1994). Como não existem soluções analíticas para os modelos matemáticos que governam a atividade elétrica e mecânica cardíaca, simulações do tipo *benchmark* constituem bons aparatos para a verificação de softwares neste contexto, pois definem problemas padronizados que ao serem simulados devem convergir para a mesma solução. Niederer et al., 2011 avaliaram a simulação da eletrofisiologia cardíaca por meio da definição de um benchmark verificado em 11 diferentes plataformas de simulação desenvolvidas por diferentes grupos de pesquisa, obtendo soluções referência. Por outro lado, para a verificação da atividade mecânica cardíaca, Land et al., 2015 fornecem 3 problemas do tipo *benchmark* simulados em outras 11 plataformas computacionais e geram soluções de referência para verificação.

Diante deste contexto, onde testes de software se mostram benéficos ao desenvolvimento de aplicações científicas e a verificação e validação de modelos computacionais são essenciais para que se assegure a precisão do resultado, o presente trabalho aplica as técnicas de teste unitário de software para um simulador baseado no método dos elementos finitos que modela a atividade mecânica cardíaca, se aproveitando de um framework de testes unitários para inclusive, acoplar os problemas de benchmark da atividade mecânica do coração aos testes do simulador. Verificou-se a importância do uso destas técnicas para esse tipo de software, de maneira que foi possível a obtenção de relatórios e a identificação de erros na etapa de desenvolvimento e de possíveis extensões do software para se adotar diferentes métodologia para a solução desses problemas.

2 MÉTODOS

2.1 Teste unitário e verificação de código

A modelagem computacional da atividade cardíaca resulta em códigos complexos por conta das características multi-escala e multi-físicas, dos fenômenos envolvidos. Softwares desta natureza normalmente empregam técnicas computacionais como o método dos elementos finitos (Hughes, T. J. R., 2000) para a resolução das equações dos modelos matemáticos que representam o fenômeno, implicando na necessidade da utilização de métodos numéricos e estruturas de dados eficientes para a implementação computacional. Tal implementação é custosa do ponto de vista computacional, e deve portanto, estar constantemente buscando novos modelos e algoritmos numéricos mais eficientes. Desta forma, é desejável que o código destes softwares seja de fácil

depuração e manutenção, garantindo extensibilidade, e também confiabilidade. Estas características podem ser obtidas através de testes automatizados de software e verificação de código.

O teste unitário de software é uma avaliação conduzida de maneira a garantir a qualidade e o correto funcionamento do software. O teste unitário de um software se dá através da definição de unidades mínimas do sistema, as quais são avaliadas por meio de testes automatizados que serão capazes de localizar os problemas na implementação do código. A implementação de bons testes garantirá que cada funcionalidade do sistema seja avaliada, de forma que defeitos no software serão capturados por uma pequena quantidade de testes, e os testes serão capazes de indicar a localização do erro no código. Com isto, é possível restringir o sistema de tal forma que a detecção de modificações indesejadas se dá facilmente, os defeitos ficam localizados e o risco da introdução de erros no software diminui. Estas propriedades ajudam na obtenção das características almejadas de manutenibilidade, extensibilidade e facilidade de depuração de código.

Em softwares científicos, operações de álgebra linear, estruturas de dados específicas, malhas computacionais e modelos numéricos com soluções analíticas podem ser considerados unidades de código facilmente testáveis. Existem, no entanto, certos desafios quanto ao teste unitário de determinados aspectos de uma aplicação científica (Pitt-Francis et al., 2008). Na simulação da atividade cardíaca, descrita por equações diferenciais parciais, alguns dos principais problemas encontrados são a dificuldade de testar modelos numéricos cujas soluções analíticas são inexistentes, a dificuldade de elaborar testes para rotinas de código que possuem muitas operações e decisões acopladas ou que podem apresentar falhas relacionadas à aritmética de ponto flutuante e erros de truncamento e arredondamento, a difícil determinação de tolerâncias razoáveis para algoritmos numéricos, e a escolha de parâmetros adequados.

Uma forma de testar o código de modelos com soluções analíticas inexistentes é por meio da verificação de código. A verificação de código se dá através da avaliação das propriedades de convergência em tempo e espaço para um problema benchmark padronizado. O uso de um problema padronizado significa que todas os códigos de simulações devem convergir, em algum sentido, para a mesma solução à medida que o espaçamento e os passos de tempo são reduzidos, independentemente do método numérico, arquitetura de computador ou linguagem de programação (Niederer et al., 2011). Soluções de referência são obtidas através da avaliação de múltiplos sistemas simultaneamente para um único problema. Esta estratégia é chamada avaliação de código N-version (Hatton e Roberts, 1994).

2.2 Framework de testes unitários

Uma rotina de teste unitário de software se inicia com o estabelecimento de parâmetros e a configuração de recursos (arquivos de entrada, objetos, entre outros). Em seguida a unidade de código a ser testada é chamada e submetida aos parâmetros e recursos previamente obtidos. O retorno da unidade de código testada é então averiguado por meio de certas condições de teste e caso o código atenda aos requisitos

do teste a rotina libera os recursos e termina, caso contrário um alerta é sinalizado, indicando que o código falhou no teste. A Figura 1 ilustra este procedimento.

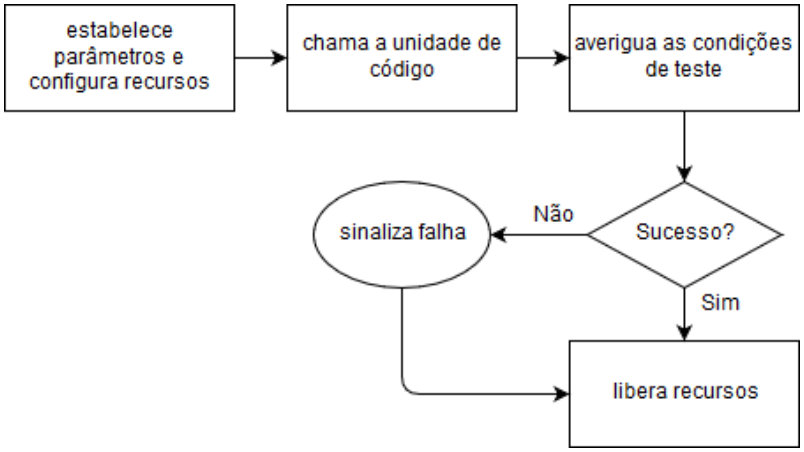


Figura 1. Rotina de teste unitário.

Normalmente, realizam-se diversos testes unitários que devem ser executados em sequência sem que uma eventual falha termine a execução dos testes. Linguagens de programação costumam possuir frameworks de testes unitários que realizam estas tarefas. Estes frameworks fornecem uma estrutura que simplifica a criação e execução de rotinas de teste, permitindo a organização lógica dos testes e fornecendo um conjunto de métodos de testificação dos resultados das chamadas às unidades de código. Ao fim da execução dos testes, um sumário com os resultados é gerado, possibilitando a fácil verificação dos defeitos encontrados no código.

Para a implementação dos testes unitários do software da simulação da atividade mecânica cardíaca neste trabalho optou-se pelo framework de testes unitários para a linguagem C++ denominado CppTest (Lundell, 2017). O CppTest é um framework simples, porém robusto. É focado na usabilidade e extensibilidade. Nele as rotinas de testes devem ser agrupadas em classes de teste que provêm macros para a testificação das unidades de código e geram relatórios de testes em diversos formatos, incluindo HTML.

2.3 Problemas benchmark para mecânica cardíaca

Testes capazes de verificar as soluções do simulador para a mecânica cardíaca são uns dos mais importantes a serem executados tendo em vista a complexidade do problema descrito por um sistema de equações diferenciais parciais não-linear e com outras propriedades que dificultam sua solução. Em alguns casos simplificados, pode-se encontrar solução analítica para as equações da mecânica não-linear, mas quando se considera o material do tecido cardíaco e a anisotropia das fibras não existem soluções analíticas para o problema. Nesse caso, a verificação através de problemas de benchmark, em particular os da mecânica cardíaca propostos por Land et al., 2017, é extremamente importante para verificar a implementação do software que simula este fenômeno. Nesse sentido, a implementação desses problemas benchmark da atividade

mecânica cardíaca foi embutida nos testes unitários implementados neste trabalho. A Figura 2 mostra a geometria dos problemas e exemplos de soluções para os benchmarks.

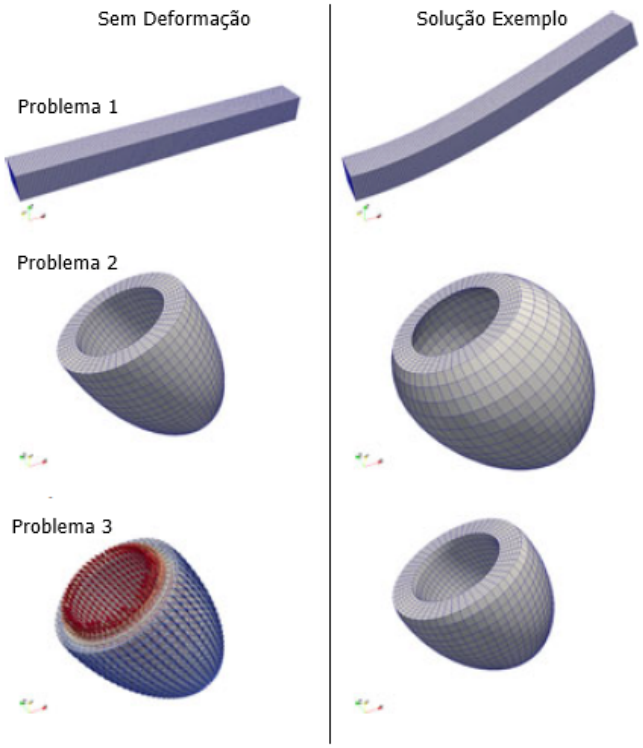


Figura 2. Problemas benchmark e suas soluções.

Fonte: (Campos et al., 2018)

Cada um dos 3 problemas propostos testam aspectos importantes para simuladores da mecânica cardíaca. O primeiro problema consiste da deformação de uma barra retangular. O problema testa a aplicação de carregamento do tipo pressão para as quais a direção muda de acordo com a orientação da superfície deformada, a implementação correta da mudança das direções de fibras com a deformação, a lei constitutiva transversalmente isotrópica e as condições de contorno de Dirichlet. O segundo problema é a dilatação de um elipsoide representando um ventrículo esquerdo com propriedades de material isotrópico. O problema simula, até certo ponto, a pressão exercida pelo sangue na superfície interna da cavidade ventricular. O terceiro problema descreve a pressão exercida pelo sangue e a contração ativa do ventrículo esquerdo. O problema testa a reprodutibilidade de padrões de fibras complexos e a implementação da contração ativa (Land et al., 2017).

2.4 Modelos matemáticos

O tecido cardíaco também deve obedecer as leis da mecânica clássica quando sujeito à alguma força. Partindo da generalização da segunda lei de Newton pode-se

derivar a equação de equilíbrio de forças a partir do princípio de conservação de momento linear. Desconsiderando os efeitos inerciais, as equações governantes usadas para descrever a deformação do tecido cardíaco se resumem ao problema de encontrar o campo de deslocamentos u, tal que

$$\begin{cases} \nabla \cdot (FS) + B = 0, em \Omega_0 \\ u = \bar{u}, em \partial\Omega_0^D \\ FSN = \overline{T}, em \partial\Omega_0^N \end{cases} . \quad (1)$$

onde u são os deslocamentos prescritos no contorno $\partial\Omega^D_0$ e T é a tensão aplicada no contorno $\partial\Omega^N_0$, que tem vetor normal N. F representa o tensor gradiente de deformação, S é o segundo tensor de Piola-Kirchhoff, e B são as forças de corpo.

2.5 Modelos constitutivo do tecido cardíaco

O uso de equações constitutivas é necessário para relacionar tensão e deformação. Equações constitutivas na mecânica do contínuo são modelos matemáticos desenvolvidos para descrever o comportamento físico observado dos materiais. Os modelos constitutivos podem descrever o relacionamento entre tensão e deformação diretamente através de uma equação ou por uma função energia de deformação. A função energia de deformação é uma função de Helmholtz Ψ que descreve a energia armazenada no material devido à uma deformação e é definida por unidade de volume de referência.

O tecido cardíaco é composto por fibras, que estão organizadas em camadas denominadas folhas, o que resulta em rigidez anisotrópica. Na direção da fibra, por exemplo, a deformação do tecido é maior do que nas outras direções, então modelos constitutivos tentam descrever esta deformação anisotrópica. O modelo constitutivo para o tecido cardíaco proposto por Guccione, (1995) foi utilizado neste trabalho. Este modelo é transversalmente isotrópico, ou seja, considera que a deformação do tecido é diferente em três direções ortogonais: paralela às fibras (f); perpendicular às fibras, mas paralela às folhas (s); e perpendicular às folhas (n). A função energia de deformação deste modelo é definida como

$$\Psi = \frac{c}{2}\left(e^{Q} - 1\right), \quad (2)$$

onde

$$Q = b_f E_{11}^2 + b_t\left(E_{22}^2 + E_{33}^2 + E_{23}^2 + E_{32}^2\right) + B_{fs}\left(E_{12}^2 + E_{21}^2 + E_{13}^2 + E_{31}^2\right). \quad (3)$$

Os parâmetros c, b_f , b_t e b_{fs} estão relacionados com o material e E_{ij} são as componentes do tensor de deformação de Green-Lagrange.

Quando se considera a contração do tecido cardíaco, uma abordagem consiste em dividir o segundo tensor de Piola-Kirchhoff em suas componentes ativa e passiva

$$S = S_p + S_a, \quad (4)$$

onde a parte passiva S_p descreve o comportamento passivo do coração e é derivado do modelo constitutivo, enquanto a parte ativa S_a descreve a cinética da contração celular e, em geral, é definida como

$$S_a = T_a f \otimes f \quad , \tag{5}$$

sendo T_a um valor escalar representando a tensão ativa aplicada na direção da fibra f.

2.6 Métodos numéricos

As equações governantes foram resolvidas usando o método dos elementos finitos, através de uma formulação variacional mista de três campos. O domínio foi dividido em hexaedros com o campo de deslocamentos sendo aproximado por funções lineares por partes, enquanto os demais campos são aproximados por funções constantes, que é a aproximação denominada Q1 − Q0 − Q0. Esta discretização resulta em um sistema não linear, o qual foi resolvido usando o método de Newton. Em cada iteração do método de Newton um sistema linear precisa ser resolvido, então o método iterativo GMRES foi utilizado junto a um precondicionador multi-grid algébrico em blocos. Mais detalhes sobre a solução das equações governantes podem ser encontrados em Campos (2018).

3 EXPERIMENTOS COMPUTACIONAIS

3.1 Detalhes da implementação

Os testes unitários implementados foram embutidos no código do simulador da atividade eletromecânica cardíaca baseado no método dos elementos finitos chamado Cardiax (Campos et al., 2018). O simulador é implementado com a linguagem C++ e utiliza bibliotecas de álgebra linear como o PETSc (Balay et al., 2017) e o Armadillo (Sanderson, 2010), e os testes foram implementados com o uso do framework de testes unitários CppTest.

O código do Cardiax se aproveita das ferramentas de orientação a objetos da linguagem C++ e implementa classes organizadas logicamente em módulos que abstraem as diversas componentes do simulador, como mostra a Figura 3.

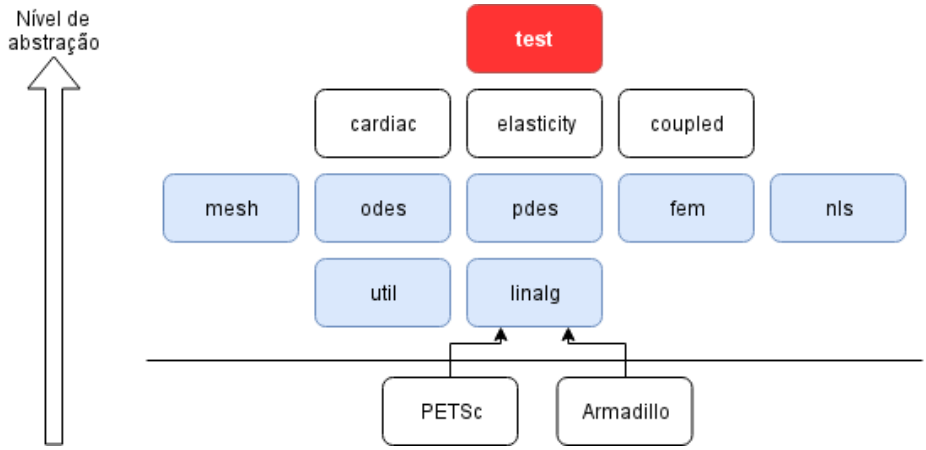


Figura 3. Módulos do simulador Cardiax. O módulo em vermelho representa o módulo de teste implementado. Os módulos representados em azul são os módulos testados.

Os testes unitários foram organizados em um novo módulo desacoplado do resto do código, garantindo que o funcionamento do simulador se mantenha independente dele. Neste módulo, através das ferramentas do framework, foram implementadas classes de teste para cada módulo testado, cada uma com suas rotinas de teste. A Tabela 1 apresenta uma breve descrição de cada módulo testado e explicita os testes realizados em cada um. Além disto, recursos e objetos de teste também foram agrupados dentro do módulo de teste. Uma classe geral agrupa as classes de teste de cada módulo e permite a organização das rotinas de teste, possibilitando por exemplo, a execução dos testes de apenas um módulo do software. Desta forma, a cada modificação no código do simulador, o módulo de teste pode ser acionado e conduzir a execução dos testes, resultando no fim em um relatório que sumariza os detalhes de cada teste, apontando as falhas e os sucessos de cada teste unitário. A Tabela 1 descreve de forma sucinta os módulos do Cardiax e destaca o módulo *elasticity,* o qual foi o principal objeto de estudo do presente trabalho.

Tabela 1. Módulos do simulador e os testes unitários realizados.

Módulos	Descrição	Testes
util	Abstrai os parâmetros passados para algoritmos numéricos.	Testes das operações dos objetos de parâmetros, como atribuição, busca e aninhamento.
linalg	Implementa operações de álgebra linear computacional.	Testes das operações básicas de álgebra linear como multiplicações matriz e vetor, produtos escalares e convergência de algoritmos numéricos.
mesh	Abstrai malhas computacionais e implementa operações de entrada e saída de malhas.	Teste na criação e operações dos objetos que abstraem malhas e testes de entrada e saída dos formatos suportados.
odes	Implementa métodos implícitos e explícitos para a solução numérica de modelos celulares	Testa os objetos que representam modelos celulares e suas operações, e os

	representados através de sistemas de equações diferenciais ordinárias.	resolvedores dos modelos baseados em equações diferenciais ordinária.
pdes	Resolve modelos de equações diferenciais parciais como o modelo de Poisson e os modelos do problema da elasticidade não linear e equações da atividade mecânica do coração.	Testa a ordem de convergência da resolução do modelo de Poisson e a adequação da resolução dos problemas de benchmark cardíaco com os resultados propostos por Land et al., 2017.
fem	Implementa as operações para a solução de modelos de equações diferenciais parciais utilizando o método dos elementos finitos.	Testa a convergência algoritmo da quadratura de Gauss para a integração numérica.
nls	Implementa algoritmos para a resolução de sistemas não lineares.	Testa a convergência dos algoritmos para a resolução de sistemas não lineares.

A verificação do código para os problemas de elasticidade não-linear e atividade mecânica cardíaca por meio dos problemas benchmark foi embutida nas rotinas de teste unitários. Para o problema 1 foram avaliadas malhas de 50 x 5 x 5 e 100 x 10 x 10 elementos. Para os problemas 2 e 3 foram avaliadas malhas de 12 x 27 x 2 e 24 x 54 x 6 elementos.

3.2 Resultados

Com a implementação, através do uso do framework CppTest, dos testes unitários automatizados para o software Cardiax, tornou-se possível a avaliação do funcionamento código como um todo.

Summary

Tests	Errors	Success	Time (s)
5	0	100%	5127.174383

Test suites

Name	Tests	Errors	Success	Time (s)
PoissonTestSuite	2	0	100%	2.249565
NonlinearElasticityTestSuite	3	0	100%	5124.924818

Suite: PoissonTestSuite

Name	Errors	Success	Time (s)
tri_convergence_test	0	true	1.211556
quad_convergence_test	0	true	1.038009

Back to top

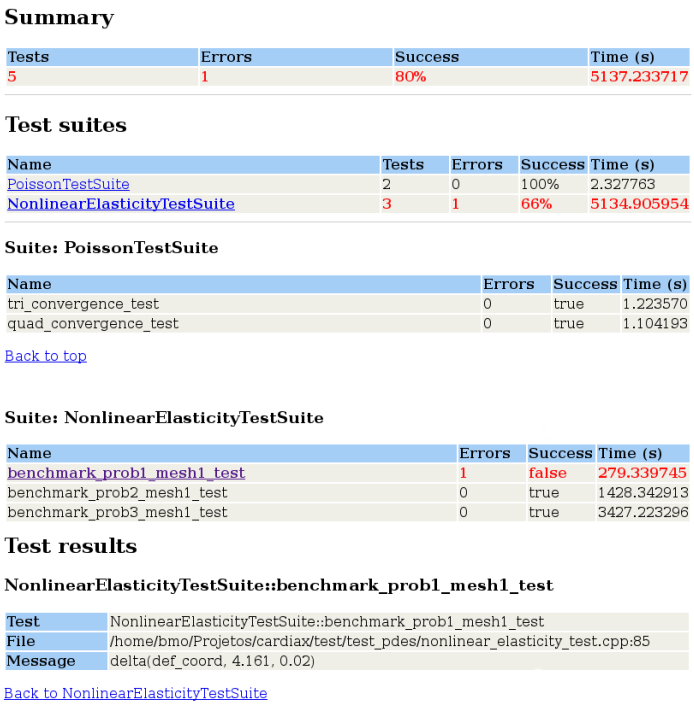
Suite: NonlinearElasticityTestSuite

Name	Errors	Success	Time (s)
benchmark_prob1_mesh1_test	0	true	276.755179
benchmark_prob2_mesh1_test	0	true	1424.641398
benchmark_prob3_mesh1_test	0	true	3423.528241

Figura 4. Relatório para o teste do módulo "pdes" do simulador, envolvendo o teste do problema de Poisson e dos problemas benchmark para as malhas menos refinadas. Saída em formato HTML do framework de testes CppTest. Neste caso todos os testes passaram.

O uso do framework CppTest permitiu a execução de baterias de testes para todo o código ou para um módulo em específico. Com os sumários dos testes em formato HTML a avaliação dos resultados e localização de falhas se dá de maneira simples e

objetiva, proporcionando um processo de testagem eficiente. As Figuras 4 e 5 ilustram exemplos de sumários para os testes de alguns módulos do simulador.



Summary

Tests	Errors	Success	Time (s)
5	1	80%	5137.233717

Test suites

Name	Tests	Errors	Success	Time (s)
PoissonTestSuite	2	0	100%	2.327763
NonlinearElasticityTestSuite	3	1	66%	5134.905954

Suite: PoissonTestSuite

Name	Errors	Success	Time (s)
tri_convergence_test	0	true	1.223570
quad_convergence_test	0	true	1.104193

Back to top

Suite: NonlinearElasticityTestSuite

Name	Errors	Success	Time (s)
benchmark_prob1_mesh1_test	1	false	279.339745
benchmark_prob2_mesh1_test	0	true	1428.342913
benchmark_prob3_mesh1_test	0	true	3427.223296

Test results

NonlinearElasticityTestSuite::benchmark_prob1_mesh1_test

Test	NonlinearElasticityTestSuite::benchmark_prob1_mesh1_test
File	/home/bmo/Projetos/cardiax/test/test_pdes/nonlinear_elasticity_test.cpp:85
Message	delta(def_coord, 4.161, 0.02)

Back to NonlinearElasticityTestSuite

Figura 5. Relatório para o teste do módulo "pdes" do simulador, envolvendo o teste do problema de Poisson e dos problemas benchmark para as malhas menos refinadas. Saída em formato HTML do framework de testes CppTest. Neste caso o teste para o benchmark do problema 1 falhou devido à tolerância propositalmente muito baixa.

Nos problemas de benchmark, buscou-se avaliar a solução do simulador por meio da comparação do resultado com os resultados de referência, permitindo-se uma tolerância. Para o problema 1 comparou-se a deflexão de um ponto no fim da barra avaliada na coordenada do eixo z. Para os problemas 2 e 3 comparou-se deformação de dois pontos na região apical do ventrículo esquerdo avaliada na coordenada z. As coordenadas de referência para os problemas são dadas pela solução média e são descritas em Campos et al., 2018.

Tabela 2. Coordenadas de referência e as tolerâncias escolhidas para os problemas benchmark.

Problema	Coordenada de referência		Tolerância	
Problema 1	4,161		0,05	
Problema 2	-28,196	-26,482	0,5	0,5
Problema 3	-15,405	-12,098	0,5	0,5

Para os testes foram designadas tolerâncias para a diferença absoluta entre as coordenas de referência e as coordenadas da solução. As coordenadas de referência e as tolerâncias escolhidas são mostradas na Tabela 2, todos os testes passaram para todas as malhas testadas. A Figura 6 mostra um exemplo de código para o teste unitário desenvolvido para a verificação do problema 1.

```
void NonlinearElasticityTestSuite::benchmark_prob1_mesh1_test() {

    const std::string meshname = "prob1_50";
    double coords[3] = {10.0, 1.0, 1.0};
    double def_coord;

    setup(meshname);
    def_coord = run_script(meshname, 10, 3, coords);

    TEST_ASSERT_DELTA(def_coord, 4.161, 0.05) // 4.19405

    tear_down(meshname);
}
```

Figura 6. Função principal para o teste unitário que verifica a adequação da solução do software à solução de referência do problema *benchmark* 1. A chamada a *setup* configura os recursos para o teste (lê a malha de entrada de roda a simulação), o método *run_script* roda um *script* que encontra e retona a coordenda da malha resultante, o macro *TEST_ASSERT_DELTA* verifica a adequação da solução em relação à solução de referência e o método *tear_down* libera os recursos (destrói objetos, fecha arquivos de malhas etc.).

Embora os valores de tolerância apresentados na Tabela 2 sejam relativamente altos para o problema benchmark, ressalta-se que os mesmos foram escolhidos para os testes unitários pois foram realizados com malhas contendo poucos elementos, uma vez que os problemas com malhas refinadas não foram incluídos nos testes devido ao alto custo computacional da solução

4 CONCLUSÃO

Foram aplicadas técnicas de teste unitário ao simulador da atividade mecânica do coração baseado no método dos elementos finitos. O framework CppTest para testes unitários em C++ foi utilizado para a implementação de testes unitários automatizados e para a verificação dos resultados dos modelos de elasticidade não-linear do coração através da execução de problemas benchmark.

Com testes envolvendo o escopo de cada módulo do simulador, a detecção de falhas se tornou mais fácil, sendo possível a localização de erros e discordâncias diretamente no código. Com isto, obteve-se uma maior confiabilidade nos resultados finais do programa, uma maior facilidade na depuração do código, uma vez que os

testes são capazes de localizar os defeitos introduzidos por uma eventual modificação no código, facilitando ainda desta forma a manutenção e extensibilidade do software.

AGRADECIMENTOS

Esse trabalho foi apoiado pela CAPES, CNPq, UFJF, CEFET-MG e FAPEMIG APQ-02537-15.

REFERÊNCIAS

Balay, S., Abhyankar, S., Adams, M., Brown, J., Brune, P., Buschelman, K., ... & Knepley, M. (2017). *Petsc users manual revision 3.8* (No. ANL-95/11 Rev 3.8). Argonne National Lab.(ANL), Argonne, IL (United States).

Campos, J. O., dos Santos, R. W., Sundnes, J., & Rocha, B. M. (2018). Preconditioned augmented Lagrangian formulation for nearly incompressible cardiac mechanics. *International journal for numerical methods in biomedical engineering*, *34*(4), e2948

Hatton, L., & Roberts, A. (1994). How accurate is scientific software?. *IEEE Transactions on Software Engineering*, *20*(10), 785-797.

Hughes, T. J. R. (2000). The Finite Element Method: Linear Static and Dynamic Finite Element Analysis. *Dover Civil and Mechanical Engineering*, 1st Edition.

Kane, D. W., Hohman, M. M., Cerami, E. G., McCormick, M. W., Kuhlmman, K. F., & Byrd, J. A. (2006). Agile methods in biomedical software development: a multi-site experience report. *Bmc Bioinformatics*, *7*(1), 273.

Kanewala, U., & Bieman, J. M. (2014). Testing scientific software: A systematic literature review. *Information and software technology*, *56*(10), 1219-1232.

Kelly, D., Hook, D., & Sanders, R. (2009). Five recommended practices for computational scientists who write software. *Computing in Science & Engineering*, *11*(5), 48-53.

Kerckhoffs, R. C., Healy, S. N., Usyk, T. P., & McCULLOCH, A. D. (2006). Computational methods for cardiac electromechanics. *Proceedings of the IEEE*, *94*(4), 769-783.

Koteska, B., Pejov, L., & Mishev, A. (2015). Scientific Software Testing: A Practical Example. In *SQAMIA* (pp. 27-34).

Land, S., Gurev, V., Arens, S., Augustin, C. M., Baron, L., Blake, R., ... & Fastl, T. E. (2015). Verification of cardiac mechanics software: benchmark problems and solutions for testing active and passive material behaviour. *Proc. R. Soc. A*, *471*(2184), 20150641.

Lundell, N. (2017). CppTest. Adquirido Agosto 22, 2018, de http://sourceforge.net/projects/cpptest/

National Research Council. (2012). *Assessing the reliability of complex models: mathematical and statistical foundations of verification, validation, and uncertainty quantification*. National Academies Press.

Niederer, S. A., Kerfoot, E., Benson, A. P., Bernabeu, M. O., Bernus, O., Bradley, C., ... & Heidenreich, E. (2011). Verification of cardiac tissue electrophysiology simulators using an N-version benchmark. *Phil. Trans. R. Soc. A*, *369*(1954), 4331-4351.

Pitt-Francis, J., Bernabeu, M. O., Cooper, J., Garny, A., Momtahan, L., Osborne, J., ... & Gavaghan, D. J. (2008). Chaste: using agile programming techniques to develop computational biology software. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, *366*(1878), 3111-3136.

Sanderson, C. (2010). Armadillo: An open source C++ linear algebra library for fast prototyping and computationally intensive experiments.

Vigmond, E. J., Dos Santos, R. W., Prassl, A. J., Deo, M., & Plank, G. (2008). Solvers for the cardiac bidomain equations. *Progress in biophysics and molecular biology*, *96*(1-3), 3-18.