PROPAGAÇÃO DE INCERTEZAS NO ENSINO E PESQUISA EXPERIMENTAL: UMA ABORDAGEM PRÁTICA COM A LINGUAGEM DE PROGRAMAÇÃO JULIA

DOI: 10.37702/2175-957X.COBENGE.2023.4513

João Marcello Pereira - joao.pereira@ifc.edu.br
IFC

Catia Brinckmann - catia.brinckmann@ifc.edu.br
Instituto Federal Catarinense

Eduardo Augusto Flesch - eduardo.flesch@ifc.edu.br
Instituto Federal Catarinense

Gustavo Tronchoni - gtronchoni@gmail.com
IFC

Ícaro Ilo da Silva - icaro.silva@ifc.edu.br
Instituto Federal Catarinense

Resumo: *Na realização de um experimento, sempre estaremos sujeitos a incertezas devido às limitações dos instrumentos, habilidade do experimentador e complexidade do fenômeno. Para calcular medidas indiretas a partir de uma função, é utilizada a técnica de propagação de incertezas, que avalia a incerteza do cálculo considerando as incertezas das variáveis de entrada. No entanto, esse cálculo não é simples e requer o uso de derivadas parciais de uma função, o que pode levar o experimentador a cometer erros quando não são utilizados recursos computacionais adequados. Apesar da existência de softwares e pacotes específicos para a propagação de incertezas em várias linguagens de programação, existem alguns inconvenientes, como a necessidade de compra de licenças ou a dependência de softwares proprietários. Neste contexto, a linguagem de programação Julia em conjunto com o pacote "Measurements.jl", configura-se como alternativa livre para realizar cálculos de propagação de incertezas de forma prática e intuitiva, além de integrável a outros pacotes da própria linguagem e até a outros pacotes e recursos de outras linguagens de programação.*

Palavras-chave: *linguagem julia, medidas, propagação de incertezas,*

experimentos

PROPAGAÇÃO DE INCERTEZAS NO ENSINO E PESQUISA EXPERIMENTAL: UMA ABORDAGEM PRÁTICA COM A LINGUAGEM DE PROGRAMAÇÃO JULIA

1 INTRODUÇÃO

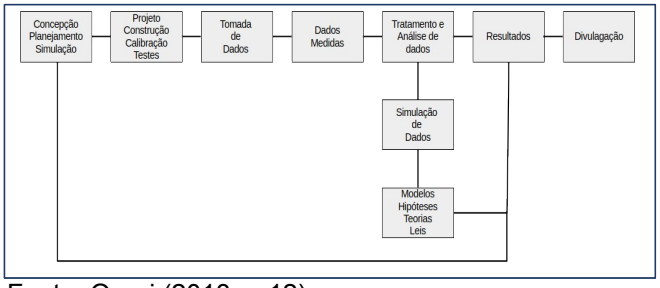
Na realização de experimentos, sempre há uma margem de incerteza associado às medidas realizadas, em consequência das limitações do instrumento de medição, a falta de habilidade do experimentador ou até mesmo a complexidade do fenômeno em análise. Ainda que sejam utilizados equipamentos de alta qualidade e técnicas experimentais cuidadosas, é impossível obter medidas livres de incertezas. Dessa forma, quando é necessário calcular medidas indiretas de uma grandeza a partir de uma função, é utilizado a técnica de propagação de incertezas, no qual a incerteza do cálculo de uma função é avaliado a partir das variáveis de entrada e suas respectivas incertezas. Embora seja aparentemente elementar, este cálculo não é simples de ser realizado manualmente ou através de calculadoras ou planilhas eletrônicas, pois requer o uso de derivadas parciais de uma função matemática. Em razão disso, o experimentador pode ser induzido a cometer erros comprometendo a qualidade dos resultados do experimento. Ainda que existam softwares, aplicativos para celular, aplicativos web e pacotes específicos de propagação de incertezas para diversas linguagens de programação, há alguns inconvenientes, como a necessidade de comprar licenças (software Engineering Equation Solver), a dependência de softwares proprietários quando os pacotes são gratuitos (“Metas unclib” para o MATLAB) e a falta de integração com outros softwares. Neste contexto, a linguagem de programação Julia em conjunto com o pacote “Measurements.jl”, configuram como alternativa livre para realizar cálculos de propagação de incertezas de forma prática e intuitiva, além de integrável a outros pacotes da própria linguagem e até a outros pacotes e recursos de outras linguagens de programação. Neste trabalho, será feito uma contextualização do ensino experimental e aplicações da linguagem de programação Julia no cálculo de propagação de incertezas.

2 ENSINO E PRÁTICA EXPERIMENTAL

Aulas de conteúdo experimental são importantes no ensino de ciências naturais em geral, tendo em vista que aguça a curiosidade, estimula a criatividade e desenvolve o pensamento crítico dos discentes, extrapolando o aprendizado além das fórmulas e textos. Isto não é uma preocupação recente, “foi percebida há aproximadamente 300 anos por John Locke, sendo que no final do século XIX, aulas experimentais já faziam parte do currículo de ciências na Inglaterra e nos Estados Unidos” (BARBERÁ;VALDÉS,1996, p.365 apud PARREIRA;DICKMAN, p.1). Em relação ao ensino superior, a prática experimental é fundamental para a formação de engenheiros, professores e pesquisadores, pois permite que os futuros profissionais desenvolvam habilidades manuais e computacionais, além do pensamento crítico e criativo na resolução de problemas, com o objetivo de compreender o mundo físico, para estimular o desenvolvimento de soluções inovadoras. Neste contexto,

a prática experimental implica na realização de medições, análise de dados, interpretação de resultados e avaliação da qualidade de modelos e teorias científicas, ou seja, todo experimento segue um conjunto de etapas baseado no método científico “Figura 1”. Além disso, as técnicas desenvolvidas serão essenciais na pesquisa científica experimental, tendo em vista que cabe ao pesquisador obter os melhores resultados de um experimento. Em resumo, o ensino experimental e as suas técnicas envolvidas, são essenciais para a formação profissional, contribuindo para a consolidação de conceitos teóricos, desenvolvimento de habilidades práticas e tomada de decisões fundamentadas, tendo em vista que segundo Ausubel(1963) a aprendizagem é significativa por recepção ou descoberta.

Figura 1 – Diagrama das etapas de um experimento genérico em física



Fonte: Oguri (2013, p.12).

3 A PROPAGAÇÃO DE INCERTEZAS

Todo experimento, independentemente da área de conhecimento científico, envolve a medição de uma ou *n* grandezas e sempre há uma margem de incerteza associado às medidas realizadas, em função das limitações do instrumento de medição, a falta de habilidade do experimentador ou até mesmo a complexidade do fenômeno em análise. De acordo com Oguri (2013, p.11) :

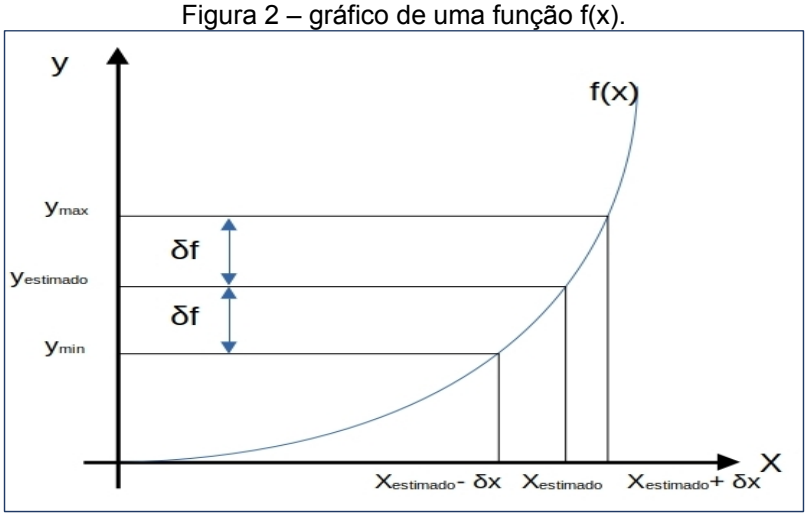
> Mesmo que as medições tenham sido realizadas com todo esmero, os valores encontrados (medidas) encontrados estão sujeitos inevitavelmente a incertezas, e somente a partir de uma criteriosa análise de erros – que permita estimá-las e, em muitos casos, reduzi-las ou controlá-las – torna-se possível quantificar apropriadamente um resultado experimental

A incerteza pode ser definida como “parâmetro não negativo que caracteriza a dispersão dos valores atribuídos a uma mensurando, com base nas informações utilizadas” (INMETRO, 2012, p. 24), ou seja, a incerteza pode ser entendida como um parâmetro de dúvida intrínseco ao processo de medição. Tendo em vista que em um experimento são realizadas medições diretas de *n* grandezas com respectivas incertezas, a manipulação numérica dessas medidas implica em uma propagação da incerteza no resultado final. Dessa forma, Oguri (2013, p.63) esclarece que o método de propagação de erros é o problema da estimativa do valor esperado e sua incerteza. Esse método é fundamental no ensino e prática experimental, pois permite avaliar a precisão do resultado numérico de um experimento ou de um sistema de medição, bem como avaliar a incerteza associada a uma medida em particular. Dessa forma, a qualidade dos resultados experimentais está diretamente ligada aos procedimentos metrológicos e matemáticos envolvendo medidas e grandezas com incertezas.

3.1 Regra geral da propagação de incertezas

Considere o gráfico abaixo ("Figura 2") da função $f(x)$, sendo $x_{estimado}$ o valor medido de uma grandeza, $y_{estimado}$ o valor calculado de de $f(x_{estimado})$ e δx e δy suas respectivas incertezas associadas.

Figura 2 – gráfico de uma função f(x).



Fonte: Autores.

Conforme "Figura 2", os valores máximos e mínimos de x e y são respectivamente, $x \pm \delta x$ e $y \pm \delta y$. Se assumirmos que a incerteza δx é de valor suficiente pequeno para considerar a seção do gráfico em consideração como aproximadamente uma reta entre os pontos com y_{max}, y_{min} , a incerteza δy pode ser estimada como:

$$\delta y = (x_{estimado} + \delta x) = f(x_{estimado}) = y_{estimado} \quad (1)$$

Agora, uma aproximação fundamental do cálculo informa que, para qualquer função $f(x)$ com incremento suficientemente pequeno v temos:

$$f(x + v) - f(x) = \frac{df}{dx} \cdot v \quad (2)$$

Assumindo que a incerteza x é suficientemente pequena e igualando "Equação (1)" e "Equação (2)", resulta:

$$\delta f = \frac{df}{dx} \delta x \quad (3)$$

Considerando que temos uma função de 3 (três) variáveis $f(x,y,z)$ com respectivas incertezas δx, δy e δz , podemos assumir que todas as incertezas medidas são suficientemente pequenas, então δy também pode ser encontrado usando a aproximação da série de Taylor, ou seja:

$$\delta f^2 = \left[\left(\frac{\partial f}{\partial x}\delta x\right)^2 + \left(\frac{\partial f}{\partial y}\delta y\right)^2 + \left(\frac{\partial f}{\partial x}\delta z\right)^2\right] + \left[\frac{\partial f}{\partial x}\cdot\frac{\partial f}{\partial y}\delta x\delta y + \frac{\partial f}{\partial y}\cdot\frac{\partial f}{\partial z}\delta y\delta z + \frac{\partial f}{\partial z}\cdot\frac{\partial f}{\partial x}\delta z\delta x\right] \quad (4)$$

Os termos quadrados são sempre positivos e nunca se cancelam. No entanto, de acordo com Shaheen e Anwar (2013, p.20) os termos restantes podem se anular quando as variáveis de entrada são independentes, logo, todas as covariâncias serão nulas. A regra geral de propagação de incertezas é baseada no conceito de diferenciais e utiliza a derivada parcial para determinar a incerteza de uma quantidade calculada a partir de *n* grandezas com incertezas conhecidas. Em termos simplificado, temos:

$$\delta f(x,y,z) = \sqrt[2]{\left(\frac{\partial f}{\partial x}\,\delta x\right)^2 + \left(\frac{\partial f}{\partial y}\,\delta y\right)^2 + \left(\frac{\partial f}{\partial z}\,\delta z\right)^2} \qquad (5)$$

De acordo com a “Equação (5)”, podemos observar uma relação direta entre as múltiplas variáveis de entrada e suas incertezas.

4 A LINGUAGEM DE PROGRAMAÇÃO JULIA

Julia foi criada em 2009 pelos pesquisadores Stefan Karpinski, Jeff Bezanson, Alan Edelman e Viral Shah e em 2012 foi lançada sob licença MIT. “Julia é uma linguagem de programação compilada (JIT – Just in time) livre de alto nível projetado com foco na computação científica e numérica de alto desempenho” (BENZANSON et al., 2015). Em relação ao campo de aplicação, Pereira e Siqueira (2016) esclarecem que:

> Foi pensada como uma linguagem para computação científica suficientemente rápida como C ou Fortran, mas igualmente fácil de aprender como o MATLAB e o Mathematica, com o objetivo de facilitar a modelagem computacional.

Julia é uma linguagem de código aberto e livre, que segundo Da Silveira (2004), está de acordo com as seguintes diretrizes de liberdades:

- Liberdade 0: A liberdade de executar o programa;
- Liberdade 1: A liberdade de estudar e adaptar o programa para suas necessidades;
- Liberdade 2: A liberdade de redistribuir cópias, para ajudar o próximo;
- Liberdade 3: A liberdade de aperfeiçoar e liberar o programa para que toda a comunidade seja beneficiada.

Em relação ao desempenho, no sítio oficial da linguagem Julia (www.julialang.org/benchmarks/) existe a publicação de um benchmark entre linguagens de programação (C, Fortran, Go, Java, JavaScript, Julia, LuaJIT, Mathematica, Matlab, Python, R, Rust e Octave) no qual os resultados foram compilados em um gráfico (omitido neste trabalho em função do espaço) de desempenho relativo à linguagem C. De acordo com sítio oficial, o benchmark visa testar o desempenho dos compiladores em uma variedade de código comuns excluindo o tempo de compilação, no qual foi utilizado um computador com a seguinte configuração: CPU Intel® Core™ i7-3960X 3,30 GHz com 64 GB de RAM DDR3 de 1600 MHz, executando openSUSE LEAP 15.0 Linux. Os algorítimos testados foram: iteration_pi_sum, matrix_multiply, matrix_statistics, parse_integers, print_to_file, recursion_fibonacci, recursion_quicksort, userfunc_mandelbrot. Conforme o gráfico apresentado no sítio oficial da linguagem, Julia se destaca em relação ao seus concorrentes diretos (Matlab, Fortran, Python, Mathematica e R) devido sua compilação via LLVM. Principais características da linguagem de programação Julia:

- Linguagem de programação moderna, expressiva e de alto desempenho, projetada para computação científica e manipulação de dados;
- Sintaxe de codificação semelhante à sintaxe do MATLAB/GNU-Octave;
- Projetado para computação distribuída e paralela;
- Extensa biblioteca de funções matemáticas com precisão numérica arbitrária;
- Multiple dispatch (despacho múltiplo);
- Recurso de chamada de funções escritas em C, Fortran e Python;
- Recurso de Shell para gerenciar outros processos no sistema;
- Tipos definidos pelo usuário são tão rápidos e compactos como os tipos da própria linguagem;
- Não há necessidade de vetorizar códigos para obter melhor desempenho;
- Macros semelhantes à macros de Lisp e outras facilidades de metaprogramação;
- Possui suporte eficiente ao Unicode;
- Compilação JIT via LLVM;
- Possui um excelente gerenciador de pacotes embutido (Pkg);
- IDE de livre escolha.

Em relação às suas principais características, Julia destaca-se pela sua sintaxe similar à do Matlab/GNU-Octave conforme pode ser visto no “Quadro 1”.

Quadro 1 – Sintaxe dos comandos.

Matlab	Julia

%Declaração de variáveis
`var = 10.5;`
 	
%Vetores e Matrizes
`vet  = [1; 2; 3]`
`mat = [1 2 3; 4 5 6]`
 	
%funções
`function var = nome(arg)`
`  var = instrução`
`end`
 	
%condicional IF-ELSE
`if condição`
`  instruções`
`elseif condição`
`  instruções`
`else`
`  instruções`
`end`
 	
%Laço FOR
`for variável = vetor`
`  instruções`
`end`
 	
%Laço WHILE
`while condição`
`  instruções`
`end`
	#Declaração de variáveis
	`var = 10.5;`
	
	#Vetores e Matrizes
	`vet  = [1, 2, 3]`
	`mat = [1 2 3; 4 5 6]`
	
	#funções
	`function nome(arg)`
	`  return instrução`
	`end`
	
	#condicional IF-ELSE
	`if condição`
	`  instruções`
	`elseif condição`
	`  instruções`
	`else`
	`  instruções`
	`end`
	
	#Laço FOR
	`for variável = vetor`
	`  instruções`
	`end`
	
	#Laço WHILE
	`while condição`
	`  instruções`
	`end`

Fonte : Autores.

Atualmente a linguagem Julia está disponível na sua versão estável 1.9.0 (07/05/2023), contendo melhorias, correções de bugs e atualizações do kernel. Em relação a sua popularidade, ela ocupa a 30º posição (maio/2023) no ranking “TIOBE Index”.

4.1 Pacote Measurements.jl

É definido um pacote no contexto de linguagens de programação, como sendo um conjunto de módulos que contém conjuntos de funções que realizam tarefas específicas do módulo. “Measurements.jl” é um pacote totalmente escrito em linguagem Julia que permite calcular a propagação de incertezas provenientes de medições diretas e indiretas. De acordo com o seu criador, ele foi desenvolvido para tornar prático o trabalho com dados experimentais e incertezas associadas, permitindo que a propagação da incerteza seja precisa e confiável. Principais características do pacote:

- Suporte para precisão arbitrária;
- Definição de variável com incerteza a partir de uma string;
- Propagação de incertezas com variáveis complexas;
- Cálculo matricial com incertezas;
- Propagar a incerteza para qualquer função de argumentos reais (incluindo funções baseadas em chamadas de C ou Fortran), usando a macro @uncertain;
- Funções para obter a derivada e o gradiente de uma função;
- Permite utilizar o símbolo “±” para representar incerteza;
- Permite a integração com outros pacotes.

Instalação do pacote

O pacote “Measurements.jl” é instalado via linha de comando em uma sessão Julia através do gerenciador de pacotes Pkg conforme instruções contidas no “Quadro 2”. Uma vez instalado, o pacote fica disponível permanentemente no repositório local da linguagem Julia até ser removido manualmente.

Quadro 2 – Instalação do pacote “Measurements.jl”.

```
julia> using Pkg
julia> Pkg.add("Measurements")
    Updating registry at `~/.julia/registries/General.toml`
   Resolving package versions...
   Installed Preferences ──────── v1.4.0
   Installed Measurements ─────── v2.9.0
   Installed PrecompileTools ──── v1.1.1
    Updating `~/.julia/environments/v1.9/Project.toml`
  [eff96d63] + Measurements v2.9.0
    Updating `~/.julia/environments/v1.9/Manifest.toml`
  [49dc2e85] + Calculus v0.5.1
  [eff96d63] + Measurements v2.9.0
  [aea7be01] + PrecompileTools v1.1.1
  [21216c6a] + Preferences v1.4.0
  [8f399da3] + Libdl
  [37e2e46d] + LinearAlgebra
  [de0858da] + Printf
  [ea8e919c] + SHA v0.7.0
  [fa267f1f] + TOML v1.0.3
  [e66e0078] + CompilerSupportLibraries_jll v1.0.2+0
  [4536629a] + OpenBLAS_jll v0.3.21+4
Precompiling project...
  6 dependencies successfully precompiled in 4 seconds
```

Fonte : Autores

Sintaxe básica dos comandos

O pacote "Measurements.jl" possui uma sintaxe flexível e amigável para definir uma variável com incertezas, resultando em várias possibilidades sintáticas intercambiáveis entre si. Os comandos a seguir foram digitados no próprio ambiente de execução da linguagem Julia, tendo em vista que o REPL (read-eval-print loop) é suficiente para testar todos os códigos apresentados neste trabalho. Uma vez importado, o pacote "Measurements.jl" fica disponível até encerrar a sessão em uso. Para ilustrar os comandos básicos foram selecionados 3 (três) exemplos a seguir.

- **Exemplo 1:** Definir uma variável "massa" de massa 2,00kg e incerteza de escala 0,01kg, e realizar operações algébricas básicas. Código e comentários "Quadro 3".

Quadro 3 – Código exemplo 1.

```
# importação do pacote
julia> using Measurements

# possibilidades sintáticas
julia> massa = measurement(2.00, 0.01)
2.0 ± 0.01

julia> massa = measurement("2.00 +/- 0.001")
2.0 ± 0.001

julia> massa = measurement("2.00 ± 0.001")
2.0 ± 0.001

julia> massa = measurement("2.00(0.01)")
2.0 ± 0.01

# operações algébricas
julia> massa - massa
0.0 ± 0.0

julia> massa + massa
0.04 ± 0.0002

julia> massa/massa
1.0 ± 0.0

julia> 2*massa
0.04 ± 0.0002

julia> massa*massa
0.0004 ± 4.0e-6

julia> massa^2
0.0004 ± 4.0e-6

julia> massa/4
0.005 ± 2.5e-5
```

Fonte: Autores

- **Exemplo 2:** Definir a variável complexa $Z = (32{,}7 \pm 1{,}1) - (3{,}1 \pm 0{,}2)i$ e realizar operações algébricas. Código e comentários "Quadro 4".

Quadro 4 – Código exemplo 2.

```
# importação do pacote
julia> using Measurements

# definição sintática
julia> Z = complex(32.7 ± 1.1, -3.1 ± 0.2)
(32.7 ± 1.1) - (3.1 ± 0.2)im

# operações algébricas
julia> Z + Z
(65.4 ± 2.2) - (6.2 ± 0.4)im

julia> Z - Z
(0.0 ± 0.0) + (0.0 ± 0.0)im

julia> 2*Z
(65.4 ± 2.2) - (6.2 ± 0.4)im

julia> Z/2
(16.35 ± 0.55) - (1.55 ± 0.1)im

julia> Z*Z
(1060.0 ± 72.0) - (203.0 ± 15.0)im

julia> cos(Z)
(3.1 ± 12.0) + (10.6 ± 4.1)im
```

Fonte: Autores

- **Exemplo 3:** Calcular o volume de um cilindro reto, sendo $r = (0{,}5000 \pm 0{,}0005)m$ e $h = (2{,}0000 \pm 0{,}0005)m$. Código e comentários no "Quadro 5".

Quadro 5 – Códigos exemplo 3.

```
# importação do pacote
julia> using Measurements

# definição das variáveis
julia> r = measurement(0.5000, 0.0005)
0.5 ± 0.0005

julia> h = measurement(2.0000, 0.0005)
2.0 ± 0.0005

# cálculo do volume em m³
julia> volumeC = pi*r^2*h
1.5708 ± 0.0032

# uma função volume pode ser definida para calcular o volume do cilindro
# de qualquer valor de r e h
julia> fvolumeC(r, h) = pi*r^2*h
fvolumeC (generic function with 1 method)

julia> fvolumeC(measurement(1.5000, 0.0005), measurement(2.5000, 0.0005))
17.671 ± 0.012
```

Fonte Autores

Integração com outros pacotes

O pacote "Measurements.jl" permite integração com diversos pacotes desenvolvidos para a linguagem de programação Julia, assim como a integração com pacotes e recursos de outras linguagens de programação, como Python, C, Fortran, R, Java entre outras. Isso permite diversificar as possibilidades de artifícios computacionais aplicáveis a diversos tipos de problemas. Para ilustrar a integração com outros pacotes, foram selecionados 2 (dois) exemplos a seguir:

- **Exemplo 4:** Calcular $\int_a^b \left(x^2 - 4\right) dx$, sendo $a = (5{,}00 \pm 0{,}05)$ e $b = (8{,}00 \pm 0{,}05)$. Código e comentários no "Quadro 6".

Quadro 6 – Códigos exemplo 4.

```
# importação do pacote para cálculo da integral
julia> using QuadGK

# definição da função f(x)
julia> f(x) = x^2 - 4
f (generic function with 1 method)

# definição das variáveis
julia> a = measurement(5.00, 0.05)
5.0 ± 0.05

julia> b = measurement(8.00, 0.05)
8.0 ± 0.05

# opções sintáticas para o cálculo da integral definida
julia> quadgk(f, x1, x2)
(117.0 ± 3.2, 2.842e-14 ± 5.0e-16)

julia> quadgk(f, 5.00 ± 0.05, 8.00 ± 0.05)
(117.0 ± 3.2, 2.842e-14 ± 5.0e-16)
```

Fonte: Autores

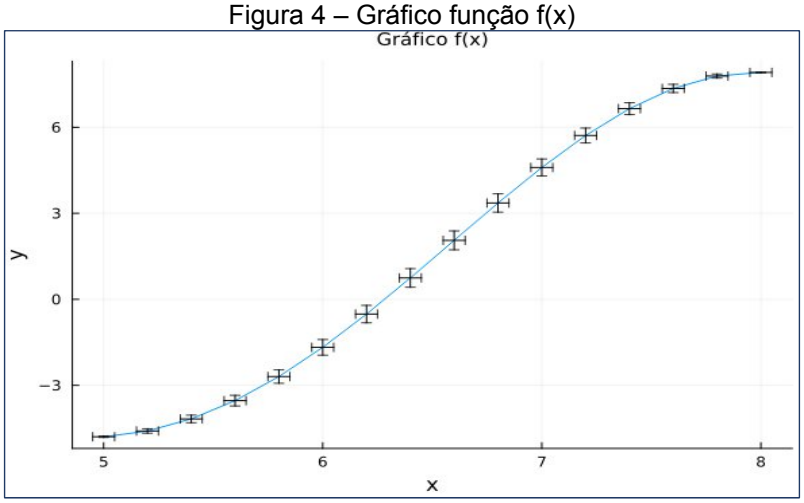
- **Exemplo 5:** Desenhar o gráfico da função $f(x) = x{\cdot}sin(x)$ entre x_1 e x_2, sendo $x_1 = (5{,}00 \pm 0{,}05)$ e $x_2 = (8{,}00 \pm 0{,}05)$. Observe que o gráfico gerado ("Figura 4") possui as barras de erros na vertical e horizontal. Código e comentários "Quadro 7".

Quadro 7 – Código exemplo 5.

```
# importação dos pacotes
julia> using Measurements, Plots

# definição da função f(x)
julia> f(x) = x*sin(x)
f (generic function with 1 method)

# definição das variáveis
julia> x1 = measurement(5.00, 0.05)
5.0 ± 0.05

julia> x2 = measurement(8.00, 0.05)
8.0 ± 0.05

# gráfico da função
julia> plot(f, [x ± 0.05 for x in 5.00:0.20:8.00], label = "", title =
"Gráfico f(x)", xlabel = "x", ylabel = "y")
```

Fonte: Autores

Figura 4 – Gráfico função f(x)



Fonte: Autores

Aplicação em física experimental

Feito as demonstrações das potencialidades do pacote "Measurements.jl", apresentaremos neste tópico uma aplicação simples, porém comum nas práticas de física experimental, no qual é determinado as funções horárias de um móvel através do método numérico de ajuste polinomial de curvas .

- **Exemplo 6**: Um carrinho de laboratório é posto para movimentar-se sobre um trilho inclinado "Figura 5". Os dados obtidos pelo experimento foram compilados na "Tabela 1". A partir dos dados coletados, determinar as funções horárias das posições e velocidades, X(t) e V(t). Código e comentários no "Quadro 8".

Figura 5 – Carrinho de laboratório descendo plano inclinado

Fonte: Autores.

Tabela 1 – Dados experimentais do deslocamento de um móvel

Tempo (s)	Posição (m)
0,0 ± 0,1	0,0000 ± 0,0005
3,2 ± 0,1	0,2000 ± 0,0005
4,5 ± 0,1	0,4000 ± 0,0005
5.8 ± 0,1	0,6000 ± 0,0005
6,8 ± 0,1	0,8000 ± 0,0005
7,6 ± 0,1	1,0000 ± 0,0005

Fonte: Autores.

Quadro 8 – Códigos exemplo 6.

```
# importação dos pacotes
julia> using Measurements, CurveFit, Polynomials

# definição das variáveis
julia> tempo = measurement.([0.0,3.2,4.5,5.8,6.8,7.6],[0.1,0.1,0.1,0.1,0.1,0.1])
6-element Vector{Measurement{Float64}}:
 0.0 ± 0.1
 3.2 ± 0.1
 4.5 ± 0.1
 5.8 ± 0.1
 6.8 ± 0.1
 7.6 ± 0.1

julia> posicoes =
measurement.([0,0.2,0.4,0.6,0.8,1,0],[0.0005,0.0005,0.0005,0.0005,0.0005,0.0005])
6-element Vector{Measurement{Float64}}:
  0.0 ± 0.0005
  0.2 ± 0.0005
  0.4 ± 0.0005
  0.6 ± 0.0005
  0.8 ± 0.0005
  1.0 ± 0.0005

# função X(t) obtida pelo ajuste de curva polinomial
julia> X = curve_fit(Polynomial, tempo, posicoes, 2)
Polynomial(-0.00093 ± 0.0031 + 0.0188 ± 0.0068*x + 0.0148 ± 0.0011*x^2)

# função V(t) obtida pela derivada da função X(t)
julia> V = Polynomials.derivative(X)
Polynomial(0.0188 ± 0.0068 + 0.0295 ± 0.0022*x)
```

Fonte: Autores

Conforme execução do código do "Quadro 8", a função horária $X(t)$ ("Equação 6") foi obtida por ajuste de curva polinomial de grau 2 (dois) e a função horária $V(t)$ foi obtida pela derivada simbólica de $X(t)$ ("Equação 7"). As funções horárias permitem a passagem de valores reais para o argumento "t".

$$X(t) = (-0{,}00093 \pm 0{,}0031) + (0{,}0188 \pm 0{,}0068)\cdot x + (0{,}0148 \pm 0{,}0011)\cdot x^2 \quad (6)$$

$$V(t) = (0{,}0188 \pm 0{,}0068) + (0{,}0295 \pm 0{,}0022)\cdot x \quad (7)$$

5 Considerações FINAIS

A partir das considerações apontadas no decorrer do texto, podemos conferir que o ensino experimental é fundamental na formação de discentes e profissionais pesquisadores, pois envolve a realização de atividades práticas que permitem vivenciar e explorar conceitos científicos de forma concreta. Tendo em vista a complexidade do cálculo de propagação de incertezas, a linguagem de programação Julia em conjunto com o pacote "Measurements.jl" configuram uma excelente alternativa livre e gratuita, que permite explorar o cálculo de propagação de incertezas de forma prática e integrável com outros pacotes e outras linguagens de programação.

AGRADECIMENTOS

Agradecemos ao Instituto Federal Catarinense pelo apoio e incentivo.

REFERÊNCIAS

AUSUBEL, David P.. **The Psychology of Meaningful Verbal Learning: An Introduction to School Learning**. New York: Grune & Stratton, 1963.

BEZANSON, J. et al. **Julia Language Documentation**. Disponível em: http://www.julialang.org. Acesso em: 01 de Mai. 2023.

INMETRO. **Vocabulário Internacional de Metrologia (VIM)**. Instituto português da qualidade,1ª Edição, Caparica, Portugal: 2012.

OGURI, V. (ORG.), **Estimativas e Erros em Experimentos de Física**. Editora UERJ, 2ª edição, 2013.

PARREIRA, Júlia E. ; DICKMAN, A. G. . **Objetivos das aulas experimentais no ensino superior na visão de professores e estudantes da engenharia**. Disponível em: https://www.scielo.br/j/rbef/a/xBPRrtNZKLT3fY8T3Wp9fCh/?format=pdf&lang=pt. Acesso em: 10 de Mai. 2023.

PEREIRA, J.M; SIQUEIRA, M.B.B. **Linguagem de programação Julia:uma alternativa open source e de alto desempenho ao Matlab**. Cobenge, 2016.

SHAHEEN, A., ANWAR, M. S. **Uncertainties and Measurements in Experimental Physics. LUMS School of Science and Engineering**. Disponível em: https://physlab.org/wp-content/uploads/2016/03/uncertainties_newmanual1n1.pdf. Acesso: 15 de Mai. 2023.

PROPAGATION OF UNCERTAINTIES IN TEACHING AND EXPERIMENTAL RESEARCH: A PRACTICAL APPROACH WITH THE JULIA PROGRAMMING LANGUAGE

Abstract: *When carrying out an experiment, we will always be subject to uncertainties due to the limitations of the instruments, the experimenter's skill and the complexity of the phenomenon. To calculate indirect measures from a function, the uncertainty propagation technique is used, which evaluates the uncertainty of the calculation considering the uncertainties of the input variables. However, this calculation is not simple and requires the use of partial derivatives of a function, which can lead the experimenter to make mistakes when adequate computational resources are not used. Despite the existence of specific software and packages for the propagation of uncertainties in several programming languages, there are some inconveniences, such as the need to purchase licenses or dependence on proprietary software. In this context, the Julia programming language together with the "Measurements.jl" package, is configured as a free alternative to carry out uncertainty propagation calculations in a practical and intuitive way, in addition to being able to integrate to other packages of the language itself and to other packages and features from other programming languages.*

Keywords: *julia language, measurements, uncertainty propagation, experiments*