miniEP1

March 24, 2020

1 MiniEP1 - Aproximação de Integrais usando Médias e Amostragem Uniforme

1.1 Entrega do miniEP1

Preencha o nome dos 5 membros do seu grupo na tabela abaixo:

Nome	NUSP
Membro1	12345789
Membro2	12345789
Membro3	12345789
Membro4	12345789
Membro5	12345789

Apenas um dos membros deverá entregar **este arquivo** `.ipynb`, com as soluções do miniEP1 **feitas pelo grupo**.

1.2 Configuração do Ambiente

A primeira tarefa do miniEP1 é instalar e configurar o ambiente.

1.2.1 Julia, Jupyter, IJulia

Para fazer o miniEP1, você vai precisar:

- Instalar o Jupyter Notebook
- Instalar Julia 1.3:
  - Baixando o binário
  - **ou** usando seu gerenciador de pacotes
- Instalar o pacote *IJulia*:
  - Inicie o interpretador Julia
  - Digite `] add IJulia` e pressione `<ENTER>`

Depois disso, você vai conseguir iniciar o arquivo `.ipynb` do miniEP1.

1.2.2 Pacotes Julia para o miniEP1

Os pacotes necessários para o miniEP1 estão listados no arquivo `Project.toml`, mas você pode instalar e atualizar os pacotes rodando a célula abaixo:

[ ]:
```
] up
```

Verifique o status dos pacotes, e se há algum problema, com o comando:

[ ]:
```
] st
```

1.3 Integração por Método de Monte Carlo

O objetivo deste miniEP é relembrar e aplicar alguns conceitos de estatística, independentes de linguagens de programação. Escolhemos usar a linguagem Julia para os experimentos, análises e geração de gráficos, mas a metodologia desenvolvida pode ser usada com qualquer ferramenta.

Você deve ler o texto neste arquivo e ir executando as células de código que encontrar. Assim vai definindo as funções progressivamente e vai gerar gráficos no final. Julia é uma linguagem compilada *Just-in-Time*, e a primeira vez que você executa uma célula pode levar algum tempo. Isso é por que os pacotes estão sendo compilados. A segunda execução da mesma função será mais rápida, verifique!

Vamos estimar o valor de algumas integrais usando o método de integração de Monte Carlo. A intuição por trás do método é que a integral de uma função f pode ser estimada pela média do valor de f num conjunto suficientemente grande de pontos obtidos a partir de uma distribuição uniforme.

Mais formalmente, para um conjunto de pontos $x_1, \ldots, x_N$ uniformemente amostrados num intervalo $[a, b]$, a integral de f no intervalo $[a, b]$ pode ser aproximada por:

$$\int_a^b f(x)dx \approx \mathbb{E}\left[(b-a)\,\frac{1}{N}\sum_{i=1}^{N} f(x_i)\right]$$

Para uma representação gráfica da intuição, e para a prova dessa aproximação, veja esta página.

O código abaixo define a função `monte_carlo_integrate`, que estima a integral de uma função `f` num intervalo `interval`, usando o método de Monte Carlo com amostras dadas pelo valor de `samples`. O código usa aplicação vetorizada de funções, através da sintaxe `função.(vetor)`, mas os comentários contêm código equivalente de uma versão usando loops. Usamos o pacote `Distributions` para gerar números pseudoaleatórios no intervalo $[a, b]$. Execute a célula abaixo:

[ ]:
```
using StatsPlots, Distributions, DataFrames, Query

function monte_carlo_integrate(f, interval, samples = 100)
    xs = rand(Uniform(interval[1], interval[2]), samples)

    # Using for loops:
    # accumulator = 0
    #
    # for x in xs
    #     accumulator += f(x)
    # end
    #
```

```
    # return accumulator / samples

    # Using vectorized function application:
    return sum(f.(xs)) / samples
end
```

1.3.1 Primeiro Exemplo

Usamos a função `make_f1` abaixo para encapsular o parâmetro a de uma função definida por:

$$f_1(x) = \frac{2}{\sqrt{a^2 - x^2}}$$

Se quiser saber mais, leia sobre funções anônimas em Julia.

Usando $a = 1$, definimos a função `f1`. A integral da função f_1, definida acima, é dada por:

$$\int_0^a f_1(x)dx = \int_0^a \frac{2}{\sqrt{a^2 - x^2}}dx = \pi$$

Vamos estimar a integral da função f_1 usando o método de Monte Carlo. Execute a célula abaixo:

[ ]:
```
function make_f1(a)
    return x -> 2 / (sqrt((a ^ 2) - (x ^ 2)))
end


a = 1
f1 = make_f1(a)
```

1.3.2 Executando Experimentos e Gerando Resultados

A função `run_experiments` recebe uma função alvo `target_f`, um intervalo `interval`, um número de amostras `samples`, e a quantidade de repetições da estimação `repetitions` que devem ser feitas. A função devolve uma estrutura do tipo `DataFrame`, que é essencialmente uma matriz com colunas nomeadas e de tipo fixo.

Abstrações como os `DataFrames` são extremamente úteis para representar dados tabelados, e são muito usadas em computação científica e em experimentos computacionais em geral. As linguagens Python e R também têm abstrações equivalentes.

Nossa função "cresce" um `DataFrame` linha a linha, à medida que executa os experimentos. É prática comum repetir parâmetros para os experimentos, como a coluna `func` do nosso `DataFrame`, pois assim podemos misturar experimentos diferentes numa mesma estrutura.

[ ]:
```
function run_experiments(target_f, interval, samples, repetitions)
    results = DataFrame(func = String[], samples = Int[], integral = Float64[])

    for s in samples
```

```
        for r in 1:repetitions
            append!(results, DataFrame(func = ["target_f"],
                    samples = [s],
                    integral = monte_carlo_integrate(target_f, interval, s)))
        end
    end

    return results
end

results = run_experiments(f1, [0, a], [10 ^ x for x in 0:5], 20)
```

1.3.3 Estimando a Integral e Manipulando os Resultados

Ao repetir a estimativa de uma integral para um número fixo de amostras, esperamos observar propriedades interessantes da variação do valor estimado. Neste miniEP, vamos nos preocupar com a *média* e o *desvio padrão* das amostras, e vamos calcular um *intervalo de confiança* de 95%.

A função `parse_results` recebe um `DataFrame` gerado pela função `run_experiments`, e gera um novo `DataFrame` com os valores da média e intervalo de confiança 95% das estimativas repetidas.

Utilizamos o operador *pipe* `|>` e pacote Query para fazer os cálculos. O operador `|>` é binário, e seu significado é "*passe o argumento da esquerda como primeiro parâmetro da função à direita*":

[ ]:
```
? |>
```

A macro `@groupby` agrupa as linhas de um `DataFrame` pela variável passada como parâmetro. O caractere `_` faz referência ao `DataFrame` passado pelo operador `|>`.

A macro `@map` recebe um dicionário de funções, onde a chave será uma coluna do novo `DataFrame`, e o valor é uma função envolvendo ou não as colunas originais.

A função de chave `mean_integral`, por exemplo, calcula a média dos valores da coluna `integral` original, agrupada por `samples`. A coluna `ci_estimate`, por sua vez, conterá uma estimativa do intervalo de confiança 95%.

O operador `|>` e o pacote Query permitem escrever código muito similar ao produzido pela biblioteca dplyr da linguagem R. Execute a célula abaixo:

[ ]:
```
function parse_results(results, target_integral)
    parsed_results = results |>
                     @groupby(_.samples) |>
                     @map({samples = key(_),
                           mean_integral = mean(_.integral),
                           ci_estimate = 1.96 * std(_.integral),
                           target = target_integral}) |>
                     DataFrame

    return parsed_results
end
```

```
parsed_results = parse_results(results, pi)
```

1.3.4 Análise Visual: Gerando Gráficos

A função `plot_raw_results` gera um gráfico com cada uma das repetições da estimativa do valor da integral, no eixo y, para cada uma das quantidades de repetições, no eixo x. Usamos o pacote Plots para gerar os gráficos.

Para o miniEP1, você não precisa escrever seus próprios gráficos, mas recomendamos fortemente que você aprenda alguma biblioteca de plotagem, pois vai precisar delas nos EPs seguintes.

Sugerimos aprender uma das seguintes bibliotecas:

- ggplot2 (R)
- matplotlib (Python)
- Plots (Julia)

Todas essas bibliotecas podem ser usadas em Notebooks Jupyter. Execute a célula abaixo:

[ ]:
```
function plot_raw_results(parsed_results, target_label)
    plot(parsed_results.samples,
         parsed_results.target,
         labels = target_label,
         line = :dash,
         width = 2.0)
    scatter!(results.samples,
             results.integral,
             xscale = :log10,
             alpha = 0.4,
             labels = "integral estimate")
end

plot_raw_results(parsed_results, "pi")
```

1.3.5 Exercícios

Os exercícios procuram revisitar conceitos de estatística básica, dando a oportunidade de analisar qualitativa e quantitativamente resultados de experimentos computacionais. A metodologia exercitada aqui será útil nos EPs subsequentes, e em casos futuros onde a análise de experimentos seja necessária.

Exercício 1 **a)** Execute a célula acima para gerar o gráfico. Observando o gráfico gerado, descreva *qualitativamente* o comportamento das estimativas da integral, em relação ao seu valor esperado, à medida que aumentamos o número de pontos amostrados. Use a célula abaixo para escrever sua resposta.

b) Leia o verbete sobre valor esperado na Wikipedia, e escreva um parágrafo definindo a média de uma amostra. Como podemos interpretar a média? Escreva uma fórmula em LaTeX para a média.

c) Leia o verbete sobre intervalo de confiança na Wikipedia, e escreva um parágrafo com uma **descrição da intuição** por trás da definição de intervalo de confiança. Explique também a relação entre a variância estimada de uma amostra, o desvio padrão, e o intervalo de confiança a 95%.

Escreva fórmulas em LaTeX para a variância e o desvio padrão, mas não se preocupe com a formalização da definição do intervalo de confiança, estamos interessados na intuição por trás da ideia.

d) A função `plot_results` gera um gráfico sumarizando os dados da figura anterior. O gráfico gerado mostra a média das estimativas e o intervalo de confiança 95%, calculados anteriormente.

Execute a célula abaixo e gere o gráfico. Interprete os resultados *quantitativamente*, em termos da média e do intervalo de confiança. Responda também:

1. Esse é o comportamento que você esperava para a estimativa pelo método de Monte Carlo? Por que?
2. Que engano poderíamos cometer, se não calculássemos o intervalo de confiança?

[ ]:
```
function plot_results(parsed_results, target_label)
    plot(parsed_results.samples,
         parsed_results.target,
         labels = target_label,
         line = :dash,
         width = 2.0)
    scatter!(parsed_results.samples,
             parsed_results.mean_integral,
             yerror = parsed_results.ci_estimate,
             xscale = :log10,
             labels = "mean(integral estimates)+CI")
end

plot_results(parsed_results, "pi")
```

Exercício 2 **a)** Implemente em Julia a função descrita abaixo:

$$f_2(x) = 2xe^{x^2}$$

A integral de f_2 é dada por:

$$\int_0^1 f_2(x)dx = \int_0^1 2xe^{x^2}dx = e - 1$$

Vamos estimar essa integral usando método de Monte Carlo. Use a célula abaixo, e apenas complete a definição de `f2`. Execute a célula.

[ ]:
```
function f2(x)
end
```

```
results = run_experiments(f2, [0.0, 1.0], [10 ^ x for x in 0:5], 30)
parsed_results = parse_results(results, Base.MathConstants.e - 1)
```

b) Gere as duas figuras, executando as células abaixo. Interprete os resultados *quantitativamente*, como fez no exercício **1d**. Escreva sua resposta na célula abaixo.

[ ]:
```
plot_raw_results(parsed_results, "e - 1")
```

[ ]:
```
plot_results(parsed_results, "e - 1")
```

c) Implemente em Julia a função abaixo:

$$f_3(x) = \frac{1}{\ln x} + \frac{1}{1-x}$$

A integral de f_3 é dada por:

$$\int_0^1 f_3(x)dx = \int_0^1 \left(\frac{1}{\ln x} + \frac{1}{1-x}\right) dx = \gamma$$

Use a célula abaixo, e apenas complete a definição de `f3`. Execute a célula.

[ ]:
```
function f3(x)
end

results = run_experiments(f3, [0.0, 1.0], [10 ^ x for x in 0:5], 20)
parsed_results = parse_results(results, Base.MathConstants.eulergamma)
```

d) Gere as duas figuras, executando as células abaixo. Interprete os resultados *quantitativamente*, como fez no exercício **1d**. Escreva sua resposta na célula abaixo.

[ ]:
```
plot_raw_results(parsed_results, "eulergamma")
```

[ ]:
```
plot_results(parsed_results, "eulergamma")
```