Projeto de π

CM116 - Exercício 3

Entrega: 01 de Junho até 17h30

1 Introdução

π foi definido há bastante tempo, e seu uso na matemática e em campos derivados é enorme. π é um número famoso na matemática por vários motivos: ele é irracional e transcedental; aparece em várias situações e áreas; é confuso para os não matemáticos; tem citações e equações "legais" envolvendo ele, etc.

Apesar de tudo isso, pela maneira que é definido, seu valor foi razoavelmente difícil de obter, isto é, com algumas casas de precisão. Este trabalho é sobre π e seu valor.

2 Trabalho

Este trabalho é **individual** e **colaborativo**. Ele envolve criar **um único projeto** com várias partes; Trabalhem no Sharelatex separando o trabalho em seções e subseções. Wikipedia é seu amigo, mas quero outras duas fontes pelo menos, citadas e acessíveis. São dez partes, sendo cinco não computacionais, e cinco computacionais. As partes computacionais, como sempre, devem ser feitos em Julia. O código deve estar incluído no relatório. A entrega será no segunda **01 de Junho, até 17h30**. Tenha certeza de incluir tudo no relatório até esse momento. Mais importante, vocês **podem** e **devem** trabalhar juntos. Não deixem tudo para o fim.

A avaliação será feita sobre o projeto inteiro e sobre as seções individuais.

- Uma breve história sobre π;
- Representações gráficas de π;
- Relação de π com polígonos;
- Discussão sobre a aproximação 22/7;
- Aproximação manual de π usando moedas;
- Aproximações numéricas de Gauss-Legendre;
- Aproximações numéricas pela expansão de Taylor de $\frac{\pi}{4} = 4 \arctan \frac{1}{5} - \arctan \frac{1}{239}$;
- Aproximações numéricas pela fórmula de Viète;
- Aproximações numéricas pela fórmula de Bailey-Borwein-Plouffe;
- Aproximações numéricas pela Transformação de Newton/Euler.

3 Parte computacional

Todos os algoritmos acima são dependentes de algum valor que trunca a aproximação. Esse valor, quando muito grande, irá afetar a estabilidade numérica do método. Você deve tomar cuidado com isso para que o método não exploda ou algo parecido.

Por exemplo, lembre-se da expansão de Taylor

$$e^x = 1 + x + \frac{x^2}{2} + \frac{x^3}{3!} + \frac{x^4}{4!} + \cdots + \frac{x^n}{n!} + \cdots = 1 + \sum_{n=1}^{\infty} \frac{x^n}{n!}.$$

Numericamente, escolhemos um valor finito e truncamos a série até esse número. Uma implementação ruim seria

```
function exp1(x; N = 10)
  s = 1.0
  for n = 1:N
    s += x^n/factorial(n)
  end
  return s
end
```

Usando um valor `N` maior que 20, o fatorial explode e não conseguimos calcular essa expansão. Uma opção melhor seria

```
function exp2(x; N = 10)
  s = 1.0
  t = 1.0
  for n = 1:N
    t *= x/n
    s += t
  end
  return s
end
```

Outra coisa a considerar é a qualidade da solução para muitas casas decimais. Veja:

```
julia> exp2(5) - e^5
-2.032558077444321
julia> exp2(5; N = 40) - e^5
0.0
```

Isto é, com `N` igual a 10, a aproximação é ruim, mas com `N` igual a 40, ela já é ótima. No entanto, quando eu uso `BigFloat`, obtenho

```
julia> exp2(5; N = 40) - BigFloat(e)^5
-3.4863514900464197605...e-15
julia> exp2(5; N = 50) - BigFloat(e)^5
-3.4863514900464197605...e-15
```

Isto é, a aproximação tem 14 casas decimais corretas, e aumentar o N não ajuda mais, porque não usamos BigFloat.

Os alunos que quiserem realmente uma aproximação decente para π deverão usar `BigFloat` em alguns lugares estratégicos. **Mas primeiro façam sem BigFloat**. Por exemplo

```
function exp3(x; N = 10)
  s = BigFloat(1)
  t = BigFloat(1)
  for n = 1:N
```

```
    t *= x/n
    s += t
  end
  return s
end
```

```
julia> exp3(5; N = 40) - BigFloat(e)^5
-1.542...e-15
julia> exp3(5; N = 50) - BigFloat(e)^5
-3.1669...e-31
julia> exp3(5; N = 100) - BigFloat(e)^5
-2.210859...e-75
julia> exp3(5; N = 200) - BigFloat(e)^5
-2.210859...e-75
```

O número de dígitos corretos aumenta bastante. Note, no entanto que depois uma certa quantidade ele limita novamente. Isso acontece porque o BigFloat usa uma certa quantidade de bits para fazer essa conta. Uma quantidade maior que a normal, mas ainda uma finita. Podemos aumentar essa quantidade e obter mais precisão. O inicial é `256`. Vamos a

```
julia> set_bigfloat_precision(512)
julia> exp3(5; N = 100) - BigFloat(e)^5
-4.4...e-90
julia> exp3(5; N = 200) - BigFloat(e)^5
-1.9...e-152
```

Podemos continuar até ficar satisfeito ou esgotar a memória do computador.