UNIVERSIDADE FEDERAL DE VIÇOSA

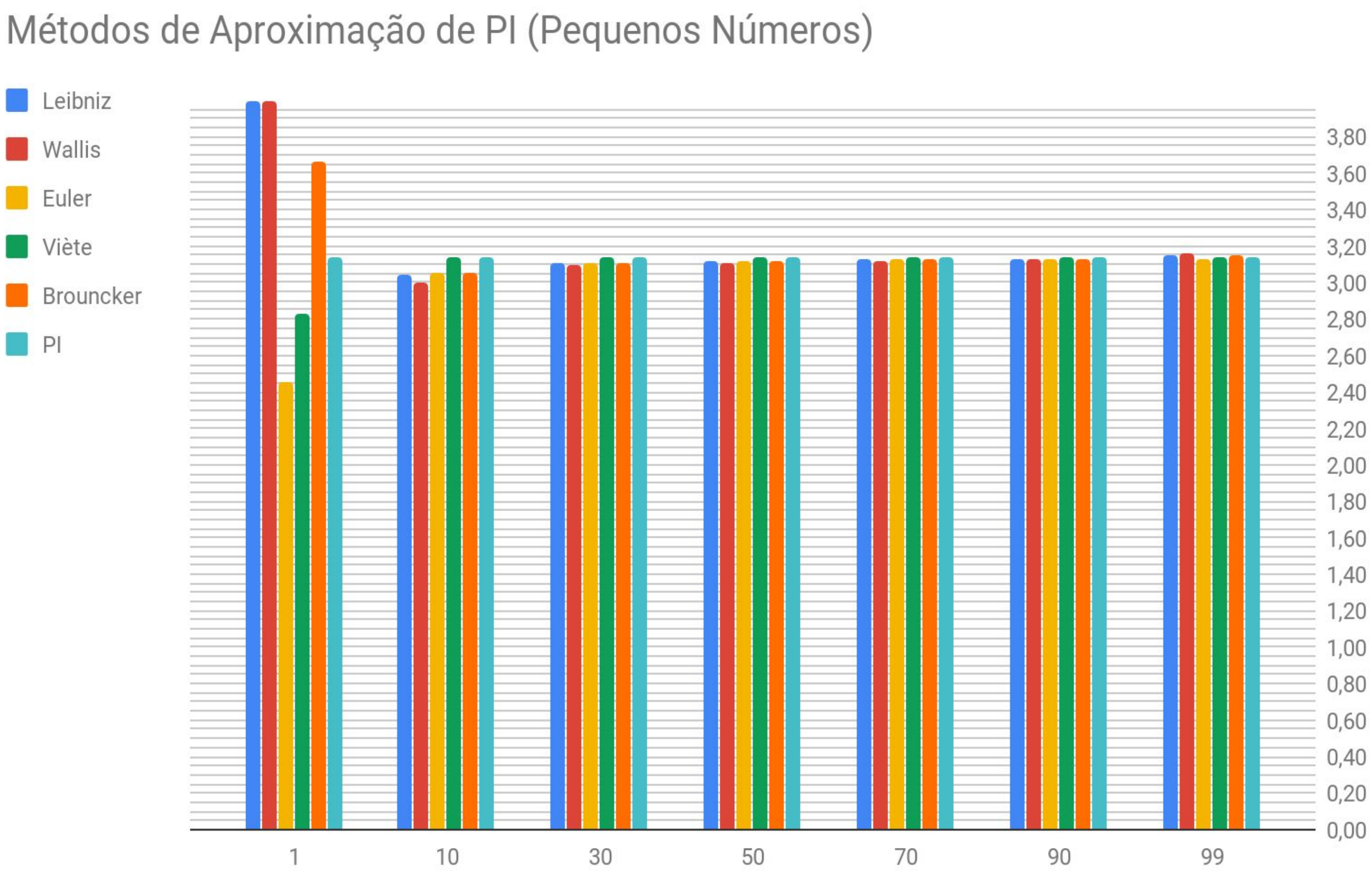
Aluno: Erick Lima Figueiredo
Matrícula: 98898
Data: 20/05/2019

Professor: André Augusto
Disciplina: Programação I

Relatório: Métodos de aproximação de **π**

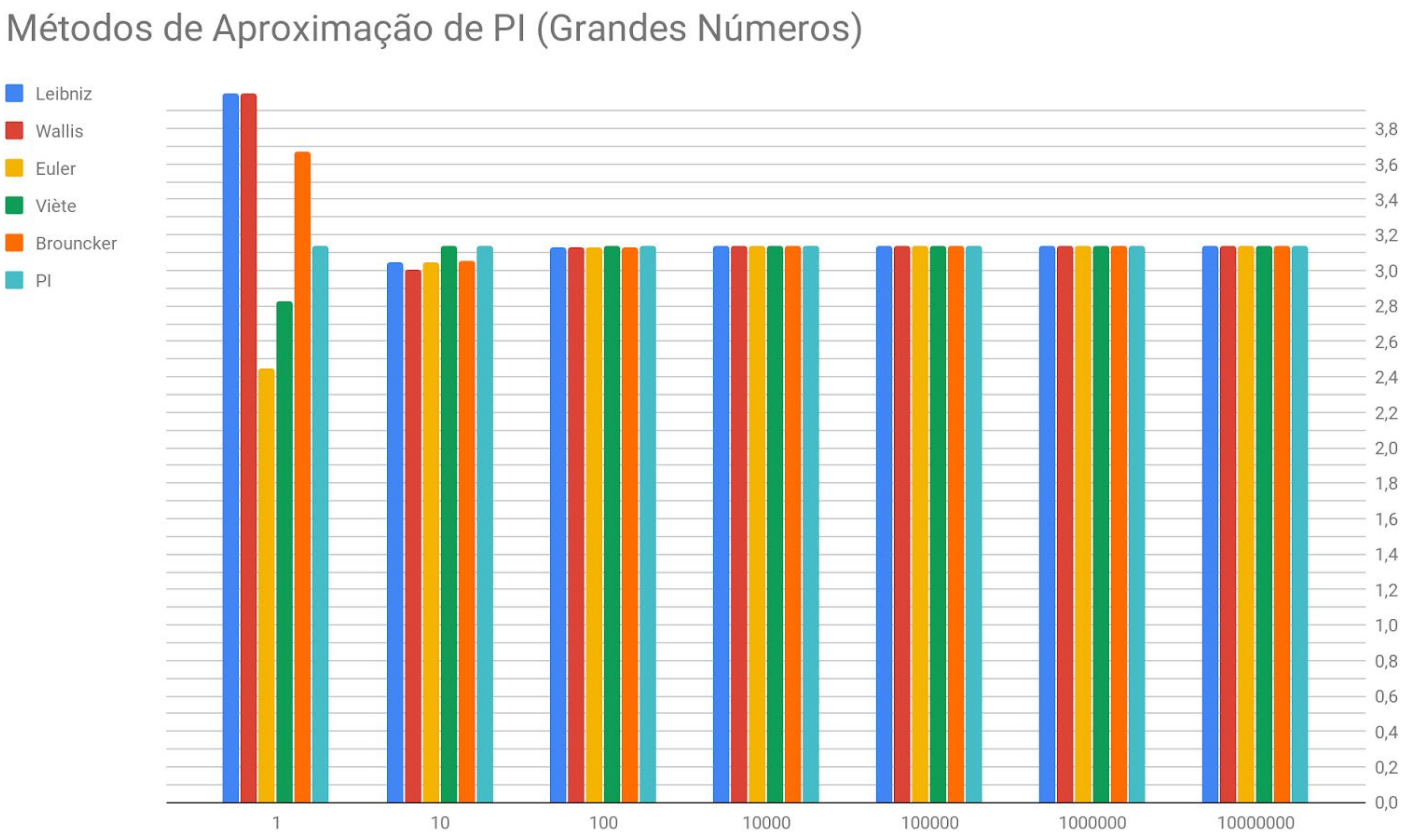
Inicialmente, foi desenvolvido um algoritmo que engloba os cinco métodos matemáticos de aproximação de pi, por série infinita, apresentados no enunciado do trabalho. Cabe ao usuário escolher, a partir do menu, um dos métodos enumerados de um à cinco, seguindo a ordem em que foram apresentados no enunciado, sendo a 6 opção a seleção de todos os métodos para a submissão à um mesmo número limite de elementos na sequência. As fórmulas se manifestam no programa por meio de procedimentos e o laço do algoritmo é finalizando quando a escolha do usuário no menu inicial recebe zero.

Veremos à seguir um grafico gerado a partir da ferramenta Google Sheets[1], reprensentando o quão próximo do número pi chegam as fórmulas com sequências que estão no intervalo [1, 99].



No primeiro gráfico percebemos que os métodos de “Leibniz”, “Wallis” e “Brouncker” têm valores acima de pi, com apenas um termo na sequência. Percebemos também, que entre entre 10 e 30 termos na sequência, todos os métodos já se aproximam bastante do valor referencial, com destaque para o “Método de Viète”, que entre 1 e 10 termos já apresenta um valor muito próximo do desejado.

Partindo para sequências com mais números, temos o seguinte resultado representado graficamente:



Analisando o segundo gráfico, percebemos que, com os recursos que temos acesso até o momento, os valores representados acima, no intervalo [1x10^2, 1x10^8] oscilam muito pouco ficando próximos de 3,14159265. Porém, mais uma vez, o “Método de Viète” tem destaque por ir além da precisão obtida pelos demais no maior número de elementos calculados, chegando em 3.14159265358979.

Por fim, acredito que o trabalho possibilitou aplicar bem o conteúdo estudado até o momento, com um tema que será abordado futuramente pela disciplina de cálculo, de forma divertida e instigante.

[1]**Observação:** A ferramenta do Google limita ao uso de 14 casas depois da vígula.

Tabela dos Métodos de aproximação de Pi

https://docs.google.com/spreadsheets/d/1MaLHj3WSKqWPsO59WQt3vl6UM6Icryu5Yn-f1vavRUw/edit?usp=sharing

Código fonte

```
//Matricula: 98898 | Aluno: Erick Lima Figueiredo.

#include <iostream>
#include <iomanip>
#include <cmath>

using namespace std;

//Prototipação dos procedimentos

void leibniz(long long int num);
void wallis(long long int num);
void euler(long long int num);
void viete(long long int num);
void brouncker(long long int num);

//Método Principal

int main(){
	int escolha = -1;
	do{
		cout <<"CALCULO APROXIMADO DO VALOR DE PI\n\n\n";
		cout << "1-Leibniz\n2-Wallis\n3-Euler\n4-Viete\n5-Brouncker\n6-Executar
todos\n0-Sair\n\nOpcao: ";
		cin >> escolha;

		//Verifica se o usuário digitou 0, caso verdadeiro o loop finaliza

		if(escolha == 0)
			break;

		//Verifica se a escolha é uma opção válida

		else if((escolha >=1) &&(escolha <=6))
			cout << "Informe o numero de elementos da sequencia (quanto maior, maior a
precisao): ";
		else
			cout << "\n\n\nOpcao indisponivel!!!\n\n\n";
		switch(escolha){
			long long int num;

			case 1:
				cin >> num;
```

```
                    leibniz(num);
                    break;
                case 2:
                    cin >> num;
                    wallis(num);
                    break;
                case 3:
                    cin >> num;
                    euler(num);
                    break;
                case 4:
                    cin >> num;
                    viete(num);
                    break;
                case 5:
                    cin >> num;
                    brouncker(num);
                    break;
                case 6:
                    cin >> num;
                    cout<<"Isso pode demorar um pouco...\n";
                    leibniz(num);
                    wallis(num);
                    euler(num);
                    viete(num);
                    brouncker(num);
                    break;

            }
            cout<<"\n\n\n\n\n"<<endl;

        // Sempre executa até que haja um break

        }while(true);
        return 0;
}

void leibniz(long long int num){

        //Iniciamos o valor do denominador

        long long int cont = 1;

        //Iniciamos o resultado com o elemento neutro da adição/subtração

        long double result = 0;
```

```
	//laço para realizar a sequência aé o número escolhido pelo usuário

	for(int i = 0; i < num; i++){
		if(i%2 == 0){

			//caso o número ocupe uma posição par ele é positivo

			result += (1.0/cont);
		}else{

			//caso contrário é negativo

			result += -(1.0/cont);
		}

		//A cada execução o denominador recebe +2 em seu valor

		cont += 2;
	}

	/*Utilizamos o precision com 30 casas decimais depois da vírgula e multipli
	camos o resultado da operação por 4, pois a formula dá o resultado da quarta
	parte de pi*/

	cout <<"Na sequencia de "<< num <<" numeros pelo metodo de Leibniz, temos pi como:
"<<fixed<<setprecision(30)<< result*4 << endl;
}

void wallis(long long int num){

	/*Iniciamos o numerador com 0 pois desse modo ao receber +2 em seu valor conti
	nuará par, já o denominador foi iniciado com 1, pois sempre será impar e,
	assim como o númerador, tem diferenca de 2 com o próximo denominador. Já o
	resultado recebe 1, por ser o elemento neutro da multiplicação e não afetar
	na acumulação*/

	long double numerador = 0.0, denominador = 1.0, result = 1;

	//laço para realizar a sequência aé o número escolhido pelo usuário

	for(int i = 0; i < num; i++){
		if(i%2 == 0){

			//caso o número ocupe uma posição par o numerador recebe +2 em seu valor

			numerador += 2;
			result *= (numerador/denominador);
```

```
        }else{

            //caso contrário o denominador recebe +2 em seu valor

            denominador += 2;
            result *= (numerador/denominador);
        }
    }

    /*Utilizamos o precision com 30 casas decimais depois da vírgula e multipli
    camos o resultado da operação por 2, pois a formula dá o resultado da metade
    de pi*/

    cout <<"Na sequencia de "<< num <<" numeros pelo metodo de Wallis, temos pi como:
"<<fixed<<setprecision(30)<< result*2 << endl;
}

void euler(long long int num){
    long long int agregador  = 0;

    /*Denominador inicial recebe 1 e o valor do primeiro resultados dos elementos
    da sequência também*/

    long double denominador = 1.0, result = 1.0;

    //laço para realizar a sequência aé o número escolhido pelo usuário

    for(int i = 0; i < num; i++){
        if(i == 0){

            //Agregador ao denominador recebe 3 na primeira execução do laço

            agregador += 3;
        }else{

            //Adiciona o agregador ao denominador

            denominador += agregador;

            //Realiza a divisão e adiciona seu resultado à soma já obtida até então

            result += (1/denominador);

            /*A partir da segunda execução do laço o agregador vai recebendo mais
            dois em seu valor que na próxima execução vai ser agregada ao
denominador*/
```

```
                    agregador += 2;
            }
        }
        cout <<"Na sequencia de "<< num <<" numeros pelo metodo de Euler, temos pi como: 
"<<fixed<<setprecision(30)<< sqrt(result*6) << endl;
}

void viete(long long int num){

        /*vamos considerar o númerador e o denominador invertidos. o denominador recebe
        0 e o resutado recebe o elemento neutro da multiplicação*/

        long double denominador = 0.0, result = 1;

        //laço para realizar a sequência aé o número escolhido pelo usuário

        for(int i = 0; i < num; i++){
            if(i == 0){

                    /*Denominador recebe raiz de dois e é feita a divisão por dois e a
                    multiplicação pelo resultado acumulado já na primeira execução do
                    laço de repetição*/

                    denominador = sqrt(2);
                    result *= 2/denominador;
            }else{

                    /*Denominador a partir do segundo laço recebe a raiz de 2 mais ele mesmo*/

                    denominador = sqrt(2+denominador);
                    result *= 2/denominador;
            }
        }

        /*Como invertemos a equação, basta multiplicar o resultado por 2, para obter
        uma aproximação de pi*/

        cout <<"Na sequencia de "<< num <<" numeros pelo metodo de Viete, temos pi como: "<<
fixed << setprecision(30) << result*2 << endl;
}

void brouncker(long long int num){

        //Iniciamos o denominador com 0 e o resultado recebe o elemento neutro da adição

        long double den = 1, result = 0;
        for(int i = 0; i < num; i++){
```

```
            //Para fazer a operação precisamos do último de todos os denominadores

            den += 2;
        }
        for(int i = 0; i < num; i++){

            //Percebe-se que, a base de cada numerador é -2 a base do númerador

            if(i == 0){

                /*No ultimo valor do denominador da sequência começamos a operação,
                lembrando da relação numerador - denominador descrita anteriormente*/

                result = ((den-2)*(den-2))/(2+(den*den));

                //Feita a operação decrementamos 2 do valor ultimo do denominador

                den -= 2;
            }else{

                /*A partir daqui, a cada passo, pegamos o resultado anterior e com a
                relação numerador - denominador, obtemos um novo resultado */

                result = ((den-2)*(den-2))/(2+result);

                //Decrementamos 2 do valor do ultimo denominador
                den -= 2;
            }
        }

        //Feita a parte sequencial, somamos 1 ao resultado, dividimos o mesmo por 4

        result = (result+1)/4;

        /*Exibimos como 1/result, pois na formula o a relação deixa de ser 1/pi = result
        para ser pi = 1/result*/

        cout <<"Na sequencia de "<< num <<" numeros pelo metodo de Brouncker, temos pi como:
"<< fixed << setprecision(30) << (1/result) << endl;
}
```