



MARCELLO HENRIQUE FERREIRA DOS SANTOS

Tecnologia Chebfun2

Campinas
22-11-2024

MARCELLO HENRIQUE FERREIRA DOS SANTOS

Tecnologia Chebfun2

Monografia apresentada ao Instituto de Matemática, Estatística e Computação Científica da Universidade Estadual de Campinas como parte dos requisitos para obtenção de créditos na disciplina Projeto Supervisionado, sob a orientação do(a) Prof^o Lucio Tunes dos Santos.

Resumo

Este projeto teve como objetivo explorar o uso do pacote Chebfun no MATLAB, com foco em interpolação polinomial e análise numérica de funções de uma, duas e três variáveis. Inicialmente, foram investigados os fundamentos matemáticos por trás do Chebfun, com destaque para os pontos de Chebyshev e a fórmula de interpolação baricêntrica. A transição para o Chebfun2 possibilitou o estudo de funções bivariadas, destacando a eficiência da aproximação de low-rank e o uso de eliminação gaussiana com pivoteamento completo em substituição à SVD tradicional. Foram analisadas aplicações práticas como diferenciação, integração e localização de zeros em funções bivariadas, com ênfase na precisão e desempenho do Chebfun2. Por fim, o Chebfun3 foi utilizado para estender esses conceitos a funções trivariadas, explorando sua estrutura baseada em decomposições tensoriais e as aplicações práticas associadas. A metodologia baseou-se em experimentos computacionais no MATLAB, aliados à revisão teórica e análise de desempenho. Os resultados demonstraram a eficácia do Chebfun como ferramenta para problemas computacionais envolvendo funções suaves e destacaram suas limitações, como a dependência de funções analíticas e restrições em alta dimensionalidade.

Abstract

This project aimed to explore the use of the Chebfun package in MATLAB, focusing on polynomial interpolation and numerical analysis of functions of one, two, and three variables. Initially, the mathematical foundations behind Chebfun were investigated, with emphasis on Chebyshev points and the barycentric interpolation formula. The transition to Chebfun2 enabled the study of bivariate functions, highlighting the efficiency of low-rank approximations and the use of Gaussian elimination with complete pivoting as a replacement for traditional SVD. Practical applications such as differentiation, integration, and zero-finding in bivariate functions were analyzed, with a focus on the precision and performance of Chebfun2. Finally, Chebfun3 was utilized to extend these concepts to trivariate functions, exploring its tensor decomposition-based structure and associated practical applications. The methodology relied on computational experiments in MATLAB, combined with theoretical review and performance analysis. The results demonstrated the effectiveness of Chebfun as a tool for computational problems involving smooth functions and highlighted its limitations, such as its dependence on analytical functions and restrictions in high-dimensional settings.

Sumário

1	Introdução.	6
2	Um resumo sobre o Chebfun e algumas aplicações.	6
2.1	Desenvolvimento do Chebfun.	7
2.2	Aplicações.	8
3	Chebfun2: A evolução do Chebfun.	11
3.1	Aspectos Teóricos.	11
3.2	Aspectos Práticos.	12
3.3	Chebfun2v e Funções Vetoriais.	17
4	Chebfun3 e funções de três variáveis.	21
5	Aplicações do Chebfun2.	23
5.1	A função de Dixon-Szego.	23
5.2	Interseção de Curvas no Plano Complexo.	25
5.3	Identificação de pontos críticos.	26
6	Conclusão	27
7	Referências.	28

1 Introdução.

A interpolação polinomial é um método usado para encontrar um polinômio que passe exatamente por um conjunto de pontos (x, y) dados. Em outras palavras, a interpolação polinomial procura um polinômio de grau adequado, tal que, um polinômio $P(x)$ satisfaça $P(x_i) = y_i$ para cada ponto (x_i, y_i) no conjunto de pontos dados. Existem diferentes formas de encontrar o polinômio que interpola esses pontos sendo os mais famosos o método de Newton e a forma de Lagrange.

A unicidade desse polinômio interpolador é obtida por um teorema de álgebra que diz que existe um e só um polinômio de grau n ou menor que assume valores específicos para $n + 1$ valores de x .

Para algumas funções como é o caso da função de Runge: $f(t) = \frac{1}{1 + (kt)^2}$ para k uma constante real, ocorre o fenômeno de Runge, que nada mais é do que o erro entre o polinômio e o valor da função aumentar nos extremos do intervalo que o polinômio foi calculado e, ainda, pode se provar que o erro de interpolação tende para infinito quando o grau do polinômio aumenta.

Para minimizar esse erro, uma alternativa é olhar para o conjunto de pontos e nisso, alguns conjuntos se mostram melhores que outros, pontos igualmente espaçados não produzem o mesmo resultado que pontos acumulados nos extremos. Alguns desses pontos são os chamados pontos de Chebyshev, que é a base do pacote para MATLAB Chebfun, o principal elemento de estudo desse trabalho.

Sabendo dessas questões, em 2002, Zachary Battles e Nick Trefethen começaram a desenvolver para MATLAB, um pacote chamado *Chebfun* que tem como objetivo lidar com problemas que envolvem funções matemáticas, ele representa funções usando como base a expansão em série dos polinômios de Chebyshev. Segundo Trefethen, "Nosso objetivo é alcançar para funções o que a aritmética de ponto flutuante alcança para números: Cálculo rápido no qual cada operação sucessiva é realizada exatamente, com exceção de um erro de arredondamento que é muito pequeno em termos relativos.". Por isso, é uma ferramenta poderosa para resolver problemas matemáticos de forma numérica, por exemplo calcular derivadas, integrais, problemas envolvendo EDO's e diversos outros.

Nesse trabalho, será feita uma introdução para aqueles que nunca ouviram falar sobre o Chebfun e algumas aplicações, depois, o principal objeto de estudo será o Chebfun2, uma extensão do Chebfun para lidar com problemas em duas dimensões e por fim, será feito um breve comentário sobre o Chebfun3, outra generalização mas dessa vez para 3 dimensões, porém, essa parte ainda está em desenvolvimento pelos criadores.

2 Um resumo sobre o Chebfun e algumas aplicações.

O Chebfun é um pacote do MATLAB que realiza operações com funções representando-as usando a base de polinômios de Chebyshev com o intuito de obter boa estabilidade numérica e cálculos mais eficientes, além disso, ele permite efetuar operações sobre funções como integrações, derivações, solução de EDO's, encontrar zeros entre outras inúmeras possibilidades e que rotineiramente precisam ser resolvidas, o Chebfun, sempre que possível, usa a precisão da máquina para entregar seus resultados, que é geralmente de 16 dígitos.

Para o pacote, uma função de uma variável num intervalo $[a, b]$ é um *chebfun*, eles são o principal objeto do pacote pois são eles que representam as funções, logo carregam os métodos para se resolver problemas como uma integral definida da função no intervalo. A sintaxe mais usual para criar um chebfun é gerar uma variável que tem como argumento

uma *string* com o texto da função identidade, ou seja, 'x' e depois especificar seu intervalo, caso não seja especificado, será usado o padrão, que é $[-1, 1]$ e, por fim, esse chebfun criado pode ser usado para gerar outros chebfuns. Um exemplo desse processo para a função $\sin(5x)$ no intervalo $[-2, 2]$ está na Figura 1.

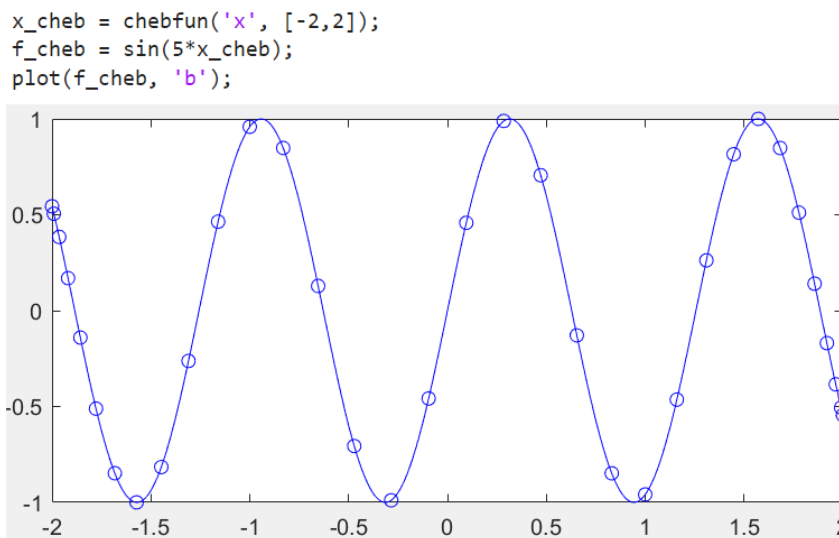


Figura 1: Sintaxe do comando chebfun e pontos usados representados em cima da curva para $\sin(5x)$

2.1 Desenvolvimento do Chebfun.

A tecnologia de polinômios de Chebyshev, que começou a ser utilizada para interpolações no século XIX pelo matemático russo Pafnuty Chebyshev e seus discípulos, foi uma descoberta significativa. Eles identificaram que, assim como as séries de Fourier servem para representar funções periódicas, as séries de Chebyshev poderiam ser aplicadas a funções não periódicas. Com o advento dos computadores, métodos mais avançados foram desenvolvidos, como a Transformada Rápida de Fourier (FFT) e técnicas que empregam polinômios de Chebyshev para encontrar interpoladores. Isso culminou na criação do software Chebfun.

Como dito anteriormente, o Chebfun é baseado em aproximar funções usando polinômios de Chebyshev, ele faz isso usando do fato de que funções Lipschitz, ou seja, que não variam demasiadamente rápido, podem ser expressadas usando uma série de Chebyshev:

$$f(x) = \sum_{k=0}^{\infty} a_k T_k$$

Daí então o pacote faz com que as funções sejam representadas em séries de polinômios de Chebyshev, ao invés de outras formas como a clássica expansão em série de Taylor. Por exemplo, ao trocar a base, a função $f_1(x) = x^2 + 2x + 1$ pode ser representada por $p_1(x) = \frac{1}{2}T_2(x) + 2T_1(x) + \frac{3}{2}T_0(x)$.

O Chebfun realiza as suas aproximações da seguinte maneira:

1. Encontrar os pontos de Chebyshev.

Os $n + 1$ pontos de Chebyshev são definidos por:

$$x_j = \cos\left(\frac{j\pi}{n}\right) \text{ para } j = 0, 1, \dots, n \quad (1)$$

2. Encontrar os coeficientes da interpolação polinomial.

O Chebfun usa esses pontos para construir um polinômio de interpolação que aproxima a função $f(x)$. O polinômio é expresso na base de polinômios de Chebyshev:

$$p(x) = \sum_{k=0}^{\infty} c_k T_k(x)$$

onde:

$T_k(x)$ são os polinômios de Chebyshev de primeira espécie e c_k são os coeficientes calculados a partir da função $f(x)$ avaliada nos pontos de Chebyshev, esses são calculados usando a transformada discreta do cosseno (TDC).

3. Encontrar o menor número de pontos que fornece a maior precisão numérica.

O Chebfun ajusta o número de pontos n para atingir a precisão desejada. Ele começa com um número pequeno de pontos e os aumenta progressivamente até que o erro de aproximação esteja dentro do critério de aceite, o padrão é usar a precisão da máquina mas isso pode ser mudado para a precisão desejada.

Esse processo é chamado de *adaptive sampling*, e o Chebfun verifica a convergência medindo a magnitude dos coeficientes c_k e quando esse valor para índices altos se torna suficientemente pequenos, ele para o refinamento.

A partir disso, é possível resolver diversos problemas, cada um usa uma abordagem diferente mas todos partem do princípio mostrado nessa seção, garantindo eficiência, estabilidade numérica e precisão. Um exemplo é na avaliação da função em valores pontuais, para esse processo, a fórmula de interpolação baricêntrica é usada mas para operações como derivação, integração, os coeficientes c_k são extremamente importantes.

2.2 Aplicações.

Para o Chebfun, o método *sum* com um chebfun f de argumento, calcula a integral definida de f , no intervalo que o chebfun foi definido ou, ainda, no intervalo que for especificado nos argumentos.

Um exemplo usando a função $f(x) = e^x \sin(x)$ e depois computando seu valor exato está na Figura 2, fica evidenciado que o erro se torna notável a partir da décima sexta casa decimal.

```
x = chebfun('x', [-2,2]);
f = exp(x)*sin(x);

int_f = sum(f,[0,1]);
int_exata = exp(1)*(sin(1)-cos(1))/2 - exp(0)*(sin(0)-cos(0))/2;

dif = int_exata - int_f;
-----
Saída:
int_f = 0.9093306736314782, int_exata = 0.9093306736314786
e dif = 0.0000000000000003
```

Figura 2: Utilização do método *sum* e comparação com seu valor exato.

Ainda no tópico de integração, muitas vezes o problema exige a solução da integral indefinida e isso não é problema para o Chebfun, usando *cumsum* e como argumento o chebfun, o resultado é um chebfun da integral indefinida do chebfun usado como argumento.

Aplicando o método *diff* em um chebfun, temos como retorno um outro chebfun que representa a derivada do que foi usado como argumento.

O exemplo que está na Figura 3 é o da função $f(x) = \ln(x)$ que tem como derivada a $f' = \frac{1}{x}$. Foi escolhida essa função pois o gráfico é fácil de visualizar que a derivada de fato da função logaritmo.

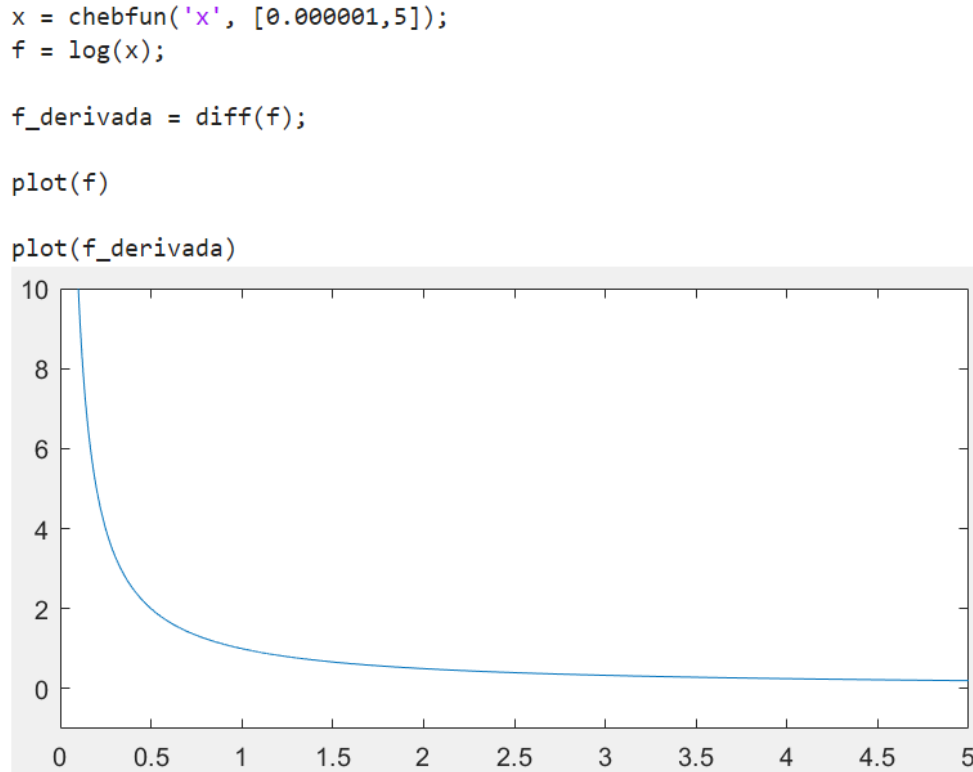


Figura 3: Método *diff* aplicado na função logaritmo, a figura mostra o gráfico da derivada, ou seja, a função $f'(x) = \frac{1}{x}$.

O intervalo de definição do exemplo exclui o zero pela função log não estar definida para $x = 0$.

Além disso, o pacote lida com problemas de valor inicial (PVI) e de contorno (PVC), proporcionando resultados precisos e confiáveis. Trabalhando com equações diferenciais (ou também equações integrais), é introduzido um novo conceito, o *chebop*, que é um representante para o operador linear, da mesma forma que o chebfun é o análogo de uma função, ele contém informações que são essenciais para a definição de um operador, como seu intervalo de definição, a operação que ele realiza num chebfun e as condições de contorno. Para resolver esses tipos de equações, o Chebfun usa os métodos espectrais de *Driscoll-Hale* e *Olver-Townsend*.

Define-se um *chebop* no Chebfun da forma mostrada na Figura 4.

```
L = chebop(-pi, pi);
L.op = @(x,u) diff(u,2) + diff(u);
L.lbc = @(u) diff(u)-1;
L.rbc = 1;
```

Figura 4: Sintaxe para a criação de um chebop.

O Chebfun tenta seguir alguns padrões do MATLAB, o que transforma a experiência do usuário. Para resolver um sistema linear $Ax = b$, basta fazer $x = A \setminus b$, enquanto isso,

usando o `chebop`, o usuário pode fazer $u = L \backslash f$, que resolve a equação $Lu = f$, onde L representa o operador, u a função que deve ser encontrada e f é a parte independente da equação. Suponha que a equação que deve ser resolvida é: $u''(x) - \frac{1}{2}u'(x) = 3$, com as condições de contorno dadas por $u(0) = 1$ e $u(1) = 2$, seguindo os comandos abaixo, a função u que resolve a equação pode ser facilmente encontrada.

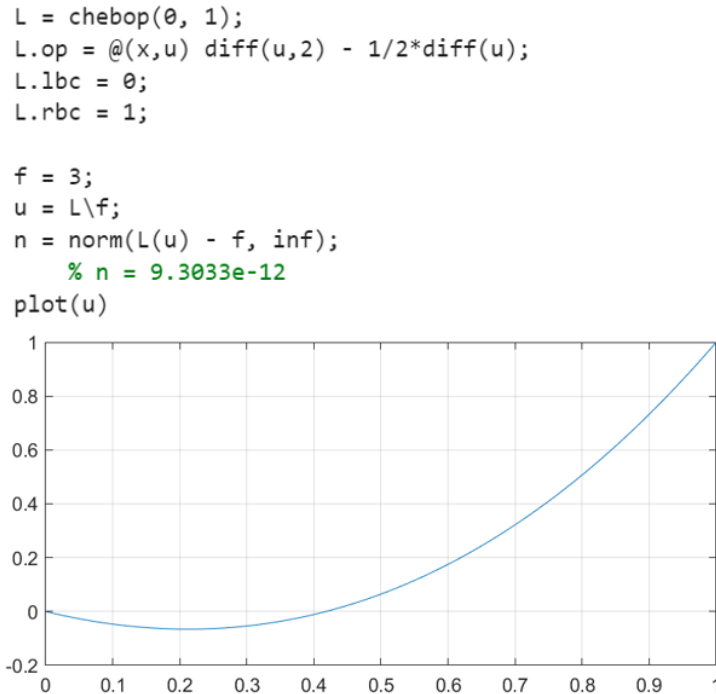


Figura 5: Resolução de uma EDO usando o `chebops`.

E de fato, verificando a norma de Lu , temos que é igual a f .

Esses são apenas alguns exemplos mas o pacote pode fazer muito mais, ele lida bem com aritmética complexa, encontra zeros de funções (isso inclui as complexas), encontra os coeficientes de uma função para séries de Taylor e outros diversos usos, o que o torna muito útil para construir projetos e aplicações.

O `Chebfun`, embora extremamente poderoso para trabalhar com funções unidimensionais, apresenta algumas limitações quando se lida com problemas que envolvem funções de duas ou mais variáveis. Essas limitações aparecem por conta da incapacidade de representar domínios multidimensionais e isso motivou o desenvolvimento de extensões como o `Chebfun2` e `Chebfun3`, que são projetados para lidar com funções bivariadas e trivariadas, respectivamente.

O `Chebfun1` é projetado para trabalhar exclusivamente com funções de uma única variável. Em problemas envolvendo várias variáveis, como:

$$f(x, y) = x^2 + y^2,$$

o `chebfun` não consegue representar diretamente a função $f(x, y)$ ou operar sobre ela, impedindo a resolução de problemas como soluções de EDP's envolvendo esse tipo de problema.

Outras limitações encontradas são o cálculo de derivadas parciais, integrais duplas ou triplas, operar sobre funções de múltiplas variáveis e realizar interpolação bivariada. Além disso o `Chebfun` é restrito a gráficos de funções 1D, como curvas, para visualizar superfícies, contornos ou volumes, é necessário um sistema que suporte representações gráficas 2D ou 3D, algo que o `Chebfun2` e `Chebfun3` podem fornecer.

3 Chebfun2: A evolução do Chebfun.

Dada as limitações do Chebfun, em 2013, foi implementada uma nova funcionalidade, o Chebfun2 foi desenvolvido principalmente por Alex Townsend para expandir as funcionalidades do pacote para funções bivariadas. Ele utiliza aproximações baseadas em **decomposições de baixo rank** para representar funções em duas variáveis de maneira eficiente e precisa, além de oferecer suporte para derivadas parciais, integrais duplas, integrais de linha e muito mais.

Essa transição de 1D para 2D no Chebfun marca uma evolução no pacote, permitindo que problemas mais complexos em análise numérica sejam tratados com a mesma eficiência e precisão que caracterizam o Chebfun.

3.1 Aspectos Teóricos.

O Chebfun2 usa do fato de que muitas funções de duas variáveis podem ser aproximadas por aproximantes de baixo rank. Uma função $f(x, y)$ é dita ser de baixo rank, ou separável, quando pode ser escrita como $f(x, y) = u(y)v(x)$ e é de rank k quando é formada por uma soma de rank 1, ou seja,

$$f(x, y) \approx \sum_{i=0}^k \sigma_i u_i(x) v_i(y)$$

onde σ_i são os valores singulares (advindos da decomposição SVD adaptada), esses valores decaem rapidamente para funções suaves, o motivo que torna essas aproximações boas para o desenvolvimento dessa funcionalidade, $u_i(x)$ e $v_i(y)$ são funções univariadas de x e y , respectivamente, calculadas usando interpolação polinomial, e k é o rank da aproximação, para melhorar a eficiência dos cálculos, o valor k , ao invés de ser determinado pelo cálculo da SVD, é determinado de forma automática usando um algoritmo que funciona como uma aplicação da eliminação gaussiana com pivoteamento completo, ele está disponível com detalhes em [11]. O valor de k pode ser visto usando o comando "length" e passando como argumento um chebfun2, isso é usado na Figura 6 para gerar o nome do gráfico.

```
ff = @(x,y) sin(-40*(- x.*y - 1/2).^2);  
f = chebfun2(ff);  
levels = 0.1:0.1:0.9;  
contour(f,levels), axis([-1 1 -1 1]), axis square  
title(['rank ' int2str(length(f))], 'fontsize', 12)
```

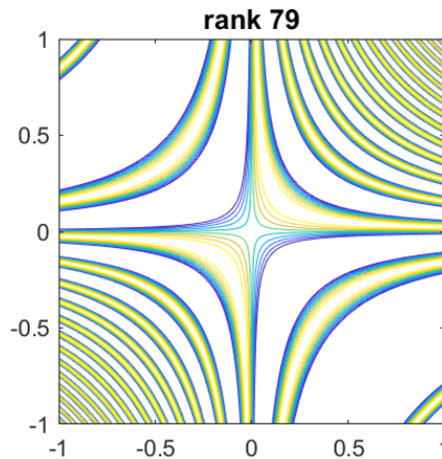


Figura 6: Sintaxe do Chebfun2 e curvas de nível com o número do rank.

A vantagem de se usar a eliminação Gaussiana ao invés da SVD é que, primeiro, é um processo iterativo, ou seja, não é necessário calcular todos os valores singulares de uma vez, tem boa eficiência computacional, é um método adaptativo, ou seja, apenas usa novos pontos de Chebyshev para um novo cálculo quando for preciso além de funcionar bem para funções complexas e mal condicionadas.

A figura 7 mostra diferentes curvas de nível de uma função usando valores diferentes para o rank de sua aproximação e daí pode se perceber que com um rank igual a 56, já é possível atingir uma "precisão de gráfico" pois o gráfico representa bem o formato que as curvas de nível da função tem, como essa função tem rank 79, essa diferença é usada para atingir a precisão numérica.

```

levels = 0.2:0.2:0.8;
clf
for k = 1:9
    axes('position',[.03+.33*mod(k-1,3) .67-.3*floor((k-1)/3) .28 .28])
    contour(chebfun2(ff,7*k),levels,'k')
    xlim([-1 1]), axis equal, axis off
end

```

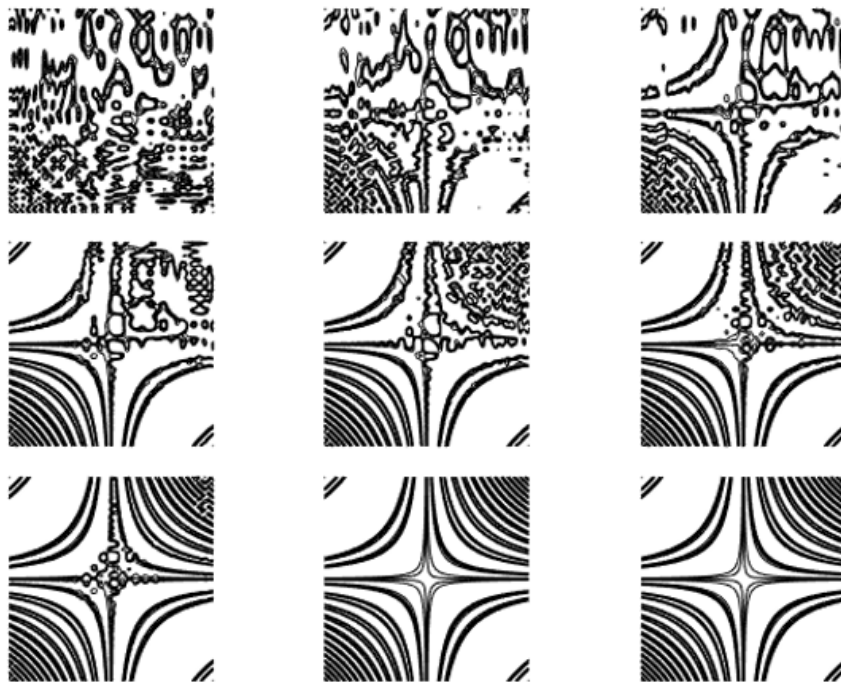


Figura 7: Curvas de nível da função sendo criada usando diferentes valores de rank.

O Chebfun2 também tem suas limitações, algumas serão discutidas no final da próxima seção, mas aqui pode-se notar que uma limitação é em relação ao seu domínio de definição, que deve ser um retângulo. Formas para contornar essa questão foram desenvolvidas dando origem ao "Diskfun" que não será abordado nesse trabalho mas é uma forma de lidar com funções bivariadas mas usando coordenadas polares, ou seja, os problemas serão tratados numa região circular, mais sobre o Diskfun pode ser encontrado em [5].

3.2 Aspectos Práticos.

O Chebfun2 é uma ferramenta poderosa para manipular funções contínuas de duas variáveis de forma numérica. Ele fornece funcionalidades que vão desde operações aritméticas até a solução de problemas complexos envolvendo derivadas, integrais, raízes e interpolação.

Além disso, um fato curioso é que, usando alguns métodos para `chebfun2`, o retorno é um `chebfun`, o que faz sentido matematicamente falando, por exemplo, ao integrar uma função de duas variáveis em apenas uma variável, o resultado ainda é uma função de uma variável, que é análogo a um `chebfun`. Essa seção apresenta os principais recursos práticos e como utilizá-los.

O comando "`chebfun2`" é usado para criar um objeto que representa uma função contínua de duas variáveis, da mesma forma que o `Chebfun`, ele leva como argumento um handle para a função, ou dois `chebfun2` existentes para cada variável e existe ainda uma terceira forma de se definir esse objeto, as três formas estão representadas na Figura 8. O domínio pode ser especificado diretamente, permitindo maior flexibilidade na análise, senão será usado o domínio padrão, $[-1, 1] \times [-1, 1]$.

```
% Definir usando um handle
f = chebfun2(@(x,y) exp(x.^2-y.^2)-2);

% Definir chebfun2 e depois junta-los
x = chebfun2(@(x,y) x);
y = chebfun2(@(x,y) y);
f = chebfun2(@(x,y) exp(x.^2-y.^2)-2);

% Cria as duas variáveis para serem usadas
cheb.xy
f = chebfun2(@(x,y) exp(x.^2-y.^2)-2);

plot(f), zlim([-2 2])
```

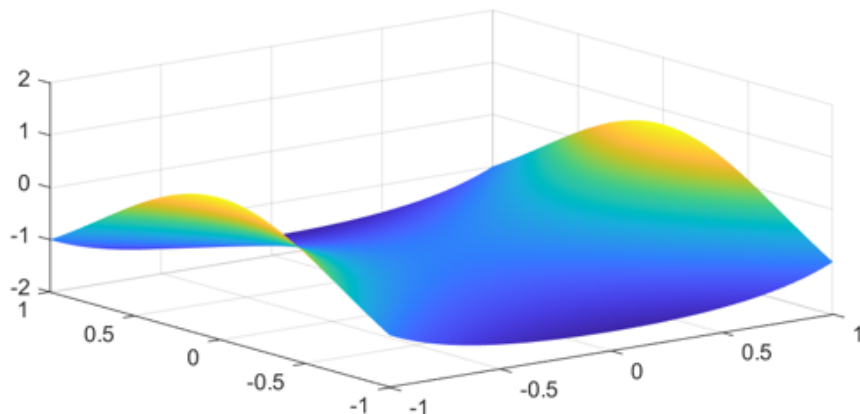


Figura 8: Diferentes formas de se definir um `chebfun2`.

Ao pedir para ver a saída de um `chebfun2`, ele sumariza importantes informações sobre a aproximação realizada, o retorno está mostrado na Figura 9 e as informações são o nome do objeto, o valor do rank usado para aproximar, o domínio de definição da função e os valores que a função assume em cada uma das quinas.

Definido um `chebfun2`, nome dado ao objeto que representa a função no pacote, várias operações podem ser feitas sobre ele. Começando pela integração, da mesma forma que o `Chebfun`, aqui será utilizada a função "`sum2`" que leva como argumento o `chebfun2` e o resultado é a integral definida da função no intervalo de definição da função, ou seja, é uma aproximação de $\iint f(x,y) dx dy$. A figura 10 mostra essa operação comparando com o valor exato dessa aproximação para $f(x,y) = \sin(10xy)$ no intervalo $[0, \pi] \times [0, \pi/6]$.

Usando o método `sum` com argumento sendo um `chebfun2`, o resultado é o cálculo da integral em apenas uma variável, no caso, a primeira, geralmente é o x , para o pacote isso

```
f =
    chebfun2 object
      domain          rank      corner values
    [-1,  1] x [-1,  1]        2    [-1 -1 -1 -1]
vertical scale = 1.6
```

Figura 9: Representação do objeto chebfun2 no pacote.

```
f = chebfun2(@(x,y) sin(10*x.*y),[0 pi 0 pi/6]);
integral = sum2(f);
% ans = 0.341559417294556

exato = 0.3415594172945559;
dif = integral - exato;
% ans = 3.885780586188048e-16
```

Figura 10: Cálculo de uma integral dupla usando "sum2".

é um chebfun e o que faz sentido, uma vez que o resultado obtido é uma aproximação de $I(y) = \int_a^b f(x,y)dx$ mas também, usando $sum(f,2)$, com f sendo um chebfun2 obtém-se uma aproximação para $I(x) = \int_c^d f(x,y)dy$, um exemplo disso está na Figura 11.

```
f = chebfun2(@(x,y) sin(10*x.*y),[0 pi 0 pi/6]);
integral = sum(f);

plot(integral)
```

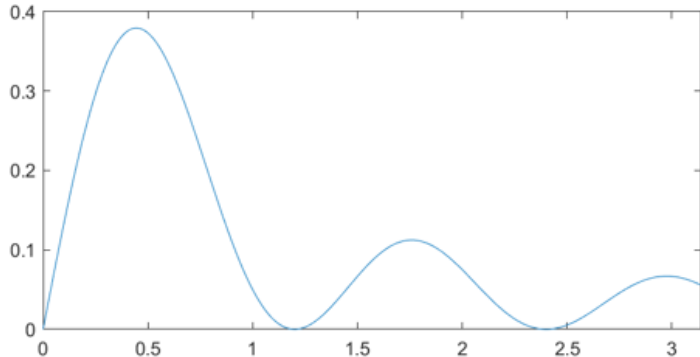


Figura 11: Cálculo de uma integral usando "sum" e tendo como retorno um chebfun.

Obviamente, usando o comando sum no chebfun obtido no passo anterior, o resultado é igual a $sum2$, pois $sum(sum(f)) = sum2(f)$.

Sobre o desempenho computacional dos métodos de integração, o Chebfun2 não foi desenhado para resolver esse tipo de problema e por isso pode não entregar a melhor eficiência computacional quando comparado com outros métodos existentes, até mesmo internos ao próprio MATLAB mas entregam uma precisão numérica legal.

Ainda no tópico de integração, o comando $cumsum2$ retorna um chebfun2 equivalente a integral indefinida da função usada como argumento. Além disso, pode-se usar $cumsum$ para integral em apenas uma variável e aplicando duas vezes, o resultado é parecido com o obtido anteriormente $cumsum(cumsum(f)) = cumsum2(f)$, exemplo está mostrado na Figura 12.

Para finalizar esse tópico sobre integrais, será mostrado como integrar sobre curvas, ou

```
f = chebfun2(@(x,y) sin(3*((x+1).^2+(y+1).^2)));
contour(cumsum2(f)), axis equal
title('Contours of cumsum2(f)'), axis([-1 1 -1 1])
norm( cumsum(cumsum(f),2) - cumsum2(f) )
%ans = 0
```

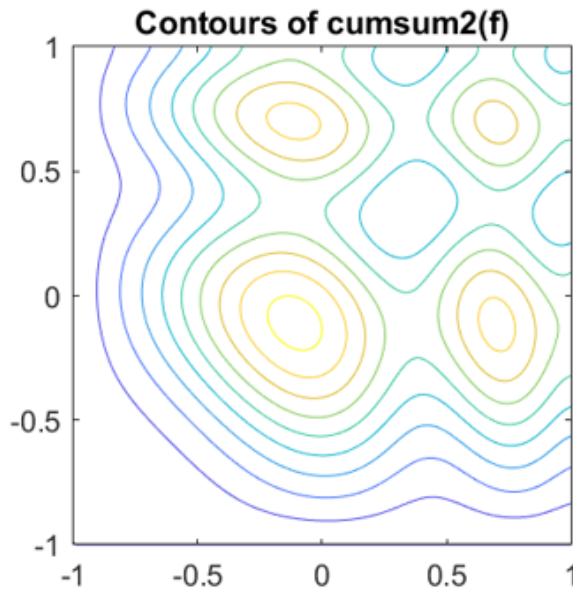


Figura 12: Utilização do "cumsum" para realizar a integração de uma função.

seja, como calcular uma integral em cima da curva C . Primeiro, se define a curva C , que a integração será realizada, daí se define f , o chebfun2 que a integração será realizada e, por fim, usando $sum(f(C))$, o valor da integral é aproximado. Esse processo está representado na Figura 13.

```
C = chebfun(@(t) t*exp(20*i*t),[0 1]);
plot(C)
f = chebfun2(@(x,y) sin(-(-x.*y - 1/2).^2));
plot(f), hold on
plot3(real(C), imag(C), f(C), 'k')
sum(f(C))
% ans = -0.275539630554711
```

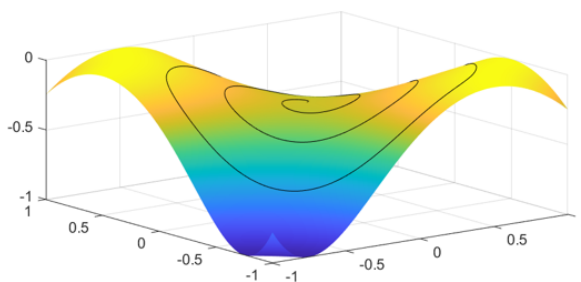


Figura 13: Cálculo de uma função sobre os valores de uma curva.

O Chebfun2 também permite calcular a norma L^2 de uma função, que é definida por:

$$|\phi|^2 = \iint |\phi(x, y)|^2 dx dy$$

A próxima operação feita pelo Chebfun2 é a diferenciação de funções de duas variáveis. O Chebfun2 simplifica esse processo ao fornecer uma interface direta para calcular deri-

vadas parciais e gradientes de funções bivariadas com alta precisão. O comando *diff* é o responsável por esse cálculo, porém para evitar confusões no uso dos argumentos para definir em qual variável a diferenciação seria feita, também foi implementado os comandos *diffx* e *diffy* que facilitam a identificação de qual variável a derivada deve ser calculada. Um exemplo de cálculo de derivada está na Figura 14.

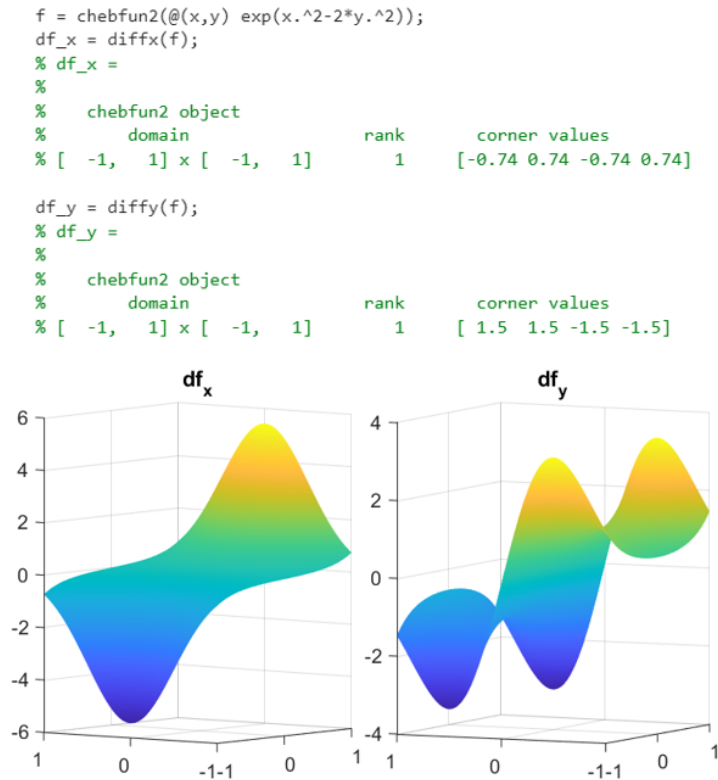


Figura 14: Cálculo de derivadas parciais com o Chebfun2 e o retorno é um chebfun2.

Além disso, é possível também calcular o gradiente de um chebfun2 usando *gradient*. Na Figura 15, esse comando foi aplicado com a mesma função da Figura 13 e os resultados comparados.

```
grad = gradient(f);
norm(grad(1) - df_x)
%ans = 0
norm(grad(2) - df_y)
%ans = 0
```

Figura 15: Comparação dos valores obtidos após usar o gradiente e o método de derivação.

Outra uma aplicação muito útil do Chebfun é encontrar máximos e mínimos, essa função foi estendida para o Chebfun2, possibilitando assim realizar otimizações de forma mais simples e rápidas. Os comandos para encontrar mínimos e máximos são parecidos com os do Chebfun, por exemplo, para encontrar mínimo e o ponto de mínimo, o comando é $[min, min_loc] = min2(f)$, com *min* sendo o valor mínimo e *min_loc* sendo o ponto de mínimo e analogamente para máximos usando *max2*. Usando o comando $[X, Y] = minandmax2(f)$, os valores obtidos são os mesmos que seriam obtidos usando as funções para mínimo e máximo separadamente.

Por fim, é fundamental para funções identificar os pontos onde seu valor é zero e no caso de funções bivariadas, esses são os pontos (x, y) tais que $f(x, y) = 0$ que, em termos geométricos, é uma curva de nível, uma aplicação para esse problema é, por exemplo, resolver sistemas de equações.

O comando para encontrar os zeros da função $f(x, y)$ é `roots(f)`. Na Figura 16, os pontos pretos são os valores tais que $f(x, y) = g(x, y) = 0$, eles podem ser facilmente encontrados fazendo `roots(f, g)`.

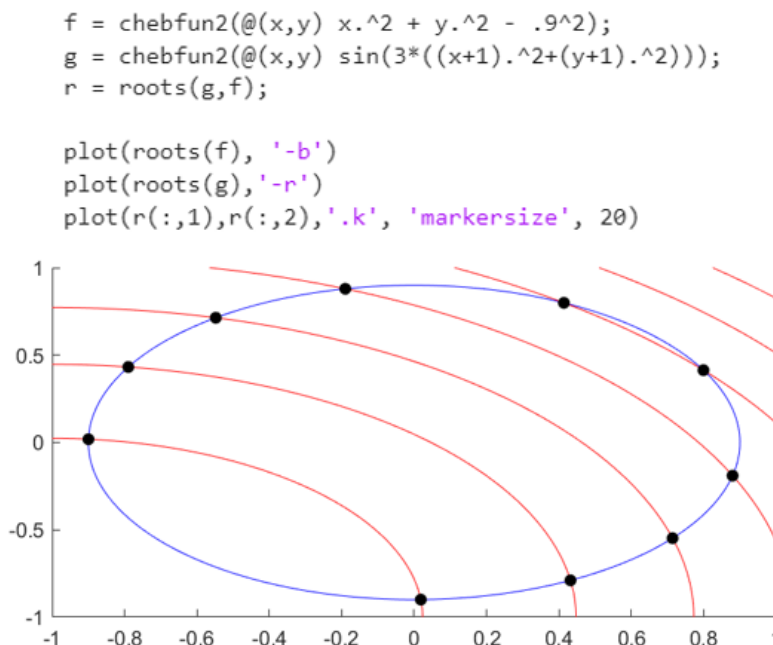


Figura 16: Pontos de interseção de duas funções usando "roots".

3.3 Chebfun2v e Funções Vetoriais.

Uma extensão para o Chebfun2 é o Chebfun2v que permite o usuário trabalhar com funções vetoriais e fazer análises de campos vetoriais. O Chebfun2v possui um ferramental para lidar com problemas vetoriais como funções para o cálculo do divergente e o rotacional.

Uma convenção adotada é o uso de letras maiúsculas para representar funções vetoriais, logo, $F(x, y)$ é uma função vetorial ou um chebfun2v, que é definido duas formas que estão representadas na Figura 17. A primeira consiste em criar a função toda de uma vez e a segunda, criar cada argumento individualmente e juntar depois, ambas resultam no mesmo chebfun2v.

Algumas operações com esse objeto é o produto vetorial dado por `cross(F, G)`, que retorna $F(1)G(2) - F(2)G(1)$. Se F e G possuem duas componentes e o produto vetorial usual no caso de possuir três componentes. Outra importante função é o produto interno `dot(F, G)`, o resultado é um escalar e se o resultado for zero significa que as funções são ortogonais ou uma delas é a função nula. Na Figura 18, a curva mostrada possui os pontos tais que as duas funções vetoriais são ortogonais pois o produto interno é zero. O Chebfun2v também trata dos operadores diferenciais para funções vetoriais, os escalares são o gradiente e o divergente, enquanto os vetoriais são o rotacional e o laplaciano, cada um deles tem a sua importância e aplicabilidade, todas levando como argumento o chebfun2v.

O gradiente pode ser obtido fazendo `gradient(f)`, ele representa a direção e a magnitude que f cresce mais rápido, além de que se seu valor for zero em um ponto, significa que

```

F = chebfun2v(@(x,y) sin(x.*y), @(x,y) cos(y), [-1 1 -1 1]);
f = chebfun2(@(x,y) sin(x.*y), [-1 1 -1 1]);
g = chebfun2(@(x,y) cos(y), [-1 1 -1 1]);
G = [f;g];
chebfun2v object (Column vector) containing:

    chebfun2 object
    domain                rank        corner values
    [-1, 1] x [-1, 1]      6        [0.84 -0.84 -0.84 0.84]
    vertical scale = 0.84

    chebfun2 object
    domain                rank        corner values
    [-1, 1] x [-1, 1]      1        [0.54 0.54 0.54 0.54]
    vertical scale = 1

```

Figura 17: Sintaxe para a criação de um objeto chebfun2v.

```

F = chebfun2v(@(x,y) sin(x.*y), @(x,y) cos(y), [0 2 0 2]);
G = chebfun2v(@(x,y) cos(4*x.*y), @(x,y) -1+ 2*x + x.*y.^2, [0 2 0 2]);
plot(roots(dot(F,G))), axis equal, axis([0 2 0 2])

```

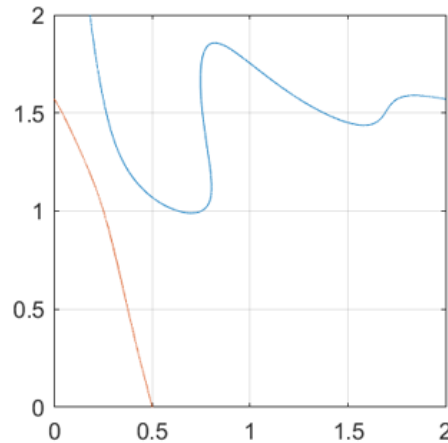


Figura 18: Os pontos das curvas são valores que tornam duas funções ortogonais.

esse é um ponto crítico. Na seção de aplicações, os pontos críticos de uma função serão obtidos usando o gradiente.

O divergente é uma função escalar que representa a distribuição de fontes e poços no campo vetorial, seu comando é `divergence(F)` e é a soma das derivas parciais em relação a cada componente.

O rotacional é uma função vetorial que mede a vorticidade no campo vetorial, seu comando é `curl(F)`.

É possível realizar cálculos de integrais usando esse tipo de função, para isso é preciso definir a curva C que a integral será realizada e depois aplicar o comando `integral(F, C)`. Usando o comando `quiver(F)`, é possível desenhar o campo vetorial de uma função vetorial como mostrado na Figura 19.

Embora o Chebfun2 não tenha sido planejado para esse intuito, é possível representar superfícies tridimensionais, por exemplo, é possível representar com o uso de coordenadas esféricas uma esfera em \mathbb{R}^3 como apresentado na Figura 20. Uma coisa muito útil ao trabalhar com superfícies, é encontrar seus vetores normais e unitários, obtidos usando o comando `normal(F, 'unit')` e podem ser observados usando o `quiver3` na Figura 21, onde foi usada a mesma esfera criada na Figura 20. Uma utilidade desses vetores normais é aplicar o teorema do divergente e assim, obter o volume do toro porque não é possível

```

F = chebfun2v(@(x,y) sin(x.*y), @(x,y) cos(y), [-2 2 -2 2]);
G = chebfun2v(@(x,y) cos(x.*y), @(x,y) 2*x + x.*y.^2, [-2 2 -2 2]);
quiver(F,'b'), axis equal, hold on
quiver(G,'r')

```

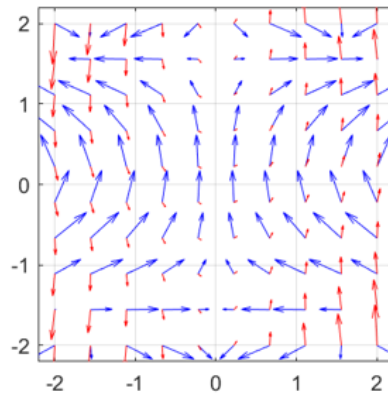


Figura 19: Campo vetorial de uma função vetorial usando "quiver".

```

th = chebfun2(@(th,phi) th, [0 pi 0 2*pi]);
phi = chebfun2(@(th,phi) phi, [0 pi 0 2*pi]);
x = sin(th).*cos(phi);
y = sin(th).*sin(phi);
z = cos(th);

F = [x;y;z];
surf(F), colormap('default'), camlight, axis equal

```

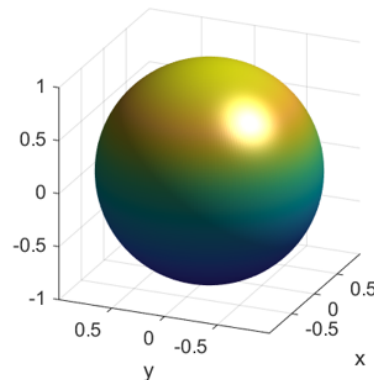


Figura 20: Código para gerar uma superfície em 3D usando o Chebfun2.

integrar sobre o volume 3D usando o Chebfun2 mas é possível integrar sobre uma superfície 2D e o processo do cálculo está mostrado na Figura 22.

Para finalizar essa seção, é importante falar das limitações do Chebfun2 e Chebfun2v, objetos que são tridimensionais vem com uma *flag* de aviso pois o chebfun2 funciona com funções bivariada e não trivariadas, na Figura 21, foi possível calcular o volume do toro por conta do teorema do divergente que permitiu pular o obstáculo que seria a integral tripla e essa é uma limitação do Chebfun2, o que levou os pesquisadores e criadores do pacote a pensar numa forma de resolver problemas assim chegando ao Chebfun3.

```

F = [x;y;z];
quiver3(F(1),F(2),F(3),normal(F,'unit'),'numpts',10)
axis equal, hold on
surf(F), axis equal

```

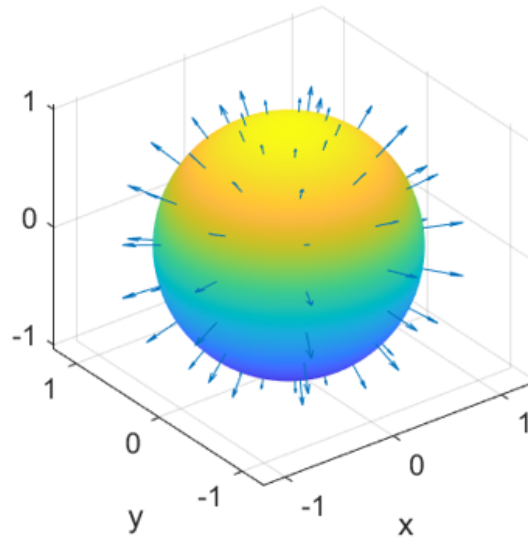


Figura 21: Vetores normais da esfera usando o método "normal".

```

r1 = 1; r2 = 1/8;
d = [0 2*pi 0 2*pi];
u = chebfun2(@(u,v) u,d);
v = chebfun2(@(u,v) v,d);
F = [-(r1+r2*cos(v)).*sin(u); (r1+r2*cos(v)).*cos(u); r2*sin(v)];
surf(F)
vol = integral2(dot(F./3,normal(F)));
exact = 2*pi^2*r1*r2.^2;
dif = vol - exact;
% ans = 5.329070518200751e-15

```

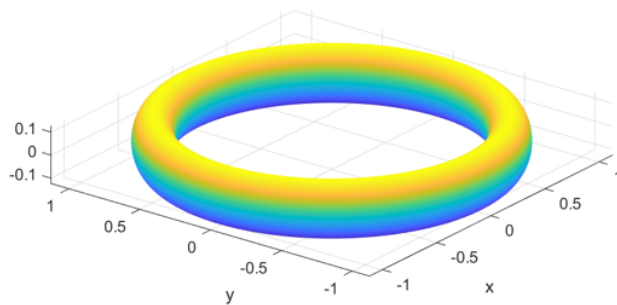


Figura 22: Cálculo do volume do toro usando o teorema do divergente.

4 Chebfun3 e funções de três variáveis.

O Chebfun3 surgiu com o mesmo intuito do Chebfun2, resolver as limitações que o seu antecessor possui, por isso, em 2017, foi adicionado ao pacote Chebfun, a primeira versão do Chebfun3 que possui alguns métodos para lidar com funções de 3 variáveis. Esse pacote ainda hoje se encontra em desenvolvimento por conta da dificuldade de encontrar a melhor forma de representar essas funções computacionalmente. No momento, a forma de representar esse objeto, é usando o formato *tucker* que é um produto em três dimensões envolvendo três *quasimatrices*, que são matrizes contendo em cada coluna (ou linha), um chebfun.

Por conta de ainda estar sendo desenvolvido, não é possível afirmar que os comandos mostrados aqui serão o que participarão da versão final, porém, é possível mostrar como anda esse projeto. Mais informações podem ser obtidas em [12].

A sintaxe dos métodos é a mesma que o Chebfun2 e alguns deles são:

- `max3` para máximos.
- `min3` para mínimos.
- `sum3`, `sum2` (que retorna um chebfun) e `sum` (que retorna um chebfun2) para integrais.
- `diffx`, `diffy` e `diffz` para derivadas.
- `Chebfun3v` para lidar com funções vetoriais.
- `roots` para encontrar raízes

Dos métodos acima, o de encontrar o zero é interessante pois o processo atual só lida com objetos do tipo $F = [f, g, h]$ e daí calcula uma chute inicial a partir de $f^2 + g^2 + h^2 = 0$ e daí usa o método de Newton para melhorar a precisão do valor encontrado, não é muito eficiente e por isso pesquisas estão sendo feitas para encontrar um método melhor. A figura 23 mostra a raiz calculada e colocada de forma visual.

A próxima seção trata sobre algumas aplicações usando o Chebfun.

```

f = chebfun3(@(x,y,z) y-x.^2);
g = chebfun3(@(x,y,z) z-x.^3);
h = chebfun3(@(x,y,z) cos(exp(x.*sin(-2+y+z))));
isosurface(f, 0, 'g')
isosurface(g, 0, 'b')
isosurface(h, 0, 'r')
r = root(f, g, h);

plot3(r(1), r(2), r(3), 'yh', 'markersize', 50, 'LineWidth', 20)

```

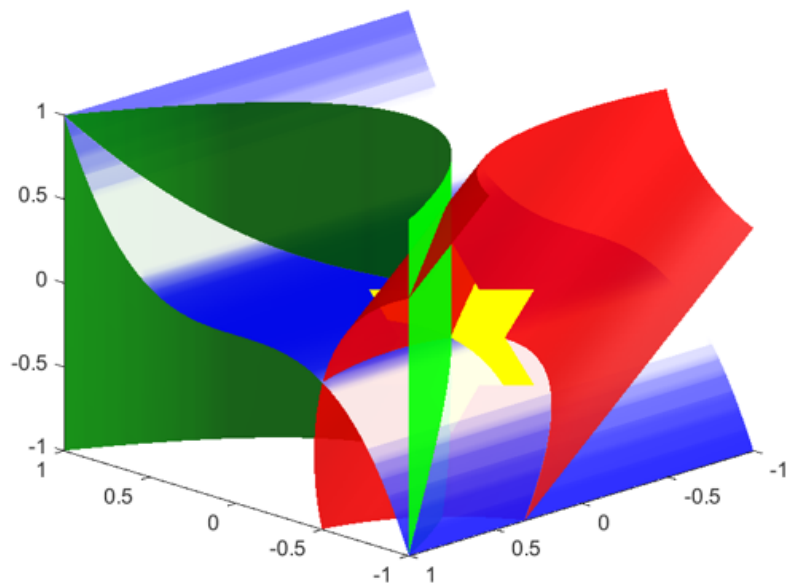


Figura 23: Gráfico da interseção de três superfícies e a estrela amarela é a interseção.

5 Aplicações do Chebfun2.

Para ilustrar a utilidade do Chebfun2, serão demonstradas algumas de suas aplicações.

5.1 A função de Dixon-Szego.

Esta aplicação consiste em otimizar a função de Dixon-Szego, dada por:

$$f(x, y) = \left(4 - 2.1x^2 + \frac{x^4}{3}\right)x^2 + xy + 4(y^2 - 1)y^2$$

Mostrando primeiro como foi feita a otimização usando o Chebfun e depois usando o Chebfun2.

A otimização será feita no domínio retangular dado por $[-2.5, 2.5] \times [-1.5, 1.5]$, a função pode ser representada através de curvas de nível para facilitar o entendimento. Infelizmente, o Chebfun não é capaz de obter vários dos mínimos por conta da simetria do problema mas identifica um deles. A forma para resolver esse problema é minimizar em um dos eixos e depois, minimizar a função no outro eixo. A seguir, na Figura 24, segue o código e as curvas com o ponto mínimo, que é obtido em $(x, y) = (0.0898, -0.7127)$

```
% APLICAÇÃO 1 - OTIMIZAÇÃO DA FUNÇÃO DIXON-SZEGO
% Definir a função como função anônima
f = @(x,y) (4-2.1*x.^2 + (1/3)*x.^4).*x.^2 + x.*y + ...
4*(y.^2-1).*y.^2;

% Criar as curvas de nível
X = linspace(-2.5,2.5,100); Y = linspace(-1.5,1.5,100);
[XX,YY] = meshgrid(X,Y);
ff = f(XX,YY);
contour(X,Y,ff,150); colorbar; hold on

% Definir a função que vai ser minimizada como um chebfun
fmin_x0 = @(x0) min(chebfun(@(y) f(x0, y), [-1.5, 1.5]));
fmin_x = chebfun(fmin_x0, [-2.5, 2.5], 'vectorize','splitting','on');

[minx_func, minx_value] = min(fmin_x);
[miny_func, miny_value] = min(chebfun(@(y) f(minx_value,y)));

plot(minx_value,miny_value,'.r','MarkerSize',20);

% minx_func = -1.0316
% miny_func = -1.0316
% minx_value = 0.0898
% miny_value = -0.7127
```

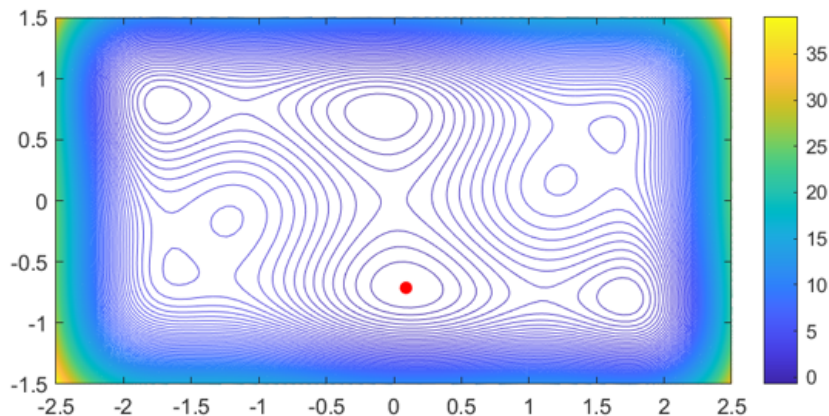


Figura 24: Otimização da função de Dixon-Szego.

Agora, usando o Chebfun2, esse código fica como na Figura 25.

```
f = @(x,y) (4-2.1*x.^2+ x.^4/3).*x.^2 + x.*y + 4*(y.^2-1).*y.^2;  
  
% Cria o chebfun2  
F = chebfun2(f,[-2,2,-1.25,1.25]);  
[min_val, min_pos] = min2(F);  
% min_val = -1.031628453489878  
% min_pos = (0.089842010734102; -0.712656405973724)  
  
contour(F,30,'linewidth',1.2), colorbar, axis([-2 2 -1.25 1.25])  
hold on, plot(minx(1),minx(2),'.k','MarkerSize',20)
```

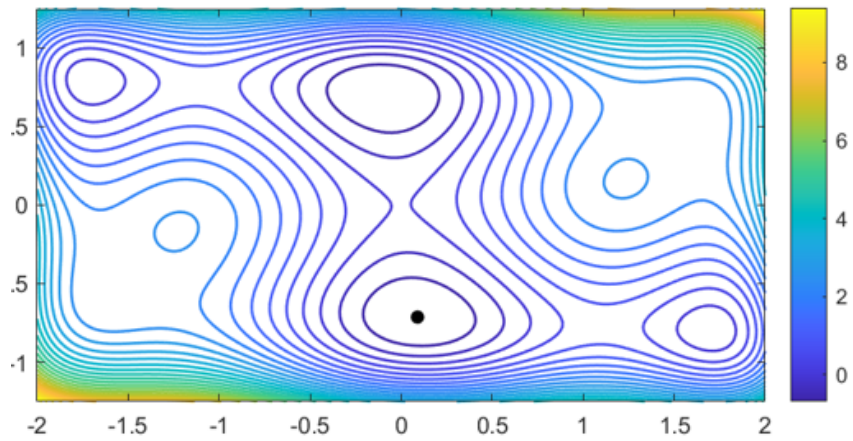


Figura 25: Otimização da função de Dixon-Szegő usando Chebfun2.

5.2 Interseção de Curvas no Plano Complexo.

Para encontrar a interseção de duas curvas C_1 e C_2 , primeiro, defina as duas curvas, defina uma função f de forma que $f(s, t) = C_1(t) - C_2(s)$ e depois compute a interseção da parte real com a parte imaginária de f , ou seja, computar os valores de s e t tais que $Re(f) = Im(f) = 0$.

Isso funciona porque sendo (t, s) um ponto que f vale zero, então:

$$0 = Re(f) \Rightarrow 0 = Re(C_1(t) - C_2(s)) \Rightarrow Re(C_1(t)) = Re(C_2(s))$$

e analogamente para a parte imaginária

$$0 = Im(f) \Rightarrow 0 = Im(C_1(t) - C_2(s)) \Rightarrow Im(C_1(t)) = Im(C_2(s)),$$

somando essas duas equações, o resultado é

$Re(C_1(t)) + Im(C_1(t)) = Re(C_2(s)) + Im(C_2(s)) \Rightarrow C_1(t) = C_2(s)$ O código para o processo de interseção de curvas está na Figura 26.

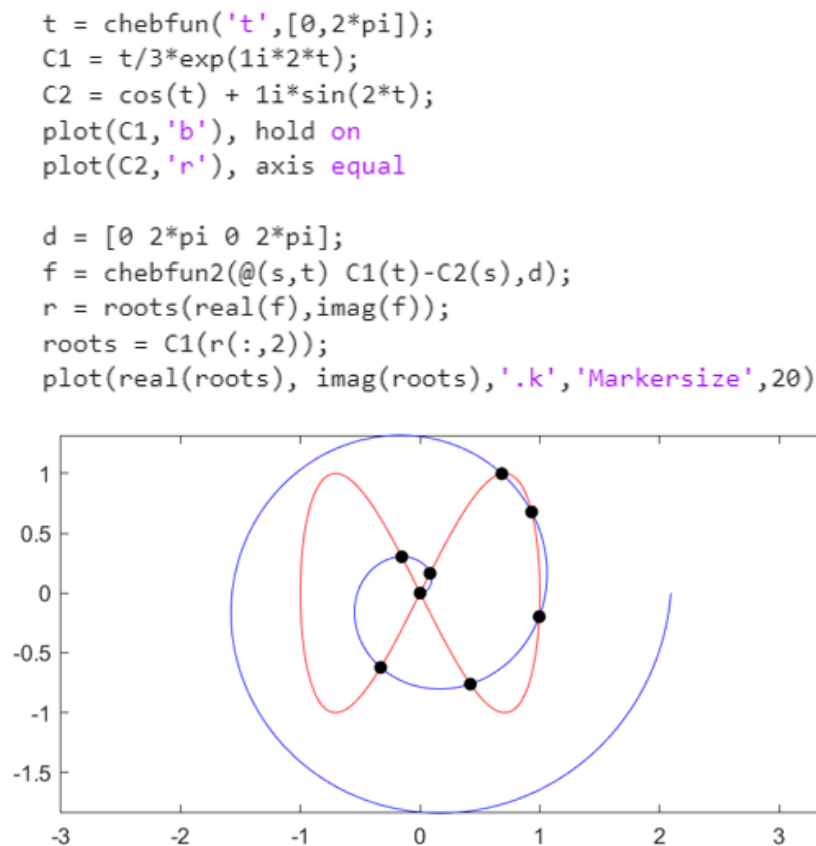


Figura 26: Código para computar a interseção de duas curvas no plano complexo.

5.3 Identificação de pontos críticos.

Identificar pontos críticos basta encontrar os pontos da função que seu gradiente vale zero. Um exemplo desse código está na Figura 27.

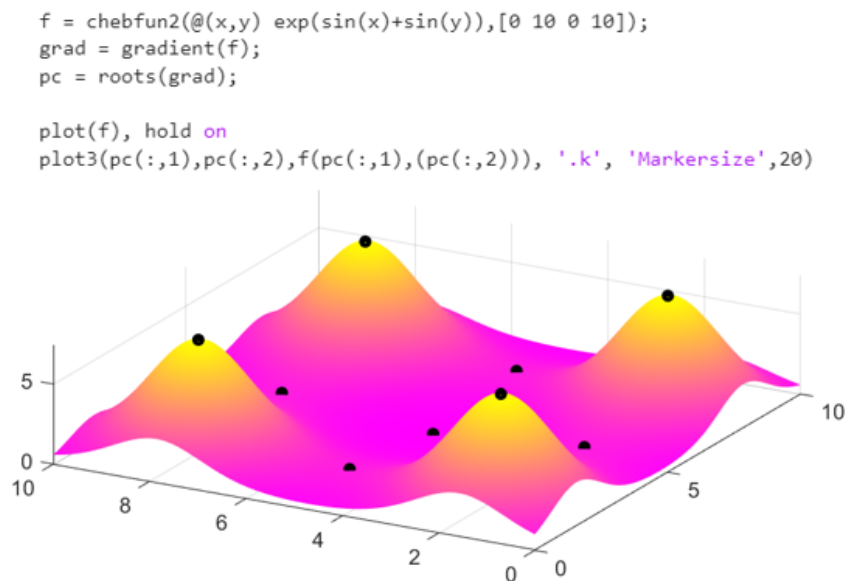


Figura 27: Código para encontrar pontos críticos usando o método "gradient".

6 Conclusão

O projeto como um todo explorou de maneira aprofundada o uso de polinômios ortogonais em análise numérica, com foco nos polinômios de Chebyshev e suas extensões para funções bivariadas e trivariadas utilizando o pacote Chebfun no MATLAB. A implementação computacional dessas técnicas permitiu investigar aspectos teóricos e práticos que fundamentam o comportamento e a eficiência desses métodos.

A partir do Chebfun, foi possível compreender como aproximações baseadas na interpolação baricêntrica e nos nós de Chebyshev são utilizadas para representar funções univariadas com alta precisão. Contudo, as limitações desse modelo unidimensional motivaram o estudo do Chebfun2 e Chebfun3, que expandem essas ideias para funções em duas e três variáveis. Em particular, foi destacado o papel da decomposição de baixo rank e o uso da eliminação gaussiana com pivoteamento completo como métodos para representar funções multidimensionais, garantindo uma aproximação eficiente e reduzindo a complexidade computacional.

Do ponto de vista prático, o Chebfun2 mostrou-se uma ferramenta robusta para realizar operações fundamentais, como diferenciação, integração e localização de zeros, além de permitir trabalhar com funções vetoriais e resolver inúmeros problemas. A capacidade de manipular funções bivariadas de forma tão direta e precisa evidencia a sofisticação e a aplicabilidade do método. Além disso, o Chebfun3, ainda em desenvolvimento, permite estender esses conceitos para funções trivariadas, explorando representações em termos de tensores e outras técnicas avançadas.

Esse trabalho também demonstrou as limitações das abordagens computacionais, como a dificuldade em lidar com funções que possuem singularidades ou descontinuidades, e ressaltou a importância de compreender os fundamentos teóricos para superar tais desafios. A integração de ferramentas computacionais como o Chebfun possibilitou uma análise mais prática e detalhada, abrindo portas para futuras pesquisas na área de análise numérica e matemática computacional.

Embora seja um pacote notável, ele não atingiu uma ampla audiência. Isso pode ser atribuído ao fato de ser um pacote de nicho, voltado para usuários de áreas especializadas como engenharia e matemática, embora não exclua o uso por outras áreas, e também por ser projetado especificamente para o MATLAB, um software pago com custo considerável, o que é inviável para muitas pessoas, especialmente estudantes universitários que, em geral, dependem de bolsas ou auxílios financeiros.

Por fim, os resultados obtidos não apenas reafirmam a importância dos polinômios ortogonais na análise numérica, mas também ressaltam o papel fundamental de ferramentas computacionais modernas na exploração e aplicação desses conceitos. O projeto deixa como legado uma base sólida de conhecimento, que poderá ser expandida em trabalhos futuros e aplicada em diversas áreas da matemática e computação científica.

7 Referências.

1. *A. Townsend and L. N. Trefethen*, **An extension of Chebfun to two dimensions**, SIAM Journal on Scientific Computing, 2013.
2. *L.N. Trefethen*, **Approximation Theory and Approximation Practice**, SIAM, 2018.
3. *A. Greenbaum & T.P. Chartier*, **Numerical Methods**, Princeton, 2012.
4. *T. A. Driscoll, N. Hale, and L. N. Trefethen*, **Chebfun Guide**, Pafnuty Publications, Oxford, 2014.
5. Package Chebfun, <https://www.chebfun.org>, 2022.
6. *N. L. Carothers*, **A short course on approximation theory**. Bowling Green State University.
7. *M.H. Mudde*, **Chebyshev approximation**. University of Groningen, 2017.
8. *Battles & Trefethen 2004*, **An extension of MATLAB to continuous functions and operators**, SIAM Journal on Scientific Computing, 2004.
9. *T. A. Driscoll, F. Bornemann, and L. N. Trefethen*, **The chebop system for automatic solution of differential equations**, BIT Numerical Mathematics, 2008.
10. *Austin, Kravanja & Trefethen*, https://people.maths.ox.ac.uk/trefethen/austin_kravanja_trefethen_revised.pdf, 2013.
11. *A. Townsend and L. N. Trefethen*, **Gaussian elimination as an iterative algorithm**, SIAM News, March 2013.
12. *B. Hashemi and L. N. Trefethen*, **Chebfun in three dimensions**. SIAM J. Sci. Comput. 2017.