Pontifícia Universidade Católica
do Rio de Janeiro

Eduardo Teles da Silva

wx2x2 - um software para sistemas não lineares

Dissertação de Mestrado

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Programa de Pós–graduação em Matemática Aplicada do Departamento de Matemática da PUC–Rio

Orientador : Prof. Carlos Tomei
Co–Orientador: Prof. Humberto J. Bortolossi

Rio de Janeiro
Fevereiro de 2007

Pontifícia Universidade Católica
do Rio de Janeiro

Eduardo Teles da Silva

wx2x2 - um software para sistemas não lineares

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Programa de Pós–graduação em Matemática Aplicada do Departamento de Matemática do Centro Técnico Científico da PUC–Rio. Aprovada pela Comissão Examinadora abaixo assinada.

Prof. Carlos Tomei
Orientador
Departamento de Matemática — PUC–Rio

Prof. Humberto J. Bortolossi
Co–Orientador
Departamento de Matemática — UFF

Prof. Nicolau C. Saldanha
Departamento de Matemática — PUC–Rio

Prof. Marcelo Gattass
Departamento de Informática — PUC–Rio

Prof. Dan Marchesin
IMPA

Prof. José Eugenio Leal
Coordenador Setorial do Centro Técnico Científico — PUC–Rio

Rio de Janeiro, 28 de Fevereiro de 2007

Eduardo Teles da Silva

Mestrado: Matemática Aplicada — Pontifícia Universidade Católica do Rio de Janeiro – PUC–Rio (2005–2007).
Graduação: Licenciatura em Matemática — Universidade Estadual de Santa Cruz – UESC (2001–2004).

Ficha Catalográfica

Teles, Eduardo

wx2x2 - um software para sistemas não lineares / Eduardo Teles da Silva; orientador: Carlos Tomei; co–orientador: Humberto J. Bortolossi. — Rio de Janeiro : PUC–Rio, Departamento de Matemática, 2007.

v., 54 f: il. ; 29,7 cm

1. Dissertação (mestrado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Matemática.

Inclui referências bibliográficas.

1. Matemática – Tese. 2. Sistemas não lineares. 3. Singularidade. 4. Dobra. 5. Cúspide. 6. Análise Numérica. I. Tomei, Carlos. II. Bortolossi, Humberto J.. III. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Matemática. IV. Título.

CDD: 510

Aos meus pais, irmãos e amigos.

Agradecimentos

Primeiramente agradeço a Deus por ter me concedido saúde e ânimo nesses dois últimos anos.

Aos meus pais e irmãos pelo amor e esperança em mim depositados.

Ao meu orientador, Carlos Tomei, pela paciência, empenho, esclarecimentos e alegria demonstrados em todos os momentos durante a realização deste trabalho.

Ao meu co-orientador e amigo, Humberto J. Bortolossi, pelos conselhos e ampla dedicação demonstrada (mesmo durante vários fins de semana).

Aos grandes e velhos amigos que nunca me abandonaram apesar de estarem tão distantes.

A todos os funcionários e professores do Departamento de Matemática pela ajuda prestada. Em especial, a Creuza que sempre manteve a burocracia imperceptível.

À Capes e à PUC-Rio, pelos auxílios concedidos.

Resumo

Teles, Eduardo; Tomei, Carlos; Bortolossi, Humberto J.. **wx2x2 - um software para sistemas não lineares**. Rio de Janeiro, 2007. 54p. Dissertação de Mestrado — Departamento de Matemática, Pontifícia Universidade Católica do Rio de Janeiro.

Apresentamos um software para inverter funções suaves genéricas do plano no plano, $F(x) = b$, bem como a teoria utilizada para implementá-lo. Em princípio, o programa calcula todas as pré-imagens de um ponto. A inversão numérica baseia-se na caracterização do conjunto crítico $\mathcal{C} = \{x \in \mathbb{R}^2 : \det DF(x) = 0\}$ e sua imagem, e em técnicas de continuação numérica ajustadas para interação controlada com $\mathcal{C}$. A interface gráfica permite o estudo de propriedades geométricas e analíticas, tanto locais quanto globais.

Palavras–chave

Sistemas não lineares. Singularidade. Dobra. Cúspide. Análise Numérica.

Abstract

Teles, Eduardo; Tomei, Carlos; Bortolossi, Humberto J.. **wx2x2 - a software for nonlinear systems**. Rio de Janeiro, 2007. 54p. MsC Thesis — Departament of Mathematics, Pontifícia Universidade Católica do Rio de Janeiro.

We present a software to invert functions from the plane to the plane $F(x) = b$, for a generic smooth function F, as well as the theory to implement it. In principle, all points in the preimage of b are computed. The numerical inversion is based on the characterization of the critical set $\mathcal{C} = \{x \in \mathbb{R}^2 : \det DF(x) = 0\}$ and its image, and in appropriate techniques of numerical continuation in situations of controlled interaction with $\mathcal{C}$. A graphical user interface allows for the study of local and global properties of the function, both of geometric and analytic nature.

Keywords

Nonlinear Systems. Singularity. Fold. Cusp. Numerical Analysis.

Sumário

Lista de figuras

1
Introdução

O **wx2x2** é um software para resolver sistemas não lineares a duas equações e duas incógnitas, ou, mais geometricamente, para inverter pontos na imagem de uma função do plano para o plano, para uma grande classe de funções. Seu ancestral direto é o **2x2**, desenvolvido pelos autores de [MST1]. O novo software tem portabilidade, melhor interface gráfica e maior robustez numérica. A interface gráfica emprega a consagrada biblioteca multiplataforma *wxWidgets*. Um *parser* matemático é utilizado para avaliar as funções. A atual versão do **wx2x2** possui cerca de dezesseis mil linhas de código em C/C++ e quatro mil linhas de comentário.

O presente trabalho surgiu da constatação que o **2x2** estava longe de ser *user friendly*. E as dificuldades aumentavam quando se tentava avançar em sua programação: a documentação era praticamente inexistente, e os interessados contavam com os artigos ([MST1], [MST2]) que descreviam de forma incompleta tanto os aspectos teóricos que subsidiam o algoritmo, quanto suas implementações, muitas vezes sutis.

À medida que aumentava a familiaridade com o programa, várias atividades foram desenvolvidas em paralelo. O código-fonte do programa, junto com executáveis para Windows e Linux, encontram-se disponíveis na homepage http://www.mat.puc-rio.br/∼hjbortol/2x2. Boa parte do código-fonte agora é comentada. A interface gráfica tornou possível experimentos mais abundantes e rigorosos, tornando o software mais robusto. Por ser um programa de análise numérica não linear, existem aspectos específicos, como as estratégias de passo locais e a escolha de certos parâmetros globais, que exigem um tratamento artesanal. Parte da extensão do programa deve-se a rotinas que são invocadas apenas em momentos especiais e raros, mas que são variados. Para uma visão global da árvore de rotinas do programa, o leitor pode consultar o Apêndice.

No Capítulo 2, o leitor é apresentado a um exemplo simples, mas expressivo, no qual se mostram os recursos numéricos e visuais do **wx2x2**. No Capítulo 3, descrevem-se os resultados teóricos fundamentais, que podem ser inesperados para o leitor sem prática de análise não linear. O material é uma combinação de teoria local (certas extensões do teorema de função inversa

habitual à teoria de singularidades de Whitney ([W])) e teoria global (grau e outros invariantes topológicos de curvas). O capítulo seguinte é um manual do usuário. No último capítulo, finalmente, a implementação de algumas rotinas mais importantes é descrita em detalhe.

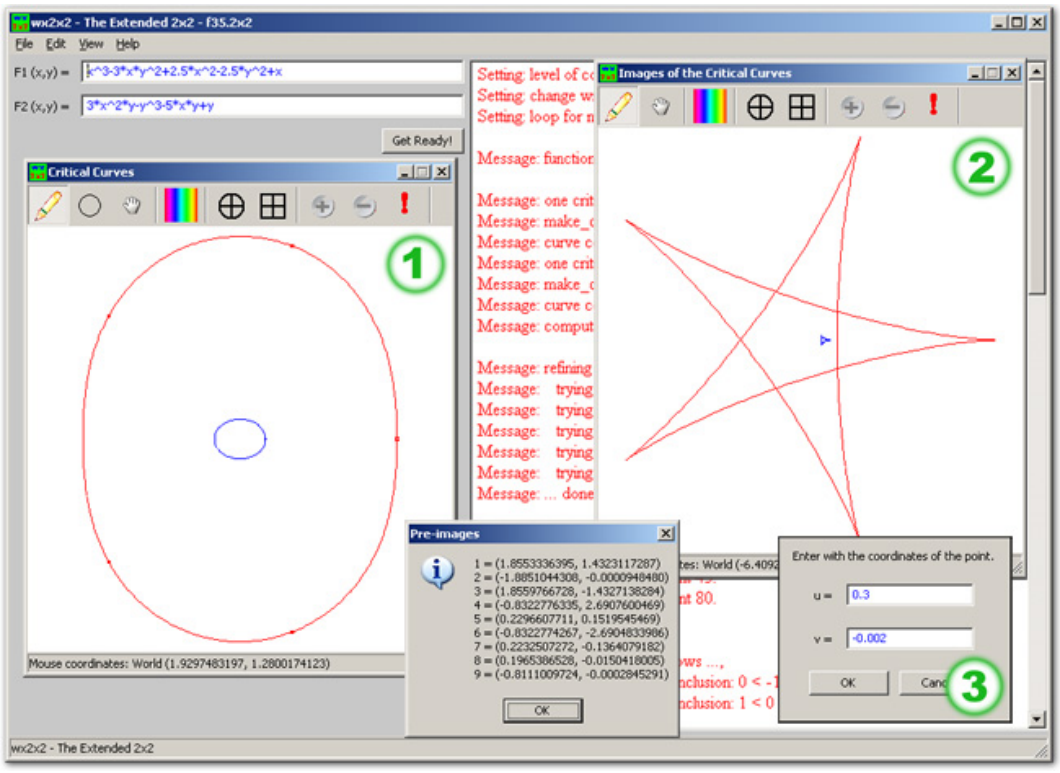
O programa começou a partir de indagações de Dan Marchesin (IMPA) associadas à velocidade de propagação de choques nos chamados problemas de Riemann, no final dos anos oitenta. Naquela época, o Departamento de Informática generosamente ofereceu o material computacional empregado por Saldanha e Tomei para a confecção do **2x2** original. Pequenos progressos acompanharam o interesse de Marchesin e, mais recentemente, de Marcelo Gattass (Depto. Informática, PUC-Rio). As possibilidades de extensão são inúmeras, tanto para contextos mais gerais no plano, quanto para funções em mais dimensões.

2
Um primeiro exemplo

Para uma matriz A de dimensão 2, o sistema linear $Ax = b$ é banal, e a solução admite até uma fórmula explícita. Para sistemas não lineares, estamos longe de métodos tão simples. Por exemplo, como resolver o sistema de equações (2-1) abaixo?

$$\begin{aligned} F_1(x,y) &= x^3 - 3xy^2 + 2.5x^2 - 2.5y^2 + x = 0.3 \\ F_2(x,y) &= 3x^2y - y^3 - 5xy + y = -0.002 \end{aligned} \tag{2-1}$$

Vamos usar o **wx2x2** para resolver o sistema (2-1). Conforme mostrado na Figura 2, basta fornecer as coordenadas F_1 e F_2 da função $F(x,y)$.



Após clicar no botão "Get Ready!", abrem-se as janelas 1 e 2 e o programa está apto a calcular as pré-imagens de qualquer ponto! Na janela à direita (janela 2) teclamos "i" e informamos, na janela de diálogo 3, o ponto que desejamos inverter, neste caso, o ponto $(0.300, -0.002)$. O programa retornará uma lista de soluções aproximadas — e, de fato, não há outras.

O que significam as curvas exibidas nas janelas 1 e 2? A *janela do domínio* 1 exibe o conjunto crítico $\mathcal{C}$ da função (isto é, os pontos onde a Jacobiana da função não é inversível). Já na *janela do contradomínio* 2 está sua imagem $F(\mathcal{C})$. Sob hipóteses flexíveis, o **wx2x2** calcula o número de pré-imagens por F de qualquer ponto a partir de $\mathcal{C}$ e $F(\mathcal{C})$.

O **wx2x2** possui outros recursos para o estudo de funções não lineares. O programa pode, por exemplo, gerar a *flor* $\mathcal{F} = F^{-1}(F(\mathcal{C}))$ da função F. A partir da flor, dispõe-se de uma idéia satisfatória do comportamento global da função, muito parecida com aquela que se tem de certas funções holomorfas quando se estudam mapeamentos conformes.

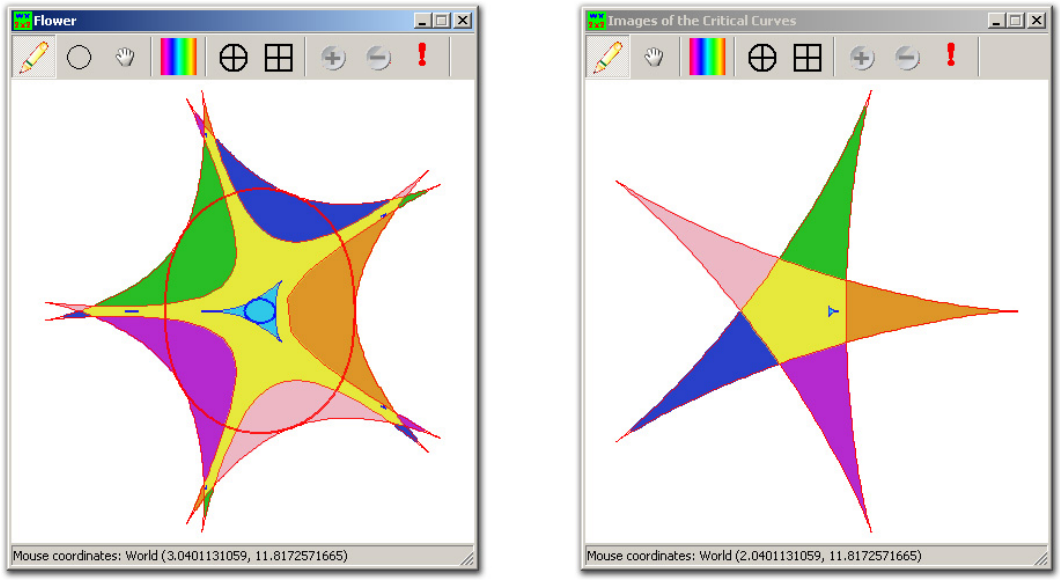


Figura 2.1: A flor de F e sua imagem

Na Figura 2.1 é exibida a flor $\mathcal{F}$ da função F e sua imagem $F(\mathcal{C})$. Tanto o domínio quanto o contradomínio partem-se em *placas*, as componentes conexas do complemento de $\mathcal{F}$ no domínio e $F(\mathcal{C})$ no contradomínio. Placas de mesma cor no domínio são levadas à placa com a mesma cor na imagem. A placa ilimitada do domínio (branca) recobre 3 vezes a placa ilimitada (branca) de $\mathbb{R}^2 - F(\mathcal{C})$. A placa simplesmente conexa envolvida pela curva crítica mais interna (em ciano) é levada difeomorficamente à sua imagem. Todos os pontos da placa amarela da imagem têm 7 pré-imagens. Duas delas estão na placa amarela do domínio, situada entre as duas curvas críticas. As outras cinco estão nas cinco pétalas amarelas no anel exterior ao conjunto crítico.

De forma um pouco mais geral, a figura já indica o número de pré-imagens de pontos em cada componente de $\mathbb{R}^2 - F(\mathcal{C})$: o número é constante em cada componente. Para a componente branca, ele é 3 — isso pode ser conferido notando que a função F, em notação complexa, se escreve $\widetilde{F}(z) = z^3 + 2.5\bar{z}^2 + z$.

Para as componentes formando as cinco pontas da estrela na imagem, o número de pré-imagens é 5. Para a região amarela, 7 e para o pequeno triângulo ciano, 9. É fácil identificar na figura as nove pré-imagens do triângulo: quatro estão no centro da janela do domínio e as outras cinco espalham-se uma em cada pétala.

Vamos ver outra figura obtida facilmente pelo **wx2x2**. Usando a ferramenta *Pencil*, o usuário desenha curvas na janela do contradomínio que são invertidas interativamente. As pré-imagens são mostradas tanto na janela do domínio, quanto na janela da flor.

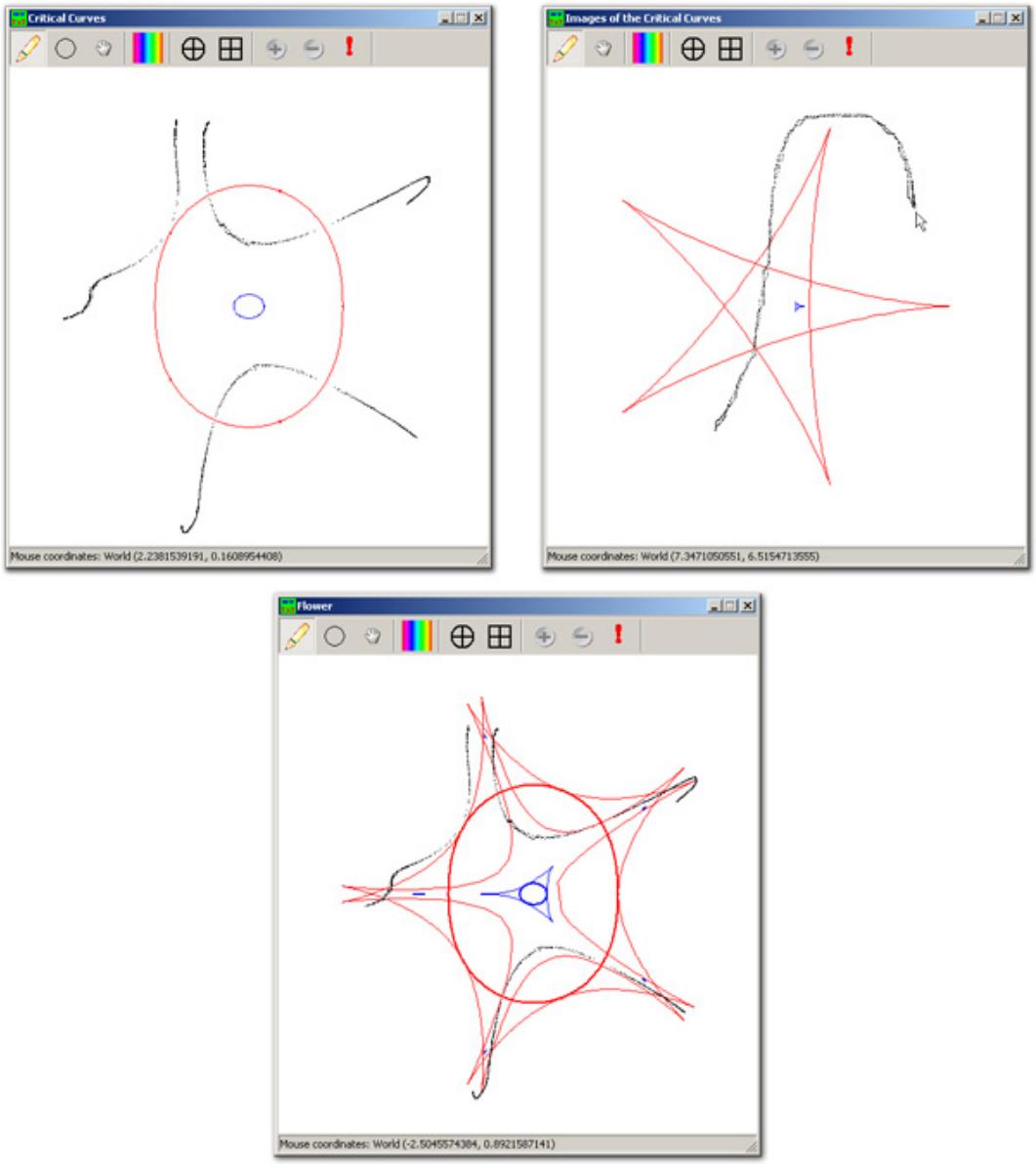


Figura 2.2: Invertendo curvas

Para cada tarefa executada pelo **wx2x2**, mensagens são exibidas na *área de log*, situada do lado direito na área de trabalho do programa. O nível

de detalhamento dessas mensagens pode ser especificado na opção *Preferences* situada no menu *Edit*. Para a função F, com verbosidade máxima, a área de log apresentada na Figura 2.3 fornece as seguintes informações a respeito de F. No bloco (A) lemos que o grau topológico de F é 3. O bloco (B) nos diz que duas curvas críticas foram criadas, uma, a curva rotulada por 0, passando por $(-1.40, -0.50)$ e contendo 423 pontos e outra, a curva 1, por $(-0.20, -0.09)$, contendo 100 pontos. Nos blocos (C), (D) e (F) estão listadas informações a respeito de certos pontos especiais nas curvas críticas — as cúspides, definidas no capítulo seguinte. O bloco (E) informa sobre a posição relativa das placas no domínio. A placa ilimitada é designada por -1: a expressão $0 < -1$ informa que a curva crítica 0 é fronteira da placa ilimitada, já a expressão $1 < 0$ diz que a curva 1 está contida no interior do disco delimitado pela curva 0. Na Figura 2.4, a área de log informa que a inversão do ponto $(-0.02791327, 0.04723276)$ foi realizada com êxito e exibe todas suas nove pré-imagens. Os detalhes serão esclarecidos no último capítulo (seção 5.4).

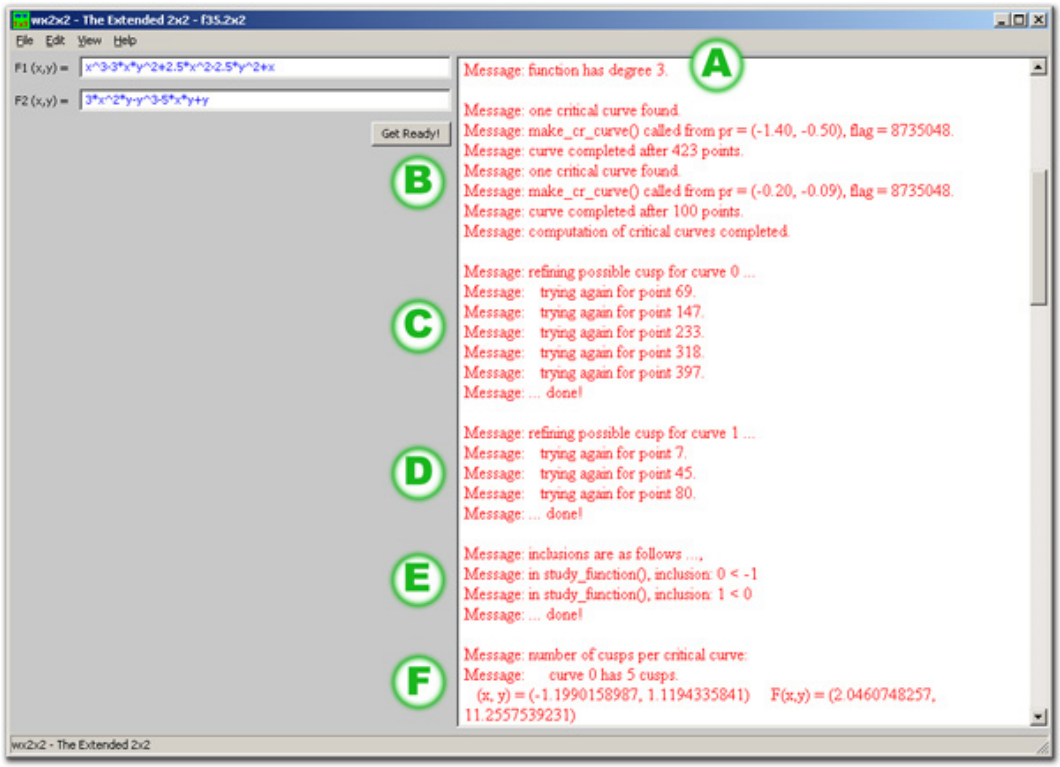


Figura 2.3: A área de log

Para resolver um sistema não linear geral, poderíamos escolher alguns pontos como condições iniciais para o método de Newton, mas não teríamos a certeza de obter todas as soluções do sistema. A iteração convergiria para algumas condições iniciais, divergiria para outras, sem assegurar a identificação do conjunto total de soluções. Mais grave talvez, o método de Newton dá pouca

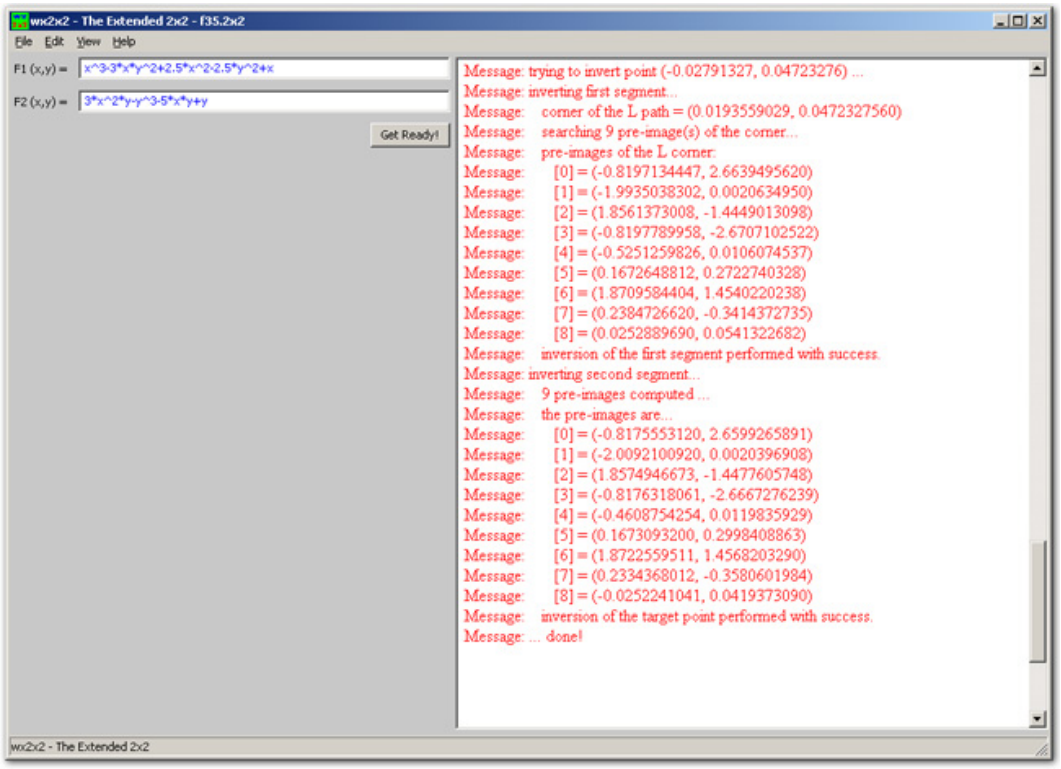


Figura 2.4: Informações sobre a inversão de um ponto

informação sobre o comportamento global da função $F = (F_1, F_2)$ associada ao lado esquerdo do sistema.

O **wx2x2** resolve sistemas não lineares a duas equações e duas incógnitas, utilizando uma outra abordagem. O método de resolução aproveita-se de informações de natureza global e, desta forma, é capaz de calcular o número de pré-imagens de um ponto antes de encontrá-las. As aproximações iniciais usadas para alcançá-las são selecionadas de forma robusta, garantindo a convergência do método de continuação numérica utilizado.

3
Preliminares teóricas: funções do plano no plano

Neste capítulo, introduziremos definições e resultados básicos da análise matemática que serão úteis para a descrição do **wx2x2**. As demonstrações podem ser encontradas nas referências [MST1], [MST2] e, para aspectos mais gerais, [LAN]. Os algoritmos implementados no **wx2x2**, por sua vez, serão apresentados no Capítulo 5.

3.1
Primeiras definições

Seja F uma função diferenciável do plano no plano. Um ponto p do domínio de F é *regular* se a matriz jacobiana DF nesse ponto é inversível, em outras palavras, se $\det(DF(p)) \neq 0$. A partir do Teorema da Função Inversa, após mudanças de variáveis em vizinhanças apropriadas de p e $F(p)$, a função F toma a forma $\tilde{F}(x, y) = (x, y)$. Mais precisamente, existem difeomorfismos locais Φ e Ψ tal que $F = \Phi \circ \tilde{F} \circ \Psi$. Um ponto do domínio que não é regular é chamado de *ponto crítico*. O ponto p é uma *pré-imagem* de q quando $F(p) = q$. Um ponto q da imagem de F é um *valor crítico* se alguma pré-imagem for um ponto crítico. Um ponto do contradomínio que não é um valor crítico é chamado de *valor regular*. Denotaremos por $\mathcal{C}$ o conjunto dos pontos críticos de F e por $F(\mathcal{C})$ o conjunto dos valores críticos de F.

Seja p um ponto crítico tal que $\text{grad}(\det DF)(p) \neq 0$. Pelo *Teorema da Função Implícita*, as soluções de $\det(DF(x)) = 0$ próximas a p formam uma curva suave. Assim, o conjunto crítico $\mathcal{C}$ seria um conjunto de curvas simples se só constasse de pontos regulares.

Um ponto crítico p da função F é uma *dobra* (resp. *cúspide*) se existir dois difeomorfismos locais (preservando orientação) ϕ e ψ em torno de p e $F(p)$, respectivamente, sobre vizinhanças da origem no plano, de modo que $\psi \circ F \circ \phi$ seja dada por (3-1) (resp. (3-2)) abaixo:

$$(u, v) \mapsto (u, v^2) \tag{3-1}$$

$$(u, v) \mapsto (u, \alpha v^3 - uv), \quad \alpha = \pm 1. \tag{3-2}$$

Vejamos uma caracterização mais explícita para dobras e cúspides. Um ponto crítico p é uma dobra se

a) $\text{grad}(\det DF)(p) \neq 0$,

b) o núcleo K de DF em p tem dimensão 1,

c) K e a reta tangente T ao conjunto crítico em p não coincidem.

E um ponto crítico p é uma cúspide se

a) $\text{grad}(\det DF)(p) \neq 0$,

b) o núcleo K de DF em p tem dimensão 1,

c) K e a reta tangente T ao conjunto crítico em p coincidem,

d) para uma parametrização suave $\gamma : (-\epsilon, \epsilon) \to \mathbb{R}^2$ de $\mathcal{C}$ perto de p, com $\gamma(0) = p$ e $\gamma'(0) \neq 0$, o ângulo $\theta(\gamma(t))$ entre o núcleo $K(\gamma(t))$ de $DF(\gamma(t))$ e a reta tangente $T(\gamma(t))$ a $\mathcal{C}$ satisfaz $\theta'(0) \neq 0$.

Seja $F : \mathbb{R}^2 \to \mathbb{R}^2$ suave. Dizemos que F é uma *função excelente* se todo ponto crítico de F for uma dobra ou uma cúspide. Para F excelente, 0 (zero) é valor regular de $\det(DF)$ pois $\text{grad}(\det DF)(p) \neq 0$ para todo p do plano e daí, como conseqüência do Teorema da Função Implícita, o conjunto $\mathcal{C}$ é uma família de curvas simples regulares disjuntas [MI]. Em [W], precisamente no Teorema 13A, Whitney demonstra que, numa topologia apropriada, funções genéricas do plano no plano são funções excelentes.

Uma função contínua F do plano no plano é *própria* se a pré-imagem de qualquer subconjunto compacto é um conjunto compacto. Isso equivale a dizer que F leva infinito a infinito, $\lim_{|(x,y)|\to\infty} |F(x,y)| = \infty$.

Seja $\gamma : S^1 \to \mathbb{R}^2$ uma função contínua ($S^1 = \{(x,y) \in \mathbb{R}^2 : x^2 + y^2 = 1\}$) e q fora de sua imagem. O *número de rotação* de γ em torno de q, $\omega(\gamma, q)$, ou *índice do ponto q com respeito a γ*, é o grau da função $\nu(\theta) = (\gamma(\theta) - q)/|\gamma(\theta) - q|$ de S^1 em S^1. Informalmente, o número de rotação conta as voltas da curva (orientada) dada pela imagem de γ em torno do ponto q. Se, além disso, γ tem uma derivada sempre não nula, a *giração* $\tau(\gamma)$ de γ é o número de rotação da função de S^1 em S^1 dada por $\vartheta(t) = \gamma'(t)/|\gamma'(t)|$ em torno da origem, $\tau(\gamma) = \omega(\vartheta, 0)$. A giração conta as voltas do vetor tangente a γ ao longo de γ.

Para U aberto limitado de $\mathbb{R}^2$, e uma função contínua $F : \overline{U} \to \mathbb{R}^2$ com $q \notin \partial U$, definimos seu *grau topológico* (ou, simplesmente, grau) que, genericamente, é dado pela expressão

$$\deg(F, U, q) = \sum_{F(p)=q, p\in U} \text{sgn}\det(DF(p)).$$

Suponha que, por alguma razão, a expressão acima não faça sentido (ainda que mantenhamos $q \notin \partial U$): F, por exemplo, poderia não ser diferenciável, ou q poderia ter um número infinito de pré-imagens em U. É parte da construção do grau topológico que qualquer aproximação razoável de F para o qual a expressão faz sentido é adequada para o cálculo do grau. Informalmente, o grau conta algebricamente (isto é, com uma escolha apropriada de sinais) o número de pré-imagens de q em U. O grau $\deg(F, \mathbb{R}^2, q)$ também é bem definido para funções próprias $F : \mathbb{R}^2 \to \mathbb{R}^2$, e, nesse caso, independe de q.

Seja $F : \mathbb{R}^2 \to \mathbb{R}^2$ uma função excelente e própria. Ela é *cordata* se seu conjunto crítico $\mathcal{C}$ é limitado e a imagem $F(\mathcal{C})$ consiste de uma família normal de curvas. Uma curva $F(\gamma)$ é normal quando nenhum ponto seu tem mais de duas pré-imagens em γ e, em pontos de autointerseção, os vetores tangentes a $F(\gamma)$ são linearmente independentes.

O conjunto $\mathcal{F} = F^{-1}(F(\mathcal{C}))$ é a *flor* da função F. Dado um subconjunto X do plano as componentes conexas de $\mathbb{R}^2 \backslash X$ são as *placas* de X. Neste texto, consideraremos com freqüência as placas de $\mathcal{F}$, de $\mathcal{C}$ e de $F(\mathcal{C})$.

Seja Γ uma curva crítica de uma função cordata F, limitando um disco topológico aberto D. Uma cúspide $p \in \Gamma$ é chamada de *interna* se, para toda vizinhança suficientemente pequena U_p de p, $F^{-1}(F(\Gamma)) \cap U_p$ está contida em $\overline{D}$; caso contrário, o ponto p é chamado de cúspide *externa*.

Para F cordata, o *sentido de dobra* de uma curva crítica Γ no domínio é a orientação que deixa os pontos p com $\det(DF(p)) > 0$ imediatamente à esquerda de Γ, como indicado na Figura 3.1. A composição de F com uma parametrização da curva crítica, orientada com sentido de dobra, induz uma orientação para $F(\Gamma)$, o *sentido de dobra* de $F(\Gamma)$, com a seguinte propriedade geométrica: a imagem de uma pequena vizinhança de um ponto de dobra $p \in \Gamma$ está inteiramente à esquerda de $F(\Gamma)$, como na Figura 3.1. Isso segue da forma normal da dobra. O sentido de dobra de Γ corresponde à orientação positiva (sentido anti-horário) se $\det(DF)$ é positivo imediatamente dentro de Γ.

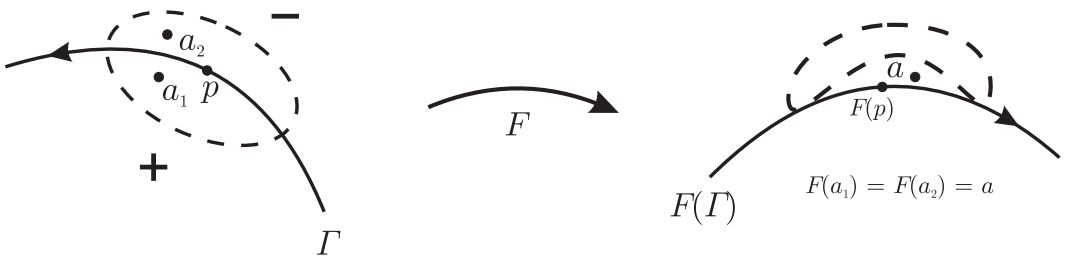


Figura 3.1: Sentido de dobra

3.2 Um pouco da teoria geral

Nesta seção apresentaremos os teoremas que embasam os algoritmos que compõem o **wx2x2**.

Teorema 3.1 *Seja $F : \mathbb{R}^2 \to \mathbb{R}^2$ uma função suave própria, $q \in \mathbb{R}^2$ e γ uma parametrização de uma curva fechada normal simples Γ orientada positivamente tal que todas as pré-imagens de q estão do lado de dentro de Γ. Então*

a) O número de rotação $\omega(F \circ \gamma, q)$ independe de γ e q;

b) Se q é um valor regular de F então

$$\omega(F \circ \gamma, q) = \sum_{F(p)=q} \operatorname{sgn} \det(DF(p)).$$

O teorema é empregado para o cálculo do grau de uma função, um dos primeiros procedimentos executados pelo programa ao receber uma função declarada pelo usuário como cordata. Assim, o grau é o número de rotação em torno de zero da imagem de um grande círculo orientado positivamente centrado na origem.

Teorema 3.2 *Seja E um recobrimento de um conjunto simplesmente conexo X por uma função F. Então F é um homeomorfismo.*

Um exemplo desse resultado já foi mostrado no Capítulo 2: a função F estudada leva cada placa ciano ao triângulo ciano da imagem homeomorficamente. Isso não acontece necessariamente para placas com buracos, como é o caso de uma das placas amarelas do domínio, nem para placa branca.

Teorema 3.3 *Seja F cordata, E uma placa de $\mathcal{F}$ e X a placa de $F(\mathcal{C})$ que contém $F(E)$. Então E é um recobrimento de ordem finita de X por F.*

Assim, podemos falar em *pré-imagens* de X quando X for uma placa de $F(\mathcal{C})$, pois o conjunto $F^{-1}(X)$ é uma reunião disjunta de placas de $\mathcal{F}$, cada uma das quais chamada de *pré-imagem de X*. Em particular, nesse caso, o número de pré-imagens de qualquer ponto em uma mesma placa X é constante.

Corolário 3.4 *Seja F cordata. Então a pré-imagem da placa ilimitada de $F(\mathcal{C})$ é a placa ilimitada de $\mathcal{F}$. O grau de F não é nulo (logo F é sobrejetora) e o número de pré-imagens da placa ilimitada de $F(\mathcal{C})$ é igual ao módulo do grau.*

No Capítulo 2, para a função F estudada, a placa amarela possui 6 placas pré-imagens. Todavia, cada ponto dessa placa possui 7 pré-imagens. Para a placa ilimitada, cada ponto possui 3 pré-imagens pois $\deg(F) = 3$ (o programa informa este valor na área de log).

Aliás, o comportamento de uma função cordata em infinito é bastante simples.

Teorema 3.5 *Seja F uma função cordata e $n = |\deg(F)| \neq 0$. Então, em vizinhanças de ∞, no domínio e na imagem, a menos de trocas de variáveis, F comporta-se como $z \mapsto z^n$ ou $z \mapsto \overline{z}^n$. O primeiro caso ocorre exatamente quando* $\deg(F) > 0$.

Assim, em princípio, para uma função cordata, não é difícil calcular as outras pré-imagens de um ponto $q = F(p)$, onde p é dado, suficientemente longe do conjunto crítico $\mathcal{C}$.

O resultado seguinte mostra como empregar informação sobre o número de pré-imagens em infinito para a contagem de pré-imagens das placas limitadas da imagem.

Teorema 3.6 *Seja F uma função cordata. Então numa vizinhança de uma imagem q de uma dobra, na imagem q' de uma cúspide ou de um ponto de intersecção q'' com k pré-imagens, o número de pré-imagens muda de acordo com os diagramas na Figura 3.2, onde as setas indicam o sentido de dobra.*

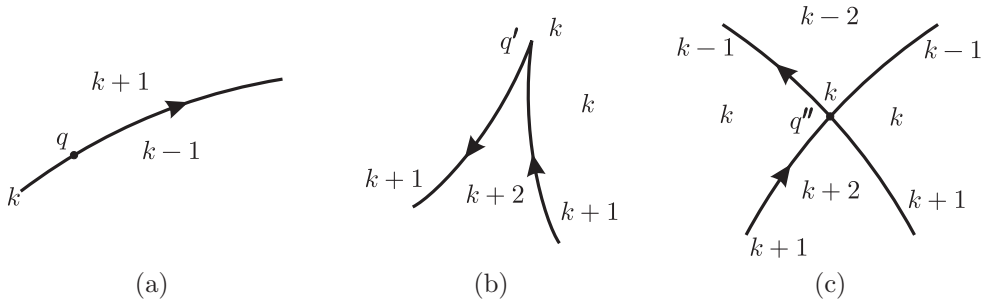


Figura 3.2: Diagrama de mudança de pré-imagens

Mais uma vez, o Capítulo 2 mostra o teorema em ação para as placas na imagem da função F, onde seus pontos possuíam 3, 5, 7 ou 9 pré-imagens.

A partir de agora, apresentaremos relações de compatibilidade que serão úteis na identificação das curvas críticas de uma função F cordata. Os teoremas a seguir buscam caracterizar seus conjuntos $\mathcal{C}$ e $F(\mathcal{C})$. Como veremos na próxima seção, o **wx2x2** utiliza o conhecimento desses conjuntos para inverter pontos. Na pesquisa de curvas críticas, o programa usa essas relações para

determinar se o conjunto de curvas críticas já calculado $\mathcal{C}_\bullet$ e sua imagem $F(\mathcal{C}_\bullet)$ de fato são iguais aos conjuntos críticos $\mathcal{C}$ e $F(\mathcal{C})$ de F, respectivamente.

Para uma curva orientada Γ, definimos a giração de sua imagem $\nu(\Gamma) = \tau(F(\Gamma))$. Denotamos por $\kappa_{int}(\Gamma)$ e $\kappa_{ext}(\Gamma)$ o número de cúspides internas e externas em Γ.

Teorema 3.7 (Teste de Contagem) *Seja F uma função cordata, com conjunto crítico $\mathcal{C}$. Considere uma placa de $\mathcal{C}$, com fronteira exterior Γ_0 (caso exista) e fronteiras interiores $\Gamma_1, \ldots, \Gamma_m$ (caso existam). Então*

$$\nu(\Gamma_0) - \kappa_{int}(\Gamma_0) + \sum_{1 \le i \le m} (\nu(\Gamma_i) - \kappa_{ext}(\Gamma_i)) = 1 - m.$$

Se não existir fronteira exterior, $\nu(\Gamma_0)$ deve ser interpretado como $|\deg(F)|$.

Corolário 3.8 *Seja F uma função cordata com conjunto crítico $\mathcal{C}$ consistindo de curvas $\Gamma_1, \ldots, \Gamma_n$, com um total de κ cúspides. Então*

$$|\deg(F)| = 1 + \kappa - 2 \sum_{1 \le i \le n} \nu(\Gamma_i).$$

O Teorema 3.7 fornece um conjunto de condições necessárias, uma em cada placa de $\mathcal{C}$, que devem ser satisfeitas para conjuntos de curvas candidatas a formar o conjunto crítico de uma função cordata, mas essas condições não são suficientes. Por exemplo, suponha que para uma função cordata F, tivéssemos encontrado duas curvas críticas sem cúspides Γ_0 e Γ_1, orientadas por sentido de dobra como na Figura 3.3, junto com suas imagens. É fácil ver que o Teorema 3.7 é satisfeito na placa interior a Γ_0, mas não vale para a placa entre Γ_0 e Γ_1: ao orientar as curvas como indicado no teorema, obtemos $1+1 = 1-1$, mostrando que existem pontos críticos nessa placa.

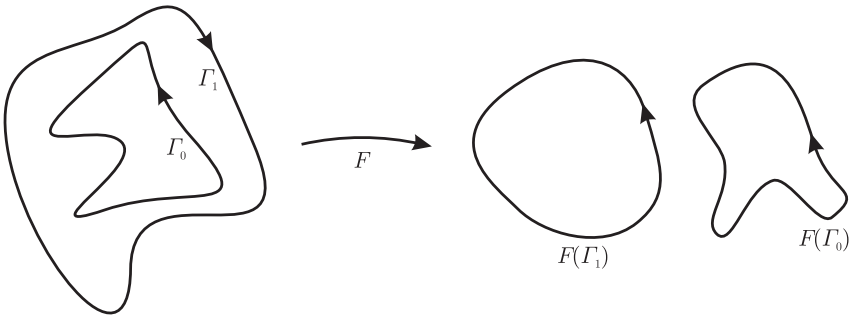


Figura 3.3: Faltam curvas críticas no anel entre Γ_0 e Γ_1

O Teorema 3.7 fornece condições que devem ser satisfeitas por $\mathcal{C}$ — quando parar? Condições adicionais, que são também suficientes, seguem da *Teoria de Blank-Troyer* ([MST1]). Os resultados fazem uso de construções de topologia combinatória, e foram detalhados nas referências. Nesse texto, não acrescentamos nada a esta parte do programa. Vale, entretanto, observar que, ao longo de uma quantidade substancial de exemplos estudados, verificou-se que os testes de contagem são muito rigorosos: sempre que um conjunto de curvas críticas $\mathcal{C}_\bullet$ e sua imagem satisfaziam esses testes, eles também satisfaziam aos testes de Blank-Troyer. Esses testes adicionais são bastante exigentes computacionalmente, e o programa tem um parâmetro, descrito no capítulo seguinte, que permite desativá-los.

3.3
Um exemplo explícito: $\widetilde{F}(z) = z^2 + \bar{z}$

Vamos calcular o conjunto crítico e a flor da função $F(x, y) = (x^2 + y^2 + x, 2xy - y)$, ou $\widetilde{F}(z) = z^2 + \bar{z}$, em notação complexa. Sua jacobiana no ponto (x, y) é

$$DF(x, y) = \begin{bmatrix} 2x + 1 & -2y \\ 2y & 2x - 1 \end{bmatrix}$$

e seu determinante,

$$g(x, y) = \det(DF(x, y)) = 4(x^2 + y^2) - 1. \tag{3-3}$$

Daí, o conjunto crítico de F, $\mathcal{C} = \{(x, y) \in \mathbb{R}^2 : g(x, y) = 0\}$, é dado pela circunferência de raio $\frac{1}{2}$ centrada na origem, que parametrizamos por $\gamma(t) = (\frac{1}{2}\cos t, \frac{1}{2}\operatorname{sen} t)$. Notemos que $\nabla g(x, y) = (8x, 8y)$ é não nulo para todos os pontos em $\mathcal{C}$, e assim a primeira condição para um ponto crítico ser dobra ou cúspide é satisfeita. O núcleo de DF no ponto $(\frac{1}{2}\cos t, \frac{1}{2}\operatorname{sen} t)$ é gerado pelo vetor $(\operatorname{sen}\frac{t}{2}, \cos\frac{t}{2})$, pois

$$DF(\frac{1}{2}\cos t, \frac{1}{2}\operatorname{sen} t)\begin{bmatrix} \operatorname{sen}\frac{t}{2} \\ \cos\frac{t}{2} \end{bmatrix} = \begin{bmatrix} \cos t + 1 & -\operatorname{sen} t \\ \operatorname{sen} t & \cos t - 1 \end{bmatrix}\begin{bmatrix} \operatorname{sen}\frac{t}{2} \\ \cos\frac{t}{2} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

Alinhando o vetor gerador do núcleo e o vetor $\gamma'(t)$, obtemos os valores de t para os quais a reta tangente ao conjunto crítico coincide com o núcleo da jacobiana: $t = 0$, $\frac{2\pi}{3}$ e $\frac{4\pi}{3}$. Logo, com exceção das raízes cúbicas da unidade (divididas por 2), os pontos de $\mathcal{C}$ são dobras. Para mostrar que esses três pontos são de fato cúspides falta verificar o item (d) da definição (pág. 18). Sejam $\theta_1(t) = \frac{\pi}{2} + t$ e $\theta_2(t) = \frac{\pi}{2} - \frac{t}{2}$, respectivamente, os argumentos do vetor tangente e do gerador do núcleo de DF. Então $\theta(t) = \theta_1(t) - \theta_2(t) = \frac{3t}{2}$, e daí

$\theta'(t)$ é sempre não nulo. Assim, os três pontos são de fato cúspides.

Agora mostraremos que $F(\mathcal{C})$ é uma curva fechada sem auto-intersecções. Consideremos dois pontos (x, y) e (u, v) em $\mathcal{C}$ com a mesma imagem. Algebricamente, isso corresponde ao sistema

$$\begin{aligned} x^2 + y^2 = \frac{1}{4} \quad &, \quad u^2 + v^2 = \frac{1}{4} \\ x^2 - y^2 + x = u^2 - v^2 + u \quad &, \quad 2xy - y = 2uv - v \end{aligned}$$

que tem a única solução $x = u$ e $y = v$. Portanto, $F(\mathcal{C})$ não possui pontos de auto-intersecção.

Finalmente, para mostrarmos que F é cordata falta verificar que F é própria, o que é imediato de sua forma complexa, já que $\lim_{|z|\to\infty} |\tilde{F}(z)| = \infty$.

Observamos, a partir da Equação (3-3) que o sinal de $\det DF$ na origem é negativo, portanto o sentido de dobra coincide com o sentido horário. O módulo do grau de F pode ser obtido através do Corolário 3.8: $|\deg(F)| = 1+3-2\cdot 1 = 2$, ou da forma complexa $\widetilde{F}$. Circunferências de raio arbitrariamente grande centradas na origem e orientadas positivamente são levadas em circunferências a curvas que dão duas voltas no sentido anti-horário em torno da origem: assim, $\deg(F) = 2$.

A Figura 3.4 exibe os conjuntos $\mathcal{C}$, $F(\mathcal{C})$ e $\mathcal{F}$ para a nossa função exemplo. O Corolário 3.4 nos diz que F é sobrejetiva, a imagem da placa ilimitada P_0 é a placa $\widetilde{P}_0$ e que cada ponto de $\widetilde{P}_0$ possui duas pré-imagens na placa P_0. Mais ainda, de acordo com os Teoremas 3.2 e 3.3, sendo $\widetilde{P}_1$ simplesmente conexo, F restrita a cada umas das placas P_i, $i = 1, \ldots, 4$, é um difeomorfismo sobre $\widetilde{P}_1$. Logo, cada ponto em $\widetilde{P}_1$ possui quatro pré-imagens, o que também pode ser deduzido utilizando o Teorema 3.6.

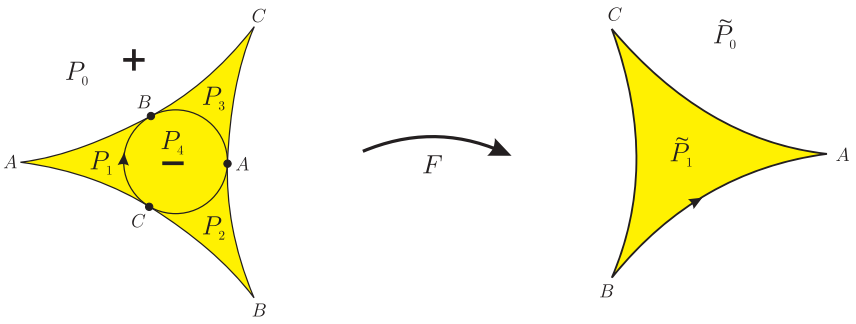


Figura 3.4: $\mathcal{C}$, $F(\mathcal{C})$ e $\mathcal{F}$ para a função $\widetilde{F}(z) = z^2 + \bar{z}$

3.4
Calculando as pré-imagens de um ponto de $\widetilde{F}(z) = z^2 + \bar{z}$

Dada F cordata com conjuntos críticos $\mathcal{C}$ e $F(\mathcal{C})$ conhecidos, como obter todas as pré-imagens de um ponto q dado? O processo de inversão que apresentaremos aqui é geral, ou seja, é o mesmo para a inversão de pontos para qualquer função cordata, e será detalhado no Capítulo 5. Para simplificar, aqui consideraremos a inversão de pontos pela função $\tilde{F}(z) = z^2 + \bar{z}$.

O primeiro passo para resolver o sistema $F(p) = q$ é obter todas as pré-imagens de algum ponto remoto q_0, processo que passamos a descrever. Escolha um ponto p_0^0 suficientemente longe da origem, de forma que $F(p_0^0)$ esteja fora de um disco centrado na origem contendo $F(\mathcal{C})$: isso pode ser feito porque F é própria e, portanto, $\lim_{|z|\to\infty} |F(z)| = \infty$. Defina $q_0 = F(p_0^0)$: uma das pré-imagens de q_0 é obviamente p_0^0.

Como o número de pré-imagens de um ponto pertencente à placa ilimitada de $F(\mathcal{C})$ é $\deg(F) = 2$, q_0 ainda possui uma outra pré-imagem e, para obtê-la, considere ω uma circunferência centrada na origem do contradomínio, percorrida de modo a começar e terminar em q_0. Agora, por algum método de continuação, obtemos a imagem inversa de ω partindo de p_0^0, que nos levará à outra pré-imagem p_0^1 de q_0 quando completarmos a inversão de ω.

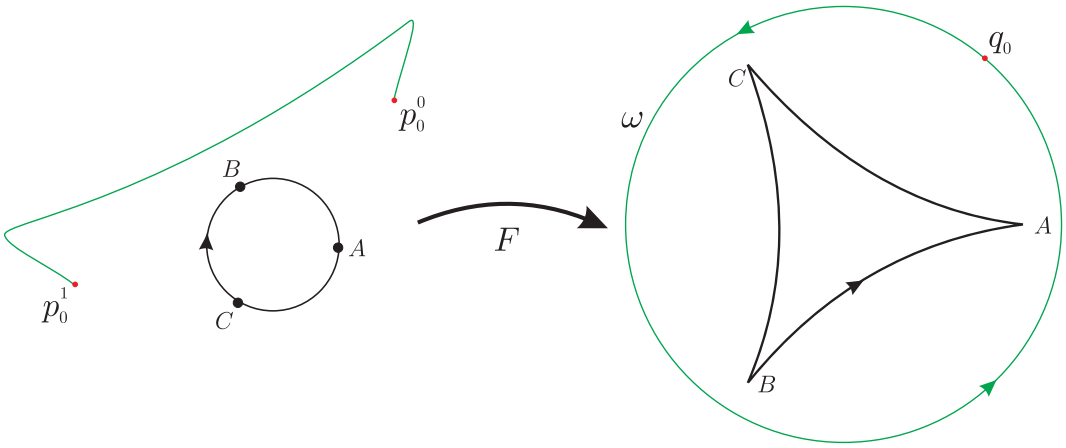


Figura 3.5: A outra pré-imagem de q_0

Qualquer outro ponto q_1 na placa ilimitada de $F(\mathcal{C})$ possui duas pré-imagens. Para resolver $F(p) = q_1$, basta traçar uma curva β ligando q_0 a q_1 sem interceptar $F(\mathcal{C})$ e invertê-la por continuação a partir das duas condições iniciais p_0^0 e p_0^1, obtendo as pré-imagens p_1^0 e p_1^1 de q_1.

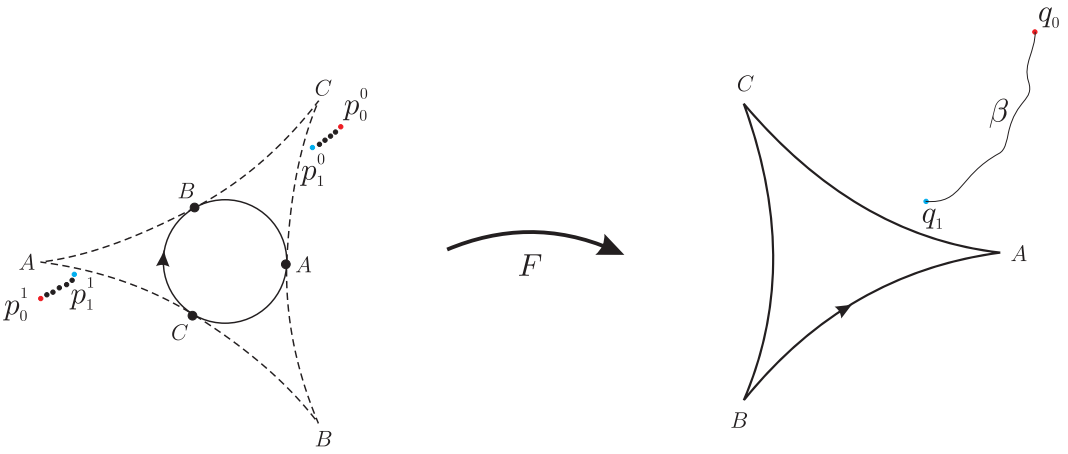


Figura 3.6: Inversão de um ponto pertencente à placa ilimitada

Finalmente, seja q_2 um ponto da placa limitada de $F(\mathcal{C})$ que, portanto, possui quatro pré-imagens. Para resolver $F(p) = q_2$, ligue q_0 a q_2, preferencialmente por um caminho β que intercepte $F(\mathcal{C})$ em um ponto só, q_*, que seja a imagem de uma dobra.

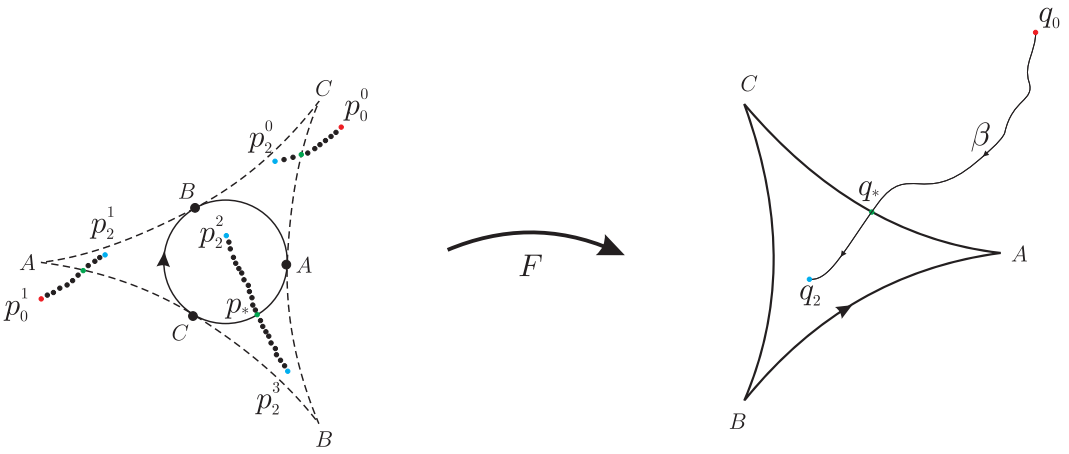


Figura 3.7: Inversão de um ponto pertencente à placa limitada

Iniciamos o processo de inversão por continuação ao longo de β a partir do ponto inicial q_0. Fazemos isso para cada pré-imagem de q_0 (p_0^0 e p_0^1), obtendo duas pré-imagens de q_2: p_2^0 e p_2^1. Para obter as duas pré-imagens que faltam, calculamos o ponto p_* em $\mathcal{C}$ tal que $F(p_*) = q_*$ — como veremos no Capítulo 5, isto se obtém a partir da lista dos pontos das curvas críticas, já calculadas pelo

programa. Para gerar as duas pré-imagens perto de p_*, calculamos o núcleo K da jacobiana no ponto p_* e escolhemos dois pontos s_1 e s_2 perto de p_* na reta $p_* + K$, um de cada lado da curva crítica. As imagens $d_1 = F(s_1)$ e $d_2 = F(s_2)$ provavelmente não pertencem ao nosso caminho de inversão β. Conectamos d_1 e d_2 a β por pequenos caminhos α_1 e α_2, invertemo-los com condições iniciais s_1 e s_2 e, finalmente, continuamos a inversão ao longo do caminho de inversão original β, obtendo assim, as pré-imagens restantes p_2^2 e p_2^3.

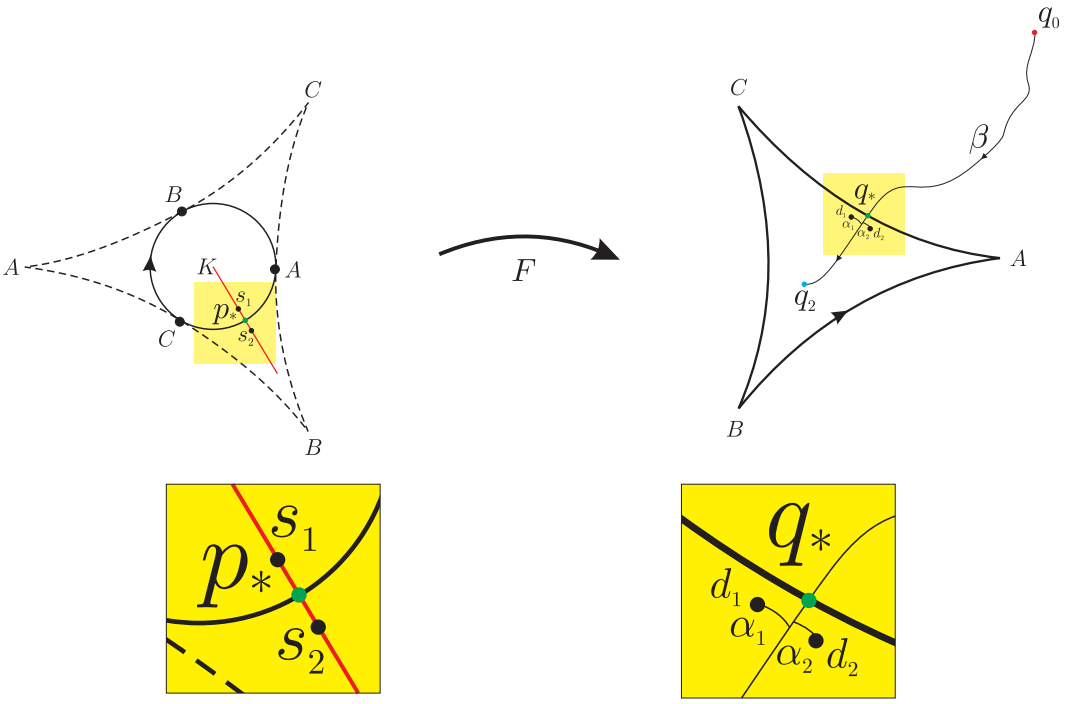


Figura 3.8: Inversão com condições iniciais próximas à $\mathcal{C}$

4
Usando o wx2x2

Em linha gerais, dada uma função F cordata, o **wx2x2** calcula inicialmente seu conjunto crítico $\mathcal{C}$ e coloca-se à disposição do usuário para realizar a tarefa seguinte, que pode ser o cálculo da flor $\mathcal{F}$ ou o cálculo das pré-imagens de algum ponto (ou conjunto de pontos), ou ainda a interação gráfica que permite ao usuário obter uma variedade de informações geométricas e numéricas.

Não seria exagerado afirmar que a quase totalidade dos procedimentos computacionais do programa dedica-se à resolução do sistema

$$F(x) = q, \quad x,\, q \in \mathbb{R}^2. \tag{4-1}$$

Quando o programa resolve esse sistema, ou seja, quando todas as pré-imagens de q são obtidas, dizemos que o ponto q foi invertido ou que q é *um ponto resolvido*.

No caso de funções sabidamente não cordatas, como é o caso de funções que não têm conjunto crítico limitado, o **wx2x2** fornece alternativas, que levam ao estudo detalhado da função em um domínio limitado do plano, determinado pelo usuário.

4.1
A área de trabalho do wx2x2

Na tela inicial do programa, sua *área de trabalho*, o usuário deve apenas fornecer as funções coordenadas da função $F = (F_1, F_2)$ em termos das variáveis x e y e clicar no botão "Get Ready!". À direita da área de trabalho, na área de log, o programa registra um conjunto de informações específicas da função, como mostrado no Capítulo 2. Duas novas janelas são sobrepostas à área de trabalho. A janela frontal à esquerda representa o domínio da função. Nela, vê-se o conjunto crítico $\mathcal{C}$ e as cúspides de F (Figura 4.2a). Na segunda, representando o contradomínio da função, vê-se a imagem do conjunto crítico $F(\mathcal{C})$ (Figura 4.2b).

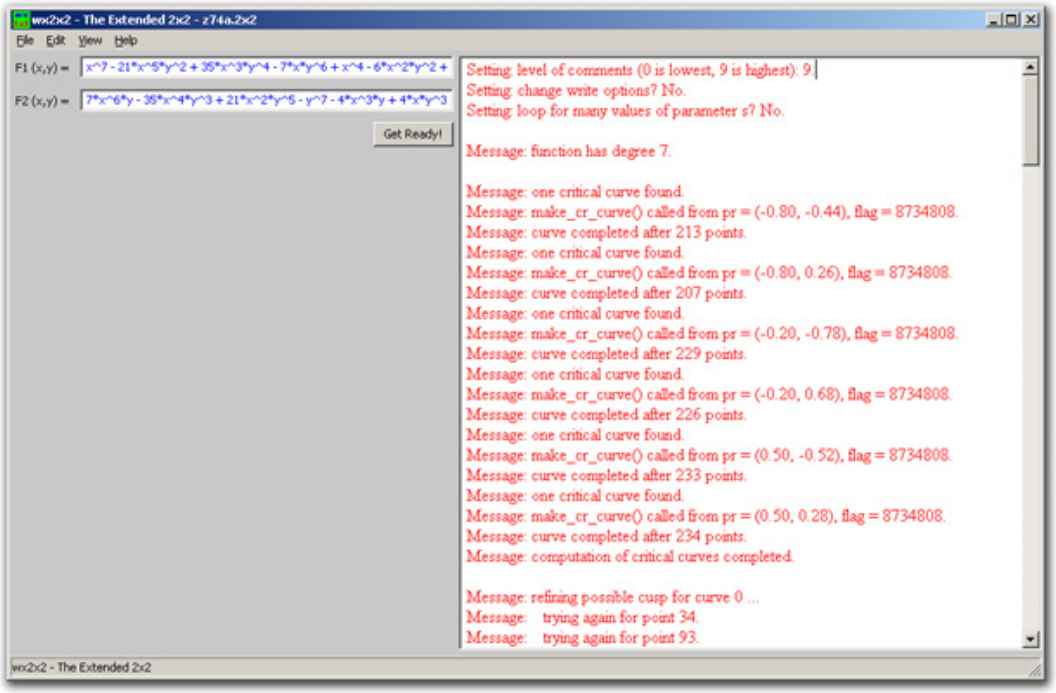
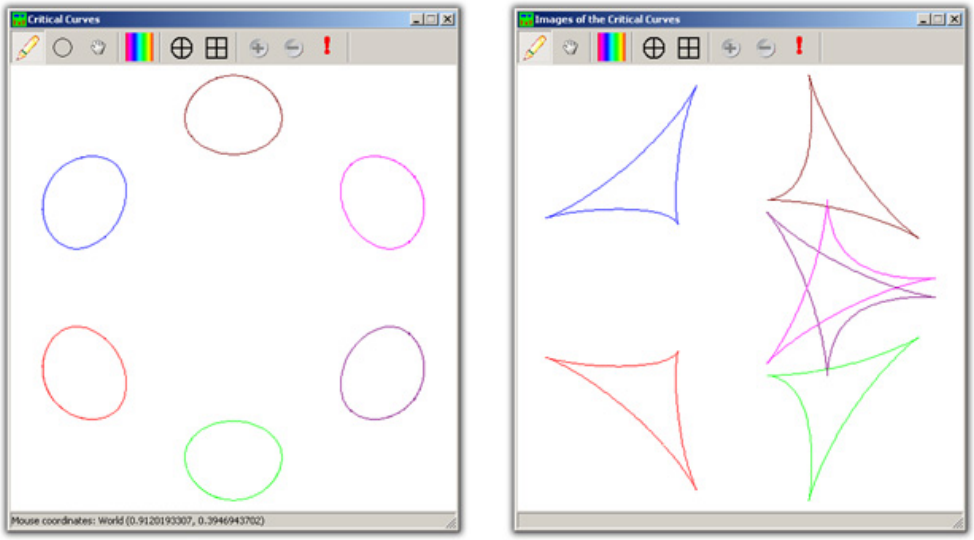


Figura 4.1: Área de trabalho do wx2x2



(a) Conjunto Crítico (b) Imagem do Conjunto Crítico

Figura 4.2: Janelas do Domínio e do Contradomínio

4.2
Modos de Análise: funções próprias e domínios limitados

O **wx2x2** possui três modos distintos para analisar funções do plano no plano. O primeiro, o modo *standard*, cuja interface é exibida na Figura 4.1, trata de funções *cordatas*. Já o segundo, o *modo bump*, exibido na Figura 4.3,

trabalha com funções da forma

$$H(\mathbf{x}) = B(\mathbf{x}) \cdot F(\mathbf{x}) + [1 - B(\mathbf{x})] \cdot G(\mathbf{x}), \quad \mathbf{x} \in \mathbb{R}^2,$$

onde $B : \mathbb{R}^2 \to \mathbb{R}$ é uma *função bump* e H é *cordata*. Finalmente, o terceiro modo é o *mask*, exibido na Figura 4.6, no qual o usuário restringe o domínio da função F a um disco de sua preferência.

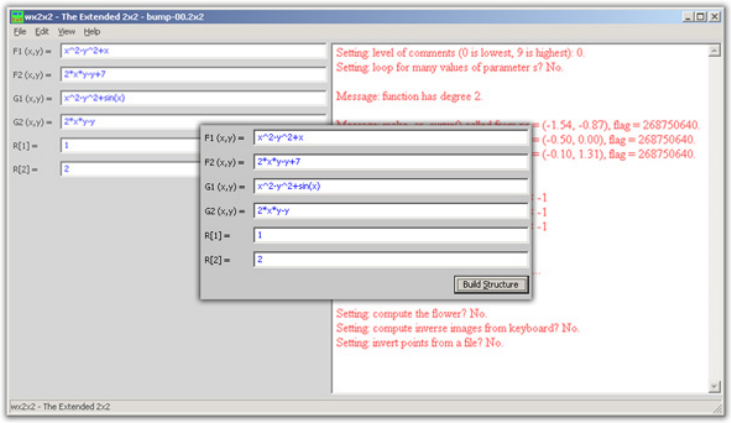


Figura 4.3: Modo Bump

Em princípio, o usuário não sabe que a função de seu interesse é cordata. A propriedade é genérica, para uma topologia adequada, mas ainda assim, para fins de análise numérica, até uma função cordata pode se comportar como se não o fosse — em vários procedimentos, o programa envia uma mensagens de erro causadas por esse fenômeno. Existem parâmetros reguláveis pelo usuário que fazem com que os algoritmos procedam com maior cautela, justamente para poder recuperar, dispondo de precisão suficiente, o fato da função original ser cordata.

Existem casos em que o usuário tem de fato interesse numa função F não cordata. Isso ocorre por exemplo quando o conjunto crítico $\mathcal{C}$ não é limitado: um exemplo quase trivial é a dobra em sua forma normal, $(x, y) \mapsto (x, y^2)$. Nessa situação, convém usar o modo *bump*. Essencialmente, o usuário justapõe a função F à outra função G de sua escolha, obtendo uma função H de forma que, dentro de um disco escolhido pelo usuário, H comporta-se como F, e fora de outro disco devidamente escolhido, como G. O procedimento é transparente ao usuário, depois de fornecidos F, G e os raios dos discos (centrados na origem). O programa define H como a combinação descrita acima de F e G

com pesos determinados por um *bump* B dado por

$$B(\mathbf{x}) = \begin{cases} 1 & 0 \leq ||\mathbf{x}|| \leq R_1 \\ \sum_{i=0}^{7} \alpha_i ||\mathbf{x}||^i & R_1 < ||\mathbf{x}|| \leq R_2 \\ 0 & ||\mathbf{x}|| > R_2 \end{cases}$$

onde os coeficientes α_i's dependem dos raios dos discos e $||\cdot||$ é a norma euclidiana de $\mathbb{R}^2$. A partir daí, o programa opera com H como se fosse uma função cordata.

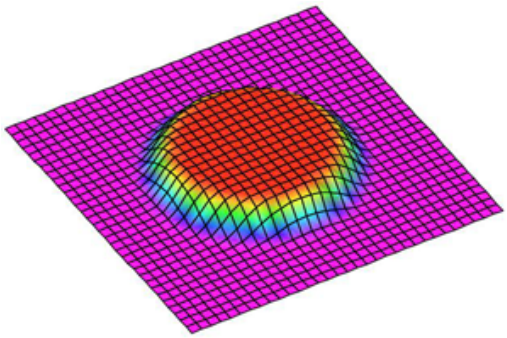


Figura 4.4: Gráfico de uma bump

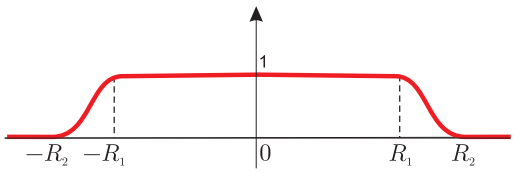


Figura 4.5: Traço no plano $x = 0$

No modo *mask*, o programa invoca o modo *bump* a partir de uma função H obtida amalgamando a função F original com a função G dada pela identidade. O raio interno é dado pelo usuário e o externo vale 1.5 vezes o interno. Nas janelas gráficas, são mostradas apenas as informações contidas no disco escolhido pelo usuário.

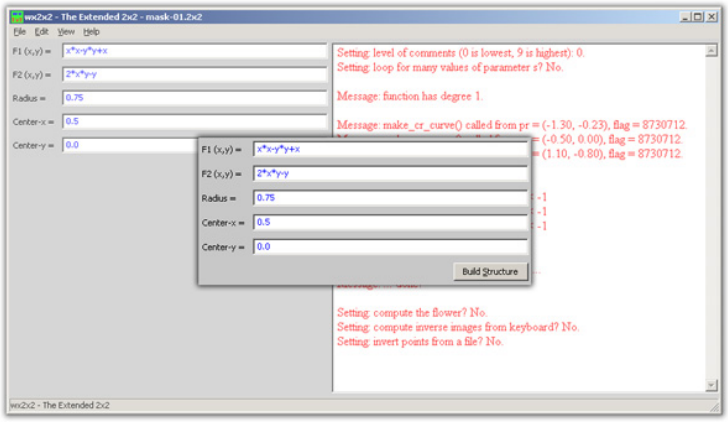


Figura 4.6: Modo mask

4.3 Barras de ferramenta nas janelas de domínio e contradomínio

No alto das figuras 4.2a e 4.2b encontra-se a barra de ferramentas das janelas do domínio e contradomínio. Suas funções são descritas a seguir.

(a) Barra de Ferramentas da janela do domínio

(b) Barra de Ferramentas da janela do contradomínio

Figura 4.7: Barras de Ferramentas

1- Pencil: No domínio, é usada para desenhar um ponto e, automaticamente, sua imagem é calculada pelo programa. Na imagem, desenha um ponto e desenha suas pré-imagens, também calculadas pelo programa.

2- Circle: Desenha uma circunferência e sua imagem. O círculo é dado clicando seu centro e arrastando o cursor a um ponto da circunferência.

3- Hand: Translada a janela de visualização.

4- Forecolor: Permite escolher uma cor para o *Pencil* e para *Circle*.

5- Center: Exibe ou oculta uma pequena marca que indica o centro da janela. É útil para posicionar objetos ao realizar *zooms*.

6- Axes/Grid: Na janela do domínio, exibe a malha utilizada na detecção das curvas críticas (v. seção 5.3); no contradomínio, exibe os eixos cartesianos (v. Figura 4.8).

7, 8- Zooms In e Out: Amplia/Reduz a janela de visualização, fixando o centro da janela.

9- Reset: Exclui tudo o que foi desenhando através do *Pencil*, nas duas janelas ao mesmo tempo.

Todas as ferramentas são acessíveis por meio de teclas de atalho, indicadas na Tabela 4.1.

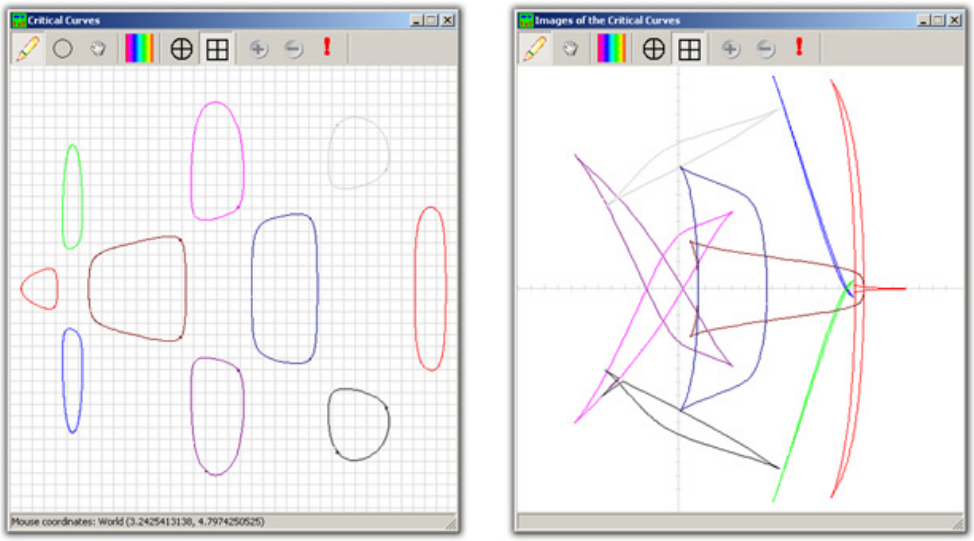


Figura 4.8: A ferramenta Axes/Grid e seus comportamentos

Ferramenta	Tecla de Atalho
Pencil	P ou p
Circle	E ou e
Hand	H, h ou a Barra de espaços pressionada
Forecolor	F ou f
Center	V ou v
Axes/Grid	A ou a
Zoom In	+
Zoom Out	−
Reset	R ou r

Tabela 4.1: Teclas de Atalho

4.4 Parâmetros Globais

O programa permite o ajuste de vários parâmetros globais, que determinam a robustez de vários processos numéricos. Na opção *Preferences* do menu *Edit*, o programa exibe uma janela de diálogo onde o usuário pode ajustar esses parâmetros, descritos a seguir. Na guia *General*, encontram-se

Level of comments: Um número inteiro entre 0 e 9, que determina o nível de detalhe das mensagens exibidas na área de log. No Capítulo 2, mostramos um exemplo onde este parâmetro estava definido com o valor máximo.

Blank: Vale 0 ou 1. Habilita ou desabilita os testes de *Blank-Troyer* (veja Capítulo 4 de [MST2] para detalhes).

Number of tries: Quando o teste de contagem ou o de Blank-Troyer reprova uma placa X do domínio, assim estabelecendo que falta uma curva crítica nessa placa, o programa passa a procurar dois pontos distintos p e q em X nos quais F tem orientações opostas. Isso garante a existência de um ponto crítico no segmento entre p e q. O parâmetro indica o número de tentativas para localizar p e q (veja 5.3.4).

Banksys: O programa armazena o conjunto de pré-imagens de alguns pontos num banco de pontos resolvidos. Banksys igual a -1 indica que todo ponto p cujas pré-imagens são calculadas serão armazenados nesse banco. Se Banksys é igual 1, um ponto p é armazenado só quando o segmento terminando em p invertido na imagem intercepta $F(\mathcal{C})$. Quando Banksys é 0, o banco de pontos resolvidos guarda apenas quatro pontos especiais. (v. Capítulo 5).

Number of neigh: Um número inteiro positivo menor ou igual à constante *MAX_NUM_NEIGH*. Especifica, no processo de inversão de um ponto, a quantidade de caminhos de inversão em forma de L, contados a partir de comprimento mínimo, que serão considerados na rotina que procura um caminho adequado de inversão (v. Capítulo 5). Por *default*, *MAX_NUM_NEIGH*$= 40$.

Os parâmetros da guia *Step Settings* são utilizados nas rotinas que constroem curvas críticas. No **wx2x2** existem duas rotinas principais para essa tarefa, *make_cr_curve()* e *next_zero()*, que são rotinas numéricas de passo variado controladas por:

1- Curve Sharpness: um valor no intervalo $[0, 1)$. Quanto mais próximo de 1, mais fina é a discretização no cálculo de curvas críticas. A Figura 4.9 exibe os conjuntos $\mathcal{C}$ e $F(\mathcal{C})$, da função $(x, y) \mapsto (x^3 - 3xy^2 + 2.5x^2 - 2.5y^2 + x, 3x^2y - y^3 - 5xy + y)$, $\widetilde{F}(z) = z^3 + 2.5\bar{z}^2 + z$, para dois valores distintos do parâmetro.

2- Minimum step e Maximum step: passos extremos empregados na rotina *next_zero()*.

3- Min step in make_cr_curve(): menor passo na rotina *make_cr_curve()*.

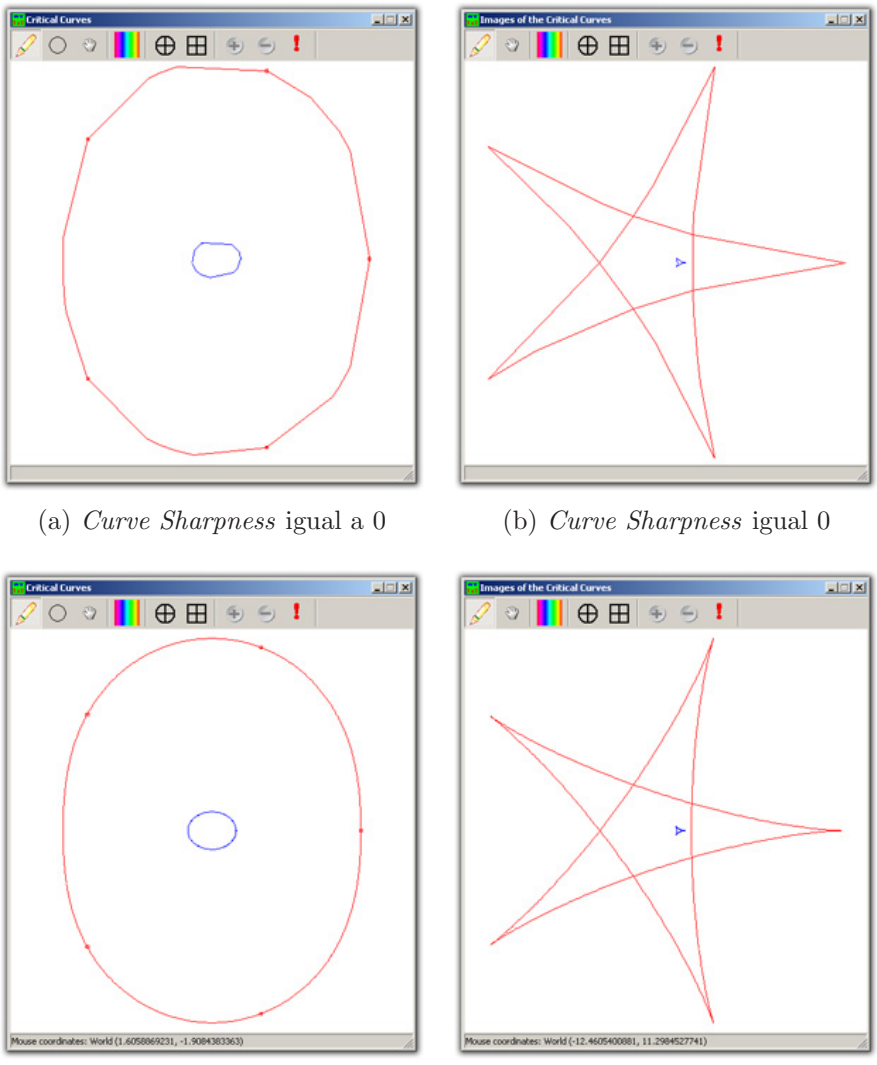
A guia *Grid Settings* contém os parâmetros que determinam o tamanho e a densidade da malha quadrangular centrada na origem utilizada na busca inicial por pontos críticos. Essa malha é uniforme, com espaçamento *Grid step* paras as NGRID colunas e linhas mais internas e expande-se geometricamente a partir daí, com razão *Grid ratio*.

1- NGRID: Determina o número de colunas e linhas da malha. A malha possui $2 \cdot \text{NGRID}$ colunas e $2 \cdot \text{NGRID}$ linhas.

2- Grid step: Espaçamento entre as NGRID linhas e colunas da malha próximo à origem.

3- Grid ratio: Razão da progressão geométrica que determina o espaçamento entre as NGRID linhas e colunas mais afastadas da origem.

Por fim, a guia *GUI Settings* possui o parâmetro *Curve frame width* que determina os tamanhos das janelas do domínio o do contradomínio.



(a) *Curve Sharpness* igual a 0 (b) *Curve Sharpness* igual 0

(c) *Curve Sharpness* igual a 0.95 (d) *Curve Sharpness* igual a 0.95

Figura 4.9: Exemplos de configurações do parâmetro *Curve Sharpness*

5
Procedimentos Computacionais

Neste capítulo descreveremos as principais rotinas do **wx2x2**, utilizadas extensivamente no processo de inversão.

A partir do momento em que o programa possui os valores dos parâmetros globais, assim como as expressões das funções coordenadas F_1 e F_2, a rotina *study_function()* efetua a chamada de várias outras sub-rotinas que, em linhas gerais, realizam as seguintes tarefas: calcular o grau da função, localizar e construir as primeiras curvas críticas e computar sua imagem, coletar a informação necessária para que se proceda aos testes de contagem e à geração de palavras de Blank-Troyer.

À medida que o conjunto crítico $\mathcal{C}$ vai sendo calculado, o programa decompõe curvas críticas e suas imagens em *arcos bimonotônicos* (ao longo dos quais as duas coordenadas são monótonas) e, a partir dessa decomposição, calcula os números de rotação e giração de várias curvas. Cúspides são localizadas com precisão, junto com sua classificação como interna ou externa. O programa calcula também as intersecções de curvas em $F(\mathcal{C})$, permitindo nomear as placas no domínio e na imagem. A seguir, são gerados grafos de adjacência entre placas, necessários para os testes de contagem e Blank-Troyer, empregados para a identificação exaustiva do conjunto crítico $\mathcal{C}$.

Uma vez que essa estrutura está disponível, o programa pode desenhar a flor da função ou inverter pontos, processos que, por sua vez, exigem uma quantidade de outras sub-rotinas. Para começar, são gerados quatro pontos na imagem para os quais é possível nesse momento calcular todas as pré-imagens. A partir desses pontos resolvidos iniciais, outros podem ser invertidos por métodos de continuação ao longo de caminhos β especiais. A inversão da função ao longo de β é natural em situações em que os arcos de $F^{-1}(\beta)$ não encontram pontos críticos. Nos pontos críticos é necessário proceder com mais cuidado: ao trespassar uma dobra, por exemplo, o número de pré-imagens de pontos em β aumenta ou diminui de 2. É conveniente também garantir que os caminhos β a inverter evitem pontos associados a uma análise numérica inutilmente difícil, como ocorre quando β passa próximo à imagem de uma cúspide. O programa tem estratégias para selecionar caminhos de inversão e

ainda acumula informação em um banco de pontos resolvidos, isto é, listas de pontos com suas pré-imagens já calculadas, selecionados por um conjunto de critérios.

Nas próximas seções, são descritas com mais detalhe várias das sub-rotinas acima.

5.1
Calculando o grau

A rotina *find_degree()* calcula o grau de F, baseada no Teorema 3.1. Como F é supostamente cordata, seu grau é o número de rotação $\omega(F \circ \gamma, 0)$, onde γ é uma circunferência centrada na origem, orientada positivamente, de raio suficientemente grande. Amostramos γ em N pontos p_i distribuídos uniformemente. Uma variável inteira *four_degree* é atualizada quando $F(p_i)$ e $F(p_{i+1})$ estão em quadrantes consecutivos: ela aumenta ou diminui de 1, dependendo da orientação. Ao completar a volta, dividimos *four_degree* por 4 e, assim, obtemos o grau da função F.

5.2
Detectando e construindo curvas críticas

Inicialmente percorremos uma malha quadrangular cuja densidade e amplitude são especificadas através dos parâmetros globais *NGRID*, *Grid step* e *Grid ratio*, em busca das primeiras curvas críticas, que inicializam o conjunto $\mathcal{C}_\bullet$. Para cada uma dessas curvas, determinamos seu sentido de dobra, calculamos suas cúspides e as classificamos em interna ou externa. Inicializamos o conjunto $F(\mathcal{C}_\bullet)$ que contém a imagem de cada curva em $\mathcal{C}_\bullet$, computamos seus pontos de intersecção e, para cada curva em $F(\mathcal{C}_\bullet)$, calculamos sua giração. A seguir, o par $\mathcal{C}_\bullet$-$F(\mathcal{C}_\bullet)$ é submetido a três tipos de testes, que buscam estabelecer se $\mathcal{C}_\bullet = \mathcal{C}$. Cada teste é efetuado em cada placa de $\mathcal{C}_\bullet$, e se algum desses testes falha em determinada placa, é porque existe ao menos uma curva crítica que não está presente em $\mathcal{C}_\bullet$. Então, na placa onde ocorreu a falha, o programa efetuará uma busca para encontrar uma nova curva crítica, que é acrescentada ao conjunto $\mathcal{C}_\bullet$.

Quando todos os testes dão certo, o **wx2x2** entende que $\mathcal{C}_\bullet = \mathcal{C}$. Os resultados teóricos apenas caracterizam os conjuntos críticos $\mathcal{C}$ e suas imagens $F(\mathcal{C})$ de funções cordatas. Isso não garante que o conjunto $\mathcal{C}_\bullet$ e sua imagem $F(\mathcal{C}_\bullet)$ são de fato o conjunto crítico completo da função dada F: eles podem ser o conjunto crítico de uma outra função $\hat{F}$. O problema, em um certo sentido, é insolúvel: curvas críticas podem ser adicionadas em qualquer região arbitrariamente pequena de um ponto regular de uma função F, sem alterar

a função fora desta região. Qualquer método numérico, que só se permita calcular uma função em um número finito de pontos, não pode alcançar a certeza de ter encontrado todas as curvas críticas. Os testes, entretanto, são muito expressivos: é difícil esconder curvas críticas deles. Ainda assim, existem parâmetros globais que aumentam o refinamento da busca por curvas críticas, entre os quais *Curve Sharpness*, *Maximum Step*, *Minimum Step*.

5.3 Detectando as primeiras curvas críticas

A seguir, descrevemos como uma curva Γ é representada no programa. Os atributos da classe CriticalCurve são:

1. *index*: inteiro que rotula Γ, começando por 0;
2. *domBoundingBox*: coordenadas extremas em x e y dos pontos de Γ;
3. *imgBoundingBox*: coordenadas extremas em x e y dos pontos de $F(\Gamma)$;
4. *orientation*: informa o sentido de dobra em Γ;
5. *domPoints* e *imgPoints*: pontos da discretização de Γ e suas imagens;
6. *numPoints*: o total de pontos da discretização em Γ;
7. *cusps*: dá a localização das cúspides, internas e externas;
8. *numInCusps* e *numOutCusps*: contam cúspides internas e externas;
9. *intersections*: lista os pontos de intersecção das curvas na imagem;
10. *domCriticalsX* e *domCriticalsY*, *imgCriticalsX* e *imgCriticalsY*: listam os extremos locais das coordenadas x e y em Γ e $F(\Gamma)$;
11. *turningNumber*: a giração de $F(\Gamma)$;
12. *winding(q)*: uma rotina que calcula o número de rotação de Γ em torno do ponto q fornecido como entrada;

Todas as curvas críticas calculadas pelo programa, junto com suas imagens, são guardadas num vetor que está disponível a qualquer momento.

Agora, passamos a descrever o processo de busca por curvas críticas. No início da rotina *study_function()*, percorremos uma malha quadrangular centrada na origem cuja geometria foi especificada através dos parâmetros globais *NGRID*, *Grid step* e *Grid ratio*. Procuram-se pontos consecutivos p_i e p_{i+1} nos quais os sinais de $\det DF$ nestes pontos sejam diferentes, o que indica

a existência de um ponto crítico no segmento $\overline{p_i p_{i+1}}$. Se o programa ainda não registrou (pela rotina *test_stick()*) a existência de um ponto crítico em $\overline{p_i p_{i+1}}$, a rotina *find_cr_point()* calcula ali um ponto crítico novo r, que serve de ponto de partida para *make_cr_curve()*, que por sua vez realiza a construção da curva crítica por r.

A rotina *make_cr_curve()* cria um objeto da classe CriticalCurve e usa *next_zero()* várias vezes, para gerar seqüencialmente, por argumentos de análise local, pontos críticos que são adicionados ao vetor *domPoints* do objeto. Assim constrói uma curva crítica que será orientada pelo sentido de dobra, quando completada. As rotinas *next_zero()* e *find_cr_point()*, juntas, implementam um método de continuação numérica, do tipo preditor-corretor, onde a etapa preditora é de passo variável. A variação do passo é necessária para levar em conta o fato freqüente de que na mesma função existem curvas críticas de tamanhos e afastamentos muito diferentes. O usuário pode interferir nessa etapa ajustando os parâmetros globais *Curve Sharpness*, *Minimum step*, *Maximum step* e *Min step in make_cr_curve()*, existentes na opção *Preferences* do **wx2x2**.

Essencialmente, *make_cr_curve()* realiza as seguintes tarefas:

1. Prepara as condições iniciais para invocar *next_zero()*.

2. Determina o término do cálculo da curva crítica (à medida que *next_zero()* vai calculando pontos críticos supostamente novos, é necessário garantir que os novos pontos não sejam discretizações de um arco da curva já percorrido).

3. Determina o sentido de dobra da curva, estabelecendo se a curva crítica foi construída no sentido anti-horário ou horário. Note que essa informação é global: ela está disponível só quando a curva está completa. A orientação é obtida examinando a vizinhança de um ponto cuja coordenada x é máxima.

Para o funcionamento de *next_zero()*, são necessárias algumas construções geométricas. O vetor tangente $r_t(r)$ à uma curva crítica num ponto $r = (x_0, y_0)$ é obtido girando o vetor gradiente $\nabla g(p)$ de 90° em sentido horário, onde $g(x, y) = \det(DF(x, y))$, para que a curva assim construída fique orientada em sentido de dobra. Como $g(x_0, y_0) = 0$, $\nabla g(r)$ é aproximado por

$$\begin{aligned} \nabla g(r) &= (g_x(r), g_y(r)) \\ &\approx \left(\frac{g(x_0 + \Delta, y_0)}{\Delta}, \frac{g(x_0, y_0 + \Delta)}{\Delta} \right), \ \Delta > 0 \text{ pequeno.} \end{aligned}$$

Daí,

$$r_t(r) = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \nabla g(r) = \left(\frac{g(x_0, y_0 + \Delta)}{\Delta}, -\frac{g(x_0 + \Delta, y_0)}{\Delta} \right)$$

A rotina *next_zero()* recebe dois pontos consecutivos de uma curva crítica e, a partir deles, calcula um novo ponto. Para realizar a primeira chamada à *next_zero()*, como *make_cr_curve* dispõe de apenas um ponto crítico r usamos o vetor tangente $r_t(r)$ para calcular um quase ponto crítico $r_{-1} = r - step \cdot r_t(p)$ e, assim realizamos a primeira chamada à rotina *next_zero()* passando os pontos r_{-1} e r. O valor *step* na definição de r_{-1} é calculado a partir de uma análise local em r para levar em consideração a curvatura da curva crítica em r: quanto maior a curvatura, menor será o valor de *step* (seu valor deve ser maior ou igual a *Min step in make_cr_curve()*).

Agora explicaremos como, na rotina *next_zero()*, obtemos um novo ponto crítico a partir de outros dois consecutivos. Sua implementação se baseia em duas etapas (Figura 5.1):

Etapa Preditora: Seja γ uma curva crítica de F contendo dois pontos r_0, r_1 calculados seqüencialmente. Calculamos $q = r_1 + sec$ a partir do vetor secante $sec = 2 \cdot step \cdot \frac{r_1 - r_0}{|r_1 - r_0|}$ e os pontos $q_a = q + R(-90) \cdot sec$ e $q_b = q + R(+90) \cdot sec$, onde $R(\theta)$ representa a matriz de rotação por θ graus. Se os sinais de $\det DF$ em q_a e q_b forem distintos, refinamos o vetor *sec* levando em conta o parâmetro global *Curve Sharpness*, para fins de robustez, e depois executamos a Etapa Corretora. Caso contrário, o vetor *sec* sofre uma contração de fator 0.5 e os pontos q, q_a e q_b são atualizados. Se num número de pré-determinando de vezes, não obtivermos sucesso em encontrar q_a e q_b com sinais distintos, a construção da curva crítica passando por r é abortada.

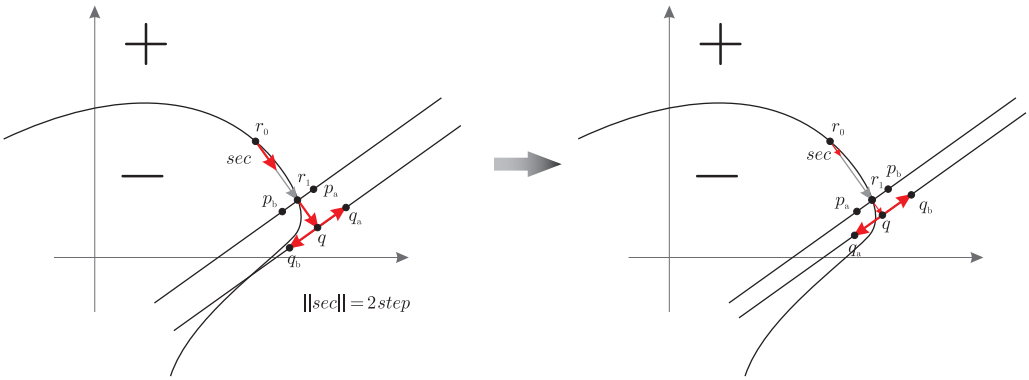


Figura 5.1: Aqui é necessário reduzir *step*

Etapa Corretora: a rotina *find_cr_point()* é chamada para calcular um ponto crítico no segmento $\overline{q_a q_b}$

A rotina *find_cr_point()*, crucial no funcionamento de *next_zero()*, recebe dois pontos m e n nos quais os sinais de $\det DF$ são distintos e retorna um ponto crítico no segmento $\overline{mn}$. O algoritmo de localização de zeros empregado é o *método de Brent* (ou *Van Wijngaarden-Dekker-Brent*), que combina os métodos da bisseção, da secante e interpolação quadrática inversa. É um método robusto, no sentido que desempenha pelo menos tão bem quanto bisseção e ainda assim pode ser tão rápido quanto métodos mais instáveis. Na maioria das iterações, o algoritmo emprega os métodos da secante ou interpolação quadrática inversa para aumentar a velocidade de convergência. O método de Brent é recomendado na bibliografia para calcular zeros de uma função real de variável real quando apenas seus valores estão disponíveis, e não suas derivadas ([PR]).

Em *next_zero()*, na Etapa Corretora, a rotina *find_cr_point()* é chamada recebendo como parâmetros m e n os pontos q_a e q ou q_b e q, respectivamente se $\det(DF(q_a)) \cdot \det(DF(q)) < 0$ ou $\det(DF(q_b)) \cdot \det(DF(q)) < 0$. E, assim, o ponto crítico procurado é da forma $r = (1 - t_0) \cdot m + t_0 \cdot n$, onde t_0 um zero para a função $\det(DF((1 - t) \cdot m + t \cdot n))$ em $[0, 1]$.

Ao término da construção da nova curva crítica marcamos os segmentos da malha que foram atravessados por essa nova curva (isso é feito para evitar repetições na busca por outras curvas) e retomamos a travessia da malha. Ao terminar de percorrer toda a malha original, algumas curvas críticas são encontradas, formando o conjunto $\mathcal{C}_\bullet$, representado por um vetor de objetos do tipo CriticalCurve.

5.3.1
Calculando alguns atributos de CriticalCurve

Explicamos aqui como alimentamos alguns atributos da classe CriticalCurve. À medida que uma curva crítica nova é construída, verifica-se as propriedades extremais em relação às coordenadas x e y tanto da curva quanto dos pontos de sua imagem: assim são gerados *domCriticalsX*, *domCriticalsY*, *imgCriticalsX* e *imgCriticalsY*, assim como *domBoundingBox* e *imgBoundingBox*, dimensões dos retângulos mínimos que contém as curvas. Com os pontos extremos determinados, temos em mãos a decomposição da curva crítica e sua imagem em arcos bimonotônicos, que consistem de arcos de pontos entre dois pontos extremos consecutivos. (Figura 5.2).

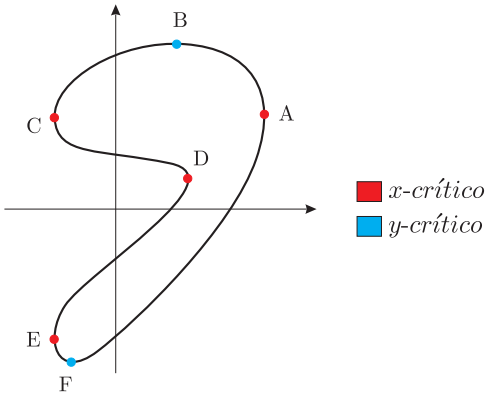


Figura 5.2: Curva dividida em arcos bimonotônicos

Identificamos também pontos candidatos a vizinhos de cúspides, isto é, pontos p_1, p_2 e p_3 no domínio para os quais há uma súbita mudança de direção entre os vetores $F(p_1) - F(p_2)$ e $F(p_2) - F(p_3)$ (Figura 5.3). Vizinhos bem calculados a uma cúspide são necessários para a classificação interna-externa, depois que o sentido de dobra é calculado. A rotina *refine_cusp()* refina o trecho de arco crítico determinado por p_1, p_2 e p_3 para que a rotina *seek_cusp()* localize e classifique uma cúspide nesse arco (caso exista) e, logo após, os atributos *cusps*, *numInCusps* e *numOutCusps* da curva crítica são atualizados.

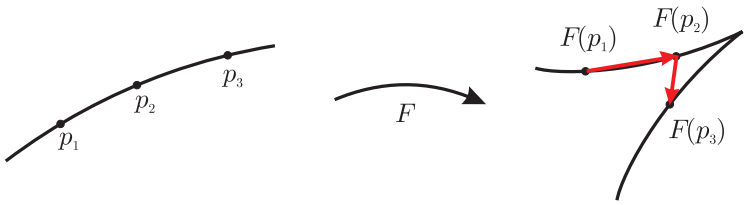
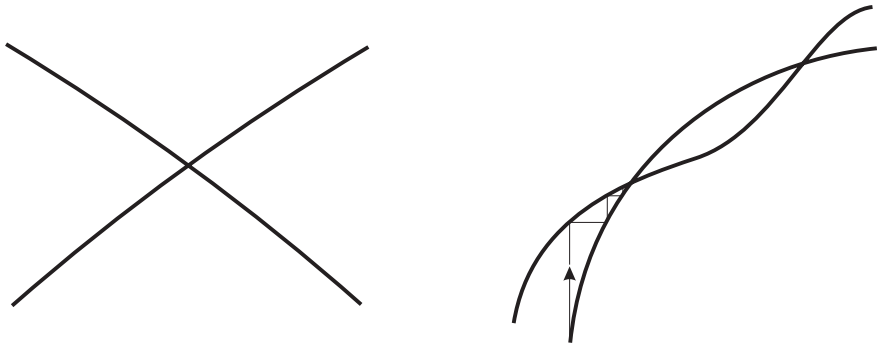


Figura 5.3: Localizando cúspides

Procuramos também por pontos de intersecção no conjunto $F(\mathcal{C}_\bullet)$, uma primeira utilização da decomposição das curvas em arcos bimonotônicos. Para cada curva em $F(\mathcal{C}_\bullet)$ procuramos por pontos de auto-intersecção e pontos de intersecção com outras curvas. Para identificar nas listas de pontos do domínio e da imagem os vizinhos de pontos de intersecção procedemos da seguinte maneira. Dadas duas imagens de curvas críticas, verificamos inicialmente se seus *bounding boxes* têm intersecção não vazia. Neste caso, passamos a considerar todos os pares de arcos bimonotônicos, um em cada curva. Mais uma vez, se os bounding boxes desses arcos não se interceptam, não há nada a fazer.



(a) (b)

Figura 5.4: Intersecções entre arcos bimonotônicos

Caso contrário, existem dois sub-casos que são ilustrados na Figura 5.4. No caso 5.4a, uma pesquisa binária nas listas de pontos dos arcos encontra o ponto de intersecção (se existir). No caso 5.4b, os pontos de intersecção são encontrados pelo algoritmo sugerido na Figura. Na realidade, esses dois métodos fornecem apenas vizinhos ao ponto de intersecção e, para obter precisamente a intersecção, inserida nas curvas tanto do domínio quanto na imagem, procedemos a uma análise numérica local.

5.3.2 Girações e rotações

Aqui explicaremos como calcular números de rotação, girações e relações de inclusão entre as curvas críticas de $\mathcal{C}_\bullet$. Essa informação é necessária para a implementação da rotina *belongsToPlaque()*, uma função que decide em que placa (do domínio ou da imagem) está um ponto.

Descrevemos o cálculo do número de rotação de uma curva γ em torno da origem (uma translação reduz o caso geral a esse). Na rotina *winding()*, percorremos os pontos extremos dos arcos bimonotônicos da curva e verificamos mudanças de quadrante entre pontos extremos consecutivos. Mudanças entre quadrantes vizinhos somam ou subtraem um quarto de volta ao número de rotação de acordo se a mudança ocorreu no sentido anti-horário ou horário, respectivamente. Se dois pontos extremos de um arco bimonotônico estão em quadrantes opostos, soma-se ou subtrai-se meia volta, mas agora é necessário invocar a rotina *x_to_pt()* para determinar se a origem está abaixo ou acima do arco. Na figura abaixo, as frações indicam as contribuições de cada arco bimonotônico para o cálculo do número de rotação da curva em torno à origem.

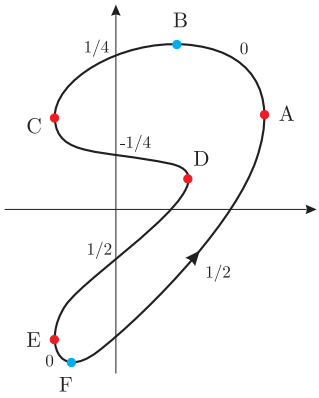


Figura 5.5: Uma volta em torno à origem

Para calcular girações de curvas do contradomínio também utilizamos a decomposição da curva em arcos bimonotônicos. Em pontos extremos suaves, o vetor tangente claramente pertence a um dos quatro raios determinados pelos eixos coordenados. Quando os arcos se encontram na imagem de uma cúspide somamos meia volta à giração. Assim podemos inicializar o atributo *turningNumber* (a giração) para cada curva em $F(\mathcal{C}_\bullet)$.

5.3.3
$\mathcal{C}_\bullet = \mathcal{C}$?

Três tipos de testes — de *coerência*, de *contagem* e de *Blank-Troyer* — são aplicados ao conjunto $\mathcal{C}_\bullet$ para determinar se este conjunto está completo, isto é, se todas as curvas críticas foram localizadas. Esses testes são realizados em cada placa de $\mathcal{C}_\bullet$: quando temos uma reprovação em alguma placa, o programa volta a procurar outras curvas críticas nessa placa.

O teste de coerência consiste em verificar se duas curvas vizinhas por inclusão possuem orientações contrárias: afinal, percorrer a curva no sentido de dobra faz com que a região à direita tenha jacobiana com determinante positivo. Isso é feito comparando o atributo *orientation* das curvas durante uma travessia do grafo de adjacência.

Os testes de contagem são as verificações do Teorema 3.7. Para cada placa X do domínio, todos os ingredientes necessários para aplicar esse teste foram calculados nas seções anteriores, a saber: grau da função, girações e o número de cúspides internas e externas.

O teste de Blank-Troyer é implementado conforme descrito em [MST1], e baseia-se na caracterização de conjuntos críticos e suas imagens. Esse teste pode ser habilitado ou desabilitado definindo o parâmetro global *Blank*, na opção *Preferences* do **wx2x2**, conforme descrito na seção 4.4.

5.3.4
Procurando outras curvas críticas

Quando uma placa X de $\mathcal{C}_\bullet$ não passa num dos testes mencionados acima é porque existe alguma curva crítica ainda não detectada no interior de X. Passamos então a procurar por dois pontos p_0 e p_1 em X nos quais $\det DF$ tem sinais opostos. A partir deste segmento, a rotina *find_cr_point()* calcula um ponto crítico, que é estudado pela rotina *test_san()* para determinar sua pertinência a alguma curva crítica já calculada. Se o ponto é considerado novo, *make_cr_curve()* constrói a nova curva.

Uma reprovação no teste de coerência é tratada da forma descrita a seguir. Suponhamos X limitada. Curvas na fronteira de X são classificadas como *boas* ou *más*. A fronteira exterior é boa. Fronteiras interiores são boas se sua orientação dada pelo sentido de dobra é oposta à da fronteira exterior de X; as demais curvas são más. A reprovação de X no teste de coerência significa que existe pelo menos uma curva má. Entre as curvas más, seja p' o ponto com coordenada y mínima. Dentre os pontos *em curvas boas* com coordenada y menor ou igual à de p', tomamos p'' minimizando a distância a p' na métrica $d((x_0, y_0), (x_1, y_1)) = \max\{|x_0 - x_1|, |y_0 - y_1|\}$, obtido como em (I) abaixo. Obtenha agora no segmento $\overline{p'p''}$ dois pontos interiores p_0 e p_1 vizinhos aos vértices, usados agora como parâmetros para *find_cr_point()*: o ponto crítico encontrado será aprovado por *test_san()*, a menos de algum problema numérico.

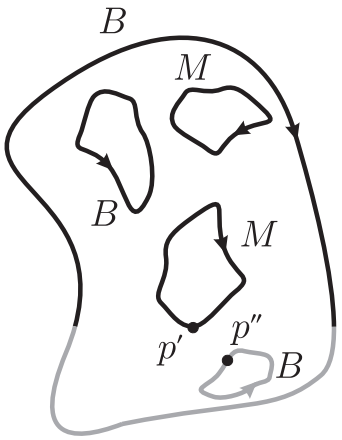


Figura 5.6: Encontrando uma curva crítica nova

Na métrica definida acima, o ponto mais próximo a p num arco bimonotônico $\widehat{\mathrm{AB}}$ é A, B ou uma intersecção de $\widehat{\mathrm{AB}}$ com uma das duas retas passando por p com inclinação ± 1. Um simples teste verificando a localização do *bounding box* do arco nos diz em qual caso no encontramos. Para procu-

rar a interseção com uma reta, uma pesquisa binária entre os pontos do arco desempenha bem.

Suponhamos agora que a placa X foi aprovada no teste de coerência, mas reprovada num dos outros dois testes. Em X, o determinante da jacobiana em pontos próximos à fronteira têm um sinal bem definido, digamos positivo. Estamos procurando por pontos em X com determinante da jacobiana negativa, a partir do qual *find_cr_point()* gera um novo ponto candidato a inicializar *make_cr_curve()*.

Para começar, geramos aleatoriamente um ponto p em X. Isso se faz escolhendo pontos dentro do *bounding box* da fronteira exterior e verificando sua pertinência a X com a rotina *belongsToPlaque()*. Se $p \in X$ satisfaz $\det(DF(p)) < 0$, basta invocar *find_cr_point()*. Se isso não acontece, procuramos diminuir o determinante ao longo de uma reta por p alinhada com uma discretização de $\text{grad}(\det(DF))(p)$. A busca se repete ao longo de segmentos sempre em X, por um certo número de vezes. Se o ponto desejado não é encontrado, outro ponto p é sorteado. O processo é cotado pelo parâmetro global *Number of tries*.

5.4 Invertendo um ponto

Agora estamos prontos para descrever o processo de resolução numérica da equação

$$F(x) = q, \qquad x,\, q \in \mathbb{R}^2. \tag{5-1}$$

Uma das tarefas principais é a inversão de F por um *método de continuação* ao longo de um segmento $\overline{q_0 q_1}$ na imagem para o qual uma pré-imagem p_0 de q_0 é conhecida. O segmento é discretizado e novas pré-imagens são obtidas por um método de Euler-Newton a partir das pré-imagens anteriores ([AG]). No **wx2x2**, a rotina *cont_pre_im()* implementa esse método, tendo como parâmetros de entrada os pontos q_0, q_1 e p_0. A rotina retorna os pontos p_1, p_* e um indicador que informa um dos três resultados a seguir:

a) a rotina foi bem-sucedida e a pré-imagem desejada é p_1;

b) a rotina descobriu que o processo de continuação não vai além de um ponto p_*. Isto ocorre quando o segmento $\overline{q_0 q_1}$ intercepta $F(\mathcal{C})$, sendo $q_* = F(p_*)$ a intersecção;

c) ocorrência de um erro numérico na rotina, indicada na área de log.

A resolução de (5-1) é realizada em três etapas:

a) cálculo de todas as pré-imagens de alguns pontos;

b) obtenção de um caminho β, em forma de "L" (veja Figura 5.8), composto por dois segmentos de reta, β_0 e β_1, satisfazendo certos critérios descritos abaixo;

c) inversão dos segmentos β_0 e β_1.

5.4.1
Calculando todas as pré-imagens de alguns pontos

A rotina *init_invert()* é responsável por esta tarefa. Inicialmente procuramos um *bounding box* $B_{\mathcal{C}}$ para as curvas de $\mathcal{C}$ e outro $B_{F(\mathcal{C})}$ para as curvas de $F(\mathcal{C})$. Tomamos a seguir um ponto p fora de $B_{\mathcal{C}}$ com imagem $q = F(p)$ fora de $B_{F(\mathcal{C})}$. Isto é possível porque F é cordata. Em outras palavras, q está na placa ilimitada de $F(\mathcal{C})$. A partir de q, completamos um quadrado como na Figura 5.7).

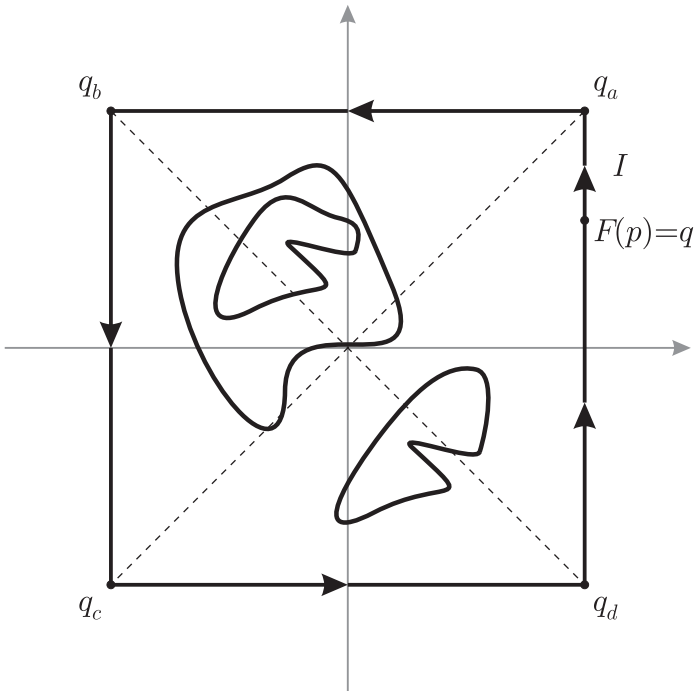


Figura 5.7:

Obtemos uma pré-imagem p_a de q_a, através da rotina *cont_pre_im()*, usada invertendo por continuação ao longo do segmento I, com a condição inicial p.

Depois, usando p_a como uma condição inicial para inverter ao longo de $\overline{q_a q_b}$, obtemos uma pré-imagem de q_b. Invertendo os outros lados do quadrado, obtemos pré-imagens de q_c e q_d e uma outra pré-imagem p'_a de q_a. Repetimos então o ciclo de inversões ao longo dos lados do quadrado a partir de p'_a, até obtermos o número total de pré-imagens de cada um desses quatro pontos,

que deve ser, segundo o Corolário 3.4, igual a $|\deg(F)|$. Aliás, $\deg(F)$ já foi calculado pela rotina *find_degree()*. Se este processo falhar por razões numéricas, começamos novamente selecionado um ponto p mais longe da origem. No programa, a cada falha aplicamos uma escala de 1.2 ao ponto p anterior, ou seja, $p \leftarrow 1.2 \cdot p$. Pelo Teorema 3.5, este processo deve funcionar para p suficientemente longe da origem.

Em conclusão, temos quatro *pontos resolvidos* q_a, q_b, q_c e q_d, cujas pré-imagens são todas conhecidas. Estes pontos e suas pré-imagens são armazenados no *banco de pontos resolvidos* e jamais são removidos. O banco é útil para a execução de inversões de pontos genéricos: dado q_1 na imagem, selecionamos um ponto q_0 no banco de pontos resolvidos, ligamos q_0 a q_1 por um caminho em forma de "L" e invertemos, por continuação, de q_0 até q_1, algumas pré-imagens de q_1 a partir das pré-imagens de q_0.

Para exemplificar, vamos considerar novamente a função $\widetilde{F}(z) = z^2 + \bar{z}$. A inversão ao longo dos segmentos β_0 e β_1 saindo de q_0 a partir das pré-imagens p_0^0 e p_0^1 dão origem a duas pré-imagens de q_1, p_1^0 e p_1^1. É claro, entretanto, que duas pré-imagens estão faltando, como deixa claro a figura. Essa é a dificuldade considerada a seguir.

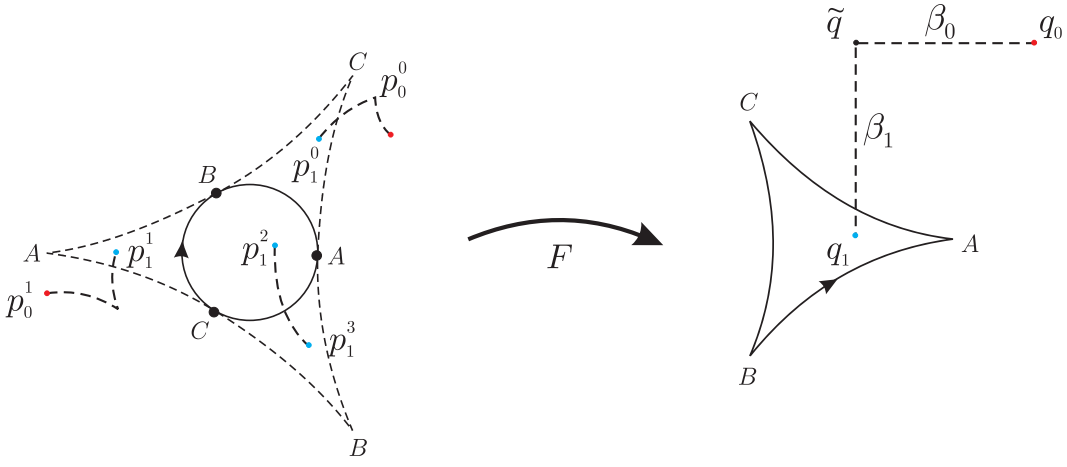


Figura 5.8: Obtendo todas as pré-imagens de pontos remotos

5.4.2
Escolhendo caminhos e invertendo

É inevitável, entretanto, que o caminho em "L" ligando um ponto resolvido q_0 a um ponto a inverter q_1 cruze às vezes o conjunto $F(\mathcal{C})$. Mais precisamente, suponha que um segmento orientado $\beta = \overline{\alpha\omega}$ desse caminho L encontre $F(\mathcal{C})$ num ponto q_*, que divide β em dois segmentos menores, $\overline{\alpha q_*}$ e $\overline{q_*\omega}$. O número de pré-imagens de $\overline{\alpha q_*} - q_*$ e o de $\overline{q_*\omega} - q_*$ diferem de 2, pelo Teorema 3.6. Quando o número de pré-imagens diminui, é porque um

par de arcos no domínio obtido por inversão de $\overline{\alpha q_*} - q_*$ estão convergindo para o mesmo ponto p_* no domínio, para o qual vale $F(p_*) = q_*$. O programa identifica esses arcos e interrompe a inversão de β para esses arcos.

Quando o número de pré-imagens aumenta, é porque dois novos arcos devem surgir na inversão de $\overline{q_*\omega} - q_*$. Para identificá-los, o programa encontra pontos críticos cujas imagens estão próximas a q_* e, por um refinamento tipo Newton, obtém p_* para o qual $F(p_*) = q_*$. Como cúspides foram evitadas, p_* é uma dobra. Agora, a partir de p_*, a análise numérica faz uso da forma local das dobras, implementada na rotina *beget_preim()*. O método é mais sutil do que simplesmente uma iteração do método de Euler-Newton: a condição inicial para a continuação da inversão de $\overline{q_*\omega} - q_*$ não é um ponto onde a Jacobiana da função é inversível.

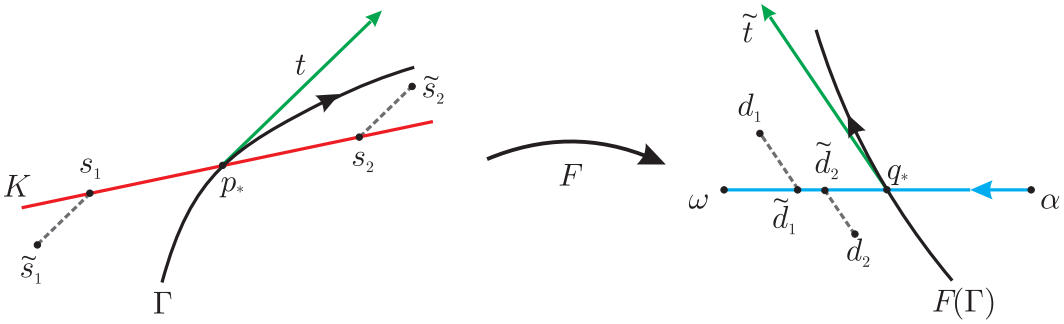


Figura 5.9: Nascem duas pré-imagens

O programa calcula o núcleo K de $DF(p_*)$, indicado em vermelho na figura acima, e o vetor tangente t à curva crítica Γ, e escolhe dois pontos s_1 e s_2 sobre a reta $p_* + K$ próximos a $\mathcal{C}$, um de cada lado de Γ. Pela expansão de Taylor de F em torno de p_*,

$$F(s_i) = F(p_*) + \underbrace{DF(p_*)[s_i - p_*]^T}_{0,\text{ pois } p_* \in K} + O(||s_i - p_*||^2) \approx q_*, \quad i = 1, 2.$$

Assim, se s_i distam ϵ de p_*, as imagens $d_i = F(s_i)$ distam $O(\epsilon^2)$ de q_* e estão do mesmo lado de $F(\Gamma)$, mas não estão em β. Queremos encontrar pontos $\tilde{s}_i$ próximos a s_i cujas imagens estejam de fato em β. Usar s_i como condição inicial para um método iterativo revelou-se instável numericamente. Em vez disso, o programa escolhe uma condição inicial mais refinada, descrita a seguir. Note que o vetor $\tilde{t} = DF(p_*)t$ é tangente a $F(\Gamma)$. Determine λ_i tal que o ponto $\tilde{d}_i = d_i + \lambda_i \tilde{t}$ pertença a β. Agora, use $\tilde{s}_i = s_i + \lambda_i t$ como condição inicial para resolver $F(\tilde{s}_i) = \tilde{d}_i$.

Finalmente, o processo de inversão de $\overline{q_*\omega} - q_*$ é substituído pela inversão de $\overline{\tilde{d}_i\omega}$, com condições iniciais $\tilde{s}_i$.

Agora, que vimos como o programa procede ao inverter segmentos que trespassam $F(\mathcal{C})$ em dobras, voltamos à escolha de caminhos em forma de L apropriados. Temos que ligar um ponto q_1 na imagem de F, cujas pré-imagens queremos calcular, a um ponto resolvido q_0. O caminho, orientado de q_0 a q_1, é formado por dois segmentos β_0 e β_1, sendo que um é horizontal e outro, vertical. A Figura 5.8 exibe um tal caminho e suas pré-imagens para a função $\widetilde{F}(z) = z^2 + \bar{z}$. O ponto q_0 é obtido no banco de pontos resolvidos, entre os *Number of neigh* (um parâmetro global) mais próximos de q_1. Para escolher entre esses candidatos, consideram-se as seguintes propriedades:

(a) os segmentos devem passar longe de cúspides e de extremos locais das coordenadas x e y em $F(\mathcal{C})$;

(b) ambos os segmentos devem ter poucas intersecções com $F(\mathcal{C})$;

(c) segmentos curtos são preferidos.

A inversão de pontos próximos a cúspides e a extremos locais pode gerar instabilidade numérica no método. Assim, as condições (a) e (c) são naturais. Aqui, a distância entre pontos é ainda definida pela norma sup, isto é, pela soma dos segmentos do L. A rotina *find_L()* é empregada para selecionar o caminho adequado considerando as propriedades acima. Essa rotina faz chamadas às subrotinas *x_censor()* e *y_censor()* para verificar se os segmentos de reta β_0 e β_1 estão próximos de cúspides ou de extremos locais, e também às subrotinas *x_to_ints()* e *y_to_ints()* para calcular, ordenar e contar as intersecções entre segmentos β e o conjunto $F(\mathcal{C})$.

Se a inversão do segmento β ocorreu com sucesso, decidimos se q_1 é incorporado ao banco de pontos resolvidos, verificando o valor do parâmetro global *Banksys*. Se este valor for -1, q_1 e suas pré-imagens são incorporados; se for 1, verificamos se houve intersecção de β com $F(\mathcal{C})$ e somente em caso afirmativo armazenamos q_1; se *Banksys* for 0, nada é armazenado. Percebemos aqui a possibilidade de mantermos pontos resolvidos em várias placas de $F(\mathcal{C})$ que poderão se úteis na inversão de pontos situados nessas placas. O banco de pontos é implementado com a estrutura de dados *fila* (*FIFO - First In, First Out*) possuindo um tamanho fixo e, quando o banco está cheio, ao adicionarmos um novo ponto removemos o ponto mais antigo.

Resumindo, para proceder à inversão do ponto q_1, escolhemos um L apropriado em *find_L()* e realizamos a inversão por continuação ao longo do L, levando em conta as alternativas a considerar no caso de um segmento encontrar $F(\mathcal{C})$.

5.4.3
Calculando a flor

A flor é o conjunto $\mathcal{F} = F^{-1}(F(C))$. Para gerá-la, o **wx2x2** considera os *arcos maximais* de $F(C)$, ou seja, as componentes conexas de $F(C)$ menos os pontos de intersecção e imagens de cúspides. A rotina *make_flor()* percorre todas as curvas que compõem $F(C)$ e envia cada arco maximal da curva à rotina *make_pre_arc()* — responsável por inverter arcos maximais.

A rotina *make_pre_arc()* tem parâmetros inteiros que informam os índices da imagem de curva crítica e dos pontos inicial e final do arco maximal, respectivamente, j_{crv}, j_a e j_z. A partir daí, seleciona o ponto P_m de índice médio ($j_m = (j_a + j_z)/2$) e computa um ponto P perto de P_m, mas do lado da curva que possui menos pré-imagens. Calcula-se as k pré-imagens de P, e pelo Teorema 3.6, sabemos que P_m possui $k + 1$ pré-imagens. Uma das pré-imagens de P_m já é conhecida: é o ponto de índice j_m na curva crítica j_{crv}. Para encontrar as outras k pré-imagens de P_m, realiza-se uma inversão ao longo do segmento de reta que liga P à P_m para cada pré-imagem de P como condição inicial. Agora, inicia-se a inversão do arco maximal em P_m e inverte-se cada uma de suas metades. A Figura 5.10 ilustra o processo de criação de $\mathcal{F}$ para a função $\widetilde{F}(z) = z^2 + \bar{z}$.

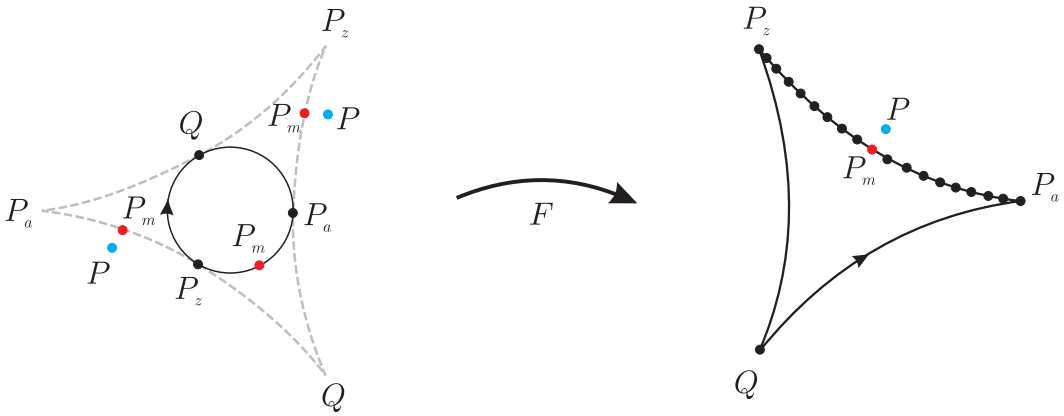
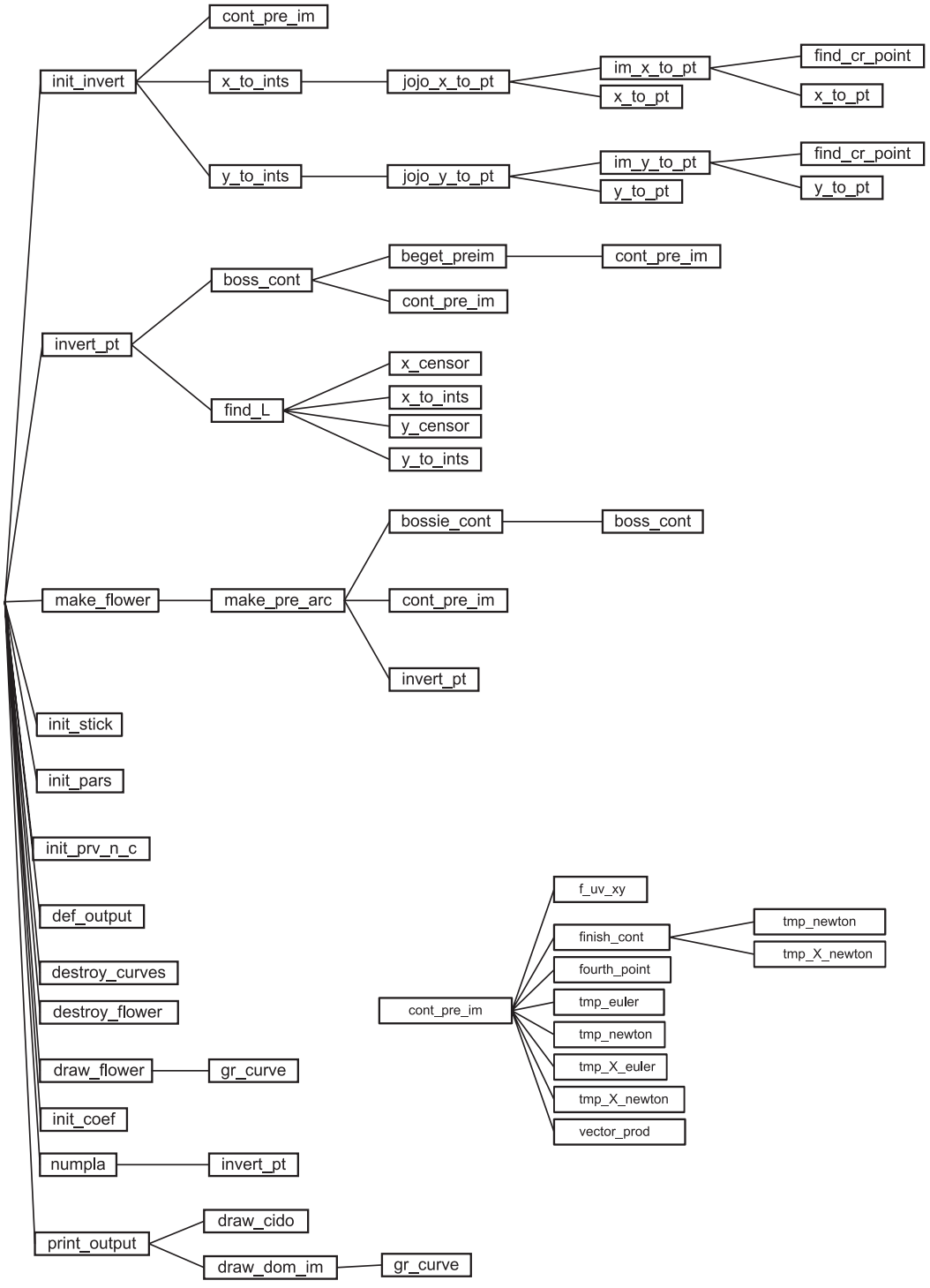
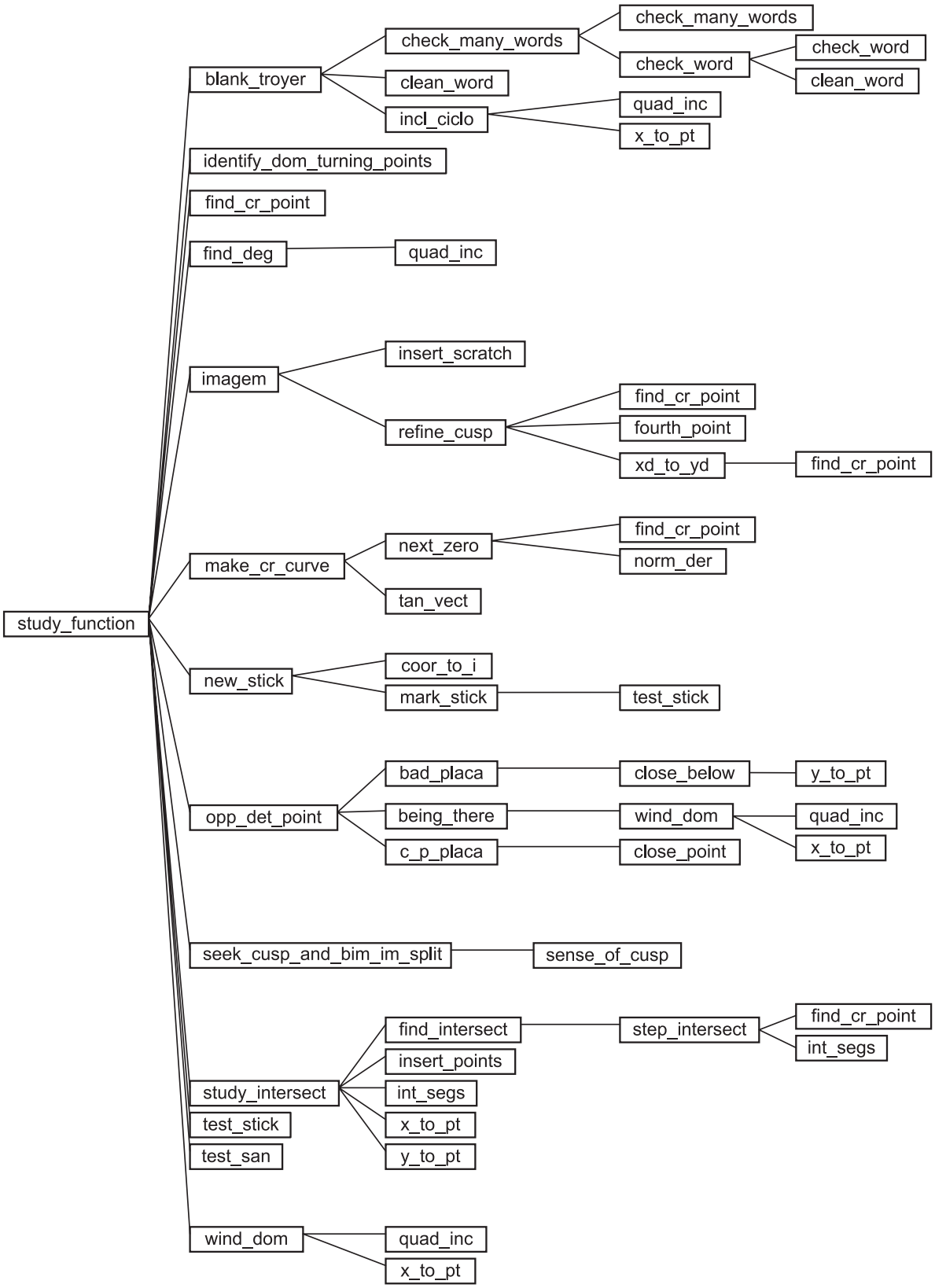


Figura 5.10: Invertendo um arco maximal

A análise numérica torna-se especialmente delicada quando os pontos a inverter se aproximam de cúspides. Flores são mais fáceis de calcular a partir de grandes bancos de pontos resolvidos. Freqüentemente, a flor de uma função é visualmente muito complicada: nem sempre vale a pena calculá-la. Por outro lado, é confortador que o processo de inversão de pontos não depende da construção da flor.

A
Árvore de chamada de rotinas





Referências Bibliográficas

[AG] ALLGOWER, E.L., GEORG, K. *Numerical continuation methods: an introduction*. Springer-Verlag, New York, 1991.

[BE] BERGER, M. S. *Nonlinearity and functional analysis: lectures on nonlinear problems in mathematical analysis*. Academic Press. New York, 1977.

[LAN] LANG, Serge. *Real Analysis*. 2ª ed. Yale University, New Haven, 1983.

[MI] MILNOR, J. W. *Topology from the differentiable viewpoint*. University Press of Virginia, Charlottesville, 1969.

[MST1] MALTA, I. P., SALDANHA, N. C., TOMEI, C. *The numerical inversion of functions from the plane to the plane*. Math. of Computation 216, vol. 65, 1996, 1531-1552.

[MST2] MALTA, I. P., SALDANHA, N. C., TOMEI, C. *Geometria e Análise Numérica de Funções do Plano no Plano*. 19º Colóquio Brasileiro de Matemática, Impa, 1993.

[PR] PRESS, W. H. et al. *Numerical Recipes in C: The Art of Scientific Computing*. 2ª ed. Cambridge University Press, 1992.

[W] WHITNEY, H. *On singularities of mappings of Euclidean spaces*. I: Mappings of the plane into the plane, Ann. of Math. 62, 1981, 374-410.

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo