UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE MATEMÁTICA E ESTATÍSTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM MATEMÁTICA APLICADA

Computação Paralela com OpenMP
e aplicações a EDPs não lineares

por

ROBERTO KAI HSI HOO

Dissertação submetida como requisito parcial
para a obtenção do grau de

Mestre em Matemática Aplicada

Prof. Paulo Ricardo de Avila Zingano
Orientador

Porto Alegre, 21 de Dezembro de 2018

CIP - Catalogação na Publicação

Hoo, Roberto Kai Hsi
Computação Paralela com OpenMP e aplicações a EDPs não lineares / Roberto Kai Hsi Hoo. -- 2018.
123 f.
Orientador: Paulo Ricardo de Avila Zingano.

Dissertação (Mestrado) -- Universidade Federal do Rio Grande do Sul, Instituto de Matemática e Estatística, Programa de Pós-Graduação em Matemática Aplicada, Porto Alegre, BR-RS, 2018.

1. Função p-harmônica e equação evolutiva do p-Laplaciano. 2. Método das diferenças finitas para p-Laplaciano. 3. Computação científica com Fortran e Matlab. 4. Programação serial e paralela de alto desempenho com OpenMP. 5. Resolução de Equação Diferencial Parcial não linear via computador. I. Zingano, Paulo Ricardo de Avila, orient. II. Título.

Elaborada pelo Sistema de Geração Automática de Ficha Catalográfica da UFRGS com os dados fornecidos pelo(a) autor(a).

Computação Paralela com OpenMP
e aplicações a EDPs não lineares

por

Roberto Kai Hsi Hoo

Trabalho submetido ao Programa da Pós-Graduação em Matemática Aplicada do Instituto de Matemática e Estatística da Universidade Federal do Rio Grande do Sul, como requisito parcial para a obtenção do grau de

Mestre em Matemática Aplicada

Linha de Pesquisa: Computação Científica
Orientador: Prof. Dr. Paulo Ricardo de Avila Zingano (UFRGS)

Banca Examinadora:
Profa. Dra. Lucineia Fabris (UFSM)
Profa. Dra. Patrícia Lisandra Guidolin (IFRS)
Prof. Dr. Leonardo Prange Bonorino (UFRGS)

Tese apresentada e aprovada em 21 de dezembro de 2018.

Prof. Dr. Carlos Hoppen - Coordenador da PPGMAP

Dedico esta conclusão do Mestrado em Matemática Aplicada aos meus queridos pais, Pedro e Marta Hoo, minha esposa Manoela Soares, aos amigos e a todas as pessoas amáveis que encontrei em Caxias do Sul, Porto Alegre, Florianópolis, Índia e China.

Também dedico esta conclusão a todas as boas pessoas que fazem este mundo mais leve, mais alegre e criativo, mais amável e pacífico.

Agradecimentos

O autor agradece ao Centro Nacional de Processamento de Alto Desempenho de São Paulo e à Universidade Estadual de Campinas pela permissão de uso de seus recursos computacionais, em especial o cluster SGI Altix 1350/450. O autor também agradece ao governo francês pela disponibilização para pesquisa acadêmica do cluster Grid 5000, em especial o centro localizado em Lille, e ao governo grego pela permissão do uso do sistema Okeanos.

O autor também registra seu agradecimento à sua família pelo carinho e apoio recebidos, particularmente sua esposa, Maria Manoela de Melo Soares, e seus pais, Pedro e Marta Hoo. E ao professor Paulo R. Zingano pelo apoio, orientação, generosidade e bom humor. Também agradece ao Programa de Pós-Graduação em Matemática Aplicada e Computacional da UFRGS (PPGMAp), em especial seus professores e estudantes, por todo o apoio e suporte recebidos durante este período.

Resumo

Neste trabalho, mostra-se como utilizar programação paralela com OpenMP para a simulação numérica de soluções de equações diferenciais parciais em multiprocessadores, com aplicação à investigação numérica do chamado problema de campo distante para funções p-harmônicas limitadas em domínios $\mathbb{R}^n \setminus P$, sendo $p \geq n$ e $P \subset \mathbb{R}^n$ um conjunto finito de singularidades.

Abstract

In this work, we show how to use parallel programming with OpenMP for the numerical simulation of solutions to partial differential equations on multiprocessors. An application is given to the numerical investigation of far-field values of bounded p-harmonic functions on exterior domains $\Omega = \mathbb{R}^n \setminus P$, where $p \geq n$ and $P \subset \mathbb{R}^n$ is an arbitrary finite set.

Lista de Símbolos

a_+	PARTE POSITIVA de $a \in \mathbb{R}$, ou seja: $a_+ = \max\{a, 0\}$
a_-	PARTE NEGATIVA de $a \in \mathbb{R}$, ou seja: $a_- = \max\{-a, 0\}$
f_+	PARTE POSITIVA da função real f, ou seja: $f_+(x) = \max\{f(x), 0\}$
f_-	PARTE NEGATIVA da função real f, ou seja: $f_-(x) = \max\{-f(x), 0\}$
$\lvert a \rvert$	VALOR ABSOLUTO de $a \in \mathbb{R}$
$\lvert a \rvert$	NORMA EUCLIDEANA de $a \in \mathbb{R}^n$
$\lvert E \rvert$	MEDIDA DE LEBESGUE em $\mathbb{R}^n$ de $E \subseteq \mathbb{R}^n$ (E mensurável)
$a \cdot b$	PRODUTO de $a, b \in \mathbb{R}$
$a \cdot b$	PRODUTO ESCALAR de $a, b \in \mathbb{R}^n$
χ_E	FUNÇÃO CARACTERÍSTICA (ou INDICADORA) de $E \subseteq \mathbb{R}^n$
∇f	GRADIENTE da função f
$\operatorname{div} \boldsymbol{f}$	DIVERGENTE do campo f
$B_R(a)$	BOLA ABERTA de centro $a \in \mathbb{R}^n$ e raio R
υ_n	VOLUME n-dimensional da bola unitária $B_1(0)$, ou seja: $\upsilon_n = \lvert B_1(0) \rvert$
Δu	LAPLACIANO da função u
$\Delta_p u$	p-LAPLACIANO de u, ou seja: $\Delta_p u = \operatorname{div}\left(\lvert \nabla u \rvert^{p-2} \nabla u\right)$
$C^0(E)$	CONJUNTO DE FUNÇÕES CONTÍNUAS em um domínio E
$C^k(\Omega)$	CONJUNTO DE FUNÇÕES k-VEZES CONTINUAMENTE DIFERENCIÁVEIS no aberto Ω
$C_0^k(\Omega)$	CONJUNTO DE FUNÇÕES $u \in C^k(\Omega)$ COM SUPORTE COMPACTO contido em Ω
$L^q(\Omega)$	CONJUNTO DE FUNÇÕES f MENSURÁVEIS em um dado aberto Ω e tais que $\lvert f \rvert^q$ LEBESGUE-INTEGRÁVEL em Ω $(0 < q < \infty)$
$L^q_{\mathtt{loc}}(\Omega)$	CONJUNTO DE FUNÇÕES $f \in L^q(E)$ com $\lvert f \rvert^q$ LEBESGUE-INTEGRÁVEL em cada subdomínio COMPACTO $K \subset \Omega$ $(0 < q < \infty)$
$L^\infty(\Omega)$	CONJUNTO DE FUNÇÕES f MENSURÁVEIS no aberto Ω e ESSENCIALMENTE LIMITADAS em Ω, ou seja: $\operatorname*{ess\,sup}_{x \in \Omega} \lvert f(x) \rvert < \infty$
$L^\infty_{\mathtt{loc}}(\Omega)$	CONJUNTO DE FUNÇÕES f MENSURÁVEIS no aberto Ω tais que $f \in L^\infty(U)$ para cada aberto $U \subset \Omega$ com fecho compacto e contido em Ω

Lista de Siglas

(acronyms)

AIX	Advanced Interactive eXecutive operating system
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
CENAPAD	Centro Nacional de Processamento de Alto Desempenho
CFD	Computational Fluid Dynamics
CPU	Central Processing Unit
Fortran	Formula translation Language
GNU	Gnu's Not Unix
IDE	Integrated Development Environment
IP	Internet Protocol
Matlab	Matrix Laboratory
MIMD	Multiple Instruction, Multiple Data
MISD	Multiple Instruction, Single Data
MPI	Message Passing Interface
MPP	Massively Parallel Processor
NUMA	Non-Uniform Memory Access
OpenMP	Open Multi-Processing
OS	Operating System
PBS	Portable Batch System
PC	Personal Computer
POSIX	Portable Operating System Interface
PVM	Parallel Virtual Machine
RAM	Random Access Memory
RNP	Rede Nacional de Pesquisa
ROM	Read-Only Memory
SIMD	Single Instruction, Multiple Data
SISD	Single Instruction, Single Data
SINAPAD	Sistema Nacional de Processamento de Alto Desempenho
SMP	Single Memory Program
SPMD	Single Program, Multiple Data
TCP	Transmission Control Protocol

Índice

Capítulo I

Introdução

Neste trabalho, será investigado por vias numéricas o VALOR DE CAMPO DISTANTE (ou FAR-FIELD VALUE, na expressão correspondente em inglês) de funções p-harmônicas no exterior de um conjunto finito $P = \{ x_1, x_2, ..., x_m \} \subset \mathbb{R}^n$ dado, no caso de serem *limitadas* em seu domínio, e supondo-se ainda $p \geq n$. Em outras palavras, consideram-se aqui soluções $u \in C^0(\mathbb{R}^n) \cap C^1(\mathbb{R}^n \backslash P) \cap L^\infty(\mathbb{R}^n)$ da equação

$$\operatorname{div}\left(|\nabla u|^{p-2} \nabla u \right) \; = \; 0 \qquad \text{em } \mathbb{R}^n \backslash P \tag{1.1}$$

(sendo $p \geq n$) e investiga-se experimentalmente o valor numérico b_∞ do limite

$$\lim_{|x| \to \infty} u(x) \; =: \; b_\infty, \tag{1.2}$$

que, de acordo com a teoria conhecida das funções p-harmônicas, está bem definido (ver e.g. [2, 12, 13]), mas cujo valor (que naturalmente depende da solução u considerada) não se encontra ainda determinado na literatura corrente. Para uma descrição mais detalhada deste problema, ver Capítulo II a seguir.

Um dos métodos que podem ser usados para uma primeira sondagem sobre b_∞ é a realização de experimentos numéricos. Em particular, o problema serve como interessante motivação para ilustrar o emprego de multiprocessadores na obtenção (ou, mais exatamente, na *aproximação numérica*) de soluções de EDPs não lineares, que costumeiramente requerem o uso maciço destes recursos, seja em problemas estacionários (como o problema elíptico acima) ou evolutivos. Estes últimos aproveitam muito facilmente os recursos disponíveis em redes de processadores quando simulados por esquemas numéricos *explícitos*, dado o alto grau de paralelismo presente nestes métodos. Por essa razão, esquemas explícitos são os mais comumente empregados em MPPs (ou *massively parallel processors*, em inglês) [33], e constituem também uma escolha frequente em plataformas com número muito menor ou moderado de processadores disponíveis, como no caso dos centros de supercomputação brasileiros (acessíveis à comunidade acadêmica via o sistema SINAPAD e a chamada REDE NACIONAL DE PESQUISA, mantidos pelo MINISTÉRIO DE CIÊNCIA E TECNOLOGIA), sejam sistemas de memória compartilhada (*shared memory*), distribuída (*distributed memory systems*) ou mista, que é o caso mais comum atualmente. Para o bom uso destes

recursos, noções importantes de COMPUTAÇÃO CIENTÍFICA são necessárias, como o uso de bibliotecas especiais para processamento paralelo e diretivas de compiladores. Estes e outros conceitos são abordados e revistos no Capítulo IV. No caso presente deste trabalho, optou-se pelo uso de OpenMP, sendo a programação realizada em linguagem FORTRAN 90 e MATLAB (esta última, para o pós-processamento apenas). Entre as máquinas utilizadas, as principais foram os *clusters* SGI ALTIX 1350/450 (Universidade Estadual de Campinas, SP), GRID 5000 (França) e OKEANOS (Grécia), descritos resumidamente no Capítulo IV e Apêndice B.

Para determinar o tipo de paralelismo a ser explorado nos códigos desenvolvidos, optou-se pela solução de um problema *evolutivo* naturalmente associado a (1.1), dado pelo problema parabólico (mais detalhes no início do capítulo III)

$$u_t \;=\; \mathrm{div}\left(|\nabla u|^{p-2}\nabla u\right), \qquad x \in \mathbb{R}^n \backslash P,\;\; t > 0, \tag{1.3a}$$

$$u(x_j, t) \;=\; b_j, \quad 1 \leq j \leq m,\;\; t > 0, \tag{1.3b}$$

$$u(x, 0) \;=\; u_0(x) \in C^0(\mathbb{R}^n) \cap L^\infty(\mathbb{R}^n), \tag{1.3c}$$

onde $b_1, ...,b_m$ denotam os valores da função p-harmônica nos pontos do conjunto P, para várias escolhas do estado inicial u_0. Intuitivamente, espera-se que a solução $u(x,t)$ do problema (1.3) acima forneça, para $t \gg 1$, boa aproximação para a função p-harmônica considerada, o que foi constatado na prática. Como condição de contorno *numérica* adotada na fronteira da malha computacional (retangular), optou-se por condições de Neumann homogêneas, que são condições naturais neste contexto. Nos experimentos computacionais realizados (sempre em dimensão $n = 2$, por limitações de memória e tempo de processamento), a estimativa numérica obtida para b_∞ foi **aproximadamente**

$$b_\infty \;\approx\; \frac{1}{m}\sum_{j=1}^{m} b_j, \tag{1.4}$$

ou seja: a média aritmética simples dos valores assumidos pela função p-harmônica em questão no conjunto P de seus pontos singulares forneceu nos casos considerados boa aproximação para o valor de campo distante b_∞. No Cap. II, é mostrado que a igualdade em (1.4) é sempre válida se $m = 2$, mas **falha (em geral)** quando $m \geq 3$: ver **Teorema 2.2** e **Observação 2.1**. Os inúmeros detalhes envolvidos na obtenção do resultado (1.4) nos experimentos numéricos realizados formam precisamente o objeto dos capítulos a seguir.

Capítulo II

Problema Matemático Investigado

Neste capítulo, vamos descrever com mais detalhe o problema matemático investigado nos capítulos seguintes por métodos de computação científica. Trata-se basicamente do *comportamento do campo distante* (i.e., ao $|\,x\,| \to \infty$) de funções p-harmônicas em domínios $\Omega = \mathbb{R}^n \setminus P$, limitadas neste domínio Ω, onde $P \subset \mathbb{R}^n$ denota um conjunto finito qualquer dado. Em todo o capítulo, estaremos sempre supondo $p \geq n$, com $n \geq 2$, ainda que muito seja válido também para $p > 2$ qualquer. Por *funções p-harmônicas* em $\Omega \subseteq \mathbb{R}^n$ entendem-se funções $u \in C^1(\Omega)$ satisfazendo a equação

$$\Delta_p u \;\equiv\; \mathrm{div}\,(\,|\,\nabla u\,|^{p-2}\nabla u\,) \;=\; 0 \qquad \text{em} \;\; \Omega, \tag{2.1}$$

Caso $u \notin C^2(\Omega)$, podemos reescrever a equação acima numa forma equivalente para obtermos a solução "fraca" conforme (2.2) abaixo. Para isso utilizamos o teorema do divergente de Gauss.

$$\int_{\partial\Omega} \big(G \cdot \mathbf{n}\big)\, dS = \int_{\Omega} \big(\mathrm{div}\, G\big)\, dV \;\; e \;\; G(x) = F(x)\Phi(x),\; F(x) = |\,\nabla u(x)\,|^{p-2}\nabla u(x)$$

Onde $\Phi(x) \in C_0^1(\Omega)$ e $C_0^1(\Omega)$ denota o subespaço de funções $\Phi \in C^1(\Omega)$ de suporte compacto em Ω. Utilizando o teorema de Gauss com as funções acima, obtemos:

$$\int_{\partial\Omega} \big[\Phi(x)\,|\,\nabla u(x)\,|^{p-2}\,\nabla u(x)\big] \cdot \mathbf{n}\, dS = \int_{\Omega} \Phi(x)\Delta_p u(x) + |\,\nabla u(x)\,|^{p-2}\,\nabla u(x) \cdot \nabla\Phi(x)\; dV$$

A integral da esquerda vale 0 pois $\Phi(x) = 0$ em $\partial\Omega$ por causa do suporte compacto e $\Phi(x)\Delta_p u(x) = 0$ por (2.1). Resultando em:

$$\int_{\Omega} |\,\nabla u(x)\,|^{p-2}\,\nabla u(x) \cdot \nabla\Phi(x)\, dx \;=\; 0, \qquad \forall\;\; \Phi \in C_0^1(\Omega), \tag{2.2}$$

(Para (2.2) fazer sentido, poderíamos ter pedido apenas $u \in L^p_{\mathtt{loc}}(\Omega)$ satisfazendo $\nabla u \in L^p_{\mathtt{loc}}(\Omega)$, mas disso resultaria, como mostrado em [8, 10, 18] (ver também [30]), que $u \in C^1(\Omega)$. Em outras palavras, funções p-harmônicas são necessariamente *continuamente diferenciáveis* no interior de seus domínios, como foi colocado acima.)

Para $p > 2$, a equação (2.1) é *degenerada* nos pontos $x_0 \in \Omega$ onde $\nabla u(x_0) = \mathbf{0}$, e localmente uniformemente elíptica no aberto $\Omega_{\mathsf{u}} = \{\, x \in \Omega : \nabla u(x) \neq \mathbf{0} \,\}$, tendo-se, em particular, $u \in C^\infty(\Omega_{\mathsf{u}})$. Exemplos de funções p-harmônicas que *não* são de classe C^2 em seus domínios são dados em [17].

Consideremos agora a função v_{x_0}, a seguir, de simetria radial. Dado $x_0 \in \mathbb{R}^n$, definimos

$$v_{x_0}(x) := \begin{cases} |\, x - x_0 \,|^{\frac{p-n}{p-1}} & \text{se } \ p \neq n, \\ \\ -\log |\, x - x_0 \,| & \text{se } \ p = n, \end{cases} \tag{2.3}$$

para $x \in \Omega_0 = \mathbb{R}^n \setminus \{\, x_0 \,\}$. Temos que v_{x_0} é p-harmônica e também é chamado de ***função p-harmônica fundamental*** ou ***solução fundamental*** de (2.1) em seu domínio Ω_0. Da mesma forma, $u(x) := a + c\, v_{x_0}(x)$, é p-harmônica, ou seja, também é solução de (2.1), dados $a, c \in \mathbb{R}$ quaisquer.

2.1. Comportamento de campo distante

Consideremos agora o caso de interesse especial neste trabalho: $\Omega = \mathbb{R}^n \backslash P$, onde $P \subset \mathbb{R}^n$ denota um conjunto finito dado qualquer — digamos, $P = \{\, x_1, x_2, ..., x_m \,\}$, e funções p-harmônicas em Ω,

$$\Delta_p\, u \ = \ 0 \qquad \text{em } \ \mathbb{R}^n \setminus P, \tag{2.4}$$

e que sejam *limitadas* em seu domínio Ω (ou seja, $u \in L^\infty(\mathbb{R}^n)$). Neste caso, é conhecido (ver e.g. [13, 26]) que existem os limites

$$\lim_{x \to x_j} u(x) \ =: \ b_j, \qquad 1 \ \leq \ j \ \leq \ m, \tag{2.5}$$

de modo que, estendendo-se a função u a todo o domínio $\mathbb{R}^n$ pondo-se $u(x_j) := b_j$ para cada $1 \leq j \leq m$, resulta que $u \in C^0(\mathbb{R}^n)$. De maneira reversa, dados valores quaisquer $b_j \in \mathbb{R}$, $1 \leq j \leq m$, e um conjunto qualquer $P = \{\, x_1, x_2, ..., x_m \,\} \subset \mathbb{R}^n$, é natural considerar o problema de determinar $u \in C^0(\mathbb{R}^n) \cap L^\infty(\mathbb{R}^n)$ dada por

$$\Delta_p u \ = \ 0 \qquad \text{em } \ \mathbb{R}^n \setminus P, \tag{2.6a}$$

$$u(x_j) \ = \ b_j, \qquad 1 \leq j \leq m. \tag{2.6b}$$

A *existência* de soluções $u \in C^0(\mathbb{R}^n) \cap L^\infty(\mathbb{R}^n)$ para o problema (2.6) é estabelecida em [2], onde também é mostrada sua *unicidade* (ver Seção 2.2 para mais detalhes). Mais ainda: do TEOREMA 1.12 de [13](demonstrado no Apêndice A de [12]), resulta que o limite

$$\lim_{|x| \to \infty} u(x) \;=:\; b_\infty \tag{2.7}$$

está também bem definido, embora seu valor não seja determinado na análise em [12] e em [13]. Evidências numéricas para o valor de b_∞ nos experimentos realizados é justamente do que trata este trabalho, como descrito nos capítulos seguintes.

Além das propriedades das soluções u mencionadas acima, outras também são conhecidas na literatura. Por exemplo, do *princípio de comparação* obtido em [2], cuja prova é revisada na Seção 2.2, resulta, em particular, que

$$\min_{1 \le j \le m} b_j \;=:\; b \;\le\; u(x) \;\le\; B := \max_{1 \le j \le m} b_j \tag{2.8}$$

para todo $x \in \mathbb{R}^n$. Além disso, sabe-se que, para cada $1 \le j \le m$ e x próximo de x_j:

$$u(x) \,-\, b_j \;\sim\; c_j \, v_{x_j}(x) \tag{2.9}$$

para alguma constante $c_j \in \mathbb{R}$ apropriada, onde $v_{x_j} \in C^0(\mathbb{R}^n) \cap C^\infty(\mathbb{R}^n \backslash \{x_j\})$ denota a solução radial fundamental (centrada no ponto x_j) introduzida em (2.3) acima. (Estes resultados são verificados pelas aproximações numéricas obtidas para u neste trabalho.) Sendo a solução de (2.6*a*), (2.6*b*) *única*, resulta então de (2.7) que o VALOR DE CAMPO DISTANTE $b_\infty = \lim_{|x| \to \infty} u(x)$ seja univocamente determinado pelos valores b_1, ..., b_m dados e (possivelmente) os pontos singulares x_1, ..., x_m, ou seja,

$$b_\infty \;=\; F(b_1, b_2, ..., b_m; P) \tag{2.10}$$

para certa função F apropriada (não determinada na literatura corrente se $m > 2$). Nos experimentos numéricos realizados, obteve-se

$$F(b_1, b_2, ..., b_m; P) \;\approx\; \frac{1}{m} \sum_{j=1}^{m} b_j \tag{2.11}$$

e, em particular, b_∞ aparentemente não dependendo do conjunto P ou sua geometria,

mas sim dos valores b_j dados. A mesma questão pode ser posta mais amplamente com respeito ao *problema exterior* em geral: dados $\mathbb{K} \subset \mathbb{R}^n$ compacto arbitrário e $p \geq n \geq 2$, considerando-se $u \in C^0(\overline{\mathbb{R}^n \setminus \mathbb{K}}) \cap L^\infty(\mathbb{R}^n \setminus \mathbb{K})$ solução de

$$\Delta_p u \;=\; 0 \qquad \text{em } \; \mathbb{R}^n \setminus \mathbb{K}, \tag{2.12}$$

também neste caso (ver [2, 12, 13]) o valor limite b_∞ dado em (2.7) estará bem definido e dependendo unicamente do conjunto $\mathbb{K}$ e dos valores $\mathcal{U}$ da solução u na fronteira $\Gamma = \partial\mathbb{K}$, tendo-se novamente

$$\min_{\xi \in \Gamma} u(\xi) \;\leq\; u(x) \;\leq\; \max_{\xi \in \Gamma} u(\xi) \tag{2.13}$$

para todo $x \in \mathbb{R}^n \setminus \mathbb{K}$ (cf. Teorema 1 em [2]). Em particular, tem-se neste caso

$$\lim_{|x| \to \infty} u(x) \;=\; \mathrm{G}(\mathcal{U}; \mathbb{K}) \tag{2.14}$$

para certa função G que depende dos parâmetros indicados em (2.14), mas cuja determinação permanece em aberto na literatura presente.

2.2. Princípio de Comparação e valor de $\mathbf{b}_\infty$

Nesta seção, será demonstrada uma propriedade de comparação (Teorema 2.1) das soluções $u \in \mathcal{C} \equiv C^0(\mathbb{R}^n) \cap L^\infty(\mathbb{R}^n) \cap C^1(\mathbb{R}^n \setminus P)$ do problema (2.6), onde $p \geq n$, com base na discussão desenvolvida em [2], onde este e outros resultados são obtidos, válidos em um contexto bem mais geral que o considerado no presente trabalho. Como consequência direta deste *princípio de comparação*, resulta que as soluções do problema (2.6) são *únicas* na classe $\mathcal{C}$ acima (Corolário 2.1.1) e cotadas inferiormente e superiormente (Corolário 2.1.2).

Há também o Teorema 2.2 que demonstra que o valor de b_∞ é $(b_1 + b_2)/2$ para o caso particular de duas singularidades. E embora todos os dez resultados de b_∞ do Capítulo V estejam próximos da média, há a importante Observação 2.1, no final deste capítulo, que mostra que quando temos mais de duas singularidades ($m > 2$), em geral, b_∞ **não** é a média aritmética dos b_j das singularidades. Ou seja, o valor exato de b_∞ quando $m > 2$ permanece em aberto na literatura presente.

Lema 2.1. *Sendo $\alpha > 0$, tem-se*

$$\big|\,|\boldsymbol{b}|^{\alpha}\boldsymbol{b} - |\boldsymbol{a}|^{\alpha}\boldsymbol{a}\,\big| \;\le\; c_1(\alpha)\big(|\boldsymbol{a}| + |\boldsymbol{b}|\big)^{\alpha}|\boldsymbol{b}-\boldsymbol{a}| \tag{2.15a}$$

e

$$\big(|\boldsymbol{b}|^{\alpha}\boldsymbol{b} - |\boldsymbol{a}|^{\alpha}\boldsymbol{a}\big)\cdot(\boldsymbol{b}-\boldsymbol{a}) \;\ge\; c_2(\alpha)\big(|\boldsymbol{a}| + |\boldsymbol{b}|\big)^{\alpha}|\boldsymbol{b}-\boldsymbol{a}|^2 \tag{2.15b}$$

para quaisquer $\boldsymbol{a},\boldsymbol{b}\in\mathbb{R}^n$, onde $c_1(\alpha) = 1+\alpha$ e $c_2(\alpha) = 2^{-(\alpha+1)}$.

Prova: Começando-se com (2.15a), observa-se que esta desigualdade é imediata se $\boldsymbol{a}=\boldsymbol{0}$, ou $\boldsymbol{b}=\boldsymbol{0}$, ou ainda se $|\boldsymbol{a}| = |\boldsymbol{b}|$, de modo que é suficiente considerar o caso $|\boldsymbol{b}| > |\boldsymbol{a}| > 0$. Observando-se que

$$\big|\,|\boldsymbol{b}|^{\alpha}\boldsymbol{b} - |\boldsymbol{a}|^{\alpha}\boldsymbol{a}\,\big| \;\le\; \big||\boldsymbol{b}|^{\alpha} - |\boldsymbol{a}|^{\alpha}\big||\boldsymbol{a}| \;+\; |\boldsymbol{b}|^{\alpha}|\boldsymbol{b}-\boldsymbol{a}| \tag{2.16a}$$

e

$$\big|\,|\boldsymbol{b}|^{\alpha}\boldsymbol{b} - |\boldsymbol{a}|^{\alpha}\boldsymbol{a}\,\big| \;\le\; \big||\boldsymbol{b}|^{\alpha} - |\boldsymbol{a}|^{\alpha}\big||\boldsymbol{b}| \;+\; |\boldsymbol{a}|^{\alpha}|\boldsymbol{b}-\boldsymbol{a}|, \tag{2.16b}$$

pode-se prosseguir como segue. Se $0<\alpha<1$, tem-se, por (2.16a) e o teorema da média,

$$\begin{aligned}\big|\,|\boldsymbol{b}|^{\alpha}\boldsymbol{b} - |\boldsymbol{a}|^{\alpha}\boldsymbol{a}\,\big| \;&\le\; \big||\boldsymbol{b}|^{\alpha} - |\boldsymbol{a}|^{\alpha}\big||\boldsymbol{a}| \;+\; |\boldsymbol{b}|^{\alpha}|\boldsymbol{b}-\boldsymbol{a}| \\ &=\; \alpha\,\xi^{\alpha-1}\big||\boldsymbol{b}| - |\boldsymbol{a}|\big||\boldsymbol{a}| \;+\; |\boldsymbol{b}|^{\alpha}|\boldsymbol{b}-\boldsymbol{a}|\end{aligned}$$

para algum ξ entre $|\boldsymbol{a}|$ e $|\boldsymbol{b}|$, de modo que, como $\alpha-1<0$ e $\big||\boldsymbol{b}| - |\boldsymbol{a}|\big| \le |\boldsymbol{b}-\boldsymbol{a}|$,

$$\begin{aligned}\big|\,|\boldsymbol{b}|^{\alpha}\boldsymbol{b} - |\boldsymbol{a}|^{\alpha}\boldsymbol{a}\,\big| \;&\le\; \alpha\,|\boldsymbol{a}|^{\alpha-1}\big|\boldsymbol{b}-\boldsymbol{a}\big||\boldsymbol{a}| \;+\; |\boldsymbol{b}|^{\alpha}|\boldsymbol{b}-\boldsymbol{a}| \\ &=\; \alpha\,|\boldsymbol{a}|^{\alpha}\big|\boldsymbol{b}-\boldsymbol{a}\big| \;+\; |\boldsymbol{b}|^{\alpha}|\boldsymbol{b}-\boldsymbol{a}| \\ &=\; \big(\alpha\,|\boldsymbol{a}|^{\alpha} + |\boldsymbol{b}|^{\alpha}\big)|\boldsymbol{b}-\boldsymbol{a}| \\ &\le\; (1+\alpha)\big(|\boldsymbol{a}| + |\boldsymbol{b}|\big)^{\alpha}|\boldsymbol{b}-\boldsymbol{a}|,\end{aligned}$$

como afirmado. No caso $\alpha \ge 1$, procede-se de modo análogo, usando-se (2.16b) em vez de (2.16a), resultando então, neste caso, para certo $|\boldsymbol{a}| < \xi < |\boldsymbol{b}|$, que

$$\begin{aligned}\big|\,|\boldsymbol{b}|^{\alpha}\boldsymbol{b} - |\boldsymbol{a}|^{\alpha}\boldsymbol{a}\,\big| \;&\le\; \big||\boldsymbol{b}|^{\alpha} - |\boldsymbol{a}|^{\alpha}\big||\boldsymbol{b}| \;+\; |\boldsymbol{a}|^{\alpha}|\boldsymbol{b}-\boldsymbol{a}| \\ &=\; \alpha\,\xi^{\alpha-1}\big||\boldsymbol{b}| - |\boldsymbol{a}|\big||\boldsymbol{b}| \;+\; |\boldsymbol{a}|^{\alpha}|\boldsymbol{b}-\boldsymbol{a}| \\ &\le\; \alpha\,|\boldsymbol{b}|^{\alpha-1}\big|\boldsymbol{b}-\boldsymbol{a}\big||\boldsymbol{b}| \;+\; |\boldsymbol{a}|^{\alpha}|\boldsymbol{b}-\boldsymbol{a}| \\ &=\; \alpha\,|\boldsymbol{b}|^{\alpha}\big|\boldsymbol{b}-\boldsymbol{a}\big| \;+\; |\boldsymbol{a}|^{\alpha}|\boldsymbol{b}-\boldsymbol{a}| \\ &=\; \big(|\boldsymbol{a}|^{\alpha} + \alpha\,|\boldsymbol{b}|^{\alpha}\big)|\boldsymbol{b}-\boldsymbol{a}|\end{aligned}$$

$$\leq\ (1+\alpha)\,\big(\,|\,\boldsymbol{a}\,|+|\,\boldsymbol{b}\,|\,\big)^{\alpha}\,|\,\boldsymbol{b}-\boldsymbol{a}\,|,$$

o que conclui a prova de $(2.15a)$. Para mostrar $(2.15b)$, observando novamente que

$$|\,\boldsymbol{b}\,|^{\alpha}\boldsymbol{b}-|\,\boldsymbol{a}\,|^{\alpha}\boldsymbol{a}\ =\ |\,\boldsymbol{b}\,|^{\alpha}\big(\boldsymbol{b}-\boldsymbol{a}\big)\ +\ \big(\,|\,\boldsymbol{b}\,|^{\alpha}-|\,\boldsymbol{a}\,|^{\alpha}\big)\,\boldsymbol{a}$$

e

$$|\,\boldsymbol{b}\,|^{\alpha}\boldsymbol{b}-|\,\boldsymbol{a}\,|^{\alpha}\boldsymbol{a}\ =\ \big(\,|\,\boldsymbol{b}\,|^{\alpha}-|\,\boldsymbol{a}\,|^{\alpha}\big)\,\boldsymbol{b}\ +\ |\,\boldsymbol{a}\,|^{\alpha}\big(\boldsymbol{b}-\boldsymbol{a}\big),$$

obtém-se, somando-se as duas igualdades acima,

$$|\,\boldsymbol{b}\,|^{\alpha}\boldsymbol{b}-|\,\boldsymbol{a}\,|^{\alpha}\boldsymbol{a}\ =\ \frac{1}{2}\,\big(\,|\,\boldsymbol{a}\,|^{\alpha}+|\,\boldsymbol{b}\,|^{\alpha}\big)\,(\boldsymbol{b}-\boldsymbol{a})\ +\ \frac{1}{2}\,\big(\,|\,\boldsymbol{b}\,|^{\alpha}-|\,\boldsymbol{a}\,|^{\alpha}\big)\,(\boldsymbol{b}+\boldsymbol{a}).$$

Tomando-se o produto escalar da expressão acima com $\boldsymbol{b}-\boldsymbol{a}$, resulta

$$\begin{aligned}\big(\,|\,\boldsymbol{b}\,|^{\alpha}\boldsymbol{b}-|\,\boldsymbol{a}\,|^{\alpha}\boldsymbol{a}\big)\cdot(\boldsymbol{b}-\boldsymbol{a})\ &=\ \frac{1}{2}\big(\,|\,\boldsymbol{a}\,|^{\alpha}+|\,\boldsymbol{b}\,|^{\alpha}\big)\,|\,\boldsymbol{b}-\boldsymbol{a}\,|^{2}+\\ &\qquad+\frac{1}{2}\big(\,|\,\boldsymbol{b}\,|^{\alpha}-|\,\boldsymbol{a}\,|^{\alpha}\big)\,\big(\,|\,\boldsymbol{b}\,|^{2}-|\,\boldsymbol{a}\,|^{2}\big)\\ &\geq\ \frac{1}{2}\big(\,|\,\boldsymbol{a}\,|^{\alpha}+|\,\boldsymbol{b}\,|^{\alpha}\big)\,|\,\boldsymbol{b}-\boldsymbol{a}\,|^{2}\\ &\geq\ 2^{-(\alpha+1)}\,\big(\,|\,\boldsymbol{a}\,|+|\,\boldsymbol{b}\,|\big)^{\alpha}|\,\boldsymbol{b}-\boldsymbol{a}\,|^{2}\end{aligned}$$

pelo fato de se ter

$$\big(\,|\,\boldsymbol{a}\,|+|\,\boldsymbol{b}\,|\big)^{\alpha}\ \leq\ \Big[\,2\cdot\max\big\{\,|\,\boldsymbol{a}\,|,|\,\boldsymbol{b}\,|\,\big\}\Big]^{\alpha}=\ 2^{\alpha}\cdot\max\big\{\,|\,\boldsymbol{a}\,|^{\alpha},|\,\boldsymbol{b}\,|^{\alpha}\big\}\ \leq\ 2^{\alpha}\big(\,|\,\boldsymbol{a}\,|^{\alpha}+|\,\boldsymbol{b}\,|^{\alpha}\big).\quad\square$$

Lema 2.2*a* (L. P. Bonorino [2]). *Seja* $u\in C^{0}(\mathbb{R}^{n})\cap L^{\infty}(\mathbb{R}^{n})\cap C^{1}(\mathbb{R}^{n}\setminus P)$ *solução do problema* (2.6)*, onde* $P=\{\,x_{1},...,x_{m}\,\}$*. Se* $p=n$*, então* $\nabla u\in L^{p}(\mathbb{R}^{n})$ *e*

$$\|\,\nabla u\,\|_{L^{p}(\mathbb{R}^{n})}\ \leq\ C(p,m)\,\|\,u\,\|_{L^{\infty}(\mathbb{R}^{n})},\tag{2.17}$$

onde a constante $C(p,m)>0$ *depende apenas de* p,m *(e não da solução* u*).*

Prova: Seja $R_{0}=1+\max\{\,|\,x_{j}\,|\colon 1\leq j\leq m\,\}$, e tome-se $\epsilon\in(0,1)$ qualquer. Para tal ϵ, seja $\Omega_{\epsilon}:=\mathbb{R}^{n}\setminus\cup_{j=1}^{m}B_{\epsilon}(x_{j})$. Para $R>R_{0}$, seja $\psi_{R}\in C_{0}^{1}(\mathbb{R}^{n})$ função de corte satisfazendo as seguintes propriedades:

(i) $0\leq\psi_{R}(x)\leq 1$ para todo $x\in\mathbb{R}^{n}$;
(ii) $\psi_{R}(x)=1$ para todo $x\in\Omega_{\epsilon}\cap B_{R}(0)$;
(iii) $\psi_{R}(x)=0$ para todo $x\in B_{\epsilon/2}(x_{j})$, $1\leq j\leq m$;
(iv) $\psi_{R}(x)=0$ para todo $|\,x\,|\geq 2R$;

$(v)\ |\nabla\psi_{_R}(x)| \le 4/\epsilon$ para todo $x \in B_\epsilon(x_j)$, $1 \le j \le m$;
$(vi)\ |\nabla\psi_{_R}(x)| \le 2/R$ para todo $x \in S_{_R} = \{\, x \in \mathbb{R}^n\colon R < |x| < 2R\}$.

Tomando-se $\Phi(x) := u(x)\,\psi_{_R}(x)^p$ em (2.2), obtém-se

$$\int_{E(\epsilon,R)} |\nabla u|^{p-2}\,\nabla u(x)\cdot\big(\psi_{_R}(x)^p\,\nabla u \,+\, p\,u(x)\,\psi_{_R}(x)^{p-1}\,\nabla\psi_{_R}(x)\big)\,dx \;=\; 0,$$

onde $E(\epsilon,R) = B_{2R}(0)\backslash\bigcup_{j=1}^m B_{\epsilon/2}(x_j)$. Sendo $M := \|\,u\,\|_{L^\infty(\mathbb{R}^n)}$, tem-se, então:

$$\begin{aligned}
\int_{E(\epsilon,R)} \psi_{_R}(x)^p\,|\nabla u(x)|^p\,dx \;&=\; -\,p\int_{E(\epsilon,R)} u(x)\,\psi_{_R}(x)^{p-1}\,|\nabla u(x)|^{p-2}\,\nabla u(x)\cdot\nabla\psi_{_R}(x)\,dx\\
&\le\; p\int_{E(\epsilon,R)} |u(x)|\,\psi_{_R}(x)^{p-1}\,|\nabla u(x)|^{p-1}\,|\nabla\psi_{_R}(x)|\,dx\\
&\le\; p\,M\int_{E(\epsilon,R)} \psi_{_R}(x)^{p-1}\,|\nabla u(x)|^{p-1}\,|\nabla\psi_{_R}(x)|\,dx\\
&\le\; p\,M\left\{\int_{E(\epsilon,R)} \psi_{_R}(x)^p\,|\nabla u(x)|^p\,dx\right\}^{1-1/p}\left\{\int_{E(\epsilon,R)} |\nabla\psi_{_R}(x)|^p\,dx\right\}^{1/p}
\end{aligned}$$

de modo que

$$\int_{E(\epsilon,R)} \psi_{_R}(x)^p\,|\nabla u(x)|^p\,dx \;\le\; \big(p\,M\big)^p\int_{E(\epsilon,R)} |\nabla\psi_{_R}(x)|^p\,dx. \tag{2.18}$$

Visto que

$$\begin{aligned}
\int_{E(\epsilon,R)} |\nabla\psi_{_R}(x)|^p\,dx \;&=\; \sum_{j=1}^m \int_{\epsilon/2<|x-x_j|<\epsilon} |\nabla\psi_{_R}(x)|^p\,dx \;+\; \int_{R<|x|<2R} |\nabla\psi_{_R}(x)|^p\,dx\\
&\le\; \big(4/\epsilon\big)^p m\,\upsilon_n\,\epsilon^n \,+\, \big(2/R\big)^p\upsilon_n\,(2R)^n\\
&=\; 4^p\,\upsilon_n\,(m+1),
\end{aligned}$$

onde υ_n denota o volume da bola $B_1(0)$, resulta, por (2.18):

$$\int_{E(\epsilon,R)} \psi_{_R}(x)^p\,|\nabla u(x)|^p\,dx \;\le\; \big(p\,M\big)^p 4^p\,\upsilon_n\,(m+1).$$

Agora quando $R\to\infty$ e $\epsilon\to 0$, resulta

$$\int_{\mathbb{R}^n} |\nabla u(x)|^p\,dx \;\le\; \big(p\,M\big)^p 4^p\,\upsilon_n\,(m+1),$$

ou seja: $\|\,\nabla u\,\|_{L^p(\mathbb{R}^n)} \le 4\,p\,\big\{\upsilon_n(m+1)\big\}^{1/p}\,\|\,u\,\|_{L^\infty(\mathbb{R}^n)}$, concluindo a demonstração. $\square$

Lema 2.2*b* (L. P. Bonorino [2]). *Seja* $u \in C^0(\mathbb{R}^n) \cap L^\infty(\mathbb{R}^n) \cap C^1(\mathbb{R}^n \setminus P)$ *solução do problema* (2.6)*, onde* $P = \{x_1,...,x_m\}$*. Dado* $\epsilon > 0$*, seja* $\Omega_\epsilon := \mathbb{R}^n \setminus \cup_{j=1}^m B_\epsilon(x_j)$*. Se* $p > n$*, então* $\nabla u \in L^p(\Omega_\epsilon)$ *e*

$$\| \nabla u \|_{L^p(\Omega_\epsilon)} \leq C(p,m,n)\, \epsilon^{-1+n/p} \| u \|_{L^\infty(\mathbb{R}^n)}, \tag{2.19}$$

onde $C(p,m,n)$ *depende apenas de* p, m, n (*e não de* ϵ *ou da solução* u *considerada*)*.*

Prova: Repetindo-se o argumento anterior (que permanece válido até (2.18)) e a notação ali usada, obtém-se, para $p > n$:

$$\int_{E(\epsilon,R)} |\nabla \psi_R(x)|^p\, dx = \sum_{j=1}^m \int_{\epsilon/2 < |x-x_j| < \epsilon} |\nabla \psi_R(x)|^p\, dx + \int_{R < |x| < 2R} |\nabla \psi_R(x)|^p\, dx$$

$$\leq \big(4/\epsilon\big)^p m\, \upsilon_n\, \epsilon^n + \big(2/R\big)^p \upsilon_n\, (2R)^n$$

onde, como antes, $\upsilon_n = |B_1(0)|$. Por (2.18), resulta, então,

$$\int_{E(\epsilon,R)} \psi_R(x)^p\, |\nabla u(x)|^p\, dx \leq \big(pM\big)^p\, 2^p\, \upsilon_n \Big\{ \frac{2^p m}{\epsilon^{p-n}} + \frac{2^n}{R^{p-n}} \Big\}$$

(para todo $R > R_0$), onde $M = \| u \|_{L^\infty(\mathbb{R}^n)}$. Fazendo-se $R \to \infty$, isso fornece

$$\int_{\Omega_\epsilon} |\nabla u(x)|^p\, dx \leq \big(pM\big)^p\, 2^p\, \upsilon_n\, \frac{2^p m}{\epsilon^{p-n}},$$

ou seja: $\| \nabla u \|_{L^p(\Omega_\epsilon)} \leq 4p \big\{ \upsilon_n m \big\}^{1/p} \epsilon^{-1+n/p} \| u \|_{L^\infty(\mathbb{R}^n)}$, mostrando (2.19). □

Estamos finalmente em condições de obter o seguinte *princípio de comparação* para as soluções da equação (2.6*a*):

Teorema 2.1 - Princípio de Comparação (L. P. Bonorino [2]).
Sejam $u, v \in C^0(\mathbb{R}^n) \cap L^\infty(\mathbb{R}^n) \cap C^1(\mathbb{R}^n \setminus P)$ *soluções da equação* (2.6*a*) *em* $\mathbb{R}^n \setminus P$*, onde* $P = \{x_1,...,x_m\}$ *e* $p \geq n$*. Então, tem-se*

$$u(x_j) \leq v(x_j) \quad \forall\ 1 \leq j \leq m \quad \Longrightarrow \quad u(x) \leq v(x) \quad \forall\ x \in \mathbb{R}^n \tag{2.20}$$

ou seja: *se* $u \leq v$ *em* P*, então* $u \leq v$ *em todo o* $\mathbb{R}^n$*.*

Prova: Dado $\eta \in (0,1)$, tem-se $v - u + \eta > 0$ em P, de modo que se pode tomar $\epsilon = \epsilon(\eta) \in (0,1)$ suficientemente pequeno tal que

$$v(x) - u(x) + \eta > 0, \qquad \forall\ x \in \bigcup_{j=1}^{m} B_\epsilon(x_j). \tag{2.21}$$

Sejam $\Omega_\epsilon := \mathbb{R}^n \setminus \bigcup_{j=1}^m B_\epsilon(x_j)$, $R_0 = 1 + \max\{\,|\,x_j\,| : 1 \le j \le m\}$ e, para cada $R > R_0$, $\psi_R \in C_0^1(\mathbb{R}^n)$ construída como na prova do Lema 2.2. Aplicando-se (2.2) para a função teste $\Phi(x) := \psi_R(x)^p\,(v - u + \eta)_-$, tem-se

$$\int_{\Omega_\epsilon} \nabla\Phi \cdot \big(\,|\,\nabla u\,|^{p-2}\,\nabla u\big)\,dx = 0, \qquad \int_{\Omega_\epsilon} \nabla\Phi \cdot \big(\,|\,\nabla v\,|^{p-2}\,\nabla v\big)\,dx = 0,$$

de modo que

$$\int_{\Omega_\epsilon} \nabla\Phi \cdot \big(\,|\,\nabla v\,|^{p-2}\,\nabla v - |\,\nabla u\,|^{p-2}\,\nabla u\big)\,dx = 0.$$

Denotemos, por conveniência, $\boldsymbol{\mathcal{A}}(u,v) := |\,\nabla v\,|^{p-2}\,\nabla v - |\,\nabla u\,|^{p-2}\,\nabla u$. Como (ver [14], p.152, Lemma 7.6): $\nabla\Phi(x) = -\,\psi_R(x)^p\,\chi_{G_\eta}(x)\,\nabla(v-u) + p\,\psi_R(x)^{p-1}(v-u+\eta)_-\nabla\psi_R(x)$, onde

$$G_\eta := \{\,x \in \mathbb{R}^n\colon v(x) - u(x) + \eta < 0\,\} \subseteq \Omega_\epsilon, \tag{2.22}$$

resulta

$$\int_{\Omega_\epsilon} \psi_R(x)^p\,\chi_{G_\eta}(x)\,\nabla(v-u)\cdot\boldsymbol{\mathcal{A}}(u,v)\,dx = p\int_{\Omega_\epsilon} \psi_R(x)^{p-1}(v-u+\eta)_-\nabla\psi_R\cdot\boldsymbol{\mathcal{A}}(u,v)\,dx,$$

ou seja,

$$\int_{G_\eta} \psi_R(x)^p\,\nabla(v-u)\cdot\boldsymbol{\mathcal{A}}(u,v)\,dx = p\int_{R<|\,x\,|<2R} \psi_R(x)^{p-1}(v-u+\eta)_-\nabla\psi_R\cdot\boldsymbol{\mathcal{A}}(u,v)\,dx.$$

Pelo Lema 2.1, obtém-se, sendo $c(p) = 2^{-\,(p-1)}$:

$$\begin{aligned}
c(p)&\int_{G_\eta} \psi_R(x)^p\,\big(\,|\,\nabla u\,| + |\,\nabla v\,|\big)^{p-2}\,|\,\nabla(u-v)\,|^2\,dx \\
&\le\ p\int_{R<|\,x\,|<2R} \psi_R(x)^{p-1}(v-u+\eta)_-\,|\,\nabla\psi_R(x)\,|\cdot|\,\boldsymbol{\mathcal{A}}(u,v)\,|\,dx \\
&\le\ p\,(p-1)\int_{R<|\,x\,|<2R} \psi_R(x)^{p-1}(v-u+\eta)_-\,|\,\nabla\psi_R(x)\,|\,\big(\,|\,\nabla u\,| + |\,\nabla v\,|\big)^{p-2}\,|\,\nabla u - \nabla v\,|\,dx \\
&\le\ p\,(p-1)\int_{R<|\,x\,|<2R} \psi_R(x)^{p-1}(v-u+\eta)_-\,|\,\nabla\psi_R(x)\,|\,\big(\,|\,\nabla u\,| + |\,\nabla v\,|\big)^{p-1}\,dx.
\end{aligned}$$

Sendo $M := 1 + \|\,u\,\|_{L^\infty(\mathbb{R}^n)} + \|\,v\,\|_{L^\infty(\mathbb{R}^n)}$, e observando que $|\,\nabla(u-v)\,| \le |\,\nabla u\,| + |\,\nabla v\,|$, resulta, então,

$$
\begin{aligned}
c(p)\int_{G_\eta}\psi_{_R}(x)^p|\,\nabla(u-v)\,|^p\,dx
&\\
&\leq\ p\,(p-1)\int_{R<|\,x\,|<2R}\psi_{_R}(x)^{p-1}(v-u+\eta)_{-}\,|\,\nabla\psi_{_R}(x)\,|\,\big(\,|\,\nabla u\,|+|\,\nabla v\,|\,\big)^{p-1}\,dx\\
&\leq\ p\,(p-1)\,2^{p-1}\int_{R<|\,x\,|<2R}\psi_{_R}(x)^{p-1}(v-u+\eta)_{-}\,|\,\nabla\psi_{_R}(x)\,|\,\big(\,|\,\nabla u\,|^{p-1}+\,|\,\nabla v\,|^{p-1}\big)\,dx\\
&\leq\ p\,(p-1)\,2^{p-1}\,M\int_{R<|\,x\,|<2R}\psi_{_R}(x)^{p-1}\,|\,\nabla u\,|^{p-1}\,|\,\nabla\psi_{_R}(x)\,|\,dx\\
&\qquad\qquad +\,p\,(p-1)\,2^{p-1}\,M\int_{R<|\,x\,|<2R}\psi_{_R}(x)^{p-1}\,|\,\nabla v\,|^{p-1}\,|\,\nabla\psi_{_R}(x)\,|\,dx\\
&\leq\ p\,(p-1)\,2^{p-1}\,M\left\{\int_{R<|\,x\,|<2R}\psi_{_R}(x)^p\,|\,\nabla u\,|^p\,dx\right\}^{1-1/p}\left\{\int_{R<|\,x\,|<2R}|\,\nabla\psi_{_R}(x)\,|^p\,dx\right\}^{1/p}\\
&\qquad +\,p\,(p-1)\,2^{p-1}\,M\left\{\int_{R<|\,x\,|<2R}\psi_{_R}(x)^p\,|\,\nabla v\,|^p\,dx\right\}^{1-1/p}\left\{\int_{R<|\,x\,|<2R}|\,\nabla\psi_{_R}(x)\,|^p\,dx\right\}^{1/p}\\
&\leq\ p\,(p-1)\,2^{p-1}\,M\left\{\,\|\,\nabla u\,\|^{p-1}_{L^p(S_{_R})}+\,\|\,\nabla v\,\|^{p-1}_{L^p(S_{_R})}\right\}\left\{\,2^{p+n}\,\upsilon_n\,R^{-p+n}\right\}^{1/p}
\end{aligned}
$$

onde $S_{_R}=\{\,x\in\mathbb{R}^n\colon R<|\,x\,|<2R\}$ e, como antes, υ_n denota o volume da bola unitária. Portanto, tem-se, para todo $R>R_0$:

$$
\int_{G_\eta}\psi_{_R}(x)^p|\,\nabla(u-v)\,|^p\,dx\ \leq\ C(p,n)\,\frac{M}{R^{1-n/p}}\left\{\,\|\,\nabla u\,\|^{p-1}_{L^p(S_{_R})}+\,\|\,\nabla v\,\|^{p-1}_{L^p(S_{_R})}\right\}\qquad(2.23)
$$

onde $C(p,n)=2\,p\,(p-1)\,4^{p-1}\,2^{n/p}\,\upsilon_n^{1/p}$. Como, pelo Lema 2.2, tem-se $\nabla u\in L^p(\mathbb{R}^n\backslash B_{R_0}(0))$ e $\nabla v\in L^p(\mathbb{R}^n\backslash B_{R_0}(0))$, de modo que $\|\,\nabla u\,\|_{L^p(S_{_R})}\to 0$, $\|\,\nabla v\,\|_{L^p(S_{_R})}\to 0$ ao $R\to\infty$, resulta que, fazendo-se $R\to\infty$ em (2.23) acima,

$$
\int_{G_\eta}|\,\nabla(u-v)\,|^p\,dx\ =\ 0\qquad(2.24)
$$

(para cada $\eta\in(0,1)$ dado), já que $\psi_{_R}\to 1$ (ao $R\to\infty$) em G_η. Assim, lembrando que

$$
\nabla(v-u+\eta)_{-}=\ \chi_{_{G_\eta}}\nabla(u-v)\qquad\text{q. s. em }\mathbb{R}^n,
$$

tem-se que

$$
\int_{\mathbb{R}^n}|\,\nabla(v-u+\eta)_{-}|^p\,dx\ =\ 0,\qquad(2.24')
$$

ou seja, a função contínua $(v-u+\eta)_{-}$ é constante em $\mathbb{R}^n$. Como se sabe que

$$
(v-u+\eta)_{-}=\ 0\qquad\text{em}\ \bigcup_{j\,=\,1}^{m}B_\epsilon(x_j),
$$

segue, portanto, que $(v(x) - u(x) + \eta)_- = 0$ para todo $x \in \mathbb{R}^n$, ou seja:

$$u(x) \leq v(x) + \eta \qquad \forall\ x \in \mathbb{R}^n.$$

Como $\eta \in (0,1)$ é arbitrário, conclui-se que $u(x) \leq v(x)$ para todo $x \in \mathbb{R}^n$, como era para ser mostrado. □

Corolário 2.1.1 (UNICIDADE). *Sejam* $u, v \in C^0(\mathbb{R}^n) \cap L^\infty(\mathbb{R}^n) \cap C^1(\mathbb{R}^n \backslash P)$ *soluções de um mesmo problema* (2.6)*, onde* $p \geq n$*. Então, tem-se* $u = v$*, ou seja,*

$$u(x) = v(x), \qquad \forall\ x \in \mathbb{R}^n. \tag{2.25}$$

Prova: Por hipótese, tem-se $u = v$ em P. Em particular, como $u \leq v$ em P, segue do TEOREMA 2.1 que $u \leq v$ em $\mathbb{R}^n$. Por outro lado, tem-se também $v \leq u$ em P, de modo que, pelo TEOREMA 2.1, tem-se $v \leq u$ em $\mathbb{R}^n$. Logo, $u = v$ em $\mathbb{R}^n$, como afirmado. □

Corolário 2.1.2 (LIMITES DA SOLUÇÃO). *Seja* $u \in C^0(\mathbb{R}^n) \cap L^\infty(\mathbb{R}^n) \cap C^1(\mathbb{R}^n \backslash P)$ *solução do problema* (2.6)*, onde* $p \geq n$*. Então, tem-se*

$$b \leq u(x) \leq B \qquad \forall\ x \in \mathbb{R}^n, \tag{2.26}$$

onde $b := \min\{b_j,\ 1 \leq j \leq m\}$ *e* $B := \max\{b_j,\ 1 \leq j \leq m\}$.

Prova: Como $v = b$ é solução de (2.6) quando se toma $b_j = b$ para todo $j \in \{1, 2, ..., m\}$, segue do TEOREMA 2.1 que $v \leq u$ em $\mathbb{R}^n$, ou seja,

$$b \leq u(x) \qquad \forall\ x \in \mathbb{R}^n,$$

como colocado em (2.26). Analogamente, como $w = B$ é solução de (2.6) quando $b_j = B$ para todo j, segue do TEOREMA 2.1 que $u \leq w$ em $\mathbb{R}^n$, ou seja,

$$u(x) \leq B \qquad \forall\ x \in \mathbb{R}^n,$$

o que completa a prova do COROLÁRIO 2.1.2, como afirmado. □

Teorema 2.2 - Valor de b_∞ para duas singularidades (L. P. Bonorino, [1]). *Seja* $u \in C^0(\mathbb{R}^n) \cap L^\infty(\mathbb{R}^n) \cap C^1(\mathbb{R}^n \backslash P)$ *solução do problema* (2.6)*, onde* $p > n$*. Se* $m = 2$*,* $P = \{a_1, a_2\}$*,* $u(a_1) = b_1$ *e* $u(a_2) = b_2$ *então é sempre verdadeiro que*

$$b_\infty \;=\; \lim_{|x|\to\infty} u(x) \;=\; \frac{b_1 + b_2}{2} \tag{2.27}$$

Prova: Sejam $a := \dfrac{a_1 + a_2}{2} \in \mathbb{R}^n$ e $b := \dfrac{b_1 + b_2}{2} \in \mathbb{R}$. Seja $q := \dfrac{a_1 - a_2}{2} \in \mathbb{R}^n$, e seja $Q = \{q, -q\}$. Considere $\tilde{u} \in C^0(\mathbb{R}^n) \cap L^\infty(\mathbb{R}^n) \cap C^1(\mathbb{R}^n \backslash Q)$ definida por: $\tilde{u}(x) := u(x + a) - b$. Então $\tilde{u}$ é p-harmônica em $\mathbb{R}^n \backslash Q$, limitada em $\mathbb{R}^n$ e com $\tilde{u}(q) = \tilde{b}$, $\tilde{u}(-q) = -\tilde{b}$, onde $\tilde{b} := \dfrac{b_1 - b_2}{2}$. Como $v(x) := -\tilde{u}(-x)$ é p-harmônica no mesmo domínio de $\tilde{u}$ e assume os mesmos valores nos pontos q e $-q$, isto é, $\tilde{u}(q) = v(q) = \tilde{b}$ e $\tilde{u}(-q) = v(-q) = -\tilde{b}$ segue (por unicidade, corolário 2.1.1) que $v(x) = \tilde{u}(x)$ para todo $x \in \mathbb{R}^n$, ou seja: $\tilde{u}(x) = -\tilde{u}(-x)$ para todo x. Em outras palavras, $\tilde{u}(x) + \tilde{u}(-x) = 0$ para todo x. Usando novamente o Teorema 1.12 de [13], temos:

$$\lim_{|x|\to\infty} \tilde{u}(x) = \lim_{|x|\to\infty} -\tilde{u}(-x) = k = \textit{constante}, \;\text{ ou }\; \lim_{|x|\to\infty} \tilde{u}(-x) = -k$$

Os limites acima são válidos para qualquer direção, desde que x vá para longe. Então podemos escolher x que percorre a mesma reta que passa pela origem, mas em sentidos opostos, ou seja $x = v$ no primeiro limite e $x = -v$ no último limite acima. Desse modo os limites são iguais obrigando que $k = -k = 0$. Obtém-se, então,

$$\lim_{|x|\to\infty} \tilde{u}(x) \;=\; 0, \quad \text{de modo que:} \qquad \lim_{|x|\to\infty} u(x + a) \;-\; \frac{b_1 + b_2}{2} \;=\; 0. \qquad \square$$

Observação 2.1 (L. P. Bonorino, [1]). Seja, novamente, $p > n$. No caso de *três* ou mais singularidades, o valor de campo distante de funções p-harmônicas *não* será exatamente ou mesmo aproximadamente (em geral) a média aritmética dos valores b_j que assume nas singularidades, como mostra o seguinte argumento. Considerando a_1, $a_2 \in \mathbb{R}^n$ (com $a_1 \neq a_2$), e valores b_1, $b_2 \in \mathbb{R}$ (com $b_1 \neq b_2$), seja $u \in C^0(\mathbb{R}^n) \cap C^1(\mathbb{R}^n \backslash P) \cap L^\infty(\mathbb{R}^n)$ a função p-harmônica em $\mathbb{R}^n \backslash P$ que assume os valores b_1, b_2 nos pontos singulares a_1, a_2 (onde $P = \{a_1, a_2\}$). tem-se, então, pelo resultado acima, que

$$\lim_{|x|\to\infty} u(x) \;=\; \frac{b_1 + b_2}{2}.$$

Considere, agora, um terceiro ponto $a_3 \in \mathbb{R}^n$ (distinto de a_1 e a_2) e tomemos $b_3 := u(a_3)$. Seja $\tilde{P} = \{a_1, a_2, a_3\}$, e seja $\tilde{u} \in C^0(\mathbb{R}^n) \cap C^1(\mathbb{R}^n \backslash \tilde{P}) \cap L^\infty(\mathbb{R}^n)$ a função p-harmônica em $\mathbb{R}^n \backslash \tilde{P}$ satisfazendo as condições $\tilde{u}(a_j) = b_j$, $j = 1, 2, 3$. Por unicidade,

tem-se $\tilde{u} = u$, de modo que

$$\lim_{|x|\to\infty} \tilde{u}(x) \;=\; \frac{b_1+b_2}{2} \;\neq\; \frac{b_1+b_2+b_3}{3}$$

(note que, neste exemplo, a_3 é na verdade uma SINGULARIDADE REMOVÍVEL). Analogamente, tomando-se $b_3 \neq u(a_3)$ suficientemente próximo de $u(a_3)$, então $\tilde{u}$ terá três singularidades genuínas a_1, a_2, a_3, e (intuitivamnte) se terá $\tilde{u}$ próxima de u, de modo que

$$\lim_{|x|\to\infty} \tilde{u}(x) \;\approx\; \lim_{|x|\to\infty} u(x) \;=\; \frac{b_1+b_2}{2}$$

enquanto (em geral) $\dfrac{b_1+b_2}{2}$ e $\dfrac{b_1+b_2+b_3}{3}$ podem ser bem diferentes.

Capítulo III

Código Serial

Neste capítulo, vamos derivar e descrever o código básico, de execução *serial* (ou sequencial), que no capítulo seguinte será retrabalhado e levemente modificado de modo a ser executado mais eficientemente no tempo de modo *paralelo* (com uso de recursos de bibliotecas como OpenMP).

Como explicado nos capítulos anteriores, o problema matemático a ser resolvido é o problema descrito em (2.6a), (2.6b) e (2.7). Visto que vamos tentar resolver este limite b_∞ de (2.7) por vias numéricas, utilizando um computador, precisamos de alguma iteração, tipo: $u_{k+1} = u_k + [?]$ onde $[?]$ é alguma transformação, algum processo adequado. Para descubrir este $[?]$ processo adequado vamos supor que $\{u_k\}_{k\in\mathbb{N}}$ é uma sequência de funções tal que:

a) u_k converge para u_∞,
b) $\Delta_p u_\infty \equiv \operatorname{div}(\,|\nabla u_\infty|^{p-2}\nabla u_\infty\,) = 0$, e
c) $u_k = u(x, k\Delta t)$, ou seja, u_k é a função u acrescida da variável temporal e Δt é um valor pequeno, arbitrário, mas fixado.

Então, quando $k \to \infty$, temos:

$$\frac{\partial u_k}{\partial t} \sim \frac{u_{k+1} - u_k}{\Delta t} = \frac{u(x, k\Delta t + \Delta t) - u(x, k\Delta t)}{\Delta t} \to 0 \tag{3.a}$$

$$\Delta_p u_k = \operatorname{div}(\,|\nabla u_k|^{p-2}\nabla u_k\,) \to 0 \tag{3.b}$$

Logo podemos supor:

$$\frac{\partial u_k}{\partial t} \sim \operatorname{div}(\,|\nabla u_k|^{p-2}\nabla u_k\,) \tag{3.c}$$

Na realidade, vamos reescrever (3.c) como uma igualdade. Ou seja vamos trocar o problema, a equação, estática, inicial:

$$\Delta_p u \;=\; 0 \qquad \text{em } \mathbb{R}^n \setminus P, \tag{2.6a}$$

$$u(x_j) \;=\; b_j, \qquad 1 \leq j \leq m. \tag{2.6b}$$

$$\lim_{|\,x\,|\,\to\,\infty} u(x) \;=:\; b_\infty \tag{2.7}$$

Pela equação ***evolutiva*** do p-Laplaciano:

$$u_t \;=\; \Delta_p u, \qquad x \in \mathbb{R}^n \setminus P, \tag{3.1}$$

$$u(x,0) \;=\; u_0(x), \qquad x \in \mathbb{R}^n, \tag{3.2}$$

$$u(a_j,t) \;=\; b_j, \qquad 1 \leq j \leq m,\;\; t \geq 0, \tag{3.3}$$

para um estado inicial apropriado (descrito na seção 3.2), sendo $P = \{\, a_1,\, a_2, ...,\, a_m\}$ o conjunto dado dos pontos singulares da solução $u(x,t)$, para todo $t \geq 0$, com os valores respectivos $b_1, b_2, ..., b_m$ (especificados no problema), e onde Δ_p denota o operador p-Laplaciano (na variável espacial $x = (x_1, x_2, ..., x_n)$), ou seja,

$$\Delta_p u \;=\; \operatorname{div}\big(\,|\,\nabla u\,|^{p-2}\,\nabla u\,\big). \tag{3.4}$$

Desta forma, como já mencionado no final do capítulo I espera-se que a solução $u(x,t)$ do problema (3.1), (3.2), (3.3) convirja para a solução de $(2.6a)$, $(2.6b)$ para t suficientemente grande e assim tenhamos uma boa ideia do valor b_∞, limite de (2.7), que é o assunto central desta dissertação.

Neste capítulo e nos seguintes, consideraremos sempre $p \geq n \geq 2$ (onde, por conveniência em função da alta complexidade computacional do problema em dimensão $n \geq 3$, nas simulações numéricas realizadas tomaremos sempre $n = 2$, ou seja, consideraremos para os fins práticos o problema (3.1), (3.2), (3.3) acima no *plano* somente). Em dimensão 2, vamos por simplicidade mudar levemente a notação acima e denotar um ponto genérico em $\mathbb{R}^2 \times [\,0, \infty)$ por (x, y, t), reescrevendo a equação (3.1) acima na forma

$$u_t \;=\; \frac{\partial}{\partial x}\big(\,|\,\nabla u\,|^{p-2}\,u_x\,\big) \;+\; \frac{\partial}{\partial y}\big(\,|\,\nabla u\,|^{p-2}\,u_y\,\big), \tag{3.5}$$

onde, como anteriormente, denotamos $u_t = \partial u/\partial t$, $u_x = \partial u/\partial x$, e $u_y = \partial u/\partial y$.

3.1. Aproximação por diferenças finitas

Para a discretização da equação (3.5), usaremos o *método de diferenças finitas* [21, 29, 31], aproximando as derivadas espaciais $\partial w/\partial x$, $\partial w/\partial y$ de uma dada função $w = w(x, y, t)$ num ponto $(\hat{x}, \hat{y}, \hat{t})$ por diferenças centrais,

$$\frac{\partial w}{\partial x}(\hat{x}, \hat{y}, \hat{t}) \;=\; \frac{w(\hat{x} + h/2, \hat{y}, \hat{t}) - w(\hat{x} - h/2, \hat{y}, \hat{t})}{h} \;+\; O(h^2), \tag{3.6}$$

$$\frac{\partial w}{\partial y}(\hat{x}, \hat{y}, \hat{t}) \;=\; \frac{w(\hat{x}, \hat{y} + h/2, \hat{t}) - w(\hat{x}, \hat{y} - h/2, \hat{t})}{h} \;+\; O(h^2), \tag{3.7}$$

com erro de segunda ordem, e aproximando a derivada temporal $\partial w/\partial t$ neste ponto por diferenças de primeira ordem,

$$\frac{\partial w}{\partial t}(\hat{x}, \hat{y}, \hat{t}) \;=\; \frac{w(\hat{x}, \hat{y}, \hat{t} + \Delta t) - w(\hat{x}, \hat{y}, \hat{t})}{\Delta t} \;+\; O(\Delta t), \tag{3.8}$$

para os espaçamentos $h > 0$ (nas variáveis x, y) e $\Delta t > 0$ (na variável t) considerados (ver e.g. [21, 29, 31], Ch. I). Para descrever a aplicação de (3.6), (3.7), (3.8) para a equação (3.5) acima, será conveniente considerar momentaneamente sua aplicação a uma equação de forma genérica

$$u_t \;=\; \frac{\partial}{\partial x}\,\mathcal{F}(x, y, t) \;+\; \frac{\partial}{\partial y}\,\mathcal{G}(x, y, t), \tag{3.9}$$

onde $\mathcal{F}, \mathcal{G}$ são expressões dadas. Dados $h > 0$, $\Delta t > 0$ (*grid spacings*), e considerando-se os pontos de malha $(x_i, y_j; t_k)$, onde

$$x_i \;=\; i \cdot h, \qquad y_j \;=\; j \cdot h, \qquad t_k \;=\; k \cdot \Delta t, \tag{3.10}$$

para $i, j, k \in \mathbb{Z}$ (e $k \geq 0$), obtém-se, por (3.6), (3.7) e (3.8) acima, as seguintes equações a diferenças determinando as aproximações correspondentes $v_{i,j;k}$ para o valor exato $u_{i,j;k} \equiv u(x_i, y_j, t_k)$ no ponto (x_i, y_j, t_k):

$$\frac{v_{i,j;k} - v_{i,j;k-1}}{\Delta t} \;=\; \frac{\mathcal{F}_{i+1/2,\,j;k-1} - \mathcal{F}_{i-1/2,\,j;k-1}}{h} \;+\; \frac{\mathcal{G}_{i,j+1/2;k-1} - \mathcal{G}_{i,j-1/2;k-1}}{h} \tag{3.11}$$

ou

$$\frac{v_{i,j;k} - v_{i,j;k-1}}{\Delta t} \;=\; \frac{\mathcal{F}_{i+1/2,\,j;k} - \mathcal{F}_{i-1/2,\,j;k}}{h} \;+\; \frac{\mathcal{G}_{i,j+1/2;k} - \mathcal{G}_{i,j-1/2;k}}{h} \tag{3.12}$$

ou ainda

$$\begin{aligned}\frac{v_{i,j;k} - v_{i,j;k-1}}{\Delta t} \;=\; \theta \cdot \left\{ \frac{\mathcal{F}_{i+1/2,\,j;k-1} - \mathcal{F}_{i-1/2,\,j;k-1}}{h} \;+\; \frac{\mathcal{G}_{i,j+1/2;k-1} - \mathcal{G}_{i,j-1/2;k-1}}{h} \right\} \\ +\; (1-\theta) \cdot \left\{ \frac{\mathcal{F}_{i+1/2,\,j;k} - \mathcal{F}_{i-1/2,\,j;k}}{h} \;+\; \frac{\mathcal{G}_{i,j+1/2;k} - \mathcal{G}_{i,j-1/2;k}}{h} \right\}\end{aligned} \tag{3.13}$$

para dado $\theta \in [\,0, 1\,]$ fixado (constante), ou ainda outras escolhas [21, 29, 31], onde

$$\mathcal{F}_{i-1/2,\,j;k} \equiv \mathcal{F}(x_i - h/2,\, y_j,\, t_k), \qquad \mathcal{G}_{i,\,j+1/2;k} \equiv \mathcal{F}(x_i,\, y_j + h/2,\, t_k), \tag{3.14}$$

e assim por diante. (Note que (3.11) e (3.12) são casos particulares da expressão (3.13), correspondentes às escolhas $\theta = 1$ e $\theta = 0$, respectivamente.) Em cada caso (3.11), (3.12) ou (3.13) acima, os valores da função de malha v no nível de tempo $t_{k-1} = t_k - \Delta\, t$ são supostos conhecidos, previamente calculados, utilizando-se assim as expressões (3.11), (3.12) ou (3.13) para a determinação dos valores (ainda não conhecidos) de v no nível de tempo seguinte, ou seja, no nível $t_k = t_{k-1} + \Delta t$.

Considerando-se agora, em vez da equação (3.9) acima, a equação mais geral

$$u_t \;=\; \frac{\partial}{\partial x}\mathcal{F}(x, y, t; u, u_x, u_y) \;+\; \frac{\partial}{\partial y}\mathcal{G}(x, y, t; u, u_x, u_y) \tag{3.15}$$

(que é mais próxima à equação que nos interessa neste trabalho, ver (3.5)), para dadas funções $\mathcal{F}$, $\mathcal{G}$, a equação discreta correspondente a (3.11) seria, de modo análogo,

$$\begin{aligned}\frac{v_{i,j;k} - v_{i,j;k-1}}{\Delta t} \;=\; & \frac{1}{h}\Big[\, \mathcal{F}(x_{i+1/2}, y_j, t_{k-1};\, v_{i+1/2,j;k-1},\, w_{i+1/2,j;k-1},\, z_{i+1/2,j;k-1}) - \\ & \qquad \mathcal{F}(x_{i-1/2}, y_j, t_{k-1};\, v_{i-1/2,j;k-1},\, w_{i-1/2,j;k-1},\, z_{i-1/2,j;k-1}) \,\Big] \\ + \; & \frac{1}{h}\Big[\, \mathcal{G}(x_i, y_{j+1/2}, t_{k-1};\, v_{i,j+1/2;k-1},\, w_{i,j+1/2;k-1},\, z_{i,j+1/2;k-1}) - \\ & \qquad \mathcal{G}(x_i, y_{j-1/2}, t_{k-1};\, v_{i,j-1/2;k-1},\, w_{i,j-1/2;k-1},\, z_{i,j-1/2;k-1}) \,\Big]\end{aligned} \tag{3.16}$$

onde v, w, z representam as funções (ou *projeções*) de malha discretas correspondentes às funções u, u_x e u_y, respectivamente, podendo-se tomar:

$$v_{i-1/2,j;k-1} \;=\; \frac{v_{i-1,j;k-1} + v_{i,j;k-1}}{2} \;\approx\; u(x_i - h/2,\, y_j,\, t_{k-1}), \tag{3.17}$$

$$v_{i,j-1/2;k-1} \;=\; \frac{v_{i,j-1;k-1} + v_{i,j;k-1}}{2} \;\approx\; u(x_i,\, y_j - h/2,\, t_{k-1}) \tag{3.18}$$

(e similarmente para $v_{i+1/2,j;k-1}$ e $v_{i,j+1/2;k-1}$), e, para os termos $w \approx u_x$, $z \approx u_y$:

$$w_{i-1/2,j;k-1} \;=\; \frac{v_{i,j;k-1} - v_{i-1,j;k-1}}{h} \;\approx\; u_x(x_i - h/2,\, y_j,\, t_{k-1}), \tag{3.19}$$

$$\begin{aligned} w_{i,j-1/2;k-1} \;=\; & \frac{1}{2}\left[\, \frac{v_{i+1,j-1;k-1} - v_{i-1,j-1;k-1}}{2\,h} \;+\; \frac{v_{i+1,j;k-1} - v_{i-1,j;k-1}}{2\,h} \,\right] \\ \;\approx\; & u_x(x_i,\, y_j - h/2,\, t_{k-1}), \end{aligned} \tag{3.20}$$

$$\begin{aligned} z_{i-1/2,j;k-1} \;=\; & \frac{1}{2}\left[\, \frac{v_{i-1,j+1;k-1} - v_{i-1,j-1;k-1}}{2\,h} \;+\; \frac{v_{i,j+1;k-1} - v_{i,j-1;k-1}}{2\,h} \,\right] \\ \;\approx\; & u_y(x_i - h/2,\, y_j,\, t_{k-1}) \end{aligned} \tag{3.21}$$

$$z_{i,j-1/2;k-1} \;=\; \frac{v_{i,j;k-1} - v_{i,j-1;k-1}}{h} \;\approx\; u_y(x_i,\, y_j - h/2,\, t_{k-1}), \tag{3.22}$$

e analogamente para os termos $w_{i+1/2,j;k-1}$, $w_{i,j+1/2;k-1}$, $z_{i+1/2,j;k-1}$ e $z_{i,j+1/2;k-1}$. Usando (3.17) – (3.22) em (3.16) acima, resulta uma expressão determinando explicitamente os novos valores $v_{i,j;k}$ (referentes à função de malha v no nível de tempo t_k) em termos de valores conhecidos $v_{i',j';k-1}$ da função v (já calculados na etapa anterior), ou seja, tem-se deste modo construído um esquema ou método *explícito* para

a determinação dos valores de malha $v_{i,j;k}$. Em comparação, estendendo-se procedimentos como (3.12) ou (3.13) para o caso da equação (3.15) resulta (no caso $\theta < 1$) numa expressão da forma

$$v_{i,j;k} \;=\; \mathbb{F}(x_{i-1}, x_i, x_{i+1}, y_{j-1}, y_j, y_{j+1}, t_{k-1}, t_k; \mathcal{V}_{k-1}, \mathcal{V}_k)$$

onde $\mathcal{V}_{k-1} = \big\{ v_{i+i',j+j';k-1} \colon i', j' \in \{-1, 0, 1\} \big\}$ e $\mathcal{V}_k = \big\{ v_{i+i',j+j';k} \colon i', j' \in \{-1, 0, 1\} \big\}$, correspondente a um esquema *implícito* para a determinação dos valores $v_{i,j;k}$.

Por exemplo, estendendo-se (3.12) para o caso da equação (3.15), resultaria a expressão

$$\begin{aligned} \frac{v_{i,j;k} - v_{i,j;k-1}}{\Delta t} \;=\; & \frac{1}{h} \Big[\mathcal{F}(x_{i+1/2}, y_j, t_k; v_{i+1/2,j;k}, w_{i+1/2,j;k}, z_{i+1/2,j;k}) - \\ & \qquad \mathcal{F}(x_{i-1/2}, y_j, t_k; v_{i-1/2,j;k}, w_{i-1/2,j;k}, z_{i-1/2,j;k}) \Big] \\ & + \frac{1}{h} \Big[\mathcal{G}(x_i, y_{j+1/2}, t_k; v_{i,j+1/2;k}, w_{i,j+1/2;k}, z_{i,j+1/2;k}) - \\ & \qquad \mathcal{G}(x_i, y_{j-1/2}, t_k; v_{i,j-1/2;k}, w_{i,j-1/2;k}, z_{i,j-1/2;k}) \Big] \end{aligned} \tag{3.23}$$

onde

$$v_{i-1/2,j;k} \;=\; \frac{v_{i-1,j;k} + v_{i,j;k}}{2}, \qquad v_{i,j-1/2;k} \;=\; \frac{v_{i,j-1;k} + v_{i,j;k}}{2}, \tag{3.24}$$

$$w_{i-1/2,j;k} \;=\; \frac{v_{i,j;k} - v_{i-1,j;k}}{h}, \tag{3.25}$$

$$w_{i,j-1/2;k} \;=\; \frac{1}{2} \left[\frac{v_{i+1,j-1;k} - v_{i-1,j-1;k}}{2h} + \frac{v_{i+1,j;k} - v_{i-1,j;k}}{2h} \right], \tag{3.26}$$

$$z_{i-1/2,j;k} \;=\; \frac{1}{2} \left[\frac{v_{i-1,j+1;k} - v_{i-1,j-1;k}}{2h} + \frac{v_{i,j+1;k} - v_{i,j-1;k}}{2h} \right], \tag{3.27}$$

$$z_{i,j-1/2;k} \;=\; \frac{v_{i,j;k} - v_{i,j-1;k}}{h}, \tag{3.28}$$

e analogamente para $v_{i+1/2,j;k}$, $v_{i,j+1/2;k}$, $w_{i+1/2,j;k}$, $w_{i,j+1/2;k}$, $z_{i+1/2,j;k}$ e $z_{i,j+1/2;k}$. As expressões (3.23), (3.24) – (3.28) constituem um método implícito para a obtenção dos valores $v_{i,j;k}$, determinados agora por um *sistema* de equações acopladas (de grande porte) que, no caso de $\mathcal{F}$ ou $\mathcal{G}$ dependerem não linearmente com respeito a

pelo menos uma das variáveis u, u_x, u_y (ver (3.15) acima), requer assim a solução de um complexo sistema de equações *não lineares*, cuja resolução é complicada e difícil de paralelizar eficientemente. Lembrando que, no caso de interesse dado pela equação (3.5) tem-se $\mathcal{F}$, $\mathcal{G}$ dadas por

$$\mathcal{F} \;=\; \left(u_x^2 + u_y^2\right)^{\frac{p-2}{2}} \cdot\; u_x, \qquad \mathcal{G} \;=\; \left(u_x^2 + u_y^2\right)^{\frac{p-2}{2}} \cdot\; u_y, \tag{3.29}$$

optou-se, então, no presente trabalho, pelo emprego do procedimento *explícito* descrito em (3.16) e (3.17) − (3.22) acima, mesmo com possíveis prejuízos (que se mostraram pequenos) nas propriedades de estabilidade do método resultante.

Particularizando o método explícito descrito acima para o caso da equação investigada, dada pela equação (3.5) — que, para maior correspondência com os códigos FORTRAN desenvolvidos, vamos a partir de agora reescrever segundo (3.15) na forma

$$u_t \;=\; \frac{\partial}{\partial x}\left(F(u_x, u_y)\, u_x\right) \;+\; \frac{\partial}{\partial y}\left(G(u_x, u_y)\, u_x\right), \tag{3.30}$$

onde

$$F(w, z) \;=\; \left(w^2 + z^2\right)^{\frac{p-2}{2}}, \qquad G(w, z) \;=\; \left(w^2 + z^2\right)^{\frac{p-2}{2}}, \tag{3.31}$$

tem-se então o seguinte procedimento para o cálculo dos valores das aproximações $v_{i,j;\,k}$ nos nodos ou nós $(i, j;\, k)$ da malha (ver (3.10)). Começando pelos *nodos internos* (que foram o objeto de *toda* a discussão anterior, desenvolvida em (3.11) − (3.28) acima), tem-se, pela discussão acima,

$$\begin{aligned}
\frac{v_{i,j;\,k} - v_{i,j;\,k-1}}{\Delta t} \;=\; & \frac{1}{h}\left[\, F(w_{i+1/2,\,j;\,k-1},\, z_{i+1/2,\,j;\,k-1})\, \frac{v_{i+1,\,j;\,k-1} - v_{i,\,j;\,k-1}}{h} \;-\right. \\
& \qquad \left. F(w_{i-1/2,\,j;\,k-1},\, z_{i-1/2,\,j;\,k-1})\, \frac{v_{i,\,j;\,k-1} - v_{i-1,\,j;\,k-1}}{h} \,\right] \\
& + \frac{1}{h}\left[\, G(w_{i,\,j+1/2;\,k-1},\, z_{i,\,j+1/2;\,k-1})\, \frac{v_{i,\,j+1;\,k-1} - v_{i,\,j;\,k-1}}{h} \;-\right. \\
& \qquad \left. G(w_{i,\,j-1/2;\,k-1},\, z_{i,\,j-1/2;\,k-1})\, \frac{v_{i,\,j;\,k-1} - v_{i,\,j-1;\,k-1}}{h} \,\right],
\end{aligned}$$

onde $w_{i-1/2,\,j;\,k-1}$, $w_{i+1/2,\,j;\,k-1}$, $w_{i,\,j-1/2;\,k-1}$, $w_{i,\,j+1/2;\,k-1}$ e $z_{i-1/2,\,j;\,k-1}$, $z_{i+1/2,\,j;\,k-1}$, $z_{i,\,j-1/2;\,k-1}$, $z_{i,\,j+1/2;\,k-1}$ são dados em (3.19) − (3.22) acima. Ou seja, as aproximações $v_{i,j;\,k}$ são computadas pela fórmula explícita

$$
\begin{aligned}
v_{i,j;\,k} \;=\; v_{i,j;\,k-1} \;+\; \lambda \cdot \Big[\; & F(w_{i+1/2,j;\,k-1},\, z_{i+1/2,j;\,k-1}) \left(v_{i+1,j;\,k-1} - v_{i,j;\,k-1}\right) - \\
& F(w_{i-1/2,j;\,k-1},\, z_{i-1/2,j;\,k-1}) \left(v_{i,j;\,k-1} - v_{i-1,j;\,k-1}\right) \Big] \\
+\; \lambda \cdot \Big[\; & G(w_{i,j+1/2;\,k-1},\, z_{i,j+1/2;\,k-1}) \left(v_{i,j+1;\,k-1} - v_{i,j;\,k-1}\right) - \\
& G(w_{i,j-1/2;\,k-1},\, z_{i,j-1/2;\,k-1}) \left(v_{i,j;\,k-1} - v_{i,j-1;\,k-1}\right) \Big]
\end{aligned}
\tag{3.32}
$$

para os nodos (x_i, y_j) internos, onde λ denota o *número de Courant* (CFL *number*),

$$
\lambda \;=\; \frac{\Delta t}{h^2}, \tag{3.33}
$$

escolhido suficientemente pequeno de modo a tornar o método (3.32) *numericamente estável*, e mantido constante ao se refinar a malha. Nas simulações, valores $\lambda \approx 0{,}05$ foram usados, verificando-se experimentalmente instabilidades para $\lambda > 0{,}10$.

Originalmente, o problema a ser resolvido (no contexto da equação (3.30)) está posto em *todo* o espaço $\mathbb{R}^2$, requerendo malhas de tamanho infinito (ver (3.1)-(3.3)). Na prática, evidentemente, tomam-se malhas apropriadamente grandes, mas *finitas*, com os índices i, j variando num conjunto de valores $M_1 \le i \le M_2$, $N_1 \le j \le N_2$, criando-se necessidade de procedimentos especiais nos chamados *nodos de fronteira* (correspondentes aos casos $i \in \{M_1, M_2\}$ ou $j \in \{N_1, N_2\}$), dados por condições de fronteira *numéricas* [21, 29, 31]. Estas condições precisam ser definidas adequadamente para não provocarem instabilidades ou efeitos fisicamente espúrios na solução obtida, ou mesmo tornarem a simulação menos eficiente. No caso presente, estas dificuldades foram satisfatoriamente resolvidas utilizando-se o seguinte procedimento bem conhecido: acrescentou-se uma camada extra de pontos (i, j), correspondentes aos valores $i \in \{M_1 - 1, M_2 + 1\}$ ou $j \in \{N_1 - 1, N_2 + 1\}$, impondo-se as condições de fronteira

$$
v_{M_1-1,\,j;\,k-1} \;=\; v_{M_1,\,j;\,k-1}, \qquad v_{M_2+1,\,j;\,k-1} \;=\; v_{M_2,\,j;\,k-1} \tag{3.34}
$$

para todo $N_1 \le j \le N_2$, e

$$
v_{i,\,N_1-1;\,k-1} \;=\; v_{i,\,N_1;\,k-1}, \qquad v_{i,\,N_2+1;\,k-1} \;=\; v_{i,\,N_2;\,k-1} \tag{3.35}
$$

para todo $M_1 \le i \le M_2$. Claramente, estas condições permitem a computação do

esquema (3.32) para todos os valores $M_1 \leq i \leq M_2,\ N_1 \leq j \leq N_2$, de modo que todos os nodos da malha podem agora ser tratados pelo código como nodos internos (sendo, de fato, internos à camada artificial acrescentada). Finalmente, as condições especiais (3.2) referentes aos pontos singulares da solução $u(x,t)$ foram implementadas do seguinte modo, de modo a não prejudicarem a eficiência da computação: *após* a execução completa da instrução (3.32) em todos os pontos da malha, antes de se passar ao novo nível de tempo t_{k+1} os valores $v_{i,j;k}$ obtidos no nível t_k nos pontos da malha correspondentes às singularidades foram descartados, sendo redefinidos de acordo com as condições (3.2).

O algoritmo descrito acima para a solução do problema (3.1)-(3.3) — no contexto da equação (3.30) – foi, por questões de eficiência, implementado *vetorialmente* (ver e.g. [19, 35]), como mostra o código FORTRAN a seguir, correspondente à parte principal do programa serial desenvolvido, onde Q = $(p-2)/2$, H_TO_PM2 = h^{p-2}:

```
  T = T + DT

! ************************************************************** !
!            COMPUTATION OF  F VALUES  ON THE GRID:              !
! ************************************************************** !
! ********* F(I,J)  FOR  M1 ≤ I ≤ M2 + 1  AND  N1 ≤ J ≤ N2:

  F(M1:M2+1,N1:N2) = ( ( U(M1:M2+1,N1:N2) − U(M1−1:M2,N1:N2) )**2 +  &
      ( ( U(M1−1:M2,N1+1:N2+1) + U(M1:M2+1,N1+1:N2+1) ) −            &
        ( U(M1−1:M2,N1−1:N2−1) + U(M1:M2+1,N1−1:N2−1) ) )**2 / 16 )**Q / H_TO_PM2

! ************************************************************** !
!            COMPUTATION OF  G VALUES  ON THE GRID:              !
! ************************************************************** !
! ********* G(I,J)  FOR  M1 ≤ I ≤ M2  AND  N1 ≤ J ≤ N2 + 1:

  G(M1:M2,N1:N2+1) = ( ( U(M1:M2,N1:N2+1) − U(M1:M2,N1−1:N2) )**2 +  &
      ( ( U(M1+1:M2+1,N1−1:N2) + U(M1+1:M2+1,N1:N2+1) ) −            &
        ( U(M1−1:M2−1,N1−1:N2) + U(M1−1:M2−1,N1:N2+1) ) )**2 / 16 )**Q / H_TO_PM2

! ************************************************************** !
! COMPUTATION OF  V = [ NEW U VALUES AT NEW TIME T + DT ]:       !
! ************************************************************** !
! ********* V(I,J)  FOR  M1 ≤ I ≤ M2  AND  N1 ≤ J ≤ N2:
```

```
V(M1:M2,N1:N2) = U(M1:M2,N1:N2) +    &
   CFL * ( F(M1+1:M2+1,N1:N2) * ( U(M1+1:M2+1,N1:N2) - U(M1:M2,N1:N2) ) - &
           F(M1:M2,N1:N2) * ( U(M1:M2,N1:N2) - U(M1-1:M2-1,N1:N2) ) ) +      &
   CFL * ( G(M1:M2,N1+1:N2+1) * ( U(M1:M2,N1+1:N2+1) - U(M1:M2,N1:N2) ) - &
           G(M1:M2,N1:N2) * ( U(M1:M2,N1:N2) - U(M1:M2,N1-1:N2-1) ) )

! ********* EXTENDING  V  TO THE EXTRA (ARTIFICIAL) GRID POINTS:

V(M1-1,N1:N2) = V(M1,N1:N2);  V(M2+1,N1:N2) = V(M2,N1:N2)
V(M1-1:M2+1,N1-1) = V(M1-1:M2+1,N1); V(M1-1:M2+1,N2+1) = V(M1-1:M2+1,N2)
```

Note que, no código acima, o *array* bidimensional U$(\cdot,\cdot)$ contém supostamente os valores correspondentes a $v_{i,j;k-1}$ (previamente computados), referentes ao instante de tempo T, enquanto V$(\cdot,\cdot)$ recebe os novos valores computados, referentes ao instante de tempo T + DT, correspondentes aos valores $v_{i,j;k}$. Assim, antes do ciclo ser executado novamente, a variável U deve ser atualizada recebendo os valores V, além de se corrigirem os valores computados nos pontos singulares $a_1, a_2, ..., a_m$, que devem ser substituídos pelos valores especificados em (3.3). Supondo, por exemplo, $m = 3$, e que os índices de malha dos pontos a_1, a_2, a_3 sejam (i_1, j_1), (i_2, j_2) e (i_3, j_3), respectivamente, o código acima teria de ser continuado pelas instruções a seguir, antes de ser executado novamente.

```
! ************************************************************ !

  U = V     ! <— UPDATING U (AT THE NEW TIME LEVEL)

! ********* OVERRIDING COMPUTED VALUES AT SINGULAR POINTS:

  U(I1,J1) = B1;  U(I2,J2) = B2;  U(I3,J3) = B3

! ************************************************************ !
```

Para uma ideia do TEMPO DE EXECUÇÃO de *um* ciclo completo U → V → U acima, considerando-se uma malha computacional com 10^6 pontos (como no caso de se ter, por exemplo, $M_1 = -500$, $M_2 = 500$, $N_1 = -500$, $N_2 = 500$), o tempo de execução observado com um processador Intel I3 @ 3.20 GHz foi da ordem de 0.10 segundos (sendo o código de máquina gerado pelo compilador GFORTRAN do PROJETO GNU). Isso corresponde (para DT $\approx 10^{-4}$) a um tempo total de aproximadamente 11 dias

para obter-se a solução no instante, digamos, $t = 1000$ (a partir de $t = 0$), o que motiva a utilização de computadores paralelos para este problema. Uma vez obtida a solução $u(\cdot, t)$ num instante $t_* > 0$ suficientemente grande de modo a fornecer uma aproximação razoável para a solução estacionária, o VALOR DE CAMPO DISTANTE é então estimado por

$$b \;\approx\; \frac{1}{M} \sum_{(i,\,j)\,\in\,\mathcal{M}} u(x_i, y_j, t_*) \tag{3.36}$$

onde $\mathcal{M}$ é um conjunto de pontos da malha tomados suficientemente distantes da origem $(0,0)$ e também distantes da fronteira da malha, com cardinalidade M.

3.2. Inicialização da solução numérica

Para resolver o problema (3.1), (3.2), (3.3) — no contexto da equação (3.30) — falta ainda especificar a CONDIÇÃO INICIAL $u_0 \in W^{1,p}(\mathbb{R}^n) \cap L^\infty(\mathbb{R}^n)$, onde $n = 2$.

Para isto, "desenhamos", um plano com 3 cones ou 4 cones(por exemplo, se o problema tem 3 ou 4 singularidades). Porém nada impede, definir a condição inicial uma superfície curva ou um plano inclinado com 3 ou 4 paraboloides e as singularidades nos vértices destes paraboloides ou outra superfície inicial qualquer. Escolhemos os cones e o plano horizontal, apenas porque é mais fácil programar esta condição inicial. Os detalhes desta inicialização também podem ser encontrados no parágrafo **(*d*) Initial_Solution_u0** da seção **4.5. Detalhando o código** INITIALIZE.

Isso pode ser feito do seguinte modo: uma vez dados os pontos singulares (distintos) $a_1, a_2, ..., a_m$ e os valores correspondentes $b_1, b_2, ..., b_m$, respectivamente, que a solução $u(x,t)$ deve assumir nestes pontos, computa-se $r_0 > 0$ dado por

$$r_0 \;:=\; \frac{1}{2} \cdot \min\big\{\, d(a_i, a_j) \colon 1 \le i < j \le m \,\big\}, \tag{3.37}$$

onde $d(a_i, a_j) = |a_i - a_j|$ denota a distância do ponto a_i ao ponto a_j, ou seja,

$$d(a_i, a_j) \;=\; \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \tag{3.38}$$

(sendo $a_i = (x_i, y_i)$, $a_j = (x_j, y_j)$, respectivamente). Atribuindo-se, então, um valor inicial b_0 próximo (mas propositadamente não igual) de `b_esp = (b1 + b2 + b3)/3` supondo-se aqui, por exemplo, 3 singularidades. Definimos:

$$u_0(x,y) := \begin{cases} b_0 \qquad\quad \text{, se } (x,y) \text{ está fora dos cones,} \\ b_j - \dfrac{(b_j - b_0)}{r_0}\sqrt{(x-x_j)^2 + (y-y_j)^2} \quad \text{, } (x,y) \in \text{cone j} \end{cases} \tag{3.39}$$

Onde a segunda linha de (3.39) significa que o ponto (x, y) está na base de algum cone, como ilustrado na Fig. 3.1 a seguir (no caso, $m = 3$, $a_1 = (1, 1)$, $a_2 = (-1.5, 0.5)$, $a_3 = (0, -1.5)$, $b_1 = 1$, $b_2 = 3$, $b_3 = 4$, com $b_0 = 2$). E nas figuras: Fig. 3.4 e Fig. 3.5 para t=0.

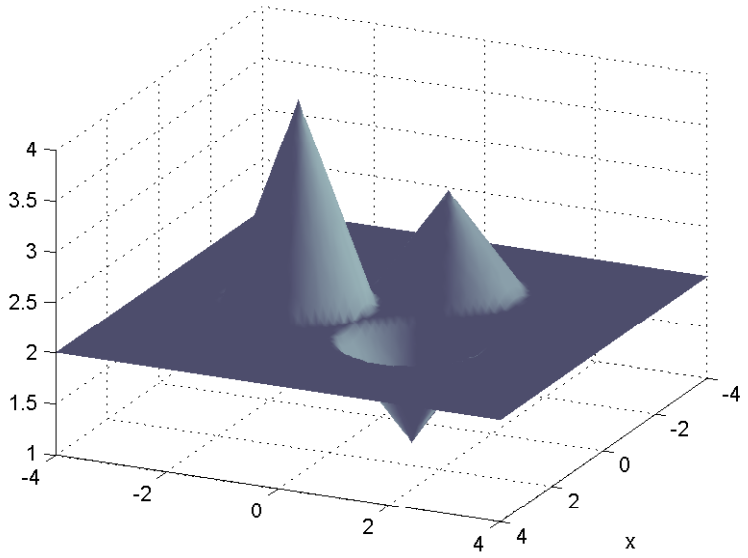


Fig. 3.1: Exemplo de estado inicial para o problema (3.1), (3.2), (3.3)

3.3. Estimativa do tempo via solução de Barenblatt

Para obter uma aproximação razoável para o estado estacionário, é preciso computar $u(\cdot, T)$ para $T \gg 1$ sufientemente grande. O ARGUMENTO HEURÍSTICO desenvolvido a seguir dá uma ideia do tamanho aproximado que T deve ter. Para isso, vamos considerar momentaneamente que $u(\cdot, t)$ seja uma *solução fundamental* (mais detalhes no Apêndice D ou ver [34], Ch. 2) da equação $u_t = \Delta_p u$ (Solução de Barenblatt) dada por

$$u(x,t) \;=\; v_*(x,t) \;=\; t^{-n\alpha} \times \Big(C \;-\; \alpha^{\frac{1}{p-1}} \cdot \frac{p-2}{p} \left(r \cdot t^{-\alpha} \right)^{\frac{p}{p-1}} \Big)_+^{\frac{p-1}{p-2}}$$

onde $r = |\, x - x_0 |$ para certo $x_0 \in \mathbb{R}^n$, $\alpha = 1/(n(p-2)+p)$, e onde $C > 0$ é uma constante dada, relacionada com a *massa* $\mathsf{m} = \int_{\mathbb{R}^n} u(x,t)\, dx$ da solução $u(\cdot,t)$, cujo valor independe do tempo t e pode ser obtido explicitamente, sendo dado por:

$$\mathsf{m} \;=\; \omega_n \cdot \frac{p-1}{p}\, \alpha^{\,-\frac{n}{p}} \cdot \Big(\frac{p}{p-2} \Big)^{\frac{n(p-1)}{p}} \cdot \; C^{\frac{(p-1)\,(p+n(p-2))}{p(p-2)}} \cdot B\big(\frac{n(p-1)}{p},\, \frac{2p-3}{p-2} \big)$$

onde $B(\cdot,\cdot)$ denota a FUNÇÃO BETA DE EULER e ω_n é a área $(n-1)$-dimensional da superfície da bola unitária (ou seja, da hipersuperfície $\{\, x \in \mathbb{R}^n \colon |\, x\, | = 1 \,\}$), dada por

$$\omega_n \;=\; \frac{2\,\pi^{n/2}}{\Gamma(n/2)}$$

(ver e.g. [11], p. 8), onde $\Gamma(\cdot)$ denota a FUNÇÃO GAMA DE EULER. expressando a constante C em termos da massa m, obtém-se, das expressões acima,

$$C \;=\; \left\{ \frac{\mathsf{m}}{\omega_n} \cdot \frac{p}{p-1} \cdot \Big(\frac{1}{p+n(p-2)} \Big)^{\frac{n}{p}} \cdot \Big(\frac{p-2}{p} \Big)^{\frac{n(p-1)}{p}} \Big/\; B\big(\frac{n(p-1)}{p},\, \frac{2p-3}{p-2} \big) \right\}^{\frac{p(p-2)}{(p-1)\,(p+n(p-2))}}$$

onde, como dito acima, $B(\cdot,\cdot)$ é a função Beta. Sendo o suporte de $v_*(\cdot,t)$ dado pela bola fechada de centro x_0 e raio $R(t)$, com

$$R(t) \;=\; C^{\,\frac{p-1}{p}} \cdot \big(p + n(p-2) \big)^{\frac{1}{p}} \cdot \Big(\frac{p}{p-2} \Big)^{\frac{p-1}{p}} \cdot \; t^{\,\frac{1}{p+n(p-2)}}$$

para todo $t > 0$. Segue assim que o tempo T para que se alcance uma distância $R > 0$ do centro x_0 é dado por

$$T \;=\; \left\{ R \cdot \big(p + n(p-2) \big)^{-\frac{1}{p}} \cdot \Big(\frac{p-2}{p} \Big)^{\frac{p-1}{p}} \cdot \; C^{\,-\frac{p-1}{p}} \right\}^{p+n(p-2)}$$

onde C depende da massa m, como mostrado acima. Equivalentemente, tem-se: para $v_*(\cdot,t)$ atingir uma distância $R > 0$ do centro x_0, é preciso aguardar um tempo $T = T\,(R, \mathsf{m}, p)$ dado por

$$T(R, \mathsf{m}, p) \;=\; K(n,p)\, \frac{1}{\mathsf{m}^{p-2}}\, R^{\,p+n(p-2)} \tag{3.40}$$

onde $K = K(n,p)$ denota a constante dada por

$$K(n,p) \;=\; \Big(\frac{1}{p+n(p-2)}\Big)^{\frac{p+n(p-2)}{p}} \cdot \Big(\frac{p-2}{p}\Big)^{\frac{(p-1)(p+n(p-2))}{p}} \cdot \; K_*(n,p)$$

com $K_*(n,p)$ dada por

$$K_*(n,p) \;=\; \Big\{\frac{1}{\omega_n}\cdot\frac{p-1}{p}\cdot\big(p+n(p-2)\big)^{\frac{n}{p}}\cdot\Big(\frac{p}{p-2}\Big)^{\frac{n(p-1)}{p}}\cdot\; B\big(\frac{n(p-1)}{p},\,\frac{2p-3}{p-2}\big)\Big\}^{p-2}$$

onde, como antes, $B(\cdot,\cdot)$ denota a função Beta. Este resultado pode ser usado do seguinte modo para se estimar um tempo mínimo T esperado para se levar a um término aproximado a interação dos m cones individuais que formam o estado inicial u_0, como ilustrado na Fig. 3.1. Supondo tais cones de massa — ou seja, volume com relação ao plano horizontal de altura b_0 — de ordem unitária (digamos, $\mathsf{m} = 1$) e com centros a_j distando entre si não mais que 10 unidades de comprimento, pode-se imaginar que valores da solução a uma distância dez vezes maior — correspondentes

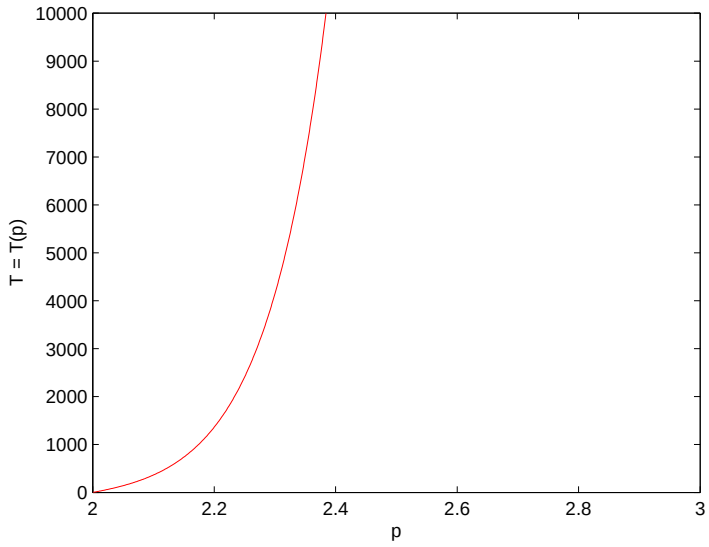


Fig. 3.2: Estimativa para o tempo de interação no caso $R = 100$, $\mathsf{m} = 1$, $n = 2$

a se considerar $R = 100$ em (3.40) acima — possam fornecer aproximações razoáveis para o VALOR DE CAMPO DISTANTE b procurado, visto que o valor b é resultado da interação não linear entre os vários pulsos (ou ondas) constituindo o estado inicial, para tempo suficientemente grande.

A Fig. 3.2 acima mostra a dependência do tempo de observação T assim previsto

com relação ao parâmetro $p > 2$, para valores típicos de R e m (e dimensão $n = 2$). Na prática, observou-se de fato a necessidade de considerar valores para p não muito superiores a 2 (nos experimentos, considerou-se geralmente $p \in (2,\, 2.3\,]$) de modo a se esperar T não muito superior a 1000 unidades, ou seja, $T = O(1000)$. Esta situação é ilustrada na Fig. 3.3 a seguir, onde se mostra a evolução no tempo t da estimativa do VALOR DE CAMPO DISTANTE b obtida via amostragem como indicado em (3.36): valores maiores de p requerem que se tenha de computar a solução $u(\cdot, t)$ para tempos significativamente maiores.

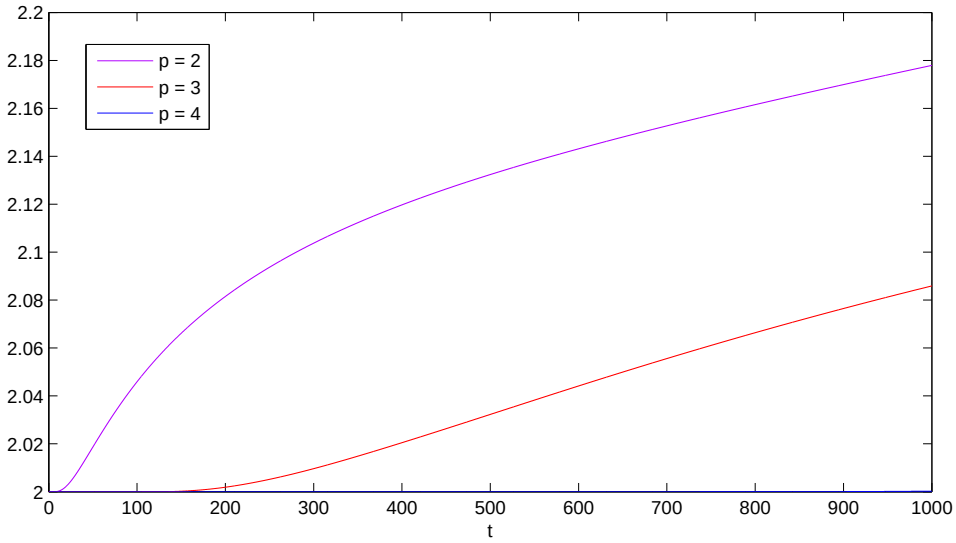
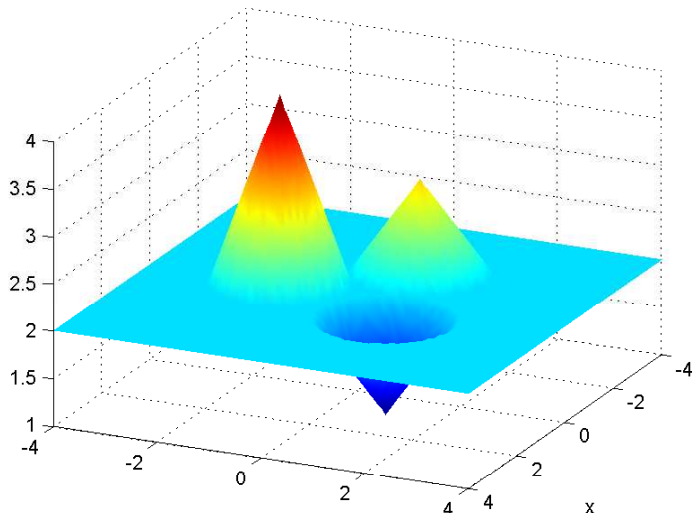
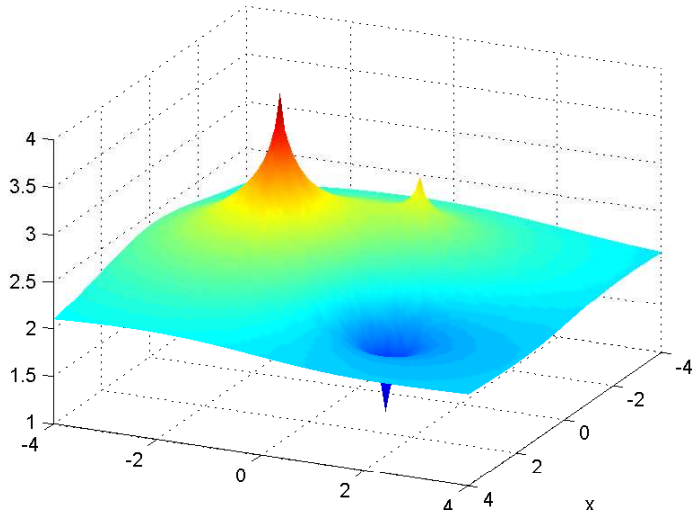


Fig. 3.3: Evolução temporal de b obtido via (3.36) nos casos $p = 2$, $p = 3$ e $p = 4$

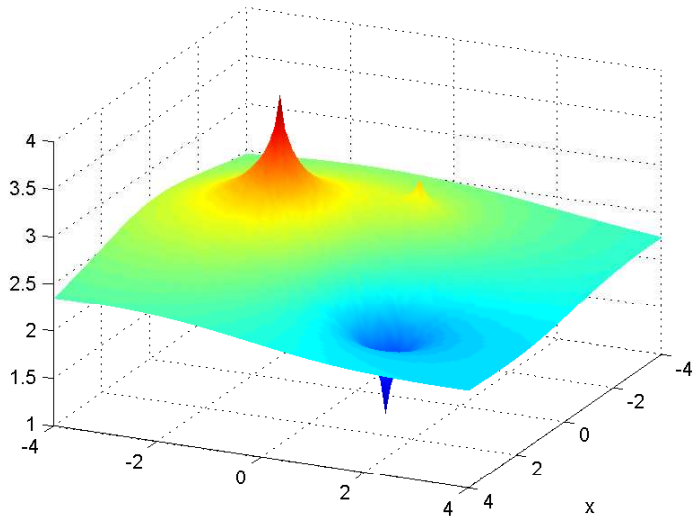
Outro aspecto interessante nas simulações é observar o *comportamento local* da solução computada $u(\cdot, t)$ na vizinhança dos pontos singulares $a_1, a_2, \ldots, a_m$, como ilustrado nas Figuras 3.4 e 3.5 a seguir, ambas obtidas a partir de um mesmo estado inicial u_0 (descrito anteriormente na Fig. 3.1 acima) mas considerando-se diferentes valores de p. Em ambos os casos, verifica-se experimentalmente o comportamento anunciado em (2.9), Capítulo 2, se $t \gg 1$ for suficientemente grande. Além disso, também no caso evolutivo (parabólico) a solução $u(\cdot, t)$ permanece sempre limitada pelos valores $b = \min\{\, b_j\colon 1 \le j \le m \,\}$ e $B = \max\{\, b_j\colon 1 \le j \le m \,\}$ (compare com (2.8), Cap. 2), o que foi sempre observado nos experimentos numéricos realizados.



$t = 0$



$t = 10$



$t = 100$

Fig. 3.4: Solução $u(\cdot, t)$ do problema (3.1), (3.2), (3.3) se $p = 2.3$

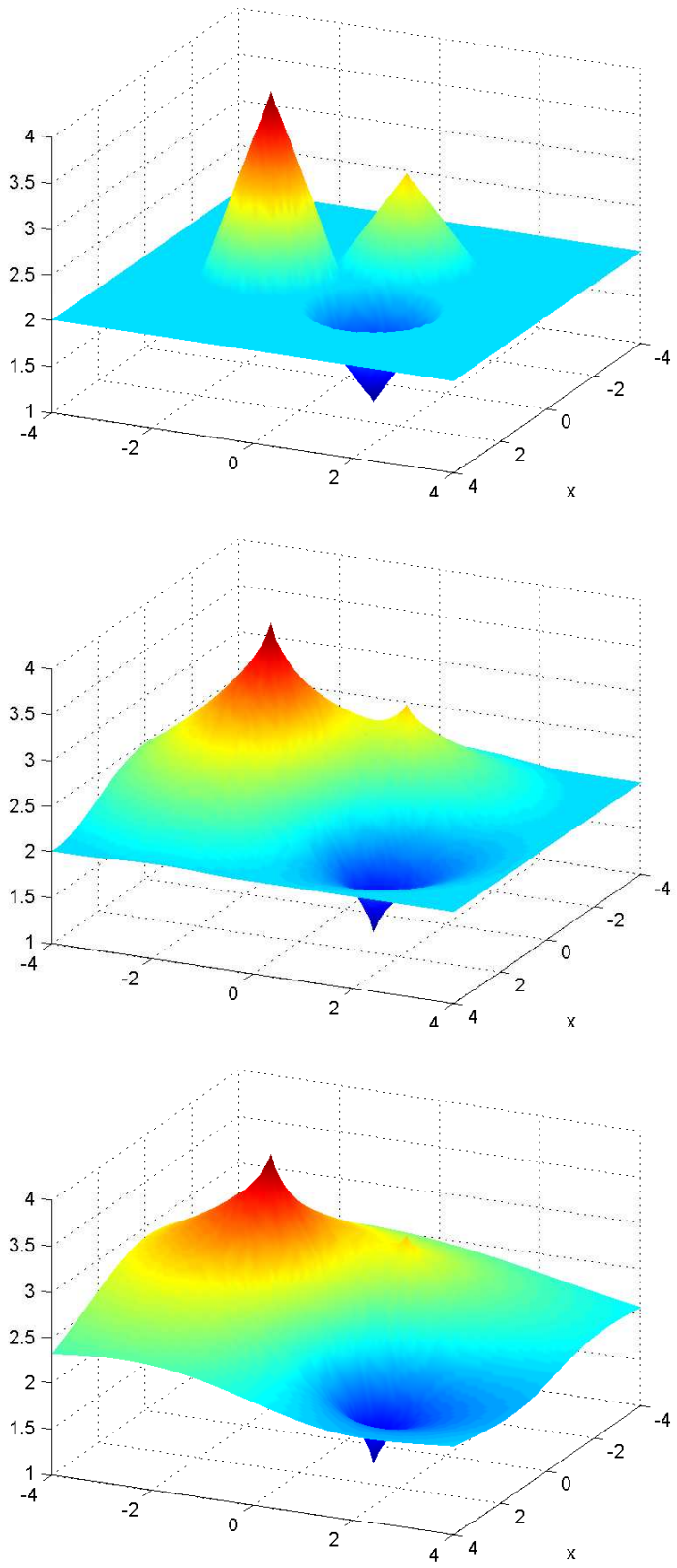
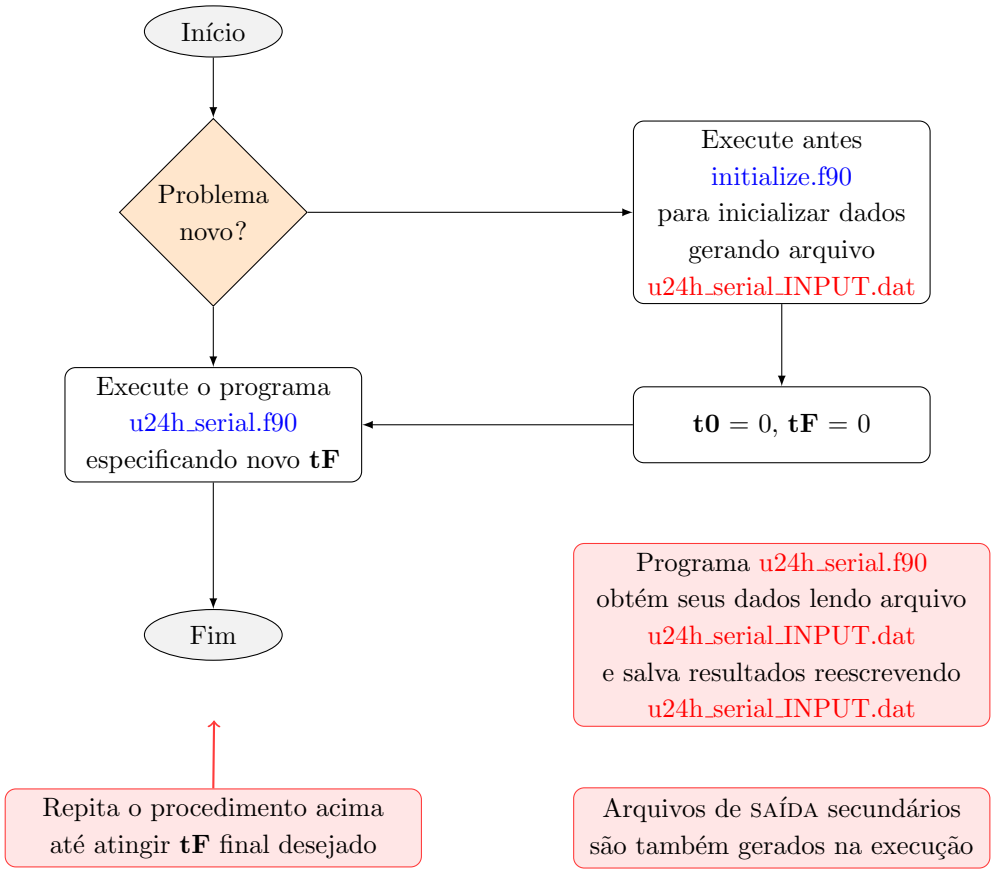
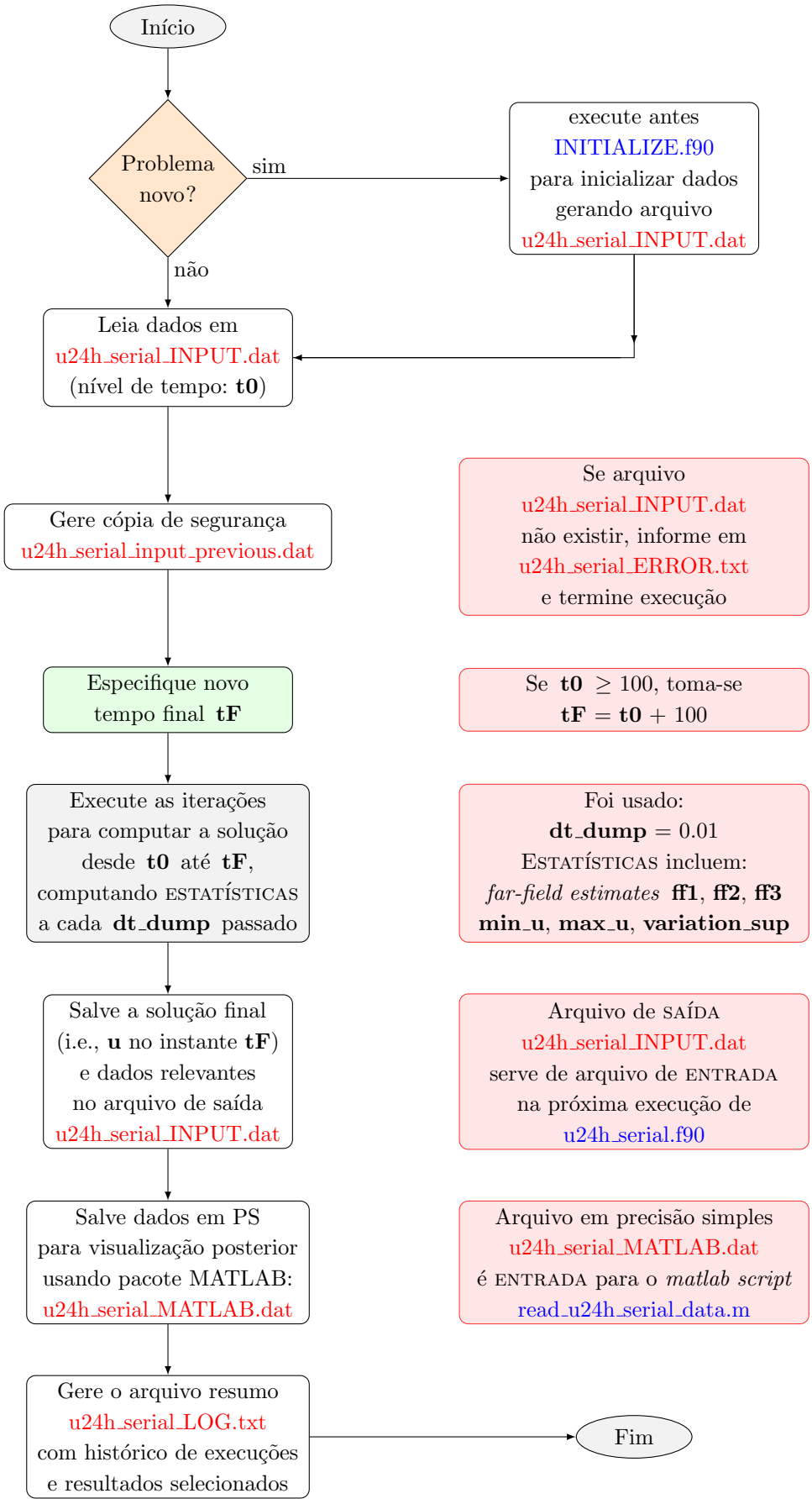


Fig. 3.5: Solução $u(\cdot, t)$ do problema (3.1), (3.2), (3.3) se $p = 4$

3.4. Procedimentos computacionais

A execução dos experimentos computacionais conduzidos em *uma* CPU foi feita desenvolvendo-se dois programas FORTRAN seriais (ou sequenciais): initialize.f90 (responsável pela inicialização de todas as variáveis envolvidas, incluindo a definição da malha numérica e do estado inicial u_0 considerado) e u24h_serial.f90 (responsável pelo cálculo da solução numérica, até um dado instante de tempo **tF** especificado, sendo este programa executado repetidamente (em sucessão) até ser observado um estado final estacionário dentro da margem de erro adotada, de ordem 10^{-4} ou 10^{-5}). A inicialização de dados feita pelo programa initialize.f90 gera um arquivo de SAÍDA de nome u24h_serial_INPUT.dat, que serve de ENTRADA para u24h_serial.f90 informando todos os dados necessários para sua execução, com exceção do nível de tempo final **tF** a ser atingido a partir do instante inicial **t0** $= 0$ na primeira execução de u24h_serial.f90. Este processo é representado esquematicamente a seguir.




Início
Problema novo?
sim
não
execute antes
INITIALIZE.f90
para inicializar dados
gerando arquivo
u24h_serial_INPUT.dat
Leia dados em
u24h_serial_INPUT.dat
(nível de tempo: t0)
Gere cópia de segurança
u24h_serial_input_previous.dat
Se arquivo
u24h_serial_INPUT.dat
não existir, informe em
u24h_serial_ERROR.txt
e termine execução
Especifique novo
tempo final tF
Se t0 ≥ 100, toma-se
tF = t0 + 100
Execute as iterações
para computar a solução
desde t0 até tF,
computando ESTATÍSTICAS
a cada dt_dump passado
Foi usado:
dt_dump = 0.01
ESTATÍSTICAS incluem:
far-field estimates ff1, ff2, ff3
min_u, max_u, variation_sup
Salve a solução final
(i.e., u no instante tF)
e dados relevantes
no arquivo de saída
u24h_serial_INPUT.dat
Arquivo de SAÍDA
u24h_serial_INPUT.dat
serve de arquivo de ENTRADA
na próxima execução de
u24h_serial.f90
Salve dados em PS
para visualização posterior
usando pacote MATLAB:
u24h_serial_MATLAB.dat
Arquivo em precisão simples
u24h_serial_MATLAB.dat
é ENTRADA para o matlab script
read_u24h_serial_data.m
Gere o arquivo resumo
u24h_serial_LOG.txt
com histórico de execuções
e resultados selecionados
Fim

Como mostrado nos fluxogramas acima, os resultados obtidos por u24h_serial.f90 são salvos novamente no mesmo arquivo u24h_serial_INPUT.dat, que fornece os dados necessários (com exceção do novo instante final **tF**) para uma nova execução (se desejada) deste programa, repetindo-se o ciclo até que um instante final **tF** definitivo tenha sido alcançado.

Para fins de ilustração, descrevemos agora o processo básico para a execução dos programas acima no supercomputador SGI ALTIX 1350/450, disponível no Centro Nacional de Processamento de Alto Desempenho de São Paulo (cenapad-sp), localizado na Universidade Estadual de Campinas (https://www.cenapad.unicamp.br). Este centro faz parte do chamado Sistema Nacional de Processamento de Alto Desempenho (https://www.lncc.br/sinapad), que consiste atualmente de nove sistemas similares (de maior ou menor porte) distribuídos pelo país: LNCC (Petrópolis), UFRGS (Porto Alegre), Unicmap (Campinas), CPTEC/INPE (Cachoeira Paulista), COPPE/UFRJ (Rio de Janeiro), UFMG (Belo Horizonte), UFC (Fortaleza), UFPE (Recife) e INPA (Manaus), mantidos pelo MCT (Ministério da Ciência, Tecnologia e Informação) do governo brasileiro. O acesso a estes sistemas se dá remotamente — utilizando-se os protocolos SSH (Secure Shell) e FTP (File Transfer Protocol) — via a Rede Nacional de Ensino e Pesquisa (https://www.rnp.br), criada pelo MCT (na época, Ministério da Ciência e Tecnologia) em 1989, e mantida pelo MCT desde então. Em cada centro do SINAPAD, contas de usuários são obtidas a partir da aprovação de projetos científicos encaminhados, de natureza computacional, que requeiram emprego de computação de alto desempenho. Para instalação dos protocolos SSH (acesso remoto) e FTP (transferência de arquivos) numa máquina-cliente Windows, por exemplo, é particularmente conveniente instalar os softwares WinSCP (https://winscp.net/eng/download.php) e PuTTY (https://www.putty.org), que permitem uma fácil utilização prática destes recursos.

No caso do ambiente ALTIX do cenapad-sp, por exemplo, depois de efetuado o acessso via `ssh` ao servidor de conexões (`cenapad.unicamp.br`), ou *front-end* do sistema, deve-se acessar (*login*) a máquina SGI ALTIX 1350/450 (de nome `athenas`), com sistema operacional Linux SUSE (ia64), executando-se o comando

```
rhoo@frontend:~$ ssh athenas
```

(e fornecendo-se a senha logo a seguir, que é a mesma senha usada para as outras máquinas do cenapad-sp: SGI Altix ICE 8400 LX, IBM P750 e IBM iDataPlex/GPU). Entrando-se no diretório onde foram colocados os programas fontes initialize.f90 e

u24h_serial.f90, estes devem ser compilados, executando-se (na máquina `athenas`)

```
rhoo@sg2gq1:~> gfortran -o initialize  initialize.f90
rhoo@sg2gq1:~> gfortran -o u24h_serial u24h_serial.f90
```

se for desejado usar o compilador `gfortran` (GNU fortran), ou

```
rhoo@sg2gq1:~> ifort -o initialize  initialize.f90
rhoo@sg2gq1:~> ifort -o u24h_serial u24h_serial.f90
```

no caso do compilador `ifort` (Intel), ambos disponíveis no sistema (supondo já terem sido executados os arquivos de script correspondentes,

```
rhoo@sg2gq1:~> source /usr/local/bin/gnuvars.sh
```

para o uso de `gfortran`, e

```
rhoo@sg2gq1:~> source /usr/local/bin/intelvars.sh
```

no caso de uso do compilador `ifort`. (Alternativamente, estes comandos podem ser inseridos no arquivo de inicialização `.bashrc`.) O caminho completo dos executáveis `gfortran` e `ifort` pode ser obtido com o comand unix `which`: executando

```
rhoo@sg2gq1:~> which gfortran
rhoo@sg2gq1:~> which ifort
```

(na máquina `athenas`), obtém-se, respectivamente, as respostas

```
/usr/local/gcc-5.3.0/bin/gfortran
/usr/local/intel/Compiler/11.1/075/bin/ia64/ifort
```

(dadas pelo sistema). A execução dos programas `gfortran` ou `ifort` para a compilação de um dado código FORTRAN pode ser feita em *modo interativo* (ou seja, digitando os comandos acima diretamente na tela e pressionando em seguida a tecla de RETURN, como feito acima), mas para a execução dos códigos gerados (`initialize` e `u24h_serial`) deve-se proceder em *modo batch*, submetendo-se um arquivo *script* via o comando `qsub` do sistema PBS (Portable Batch System). Alguns dos principais comandos PBS são, por exemplo:

comando PBS	ação
`qsub`	para submeter jobs (definidos em *shell script files*) ao PBS
`qdel`	para remover jobs das filas de execução
`qstat`	para exibir o status de filas e jobs

Para mais detalhes sobre comandos PBS, ver [4] e https://www.cenapad.unicamp.br/parque/SGI_div.shtml. Programas executados em modo interativo no SGI ALTIX são executados no servidor `athenas` (hostname: `sg2gq1.cenapad.unicamp.br`), tendo sua execução automaticamente interrompida se forem usados mais de 5 minutos de CPU em um intervalo de 10 minutos (com o usuário responsável recebendo neste caso advertência por e-mail). A máquina `sg2gq1` é interativa e destina-se exclusivamente a compilação de programas, criação e edição de arquivos, submissão de jobs ao Sistema de Filas (PBS) e conferência de resultados. De fato, existe um procedimento de monitoramento e controle para evitar que processos muito "pesados" sejam executados nessa máquina. Todos os demais processamentos devem ser feitos necessariamente através de submissão ao Sistema de Filas (i.e., em modo batch).

Em modo de execução serial (i.e., um processador), o sistema SGI ALTIX 1350/450 do CENAPAD-SP oferece três filas para processamento em batch, assim configuradas:

Fila serial	Num. max. jobs	Tempo max. CPU
`pequena`	4	24 horas
`media`	7	168 horas
`grande`	8	744 horas

Com base nestes parâmetros, escolhe-se a fila apropriada para cada job a ser submetido, levando-se em conta o tempo de CPU necessário (estimado) e, depois disso, o número de jobs já em execução em cada fila. O programa initialize.f90 fornece informações que permitem estimar com razoável confiança o tempo de processamento requerido na execução do programa u24h_serial.f90. Para o segundo item, executa-se (em modo interativo na máquina `athenas`) o comando PBS

```
rhoo@sg2gq1:~> qstat -a
```

sobre o status de todas as filas (seriais e paralelas), obtendo-se a lista de todos os jobs presentemente em execução (estado: `Running`) nas diversas filas e o tempo de

execução total de cada um destes jobs no momento atual, além da lista de jobs em espera (`queued`), se existirem, que aguardam seu processamento nas várias filas.

Na primeira simulação de um problema novo, tipicamente foi submetido um arquivo script como o seguinte ao sistema PBS:

```
#PBS -q pequena
#PBS -N job01
#PBS -V
#!/bin/sh
cd $PBS_O_WORKDIR
echo "************************************"
echo
echo "Inicio do job01:  " `date`
echo "Hostname:  " `hostname`
echo
echo "---------------------------------------"
echo "TASK:"
echo "Executing ./initialize ..."
echo "Date started:  " `date`
./initialize
echo "Done!"
echo "Date finished:  " `date`
echo "---------------------------------------"
echo "Executing ./u24h_serial ..."
echo "Date started:  " `date`
./u24h_serial
echo "Done!"
echo "Date finished:  " `date`
echo "---------------------------------------"
echo
echo "Final do job01:  " `date`
echo
echo "************************************"
```

Em tais scripts, linhas começando com `#PBS` são DIRETIVAS PBS (no caso acima, definindo a fila escolhida (`pequena`) e o nome do job (`job01`), e declarando que as variáveis de ambiente dos comandos do qsub usados (como `PBS_O_WORKDIR` acima) devem ser exportadas para o *batch job* correspondente); linhas começando com `#` são simplesmente comentários, e `#!/bin/sh` define o interpretador *shell* em uso. Por *default*, scripts PBS são executados no DIRETÓRIO HOME, e *não* no diretório onde se deu a submissão, a menos que se inclua no script o comando `cd $PBS_O_WORKDIR` (que fornece o endereço do diretório onde o arquivo batch em questão foi iniciado).

As variáveis `date` e `hostname` nos comandos de impressão echo permitem que sejam registrados a data/horário no momento de execução do comando echo, e o nome do nó (hostname) onde o job foi processado, respectivamente. Finalmente, ./initialize e ./u24h_serial são os comandos de usuário solicitando que os arquivos executáveis com estes nomes, presentes no diretório corrente, sejam executados pelo sistema (estes nomes foram pré-definidos nos comandos de compilação executados previamente, conforme mostrado anteriormente). Outros comandos úteis poderiam ter sido incluídos no arquivo batch acima, como, por exemplo, o par

```
#PBS -m bea
#PBS -M robertohoo@gmail.com
```

especificando que seja enviado e-mail para o endereço indicado no início (b) e fim (e) da execução do job, ou em caso de aborto (a) por qualquer razão. Para ainda outros comandos e variáveis de ambiente PBS, pode-se consultar as PÁGINAS DE MANUAL (*manpages*) do comando qsub executando-se (por exemplo)

```
rhoo@sg2gq1:~> man qsub > qsub_manpages
```

(em modo interativo na athenas), e similarmente para os comandos qstat e qdel (ver abaixo) ou qualquer outro comando desejado.

Supondo que o arquivo batch acima tenha sido chamado p_Laplace (ou outro nome qualquer, incluindo job01), este seria submetido ao sistema PBS, então, executando-se o comando

```
rhoo@sg2gq1:~> qsub p_Laplace
```

(em modo interativo na athenas), que submete um job de nome job01 para a fila de jobs pequena, como indicado no arquivo p_Laplace. O status deste job pode ser checado a qualquer momento executando-se o comando PBS

```
rhoo@sg2gq1:~> qstat -u rhoo
```

(em modo interativo na athenas), onde a opção -u indica que se deseja o estado de todos os jobs do usuário rhoo (que é o username de Roberto Hoo no CENAPAD-SP). Aliás, como visto nas ilustrações acima, o nome do usuário está sempre incluído no PROMPT do sistema (no caso da athenas: rhoo@sg2gq1:~>, e analogamente nas outras máquinas). Assim, para citar um exemplo, executando-se qstat -u rhoo

sete segundos após a submissão do script `p_Laplace`, uma resposta típica seria

```
sg2gq1.cenapad-sp.br:

                                                                  Req'd    Req'd       Elap
Job ID                  Username    Queue    Jobname          SessID  NDS   TSK   Memory  Time    S  Time
----------------------- ----------- -------- ---------------- ------ ----- ------ ------ --------- - ---------
13277.sg2gq1.cenapad-s  rhoo        pequena  job01             62254   --       1    --   25:00:00 R  00:00:07
```

A informação dada acima tem o seguinte significado. Na coluna de IDENTIFICAÇÃO DO JOB (`Job ID`) aparece o número atribuído pelo sistema ao job (no caso, `13277`), pelo qual ele pode ser referenciado em outros comandos PBS. (Por exemplo: se for desejado cancelar o processamento do job (interrompendo e cancelando, em particular, a execução de um programa FORTRAN que esteja sendo executado no momento), executa-se neste caso o comando

```
rhoo@sg2gq1:∼> qdel 13277
```

em modo interativo na `athenas`). Ao lado do NÚMERO DO JOB, aparece a IDENTIFICAÇÃO DO USUÁRIO (`Username`), seguido da IDENTIFICAÇÃO DA FILA (`Queue`) onde está o job (neste caso, a fila `pequena`, como solicitado pela diretiva `#PBS -q pequena` no arquivo script `p_Laplace` acima) e o NOME DO JOB (`job01`), determinado pela diretiva `#PBS -N job01`. Seguem a IDENTIFICAÇÃO DA SESSÃO (`SessID`), o NÚMERO DE NÓS ou NODES solicitado (`NDS`), determinado pela diretiva

`#PBS -l nodes=`$\langle m \rangle$

(que, nas filas seriais, só pode ser usada com o valor default $m = 1$, e assim não é necessária; outro valor para m faria o processamento ser imediatamente abortado), e o NÚMERO DE TASKS ou processos, que em filas seriais é sempre 1. No campo seguinte aparece a quantidade de memória RAM solicitada (através da diretiva `#PBS -l mem=`$\langle n \rangle$`mb`, não utilizada), seguido do TEMPO TOTAL PERMITIDO (`Req'd Time`), determinado pelo limite máximo da fila em questão (para a fila `pequena`: 25 horas). A seguir, aparece o campo mostrando o STATUS (`S`) corrente do job, que pode ser `R` (*running*) ou `Q` (*queued*) — estando as filas cheias, um job pode permanecer vários dias em estado de espera (`Q`), até ter sua execução iniciada. Finalmente, é informado o TEMPO TOTAL DECORRIDO (`Elap`sed `Time`) do job em estado *running*, que não pode ultrapassar o valor máximo da fila onde ele se encontra (campo `Require`d `Time`, à esquerda), neste caso limitado por 25 horas (fila `pequena`).

O comando `qsub p_Laplace` gera um *batch job* para a fila solicitada a partir do arquivo script `p_Laplace`, sendo este batch job executado em algum nó (node) do SGI ALTIX atribuído pelo sistema. O comando `echo "Hostname:" `hostname`` presente no arquivo script (neste caso, `p_Laplace`) imprime o nome do nó que executa o job num ARQUIVO TEXTO de saída (com NOME DEFAULT dado por ⟨`Jobname`⟩.o⟨`JobID`⟩, neste caso: `job01.o13277`). Neste arquivo também são escritas todas as mensagens geradas pelos códigos executáveis do job (em particular, aqueles gerados pelos programas FORTRAN presentes) dirigidas para o STDOUT (ou seja, tela ou monitor). Na ocorrência de erros(s), mensagens correspondentes são impressas pelo sistema num ARQUIVO DE ERROS, com nome default dado por ⟨`Jobname`⟩.e⟨`JobID`⟩, ou seja, neste caso: `job01.e13277`. Estes dois arquivos STDOUT e STDERR podem ser reunidos em um só se for incluída a diretiva `#PBS -j oe` no arquivo batch.

Consultando o arquivo `job01.o13277` gerado na execução do job `job01` acima, encontra-se (por exemplo) o texto inicial

```
************************************

Inicio do Job01:  Fri Sep 22 01:33:01 BRT 2017

ALTIX Queue:  pequena
Hostname:  sg3no2

----------------------------------------
```

identificando o nó `sg3no2` (HOSTNAME completo: `sg3no2.cenapad.unicamp.br`) presente em um dos quatro FRAMES que compõem o sistema SGI ALTIX 1350/450. De acordo com a documentação oferecida pelo CENAPAD-SP (disponível no endereço https://www.cenapad.unicamp.br/parque/SGI_div.shtml), o nó `sg3no2` é formado por 38 processadores Intel Itanium2 dual-core série 9000 @1.6GHz, o que significa na prática um total de 76 processadores, ou núcleos de processamento, cada um com desempenho máximo de 6.4 GFLOPS. Um destes 76 processadores foi encarregado pelo sistema de executar as tarefas indicadas no job `job01` acima. Isso foi seguido por sucessivas submissões subsequentes de novos jobs similares (estes, agora, *não* incluindo o executável initialize) até se atingir um nível de tempo **tF** apropriado para o problema matemático considerado. Este procedimento foi repetido nos vários experimentos realizados, considerando diversas instâncias do problema (3.1), (3.2), (3.3). As figuras abaixo mostram tempos de execução típicos verificados nos experimentos, considerando ciclos de 100 iterações temporais (correspondentes a avanços no nível de tempo de magnitude $\Delta t = 0.01$), aproximadamente proporcionais ao

tamanho da malha computacional (como esperado), com flutuações devidas às permanentes oscilações nas condições de ocupação da memória local e processadores usados, como ocorreria num computador pessoal, e um desempenho médio típico de 50 a 70 MFLOPS, ou 1 % da capacidade teórica máxima (6 a 6.4 GFLOPS).

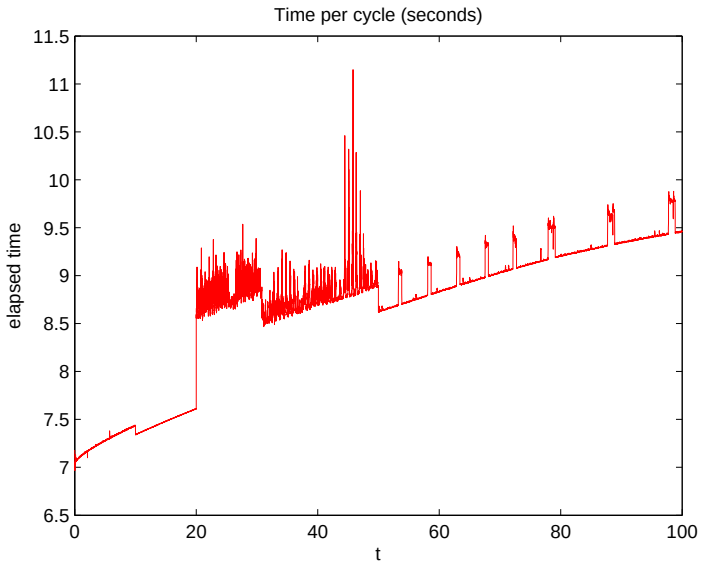


Fig. 3.6a: Tempos de processamento serial para avanços de 0.01 no tempo usando malha numérica de tamanho 501×501 (sendo $p = 2.3$)

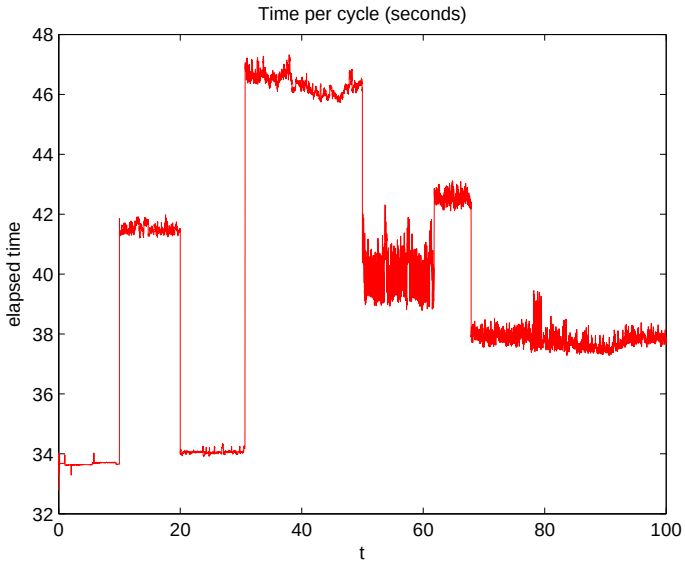


Fig. 3.6b: Tempos de processamento serial para avanços de 0.01 no tempo usando malha numérica de tamanho 1001×1001 (sendo $p = 4$)

3.5. Visualização com matlab

Nesta seção mostraremos parte do pós-processamento realizado com os resultados numéricos obtidos (na etapa de *computação numérica*, ou *number crunching*) para a finalidade de visualização (correspondente à etapa de *visualização científica*) e interpretação dos resultados, que gerou, em particular, as ilustrações mostradas no texto. Toda a visualização foi realizada utilizando-se o software MATLAB (ver e.g. [15, 16]) e suas rotinas de geração de gráficos (como meshgrid, plot, surf e surfl). Para isso, os dados de SAÍDA (computados originalmente em PRECISÃO DUPLA) foram arredondados em PRECISÃO SIMPLES e impressos para este propósito num arquivo especial chamado u24h_serial_MATLAB.dat, lido por um script em MATLAB. A razão deste cuidado é que alguns compiladores FORTRAN imprimem números muito pequenos (entre $2^{-1022} \approx 2.2 * 10^{-308}$ e 10^{-100} no caso de aritmética IEEE 754, considerando PRECISÃO DUPLA) *omitindo* o símbolo do CAMPO INDICADOR DE EXPOENTE (`D`, `E`, `d` ou `e`, todos aceitos pelo MATLAB), causando uma leitura equivocada pelo MATLAB. Por exemplo, um valor numérico $3.567 * 10^{-201}$ pode ser escrito como $3.567 - 201$, causando confusão em sua leitura pelo MATLAB. Analogamente, números em precisão dupla de magnitude superior a 10^{100} podem ser impressos sem o delimitador explícito de expoente e causar erros de leitura semelhantes no MATLAB (que precisa sempre receber os delimitadores `D`, `E`, `d` ou `e` nestes casos). Este problema pode ser evitado com a precaução de se imprimirem os resultados em PRECISÃO SIMPLES, com o uso das FUNÇÕES INTRÍNSICAS `real`$(\cdot)$ ou `sngl`$(\cdot)$ para o arredondamento em questão. Deste modo, um valor como $3.567 * 10^{-201}$ (em PRECISÃO DUPLA) seria arredondado para zero (em PRECISÃO SIMPLES), e assim impresso, não causando problemas de leitura no MATLAB, enquanto $3.567 * 10^{+201}$ seria impresso (em PRECISÃO SIMPLES) como `Infinity`, e subsequentemente reescrito (usando-se um editor de texto convencional) como `inf` ou `Inf`, que são as formas aceitas pelo MATLAB. Com estes cuidados, o arquivo de dados fornecido ao MATLAB para a visualização será sempre recebido corretamente.

Com o arquivo de dados gerado como descrito no parágrafo anterior, sua leitura para uso pelo MATLAB pode ser com o uso dos COMANDOS MATLAB `fopen` (para a abertura do arquivo de dados em disco), `scanf` (para a leitura dos dados no arquivo) e `fclose` (para o fechamento do arquivo ao fim da leitura). Por exemplo: supondo que se tivesse uma certa quantidade de dados (digamos, 10 dados) dispostos de modo variável ao longo de várias linhas (digamos, 5 linhas) em um arquivo de nome input.dat, correspondentes a variáveis de nomes `massa` e `t0` (na primeira linha), `N` (uma variável inteira, na terceira linha), e sete valores `x(1)`,...,`x(7)` de um certo

vetor real `x` na quinta linha, com a segunda e quarta linhas em branco, estes valores poderiam ser lidos usando-se o seguinte script em MATLAB:

```
fileID = fopen('input.dat','r');
A = fscanf(fileID,'%f',[1 inf]); % <--- this command reads
                                 %    the entire data file
fclose(fileID);
massa = A(1); t0 = A(2);
N = A(3);
position = 4;
x = A(position:position+6);
```

Procedendo-se como no script acima, com os COMANDOS MATLAB `fopen`, `fscanf` e `fclose`, os dados necessários (previamente obtidos na computação numérica) são passados ao MATLAB para uso das rotinas de visualização padrões desta linguagem. Para ilustração, mostramos a seguir o código MATLAB básico usado para gerar os vários gráficos exibidos nas Figs. 3.4 e 3.5 acima:

```
n0 = max( find( x <= -4 ) );
n1 = max( find( x <= +4 ) );

[X,Y] = meshgrid(x(n0:n1),y(n0:n1));
U = u(n0:n1,n0:n1);

a_view = 115;
z_view = 27.5;

figure(10)
surfl(X',Y',U)
colormap(jet)
shading interp
view(a_view,z_view)
xlabel('x')
ylabel('y')
```

(supondo os valores da solução computada no instante `t` em questão armazenados na variável `u`, com `x` e `y` contendo as abscissas e ordenadas dos pontos da malha).

Capítulo IV

Código Paralelo

Neste capítulo, vamos examinar vários aspectos de computação paralela, comentando sobre computadores paralelos e as noções fundamentais de *processos* e *threads*. Depois disso, passamos a descrever a interface de programação (*Application Programming Interface*, ou API) conhecida por OpenMP (*Open Multi-Processing*), muito usada atualmente para o desenvolvimento de programas paralelos nas linguagens Fortran ou C/C++, e que se baseia no uso de DIRETIVAS DE COMPILAÇÃO em pontos específicos do código em questão (que identificam as regiões paralelas e críticas presentes no código, e os possíveis pontos de sincronização, indicando como o paralelismo em cada região deverá ser explorado pelo programa) e certas VARIÁVEIS DE AMBIENTE ou CHAMADAS A SUBPROGRAMAS (funções ou sub-rotinas, em Fortran) especiais, particularmente indicada para a programação paralela em sistemas de MEMÓRIA COMPARTILHADA (como é o caso de processadores constituídos por múltiplos núcleos de processamento, ou *multi-core processors*, comumente presentes nos computadores e notebooks pessoais modernos, além de computadores maior porte e supercomputadores). Também apresentamos neste capítulo as principais modificações introduzidas no código serial descrito no Capítulo III para sua paralelização, indicando as várias diretivas OpenMP que foram acrescentadas ao código original (nas regiões paralelas apropriadas) e alguns detalhes relacionados.

4.1. Alguns fatos relacionados com programação paralela

Atualmente (2018), aqui no Brasil, já não existe mais à venda computador pessoal novo, *notebook* ou *tablet* com apenas um núcleo no seu processador. Todos eles possuem dois ou mais núcleos. Esta tendência é tão pronunciada que, atualmente também, qualquer *smartphone* de preço acima de R$ 400 tem dois ou mais núcleos (seja ele da LG, SAMSUNG, MOTOROLA, LENOVO, ASUS e outras marcas atuais). Aproveitando esta característica, os sistemas operacionais e programas que rodam nestas duas plataformas (computadores e celulares) realizam várias tarefas simultaneamente e utilizam largamente programação em paralelo.

Os alunos que entram nas faculdades, universidades e cursos técnicos ligados à área da computação, geralmente, hoje em dia, aprendem a programar em Java, C++ ou C# de modo sequencial (serial), isto é, faz-se uma tarefa de cada vez e somente quando esta termina vai-se para a próxima tarefa. Um exemplo concreto é o programa que foi descrito no capítulo anterior (Capítulo III). No entanto, quando o estudante de Ciência da Computação, Engenharia da Computação ou outros cursos similares chega no final de seu curso, há alguma cadeira como "Programação paralela e distribuída." Ou seja, a computação paralela já é algo tão comum que já faz parte obrigatória do currículo básico dos cursos de graduação de informática.

Professores, alunos, cientistas, engenheiros e outros pesquisadores de empresas públicas e privadas utilizam supercomputadores das suas empresas ou de algum cluster computacional (por exemplo, o Sistema Nacional de Processamento de Alto Desempenho (SINAPAD), descrito no capítulo anterior) nos seus trabalhos, projetos e pesquisas. Sobretudo quando o cálculo, a simulação ou pesquisa não conseguem ser feitos em tempo hábil em um PC (microcomputador) comum. Um supercomputador ou um *cluster* é fundamentalmente uma reunião de várias placas (às vezes, centenas, milhares, ou mesmo dezenas ou centenas de milhares de placas[1]) — digamos, m placas —, cada placa com um número n de processadores, e cada processador, por sua vez, com certo número p de núcleos, trabalhando de forma harmoniosa entre si. Se todas estas placas e processadores forem iguais, teremos um cluster homogêneo de $m \times n$ processadores e $m \times n \times p$ núcleos.

Muitos projetos e pesquisas utilizam a computação paralela em larga escala (*large scale scientific computing*), como, por exemplo:
∗ fenômenos climáticos ou geofísicos (previsão meteorológica, evolução do clima, movimento de placas tectônicas, prospecção geofísica, atividade sísmica, etc);
∗ fenômenos físicos (órbita de planetas, satélites ou sondas, dinâmica de fluidos e turbulência, mecânica estatística, eletromagnetismo, reações nucleares, etc);
∗ fenômenos químicos (reações e cinética química, fenômenos de combustão, etc);
∗ fenômenos biológicos (genoma humano, bioquímica médica, redes neurais, fisiologia celular, dinâmica do sangue, desenvolvimento de válvulas cardíacas, etc);
∗ componentes mecânicos (aerodinâmica/resistência de materiais em naves espaciais, desenho de turbinas, aerodinâmica de aviões em regimes supersônicos, etc);
∗ circuitos electrônicos (verificação de placas e circuitos digitais, etc);

[1]Para uma descrição abreviada dos supercomputadores de maior capacidade atualmente, pode-se consultar o sítio de endereço http://www.top500.org.

∗ no projeto e simulação de construção de navios, motores, satélites, aviões, foguetes, espaçonaves, bombas e reatores nucleares, estruturas para extração petrolífera, etc;
∗ controle e pesquisa de dados verídicos ou não pela Receita Federal;
∗ simulação e projeto de novos produtos químicos e medicamentos;
∗ e em muitas outras aplicações similares em ciência e tecnologia.

Também é possível, com um pouco de trabalho e conhecimento, conectar m microcomputadores com cabos de rede, cada um com p núcleos de processamento, de modo a construir um cluster, com poder de processamento similar ao de um mini-supercomputador, a um custo relativamente mais baixo. A principal desvantagem destes clusters domésticos tem a ver com a relativa lentidão da comunicação entre os computadores componentes, com impacto significativo sobre o desempenho do sistema.

4.2. Como usar estes vários núcleos de maneira eficiente?

Atualmente, existem vários protocolos, bibliotecas e ferramentas que possibilitam a comunicação e a interação entre núcleos, processadores, memórias compartilhadas e distribuídas, como, por exemplo: POSIX Threads, OpenMP e MPI, entre outros. Neste trabalho, enfocaremos o uso de OpenMP, que é um importante recurso de programação paralela baseada no modelo de multiprocessadores com memória compartilhada, e que (via a utilização de diretivas de compilação apropriadas) permite ao programador total controle sobre a paralelização do código. O modelo básico desta paralelização segue o chamado *fork-join model*, descrito a seguir. A execução de um programa OpenMP qualquer (seja implementado em Fortran ou em C/C++) começa como uma sequência *simples* de comandos, referido como *master thread*, que é executada serialmente até se alcançar a primeira região paralela identificada no programa (diretivas !$OMP PARALLEL ... !$OMP END PARALLEL em Fortran, e diretiva #pragma omp parallel em C/C++). Neste ponto, é criado um grupo de novas *threads*, conforme a quantidade solicitada (fixada pela variável de ambiente OMP_NUM_THREADS ou por chamada à sub-rotina OMP_SET_NUM_THREADS (em Fortran) ou omp_set_num_threads (em C/C++), conforme a linguagem usada no código, feita *antes* de se entrar na região paralela em questão), que se adicionam à *master thread* (identificada pelo número padrão 0) para a execução das seções a elas atribuídas pelo programador na região paralela. (Para esta divisão, que necessita da identificação de cada *thread*, faz-se uso da função OMP_GET_THREAD_NUM

(em Fortran), ou omp_get_thread_num (em C/C++), além de cópias das variáveis relevantes para cada *thread*, para seu uso privado.) Na região paralela assim identificada, cada thread recebe uma cópia do código, que é assim duplicado, com *cada* thread executando independentemente sua cópia individual do código em questão. Ao final da seção paralela iniciada deste modo (em Fortran, identificado pela diretiva !$OMP END PARALLEL; em C/C++, pelo fim do bloco de comandos (localizado entre os caracteres especiais { e } correspondentes) que segue a diretiva #pragma omp parallel), ocorre um ponto automático de *sincronização* (ou *barreira*) entre as várias threads que constituem o grupo, sincronizando o fim de execução de cada uma. A partir deste ponto, a execução do programa volta ao cargo da *master thread* somente, que é a única thread daquele grupo que continua a executar a partir deste ponto. Este processo será repetido caso existam mais regiões paralelas, funcionando sempre do mesmo modo. Também é possível definir *sub-regiões* paralelas dentro de uma dada região paralela (ou *regiões paralelas recursivas*); neste caso, a(s) thread(s) que executa(m) esta operação irão criar novas threads ela(s) mesma(s), desempenhando cada uma o papel de thread mestre com relação às novas threads criadas na sub-região correspondente. Pode-se ver a partir deste esquema geral a variedade das inúmeras possibilidades oferecidas pelo OpenMP para a introdução de paralelismo em um código, virtualmente ilimitadas. Além disso, em contraste com outros procedimentos (como, por exemplo, MPI ou bibliotecas similares de trocas de mensagens), a utilização de OpenMP é relativamente segura quanto à sempre presente possibilidade de introdução de erros de paralelismo no código, o que contribui para a popularização crescente de seu emprego pela comunidade científica.

Uma discussão mais detalhada (e ainda introdutória, mas suficiente para nossos propósitos) sobre OpenMP é dada na Seção § 4.3 a seguir. Por ora, observamos que caso o compilador utilizado para a geração do código executável correspondente não estiver habilitado a reconhecer as diretivas do OpenMP, então estas serão ignoradas (sendo consideradas como comentários do programa irrelevantes para a compilação). Como nem todo compilador de Fortran ou C/C++ disponível no mercado oferece suporte para OpenMP, é preciso averiguar este ponto. Exemplos de compiladores que reconhecem OpenMP são:

* os compiladores `gcc`/`g++` e `gfortran` da GNU;
* os compiladores `xlc` e `xlf` da IBM;
* os compiladores `icc`, `icpc` e `ifort` da Intel;
* o compilador Microsoft Visual C++ da Microsoft;
* os compiladores `pgcc`/`pgcpp` e `pgf95`/`pgfortran` do Portland Group, Inc. (PGI);
* os compiladores `suncc`/`sunCC` e `sunf77`/`sunf90`/`sunf95` da Sun Studio.

Assim, a compilação de um programa Fortran (de nome `prog.f90`, por exemplo) paralelizado com uso de OpenMP poderia ser feito com o uso dos seguintes comandos, seguindo o manual de cada um. No caso do compilador `xlf` (ambiente IBM/AIX), seria executado o comando

```
xlf -qsmp=omp prog.f90 -o prog
```

para a geração do arquivo executável correspondente, aqui chamado de `prog` (a opção `-o` determina o nome do arquivo executável). A opção `-qsmp=omp` especifica o uso da biblioteca OpenMP.

Para o caso do compilador `gfortran` da GNU, o comando de compilação correspondente seria

```
gfortran -fopenmp prog.f90 -o prog
```

(para a geração de um arquivo executável de nome `prog`, como no caso anterior). No caso de se usar o compilador `ifort` da Intel, executaríamos em vez disso

```
ifort -openmp prog.f90 -o prog
```

enquanto com o compilador Fortran da Sun Studio seria

```
sunf90 -xopenmp prog.f90 -o prog
```

e com o compilador Fortran da PGI (Portland Group, Inc.) seria

```
pgfortran -mp prog.f90 -o prog
```

ou simplesmente `pgfortran prog.f90 -o prog`

`Os comandos acima são exemplos no ambiente Unix\Linux.` No ambiente Windows o comando correspondente, utilizando, por exemplo, o compilador `ifort` da Intel seria (no Windows um executável termina com a extensão `.exe`)

```
ifort -openmp prog.f90 -o prog.exe
```

Nos comandos acima, a ordem dos vários elementos pode ser invertida livremente, sem qualquer efeito; assim, por exemplo, o comando de compilação com `gfortran` poderia também ser entrado nas formas `gfortran -o prog -fopenmp prog.f90`, ou `gfortran -o prog prog.f90 -fopenmp`, `gfortran prog.f90 -o prog -fopenmp`, etc. Similarmente, no caso de um programa em Linguagem C utilizando OpenMP (digamos, `prog.c`), os comandos de compilação seriam: `xlc -qsmp=omp prog.c -o`

`prog` (IBM/AIX), `gcc -fopenmp prog.c -o prog` (GNU/Linux), `icc -openmp prog.c -o prog` (Intel), `suncc -xopenmp prog.c -o prog` (Sun Studio), e `pgcc -mp prog.c -o prog` (PGI).

Para a execução do código paralelo executável gerado `prog.exe` em uma dada máquina paralela, segue-se as regras usuais da máquina escolhida. Supondo-se, por exemplo, que se escolha o cluster SGI Altix 1350/450 disponível no CENAPAD-SP (gerenciado pela Universidade Estadual de Campinas, em Campinas, SP), a execução do programa `prog` teria de ser feita obrigatoriamente em modo *batch* (i.e., não interativo), submetendo-se um arquivo *script* via o comando `qsub` do sistema PBS (Portable Batch System). Em modo de execução paralela (i.e., mais de um núcleo de processamento), o SGI Altix 1350/450 do CENAPAD-SP oferece três filas para processamento paralelo, assim configuradas:

Filas paralelas	**Num. max. jobs exec.**	**Num. max. cpus/jobs**	**Tempo max. (elapsed)**	**Num. max. jobs/user**
`paralela`	—	16	31 dias	—
`par48`	1	48 (fixo)	7 dias	1
`exp68`	1	68 (fixo)	24 horas	1

Para códigos paralelos, no script PBS utliza-se a diretiva `#PBS -l` (letra ℓ aqui) para especificar os recursos necessários para a execução do job. Os principais recursos são: número de nós (`nodes=`), número de processadores por nó (`ppn=`) e quantidade de memória por processador (`pmem=`). Na fila `paralela`, o número de nós solicitados tem de ser 1, obrigatoriamente. Este é o caso de códigos paralelos obtidos com OpenMP, ou com os compiladores `ifort` (Fortran) ou `icc` (C) da Intel com a inclusão da opção `-parallel` (neste caso, a paralelização normalmente é feita nos laços iterativos (*loops*) do programa, sem se ter garantia de melhora no desempenho). Também é necessário definir um valor para a variável de ambiente OMP_NUM_THREADS, que especifica o número de *threads* que serão utilizadas pelo programa em um ambiente SMP (*single memory program*). Na fila `paralela` podem também ser executados códigos obtidos com uso de MPI (*message passage interface*), que devem ser executados com o comando `mpirun`; este comando pode especificar o número de processos que serão inicializados para execução através da opção `-np`, como por exemplo

```
mpirun -np 10 prog
```

para o caso de 10 processos, digamos. Caso este número seja maior que o número n de núcleos de processamento especificado na diretiva `#PBS -l ppn=`n, o `mpirun` inciará mais de um processo por processador.

Um exemplo de arquivo script PBS típico para a execução do programa `prog` (previamente compilado) obtido com OpenMP é apresentado a seguir.

```
#PBS -q paralela
#PBS -N job_prog
#PBS -V
#PBS -m bea
#PBS -M robertohoo@gmail.com
# *************************
#PBS -l nodes=1:ppn=10
# *************************
#!/bin/sh
cd $PBS_O_WORKDIR
# *************************
export OMP_NUM_THREADS=10
# *************************
echo
echo "************************************"
echo
echo "Inicio do job_prog:  " `date`
echo "Hostname:  " `hostname`
echo
echo "----------------------------------------"
echo "TASK:"
echo "Executing ./prog ..."
echo "Date started:  " `date`
./prog
echo "Done!"
echo "Date finished:  " `date`
echo "----------------------------------------"
echo
echo "Final do job_prog:  " `date`
echo
echo "************************************"
```

Se este arquivo script for salvo como (por exemplo) `job_prog_01`, o job nele contido seria submetido executando (na máquina `athenas`) o comando PBS

```
qsub job_prog_01
```

que encaminharia o job correspondente para as filas de execução da `athenas` (neste caso, a fila `paralela`, como solicitado na diretiva `#PBS -q` do script acima).

4.3. Introdução ao OpenMP

OpenMP foi criado em 1997 e continua sendo aperfeiçoado por um grupo de empresas de hardware e software, sendo as principais citadas abaixo [3]:

Hardware: Compaq, (Digital), Hewlett Packard Company, Intel Corporation, International Business Machines (IBM), AMD, Kuck & Associates (KAI), Silicon Graphics, Sun Microsystems, U.S. Department of Energy ASCI program.

Software: Argone, Absoft Corporation, Edinburgh Portable Compilers, GENIAS Software GmBH, Myrias Computer Technologies, The Portland Group (PGI), ADINA R&D, ANSYS, Dash Associates, Fluent, ILOG CPLEX Division, Livermore Software Technology Corporation (LSTC), MECALOG SARL, Oxford Molecular Group PLC, The Numerical Algorithms Group Ltd (NAG).

O OpenMP permite o processamento paralelo com múltiplas threads, já está na versão 4.5 e pode ser utilizado nas linguagens Fortran, C e C++. Geralmente é utilizado com processadores com vários núcleos. Digamos que o processador tenha 4 nćleos e o programa em Fortran (ou C/C++) foi desenvolvido para utilizar 8 threads: neste caso, os núcleos executam simultaneamente (em paralelo) 4 grupos de duas threads, um grupo de 2 threads para cada núcleo. Enfatizando, estas duas threads de cada grupo são realizadas em sequência, mas os 4 grupos são feitos em paralelo. Lembrando que threads são processos leves (subtarefas). Por exemplo, um sistema operacional, quando carregado e ativo na memória do computador, é formado por 20 a 60 processos; cada processo é uma tarefa — digamos, um processo lê (capta) o que o usuário apertou no teclado, outro processo controla o mouse, outro controla a leitura do leitor de DVD, outro controla a atualização do próprio sistema operacional, outro processo controla o acesso à rede (a internet, por exemplo), e assim por diante. O processo que lê o teclado pode ser subdividido em 3 subprocessos (threads): a primeira thread lê o teclado digitado, a segunda thread coloca estes valores em uma memória local (buffer), a terceira lê do buffer e escreve na tela. De 1970 até a chegada dos processadores multinúcleos, os microcomputadores só tinham um núcleo, mas o processador já era suficientemente rápido para realizar cada um dos processos

do sistema operacional e dos programas ativos do usuário com a rapidez necessária de modo que para o usuário parecesse multitarefa. Na realidade, era uma pseudo-multitarefa. A multitarefa real apenas apareceu com o surgimento de processadores com vários núcleos e a programação em rede (programação distribuída) com vários computadores.

O OpenMP é utilizado, inicialmente, em programas que rodam em máquinas com um único processador com vários núcleos. Com MPI (que usa troca de mensagens), ou outras bibliotecas similares, os processadores trocam informações entre si por MENSAGENS; MPI é utilizado quando há várias máquinas interligadas em rede (por cabos ou por wi-fi). Também é possível uma programação *híbrida*, utilizando ao mesmo tempo OpenMP e MPI. A parte MPI do código se encarrega de fazer a comunicação entre os processadores distribuídos em várias máquinas de modo que o trabalho total seja dividido entres estes n processadores, digamos, enquanto o OpenMP fica encarregado de distribuir o trabalho entre os diversos núcleos de cada processador — digamos, p núcleos. Assim, o trabalho total fica dividido em $n \times p$ partes, que pode então ser finalizado (em princípio) em tempo menor.

Descreveremos a seguir as principais DIRETIVAS do OpenMP, através das quais o programador determina o modo exato como deseja desenvolver o paralelismo presente no código. Estas diretivas serão ilustradas no formato sintático em que aparecem em um programa FORTRAN, com $\langle \ldots \ \ldots \ \ldots \rangle$ denotando um bloco de comandos FORTRAN sob ação da diretiva indicada. Serão motradas apenas as opções mais frequentes ou comumente usadas; para uma descrição mais completa, e para a descrição da sintaxe correspondente em C/C++, ou exemplos em FORTRAN e C, ver e.g. [3, 5, 7, 23, 25].

(1) !$OMP PARALLEL
$\langle \ldots \ \ldots \ \ldots \rangle$
!$OMP END PARALLEL

Identifica uma parte do código (representada pelo bloco $\langle \ldots \ \ldots \ \ldots \rangle$ de comandos situados entre as duas diretivas) que será executada por várias threads. Esta é a construção fundamental do OpenMP. Entre os ATRIBUTOS que podem ser especificados, tem-se: PRIVATE(...) [para indicar as variáveis privadas de cada thread] e SHARED(...) [para especificar variáveis compartilhadas, ou globais, que são comuns para todas as threads executando esta seção].

As próximas diretivas descrevem opções dentro de uma região paralela *previamente*

identificada por um par !$OMP PARALLEL ... !$OMP END PARALLEL:

(2) !$OMP DO
⟨ bloco DO ... END DO ⟩
!$OMP END DO ou !$OMP END DO NOWAIT

Especifica que as iterações especificadas no comando DO ... END DO devem ser distribuídas entre as várias threads presentes na seção paralela em questão e executadas em paralelo por elas. Um atributo importante desta diretiva é o atributo SCHEDULE(...), que especifica o tipo de distribuição desejada: STATIC, DYNAMIC, GUIDED, AUTO ou RUNTIME. No caso de não ser explicitada a divisão (via o atributo SCHEDULE(...)), a diretiva !$OMP DO irá (a seu próprio modo, dependente de implementações) dividir as iterações do bloco DO tão igualmente quanto for possível entre as threads do grupo. Se o atributo NOWAIT for especificado, as threads não são sincronizadas ao final do DO, continuando automaticamente sua execução para a iteração seguinte do loop, sem aguardar as demais threads.

(3) !$OMP SECTIONS
!$OMP SECTION
⟨ ... ... ⟩
!$OMP SECTION
⟨ ... ... ⟩
!$OMP SECTION
⟨ ... ... ⟩
!$OMP END SECTIONS ou !$OMP END SECTIONS NOWAIT

Esta diretiva é usada para dividir a execução em seções separadas, não iterativas, sendo cada seção executada em modo sequencial por uma (e apenas uma) thread do grupo presente. (Havendo mais threads do que seções, as threads excedentes não executarão trabalho nesta parte do código.) No final da diretiva SECTIONS existe um ponto de sincronização implícita, a menos que se faça uso do atributo NOWAIT nesta posição.

(4) !$OMP SINGLE
⟨ ... ... ⟩
!$OMP END SINGLE ou !$OMP END SINGLE NOWAIT

Indica que o bloco de comandos ⟨ ... ... ⟩ seja executado por somente uma thread do grupo presente. As demais threads, que não executam este bloco, esperam pelo fim do processamento da thread que executa o bloco, a menos que o atributo NOWAIT seja usado na diretiva END SINGLE.

(5) !$OMP MASTER
⟨ ... ... ⟩
!$OMP END MASTER

Indica uma região que será executada apenas pela thread mestre do grupo presente. As demais threads ignoram esta seção e prosseguem imediatamente para a seção que segue esta região, sem sincronização implícita entre elas.

(6) !$OMP CRITICAL
⟨ ... ... ⟩
!$OMP END CRITICAL

Indica que a seção do código ⟨ ... ... ⟩ deve ser executada por todas as threads, mas *somente por uma thread de cada vez*. Se uma thread estiver executando uma região crítica, as outras threads irão parar sua execução quando alcançarem esta região, aguardando sua vez de executarem a mesma de acordo com a ordem de chegada.

(7) !$OMP BARRIER

Identifica um ponto de barreira, sincronizando todas as threads do grupo presente. Ou seja: quando uma thread executa esta diretiva, ela interrompe sua execução até que todas as demais threads do grupo tenham executado a diretiva. Após este momento, todas as threads voltam a processar o restante do código.

(8) !$OMP ATOMIC
⟨ *expressão aritmética* ⟩

Indica que o comando de atribuição aritmética imediatamente seguinte deve ser executado *atomicamente* (ou seja, pontualmente por cada thread, e *não* simultaneamente por todas elas). Corresponde assim a uma operação crítica.

(9) !$OMP WORKSHARE
⟨ ... ... ⟩
!$OMP END WORKSHARE

Permite a paralelização de operações com arranjos (*arrays*), ou com as construções WHERE e FORALL do FORTRAN. Não há atributo SCHEDULE para ser especificado: a divisão do trabalho para as threads do grupo é feita pelo compilador. No final da região (indicado pela diretiva END WORKSHARE), existe um ponto de sincronização: todas as threads devem terminar seu trabalho antes de poderem proseguir para a parte seguinte do código.

Nos casos mais simples (em que não haja uso significativo de atributos), as diretivas de compartilhamento de trabalho podem ser combinadas, se parecer conveniente, com a diretiva PARALLEL ... END PARALLEL:

(10) !$OMP PARALLEL DO
⟨ bloco DO ... END DO ⟩
!$OMP END PARALLEL DO

Representa (por abreviação e conveniência) uma combinação das diretivas !$OMP PARALLEL e !$OMP DO, reduzindo assim o número de diretivas.

(11) !$OMP PARALLEL SECTIONS
!$OMP SECTION
⟨ ... ... ⟩
!$OMP SECTION
⟨ ... ... ⟩
!$OMP SECTION
⟨ ... ... ⟩
!$OMP END PARALLEL SECTIONS

Representa (por abreviação e conveniência) uma combinação das diretivas !$OMP PARALLEL e !$OMP SECTIONS, para reduzir o número de diretivas.

(12) !$OMP PARALLEL WORKSHARE
⟨ ... ... ⟩
!$OMP END PARALLEL WORKSHARE

Representa (por abreviação e conveniência) uma combinação das diretivas !$OMP PARALLEL e !$OMP WORKSHARE, reduzindo o número de diretivas.

Para uma descrição detalhada dos ATRIBUTOS das diretivas, ver e.g. [3, 5, 7, 23, 25].

Outro recurso importante do OpenMP é a biblioteca de sub-rotinas oferecidas. As principais ou mais comumente usadas são descritas a seguir (novamente, na forma como aparecem na Linguagem FORTRAN; para C/C++ e exemplos ou mais detalhes, ver [3, 5, 7, 23, 25]):

(1) SUBROUTINE OMP_SET_NUM_THREADS(n):

Esta sub-rotina determina o número n (onde n denota uma variável ou constante inteira, ou ainda uma expressão aritmética inteira) de threads que serão usadas na(s) região(ões) paralela(s) seguinte(s), sendo executada pela thread mestre (na parte serial do código, *antes* da definição da(s) região(ões) paralela(s) correspondente(s)). As threads geradas serão identificadas pelos valores inteiros $0, 1, \ldots, n-1$, sendo 0 sempre associado à thread mestre.

(2) INTEGER FUNCTION OMP_GET_THREAD_NUM():

Função inteira (sem argumentos) que retorna a identificação (número inteiro) da thread que executa a chamada (*thread identification*, comumente denota pela variável `tid` ou outro nome similar).

(3) INTEGER FUNCTION OMP_GET_NUM_THREADS():

Função inteira (sem argumentos) que, executada em uma região paralela (por alguma das threads ativas) retorna o número total de threads que estão sendo utilizadas na região em questão.

(4) INTEGER FUNCTION OMP_GET_MAX_THREADS():

Função inteira (sem argumentos) que retorna o número máximo de threads que podem ser utilizadas (geralmente coorrespondente ao valor especificado pela sub-rotina OMP_SET_NUM_THREADS ou, na falta desta, pelo valor da variável de ambiente OMP_NUM_THREADS).

(5) INTEGER FUNCTION OMP_GET_NUM_PROCS():

Função inteira (sem argumentos) que retorna o número de processadores (ou núcleos de processamento) disponíveis para a execução do programa.

(6) LOGICAL FUNCTION OMP_IN_PARALLEL():

Função booleana (sem argumentos) que retorna se uma seção do código está sendo executada em paralelo (valor .TRUE.) ou não (valor .FALSE.).

(7) DOUBLE PRECISION FUNCTION OMP_GET_WTIME():

Função real de PRECISÃO DUPLA (sem argumentos) usada para obter o tempo de parede (*wall time*) decorrido (em segundos) desde a chamada anterior.

Para exemplos e uma discussão mais completa (com mais detalhes e também com uma lista mais completa de funções), ver e.g. [3, 5, 7, 23, 25]).

Finalmente, convém citar as VARIÁVEIS DE AMBIENTE mais comumente definidas pelos usuários para a execução de programas paralelizados com OpenMP. Estas variáveis definem parâmetros relevantes no ambiente operacional antes da execução do código paralelo.

(1) OMP_NUM_THREADS:

Variável de ambiente que define o número máximo de threads durante a execução do programa. Havendo conflito com a definição dada pela sub-rotina OMP_SET_NUM_THREADS acima, prevalece a definição desta última.

(2) OMP_SCHEDULE:

O valor (um *string*) desta variável de ambiente determina o tipo de distribuição a ser utilizado nas iterações de um comando DO paralelo (identificado pelas diretivas DO ou PARALLEL DO). Por exemplo: no KORN SHELL do sistema operacional UNIX ou Linux, poderíamos definir

```
export OMP_SCHEDULE="static"
```

ou

```
export OMP_SCHEDULE="guided, 5"
```

(e assim por diante).

(3) OMP_NESTED:

Habilita (se o valor for `TRUE`) ou não (se valer `FALSE`) o paralelismo recursivo.

4.4. Onde utilizar OpenMP no código?

As linguagens Fortran, C e C++ usam estas três estruturas de repetição (loop) clássicas (aliás, estruturas de repetição similares estão presentes em todas as linguagens de programação):

(*a*)
```
n = 1;
Repeat {
⟨ alguma tarefa a ser feita 200 vezes (digamos) ⟩
n = n + 1;
} until n < 200
```

(*b*)
```
n = 1;
While (n < 1000) {
⟨ alguma tarefa a ser feita 1000 vezes (digamos) ⟩
n = n + 1;
}
```

(*c*)
```
n = 1;
For i = 1 to 100 do {
⟨ tarefa a ser feita 100 vezes (digamos) ⟩
}
```

Então, se tivermos 5 núcleos, por exemplo, e um bloco iterativo `For`, `Repeat`, `While` ou similar que repita uma tarefa 100 vezes, é possível, muitas vezes, fazer este tra-

balho em paralelo 20 vezes em cada núcleo, obtendo-se o mesmo resultado (a menos de erros de arredondamento) quando comparado ao programa sequencial. Como visto na seção anterior, há vários comandos em OpenMP que gerenciam estas estruturas de repetição e o número de threads (i.e., sequencias de tarefas, de tamanho variável e independentes) Na prática, o OpenMP usa threads — neste exemplo, poderíamos definir 10 threads, tendo 2 threads para cada núcleo de processamento, cada thread ficando encarregada de fazer 10 vezes a tarefa. Assim, 5 threads atuariam simultaneamente, uma em cada núcleo, e, após finalizarem, as 5 threads restantes finalizariam o ciclo de 100 iterações, de acordo com o exemplo acima. Além destas 3 estruturas de repetição clássicas, observamos no código deste trabalho uma grande matriz quadrada onde é feita a parte principal da computação. A seguir, tem-se o código em Fortran onde é feita a atualização dos valores computados para a solução (no novo nível de tempo, denotada por v) nos pontos (x_i, y_j) da malha numérica (correspondente à parte principal da computação no programa):

```
V(M1:M2,N1:N2) = U(M1:M2,N1:N2) +    &
  CFL * ( F(M1+1:M2+1,N1:N2) * ( U(M1+1:M2+1,N1:N2) - U(M1:M2,N1:N2) ) - &
          F(M1:M2,N1:N2) * ( U(M1:M2,N1:N2) - U(M1-1:M2-1,N1:N2) ) ) +    &
  CFL * ( G(M1:M2,N1+1:N2+1) * ( U(M1:M2,N1+1:N2+1) - U(M1:M2,N1:N2) ) - &
          G(M1:M2,N1:N2) * ( U(M1:M2,N1:N2) - U(M1:M2,N1-1:N2-1) ) )
```

Nos testes utilizando OpenMP no Cluster Grid5000 foram adotadas estas medidas: M1 = - 260, M2 = 260, N1 = - 260, N2 = 260 nos computadores situados na cidade de Lille, França. Temos, então, uma malha de 561 pontos por 561 pontos, e 560 faixas por 560 faixas de 1 unidade cada faixa. O cluster de Lille, que faz parte do Grid 5000, possui processadores Intel com 20 núcleos físicos, que, com a tecnologia Hyper-Threading da Intel, transformam-se em 40 núcleos lógicos. Podemos dividir este grande quadrado em 40 retângulos de 14 pontos por 561 pontos, isto é, com 13 faixas na horizontal e 560 faixas na vertical, e utilizar 40 threads, uma para cada núcleo lógico, com cada uma destas threads atuando em um destes retângulos. Cada thread processa o código principal, mostrado acima, no retângulo correspondente. No final, o efeito é o mesmo que o programa serial do Capítulo III, mas, conforme os dados experimentais, observou-se uma execução cerca de 7 a 10 vezes mais rápida. Os resultados experimentais (obtidos com uso de OpenMP) estão no Capítulo V: ver resultados 3, 4A e 4B.

4.5. Detalhando o código INITIALIZE

Tanto o programa serial como o paralelo consistem de dois programas básicos: INITIALIZE_SERIAL.F90 e U24H_SERIAL.F90 no caso serial (ou sequencial), e um par semelhante INITIALIZE_OPENMP.F90 e U24H_OPENMP.F90 no caso paralelo. A sequência dos programas INITIALIZE_SERIAL.F90 e INITIALIZE_OPENMP.F90 é praticamente idêntica, sendo descrita abaixo:

(a) **Test_Far_field_zones_INCONSISTENT**:

Testa se os três aneis quadrados, que constituem as regiões ou campos distantes 1, 2 e 3 (no programa, correspondentes às variáveis `refv1`, `refv2` e `refv3`), estão todos dentro do retângulo lógico [M1, M2] × [N1, N2]. Caso não estejam, é gravado um *log* em arquivo indicando este erro e o programa termina a execução. São estes campos que irão mostrar que os valores da matriz u (solução computacional) estão convergindo ou não para o valor esperado `b_esp = (b1+b2+b3+b4)/4`, supondo-se aqui o caso de 4 singularidades.

(b) **Index_of_singularity**:

Calcula os índices `i1`, `i2`, `i3`, `i4` e `j1`, `j2`, `j3` e `j4` das singularidades e os valores ali atribuídos `(x1,y1,b1)`, `(x2,y2,b2)`, `(x3,y3,b3)` e `(x4,y4,b4)`. Matematicamente, `i1 = M1 + (x1-M1*h)/h`, `j1 = N1 + (y1-N1*h)/h`, e `i2 = M1 + (x2-M1*h)/h`, `j2 = N1 + (y2-N1*h)/h`, e assim por diante para `i3`, `j3`, `i4` e `j4` (no caso de 4 singularidades, como suposto acima).

(c) **Time_to_reach_boundary_Barenblatt_Solution**:

É feita uma estimativa do tempo mínimo e tempo máximo da chegada das singularidades na fronteira do retângulo lógico [M1, M2] × [N1, N2], usando-se para isso soluções de Barenblatt centradas em cada singularidade.

(d) **Initial_Solution_u0**:

É criado o estado inicial `u0` (o valor inicial para a solução `u`). No programa o valor inicial é um plano de altura `b0`, onde `b0` é um valor próximo (mas propositadamente não igual) do valor esperado `b_esp`, acrescido de cones de raio `r0`, cujos centros das bases estão situados nos pontos `(x1, y1)`, `(x2, y2)`, `(x3, y3)`, `(x4, y4)`, e com vértices nos pontos `(x1,y1,b1)`,..., `(x4,y4,b4)`. Isto é, os picos dos cones ocupam as posições das singularidades. Aqui, `r0` é tomado como a metade da menor distância entre os centros das bases dos vários cones (cf. Fig. 3.1 do Capítulo III).

(e) **far_field_zones_of_u0**:

Calcula os valores dos termos `ff1_value_u0`, `ff2_value_u0` e `ff3_value_u0`. A média de `u0` no primeiro anel quadrado de largura 11 e distante `refv1` = 60 % de `L`, onde `L` é o menor lado do retângulo [M1,M2] × [N1,N2], é `ff1_value_u0`. Da mesma forma, `ff2_value_u0` é a média de `u0` no segundo anel de largura 11 e distante `refv2` = 70 % de `L`, e `ff3_value_u0` é a média de `u0` no terceiro anel quadrado, também de largura 11, e distante `refv3` = 80 % de `L`. Como ainda não foi feita nenhuma atualização de `u`, neste ponto do programa, tem-se `ff1_value_u0` = `ff2_value_u0` = `ff3_value_u0` = `b0`.

(f) **Defining_updating_some_variables**:

Inicializa algumas variáveis que serão utilizadas no programa.

```
min_u0 = minval(u0);
max_u0 = maxval(u0);
mass_u0 = sum(u0-b)*h**2;
u(M1:M2N1:N2) = u0.
```

Inicializa as bordas da malha (retângulo lógico) [M1-1,M2+1] × [N1-1,N2+1]:

```
u(M1-1,N1:N2) = u(M1,N1:N2);
u(M2+1,N1:N2) = u(M2,N1:N2);
u(M1-1:M2+1,N1-1) = u(M1-1:M2+1,N1);
u(M1-1:M2+1,N2+1) = u(M1-1:M2+1,N2);
q = (p - 2D0)/2D0; h_to_pm2 = h**(p-2);
time_next_dump = t0 + dt_dump;
dt = cfl*h*h;
t = 0.
```

(g) **Update_U**:

Aqui é feita a parte crucial do programa. É o centro do programa onde é feita a atualização de `u`. Como já descrito no capítulo anterior a equação

$$u_t \;=\; \mathrm{div}\,(\,|\,\nabla u\,|^{p-2}\,\nabla u\,) \;=\; \frac{\partial}{\partial x}\Big[\,(u_x^2+u_y^2)^{\frac{p-2}{2}}\,u_x\,\Big] \;+\; \frac{\partial}{\partial y}\Big[\,(u_x^2+u_y^2)^{\frac{p-2}{2}}\,u_y\,\Big]$$

é transformada por diferenças finitas em

$$\begin{aligned} v_{i,\,j} \;=\; u_{i,\,j} \;+\; \frac{\Delta t}{h^2}\Big[\,F(i+1,j)\,(\,u_{i+1,\,j}-u_{i,\,j}\,) \;-\; F(i,j)\,(\,u_{i,\,j}-u_{i-1,\,j}\,)\Big] \\ +\;\frac{\Delta t}{h^2}\Big[\,G(i,j+1)\,(\,u_{i,\,j+1}-u_{i,\,j}\,) \;-\; G(i,j)\,(\,u_{i,\,j}-u_{i,\,j-1}\,)\Big], \end{aligned}$$

onde

$$F(i,j) \;=\; \left[\,\Big(\frac{u_{i,\,j}-u_{i-1,\,j}}{h}\Big)^2+\Big(\frac{u_{i,\,j+1}+u_{i-1,\,j+1}-u_{i,\,j-1}-u_{i-1,\,j-1}}{4h}\Big)^2\,\right]^{\frac{p-2}{2}}$$

e

$$G(i,j) \;=\; \left[\,\Big(\frac{u_{i,\,j}-u_{i,\,j-1}}{h}\Big)^2+\Big(\frac{u_{i+1,\,j}+u_{i+1,\,j-1}-u_{i-1,\,j-1}-u_{i-1,\,j}}{4h}\Big)^2\,\right]^{\frac{p-2}{2}}.$$

(h) **Save_u24h_serial_INPUT_initialize**:

É a parte final do programa INITIALIZE_SERIAL.F90 e também do programa INITIALIZE_OPENMP.F90. Todos os dados criados e gerados são salvos no arquivo U24H_SERIAL_INPUT.DAT ou U24H_OMP_INPUT.DAT, respectivamente. Estes dados serão utilizados a seguir pelo programa U24H_SERIAL.F90 ou U24H_OPENMP.F90, respectivamente.

4.6. Detalhando a parte principal do código u24h_openmp.f90

A sequência dos programas U24H e INITIALIZE são bem semelhantes, tanto no caso serial como no caso OpenMP, isto é, os 4 programas: INITIALIZE_SERIAL.F90, INITIALIZE_OpenMP.F90, U24H_SERIAL.F90 e U24H_OPENMP.F90 apresentam praticamente a mesma sequência descrita na seção anterior § 4.5 e no Capítulo III. O que diferencia o programa serial do paralelo é fundamentalmente a parte principal dos programas U24H_SERIAL.F90 e U24H_OOPENMP.F90. No caso serial, é atualizado `u` de modo sequencial, enquanto no caso paralelo `u` é atualizado de modo paralelo usando OpenMP. Visto que os 4 programas apresentam a sua estrutura bem semelhante, descreveremos abaixo apenas a parte principal do programa U24H_OPENMP.F90.

Para ilustrar as ideias vamos adotar $M1 = -1000$, $M2 = 1000$, $N1 = -1000$ e $N2 = 1000$, com $h = 0.05$, 4 singularidades, e 8 threads. Tem-se então um quadrado físico $[-50, 50] \times [-50, 50]$ de 2001×2001 pontos (correspondente à malha ou quadrado lógico $[-1000, 1000] \times [-1000, 1000]$), e um quadrado físico estendido (externo) $[-50.05, 50.05] \times [-50.05, 50.05]$ (correspondente à malha ou quadrado lógico $[-1001, 1001] \times [-1001, 1001]$) de 2003×2003 pontos computacionais.

O primeiro passo é dividir a malha em `Nthreads` (número de threads) seções retangulares (ou *chunks*) de mesmo tamanho, atribuindo uma seção a cada thread. Isto é feito na preparação que antecede a região paralela propriamente dita do código, onde são identificadas as coordenadas locais (no caso, valores do índice `ic` de sua posição horizontal) das singularidades em cada CHUNK, como mostrado a seguir.

```
!-------------------------------------------------------------------------
!-------------------------------------------------------------------------
!                  MAIN COMPUTATION SECTION
!-------------------------------------------------------------------------
!-------------------------------------------------------------------------


  CALL OMP_SET_NUM_THREADS(Nthreads)     ! (Nthreads was set by program: initialize.f90)

  main_chunk_size = (M2-M1)/Nthreads;
  extra_size = nint( 10.0D0/h );

  no_procs = OMP_GET_NUM_PROCS();
  max_no_threads = OMP_GET_MAX_THREADS();

  ALLOCATE( ISsingularity1_in_chunk(0:Nthreads-1) )
  ALLOCATE( ISsingularity2_in_chunk(0:Nthreads-1) )
  ALLOCATE( ISsingularity3_in_chunk(0:Nthreads-1) )
  ALLOCATE( ISsingularity4_in_chunk(0:Nthreads-1) )

  ALLOCATE( ic1(0:Nthreads-1) );
  ALLOCATE( ic2(0:Nthreads-1) );
  ALLOCATE( ic3(0:Nthreads-1) );
  ALLOCATE( ic4(0:Nthreads-1) );

  do i = 0,Nthreads-1
    ic1(i) = - extra_size - 1;      ! <--- ic1(i), ic2(i), ic3(i), ic4(i)
    ic2(i) = - extra_size - 1;      !   are all initialized with ILLEGAL values
    ic3(i) = - extra_size - 1;      !   for extra caution (if correction applies,
    ic4(i) = - extra_size - 1;      !   the correction is done right below:
  enddo

! chunk i = 0:
  i0 = M1;
  iK = i0 + main_chunk_size;
  ISsingularity1_in_chunk(0) = 0;
  if ( i0 < i1 .AND. i1 < iK+extra_size ) then
      ISsingularity1_in_chunk(0) = 1;        ! <--- correction
      ic1(0) = i1 - i0;                      ! <--- correction
  endif
  ISsingularity2_in_chunk(0) = 0;
  if ( i0 < i2 .AND. i2 < iK+extra_size ) then
      ISsingularity2_in_chunk(0) = 1;        ! <--- correction
      ic2(0) = i2 - i0;                      ! <--- correction
  endif
  ISsingularity3_in_chunk(0) = 0;
  if ( i0 < i3 .AND. i3 < iK+extra_size ) then
      ISsingularity3_in_chunk(0) = 1;        ! <--- correction
      ic3(0) = i3 - i0;                      ! <--- correction
  endif
  ISsingularity4_in_chunk(0) = 0;
  if ( i0 < i4 .AND. i4 < iK+extra_size ) then
      ISsingularity4_in_chunk(0) = 1;        ! <--- correction
      ic4(0) = i4 - i0;                      ! <--- correction
  endif
```

```
! internal chunks (i = 1,...,Nthreads-2):
  if (Nthreads > 2) then
     do i = 1,Nthreads-2
       i0 = M1 + i*main_chunk_size;
       iK = i0 + main_chunk_size;     ! <--- same as: iK = M1 + (i+1)*main_chunk_size
       ISsingularity1_in_chunk(i) = 0;
       if ( i0-extra_size < i1 .AND. i1 < iK+extra_size ) then
           ISsingularity1_in_chunk(i) = 1;        ! <--- correction
           ic1(i) = i1 - i0;                      ! <--- correction
       endif
       ISsingularity2_in_chunk(i) = 0;
       if ( i0-extra_size < i2 .AND. i2 < iK+extra_size ) then
           ISsingularity2_in_chunk(i) = 1;        ! <--- correction
           ic2(i) = i2 - i0;                      ! <--- correction
       endif
       ISsingularity3_in_chunk(i) = 0;
       if ( i0-extra_size < i3 .AND. i3 < iK+extra_size ) then
           ISsingularity3_in_chunk(i) = 1;        ! <--- correction
           ic3(i) = i3 - i0;                      ! <--- correction
       endif
       ISsingularity4_in_chunk(i) = 0;
       if ( i0-extra_size < i4 .AND. i4 < iK+extra_size ) then
           ISsingularity4_in_chunk(i) = 1;        ! <--- correction
           ic4(i) = i4 - i0;                      ! <--- correction
       endif
     enddo
  endif

! chunk i = Nthreads-1:
  if (Nthreads > 1) then
     i = Nthreads - 1;
     i0 = M1 + i*main_chunk_size;     ! <--- same as: i0 = M2 - main_chunk_size
     iK = M2;                         ! <--- same as: iK = i0 + main_chunk_size
     ISsingularity1_in_chunk(i) = 0;
     if ( i0-extra_size < i1 .AND. i1 < iK ) then
         ISsingularity1_in_chunk(i) = 1;        ! <--- correction
         ic1(i) = i1 - i0;                      ! <--- correction
     endif
     ISsingularity2_in_chunk(i) = 0;
     if ( i0-extra_size < i2 .AND. i2 < iK ) then
         ISsingularity2_in_chunk(i) = 1;        ! <--- correction
         ic2(i) = i2 - i0;                      ! <--- correction
     endif
     ISsingularity3_in_chunk(i) = 0;
     if ( i0-extra_size < i3 .AND. i3 < iK ) then
         ISsingularity3_in_chunk(i) = 1;        ! <--- correction
         ic3(i) = i3 - i0;                      ! <--- correction
     endif
     ISsingularity4_in_chunk(i) = 0;
     if ( i0-extra_size < i4 .AND. i4 < iK ) then
         ISsingularity4_in_chunk(i) = 1;        ! <--- correction
         ic4(i) = i4 - i0;                      ! <--- correction
     endif
  endif

  omp_iterations = extra_size;
  omp_time_advance = omp_iterations*dt;
  no_external_iterations = nint( dt_dump / omp_time_advance );
```

Cada CHUNK é acrescido de seções extras à esquerda e à direita, de largura horizontal `extra_size` (no caso, 200) pontos, com cada thread recebendo, no início de cada ciclo, não só os valores da última solução computada `u` nos pontos de seu CHUNK como também os valores de `u` nas seções extras imediatamente vizinhas. Isso permite que a thread possua computar sozinha (e independentemente) novos valores da solução `u` em seu CHUNK por um total de `extra_size` iterações no tempo, antes que precise trocar informações com as threads vizinhas e reiniciar o ciclo novamente.

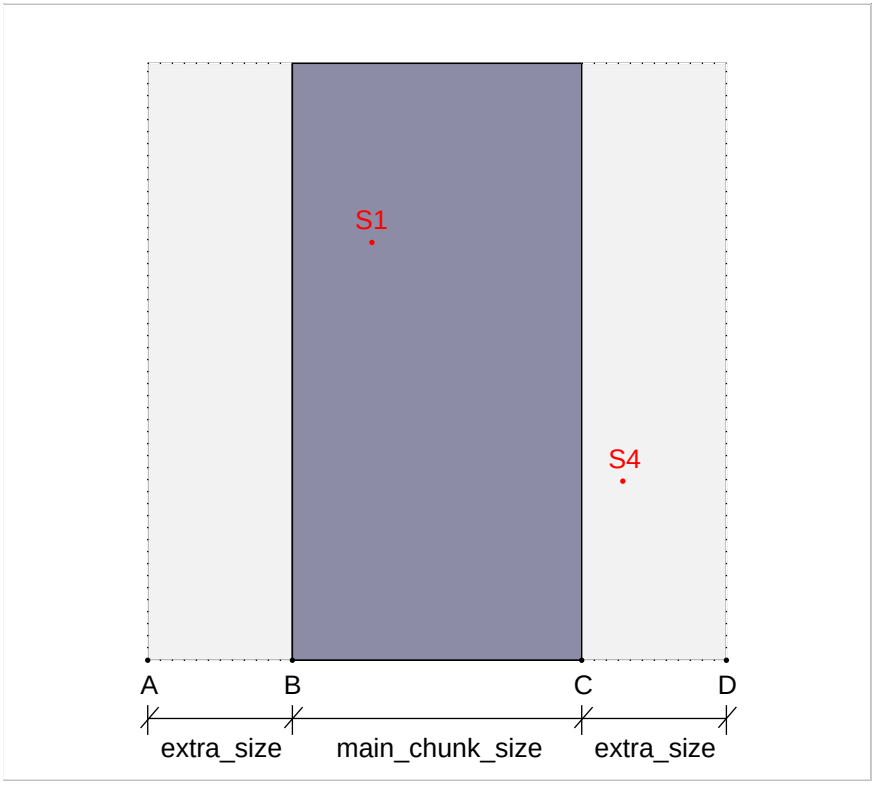


Fig. 4.1: Subdomínio da malha numérica atribuído à thread `tid` durante a execução paralela do código, contendo duas singularidades (`S1` e `S4`), descrito logicamente por um índice local `ic` (horizontal), $\texttt{ic0} \leq \texttt{ic} \leq \texttt{icF}$, e pelo índice vertical `j` da malha, $\texttt{N1} \leq \texttt{j} \leq \texttt{N2}$, onde $\texttt{ic0} = \texttt{-extra\_size}$, $\texttt{icF} = \texttt{main\_chunk\_size + extra\_size}$. Endereços locais de `S1` e `S4` são `(ic1(tid),j1)` e `(ic4(tid),j4)`, respectivamente, com os pontos `A`, `B`, `C` e `D` indicados na base descritos pelos índices locais (`ic0`, `N1`), (`0`, `N1`), (`main_chunk_size`, `N1`) e (`icF`, `N1`). respectivamente.

O código paralelo correspondente, baseado na decomposição de domínio mostrada na Fig. 4.1 acima, é apresentado integralmente a seguir.

```
   ALLOCATE( uc(-extra_size:main_chunk_size+extra_size, N1-1:N2+1) )
   ALLOCATE( vc(-extra_size:main_chunk_size+extra_size, N1-1:N2+1) )
   ALLOCATE( Fc(-extra_size:main_chunk_size+extra_size, N1:N2) )
   ALLOCATE( Gc(-extra_size:main_chunk_size+extra_size, N1:N2+1) )

   tF_check = tF - dt/4D0;

   DO WHILE ( t < tF_check )

     sum_cputime = 0D0;

     DO external_iteration_count = 1, no_external_iterations

!$OMP PARALLEL  DEFAULT(SHARED), &
!$OMP&          PRIVATE(tid,jc,i0,iK,ic0,icF,uc,vc,Fc,Gc)

       tid = OMP_GET_THREAD_NUM();

       i0 = M1 + tid*main_chunk_size;
       iK = i0 + main_chunk_size;

!$OMP MASTER
       CALL CPU_TIME(time_start)
!$OMP END MASTER

       if ( tid > 0 .AND. tid < Nthreads-1 ) then

          uc(-extra_size:main_chunk_size+extra_size, N1-1:N2+1)  &
                = u(i0-extra_size:iK+extra_size, N1-1:N2+1);

          ic0 = -extra_size + 1;                    ! <--- local indexing in chunk tid
          icF = main_chunk_size + extra_size - 1;   !       goes from  ic0-1  to  icF+1

          DO jc = 1, omp_iterations

          ! ********************************************************************** !
          !                     computation of Fc values on the local grid:        !
          ! ********************************************************************** !
          ! ******  Fc(i,j) for ic0 <= i <= icF + 1 and N1 <= j <= N2:

            Fc(ic0:icF+1,N1:N2) = ( (uc(ic0:icF+1,N1:N2) - uc(ic0-1:icF,N1:N2))**2 +      &
              ( (uc(ic0-1:icF,N1+1:N2+1)+uc(ic0:icF+1,N1+1:N2+1)) -                       &
                (uc(ic0-1:icF,N1-1:N2-1)+uc(ic0:icF+1,N1-1:N2-1)) )**2 /16 )**q / h_to_pm2;

          ! ********************************************************************** !
          !                     computation of Gc values on the local grid:        !
          ! ********************************************************************** !
          ! ******  Gc(i,j) for ic0 <= i <= icF and N1 <= j <= N2 + 1:

            Gc(ic0:icF,N1:N2+1) = ( (uc(ic0:icF,N1:N2+1) - uc(ic0:icF,N1-1:N2))**2 +      &
               ( (uc(ic0+1:icF+1,N1-1:N2)+uc(ic0+1:icF+1,N1:N2+1)) -                      &
                 (uc(ic0-1:icF-1,N1-1:N2)+uc(ic0-1:icF-1,N1:N2+1)) )**2 /16 )**q / h_to_pm2;
```

```
    ! ********************************************************************** !
    !              computation of vc = [ new uc values at the new time level ]:          !
    ! ********************************************************************** !
    ! ******  vc(i,j) for ic0 <= i <= icF and N1 <= j <= N2:

      vc(ic0:icF,N1:N2) = uc(ic0:icF,N1:N2) +                                  &
          cfl*( Fc(ic0+1:icF+1,N1:N2)*(uc(ic0+1:icF+1,N1:N2)-uc(ic0:icF,N1:N2)) -  &
                Fc(ic0:icF,N1:N2)*(uc(ic0:icF,N1:N2)-uc(ic0-1:icF-1,N1:N2)) ) +    &
          cfl*( Gc(ic0:icF,N1+1:N2+1)*(uc(ic0:icF,N1+1:N2+1)-uc(ic0:icF,N1:N2)) -  &
                Gc(ic0:icF,N1:N2)*(uc(ic0:icF,N1:N2)-uc(ic0:icF,N1-1:N2-1)) );

    ! ******  extending  vc  to the extra grid points:

      vc(ic0:icF,N1 -1) = vc(ic0:icF,N1);
      vc(ic0:icF,N2+1) = vc(ic0:icF,N2);

    ! ********************************************************************** !
    !                    computation of new uc values completed!                    !
    ! ********************************************************************** !

      uc = vc;       !  <--- updating uc (at the new time level)

    ! ******  correcting uc values at the singular points
    !           (in case they are located in its chunk region):

      if (ISsingularity1_in_chunk(tid).EQ.1) then
          uc(ic1(tid),j1) = b1;
      endif
      if (ISsingularity2_in_chunk(tid).EQ.1) then
          uc(ic2(tid),j2) = b2;
      endif
      if (ISsingularity3_in_chunk(tid).EQ.1) then
          uc(ic3(tid),j3) = b3;
      endif
      if (ISsingularity4_in_chunk(tid).EQ.1) then
          uc(ic4(tid),j4) = b4;
      endif

    ENDDO

endif

if ( tid.EQ.0 ) then

    uc(-1:main_chunk_size+extra_size, N1-1:N2+1)    &
            = u(i0-1:iK+extra_size, N1-1:N2+1);

    ic0 = 0;                                    ! <--- local indexing in chunk 0
    icF = main_chunk_size + extra_size - 1;     !       goes from  ic0-1  to  icF+1

    DO jc = 1, omp_iterations

    ! ********************************************************************** !
    !                        computation of Fc values on the local grid:                !
    ! ********************************************************************** !
    ! ******  Fc(i,j) for ic0 <= i <= icF + 1 and N1 <= j <= N2:
```

```
      Fc(ic0:icF+1,N1:N2) = ( (uc(ic0:icF+1,N1:N2) - uc(ic0-1:icF,N1:N2))**2 +       &
        ( (uc(ic0-1:icF,N1+1:N2+1)+uc(ic0:icF+1,N1+1:N2+1)) -                        &
          (uc(ic0-1:icF,N1-1:N2-1)+uc(ic0:icF+1,N1-1:N2-1)) )**2 /16 )**q / h_to_pm2;

    ! ********************************************************************** !
    !                      computation of Gc values on the local grid:       !
    ! ********************************************************************** !
    ! ******  Gc(i,j) for ic0 <= i <= icF and N1 <= j <= N2 + 1:

      Gc(ic0:icF,N1:N2+1) = ( (uc(ic0:icF,N1:N2+1) - uc(ic0:icF,N1-1:N2))**2 +       &
        ( (uc(ic0+1:icF+1,N1-1:N2)+uc(ic0+1:icF+1,N1:N2+1)) -                        &
          (uc(ic0-1:icF-1,N1-1:N2)+uc(ic0-1:icF-1,N1:N2+1)) )**2 /16 )**q / h_to_pm2;

    ! ********************************************************************** !
    !              computation of vc = [ new uc values at the new time level ]:  !
    ! ********************************************************************** !
    ! ******  vc(i,j) for ic0 <= i <= icF and N1 <= j <= N2:

      vc(ic0:icF,N1:N2) = uc(ic0:icF,N1:N2) +                                     &
        cfl*( Fc(ic0+1:icF+1,N1:N2)*(uc(ic0+1:icF+1,N1:N2)-uc(ic0:icF,N1:N2)) -   &
              Fc(ic0:icF,N1:N2)*(uc(ic0:icF,N1:N2)-uc(ic0-1:icF-1,N1:N2)) ) +     &
        cfl*( Gc(ic0:icF,N1+1:N2+1)*(uc(ic0:icF,N1+1:N2+1)-uc(ic0:icF,N1:N2)) -   &
              Gc(ic0:icF,N1:N2)*(uc(ic0:icF,N1:N2)-uc(ic0:icF,N1-1:N2-1)) );

    ! ******  extending  vc  to the extra grid points:

      vc(ic0-1, N1:N2) = vc(ic0, N1:N2);
      vc(ic0-1:icF,N1-1) = vc(ic0-1:icF,N1);
      vc(ic0-1:icF,N2+1) = vc(ic0-1:icF,N2);

    ! ********************************************************************** !
    !                      computation of new uc values completed!           !
    ! ********************************************************************** !

      uc(ic0-1:icF,N1-1:N2+1) = vc(ic0-1:icF,N1-1:N2+1);     ! <--- updating uc
                                                             ! (at the new time level)
    ! ******  correcting uc values at the singular points
    !         (in case they are located in its chunk region):

      if (ISsingularity1_in_chunk(0).EQ.1) then
          uc(ic1(0),j1) = b1;
      endif
      if (ISsingularity2_in_chunk(0).EQ.1) then
          uc(ic2(0),j2) = b2;
      endif
      if (ISsingularity3_in_chunk(0).EQ.1) then
          uc(ic3(0),j3) = b3;
      endif
      if (ISsingularity4_in_chunk(0).EQ.1) then
          uc(ic4(0),j4) = b4;
      endif

    ENDDO

endif
```

```
if ( tid.EQ.Nthreads-1 ) then

    uc(-extra_size:main_chunk_size+1, N1-1:N2+1)    &
            = u(i0-extra_size:iK+1, N1-1:N2+1);

    ic0 = -extra_size + 1;          ! <--- local indexing in chunk Nthreads-1
    icF = main_chunk_size;          !       goes from  ic0-1  to  icF+1

    DO jc = 1, omp_iterations

    ! ********************************************************************* !
    !                       computation of Fc values on the local grid:     !
    ! ********************************************************************* !
    ! ******  Fc(i,j) for ic0 <= i <= icF + 1 and N1 <= j <= N2:

      Fc(ic0:icF+1,N1:N2) = ( (uc(ic0:icF+1,N1:N2) - uc(ic0-1:icF,N1:N2))**2 +      &
        ( (uc(ic0-1:icF,N1+1:N2+1)+uc(ic0:icF+1,N1+1:N2+1)) -                       &
          (uc(ic0-1:icF,N1-1:N2-1)+uc(ic0:icF+1,N1-1:N2-1)) )**2 /16 )**q / h_to_pm2;

    ! ********************************************************************* !
    !                       computation of Gc values on the local grid:     !
    ! ********************************************************************* !
    ! ******  Gc(i,j) for ic0 <= i <= icF and N1 <= j <= N2 + 1:

      Gc(ic0:icF,N1:N2+1) = ( (uc(ic0:icF,N1:N2+1) - uc(ic0:icF,N1-1:N2))**2 +      &
        ( (uc(ic0+1:icF+1,N1-1:N2)+uc(ic0+1:icF+1,N1:N2+1)) -                       &
          (uc(ic0-1:icF-1,N1-1:N2)+uc(ic0-1:icF-1,N1:N2+1)) )**2 /16 )**q / h_to_pm2;

    ! ********************************************************************* !
    !             computation of vc = [ new uc values at the new time level ]:  !
    ! ********************************************************************* !
    ! ******  vc(i,j) for ic0 <= i <= icF and N1 <= j <= N2:

      vc(ic0:icF,N1:N2) = uc(ic0:icF,N1:N2) +                                       &
        cfl*( Fc(ic0+1:icF+1,N1:N2)*(uc(ic0+1:icF+1,N1:N2)-uc(ic0:icF,N1:N2)) -     &
              Fc(ic0:icF,N1:N2)*(uc(ic0:icF,N1:N2)-uc(ic0-1:icF-1,N1:N2)) ) +       &
        cfl*( Gc(ic0:icF,N1+1:N2+1)*(uc(ic0:icF,N1+1:N2+1)-uc(ic0:icF,N1:N2)) -     &
              Gc(ic0:icF,N1:N2)*(uc(ic0:icF,N1:N2)-uc(ic0:icF,N1-1:N2-1)) );

    ! ******  extending  vc  to the extra grid points:

      vc(icF+1, N1:N2) = vc(icF, N1:N2);
      vc(ic0:icF+1,N1-1) = vc(ic0:icF+1,N1);
      vc(ic0:icF+1,N2+1) = vc(ic0:icF+1,N2);

    ! ********************************************************************* !
    !                       computation of new uc values completed!         !
    ! ********************************************************************* !

      uc(ic0:icF+1,N1-1:N2+1) = vc(ic0:icF+1,N1-1:N2+1);    !  <--- updating uc
                                                            !  (at the new time level)


    ! ******  correcting uc values at the singular points
    !         (in case they are located in its chunk region):
```

```
            !  tid = Nthreads - 1:
              if (ISsingularity1_in_chunk(tid).EQ.1) then
                  uc(ic1(tid),j1) = b1;
              endif
              if (ISsingularity2_in_chunk(tid).EQ.1) then
                  uc(ic2(tid),j2) = b2;
              endif
              if (ISsingularity3_in_chunk(tid).EQ.1) then
                  uc(ic3(tid),j3) = b3;
              endif
              if (ISsingularity4_in_chunk(tid).EQ.1) then
                  uc(ic4(tid),j4) = b4;
              endif

            ENDDO

        endif

      ! Assembling new solution u (at new time level: t = t + omp_time_advance)
      ! from computed solutions uc on each chunk  (i = 0,1,...,Nthreads-1):

!$OMP CRITICAL

        u(i0:iK-1,N1-1:N2+1) = uc(0:main_chunk_size-1,N1-1:N2+1);

        if (tid .EQ. 0) then
            u(M1-1,N1-1:N2+1) = uc(0,N1-1:N2+1);
        endif

        if (tid .EQ. Nthreads-1) then
          u(M2,N1-1:N2+1) = uc(icF,N1-1:N2+1);
          u(M2-1,N1-1:N2+1) = uc(icF,N1-1:N2+1);
        endif

!$OMP END CRITICAL

!$OMP MASTER
        CALL CPU_TIME(time_finish)
        sum_cputime = sum_cputime + (time_finish-time_start);
!$OMP END MASTER

!$OMP END PARALLEL

       t = t + omp_time_advance;      ! <--- omp_time_advance = omp_iterations*dt

     ENDDO     !  external_iteration_count

  ENDDO

!-----------------------------------------------------------------------
!-----------------------------------------------------------------------
!                END OF MAIN COMPUTATION SECTION
!-----------------------------------------------------------------------
!-----------------------------------------------------------------------
```

Como pode ser visto, a estratégia de paralelização utilizada no código acima, correspondente à seção principal da computação realizada pelo programa, baseia o paralelismo em uma decomposição conveniente do domínio total em vários subdomínios, atribuídos às várias threads. Este procedimento é um exemplo particular dos chamados *domain decomposition methods* [9, 20, 24, 32], podendo naturalmente ser usado com outras bibliotecas de paralelização igualmente, como MPI ou bibliotecas similares.

Capítulo V

Resultados

Relembremos que estamos estudando 3 problemas:

(1) O **problema original, estático**: $\Delta_p u = 0$, (2.6a), (2.6b) e (2.7).
(2) O **p-Laplaciano evolutivo**: $u_t = \Delta_p u$, (3.1), (3.2) e (3.3) para resolver o problema original, estático.
(3) A **discretização de** (2) **usando diferenças finitas** mostrado em (g) **Update_U** na seção **4.5. Detalhando o código** INITIALIZE ou em (3.29) a (3.22).

Observe que a resolução de (2) e de (3) envolve, conforme mostrado no início do capítulo III uma sequência de funções $\{u_k\}_{k\in\mathbb{N}}$ com $u_k = u(x, k\Delta t)$, onde, u_k é a função u acrescida da variável temporal e Δt é um valor pequeno, arbitrário, mas fixado. E o problema (2) e (3) é útil, somente se, quando $k \to \infty$, temos:

$$\operatorname{div}\left(\,|\,\nabla u_k\,|^{p-2}\nabla u_k\,\right) = \frac{\partial u_k}{\partial t} \;\sim\; \frac{u_{k+1} - u_k}{\Delta t} = \frac{u(x, k\Delta t + \Delta t) - u(x, k\Delta t)}{\Delta t} \to 0 \tag{5.1}$$

Quando isto acontece u_k é uma boa aproximação de u do problema (1). Consequentemente os valores dos três campos distante de u_k é uma boa aproximação dos três campos distantes da solução u do problema (1), que por sua vez nos dá uma ideia do limite (2.7) que é o problema central desta dissertação.

Neste capítulo ilustraremos os resultados obtidos nas simulações numéricas com uma seleta de OUTPUTs representativos. Os experimentos realizados mostraram dois fatos:
(A) O uso do problema (2) e (3) para resolver o problema original (1) foi válido, pois de fato, para t e k suficientemente grande a função u do problema (2) e u_k do problema (3) convergiram para a função u do problema (1), como mostram as 4 medidas de convergência a seguir.

(B) Constatou-se o resultado observado em (1.4) do Capítulo I, isto é:

$$b_\infty := \lim_{|x|\to\infty} u(x) = \lim_{k\to\infty}\Big[\lim_{|x|\to\infty} u_k(x)\Big] \approx \frac{b_1 + b_2 + ... + b_m}{m} \tag{5.2}$$

(cf. (2.7), (2.10) e (2.11)). Onde $u(x)$ é a função do problema (1) e $u_k(x)$ é a função do problema (3) acima. Ou seja, os valores dos campos distantes `refv1`, `refv2` e sobretudo `refv3`, nos forneceram uma boa aproximação de b_∞ quando k é suficientemente grande, e esses valores estiveram próximos da média aritmética das singularidades nos dez resultados mostrados a seguir. Mas sempre devemos lembrar do **Teorema 2.2** e da **Observação 2.1**. Portanto o valor de b_∞ próximo da média em todos os resultados pode ter sido uma coincidência (ou não). O valor da média acima corresponde ao "Valor médio de b" mencionado nas seções § 5.1 a § 5.10 a seguir.
Nas tabelas seguintes, são mostrados diversos valores obtidos nas simulações de problemas evolutivos (3.1), (3.2) e (3.3) associados a (2.6a) e (2.6b), como descrito no Cap. II e Cap. III.

A coluna `Run` mostra quantas vezes o programa, de um mesmo problema, foi executado. Por exemplo, no resultado 1 e 2 o programa "u24h_omp.exe" foi executado 19 vezes. Já no resultado 3 o programa "u24h_serial.exe" foi executado 102 vezes e assim por diante. A coluna `tF` indica o tempo final. Cada unidade de `tF` equivale a 10000 (dez mil) atualicações de **u**. E `WT` informa o tempo total decorrido (*wall time*) em cada execução. Analisando as colunas `Run` e `tF` dos resultados 1 e 2 observamos que o tempo final cresce em passos cada vez maiores até que a partir da sétima execução o `tF` cresce de 100 em 100. A partir do resultado 3 até o resultado 10, visto que o tempo final é muito grande(o programa ficou rodando durante várias semanas), foram colocados nas colunas `Run` e `tF` valores espaçados, por exemplo o `tF` crescia de 300 a 300 ou de 1000 a 1000 dependendo do caso.

As colunas `refv1`, `refv2` e `refv3` mostram médias aritméticas da solução computada, nos diferentes instantes de tempo `tF`, em três regiões anelares distintas e sucessivamente mais próximas da fronteira da malha numérica utilizada. Os valores MÍNIMO e MÁXIMO da solução computada em toda a malha (em cada instante `tF`) são indicados por `min_u` e `max_u`, respectivamente; estes valores mostraram, em todas as simulações e nos 10 resultados abaixo, total conformidade com (2.8), Cap. II.

As 4 medidas de convergência $\Delta\,\mathtt{massa}$, $\|\,\Delta\,\mathtt{u}\,\|_{\mathtt{max}}$, `media_du/dt` e `max_du/dt` indicam as variações

$$\Delta\,\mathtt{massa} \;=\; \sum_{i\,=\,\mathtt{M1}}^{\mathtt{M2}}\;\sum_{j\,=\,\mathtt{N1}}^{\mathtt{N2}} (\mathtt{v}_{i,\,j} - \mathtt{u}_{i,\,j})\,h^2 \tag{5.3}$$

$$\| \Delta \mathtt{u} \|_{\max} = \max \left\{ | \mathtt{v}_{i,j} - \mathtt{u}_{i,j} | : \mathtt{M1} \leq i \leq \mathtt{M2},\ \mathtt{N1} \leq j \leq \mathtt{N2} \right\} \tag{5.4}$$

$$\mathtt{media\_du/dt} = \frac{\Delta \mathtt{massa}}{h^2(M2 - M1 + 1)(N2 - N1 + 1)\,\mathtt{dt\_dump}} \tag{5.5}$$

$$\mathtt{max\_du/dt} = \frac{\| \Delta \mathtt{u} \|_{\max}}{\mathtt{dt\_dump}} \tag{5.6}$$

onde `v` denota a solução computada no instante `tF`, `u` denota a solução computada no instante `tF - dt_dump`, e h denota o espaçamento da malha (com valores iniciais e finais dos índices i, j nos eixos x, y dados por `M1`, `M2`, `N1` e `N2`). Nas simulações foi usado $h = 0.1$ e `dt_dump` $= 0.01$.

Nas 10 simulações, mostrados a seguir, observou-se uma sequência, em valor absoluto, decrescente da Δ`volume`$(= \Delta\,$`massa`$)$ e de $\| \Delta \mathtt{u} \|_{\max}$, todos tendendo a **zero**. Isto mostra claramente que a sequência $\{u_k\}_{k \in \mathbb{N}}$, $u_k = u(x, k\Delta t)$ está convergindo, pois a diferença entre $u_{k+1} = v_{i,j}$ e $u_k = u_{i,j}$ tanto em (5.3) como em (5.4) estão diminuindo, basta olhar estas colunas nos 10 resultados a seguir, ou seja, as soluções u_k e u_{k+1} estão cada vez mais próximas umas das outras a medida que k cresce (sequência de Cauchy: sinal de convergência).

Por outro lado, observe que `media_du/dt` e `max_du/dt` é proporcional a $\Delta\,$`massa` e $\| \Delta \mathtt{u} \|_{\mathtt{max}}$, consequentemente (5.5) e (5.6) também estão converjindo para **zero** e portanto satisfazem (5.1). As 4 medidas tendem a $\mathbf{0}$ quando k tende ao ∞, mostrando que u_k está se estabilizando a medida que k cresce, comprovando a veracidade do fato (A).

Lendo novamente a explicação após (5.1), temos que quando calculamos os valores dos três campos distantes de u_k obtemos uma boa aproximação do lado esquerdo de (5.2). Por outro lado, em todas as simulações, estes resultados (far fiel value) também se aproximaram do lado direito de (5.2) o "Valor médio de b", comprovando, como dito no início deste capítulo, o fato (B), (5.2).

Para ilustrar melhor os resultados colocamos seis gráficos referentes ao Resultado 3 no final do mesmo. Os outros resultados possuem gráficos semelhantes e portanto os omitimos.

5.1. Resultado 1 (iniciado no GRID5000 na cidade de Lille): programa paralelo usando OMP, 40 núcleos lógicos e 40 threads e terminado no Okeanos com 4 núcleos lógicos e 10 threads

Parâmetros do problema: p = 2.1
S1: (1.0, 1.0), b1 = 3; S2: (-1.5, 0.5), b2 = 3
S3: (0.0, -1.5), b3 = 3; S4: (-2.0, -0.5), b4 = 2
Região computacional:
x_min = - 26, x_max = 26; y_min = - 26, y_max = 26
Amostragens de campo distante: refv1 = 10.4 refv2 = 15.6 refv3 = 20.8
Malha numérica: M1 = - 260, M2 = 260; N1 = - 260, N2 = 260
h = 0.1, dt = 0.0001, cfl = 0.01
Valor inicial para b: 2.85 Valor médio de b: 2.75
Valor mínimo de **u** em todas as iterações: `min_u` = 2
Valor máximo de **u** em todas as iterações: `max_u` = 3

Run	**tF**	**refv1**	**refv2**	**refv3**	**WT** (segundos)
0	0	2.85000	2.85000	2.85000	0.131200E+01
29	2000	2.77747	2.78216	2.78481	0.317982E+05
49	4000	2.75924	2.76074	2.76160	0.299935E+05
69	6000	2.75325	2.75369	2.75397	0.301953E+05
89	8000	2.75128	2.75138	2.75146	0.314508E+05
109	10000	2.75064	2.75062	2.75064	0.317152E+05
129	12000	2.75043	2.75038	2.75037	0.310309E+05
149	14000	2.75036	2.75029	2.75028	0.313128E+05
169	16000	2.75033	2.75027	2.75026	0.287856E+05
189	18000	2.75033	2.75026	2.75025	0.286662E+05
209	20000	2.75032	2.75026	2.75024	0.322599E+05
229	22000	2.75032	2.75026	2.75024	0.342336E+05
...	...	...	...	...	...
319	31000	2.75032	2.75026	2.75024	0.344540E+05
∞	∞	2.75032	2.75026	2.75024	

Run	**Δmassa**	$\| \Delta \mathrm{u} \|_{\max}$	**media_du/dt**	**max_du/dt**
0	-0.479131E-02	0.883523E-01	-0.176514E-03	0.883523E+01
29	-0.482776E-03	0.213298E-06	-0.177857E-04	0.213298E-04
49	-0.158817E-03	0.706021E-07	-0.585090E-05	0.706021E-05
69	-0.521488E-04	0.231654E-07	-0.192118E-05	0.231654E-05
89	-0.171069E-04	0.759003E-08	-0.630225E-06	0.759003E-06
109	-0.560987E-05	0.248792E-08	-0.206670E-06	0.248792E-06
129	-0.183946E-05	0.815659E-09	-0.677666E-07	0.815659E-07
149	-0.603123E-06	0.267431E-09	-0.222193E-07	0.267431E-07
169	-0.197743E-06	0.876637E-10	-0.728494E-08	0.876637E-08
189	-0.648269E-07	0.287326E-10	-0.238825E-08	0.287326E-08
209	-0.212522E-07	0.941469E-11	-0.782938E-09	0.941469E-09
229	-0.696046E-08	0.310862E-11	-0.256426E-09	0.310862E-09
...	...	...	...	...
319	0.000000E+00	0.000000E+00	0.000000E+00	0.000000E+00
∞	0.000000E+00	0.000000E+00	0.000000E+00	0.000000E+00

5.2. Resultado 2 (iniciado no GRID5000 na cidade de Lille): programa paralelo usando OMP, 40 núcleos lógicos e 40 threads e terminado no Okeanos com 4 núcleos lógicos e 10 threads

Parâmetros do problema: p = 2.1
S1: (1.0, 1.0), b1 = 3; S2: (-1.5, 0.5), b2 = 3
S3: (0.0, -1.5), b3 = 3; S4: (-2.0, -0.5), b4 = 2
Região computacional:
x_min = - 26, x_max = 26; y_min = - 26, y_max = 26
Amostragens de campo distante: refv1 = 10.4 refv2 = 15.6 refv3 = 20.8
Malha numérica: M1 = - 260, M2 = 260; N1 = - 260, N2 = 260
h = 0.1, dt = 0.0001, cfl = 0.01
Valor inicial para b: 2.65;
Valor médio de b: 2.75
Valor mínimo de **u** em todas as iterações: `min_u` = 2
Valor máximo de **u** em todas as iterações: `max_u` = 3

Run	**tF**	**refv1**	**refv2**	**refv3**	**WT** (segundos)
0	0	2.65000	2.65000	2.65000	0.131600E+01
29	2000	2.72299	2.71814	2.71545	0.330189E+05
49	4000	2.74138	2.73975	2.73886	0.295236E+05
69	6000	2.74739	2.74681	2.74651	0.312972E+05
89	8000	2.74936	2.74913	2.74902	0.283767E+05
109	10000	2.75001	2.74989	2.74984	0.326405E+05
129	12000	2.75022	2.75013	2.75011	0.291457E+05
149	14000	2.75029	2.75022	2.75020	0.296702E+05
169	16000	2.75031	2.75024	2.75023	0.265213E+05
189	18000	2.75032	2.75025	2.75024	0.304726E+05
209	20000	2.75032	2.75025	2.75024	0.295644E+05
229	22000	2.75032	2.75025	2.75024	0.314273E+05
...	...	...	...	...	...
319	31000	2.75032	2.75026	2.75024	0.316030E+05
324	31500	2.75032	2.75026	2.75024	0.265393E+05
∞	∞	2.75032	2.75026	2.75024	

Run	**Δmassa**	$\| \Delta \mathrm{u} \|_{\max}$	**media_du/dt**	**max_du/dt**
0	0.313129E-02	0.663598E-01	0.115358E-03	0.663598E+01
29	0.488653E-03	0.224718E-06	0.180022E-04	0.224718E-04
49	0.159585E-03	0.705309E-07	0.587918E-05	0.705309E-05
69	0.522474E-04	0.231229E-07	0.192482E-05	0.231229E-05
89	0.171224E-04	0.758689E-08	0.630796E-06	0.758689E-06
109	0.561307E-05	0.248828E-08	0.206788E-06	0.248828E-06
129	0.184032E-05	0.815926E-09	0.677982E-07	0.815926E-07
149	0.603465E-06	0.267519E-09	0.222319E-07	0.267519E-07
169	0.197849E-06	0.877076E-10	0.728884E-08	0.877076E-08
189	0.648774E-07	0.287770E-10	0.239011E-08	0.287770E-08
209	0.212847E-07	0.941469E-11	0.784136E-09	0.941469E-09
229	0.698395E-08	0.310907E-11	0.257292E-09	0.310907E-09
...	...	...	...	...
319	0.603013E-10	0.306422E-11	0.222152E-11	0.306422E-09
324	0.000000E+00	0.000000E+00	0.000000E+00	0.000000E+00
∞	0.000000E+00	0.000000E+00	0.000000E+00	0.000000E+00

5.3 Resultado 3 (gerado no CENAPAD-SP e OKEANOS)

Parâmetros do problema: p = 2.1
S1: (1.0, 0.5), b1 = 6; S2: (-1.5, 1.0), b2 = 2;
S3: (0.5,-1,5), b3 = 2; S4: (-1.5,-1.5), b4 = 2
Região computacional: x_min = - 15, x_max = 15; y_min = - 15, y_max = 15
Amostragens de campo distante: refv1 = 7.5, refv2 = 10, refv3 = 12.5
Malha numérica: M1 = - 150, M2 = 150; N1 = - 150, N2 = 150
h = 0.1, dt = 0.0001, cfl = 0.01
Valor inicial para b: 2;
Valor médio de b: 3
Valor mínimo de **u** em todas as iterações: `min_u` = 2
Valor máximo de **u** em todas as iterações: `max_u` = 6

OBS: Observe que os valores de refv1, refv2 e refv3 para uma precisão de 5 dígitos depois da vírgula(notação brasileira) ou ponto(notação americana) a partir da iteração 65 não mudaram mais. Em toda esta dissertação estamos usando a notação da América do Norte(EUA).

Run	**tF**	**refv1**	**refv2**	**refv3**	**WT** (segundos)
0	0	2.00000	2.00000	2.00000	0.115625E+01
8	300	2.62796	2.56971	2.53492	0.728490E+05
11	600	2.86632	2.83862	2.82192	0.719760E+05
15	1000	2.99868	2.98878	2.98273	0.714344E+05
25	2000	3.06728	3.06663	3.06619	0.713252E+05
35	3000	3.07257	3.07263	3.07262	0.100020E+05
45	4000	3.07298	3.07309	3.07311	0.997048E+04
55	5000	3.07301	3.07312	3.07315	0.101663E+05
65	6000	3.07301	3.07313	3.07315	0.983238E+04
75	7000	3.07301	3.07313	3.07315	0.986088E+04
85	8000	3.07301	3.07313	3.07315	0.100014E+05
87	8200	3.07301	3.07313	3.07315	0.100467E+05
88	8300	3.07301	3.07313	3.07315	0.101835E+05
89	8400	3.07301	3.07313	3.07315	0.972933E+04
102	9700	3.07301	3.07313	3.07315	0.998039E+04
∞	∞	3.07301	3.07313	3.07315	

Run	**Δmassa**	$\| \Delta \mathtt{u} \|_{\max}$	**media_du/dt**	**max_du/dt**
0	0.251875E-01	0.252946E+00	0.278004E-02	0.252946E+02
8	0.113083E-01	0.168211E-04	0.124814E-02	0.168211E-02
11	0.528586E-02	0.722756E-05	0,583421E-03	0.722756E-03
15	0.190935E-02	0.258568E-05	0.210742E-03	0.258568E-03
25	0.147455E-03	0.201462E-06	0.162752E-04	0.201462E-04
35	0.113441E-04	0.155173E-07	0.125209E-05	0.155173E-05
45	0.872489E-06	0.119357E-08	0.963001E-07	0.119357E-06
55	0.671032E-07	0.917932E-10	0.740645E-08	0.917932E-08
65	0.516143E-08	0.706102E-11	0.569687E-09	0.706102E-09
75	0.396286E-09	0.532907E-12	0.437396E-10	0.532907E-10
85	0.353585E-10	0.444089E-13	0.390266E-11	0.444089E-11
87	0.212756E-10	0.444089E-13	0.234827E-11	0.444089E-11
88	0.000000E+00	0.000000E+00	0.000000E+00	0.000000E+00
89	0.000000E+00	0.000000E+00	0.000000E+00	0.000000E+00
102	0.000000E+00	0.000000E+00	0.000000E+00	0.000000E+00
∞	0.000000E+00	0.000000E+00	0.000000E+00	0.000000E+00

OBS: Observe que a partir da iteração 88 a sequência de funções $\{\mathbf{u_k}\}_{\mathbf{k}\in\mathbb{N}}$, veja o problema (3) da página 74, convergiu para $\mathbf{u}_\infty = \mathbf{u}$ onde $\mathbf{u}_\infty$ é o limite de $\mathbf{u_k}$ quando $k \to \infty$ e este limite por sua vez é a função $\mathbf{u}$ do problema (1). Logo os valores dos campos distantes refv1, refv2 e refv3 de $\mathbf{u_{88}}$ nos dá uma boa aproximação do limite (2.7).

5.3.1 Gráficos do Resultado 3

Os gráficos são:
a) o tempo de cada ciclo,
b) o grafico de $\mathbf{u}$,
c) os valores dos campos distantes,
d) min e máx de $\mathbf{u}$,
e) o gráfico de $\Delta\,\mathtt{massa}$ e
f) o gráfico de $\| \Delta\,\mathtt{u} \|_{\mathtt{max}}$.
Visto que os valores de $\Delta\mathtt{massa}$ e $\| \Delta\,\mathtt{u} \|_{\mathtt{max}}$ são muito pequenos, para facilitar a visualização desses dois gráficos, utilizamos uma escala logaritmica de base 10.

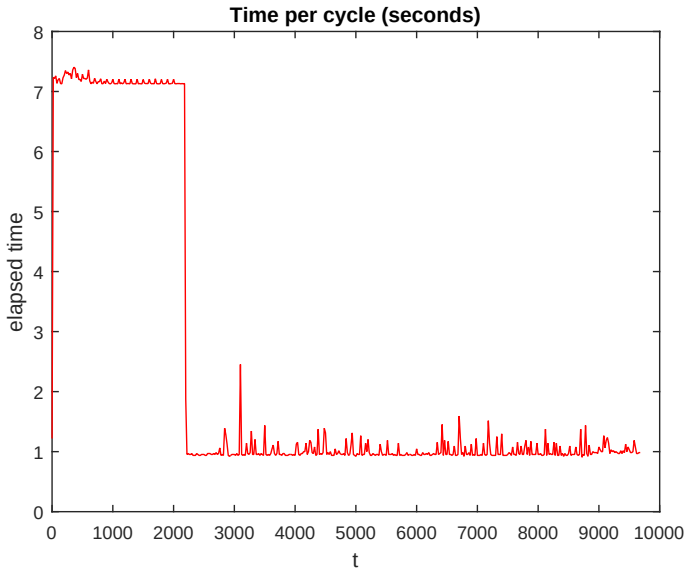


Fig. 5.3a: Cada 1 ciclo = 0.01t = 0.01**tF** = 100 atualizações de **u**, levou cerca de 7s no 1^{o} computador (t de 0 a 2200) e 1s no 2^{o} computador(t de 2300 a 9700) usando malha numérica de tamanho 301×301 e $p = 2.1$.

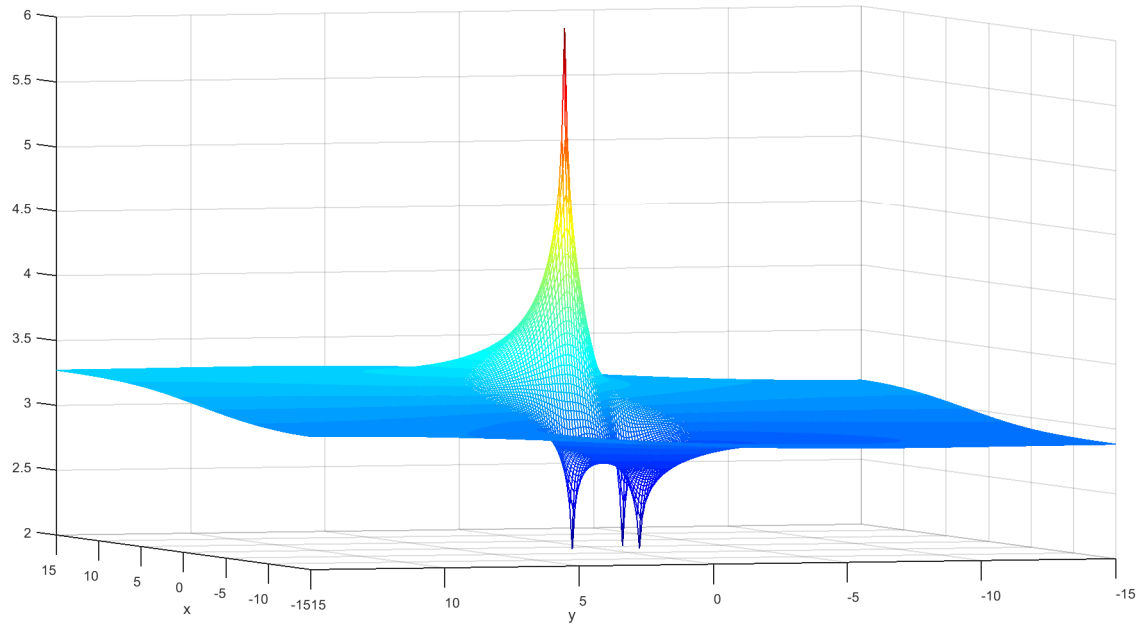


Fig. 5.3b: Gráfico de **u** para **tF** = 9700. Este $\mathbf{u} = \mathbf{u_k}$ do problema (3) é o mesmo **u** do problema inicial (1) pois para **tF** = 9700, `max_du/dt` = máximo $|\mathbf{du/dt}|$ em toda malha = 0

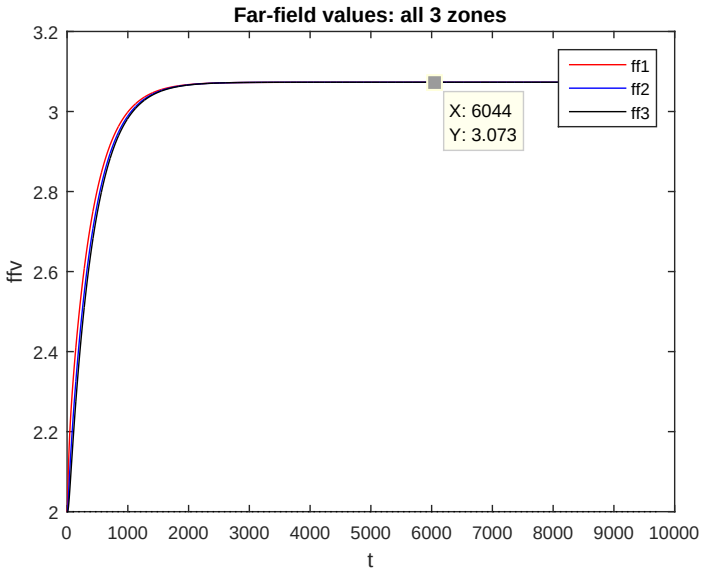


Fig. 5.3c: Os valores da média de **u** nas 3 regiões **refv1**, **refv2** e **refv3**. Observe que a partir de t = 3000 o valor de **u** se estabiliza nestas 3 regiões. (**u** praticamente não varia a partir de t = 3000).

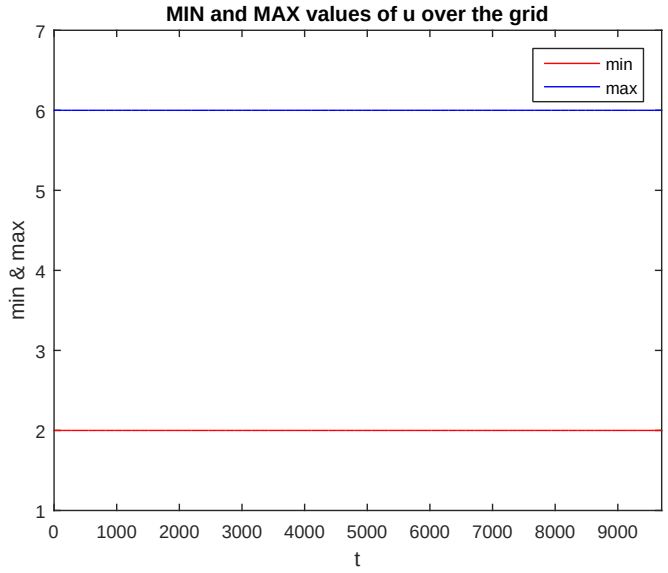


Fig. 5.3d: Os valores mínimo e máximo de **u** em todas as execuções do programa.

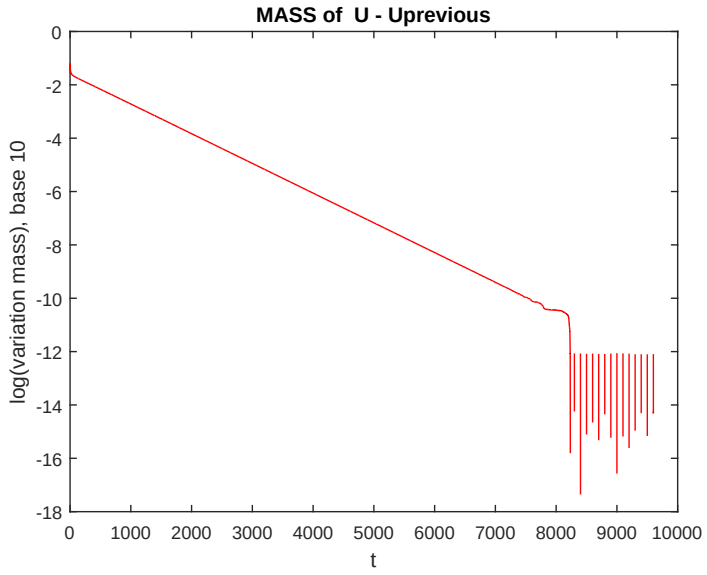


Fig. 5.3e: Visto que os valores de $\Delta\,\mathtt{massa}$ são da ordem de 10^{-1} a 10^{-10}, plotamos então o $\mathbf{log_{10}}\big(\Delta\,\mathtt{massa}\big)$ de $\mathbf{u_k}$. Observe que quando $\mathbf{Run} = 85$, $\mathbf{tF} = 8000$, $\Delta\,\mathtt{massa} = 0.353585E-10$ e $\mathbf{log_{10}}\big(0.353585E-10\big) = -10.45$, o que concorda com o gráfico acima. Para $\mathbf{tF} > 8200$ o logaritmo vai para -∞ pois $\Delta\,\mathtt{massa} = 0$.

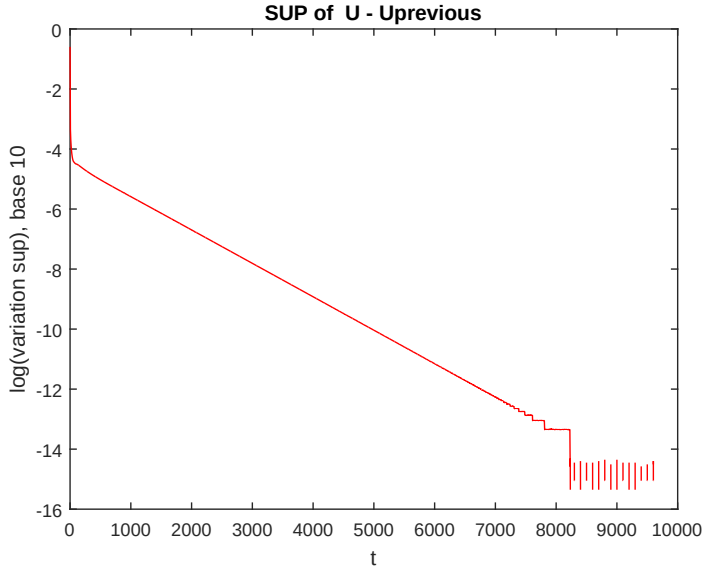


Fig. 5.3f: Os valores de $\|\,\Delta\,\mathtt{u}\,\|_{\mathtt{max}}$ são da ordem de 10^{-4} a 10^{-13} , plotamos então o $\mathbf{log_{10}}\big(\|\,\Delta\,\mathtt{u}\,\|_{\mathtt{max}}\big)$ de $\mathbf{u_k}$. Observe que quando $\mathbf{Run} = 85$, $\mathbf{tF} = 8000$, $\|\,\Delta\,\mathtt{u}\,\|_{\mathtt{max}} = 0.444089E-13$ e $\mathbf{log_{10}}\big(0.444089E-13\big) = -13.35$, o que concorda com o gráfico acima. Para $\mathbf{tF} > 8200$ o logaritmo vai para -∞ pois $\|\,\Delta\,\mathtt{u}\,\|_{\mathtt{max}} = 0$.

5.4. Resultado 4 (gerado no CENAPAD-SP)

Parâmetros do problema: p = 2.1
S1: (1.0, 0.5), b1 = 3 S2: (-1.5, 1.0), b2 = 3
S3: (0.5, -1,5), b3 = 3 S4: (-2.5, -2,5), b4 = 2
Região computacional: x_min = - 20, x_max = 20, y_min = - 20, y_max = 20
Amostragens de campo distante: refv1 = 10, refv2 = 14, refv3 = 18
Malha numérica: M1 = - 200, M2 = 200; N1 = - 200, N2 = 200
h = 0.1, dt = 0.0001, cfl = 0.01
Valor inicial para b: 2.75; Valor médio de b: 2.75
Valor mínimo de **u** em todas as iterações: `min_u` = 2
Valor máximo de **u** em todas as iterações: `max_u` = 3

Run	**tF**	**refv1**	**refv2**	**refv3**	**WT** (segundos)
0	0	2.75000	2.75000	2.75000	0.162500E+01
29	2000	2.70965	2.70995	2.71017	0.204928E+05
49	4000	2.70657	2.70643	2.70642	0.212732E+05
69	6000	2.70628	2.70610	2.70606	0.209445E+05
89	8000	2.70625	2.70607	2.70603	0.194047E+05
109	10000	2.70625	2.70606	2.70602	0.206286E+05
129	12000	2.70625	2.70606	2.70602	0.200447E+05
149	14000	2.70625	2.70606	2.70602	0.207022E+05
169	16000	2.70625	2.70606	2.70602	0.210496E+05
184	17500	2.70625	2.70606	2.70602	0.211377E+05
...	...	...	...	...	...
∞	∞	2.70625	2.70606	2.70602	

Run	**Δmassa**	$\| \Delta \mathtt{u} \|_{\max}$	**media_du/dt**	**max_du/dt**
0	-0.448500E-03	0.404215E-01	-0.278916E-04	0.404215E+01
29	-0.704167E-04	0.556416E-07	-0.437912E-05	0.556416E-05
49	-0.667638E-05	0.543703E-08	-0.415195E-06	0.543703E-06
69	-0.633161E-06	0.515676E-09	-0.393754E-07	0.515676E-07
89	-0.600433E-07	0.488942E-10	-0.373401E-08	0.488942E-08
109	-0.569433E-08	0.461853E-11	-0.354123E-09	0.461853E-09
129	-0.539552E-09	0.444089E-12	-0.335540E-10	0.444089E-10
149	-0.623002E-10	0.444089E-13	-0.387437E-11	0.444089E-11
169	0.000000E+00	0.000000E+00	0.000000E+00	0.000000E+00
184	0.000000E+00	0.000000E+00	0.000000E+00	0.000000E+00
...	...	...	...	...
∞	0.000000E+00	0.000000E+00	0.000000E+00	0.000000E+00

5.5. Resultado 5 (gerado no GRID5000 na cidade de Lille): programa paralelo usando OMP, 40 núcleos lógicos e 40 threads

Parâmetros do problema: p = 2.1
S1: (1.0, 0.5), b1 = -1; S2: (-1.5, 1.0), b2 = 1
S3: (0.5, -1,5), b3 = 5; S4: (-2.5, -2.5), b4 = 6
Região computacional:
x_min = - 20, x_max = 20; y_min = - 20, y_max = 20
Amostragens de campo distante:
refv1 = 10.0 refv2 = 14 refv3 = 18
Malha numérica:
M1 = - 200, M2 = 200; N1 = - 200, N2 = 200
h = 0.1, dt = 0.0001, cfl = 0.01
Valor inicial para b: 2.85;
Valor médio de b: 2.75
Valor mínimo de **u** em todas as iterações: `min_u` = -1
Valor máximo de **u** em todas as iterações: `max_u` = 6

Run	**tF**	**refv1**	**refv2**	**refv3**	**WT** (segundos)
0	0	2.85000	2.85000	2.85000	0.164062E+01
29	2000	2.82239	2.82337	2.82370	0.164534E+05
49	4000	2.82007	2.82072	2.82087	0.177960E+05
69	6000	2.81993	2.82057	2.82071	0.168202E+05
89	8000	2.81993	2.82056	2.82070	0.169104E+05
109	10000	2.81992	2.82056	2.82070	0.164722E+05
129	12000	2.81992	2.82056	2.82070	0.167287E+05
137	12800	2.81992	2.82056	2.82070	0.155914E+05
...	...	...	...	...	...
∞	∞	2.81992	2.82056	2.82070	

Run	**Δmassa**	$\| \Delta \mathtt{u} \|_{\max}$	**media_du/dt**	**max_du/dt**
0	-0.277857E-02	0.233406E+00	-0.172796E-03	0.233406E+02
29	-0.616158E-04	0.510781E-07	-0.383180E-05	0.510781E-05
49	-0.355653E-05	0.281820E-08	-0.221176E-06	0.281820E-06
69	-0.205043E-06	0.162448E-09	-0.127513E-07	0.162448E-07
89	-0.118219E-07	0.937028E-11	-0.735187E-09	0.937028E-09
109	-0.681153E-09	0.533351E-12	-0.423600E-10	0.533351E-10
129	-0.542537E-10	0.444089E-13	-0.337397E-11	0.444089E-11
137	0.000000E+00	0.000000E+00	0.000000E+00	0.000000E+00
...	...	...	...	...
∞	0.000000E+00	0.000000E+00	0.000000E+00	0.000000E+00

5.6. Resultado 6 (gerado no Okeanos-Grécia)

Parâmetros do problema: p = 2.1
S1: (1.0, 0.5), b1 = 5 S2: (-1.5, 1.0), b2 = 1 S3: (-0.5,-1,5), b3 = 4.5
Região computacional: x_min = - 10, x_max = 10, y_min = - 10, y_max = 10
Amostragens de campo distante: refv1 = 5, refv2 = 7, refv3 = 9
Malha numérica: M1 = - 100, M2 = 100; N1 = - 100, N2 = 100
h = 0.1, dt = 0.0001, cfl = 0.01
Valor inicial para b: 3.25;
Valor médio de b: 3.5
Valor mínimo de **u** em todas as iterações: `min_u` = 1
Valor máximo de **u** em todas as iterações: `max_u` = 5

OBS: Observe que os valores de refv1, refv2 e refv3 para uma precisão de 5 dígitos a partir da iteração 28 não mudaram mais.

Run	**tF**	**refv1**	**refv2**	**refv3**	**WT** (segundos)
0	0	3.25000	3.25000	3.25000	0.109375E+00
12	300	3.42336	3.41784	3.41513	0.241447E+04
15	600	3.44741	3.44581	3.44522	0.242844E+04
18	900	3.45118	3.45019	3.44994	0.242083E+04
21	1200	3.45177	3.45088	3.45068	0.241970E+04
24	1500	3.45186	3.45099	3.45080	0.241672E+04
27	1800	3.45188	3.45100	3.45081	0.517402E+04
28	1900	3.45188	3.45101	3.45082	0.524658E+04
41	3200	3.45188	3.45101	3.45082	0.517934E+04
42	3300	3.45188	3.45101	3.45082	0.545247E+04
43	3400	3.45188	3.45101	3.45082	0.531080E+04
...	...	...	...	...	...
206	19700	3.45188	3.45101	3.45082	0.560936E+04
∞	∞	3.45188	3.45101	3.45082	

Run	**Δmassa**	$\| \Delta \mathtt{u} \|_{\max}$	**media_du/dt**	**max_du/dt**
0	0.278093E-02	0.111072E+00	0.688331E-03	0.111072E+02
12	0.806585E-03	0.275633E-05	0.199645E-03	0.275633E-03
15	0.126544E-03	0.398006E-06	0.313219E-04	0.398006E-04
18	0.198375E-04	0.623297E-07	0.491014E-05	0.623297E-05
21	0.311032E-05	0.977358E-08	0.769863E-06	0.977358E-06
24	0.487684E-06	0.153248E-08	0.120711E-06	0.153248E-06
27	0.764666E-07	0.240297E-09	0.189269E-07	0.240297E-07
28	0.412336E-07	0.129585E-09	0.102061E-07	0.129585E-07
41	0.158357E-10	0.444089E-13	0.391963E-11	0.444089E-11
42	0.000000E+00	0.000000E+00	0.000000E+00	0.000000E+00
43	0.000000E+00	0.000000E+00	0.000000E+00	0.000000E+00
...	...	...	...	...
206	0.000000E+00	0.000000E+00	0.000000E+00	0.000000E+00
∞	0.000000E+00	0.000000E+00	0.000000E+00	0.000000E+00

OBS: Observe que a partir da iteração 42 a sequência de funções $\{\mathbf{u_k}\}_{\mathbf{k}\in\mathbb{N}}$ do problema 3) convergiu para $\mathbf{u}_\infty = \mathbf{u}$ onde $\mathbf{u}_\infty$ é o limite de $\mathbf{u_k}$ quando $k \to \infty$ e este limite por sua vez é a função $\mathbf{u}$ do problema 1). Logo os valores dos campos distantes refv1, refv2 e refv3 de $\mathbf{u_{42}}$ nos dá uma boa aproximação do limite (2.7).

5.7. Resultado 7 (gerado no Okeanos-Grécia)

Parâmetros do problema: p = 2.1
S1: (1.0, 0.5), b1 = 5 S2: (-1.5, -1.0), b2 = 1
Região computacional: x_min = - 15, x_max = 15, y_min = - 15, y_max = 15
Amostragens de campo distante: refv1 = 7.5, refv2 = 10.5, refv3 = 13.5
Malha numérica: M1 = - 150, M2 = 150; N1 = - 150, N2 = 150
h = 0.1, dt = 0.0001, cfl = 0.01
Valor inicial para b: 2.5;
Valor médio de b: 3.0
Valor mínimo de **u** em todas as iterações: `min_u` = 1
Valor máximo de **u** em todas as iterações: `max_u` = 5

OBS: Observe que os valores de refv1, refv2 e refv3 para uma precisão de 5 dígitos a partir da iteração 79 não mudaram mais.

Run	**tF**	**refv1**	**refv2**	**refv3**	**WT** (segundos)
0	0	2.50000	2.50000	2.50000	0.234375E+00
19	1000	2.93238	2.92463	2.92061	0.979325E+04
29	2000	2.99160	2.99033	2.98970	0.981827E+04
39	3000	3.00050	3.00020	3.00008	0.973944E+04
49	4000	3.00183	3.00168	3.00164	0.994220E+04
59	5000	3.00203	3.00190	3.00187	0.109453E+05
69	6000	3.00206	3.00193	3.00191	0.995527E+04
79	7000	3.00207	3.00194	3.00191	0.100541E+05
89	8000	3.00207	3.00194	3.00191	0.101294E+05
99	9000	3.00207	3.00194	3.00191	0.995553E+04
109	10000	3.00207	3.00194	3.00191	0.987847E+04
114	10500	3.00207	3.00194	3.00191	0.987192E+04
115	10600	3.00207	3.00194	3.00191	0.984498E+04
∞	∞	3.00207	3.00194	3.00191	

Run	**Δmassa**	$\| \Delta \mathtt{u} \|_{\max}$	**media_du/dt**	**max_du/dt**
0	0.419921E-02	0.101381E+00	0.463484E-03	0.101381E+02
19	0.129144E-02	0.166220E-05	0.142541E-03	0.166220E-03
29	0.193993E-03	0.249796E-06	0.214118E-04	0.249796E-04
39	0.291447E-04	0.375587E-07	0.321682E-05	0.375587E-05
49	0.311032E-05	0.977358E-08	0.769863E-06	0.977358E-06
59	0.657860E-06	0.847900E-09	0.726107E-07	0.847900E-07
69	0.988381E-07	0.127409E-09	0.109092E-07	0.127409E-07
79	0.148502E-07	0.191407E-10	0.163907E-08	0.191407E-08
89	0.223111E-08	0.288702E-11	0.246257E-09	0.288702E-09
99	0.334034E-09	0.444089E-12	0.368687E-10	0.444089E-10
109	0.396712E-10	0.444089E-13	0.437867E-11	0.444089E-11
114	0.258729E-10	0.444089E-13	0.285570E-11	0.444089E-11
115	0.000000E+00	0.000000E+00	0.000000E+00	0.000000E+00
∞	0.000000E+00	0.000000E+00	0.000000E+00	0.000000E+00

OBS: Observe que a partir da iteração 115 a sequência de funções $\{\mathbf{u_k}\}_{\mathbf{k}\in\mathbb{N}}$ do problema 3) convergiu para $\mathbf{u}_\infty = \mathbf{u}$ onde $\mathbf{u}_\infty$ é o limite de $\mathbf{u_k}$ quando $k \to \infty$ e este limite por sua vez é a função $\mathbf{u}$ do problema 1). Logo os valores dos campos distantes refv1, refv2 e refv3 de $\mathbf{u_{115}}$ nos dá uma boa aproximação do limite (2.7).

5.8. Resultado 8 (gerado no Okeanos-Grécia)

Parâmetros do problema: p = 2.1
S1: (2.0, -0.5), b1 = 5 S2: (-1.5, 1.0), b2 = -1 S3: (-0.5, -1.5), b2 = 2
Região computacional: x_min = - 20, x_max = 20, y_min = - 20, y_max = 20
Amostragens de campo distante: refv1 = 10, refv2 = 14, refv3 = 18
Malha numérica: M1 = - 200, M2 = 200; N1 = - 200, N2 = 200
h = 0.1, dt = 0.0001, cfl = 0.01
Valor inicial para b: 2.5;
Valor médio de b: 2.0
Valor mínimo de **u** em todas as iterações: `min_u` $= -1$
Valor máximo de **u** em todas as iterações: `max_u` $= 5$

OBS: Observe que os valores de refv1, refv2 e refv3 para uma precisão de 5 dígitos depois da iteração 139 não mudaram mais.

Run	**tF**	**refv1**	**refv2**	**refv3**	**WT** (segundos)
0	0	2.50000	2.50000	2.50000	0.150000E+01
19	1000	2.13618	2.15289	2.16153	0.187553E+05
29	2000	2.04022	2.04524	2.04783	0.184985E+05
39	3000	2.01148	2.01300	2.01378	0.181914E+05
49	4000	2.00287	2.00334	2.00358	0.201716E+05
59	5000	2.00029	2.00045	2.00052	0.239760E+05
69	6000	1.99952	1.99959	1.99961	0.188810E+05
79	7000	1.99929	1.99933	1.99933	0.177364E+05
89	8000	1.99922	1.99925	1.99925	0.181837E+05
99	9000	1.99920	1.99922	1.99923	0.182486E+05
109	10000	1.99919	1.99922	1.99922	0.181765E+05
119	11000	1.99919	1.99922	1.99922	0.182578E+05
129	12000	1.99919	1.99922	1.99922	0.182750E+05
139	13000	1.99919	1.99921	1.99922	0.201749E+05
149	14000	1.99919	1.99921	1.99922	0.195842E+05
159	15000	1.99919	1.99921	1.99922	0.200705E+05
169	16000	1.99919	1.99921	1.99922	0.199565E+05
209	20000	1.99919	1.99921	1.99922	0.255274E+05
∞	∞	1.99919	1.99921	1.99922	

Run	**Δmassa**	$\| \Delta \mathrm{u} \|_{\max}$	**media_du/dt**	**max_du/dt**
0	-0.656468E-02	0.158329E+00	-0.408249E-03	0.158329E+02
19	-0.286321E-02	0.226717E-05	-0.178059E-03	0.226717E-03
29	-0.857584E-03	0.627804E-06	-0.533320E-04	0.627804E-04
39	-0.256857E-03	0.187788E-06	-0.159736E-04	0.187788E-04
49	-0.769332E-04	0.562732E-07	-0.478438E-05	0.562732E-05
59	-0.230431E-04	0.168580E-07	-0.143302E-05	0.168580E-05
69	-0.690187E-05	0.504958E-08	-0.429218E-06	0.504958E-06
79	-0.206725E-05	0.151248E-08	-0.128560E-06	0.151248E-06
89	-0.619186E-06	0.453015E-09	-0.385063E-07	0.453015E-07
99	-0.185460E-06	0.135691E-09	-0.115335E-07	0.135691E-07
109	-0.555499E-07	0.406342E-10	-0.345458E-08	0.406342E-08
119	-0.166390E-07	0.121683E-10	-0.103475E-08	0.121683E-08
129	-0.498386E-08	0.364153E-11	-0.309939E-09	0.364153E-09
139	-0.149178E-08	0.108802E-11	-0.927719E-10	0.108802E-09
149	-0.446163E-09	0.333067E-12	-0.277463E-10	0.333067E-10
159	-0.131016E-09	0.888178E-13	-0.814771E-11	0.888178E-11
169	-0.431227E-10	0.444089E-13	-0.268174E-11	0.444089E-11
209	0.000000E+00	0.000000E+00	0.000000E+00	0.000000E+00
∞	0.000000E+00	0.000000E+00	0.000000E+00	0.000000E+00

OBS: Observe que a partir da iteração 209 a sequência de funções $\{\mathbf{u_k}\}_{\mathbf{k}\in\mathbb{N}}$ do problema 3) convergiu para $\mathbf{u}_\infty = \mathbf{u}$ onde $\mathbf{u}_\infty$ é o limite de $\mathbf{u_k}$ quando $k \to \infty$ e este limite por sua vez é a função $\mathbf{u}$ do problema 1). Logo os valores dos campos distantes refv1, refv2 e refv3 de $\mathbf{u_{209}}$ nos dá uma boa aproximação do limite (2.7).

5.9. Resultado 9 (gerado no Okeanos-Grécia)

Parâmetros do problema: p = 2.1
S1: (1.0, 0.5), b1 = 6 S2: (-1.5, 1.0), b2 = 5 S3: (0.5, -1.5), b2 = 1
Região computacional: x_min = - 20, x_max = 20, y_min = - 20, y_max = 20
Amostragens de campo distante: refv1 = 10, refv2 = 14, refv3 = 18
Malha numérica: M1 = - 200, M2 = 200; N1 = - 200, N2 = 200
h = 0.1, dt = 0.0001, cfl = 0.01
Valor inicial para b: 4.5;
Valor médio de b: 4.0
Valor mínimo de **u** em todas as iterações: `min_u` = 1
Valor máximo de **u** em todas as iterações: `max_u` = 6

OBS: Observe que os valores de refv1, refv2 e refv3 para uma precisão de 5 dígitos depois da da iteração 119 não mudaram mais.

Run	**tF**	**refv1**	**refv2**	**refv3**	**WT** (segundos)
0	0	4.50000	4.50000	4.50000	0.220312E+01
19	1000	4.09344	4.11357	4.12396	0.183430E+05
29	2000	3.97954	3.98586	3.98916	0.182087E+05
39	3000	3.94322	3.94513	3.94616	0.180305E+05
49	4000	3.93165	3.93216	3.93247	0.238672E+05
59	5000	3.92797	3.92804	3.92812	0.197889E+05
69	6000	3.92680	3.92672	3.92673	0.176795E+05
79	7000	3.92643	3.92631	3.92629	0.177432E+05
89	8000	3.92631	3.92617	3.92615	0.179916E+05
99	9000	3.92628	3.92613	3.92611	0.184507E+05
109	10000	3.92626	3.92612	3.92609	0.190414E+05
119	11000	3.92626	3.92611	3.92609	0.183723E+05
129	12000	3.92626	3.92611	3.92609	0.176795E+05
139	13000	3.92626	3.92611	3.92609	0.196788E+05
159	15000	3.92626	3.92611	3.92609	0.203353E+05
179	17000	3.92626	3.92611	3.92609	0.213597E+05
180	17100	3.92626	3.92611	3.92609	0.211710E+05
∞	∞	3.92626	3.92611	3.92609	

Run	**Δmassa**	$\| \Delta \mathrm{u} \|_{\max}$	**media_du/dt**	**max_du/dt**
0	-0.103408E-01	0.210758E+00	-0.643079E-03	0.210758E+02
19	-0.330817E-02	0.245719E-05	-0.205730E-03	0.245719E-03
29	-0.105645E-02	0.769848E-06	-0.656992E-04	0.769848E-04
39	-0.336493E-03	0.245052E-06	-0.209261E-04	0.245052E-04
49	-0.107077E-03	0.779162E-07	-0.665897E-05	0.779162E-05
59	-0.340632E-04	0.247797E-07	-0.211834E-05	0.247797E-05
69	-0.108351E-04	0.788143E-08	-0.673821E-06	0.788143E-06
79	-0.344642E-05	0.250680E-08	-0.214328E-06	0.250680E-06
89	-0.109622E-05	0.797323E-09	-0.681726E-07	0.797323E-07
99	-0.348682E-06	0.253664E-09	-0.216841E-07	0.253664E-07
109	-0.110908E-06	0.806466E-10	-0.689724E-08	0.806466E-08
119	-0.352783E-07	0.256684E-10	-0.219391E-08	0.256684E-08
129	-0.112218E-07	0.817124E-11	-0.697870E-09	0.817124E-09
139	-0.356800E-08	0.257661E-11	-0.221889E-09	0.257661E-09
159	-0.363900E-09	0.266898E-12	-0.226304E-10	0.266898E-10
179	-0.895741E-11	0.888178E-14	-0.557049E-12	0.888178E-12
180	0.000000E+00	0.000000E+00	0.000000E+00	0.000000E+00
∞	0.000000E+00	0.000000E+00	0.000000E+00	0.000000E+00

OBS: Observe que a partir da iteração 180 a sequência de funções $\{\mathbf{u_k}\}_{\mathbf{k}\in\mathbb{N}}$ do problema 3) convergiu para $\mathbf{u}_\infty = \mathbf{u}$ onde $\mathbf{u}_\infty$ é o limite de $\mathbf{u_k}$ quando $k \to \infty$ e este limite por sua vez é a função $\mathbf{u}$ do problema 1). Logo os valores dos campos distantes refv1, refv2 e refv3 de $\mathbf{u_{180}}$ nos dá uma boa aproximação do limite (2.7).

5.10. Resultado 10 (gerado no Okeanos-Grécia)

Parâmetros do problema: p = 2.1
S1: (1.0, 0.5), b1 = 5 S2: (-1.5, 1.0), b2 = 1 S3: (-0.5, -1.5), b2 = 4.5
Região computacional: x_min = - 20, x_max = 20, y_min = - 20, y_max = 20
Amostragens de campo distante: refv1 = 10, refv2 = 14, refv3 = 18
Malha numérica: M1 = - 200, M2 = 200; N1 = - 200, N2 = 200
h = 0.1, dt = 0.0001, cfl = 0.01
Valor inicial para b: 3.25;
Valor médio de b: 3.5
Valor mínimo de **u** em todas as iterações: `min_u` = 1
Valor máximo de **u** em todas as iterações: `max_u` = 5

OBS: Observe que os valores de refv1, refv2 e refv3 para uma precisão de 5 dígitos depois da iteração 109 (tF = 10000) não mudaram mais.

Run	**tF**	**refv1**	**refv2**	**refv3**	**WT** (segundos)
0	0	3.25000	3.25000	3.25000	0.134375E+01
19	1000	3.38842	3.38113	3.37738	0.183430E+05
29	2000	3.42809	3.42560	3.42437	0.183396E+05
39	3000	3.44090	3.43998	3.43956	0.179927E+05
49	4000	3.44505	3.44463	3.44447	0.182078E+05
59	5000	3.44639	3.44613	3.44605	0.239998E+05
69	6000	3.44682	3.44662	3.44657	0.190170E+05
79	7000	3.44696	3.44677	3.44673	0.188708E+05
89	8000	3.44701	3.44682	3.44679	0.189145E+05
99	9000	3.44702	3.44684	3.44680	0.200718E+05
109	10000	3.44703	3.44685	3.44681	0.189968E+05
119	11000	3.44703	3.44685	3.44681	0.194533E+05
129	12000	3.44703	3.44685	3.44681	0.186465E+05
139	13000	3.44703	3.44685	3.44681	0.195666E+05
149	14000	3.44703	3.44685	3.44681	0.195358E+05
169	16000	3.44703	3.44685	3.44681	0.200060E+05
173	16400	3.44703	3.44685	3.44681	0.199668E+05
174	16500	3.44703	3.44685	3.44681	0.201294E+05
∞	∞	3.44703	3.44685	3.44681	

Run	**Δmassa**	$\| \Delta \mathrm{u} \|_{\max}$	**media_du/dt**	**max_du/dt**
0	0.278093E-02	0.111072E+00	0.172942E-03	0.111072E+02
19	0.114727E-02	0.986508E-06	0.713473E-04	0.986508E-04
29	0.370832E-03	0.278409E-06	0.230615E-04	0.278409E-04
39	0.119860E-03	0.895347E-07	0.745396E-05	0.895347E-05
49	0.387542E-04	0.289605E-07	0.241007E-05	0.289605E-05
59	0.125319E-04	0.936702E-08	0.779342E-06	0.936702E-06
69	0.108351E-04	0.788143E-08	0.673821E-06	0.788143E-06
79	0.131055E-05	0.979693E-09	0.815015E-07	0.979693E-07
89	0.423818E-06	0.316813E-09	0.263567E-07	0.316813E-07
99	0.137059E-06	0.102451E-09	0.852350E-08	0.102451E-07
109	0.443244E-07	0.331295E-10	0.275648E-08	0.331295E-08
119	0.143345E-07	0.107025E-10	0.891446E-09	0.107025E-08
129	0.463558E-08	0.346434E-11	0.288280E-09	0.346434E-09
139	0.149775E-08	0.111067E-11	0.931432E-10	0.111067E-09
149	0.482419E-09	0.355715E-12	0.300010E-10	0.355715E-10
169	0.630739E-10	0.444089E-13	0.392248E-11	0.444089E-11
173	0.273691E-10	0.444089E-13	0.170205E-11	0.444089E-11
174	0.000000E+00	0.000000E+00	0.000000E+00	0.000000E+00
∞	0.000000E+00	0.000000E+00	0.000000E+00	0.000000E+00

OBS: Observe que a partir da iteração 174 a sequência de funções $\{\mathbf{u_k}\}_{\mathbf{k}\in\mathbb{N}}$ do problema 3) convergiu para $\mathbf{u}_\infty = \mathbf{u}$ onde $\mathbf{u}_\infty$ é o limite de $\mathbf{u_k}$ quando $k \to \infty$ e este limite por sua vez é a função $\mathbf{u}$ do problema 1). Logo os valores dos campos distantes refv1, refv2 e refv3 de $\mathbf{u_{174}}$ nos dá uma boa aproximação do limite (2.7).

5.11 - Conclusão

Olhando com atenção os 10 resultados acima, podemos complementar os fatos já mencionados no início deste capítulo e concluir o seguinte:

1) Transformar o **problema original, estático**: $\Delta_p u = 0$, no **p-Laplaciano evolutivo**: $u_t = \Delta_p u$, e resolver o p-Laplaciano evolutivo por meio de **diferenças finitas**, "deu certo", pois todas as 10 simulações convergiram para o problema original, isto é, com estas duas mudanças foi possível usar o computador para descobrir o comportamento da função procurada.

2) Os experimentos(resultados) 1 e 2 apresentam mesma configuração, apenas alturas iniciais diferentes, em ambos, os experimentos convergiram para o mesmo resultado:

$$b_\infty := \lim_{|x|\to\infty} u(x) = \lim_{k\to\infty}\Big[\lim_{|x|\to\infty} u_k(x)\Big] \tag{5.7}$$

basta olhar na última linha das tabelas na página 77 e 79. Ou seja, o valor inicial do plano, a configuração inicial, não altera o resultado final, não altera o valor final da convergência b_∞.

3) Por outro lado os experimentos 4 e 5 possuem a mesma localização das 4 singularidades e o mesmo

$$b_m := \frac{b_1 + b_2 + b_3 + b_4}{4} = 2.75 \tag{5.8}$$

Mas olhando na última linha das tabelas na página 86 e 88, percebemos claramente, que ambos convergem para resultados diferentes de b_∞. O que mostra que a **Observação 2.1** está correta, ou seja, o valor de b_∞ não é b_m (a média aritmética das alturas das singularidades).

4) Já o experimento 7, mostra que provavelmente o **Teorema 2.2** está correto, isto é, b_∞ é a média das alturas das singularidades quando temos somente duas singularidades.

5) Todos os 10 experimentos, levaram várias semanas ou meses para chegar ao resultado final. Isto mostra claramente a necessidade de usar outros computadores em paralelo, para acelerar a velocidade do processamento, ou seja nos próximos trabalhos, devemos usar MPI, ou MP com MPI, ou outra biblioteca de processamento paralelo que use vários computadores simultaneamente.

Apêndice A

Instalação de Compiladores Fortran e C/C++ em ambientes Windows e Linux

Neste apêndice daremos uma sugestão de como instalar os compiladores Fortran e C/C++ no seu microcomputador pessoal (PC) ou notebook.

(1) Para os que gostam de Linux, procedam nesta ordem:

   (*a*) Instale o sistema operacional Oracle Linux.

   (*b*) No Oracle Linux, instale o Java para Linux.

   (*c*) Instale o Oracle Developer Studio.

Observação: O usuário deverá ter uma certa experiência com Linux. Para o passo (*c*) acima, o instalador do Oracle Developer Studio deverá ter conhecimento do caminho (*path*, ou endereço/localização) do Java JRE.

(2) Para os que gostam do Windows, há várias opções, como, por exemplo:

   (*a*) Faça o download da máquina virtual VMware Workstation Player para Windows (gratuita). Após isso, siga os passos em (1) acima. Desta forma, você terá o Oracle Developer Studio instalado no Oracle Linux, que por sua vez roda na VMware, e este, por sua vez, roda dentro do seu Windows. Para isto, você deverá ter no mínimo 4 GB de memória RAM e um rápido processador instalado em seu PC ou notebook.

   (*b*) Instale agora o IDE (*Integrated Development Environment*, ou Ambiente de Desenvolvimento Integrado) gratuito CodeBlocks com os compiladores `gcc`, `g++` e `gfortran` da GNU inclusos (pré-instalados). O arquivo instalador em novembro/2018 é codeblocks-17.12mingw_fortran-setup.exe (procure por ele no google).

   (*c*) Se você é aluno universitário e possui e-mail de sua faculdade (por exemplo: SeuNome@ufrgs.br), você tem direito a usar gratuitamente enquanto for aluno universitário (basta renovar a inscrição a cada ano) o compilador Intel Fortran (oferta disponível em 2017/2018). Neste caso, instale o

IDE gratuito VISUAL STUDIO 2013 COMMUNITY da Microsoft (ou a versão 2015 ou 2017, se seu computador tiver ótimo desempenho). Após, instale o INTEL FORTRAN e, durante sua instalação, escolha além dos aplicativos que quer instalar a integração do INTEL FORTRAN com o VISUAL STUDIO.

(d) Ou instale os compiladores GNU (`gcc`, `g++`, `gfortran`) usando o arquivo mingw-w64-install.exe (procure no GOOGLE). Adicione o caminho do compilador GNU instalado nas variáveis do sistema do WINDOWS e após use o editor de texto de sua preferência (NOTEPAD, CODEBLOCKS, VISUAL STUDIO CODE, etc.) para escrever o código fonte. E use a linha de comando (o "velho e poderoso PROMPT"): simples, rápido e direto. Por exemplo, no diretório onde estão os arquivos fontes, abra a janela do COMMAND PROMPT (executando, por exemplo, `cmd.exe` no WINDOWS) e digite:

```
gfortran -fopenmp arquivofonte.f90 -o nomedoarquivo.exe
```

para compilar e gerar o executável de um programa fonte em Fortran que está no arquivo arquivofonte.f90. A opção `-o` serve para renomear o arquivo gerado: nomedoarquivo.exe será o nome do arquivo executável. Para um programa em C ou C++, o comando correspondente seria:

```
g++ -fopenmp arquivofonte.cpp -o nomedoarquivo.exe.
```

(e) Um outra opção seria seguir os passos de (d) e depois fazer o download e instalação do IDE: ECLIPSE PARALLEL APPLICATION DEVELOPERS. Depois disso, deve-se configurar o ECLIPSE para que ele saiba onde estão os compiladores GNU.

(f) Uma outra opção (também gratuita) para Fortran e C/C++ seria instalar o compilador FTN95 PERSONAL EDITION da SILVERFROST (e o IDE PLATO associado). Apesar deste ambiente ser voltado prioritariamente para Fortran 90/95, o FTN95 contém também compiladores C e C++ embutidos, e outros recursos descritos no endereço

https://www.silverfrost.com/ftn95-help/devel/compiler_overview.aspx

além de se poder adicionar quaisquer compiladores extras ao IDE PLATO (como `gcc`, `gfortran`, `ifort`, etc.), se desejado.

Apêndice B

Os ambientes computacionais Cenapad-SP, Grid5000 e Okeanos

Além de meu notebook Acer de 2013 com Intel i5, foram utilizados para as simulações os ambientes computacionais do cenapad-sp, Grid5000 e okeanos.

O okeanos e o okeanos global são clusters computacionaia de pesquisa gregos. O okeanos é dirigido especialmente para alunos, professores e pesquisadores da Grécia. Já o okeanos global disponibiliza acesso a duas máquinas virtuais, cada uma com 40 GB de memória no disco rígido, 2 GB de memória RAM e 2 processadores lógicos para alunos, professores e pesquisadores de diversas universidades de vários países do mundo — entre elas, a UFRGS. Basta entrar no site deles e fazer a inscrição. Também é possível montar apenas uma máquina virtual com 4 GB de RAM, 4 processadores lógicos e dois discos com 40 GB totalizando 80 GB de memória permanente no disco rígido. E foi justamente esta configuração que eu escolhi. Na hora da montagem da máquina virtual o usuário escolhe o sistema operacional que quer utilizar entre os vários disponibilizados: Windows Server 2012, Ubuntu, Debian, e outros. Eu escolhi o Windows Server 2012 e nele instalei o IDE Code Blocks e os compiladores da GNU (`gcc`, `g++`, `gfortran`) versão 7.1.0. Podem parecer modestos, ou não muito potente, apenas 4 GB de RAM e 4 processadores lógicos, mas a grande vantagem desta máquina virtual é que ela pode ficar rodando 24h por dia por tempo indeterminado: 1 dia, 10 dias, 100 dias, etc. Com esta característica, ela se torna bem poderosa. Várias simulações que duraram vários dias ou centenas de horas foram feitas e alguns dos resultados foram mostrados no Capítulo V.

O Grid 5000 é um cluster francês utilizado por pesquisadores e alunos de várias partes do mundo (além, é claro, pelos próprios alunos, professores e pesquisadores franceses). No site do Grid 5000 há uma ficha de inscrição. Nesta ficha preenchi meus dados e nela mostrei que o meu trabalho estava relacionado com o uso de computadores de alto desempenho ou interligados em rede, duas características que o Grid 5000 possui. Até o presente momento (maio/2018), após 3 renovações, minha conta no Grid 5000 continua ativa. O Grid 5000 está operando em 8 cidades da França: Grenoble, Lille, Luxembourg, Lyon, Nancy, Nantes, Rennes e Sophia. Cada cidade tem um cluster com diferentes máquinas: por exemplo, o cluster de Nancy possui 410 processadores, enquanto o de Lille possui 46 processadores. Na soma das

8 cidades, o total de processadores é 1706 (1458 da Intel e 248 da AMD). Lille é a cidade com menor número de máquinas: 23, cada uma com 2 processadores, mas cada processador possui 10 ou 14 núcleos físicos (em comparação, o meu notebook com processador Intel I5 possui apenas 2 núcleos físicos). Esta foi a razão por que escolhi o cluster de Lille para fazer as simulações em série ou em paralelo (OpenMP), pois seus computadores são muito poderosos.

O CENAPAD-SP/Unicamp faz parte do SINAPAD (Sistema Nacional de Processamento de Alto Desempenho), assim como o CESUP/UFRGS, mas tem mais recursos. Localizado na cidade de Campinas, no estado de São Paulo, o CENAPAD-SP é um poderoso centro computacional utilizado por vários pesquisadores e estudantes das mais diferentes áreas e partes do Brasil. O CENAPAD-SP é constituído basicamente por quatro ambientes computacionais distintos, a saber:
∗ Ambiente IBM P750 (Sistema Operacional: Unix AIX);
∗ Ambiente IBM/GPU (Sistema Operacional: Linux SUSE);
∗ Ambiente SGI Altix 1350/450 (Sistema Operacional: Linux SUSE);
∗ Ambiente SGI Altix ICE 8400 LX (Sistema Operacional: Linux SUSE).
Cada um desses ambientes tem seus recursos, suas ferramentas, suas limitações e especificidades. Nas simulações deste trabalho utilizamos bastante os computadores do CENAPAD-SP, principalmente o ambiente SGI Altix 1350/450 (tanto nas execuções seriais como nas paralelas). Para uma descrição detalhada dos recursos disponíveis em cada um destes ambientes, pode-se consultar

https://www.cenapad.unicamp.br/parque/equipamentos.shtml

ou a página de instruções aos usuários do CENAPAD-SP,

https://www.cenapad.unicamp.br/diversos/guia/guia.shtml,

ou ainda o GUIA DO USUÁRIO DO CENAPAD-SP, disponível igualmente no link acima. No CENAPAD-SP, a abertura de contas é feita com o FORMULÁRIO PADRÃO PARA ABERTURA DE CONTAS e o FORMULÁRIO DE PROJETO, disponíveis no endereço

https://www.cenapad.unicamp.br/politicas/formcont.shtml,

que, depois de preenchidos e assinados pelo coordenador responsável pelo projeto, devem ser enviados por email para abrecontas@cenapad.unicamp.br (anteriormente, até o ano 2017, eram enviados em papel utilizando-se o correio padrão).

Apêndice C

Os códigos fontes MatLab, Fortran e C++

Os códigos fontes em MatLab, Fortran e em C++(experimental) e esta dissertação em pdf se encontram disponíveis para download em:

https://github.com/Roberto-Hoo/Tese-RobertoHoo-PZingano-UFRGS-Matematica-2018

Visto que é um trabalho acadêmico não há nenhuma garantia que os códigos fontes sempre funcionem.

Email(contato): robertohoo@gmail.com

Apêndice D

Estimativa do tempo via solução de Barenblatt do p-Laplaciano evolutivo

Neste apêndice vamos mostrar com mais detalhes, mas de modo informal, a estimativa do tempo via solução de Barenblatt do p-Laplaciano evolutivo visto na seção (3.3), para detalhes mais profundos, ver artigos ou textos mais avançados sobre a solução de Barenblatt.

$$u_t \;=\; \mathrm{div}\,\big(\,|\nabla u\,|^{p-2}\,\nabla u\,\big) \tag{D 1}$$

Para isto vamos supor:

$$u(x,t) \;=\; t^{-\beta}w\big(\frac{|x|}{t^{\alpha}}\big) \qquad \alpha,\beta>0 \quad e \quad \xi=\frac{|x|}{t^{\alpha}}=\frac{r}{t^{\alpha}} \tag{D 2}$$

Desenvolvendo o lado esquerdo de (D 1) através de (D 2) temos:

$$u_t \;=\; -\beta t^{-\beta-1}w\big(\xi\big)-\alpha t^{-\beta-1}w'\big(\xi\big)\xi=-t^{-\beta-1}\big(\beta w(\xi)+\alpha\xi w'(\xi)\big) \tag{D 3}$$

Por outro lado, desenvolvendo o lado direito de (D 1) usando (D 2) temos:

$$\frac{\partial u}{\partial x_i}=t^{-\beta-\alpha}w'\big(\xi\big)\frac{x_i}{r} \quad e \quad |\nabla u|=t^{-\beta-\alpha}|w'\big(\xi\big)| \tag{D 4}$$

Supondo $|w'\big(\xi\big)|=-w'\big(\xi\big)$ temos:

$$\begin{aligned} |\nabla u|^{p-2}\,\frac{\partial u}{\partial x_i} &= t^{-(\beta+\alpha)(p-2)}\;|w'\big(\xi\big)|^{p-2}\;t^{-\beta-\alpha}\;w'\big(\xi\big)\;\frac{x_i}{r} \\ |\nabla u|^{p-2}\,\frac{\partial u}{\partial x_i} &= -t^{-(\beta+\alpha)(p-1)}\;\big(-w'\big(\xi\big)\big)^{p-1}\;\frac{x_i}{r} \end{aligned} \tag{D 5}$$

E portanto,

$$\mathrm{div}\,\big(\,|\nabla u\,|^{p-2}\,\nabla u\,\big)=-t^{-(\beta+\alpha)(p-1)}\;\sum_{i\,=1}^{n}\frac{\partial}{\partial x_i}\big[(-w'(\xi))^{p-1}\;\frac{x_i}{r}\big]$$

$$= -t^{-(\beta+\alpha)(p-1)} \sum_{i=1}^{n} \big[-(p-1)\frac{{x_i}^2}{r^2}\big(-w'(\xi)\big)^{p-2}\ \frac{w''(\xi)}{t^\alpha} + \big(-w'(\xi)\big)^{p-1}\big(\frac{1}{r} - \frac{{x_i}^2}{r^3}\big)\big]$$

$$= -t^{-(\beta+\alpha)(p-1)}\ \big[-(p-1)\big(-w'(\xi)\big)^{p-2}\ \frac{w''(\xi)}{t^\alpha} + \big(-w'(\xi)\big)^{p-1}\big(n-1\big)\ \frac{1}{\xi\, t^\alpha}\big]$$

Ou seja:

$$-t^{-(\beta+\alpha)(p-1)-\alpha}\ \big[-(p-1)(-w'(\xi))^{p-2}\ w''(\xi) + (n-1)(-w'(\xi))^{p-1}\xi^{-1}\ \big]$$

$$= \operatorname{div}\big(|\nabla u\,|^{p-2}\ \nabla u\big) \quad \text{(D 6)}$$

Igualando (D 3) e (D 6) segundo (D 1) temos:

$$t^{-(\beta+\alpha)(p-1)-\alpha}\ \big[-(p-1)(-w'(\xi))^{p-2}\ w''(\xi) + (n-1)(-w'(\xi))^{p-1}\xi^{-1}\ \big]$$

$$= -t^{-\beta-1}\big[\beta w(\xi) + \alpha\xi w'(\xi)\big] \quad \text{(D 7)}$$

Igualando os expoentes, resulta

$$\alpha p + \beta(p-2) = 1 \quad \text{(D 8)}$$

As expressões entre colchetes em (D 7) é equivalente a

$$\frac{d}{d\xi}\big[(-w'(\xi))^{p-1}\big] + (n-1)(-w'(\xi))^{p-1}\xi^{-1} = \beta w(\xi) + \alpha\xi w'(\xi) \quad \text{(D 9)}$$

Multiplicando (D 9) por ξ^{n-1} e impondo $\beta = \alpha n$, temos equivalentemente

$$\frac{d}{d\xi}\big[\xi^{n-1}(-w'(\xi))^{p-1}\big] = \alpha\big[\frac{d}{d\xi}\big(\xi^n w(\xi)\big)\big] \quad \text{(D 10)}$$

Usando (D 8) e $\beta = \alpha n$, achamos α

$$\alpha = \frac{1}{p + n(p-2)} \quad \text{(D 11)}$$

E β

$$\beta = \frac{n}{p + n(p-2)} \quad \text{(D 12)}$$

Integrando (D 10)

$$\xi^{n-1}(-w'(\xi))^{p-1} = \alpha\xi^n w(\xi) + \gamma \tag{D 13}$$

Na origem, $x = 0$ e portanto $\xi = |x|\, t^{-\alpha} = 0$. Então a contante $\gamma = 0$ em (D 13) Podemos reescrever (D 13) desta forma

$$w(\xi)^{-1/(p-1)}\, w'(\xi) = -\alpha^{1/(p-1)}\, \xi^{1/(p-1)} \tag{D 14}$$

(D 14) é equivalente a

$$\frac{d}{d\xi}\big[\frac{p-1}{p-2} w(\xi)^{(p-2)/(p-1)}\big] = -\alpha^{1/(p-1)}\, \frac{d}{d\xi}\big[\frac{p-1}{p}\xi^{p/(p-1)}\big] \tag{D 15}$$

Integrando (D 15) chegamos a

$$w(\xi) = \big(C - \alpha^{1/(p-1)}\, \frac{p-2}{p}\, \xi^{p/(p-1)}\big)^{(p-1)/(p-2)} \tag{D 16}$$

E assim chegamos a função $u(x,t)$

$$u(x,t) \;=\; v_*(x,t) \;=\; t^{-n\alpha} \times \Big(C \;-\; \alpha^{\frac{1}{p-1}} \cdot \;\; \frac{p-2}{p} \left(r \cdot t^{-\alpha} \right)^{\frac{p}{p-1}} \Big)_+^{\frac{p-1}{p-2}} \tag{D 17}$$

onde $r = |\, x - x_0|$ para certo $x_0 \in \mathbb{R}^n$, e $C > 0$ é uma constante dada, relacionada com a *massa* $\mathsf{m} = \int_{\mathbb{R}^n} u(x,t)\, dx$ da solução $u(\cdot, t)$, cujo valor independe do tempo t. Sendo o suporte de $v_*(\cdot, t)$ dado pela bola fechada de centro x_0 e raio $R(t)$, com

$$R(t) \;=\; C^{\,\frac{p-1}{p}} \cdot\, \alpha^{-\frac{1}{p}} \cdot \, \Big(\frac{p}{p-2} \Big)^{\frac{p-1}{p}} \cdot \;\; t^{\alpha} \tag{D 18}$$

para todo $t > 0$. Este valor é o que zera (D 17) quando substituímos r por $R(t)$.

Para calcular está constante C começamos por está primeira mudança de variável:

$$\mathsf{m} = \int_{|x|<R(t)} u(x,t)\, dx = \omega_n \int_0^{R(t)} t^{-\alpha n}\, w\big(\frac{r}{t^\alpha}\big)\, r^{n-1}\, dr \tag{D 19}$$

Onde ω_n é a área da hipersuperfície $\{\, x \in \mathbb{R}^n \colon |\, x\,| = 1 \,\}$, $\omega_1 = 2$, $\omega_2 = 2\pi$, $\omega_3 = 4\pi$, etc...

Segunda mudança de variável, $s = \xi = \dfrac{|x|}{t^{\alpha}} = \dfrac{r}{t^{\alpha}}$, substituindo em (D 19) temos:

$$\mathsf{m} = \omega_n \int_0^{R(t)/t^{\alpha}} t^{-\alpha n}\, w\big(s\big)\, s^{n-1}\, t^{\alpha(n-1)}\, t^{\alpha} ds = \omega_n \int_0^{R(t)/t^{\alpha}} w\big(s\big)\, s^{n-1}\, ds$$

$$= \omega_n \int_0^{R(t)/t^{\alpha}} \Big(C \,-\, \alpha^{\frac{1}{p-1}} \cdot \;\; \frac{p-2}{p} \;\cdot s^{\frac{p}{p-1}} \Big)^{\frac{p-1}{p-2}} \; s^{n-1}\, ds \qquad \text{(D 20)}$$

Terceira mudança de variável, $\theta = \alpha^{\frac{1}{p-1}} \cdot \;\; \dfrac{p-2}{p} \;\cdot s^{\frac{p}{p-1}}$, substituindo em (D 20) temos, após as simplificações:

$$\mathsf{m} = \omega_n \Big(\frac{p-1}{p}\Big) \alpha^{-\frac{n}{p}} \Big(\frac{p}{p-2}\Big)^{\frac{n(p-1)}{p}} \int_0^{C} \Big(\, C \,-\, \theta \, \Big)^{\frac{p-1}{p-2}} \, \theta^{\frac{n(p-1)}{p}-1}\, d\theta \qquad \text{(D 21)}$$

Quarta mudança de variável, $\theta = Cr$, substituindo em (D 21) temos:

$$\mathsf{m} = \omega_n \Big(\frac{p-1}{p}\Big) \alpha^{-\frac{n}{p}} \Big(\frac{p}{p-2}\Big)^{\frac{n(p-1)}{p}} C^{(p-1)\left[\frac{1}{p-2}+\frac{n}{p}\right]} \int_0^{1} \Big(\, 1 \,-\, r \, \Big)^{\frac{p-1}{p-2}} \, r^{\frac{n(p-1)}{p}-1}\, dr \qquad \text{(D 22)}$$

Relembrando, para $x > 0$ e $y > 0$, a função $Beta(x,y) = \displaystyle\int_0^1 (\, 1 \,-\, r \,)^{y-1}\, r^{\,x-1}\, dr$

Então em (D 22) temos $x = n(p-1)/p \quad$ e $\quad y = (2p-3)/(p-2)$.
E assim:

$$\mathsf{m} = \omega_n \Big(\frac{p-1}{p}\Big) \alpha^{-\frac{n}{p}} \Big(\frac{p}{p-2}\Big)^{\frac{n(p-1)}{p}} C^{(p-1)\left[\frac{1}{p-2}+\frac{n}{p}\right]}\, Beta\Big(\frac{n(p-1)}{p}, \frac{2p-3}{p-2}\Big) \qquad \text{(D 23)}$$

Lembrando o valor de α em (D 11) e isolando C:

$$C \;=\; \left\{ \frac{\mathsf{m}}{\omega_n} \cdot \frac{p}{p-1} \cdot \Big(\frac{1}{p+n(p-2)} \Big)^{\frac{n}{p}} \cdot \Big(\frac{p-2}{p} \Big)^{\frac{n(p-1)}{p}} \Big/ \; Beta\big(\frac{n(p-1)}{p}, \frac{2p-3}{p-2} \big) \right\}^{\frac{p(p-2)}{(p-1)(p+n(p-2))}} \qquad \text{(D 24)}$$

Finalmente, usando este C de (D 24) e isolando t = T em (D 18) temos que o tempo T para que se alcance uma distância $R > 0$ do centro x_0 é dado por

$$T \;=\; \left\{ R \cdot \big(p + n(p-2)\big)^{-\frac{1}{p}} \cdot \Big(\frac{p-2}{p} \Big)^{\frac{p-1}{p}} \cdot \;\; C^{-\frac{p-1}{p}} \right\}^{p+n(p-2)} \qquad \text{(D 25)}$$

Referências

[1] L. P. Bonorino, *private communication*, Universidade Federal do Rio Grande do Sul, Porto Alegre, RS, 2018.

[2] L. P. Bonorino, A. R. Silva and P. R. Zingano, *Liouville's theorem and comparison results for solutions of degenerate elliptic equations in exterior domains*, J. Math. Anal. Appl. **463** (2018), 794-809.

[3] CENAPAD-SP, Introdução ao OpenMP, Apostila de Treinamento, Universidade Estadual de Campinas, Campinas, SP, 2014 (disponível no endereço URL https://www.cenapad.unicamp.br/servicos/treinamentos/apostilas/apostila_openmp.pdf).

[4] CENAPAD-SP, Guia do Usuário, Centro Nacional de Processamento de Alto Desempenho em São Paulo, Universidade Estadual de Campinas, Campinas, 2016.

[5] R. Chandra, L. Dagum, D. Kohr, D. Maydan, J. McDonald and R. Menon, Parallel Programming in OpenMP, Academic Press, San Diego, 2001.

[6] S. Chapman, Fortran 95/2003 for Scientists and Engineers (3rd ed), McGraw-Hill, New York, 2007.

[7] B. Chapman, G. Jost and R. van Der Pas, Using OpenMP: portable shared memory parallel programming, MIT Press, Cambridge, 2008.

[8] E. DiBenedetto, $C^{1+\alpha}$ *local regularity of weak solutions of degenerate elliptic equations*, Nonlin. Anal. **7** (1983), 827-850.

[9] V. Dolean, P. Jolivet and F. Nataf, An Introduction to Domain Decomposition Methods: algorithms, theory, and parallel implementation, SIAM, 2015.

[10] L. C. Evans, *A new proof of local* $C^{1,\alpha}$ *regularity for solutions of certain degenerate elliptic PDE*, J. Diff. Eqs. **45** (1982), 356-373.

[11] G. B. Folland, Introduction to Partial Differential equations (2nd ed.), Princeton University Press, Princeton, 1995.

[12] M. Fraas and Y. Pinchover, *Positive Liouville theorems and asymptotic behavior for p-Laplacian type elliptic equations with a Fuchsian potential*, Confl. Math. **3** (2011), 291-323.

[13] M. Fraas and Y. Pinchover, *Isolated singularities of positive solutions of p-Laplacian type equations in* $\mathbb{R}^d$, J. Diff. Eqs. **254** (2013), 1097-1119.

[14] D. Gilbarg and N. S. Trudinger, Elliptic Partial Differential Equations of Second Order, Springer, New York, 1997.

[15] F. Gustafsson and N. Bergman, Matlab for Engineers Explained, Springer, London, 2003.

[16] D. Hanselman and B. Littlefield, Mastering Matlab 6: a comprehensive tutorial and reference, Prentice-Hall, Upple Saddle River, 2001.

[17] J. L. Lewis, *Smoothness of certain degenerate elliptic equations*, Proceed. Amer. Math. Soc. **80** (1980), 259-265.

[18] J. L. Lewis, *Regularity of the derivatives of solutions to certain degenerate elliptic equations*, Indiana Univ. Math. J. **32** (1983), 849-858.

[19] C. F. van Loan, Introduction to Scientific Computing: a matrix-vector approach using Matlab, Prentice-Hall, Upper Saddle River, 2000.

[20] T. Mathew, Domain Decomposition Methods for the Numerical Solution of Partial Diffferential Equations, Springer, New York, 2008.

[21] T. Meis and U. Marcowitz, Numerical Solution of Partial Differential Equations, Springer, New York, 1981.

[22] E. Mitidieri and S. I. Pohozaev, *Some generalizations of the Bernstein theorem*, Diff. Equations **38** (2002), 373-378.

[23] W. P. Petersen and P. Arbenz, Introduction to Parallel Computing: a practical guide with examples in C, Oxford University Press, Oxford, 2004.

[24] A. Quarteroni and A. Valli, Domain Decomposition Methods for Partial Diffferential Equations, Oxford University Press, Oxford, 1999.

[25] M. J. Quinn, Parallel Programming in C with MPI and OpenMP, McGraw-Hill, New York, 2003.

[26] J. Serrin, *Singularities of solutions of nonlinear equations*, Proc. Sympos. Appl. Math. **17** (1965), Amer. Math. Soc., Providence, RI, pp. 68-88.

[27] J. Serrin, *The Liouville theorem for homogeneous elliptic differential inequalities*, J. Math. Sci. **179** (2001), 174-183.

[28] J. Serrin and H. Zou, *Cauchy-Liouville and universal boundedness theorems for quasilinear elliptic equations and inequalities*, Acta Math. **189** (2002), 79-142.

[29] G. A. Sod, Numerical Methods in Fluid Dynamics, Cambridge University Press, Cambridge, 1985.

[30] M. F. Stapenhorst, *Regularidade $C^{1,\alpha}$ de funções p-harmônicas*, Dissertação de Mestrado, L. P. Bonorino (orientador), Programa de Pós-Graduação em Matemática (PPGMAT/UFRGS), Universidade Federal do Rio Grande do Sul, Porto Alegre, RS, Março de 2018.

[31] J. C. Strikwerda, Finite Difference Schemes and Partial Differential Equations (2nd ed.), SIAM, Philadelphia, 2004.

[32] A. Toselli and O. Widlund, Domain Decomposition Methods: algorithms and theory, Springer, New York, 2010.

[33] P. R. Woodward, *Personal comunication*, Army High Performance Computing Research Center, University of Minnesota, Minneapolis, MN, February/1995.

[34] Z. Wu, J. Zhao, J. Yin and H. Li, Nonlinear Diffusion Equations, World Scientific, New Jersey, 2001.

[35] P. R. Zingano, Supercomputing on Cray Y-MPs: what to do and not to do, Notas de Minicurso, CNMAC/Congresso Nacional de Matemática Aplicada e Computacional, Soc. Bras. Mat. Aplicada e Computacional, Uberlândia, MG, 1993, pp. 1 - 57.